МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

*В.Г. ИОФФЕ, А.В. ГРАФКИН, В.В. ГРАФКИН*

# АРХИТЕКТУРА, ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ И ПРОГРАММНЫЕ СРЕДСТВА МИКРОКОНТРОЛЛЕРОВ STM32

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для обучающихся по основным образовательным программам высшего образования по направлениям подготовки 09.03.01 Информатика и вычислительная техника, 10.03.01 Информационная безопасность, 10.05.01 Компьютерная безопасность, 10.05.03 Информационная безопасность автоматизированных систем

С А М А Р А
Издательство Самарского университета
2021

УДК 004.8(085)
ББК 32.973.26-04я7
И758

Рецензенты: д-р техн. наук, доц. Ю. Н. С е к и с о в,
канд. физ.-мат. наук, доц. Н. П. К о з л о в

***Иоффе, Владислав Германович***

И758 **Архитектура, принципы функционирования и программные средства микроконтроллеров STM32:** учебное пособие / *В.Г. Иоффе, А.В. Графкин, В.В. Графкин*. – Самара: Издательство Самарского университета, 2021. – 490 с.: ил.

**ISBN 978-5-7883-1685-7**

В пособии рассмотрены основные особенности структурной организации и принципы функционирования микроконтроллеров STM32 на примере STM32F103C8T6. Приведены методики использования сред разработки. Представлены комплексы программ для микроконтроллера STM32F103C8T6 и особенности их отладки с помощью модуля PinBoard II R3.

Предназначено для обучающихся по направлениям подготовки 09.03.01 Информатика и вычислительная техника, 10.03.01 Информационная безопасность, 10.05.01 Компьютерная безопасность, 10.05.03 Информационная безопасность автоматизированных систем. Подготовлено для дисциплин «Микропроцессорные средства и системы», «ЭВМ и периферийные устройства», «Организация ЭВМ и вычислительных систем», «Электроника и схемотехника», «Программно-аппаратные средства обеспечения информационной безопасности», «Комплексные методы защиты объектов информатизации».

Подготовлено преподавателями кафедр «Информационных систем и технологий» и «Безопасности информационных систем».

УДК 004.8(085)
ББК 32.973.26-04я7

ISBN 978-5-7883-1685-7 

# Оглавление

# Введение

Одним из основных компонентов современных систем сбора и обработки информации является микроконтроллер (МК).

Прогноз IC Insights на ближайшие пять лет предусматривает, что объем рынка микроконтроллеров в денежных средствах будет в среднем увеличиваться на 7,2% и к 2022 году достигнет $23,9 млрд. Рост в штучном выражении ожидается на уровне 11,1%. К концу рассматриваемого периода поставки микроконтроллеров достигнут примерно 43,8 млрд единиц [103].

В некоторых задачах МК представляют собой функционально законченную систему, предназначенную для ввода/вывода аналоговых и цифровых сигналов и их последующей обработки. Более сложные задачи требуют применение ограниченного набора внешних компонентов.

Главное в вычислительных системах на основе МК – это малые габариты, высокая надежность, низкая стоимость в пересчете на реализуемые функции, возможность мобильного изменения алгоритмов функционирования за счет перепрограммирования не только ПЗУ команд, но в некоторых МК и структур, которое может выполняться «на лету» (самопрограммирование, PSOC, SoC и так далее) [50].

В настоящее время выпускаются МК с разрядностью шины данных 8, 16, 32, 64 бита, но основной объем реализации приходится на 8- и 32-разрядные микроконтроллеры.

Основные различия между 8- и 32-разрядными микроконтроллерами – в стоимости, производительности процессорного ядра, функциональных возможностях, размерах корпуса и энергопотреблении.

8-разрядные МК используются в приложениях, в которых не требуется высокая производительность, а выполняется

интенсивный обмен с внешними периферийными устройствами. Для выполнения одного и того же 8-разрядного приложения на 32-разрядном микроконтроллере нужно от полутора до трех раз больше оперативной памяти, чем на 8-разрядном, увеличиваются объемы флеш-памяти. Для приложений, в которых требуется малые габариты, низкие энергопотребление и стоимость (при низкой производительности), более оптимальным является применение 8-разрядные МК.

Область применении микроконтроллеров постоянно расширяется, в связи с чем более жесткими становятся требования к производительности и быстродействию.

Последние годы развития микропроцессорной техники привели к широкому применению 32-разрядных микроконтроллеров. Это обусловлено снижением их стоимости и энергопотребления, высокой производительностью, большим набором периферийных блоков, развитыми и доступными средствами проектирования.

Наиболее популярно среди 32-разрядных микроконтроллеров семейство Cortex фирмы ARM, состоящее из трех профилей

Cortex-A (Application) – для ресурсоёмких областей применения, таких как смартфоны, телевизоры, промышленные сетевые устройства и серверы;

Cortex-R (Real-time) – оптимизированные для работы в режиме реального времени;

Cortex-M (MCU) – для приложений, где требуются пониженная стоимость и энергопотребление, малые габариты, относительно невысокая производительность (измерительные устройства, интерфейсы, автомобильные и промышленные системы управления, медицинское оборудование).

Ядро Cortex M используют многие производители МК: STMicroelectronics, Texas Instrument, NXP, ATMEL, Analog Devices, Renesas и т.д.

К достоинствам микроконтроллеров фирмы STMicroelectonics можно отнести:

- бесплатные высокоэффективные среды разработок, рассчитанные на широкую номенклатуру микроконтроллеров;
- постоянно расширяющиеся библиотеки;
- программная совместимость от младших моделей к старшим;
- «pin-to-pin совместимость», то есть назначение линий ввода/вывода сохраняется для микроконтроллеров одного класса;
- относительно низкая стоимость.

Применение МК фирмы STMicroelectonics позволяет существенно сократить сроки и стоимость разработки средств вычислительной техники.

Следует отметить, что по результатам анализа рынка России в 4-м квартале 2018 года доля продаж фирмы STMicroelectronics составляла 29,8 % (2-е место), уступая объединенной компании MicroChip/Atmtel. Эта цифра постоянно увеличивается за счет продаж Cortex-M3 [103].

Целью пособия является получение практических навыков проектирования микропроцессорных систем различного назначения на основе микроконтроллеров (МК) STM32, относящегося к семейству ARM Cortex-M3.

Учебное пособие предназначено для студентов, обучающихся по направлениям

09.03.01 – «Информатика и вычислительная техника»,

10.03.01 – «Информационная безопасность»,

10.05.01 – «Компьютерная безопасность»,

10.05.03 – «Информационная безопасность автоматизированных систем».

Пособие ориентировано на студентов, будущей сферой деятельности которых является разработка программного обеспечения с использованием ресурсов ПЭВМ.

По роду своей деятельности они редко общаются непосредственно с аппаратными средствами вычислительных систем. Поэтому изложение материала ориентировано на «функциональную электронику», которая предполагает знание функций основных элементов реальной электроники: драйверов, триггеров, регистров, счетчиков, шифраторов/дешифраторов, мультиплексоров/ демультиплексоров и так далее.

Однако авторы считают, что использование стандартных библиотек, инструментальных средств различного назначения существенно удаляют разработчика от реальных физических процессов.

Студенты должны понимать основы вычислительной техники. Это позволит:

• корректно выбирать режимы работы периферийных блоков МК и выполнять настройку виртуальной среды моделирования (например, CubeMX);

• самостоятельно дополнять библиотеки при отсутствии требуемых моделей;

• грамотно контролировать корректность работы предлагаемых моделей;

• разрабатывать более компактные и быстродействующие программы, что особенно важно в системах реального времени.

Отсутствие понимания базовых принципов функционирования аппаратных средств может привести к катастрофическим последствиям для МК и системы в целом. Поэтому в пособии достаточно подробно описывается архитектура STM32, структурная организация основных блоков и их программные модели.

Здесь уместно вспомнить басню И.А. Крылова «Листья и корни». Аппаратные средства – это корни дерева, на котором, сияя многообразием красок и интеллекта, «цветет и пахнет» программное обеспечение. «Но, если корень иссушится, не станет дерева, ни вас».

Особенности архитектуры, принципы программирования и отладки микроконтроллеров STM32 рассмотрены на примере STM32F103C8T6.

В первой главе изложены основные сведения об архитектуре микроконтроллеров STM32F103xx, рассмотрены особенности ядра и периферийных устройств [85, 53].

Если [85] содержит исчерпывающие сведения об архитектуре ядра, то [53] описывает общие свойства STM32, в которых отсутствуют конкретные сведения о программных моделях и принципах управления периферийными устройствами.

Поэтому при описании и программировании периферийных блоков использовались руководящие документы производителя [15, 17] и сайты mypractic.ru, we.easyelectronics.ru, mycontroller.ru, hubstub.ru, radiohlam.ru, dimoon.ru, myelectronics.com.ua, avislab.com, narodstream.ru, compel.ru, mkprog.ru, catethysis.ru, istarik.ru, microsin.net, microtechnics.ru, habr.com/ru.

Описание периферийных устройств выполнено в следующей последовательности: назначение, режимы работы, особенности структурной организации (для корректного использования возможностей STM32Cube, HAL), программная модель, алгоритмы программирования (при программировании на Си).

Наиболее трудоёмким и ответственным этапом проектирования микропроцессорной системы является разработка программного обеспечения. Для его разработки используют различные инструментальные средства, отличающиеся функциональными возможностями и стоимостью: программный

симулятор, внутрисхемный эмулятор, отладочный модуль, интегрированная среда разработки.

Оптимальным решением является совместное использование интегрированной среды разработки и отладочного модуля.

В учебном пособии предлагается разработку и первичную отладку программного обеспечения выполнять в среде моделирования, а окончательную – на реальном оборудовании отладочного модуля.

Во второй главе анализируются инструментальные средства проектирования, поддерживаемые STMicroelectonics (STM32Cube MX, System Workbench for STM32, STM32CubeIDE) и приведена методика их использования при разработке программ различного назначения. Рассматриваются особенности этих средств при работе с операционной системой реального времени RTOS. Приведены примеры отладки программы, разработанной в STM32CubeIDE, в популярной среде разработки Proteus.

Третья глава посвящена описанию отладочного модуля PinBoard II R3 на базе МК STM32F103C8T6 и особенностям использования его ресурсов для реализации задач ввода/вывода.

Достоинством модуля является блочная организация, которая позволяет его использовать при работе с различными МК (ATMEGA, PIC18F67J60, STM32F103C8T6,STM8L) и ПЛИС (Altera MAX II EPM240T100C5N), наличие основных средств ввода и отображения информации (клавиатура, различные индикаторы, энкодер), аналоговых источников напряжения, разъема расширения, позволяющего подключать дополнительные внешние устройства (том числе модуль Ethernet ENC28J60), коммутация устройств модуля с помощью перемычек и соединительных проводов, питание и обмен информацией через порт USB компьютера.

Возможности отладочного модуля иллюстрируют примеры работы с модулем индикации на базе контроллера HD44780,

виртуальным терминалом компьютера, энкодером, ШИМ-модулятором.

Раздел 3 использует материалы и иллюстрации с сайта разработчика отладочного модуля PinBoard II R3 easyelectronics.ru.

Четвертый раздел описывает возможности и особенности установки ST-Link Utility, которая обеспечивает работоспособность программатора ST-LINK V2.

Учебное пособие используется при проведении лекций и лабораторных работ.

Основной акцент при проведении лабораторных работ делается на задачах ввода-вывода, к которым относятся клавишный ввод, индикация (светодиоды, 7-сегментные и жидкокристаллические индикаторы), преобразование аналоговых и частотно-временных сигналов, взаимодействие с устройствами на базе портов SPI, I2C, USART, USB.

Разделы 1, 3.1, Приложение 1 написаны Иоффе В.Г., а остальные разделы – Графкиным А.В. и Графкиным В.В.

# 1 Архитектура микроконтроллера STM32F103C8T6

Микроконтроллер **STM32F103C8T6 –** представитель семейства **Cortex M3** (группа **Performance**), выполненный на основе ядра **ARMv7-M.** Его отечественным аналогом является МК **1986BE9xx** компании ЗАО «ПКК Миландр».

Выбор микроконтроллера, который принадлежит к младшим моделям семейства STM32, обусловлен его установкой в отладочном модуле PinBoard **II R3 AVR + STM32**.

**Основные характеристики:**

Разрядность – 32 бита.

Частота – до 72 МГц.

Подсистема синхронизации содержит программируемые схемы деления/умножения частоты и позволяет работать с внешним кварцевым резонатором (4-16) МГц, «часовым» кварцевым резонатором 32768 Гц, внутренними RC-генераторами (высокочастотным – 8 МГц, низкочастотным – 40 или 32 КГц).

Объем FLASH-памяти команд – 64 КБ.

Объем ОЗУ – 20 КБ.

Число линий ввода/вывода – 35.

Возможность реализации программного ввода-вывода, по прерыванию, в режиме прямого доступа к памяти.

АЦП – 1 (12 разрядов, 10 входов, максимальное время преобразования – 1 мкс).

Счетчики-таймеры – 4, разрядность 16 бит.

Контроллер прямого доступа (DMA1) – 1 (7 каналов). В других моделях STM32 предусмотрен контроллер DMA2 на 5 каналов.

Подсистема многоуровневых (16 уровней) прерываний – 43 входа запросов от периферий, 16 – от ядра Cortex M3, одно немаскируемое прерывание NMI.

Коммуникационные порты:

- UASRT – 3;

- I2C – 2;
- SPI – 2;
- CANbus 2.0 B – 1;
- USB 2.0 – 1;
- Порты отладки – SWD, JTAG;
- Корпус – LQFP48.

В состав МК входят блок часов реального времени, детектор напряжения питания, блок аппаратного формирования/контроля CRC, сторожевой таймер, 24-разрядный системный таймер SysTick.

Напряжение питания (2÷3,6) В. Ток потребления МК зависит от частоты синхронизации и количества включенных периферийных устройств (таблица 1.1).

Таблица 1.1. Зависимость тока потребления от частоты синхронизации

| | Частота, мГц | Ток потребления, мА |
|---|---|---|
| Все периферийные устройства включены | 72 | 50 |
| | 48 | 36 |
| | 36 | 29 |
| | 24 | 20 |
| | 16 | 15 |
| | 8 | 9 |
| Все периферийные устройства выключены | 72 | 33 |
| | 48 | 25 |
| | 36 | 20 |
| | 24 | 14 |
| | 16 | 11 |
| | 8 | 7 |

Максимальный ток потребления – при частоте 72 МГц и всех включенных устройствах – 50 мА. Если все устройства отключить, то на этой частоте ток будет 33 мА. Выводы МК допускают

вытекающий и втекающий ток не более 20 мА. Рекомендуется – не более 8 мА. В режиме энергосбережения ток потребления можно снизить до единиц мкА.

Предусмотрено подключение аналогового напряжения и батарейного питания, обеспечивающего работоспособность часов реального времени и ряда регистров при отключении основного напряжения питания.

При разработке программного обеспечения МК необходимо знать особенности его архитектуры, которые позволяют минимизировать затраты памяти и увеличивают быстродействие (производительность). Основные особенности архитектуры **Cortex M3** отражены в [50, 53].

**Особенностями архитектуры STM32** являются:

- МК выполнен по гарвардскому принципу, что позволяет одновременно выполнять обработку команд и данных.
- В STM32 используется единое адресное пространство 4 ГБ, разделенное на зоны с фиксированными адресами, что обеспечивает структурную и программную совместимость внутри конкретного семейства МК (рисунок 1.1).
- Максимальную эффективность использования внутреннего ОЗУ обеспечивает возможность работы с фрагментированными данными. Поддерживаются режимы адресации слов, полуслов, байт.
- В зоне ОЗУ и периферийных устройств выделены области, в которых можно выполнять битовые операции (Bit Banding). Это позволяет уменьшить размер кода программы (по сравнению с маскированием и применением логических операций) и время его выполнения.
- При обращении к Flash-памяти команд реализована опережающая выборка, предсказание ветвлений и трехуровневый конвейер.

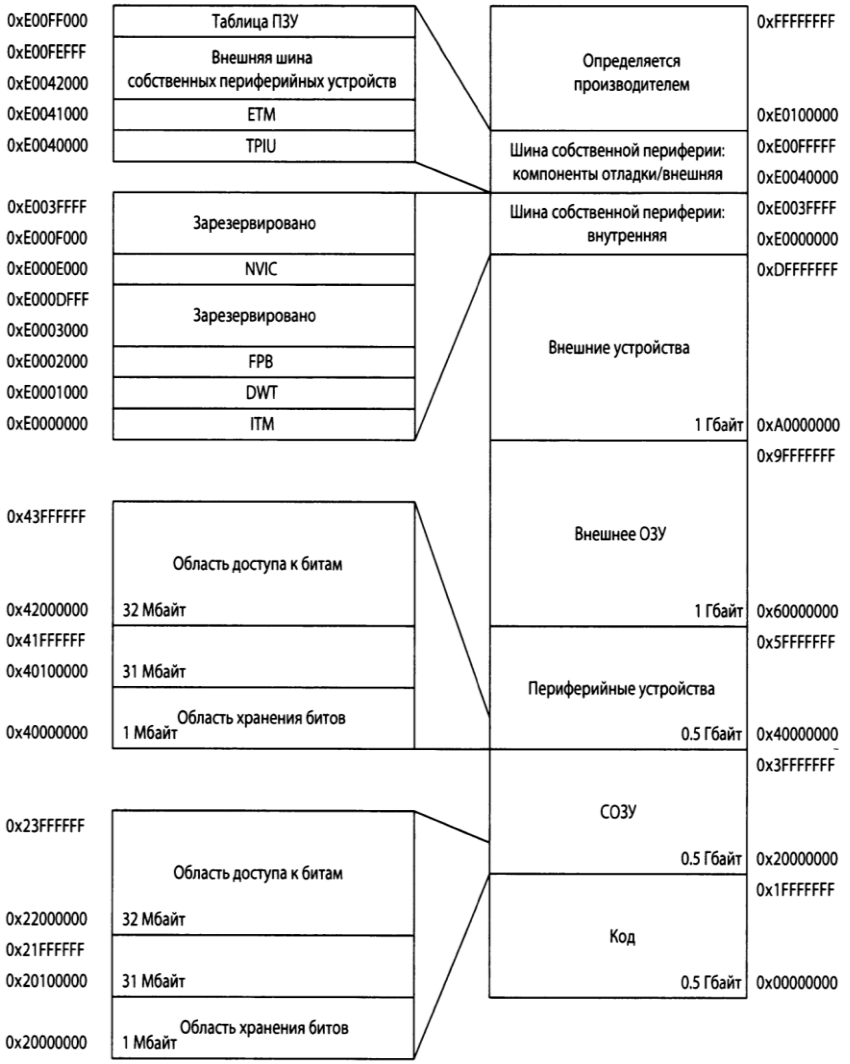



Рис. 1.1 – Адресное пространство STM32

- Используются унифицированные блоки, структура и программная модель которых в основном не зависит от типа МК. Например, **процессорное ядро, счетчики – таймеры TIMn,**

**последовательные порты USARTm и так далее во всех типах МК будут иметь одинаковую структуру и программную модель, где n, m – порядковый номер устройства.** Это обеспечивает переносимость программного обеспечения с одного типа МК на другой.

- Повышена степень интеллектуальности периферийных устройств за счет аппаратной реализации ряда функций, что приводит к повышению быстродействия и снижает степень участия процессора при обработке информации.
- Увеличено количество внутренних проблемно-ориентированных магистралей, обеспечивающих одновременный обмен между несколькими устройствами на основе матрицы шин.
- Обеспечивается программное управление синхронизацией внутренних магистралей и периферийных устройств вплоть до полного отключения их частоты, что позволяет снижать энергопотребление.
- Используются средства, обеспечивающие более простую компоновку печатной платы. К их числу относятся программируемое перераспределение (**remapping)** входов/выходов периферийных устройств на определенные линии ввода/вывода, **«pin-to-pin»** совместимость**. «Pin-to-pin»** совместимость обеспечивает **возможность замены** корпусов микросхем без изменения печатной платы **для МК с одним количеством линий ввода/вывода**. **Все сигналы сохраняются на одноименных линиях независимо от типа МК.** Это позволяет в зависимости от решаемых задач выбирать требуемый объем флеш-памяти, ОЗУ, состав периферийных блоков, **не изменяя печатную плату**. Однако эту функцию необходимо контролировать.

На рисунке 1.2 представлена упрощенная структура микроконтроллера STM32.

***Bus matrix*** **(**матрица шин**)** – контроллер высокоскоростных шин, обеспечивающий независимую связь и арбитраж между системной шиной и шиной данных ядра, DMA, Ethernet (masters) и периферией – SRAM, FLASH, AHB (slaves).

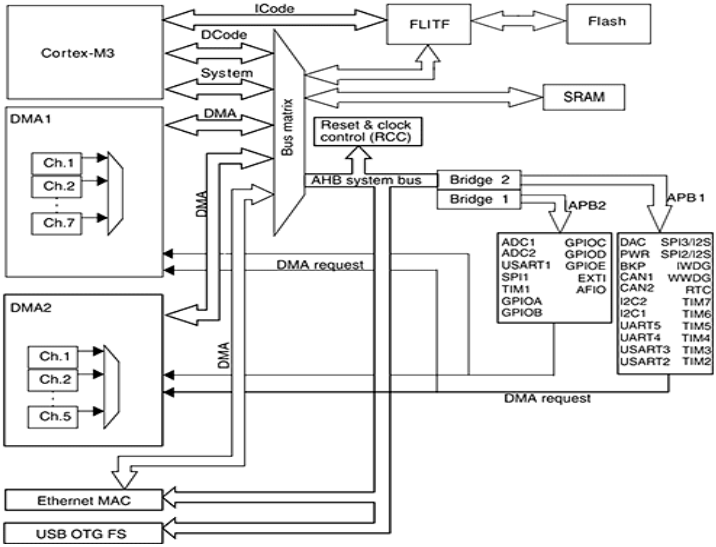



Рис. 1.2 – Упрощенная структура микроконтроллера STM 32 с ядром Cortex-M3

Flash interface ***(FLITF)*** – интерфейс Flash-памяти, обеспечивающий чтение, запись, стирание, чтение с буфером предварительной выборки, защиту памяти (от записи или чтения).

***ICode*** *bus* – шина команд (инструкций) и векторов прерываний.

***DCode*** *bus* – шина данных. Обеспечивает связь ядра с интерфейсом данных, Flash для выборки/записи данных и отладочного доступа. Эта шина может использоваться для загрузки и исполнения команд, расположенных в ОЗУ, но время их выполнения будет больше, чем при использовании флеш-памяти, так как их выборка осуществляется через системную шину.

***System*** *bus* – системная шина ядра, связывающая ядро и периферийные устройства.

***AHB system bus*** (Advanced High-performance Bus) – шина, которая связывает матрицу шин и периферийные шины ***APB*** (Advanced Peripheral Bus) с помощью мостов ***(Bridge)***.

Быстродействующие компоненты, Ethernet и USB OTG FS подключены к шине AHB напрямую.

Периферийные шины APB1 и APB2 предназначены для управления устройствами с различным быстродействием.

APB2 работает на частоте ядра и обслуживает наиболее скоростные устройства: АЦП (до 1 миллиона выборок в секунду), ЦАП, некоторые таймеры, порты ввода/вывода GPIO, USART1 (до 4.5 МБит/с), SPI1 (до 18 МБит/с) и так далее.

Быстродействие APB1 ограничено частотой 36 МГц (половина частоты ядра) и к ней подключены более медленные устройства (контроллеры USART, CANbus, $I^2C$, SPI/I2S, ШИМ и так далее).

***DMA*** (Direct Memory Access) – каналы прямого доступа к памяти. Контроллеры ***DMA*** подключены к шинной матрице по собственной шине и имеют прямой доступ ко всей памяти, включая ту, на которую отображены **управляющие регистры периферии**.

***Reset & Clock Control (RCC)*** – обеспечивает тактирование ядра и периферии, а также сброс контроллера. ***RCC*** позволяет в процессе работы МК переключаться между источниками тактирования внешним кварцем, керамическим резонатором и внутренней калиброванной RC-цепочкой, а также запрашивать у ***NVIC*** немаскируемое прерывание при сбое внешнего источника синхронизации и переключаться на внутренний, увеличивая надёжность системы.

## 1.1 Особенности ядра STM32

Ядро микроконтроллера (рисунок 1.3,) во многом определяет его технические характеристики. Ядро состоит из 32-разрядного микропроцессора, подсистем прерываний, системного таймера SysTick, средств отладки и управления памятью [85].

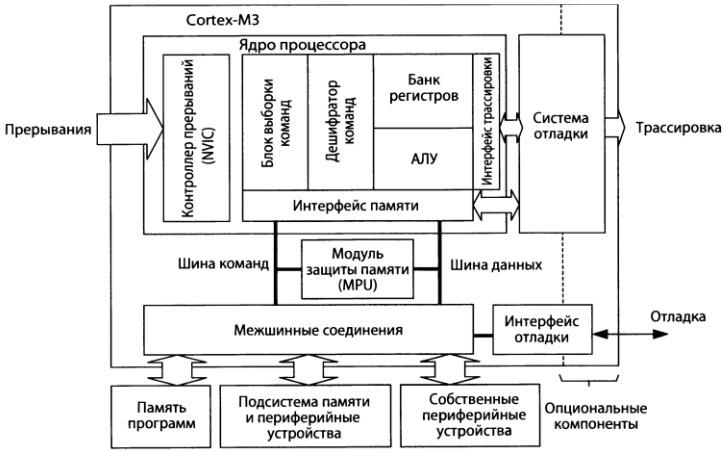



Рис. 1.3 – Ядро микроконтроллера STM32

**Микропроцессор реализует гарвардский принцип организации с использованием RISC – архитектуры, которая ориентирована в основном на однотактное выполнение арифметических и логических команд в регистрах, а связь с памятью выполняется командами записи/чтения** [50].

Особенностями системы команд являются:

- трехадресные арифметико-логические команды, при выполнении которых возможен запрет изменения флагов регистра состояния;
- расширенное количество регистровых способов адресации;
- аппаратное выполнение команд умножения и деления (знаковое и без знаковое);
- логические и арифметические сдвиги на «n» разрядов;
- наличие команд, ориентированных на задачи цифровой фильтрации (умножение с накоплением, преобразование данных с насыщением), работы с таблицами (табличный переход по индексу);
- блочные пересылки между группой регистров и памятью;

- расширенные возможности по обработке битовых переменных, включая работу с битовыми полями;
- сохранение/восстановление в стеке с помощью одной команды нескольких регистров (POP или PUSH);
- выполнение условных переходов и конвейер;
- команды выполнения последовательности типа **if – then – else,** которая позволяет реализовать до четырех условно выполняемых команд, не используя команды переходов;
- команды управления системой позволяют сбрасывать конвейер, синхронизировать работу в многопроцессорном режиме, ожидать и формировать внешние события.

Так как команды 32-разрядные, то в целях экономии объема памяти программ применяется метод компрессии/декомпрессии: в памяти программ команды кодируются в 16-разрядном формате, а при выполнении преобразуются в 32-разрядный. В микроконтроллерах STM32 используется система **Thumb-2,** состоящая как из 16-, так и 32-разрядных команд, что обеспечивает оптимальное соотношение плотности кода и производительности. **Однако в ARM Cortex M3 используется сокращенный набор команд Thumb-2.**

Уменьшение требуемого объема ОЗУ обеспечивается использованием различных форматов данных (слово, двойное слово, полуслово, байт) и возможностью работы с фрагментированными данными. На рисунке 1.4 приведена структура регистрового файла.

R0÷R12 являются регистрами общего назначения. Указатель стека R13 представлен двумя типами – основной (MSP), который включается по умолчанию и используется ядром операционной системы для исключительных ситуаций, и указатель стека процесса (PSP), необходимый в прикладных программах. Это позволяет организовывать два независимых пространства стека, что может

использоваться в операционных системах реального времени. Стек – убывающий.

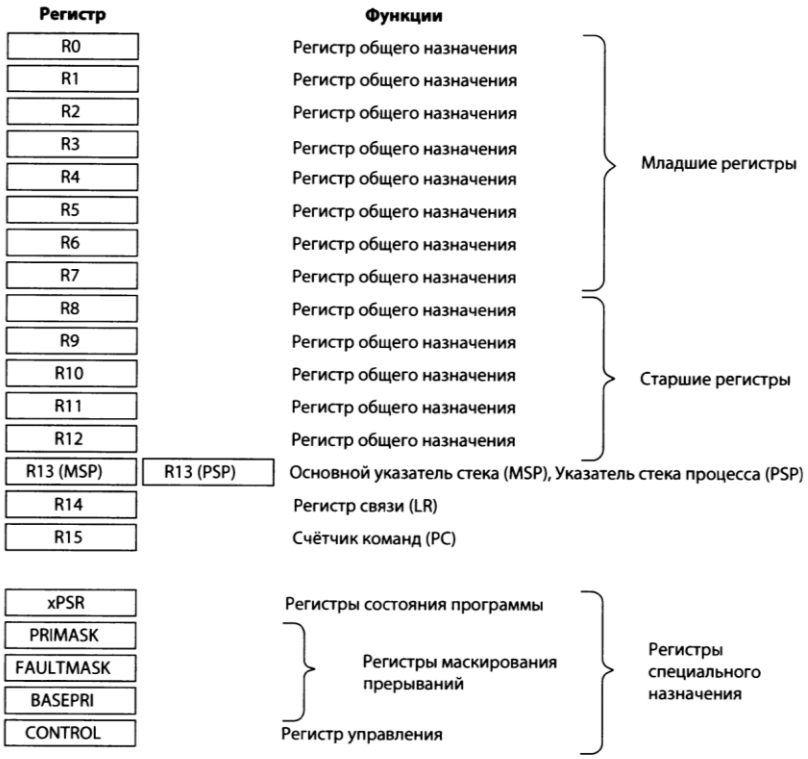



Рис. 1.4 – Структура регистрового файла STM32

Регистр связи R14 служит для сохранения адреса возврата из процедуры или функции, что сокращает время их обработки. В 8-разрядных МК адрес возврата хранится в ОЗУ.

Регистр R15 является счетчиком команд и может участвовать в операциях как обычный регистр.

Регистры специального назначения включают:

- регистр состояния программы xPSR (рисунок 1.5)

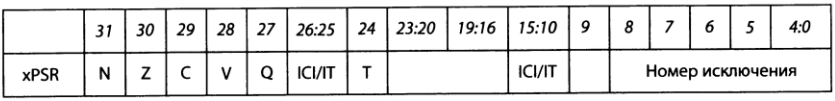



Рис. 1.5 – Объединенный регистр состояния xPSR

Возможно обращение к объединенному регистру xPSR или к регистрам, в которых записаны его отдельные поля (рисунок 1.6).

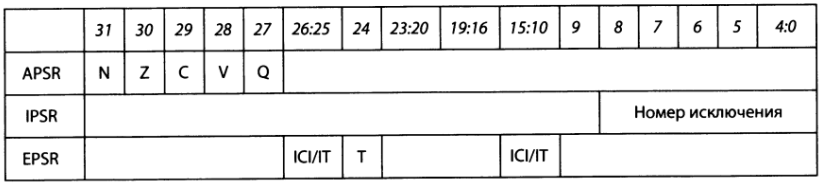



Рис. 1.6 – Разделенный регистр состояния

**APSR** – признаки выполнения команды: N – отрицательное значение, Z – результат равен 0, C – перенос/заем из/в старший разряда, V – арифметическое переполнение, Q – достижение переменной минимального или максимального значения в командах насыщения. **Флаг Q не применяется в командах условных переходов.**

- Регистр состояния прерывания IPSR хранит информацию о приостановленных (или текущих) запросах прерывания.
- Регистр состояния выполнения программы EPSR, в котором записаны T = 1 (признак работы в режиме Thumb) и ICI/IT – состояние команды, выполняемой после прерывания/состояния команды **if – then.**

Микроконтроллер в режиме Thumb выполняет альтернативный набор 16-битных команд. Большинство из этих 16-разрядных команд переводится в обычные команды ARM. Уменьшение длины команды достигается за счёт сокрытия некоторых операндов и ограничения возможностей адресации по сравнению с режимом полного набора команд ARM.

Подробное описание регистра xPSR связано с тем, что некоторые команды выполняются только при условии совпадения признаков результата выполняемой операции и содержимого APSR. Если коды не совпадают, то команда проходит по конвейеру как NOP, что минимизирует его число сбросов. Аналогичная реакция и при выполнении команды **if – then,** состояние которой отображается в поле IT.

- Регистры маскирования прерываний управляют разрешением немаскируемых прерываний. Исключением является Hard Fault (любой отказ, если соответствующий обработчик не может быть запущен) и уровень приоритета маскируемых прерываний BASEPRI. Прерывания будут запрещены, если их уровень приоритета будет ниже, чем указанный в BASEPRI.
- Регистр управления CONTROL используется для задания уровня доступа и выбора указателя стека.

Микропроцессор может работать в **режимах потока (Thread) и обработчика (Handle**). В этих режимах поддерживается два уровня доступа привилегированный и непривилегированный. В непривилегированном режиме имеются ограничения по доступу к некоторым ресурсам и специфическим областям памяти. В режиме «**Thread**» код программы может быть как привилегированным, так и непривилегированным. Переход в режим **«Handler»** происходит при возникновении исключительной ситуации (exception) и программа выполняется как привилегированная. После сброса микропроцессор находится в режиме потока с привилегированным доступом. В этом состоянии программа может обращаться к любой области памяти и использовать полный список команд. Переключение в непривилегированный режим пользователя осуществляется в регистре CONTROL. Привилегированный уровень обычно используют операционные системы, а непривилегированный – программы пользователей.

**Контроллер векторизованных вложенных прерываний NVIC имеет унифицированную структуру и программную модель для всех МК Cortex.** В зависимости от типа микроконтроллера и количества линий ввода-вывода NVIC может обрабатывать до 240 прерываний с 256 уровнями приоритета. Более подробно контроллер прерываний для STM32F103C8T6 будет рассмотрен далее.

**Системный таймер SysTick** предназначен для формирования программируемых временных интервалов, которые могут быть необходимы: в операционных системах реального времени, при формировании задержек и периодических прерываний, измерении временных событий и так далее. Системный таймер состоит из 24-разрядного вычитающего счетчика с автоматической загрузкой начального значения, мультиплексора, коммутирующего источники тактового сигнала, и предделителей входной частоты. Обычно таймер SysTick используют для реализации системных функций и его не рекомендуют использовать в программах пользователей.

В **ядро STM32** входит только **порт доступа к средствам отладки**, с помощью которого **внешние отладчики (JTAG, SWD)** могут обращаться к ресурсам микропроцессора. В функции системы отладки входят: управление процессом выполнения программы в непрерывном режиме с точками останова, в шаговом режиме, чтение регистров общего назначения (РОН), регистров управления и состояния периферийных устройств, памяти, трассировка программы.

## 1.2 Контроллер прерываний

При выборе способа ввода/вывода необходимо учитывать, что использование прерывания снижает быстродействие операций ввода-вывода, но повышает производительность МК. Под

**производительностью** нужно понимать **число задач**, решаемых в единицу времени, а **быстродействием – время выполнения** операций [50].

Особенностью контроллера прерываний STM32 является его интеграция непосредственно в ядро микроконтроллера, унификация (структура контроллера и его программная модель не должна зависеть от особенностей процессорного блока и производителя микросхемы), аппаратная реализация ряда процедур обработки прерываний. Это обеспечивает более простое программирование обработки прерываний, компактность кода, малое время реакции на запрос, его детерминированное значение. Последнее особенно важно для систем реального времени.

Аппаратно реализуются следующие процедуры:

- анализ приоритетов запросов прерывания;
- переход к подпрограмме обработки прерываний (В ATmega, например, при разрешенном прерывании вначале выполняется обращение по адресу вектора, в котором располагается команда перехода на подпрограмму обработки прерываний);
- сохранение и восстановление контекста. Время выполнения каждой из процедур составляет 12 циклов. В стек аппаратно записываются (или читаются) регистр состояния программы xPSR, счетчик команд PC(R15), регистр связи LR(R14), регистр R12, который может использоваться как рабочий внутри подпрограммы, регистры общего назначения R3-R0. Регистры R3-R0 могут также использоваться в процедуре обработки прерываний. Стек – убывающий.
- Обработка последовательности запросов с различными приоритетами при минимальных задержках между прерываниями. Например, при обработке двух последовательных прерываний классическим методом на вход/выход из подпрограммы потребуется 48 циклов, а используемые аппаратные методы

позволяют сократить это число до 30–40. Время реакции на запрос уменьшается также за счет возможности прерывания некоторых многоцикловых команд Thumb-2 [85, 53].

Основные характеристики контроллера прерываний:

- Количество запросов маскируемых прерываний определяется конкретной моделью микроконтроллера**. Для STM32F103xx – 43 входа запросов от периферий, 16 – от ядра Cortex M3, одно немаскируемое прерывание – NMI**. Управление действующими прерываниями выполняют регистры разрешения или маски прерываний. Возможен запрет всех прерываний.

- Каждому из прерываний соответствует индивидуальный фиксированный адрес-вектор, в котором хранится адрес подпрограммы обработки прерываний. Один вектор прерываний может обслуживать, как правило, несколько событий. Поэтому **процедура обработки должна использовать поллинг** (последовательный опрос флагов готовности внешних устройств).

- Многоуровневые программируемые прерывания. **Количество уровней приоритетов** – **16.** Наиболее приоритетное – прерывание меньшим номером. Установка приоритетов выполняется в специальном блоке регистров. На каждое прерывание в нем отведен один байт. Обработка прерывания может быть прервана другим прерыванием с более высоким приоритетом. Прерывания с таким же или более низким приоритетом будут ожидать окончания обработки активного обработчика прерываний. Допустимо динамическое изменение приоритетов.

- Возможна организация очереди обрабатываемых прерываний, управление ею и формирование информации об активных (находящихся в обработке) запросах. Для этих целей предусмотрены регистры действующих прерываний, установки ожидания обработки прерываний, сброса ожидания обработки прерываний. В блоке регистров ожидания обработки прерываний

каждый бит соответствует конкретному прерыванию и показывает, ожидает ли оно обработки. Структура контроллера прерываний представлена на рисунке 1.7 [61].

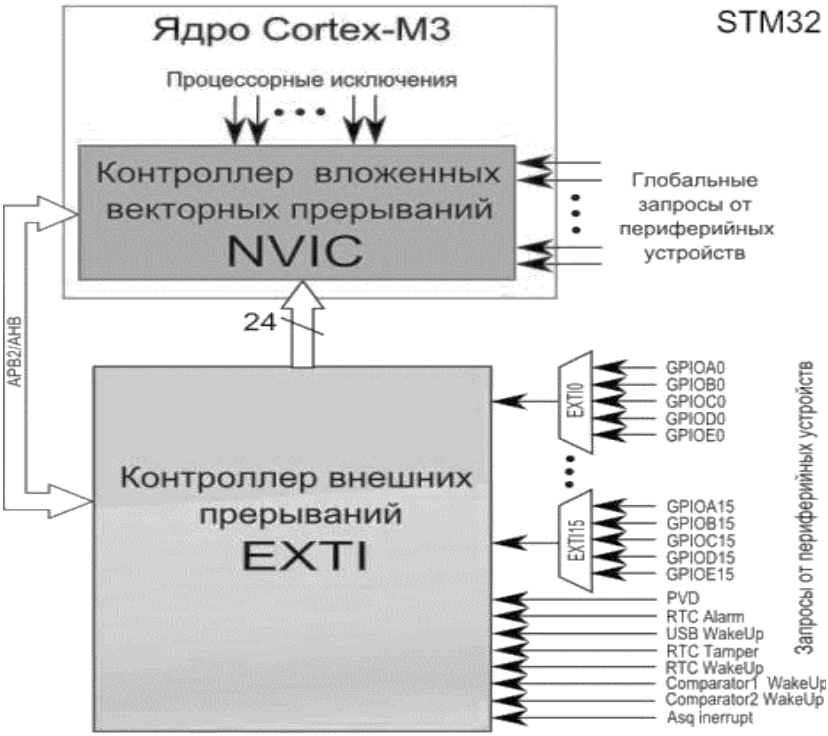



Рис. 1.7 – Структура контроллера прерываний в STM32

Для принудительного перевода прерываний в состояние ожидания биты регистров могут быть установлены или сброшены.

- Прерывания могут быть вызваны как аппаратно, так и программно. Программный вызов прерывания выполняется записью соответствующего номера в регистр программного вызова прерываний.

- Внешние прерывания могут быть настроены на фронт и/или срез входного сигнала.
- **Минимальная длительность сигнала запроса** внешнего прерывания должна быть больше периода частоты тактового генератора периферийной шины APB2.
- Время реакции на запрос – 12 циклов.

Основные функции обработки прерываний выполняет контроллер вложенных векторных прерываний **NVIC.** На его вход поступают запросы:

- системных прерываний и исключений, часть из которых недоступна пользовательским программам;
- внутренних (резидентных) периферийных устройств;
- запросы контроллера внешних прерываний **EXTI.**

Различают прерывания ядра, которые имеют наивысший приоритет и не подчиняются команде общего запрета/разрешения прерываний. Для них существуют свои флаги и свои команды: enable_fiq() и disable_fiq().

Особенности работы с системными прерываниями и исключениями в данной работе не рассматриваются. Типы исключений приведены в таблице 1.2 [85].

При настройке функционирования контроллера к некоторым системным регистрам прерывания необходимо обращаться. Разрядность перечисленных регистров отражена в таблице 1.3.

Функции NVIC:

- разрешение **(NVIC_ISERx)** или запрещение **(NVIC_ICERx)** прерываний, управление приоритетами **(NVIC_IPRx)**;
- программный вызов прерываний **(NVIC_STIR)**;
- управление очередью прерываний: **NVIC_ISPRx** (включить в очередь ожидания прерываний), **NVIC_ICPRx** (удалить из очереди ожидания прерываний), **NVIC_IABRx** (состояние прерывания – закончилась обработка или нет).

Таблица 1.2. Типы исключений Cortex-M3

| Номер исключения | Тип исключения | Приоритет | Описание |
|---|---|---|---|
| 0 | — | — | Исключение отсутствует |
| 1 | Reset | -3 (Наивысший) | Сброс |
| 2 | NMI | -2 | Немаскируемое прерывание (вход внешнего немаскируемого прерывания) |
| 3 | Hard Fault | -1 | Любой отказ, если соответствующий обработчик не разрешён |
| 4 | MemManage Fault | Программируемый | Отказ системы управления памятью; нарушение правил доступа, заданных модулем MPU, или обращение по некорректному адресу |
| 5 | Bus Fault | Программируемый | Отказ шины (отказ предвыборки или отказ данных) |
| 6 | Usage Fault | Программируемый | Отказ программы |
| 7...10 | Зарезервировано | — | Зарезервировано |
| 11 | SVCall | Программируемый | Вызов супервизора |
| 12 | Debug monitor | Программируемый | Исключение монитора отладки (точки останова, точки наблюдения или внешняя команда отладки) |
| 13 | Зарезервировано | — | Зарезервировано |
| 14 | PendSV | Программируемый | Запрос системной службы |
| 15 | SYSTICK | Программируемый | Системный таймер |
| 16 | IRQ #0 | Программируемый | Внешнее прерывание №0 |
| 17 | IRQ #1 | Программируемый | Внешнее прерывание №1 |
| … | … | … | … |
| 255 | IRQ #239 | Программируемый | Внешнее прерывание №239 |

Таблица. 1.3. Разрядность регистров прерываний

| Преры-вания | Элементы массива CMSIS | | | | |
|---|---|---|---|---|---|
| | Разрешение прерывания | Запрет прерывания | Включить в очередь ожидания прерывания | Удалить из очереди ожидания прерывания | Состояние обработки прерываний |
| 0-31 | ISER[0] | ICER[0] | ISPR(0] | ICPR[0] | IABR[0] |
| 32-63 | ISER[1] | ICER[1] | ISPR[1) | ICPR[1] | IABR[1] |
| 64-67 | ISER[2] | ICER(2] | ISPR(2] | ICPR[2] | IABR[2] |

Номера векторов прерываний определяются из таблицы 59, представленной в работе [17]. Фрагмент этой таблицы приведен в таблице 1.4.

Например, номер прерывания для **TIM2 – 28**, а для **USART1 – 37**. Тогда управление запросами прерываний **TIM2** следует выполнять в **регистрах с индексом 0**, а **USART1 – с индексом 1**.

Таблица. 1.4. Номера векторов прерываний

| позиция | приоритет | Тип приоритета | Аббревиатура | Описание | Адрес |
|---|---|---|---|---|---|
| 19 | 26 | устанавливаемый | CAN1_TX | CAN1 TX interrupts | 0x0000_008C |
| 20 | 27 | устанавливаемый | CAN1_RX0 | CAN1 RX0 interrupts | 0x0000_0090 |
| 21 | 28 | устанавливаемый | CAN1_RX1 | CAN1 RX1 interrupt | 0x0000_0094 |
| 22 | 29 | устанавливаемый | CAN1_SCE | CAN1 SCE interrupt | 0x0000_0098 |
| 23 | 30 | устанавливаемый | EXTI9_5 | EXTI Line[9:5] interrupts | 0x0000_009C |
| 24 | 31 | устанавливаемый | TIM1_BRK | TIM1 Break interrupt | 0x0000_00A0 |
| 25 | 32 | устанавливаемый | TIM1_UP | TIM1 Update interrupt | 0x0000_00A4 |
| 26 | 33 | устанавливаемый | TIM1_TRG_COM | TIM1 Trigger and Commutation interrupts | 0x0000_00A8 |
| 27 | 34 | устанавливаемый | TIM1_CC | TIM1 Capture Compare interrupt | 0x0000_00AC |
| 28 | 35 | устанавливаемый | TIM2 | TIM2 global interrupt | 0x0000_00B0 |

*Окончание табл. 1.4*

| позиция | приоритет | Тип приоритета | Аббревиатура | Описание | Адрес |
|---|---|---|---|---|---|
| 29 | 36 | устанавливаемый | TIM3 | TIM3 global interrupt | 0x0000_00B4 |
| 30 | 37 | устанавливаемый | TIM4 | TIM4 global interrupt | 0x0000_00B8 |
| 31 | 38 | устанавливаемый | I2C1_EV | $I^2C1$ event interrupt | 0x0000_00BC |
| 32 | 39 | устанавливаемый | I2C1_ER | $I^2C1$ error interrupt | 0x0000_00C0 |
| 33 | 40 | устанавливаемый | I2C2_EV | $I^2C2$ event interrupt | 0x0000_00C4 |
| 34 | 41 | устанавливаемый | I2C2_ER | $I^2C2$ error interrupt | 0x0000_00C8 |
| 35 | 42 | устанавливаемый | SPI1 | SPI1 global interrupt | 0x0000_00CC |
| 36 | 43 | устанавливаемый | SPI2 | SPI2 global interrupt | 0x0000_00D0 |
| 37 | 44 | устанавливаемый | USART1 | USART1 global interrupt | 0x0000_00D4 |
| 38 | 45 | устанавливаемый | USART2 | USART2 global interrupt | 0x0000_00D8 |
| 39 | 46 | устанавливаемый | USART3 | USART3 global interrupt | 0x0000_00DC |
| 40 | 47 | устанавливаемый | EXTI15_10 | EXTI Line[15:10] interrupts | 0x0000_00E0 |
| 41 | 48 | устанавливаемый | RTCAlarm | RTC alarm through EXTI line interrupt | 0x0000_00E4 |
| 42 | 49 | устанавливаемый | OTG_FS_WKUP | USB On-The-Go FS Wakeup through EXTI line interrupt | 0x0000_00E8 |
| — | — | — | — | Reserved | 0x0000_00EC - 0x0000_0104 |
| 50 | 57 | устанавливаемый | TIM5 | TIM5 global interrupt | 0x0000_0108 |
| 51 | 58 | устанавливаемый | SPI3 | SPI3 global interrupt | 0x0000_010C |
| 52 | 59 | устанавливаемый | UART4 | UART4 global interrupt | 0x0000_0110 |
| 53 | 60 | устанавливаемый | UART5 | UART5 global interrupt | 0x0000_0114 |
| 54 | 61 | устанавливаемый | TIM6 | TIM6 global interrupt | 0x0000_0118 |
| 55 | 62 | устанавливаемый | TIM7 | TIM7 global interrupt | 0x0000_011C |

**Регистр установки приоритета** прерывания **NVIC_IPRx** состоит из 16 регистров (**x** – от 0 до 16), в каждом из которых кодируются приоритеты 4 источников прерываний. Для STM32F103xx, у которого 16 приоритетов, они указываются в старшей тетраде (четырех разрядах) IPx[7:4]. Например, для задания приоритета 10 прерывания **IRQ 5** – **NVIC_IPR1 = 0xA0.**

Для **программного вызова прерывания** необходимо установить в регистре управления **SCB_CCR** бит **USERSETMPEND** и записать в регистр **NVIC_STIR** порядковый номер прерывания.

По умолчанию таблица векторов хранится в начальной зоне памяти. Но возможно её перемещение и в другие области с помощью соответствующей настройки регистра системных прерываний **SCB_VTOR.**

Для работы с внешними прерываниями и некоторыми внутренними периферийными устройствами предназначен контроллер внешних прерываний **EXTI.** Его применение обеспечивает повышенную функциональность при обработке прерываний внешних и внутренних событий. В микроконтроллере **STM32F103x** на вход контроллера поступают **16 внешних запросов прерываний (EXTI0 – EXTI15),** которые могут быть подключены к линиям портов **GPIO,** и внутренние события: **EXTI16** – выход программируемого детектора напряжения **PVD, EXTI17** – события часов реального времени **RTC Alarm**, **EXTI18** – события **USB, EXTI19** – события **контроллера Ethernet** (при его наличии в выбранном микроконтроллере).

Подключение линий GPIO к 16 внешним линиям прерываний показано на рисунке 1.8.

Одинаковые разряды портов (например, 15 разрядов всех портов) объединены по схеме ИЛИ и формируют один запрос прерываний.

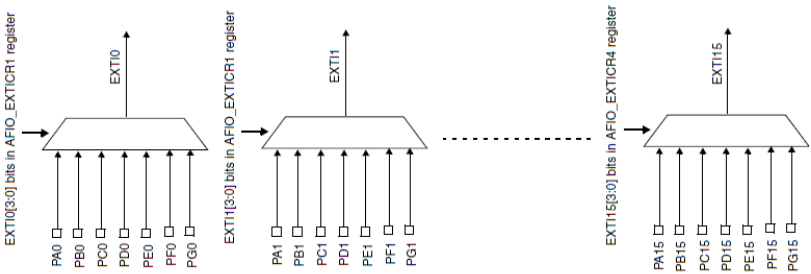



Рис. 1.8 – Внешние линии прерывания микроконтроллера

Необходимо отметить следующие особенности **EXTI** при работе с внешними прерываниями:

- внешние прерывания должны быть распределены **между различными разрядами портов**. Нельзя **одновременно** подключить прерывания, например, PA0 и PB0, PA1 и PB1, и т.д. Большим приоритетом обладают запросы с меньшим номером;
- запросы **EXTI0 – EXTI4 (IRQ6 – IRQ10)** имеют индивидуальные вектора прерываний;
- запросы **EXTI 9 – 5 (IRQ 23)** имеют общий вектор прерываний;
- запросы **EXTI 15 – 10 (IRQ 40)** имеют общий вектор прерываний;
- внешние прерывания могут быть настроены на фронт и/или срез входного сигнала;
- минимальная длительность входного сигнала должна быть больше периода частоты тактового генератора периферийной шины APB2.

Структура контроллера внешних прерываний представлена на рисунке 1.9.

Входной сигнал поступает с **Input Line** на вход соответствующего канала **EXTI**. Контроллер может быть настроен на восприятие событий по фронту, срезу или одновременно на

фронт и срез. Детектор **Edge detect circuit** управляется регистрами выбора срабатывания по фронту **(EXTI_RTSR)** и срезу **(EXTI_FTSR).**

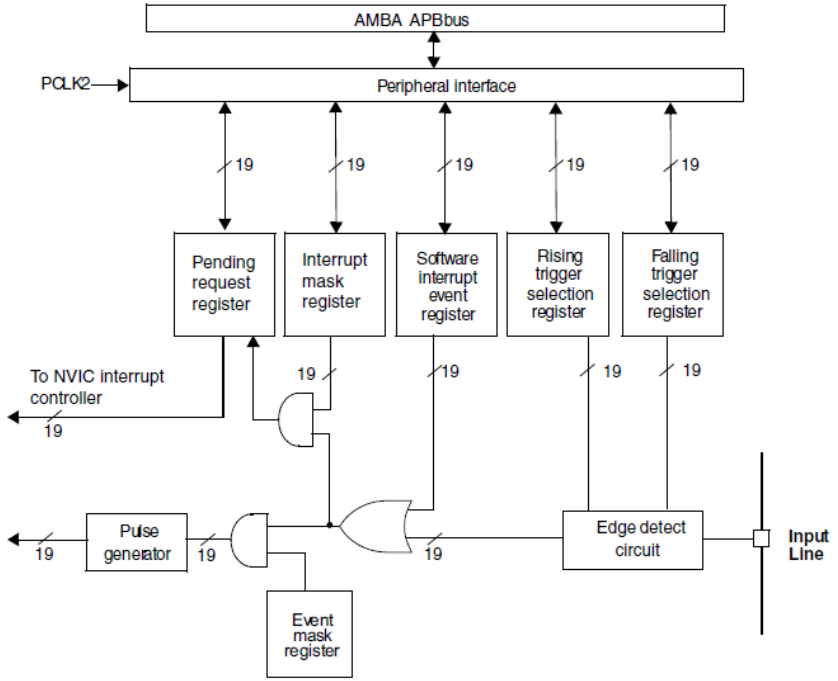



Рис. 1.9 – Структура контроллера внешних прерываний EXTI

Прерывание может быть вызвано аппаратно или программно. Программный вызов определяется состоянием регистра **EXTI_SWIER**. С выхода элемента «**ИЛИ**» сигнал поступает на схемы «**И**», которые формируют запросы на прерывание (событие), если установлены соответствующие разряды в регистрах маски внешних прерываний **EXTI_IMR** и внешних событий **EXTI_EMR**. Если прерывание разрешено, то устанавливается бит в регистре ожидания прерывания **EXTI_PR** и формируется запрос к **NVIC**. Номера бита в регистрах соответствуют номеру **EXTI (EXTI0 –**

**нулевой разряд, EXTI5-пятый разряд, EXTI15 – пятнадцатый разряд)**. Остальные настройки следует выполнять в NVIC.

**Формирование событий** выполняется аналогичным образом под управлением регистра маски **EXTI_EMR**. Если соответствующее событие (фронт или срез) произошло, и оно разрешено, то генерируется импульс **Pulse generator**. **Флаг запроса при этом не устанавливается.**

Программная модель блока прерываний приведена в приложении П1.1.

**Общий алгоритм обработки прерываний при наличии внешних прерываний:**

- Определить линии портов, на которые будут подаваться запросы внешних прерываний (см. раздел 1.6.1).
- Задать режимы работы выбранных линий **(GPIOx_CRL, GPIOx_CRH).(CNF, MODE).**
- Активизировать альтернативные функции **(AFIO_EXTICR1 – AFIO_EXTICR4),** указав в соответствующих тетрадах номер используемого порта. Например, **для EXTI0 порта «B» в регистре AFIO_EXTICR1 в младшей тетраде следует записать 0001b, для EXTI15 порта «D» в разряды (15-12) регистра AFIO_EXTICR4 следует записать 0011b**. По умолчанию (при сбросе или включении МК) в AFIO_EXTICR1÷AFIO_EXTICR4 записаны значения «0». Это значит, что **PA настраивается автоматически на восприятие внешних запросов прерываний.** Но рекомендуется порты для приема запросов прерываний **инициализировать программно**.
- Включить тактирование выбранных портов **(RCC_APB2ENR).**
- Задать активный фронт/срез **(EXTI_RTSR и EXTI_FTSR).**
- Разрешить требуемые прерывания **(EXTI_IMR).**

- Разрешить прерывание в **NVIC_ISERx** в соответствие с его номером.
- Задать приоритет прерывания в **NVIC_IPRx** в поле, соответствующем его номеру прерывания.
- Разрешить глобальное прерывание.

Обработка прерываний от периферийных устройств в **NVIC** выполняется аналогично. Необходимо только в соответствующих регистрах периферийных устройств установить биты разрешения прерываний.

## 1.3 Системный таймер

Системный таймер **SysTick (STK)** предназначен для формирования различных временных событий, необходимых при работе МК: временные задержки, генераторы, источники синхронизации, разделение времени между процессами в операционных системах реального времени и так далее.

**Системный таймер активно используется в функциях HAL, поэтому его не рекомендуется применять в программах пользователя.**

**SysTick** является 24-разрядным вычитающим счетчиком с режимом автоперезагрузки. Источником входного сигнала является частота ядра **HCLK** с программируемым коэффициентом деления 1 или 8, которая определяется битом **CLKSOURCE** регистра управления и состояния **STK_CTRL**. Если в режиме пониженного энергопотребления генератор ядра остановлен, то содержимое счетчика не изменяется.

Запуск – программный (**STK _CTRL.ENABLE).**

При переходе счетчика таймера **STK_CVR** из состояния 0 в состояние 1 (переполнение) в регистре **STK_CTRL** устанавливается флаг **COUNTFLAG.**

Состояние переполнения таймера можно проверить программно или по прерыванию, если установлен бит разрешения **TICKINT** в регистре **STK_CTRL.**

При использовании **STK** в качестве источника сигнала с повышенной точностью необходимо контролировать регистр калибровки **STK_CALIB.** В режиме калибровки на вход **STK** поступает частота **HCLKmax/8,** а калибровочное значение хранится в поле **TENMS[23:0]** регистра **STK_CALIB.** Конкретное значение **TENMS** зависит от особенностей микросхемы МК и определяется в процессе производства. Когда **HCLK** запрограммирован на максимальную частоту, период таймера **SysTick равен 1 мс.** Если информация калибровки неизвестна, можно вычислить значение калибровки по тактовой частоте процессора [73]. Достоверность калибровки определяет бит **STK_CALIB.NOREF,** информирует о наличии на входе эталонной частоты, и **STK_CALIB.SKEW** – указывает на точность значения в поле **TENMS (0** – значение точное, 1 – значение неточное или не задано). Программная модель STK приведена в приложении П1.2.

Алгоритм управления **STK**:

- Задать требуемый период переполнения таймера, который определяется входной тактовой частотой (**STK_CTRL. CLKSOURCE)** и содержимым регистра перезагрузки **STK_RVR**. Для генерации **непрерывной последовательности** сигналов переполнения с периодом **N** тактов процессора необходимо в **STK_RVR** записать число **N-1,** а для однократного использования – **N.**

**При выборе длительности периода необходимо учитывать время, необходимое для выполнения программ обработки флага переполнения.**

- Выбрать способ контроля флага переполнения **COUNTFLAG –** программный или по прерыванию (**TICKINT)**.
- Запустить таймер (**STK_CTRL.ENABLE).**

Текущее значение таймера можно прочитать в счетчике **STK_CVR.** В момент достижения нулевого значения счетчика по фронту **следующего входного импульса** содержимое **STK_RVR** перегружается в **STK_CVR.** Однако, **запись в регистр STK_CVR вызывает его сброс в «0»** и обнуление бита **COUNTFLAG** в регистре **STK_CTRL**.

**Обращение к регистрам системного таймера должно быть, как к слову.**

**После сброса требуется повторная инициализация STK,** так как текущее состояние регистров таймера неопределенное.

## 1.4 Контроллер прямого доступа к памяти

**Контроллер прямого доступа к памяти** КПДП (или контроллер DMA**) не относится к ядру МК,** но, так как он является **ресурсом общего пользования** в большинстве процедур обработки, рационально рассмотреть его особенности в этом разделе.

**Режим ПДП обеспечивает высокоскоростной обмен данными между памятью и периферийными устройствами без участия процессора**. Возможны различные типы обмена: «память-периферийное устройство», «память-память», «периферийное устройство – периферийное устройство».

В отличие от программного обмена и/или обработки прерываний, выполняемых процессором, режим ПДП реализует обмен данными аппаратно (без использования вычислительных мощностей микроконтроллера). Однако, это требует организации параллельных по отношению к процессору внутренних магистралей (шин). По сравнению с прерыванием обмен в режиме ПДП не требуется сохранение/восстановление контекста.

При обмене может возникнуть ситуация, при которой ресурсы МК требуются одновременно процессору и КПДП. В этом случае

используется процедура арбитража, которая замедляет скорость обмена. Кроме этого, могут возникнуть аналогичные конфликты между каналами КПДП. Поэтому эффект от использования режима ПДП следует оценивать с учетом особенностей архитектуры МК. Использование КПДП может привести, например, к увеличению производительности, но снижению быстродействия (скорости ввода/вывода). КПДП наиболее эффективен при реализации блочного обмена.

Особенностью функционирования КПДП МК STM32F103C8T6 является наличие матрицы шин, работающей по алгоритму round-robin (циклическое планирование), гарантируя, по крайней мере, половину пропускной способности системной шины для ЦПУ (справедливо при обращении как к памяти, так и к периферии).

Применение ПДП в МК STM32F103C8T6 целесообразно при реализации блочного (пакетного) обмена, так как в большинстве периферийных устройств (**USART**, **SPI**, **$I^2C$**, **АЦП**) отсутствует буферная память и требуется согласование с памятью передаваемых форматов данных.

Основные характеристики КПДП:

- Контроллер ПДП – один (DMA1).
- Количество независимых каналов – 7.
- За каждым каналом закреплены **аппаратные запросы** определенных периферийных устройств и возможность формирования **программного запроса MEM2MEM** (рисунок 1.10). Аппаратные запросы активируются в регистрах управления соответствующих периферийных устройств, а программные – в регистре конфигурации КПДП **DMA_CCRx,** где x – номер канала.

Запросы периферийных устройств представлены в таблице 1.5.

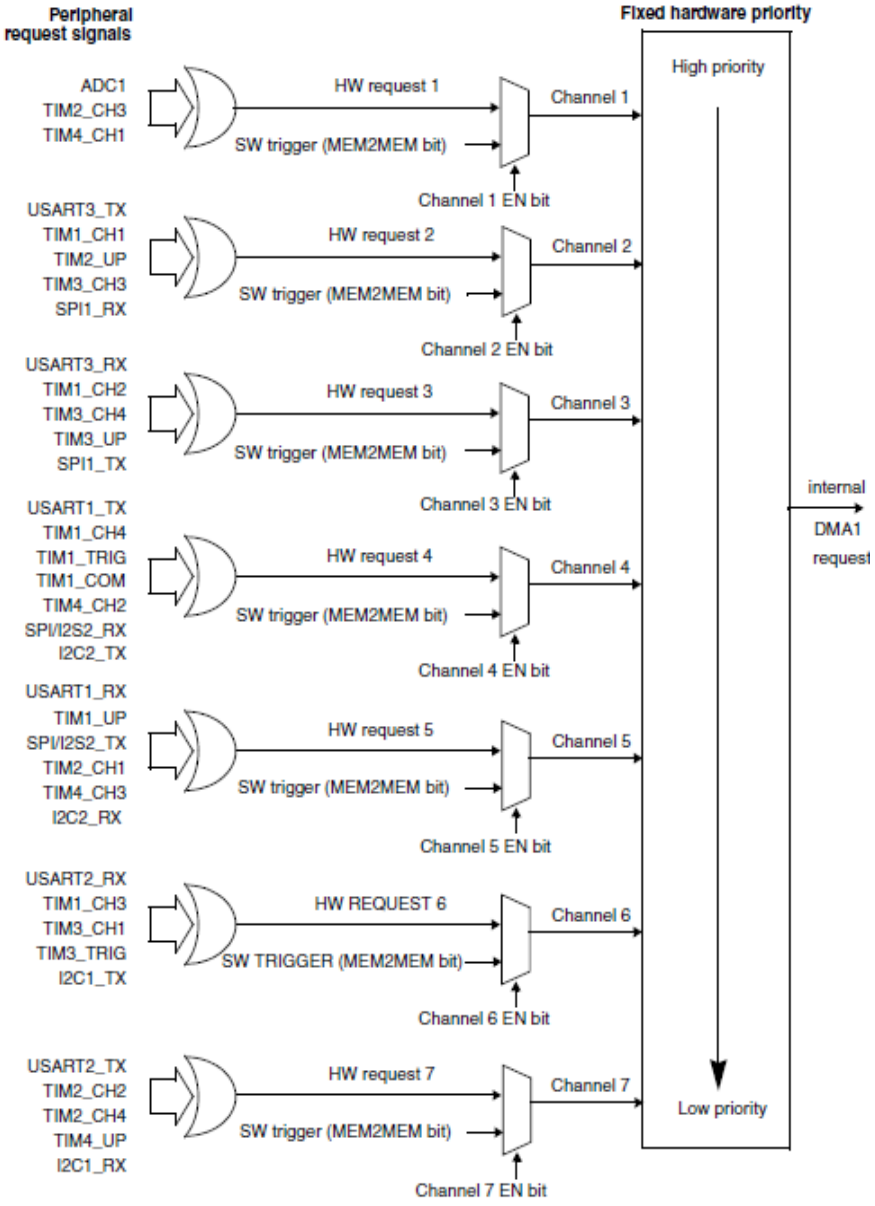



Рис. 1.10 – Запросы ПДП контроллера DMA1

Таблица 1.5. Запросы периферийных устройств DMA1

| Периферия | Канал 1 | Канал 2 | Канал 3 | Канал 4 | Канал 5 | Канал 6 | Канал 7 |
|---|---|---|---|---|---|---|---|
| ADC1 | ADC1 | | | | | | |
| SPI/I²S | | SPI1_RX | SPI1_TX | SPI2/I2S2_RX | SPI2/I2S2TX | | |
| USART | | USART3_TX | USART3_RX | USART1_TX | USART1_RX | USART2_RX | USART2_TX |
| I²C | | | | I2C2_TX | I2C2_RX | I2C1_TX | I2C1_RX |
| TIM1 | | TIM1_CH1 | TIM1_CH2 | TIM1_CH4<br>TIM1 TRIG<br>TIM1_COM | TIM1_UP | TIM1_CH3 | |
| TIM2 | TIM2_CH3 | TIM2_UP | | | TIM2_CH1 | | TIM2_CH2<br>TIM2_CH4 |
| TIM3 | | TIM3_CH3 | TIM3_CH4<br>TIM3_UP | | | TIM3_CH1<br>TIM3_TRIG | |
| TIM4 | TIM4_CH1 | | | TIM4_CH2 | TIM4_CH3 | | TIM4_UP |

- Каналы КПДП работают в режиме с разделением времени – **активным может быть только один канал.** Управление приоритетами запросов выполняет арбитр. Приоритеты между каналами устанавливаются на двух уровнях.

Программный приоритет каждого канала задается в регистре **DMA_CCRx биты PL0, PL1**: 00 – низкий приоритет, 01– средний, 10 – высокий, 11 – очень высокий.

Аппаратный приоритет определяется номером канала контроллера: канал с меньшим номером обладает большим приоритетом. **Аппаратный приоритет является вторичным**. Если у нескольких запросов установлен одинаковый программный приоритет, то активным будет канал с меньшим номером. Если же запросы будут иметь различные уровни программного приоритета, то активным будет запрос, у которого в **PL0, PL1** записано большее число.

- Обеспечивается аппаратное согласование (выравнивание) форматов предаваемых данных между приемником и

передатчиком, если они имеют вид байта, полуслова, слова. Например, порт SPI может передавать данные в виде байта или полуслова (2 байта), а разрядность памяти данных – слово (4 байта). Размер передаваемых данных указывается в **DMA_CCRx** битовыми полями **PSIZE0, PSIZE1** (для периферийного устройства) и **MSIZE0, MSIZE1** (для памяти). С правилами выравнивания данных можно ознакомиться [17, таблица 63].

- Аппаратное управление непрерывным потоком данных выполняется в **режиме цикличности** с помощью **кольцевого буфера**. Если этот режим активирован, то после передачи всех данных регистр размера данных канала **DMA_CNDTRx** перезагружается исходным значением и передача продолжается. Этот режим целесообразно использовать, например, при циклической работе АЦП в режиме сканирования, при блочных передачах последовательных портов и так далее.
- Программируемый размер данных для обмена – до 65536.
- Аппаратный инкремент адреса памяти и периферийного устройства, определяемый длиной передаваемых данных: байт – 1, полуслово – 2, слово – 4. Управление инкрементом выполняется соответственно битами **MINC** и **PINC** регистра **DMA_CCRx**. Значения этих бит определяются типом обмена. Например, в режиме «память-память» следует инкрементировать оба регистра адреса, в режиме «периферия-память» – только адрес памяти, в режиме «периферия-периферия» инкремент может не требоваться вообще.
- В процессе обмена КПДП формирует три флага событий (окончание обмена в канале **TCIFx,** половина обмена **HTIFx** и ошибка обмена **TEIFx**), которые логически объединяются с помощью функции «**ИЛИ**» и устанавливают флаг **GIFx**. Флаги хранятся в регистре состояния прерываний **DMA_ISRx**.

**Флаг ошибки обмена** может возникнуть при операции, чтении или записи в **зарезервированном адресном пространстве.** Когда происходит ошибка обмена во время чтения или записи, такой канал автоматически отключается посредством аппаратного сброса бита **EN** в регистре конфигурации соответствующего канала (**DMA_CCRx**) [17].

- Флаги событий могут вызывать прерывание, если они разрешены в регистре конфигурации канала **DMA_CCRx** (таблица 1.6). **Каждый канал имеет один индивидуальный вектор прерывания.** Поэтому для идентификации событий следует использовать polling.

Таблица 1.6. Запросы прерываний контроллера ПДП

| Событие прерывания | Флаг события | Разрешение прерывания |
|---|---|---|
| Half-transfer | HTIF | HTIE |
| Transfer complete | TCIF | TCIE |
| Transfer error | TEIF | TEIE |

- Контроллер реализует различные типы обмена («память-память», «периферия-память», «память-периферия», «периферия-периферия»), обеспечивая доступ к Flash, SRAM, периферии, шинам периферии APB1, APB2 и AHB.

- Каждая передача, осуществляемая КПДП, состоит из четырех фаз: выборки и арбитража, вычисления адреса, доступ к шине и подтверждения. Длительность фазы доступа к шине для передачи слова данных зависит от используемых шин и может длиться от 3 до 5 циклов, а остальные фазы – 1 цикл. Например, передача из SPI в ОЗУ = передача SPI (APB) + передача ОЗУ (AHB) + свободный цикл (AHB) = (2 цикла APB + 2 цикла AHB) + 2 цикла AHB + 1 цикл AHB = 2 цикла APB + 5 циклов AHB [53].

На стадии инициализации КПДП является ведомым устройством, а процессор – ведущим.

Процедура инициализации канала **«x»** в **режиме «периферия-память»** состоит в следующем:

- задать адреса в регистрах периферии (**DMA_CPARx**) и памяти (**DMA_CMARx)**;
- длину передаваемого массива данных записать в регистр **DMA_CNDTRx**;
- указать приоритет канала с помощью битов **PL[1:0]** в регистре **DMA_CCRx**;
- задать основные параметры канала в регистре **DMA_CCRx** (направление обмена, режим цикличности, включить или выключить режим инкремента и указать размер передаваемых данных как для памяти, так и для периферийного устройства, разрешить или запретить прерывания);
- разрешить работу каналу, установив бит **DMA_CCRx.ENABLE.**

Организацию взаимодействия КПДП с ресурсами МК иллюстрирует рисунок 1.11.

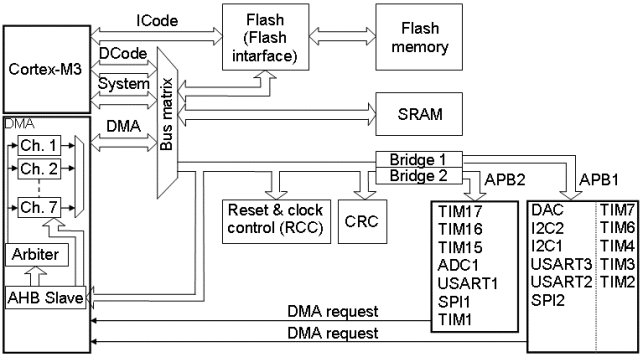



Рис. 1.11 – Взаимодействие контроллера ПДП с ресурсами микроконтроллера

После инициализации контроллер находится в состоянии ожидания момента наступления запрограммированного события.

Периферийное устройство, которому требуется ПДП, посылает запрос **DMA request** на вход арбитра. КПДП обслуживает запрос с

учетом его приоритета. Если запрос принят к обслуживанию, то КПДП формирует сигнал подтверждения периферийному устройству, которое снимает сигнал **DMA request**. После этого КПДП снимает сигнал подтверждения и переходит к процедуре ввода-вывода данных периферийного устройства, не привлекая ресурсы процессорного блока, то есть **становится ведущим устройством.** Состояние процедуры обмена отражается в битах регистра **DMA_ISRx** и соответствующим образом обрабатывается.

В процессе обмена КПДП может приостановить доступ процессорного блока к памяти или периферийному устройству, если они обращаются по одному адресу. Однако это время составляет несколько тактов, так как матрица шин реализует циклическое распределение приоритетов между процессором и КПДП.

**Режим «память – память»** может использоваться для копирования одной области памяти в другую. Для работы в этом режиме необходимо в регистры адресов ***DMA_CPARx*** и ***DMA_CMARx*** занести адреса массивов, над которыми необходимо выполнить операцию копирования, и установить бит ***MEM2MEM*** в регистре ***DMA_CCRx***. Направление передачи выбирается битом ***DMA_CCRx.DIR***. Если, например, ***DIR = 0***, то содержимое памяти с адреса ***DMA_CPARx*** пересылается в область с адреса ***DMA_CMARx***. Обмен останавливается, если значение в регистре размера массива **DMA_CNDTRx** достигает нуля. Остальные настройки общие для двух режимов. **Исключением является режим цикличности,** который нельзя использовать в режиме «память-память».

В передачах «память-память» каждый из каналов ПДП будет занимать шину данных только во время фазы доступа к шине. На передачу каждого слова данных будет затрачиваться 5 циклов. Из них один цикл – для чтения и еще один – для записи, причем данные

циклы чередуются холостыми циклами, во время которых шина освобождается для процессора. Это означает, что КПДП будет использовать не более 40% от пропускной способности шины данных даже во время непрерывной передачи данных на максимальной скорости [53].

Алгоритм работы при использовании прямого доступа к памяти состоит из процедуры инициализации КПДП (определяется режимом работы), программного включения ПДП в периферийных устройствах и памяти, анализа окончания приема/передачи данных в программном режиме или по прерыванию.

Программная модель КПДП приведена в приложении П1.3.

## 1.5 Блок синхронизации

Организацию взаимодействия во времени различных устройств МК обеспечивает блок синхронизации (БС) и делители частоты, расположенные в периферийных устройствах (последовательных и параллельных портах, таймерах АЦП и т.д.).

Основными характеристиками БС являются: количество и типы генераторов, значение их выходной частоты, точность установки частоты в требуемых условиях эксплуатации, возможность переключения генераторов в процессе работы ОМК («на лету»), время выхода на номинальную частоту при включении и перезапуске, возможность калибровки, деления и/или умножения выходной частоты генераторов, наличие автономных источников питания, возможность обнаружения отказа генераторов [50]. Структурная схема блока синхронизации STM32F103C8T6 приведена на рисунке 1.12.

Источниками сигналов синхронизации могут быть:

- **HSI** (High Speed Internal) – внутренний высокочастотный RC-генератор, работающий на частоте 8 МГц. Он калибруется при изготовлении, но не отличается высокой стабильностью.

Гарантируется стабильность частоты в 1% при температуре ядра 25 °C. Частота RC генератора зависит от температуры и может отклоняться от заданной в диапазоне (–1,9 до +1,3) % при изменении температуры МК от 0 до 70 °C. **Этот генератор включается по умолчанию при подаче питания МК.**

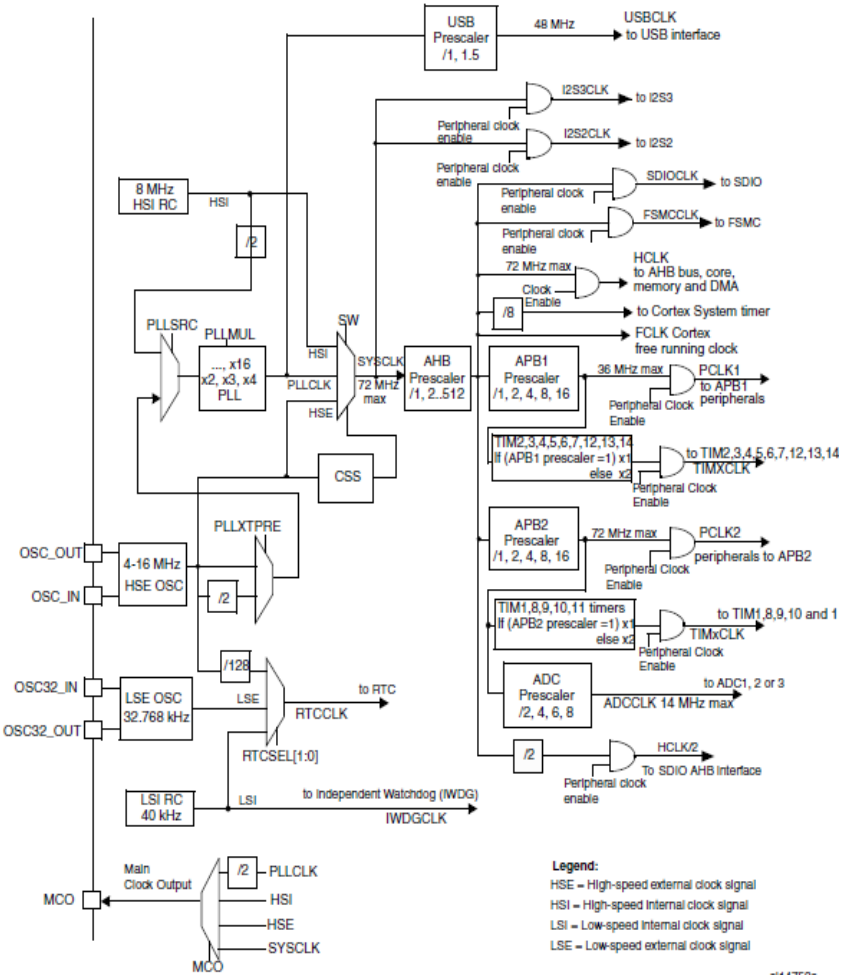



Рис. 1.12 – Структурная схема блока синхронизации

- **HSE** (High Speed External) – высокочастотный генератор с **внешним кварцевым резонатором** с частотой (4÷16) MHz или **внешний источник тактовых импульсов** со скважностью 0,5. Внешний генератор подключается к линии **OSC_IN.**
- **LSI** (сокр. Low Speed Internal) – внутренний RC-генератор на 40 кГц. Он не отличается высокой стабильностью и используется в основном для тактирования сторожевого таймера.
- **LSE** (сокр. Low Speed External) – генератор для часов реального времени с внешним кварцевым резонатором 32,768 кГц (так называемый часовой кварц).

Основным источником синхроимпульсов для МК является сигнал SYSCLK, формируемый на выходе мультиплексора. На его входы поступают сигналы HSI, HSE и PLLMUL. Блок **PLLMUL** (фазовая автоподстройка частоты) выполняет умножение входной частоты на коэффициент от 2 до 16. Частота, поступающая из HSI, предварительно делится на 2.

Для обеспечения шин и периферийных устройств синхросигналами, соответствующих техническим требованиям решаемых задач, между SYSCLK и различными приемниками находятся программно-управляемые делители/умножители частоты. Это обеспечивает независимую настройку частоты синхронизации ядра и периферийных устройств, позволяет снизить энергопотребление (см. ниже).

**Для синхронизации с внешними устройствами выходы генераторов PCLK/2, SYSCLK, HSE, HSI могут быть выведены на вывод MCO.**

Управление блоком синхронизации выполняется обширной группой регистров **RCC (**регистры управления **RCC_CR,** конфигурации **RCC_CFGR,** масок прерываний **RCC_CIR,** регистры разрешения тактирования) [17].

Для контроля корректности работы **HSE (**которая может быть нарушена из-за сбоя питания, наводок, обрыва внешних проводников и так далее) используется блок аварийного переключения **CSS.** Он аппаратно переключает тактирование на встроенный генератор **HSI** и вызывает немаскируемое прерывание **NMI**.

Особенности блока синхронизации:

- При включении питания и сбросе тактирование запрещено в целях снижения энергопотребления. Активизировать синхронизацию следует **только у используемых в данный момент устройств.**
- Необходимо выбирать **минимальную частоту** синхронизации в соответствии с техническим заданием. Зависимость энергопотребления блоков МК от частоты приведены в таблицах 1.7 и 1.8.

Таблица 1.7. Максимальное энергопотребление в рабочем режиме

| Символ | Параметр | Условие | f HCLK | Max | | Ед. измерения |
|---|---|---|---|---|---|---|
| | | | | $T_A$ = 85°C | $T_A$ = 105 °C | |
| $I_{DD}$ | Ток питания в рабочем режиме | Внешняя частота 8 МГц, все периферийные устройства включены | 72 MHz | 50 | 50.3 | mA |
| | | | 48 MHz | 36.1 | 36.2 | |
| | | | 36 MHz | 28.6 | 28.7 | |
| | | | 24 MHz | 19.9 | 20.1 | |
| | | | 16 MHz | 14.7 | 14.9 | |
| | | | 8 MHz | 8.6 | 8.9 | |
| | | Внешняя частота 8 МГц, все периферийные устройства выключены | 72 MHz | 32.8 | 32.9 | |
| | | | 48 MHz | 24.4 | 24.5 | |
| | | | 36 MHz | 19.8 | 19.9 | |
| | | | 24 MHz | 13.9 | 14.2 | |
| | | | 16 MHz | 10.7 | 11 | |
| | | | 8 MHz | 6.8 | 7.1 | |

Таблица 1.8. Энергопотребление периферии

| Периферия | | μA/MHz |
|---|---|---|
| AHB ( до 72 MHz) | DMA1 | 16.53 |
| | BusMatrix | 8.33 |
| APB1 (до 36 MHz) | APB 1-Bridge | 10.28 |
| | TIM2 | 32.50 |
| | TIM3 | 31.39 |
| | TIM4 | 31.94 |
| | SPI2 | 4.17 |
| | USART2 | 12.22 |
| | USART3 | 12.22 |
| | I2C1 | 10.00 |
| | I2C2 | 10.00 |
| | USB | 17.78 |
| | CAN1 | 18.06 |
| | WWDG | 2.50 |
| | PWR | 1.67 |
| | BKP | 2.50 |
| | IWOG | 11.67 |
| APB2 (до 72 MHz) | APB2-Bhdge | 3.75 |
| | GPIOA | 6.67 |
| | GPIOB | 6.53 |
| | GPIOC | 6.53 |
| | GPIOD | 6.53 |
| | GPIOE | 6.39 |
| | SPI1 | 4.72 |
| | USART1 | 11.94 |
| | TIM1 | 23.33 |
| | ADC1 | 17.50 |
| | ADC2 | 16.07 |

- При повышенных требованиях к быстродействию необходимо использовать генератор **HSE,** так как **максимальная частота SYSCLK** при работающем внутреннем генераторе **HSI не может быть выше 64 МГц** (4*16).

- Необходимо контролировать максимальную частоту периферийных шин **APB1** и **APB2,** допустимые значения которых равны соответственно 36 МГц и 72 МГц. Ограничения на частотный диапазон периферийных блоков отражены в соответствующих разделах.
- Частота синхронизации **USB должна быть 48МГц.** Она не зависит от **SYSCLK,** а устанавливается соответствующим множителем блока **PLLMUL** и значением предделителя **USB.**
- Синхронизация часов реального времени **RTC** и сторожевого таймера **WDG** выполняется соответственно генераторами **LSE** и **LSI.**

При программировании рекомендуется использовать генератор **кода CUBE MX.** В нем настройка синхронизации выполняется на вкладке Clock Configuration (рисунок 1.13).

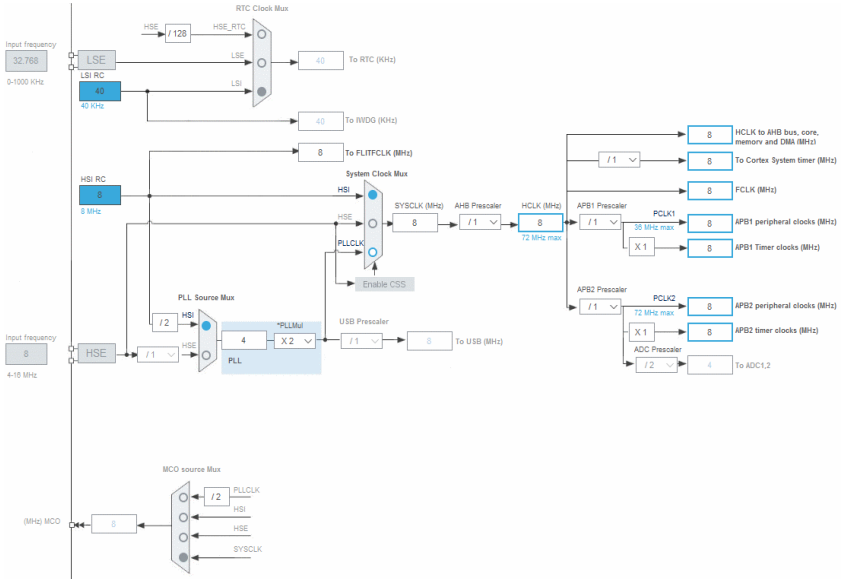
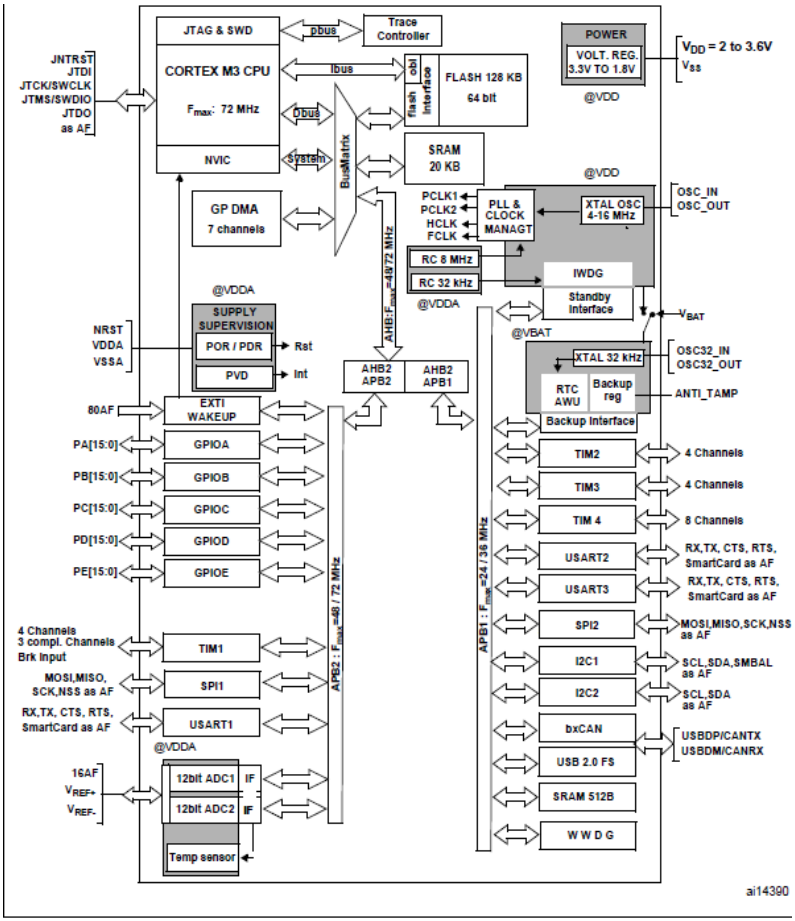



Рис. 1.13 – Окно настройки синхронизации Cube MX

## 1.6 Периферийные устройства Cortex M3

Структурная организация микроконтроллера представлена на рисунке 1.14 [17]. По сравнению со структурой, приведенной на рисунке 1.2, эта структура отражает возможный состав периферийных устройств МК **STM32F103xx.**




1. $T_A$ = –40 °C to +105 °C (junction temperature up to 125 °C).
2. AF = alternate function on I/O port pin.

Рис. 1.14 – Структура микроконтроллеров STM32F103xx

Особенности периферийных устройств (ПУ) микроконтроллера Cortex M3 по сравнению с 8-разрядными:

- расширенные функциональные возможности, значительная часть типовых процедур которых реализована аппаратно;
- использование многоуровневой быстродействующей подсистемы прерываний и каналов прямого доступа к памяти;
- подключение к различным шинам многоуровневого внутреннего интерфейса в зависимости от назначения и технических характеристик;
- более гибкая система синхронизации, позволяющая устанавливать независимо как частоту работы ПУ, так и частоту шины, к которой они присоединены;
- возможность коммутации входов/выходов периферийных устройств на альтернативные линии портов общего назначения (remapping);
- большая разрядность периферийных устройств;
- **высокая степень совмещенности альтернативных функций** линий ввода-вывода (до 4), что затрудняет одновременное использование некоторых периферийных устройств (**структурная несовместимость**). Например, к PA2 (вторая линии порта A) подключены USART2_TX (выход второго приемопередатчика), ADC_IN3 (третий канал АЦП), TIM2_CH4 (четвертый канал захвата/сравнения второго таймера).

### 1.6.1 Порты ввода-вывода общего назначения GPIO

В состав МК STM32F103xx может входить до 5 шестнадцатиразрядных портов. Они могут работать в режиме традиционного порта или в альтернативном режиме, при котором линии используются для ввода/вывода сигналов внутренних периферийных устройств.

МК STM32F103C8T6 (рисунок 1.15) содержит два 16-разрядных порта (PA, PB), 3 линии порта C (PC13÷PC15) с пониженной нагрузочной способностью и быстродействием (ток – до 3 мА, частота – до 2 МГц), две линии порта D (PD0, PD1).

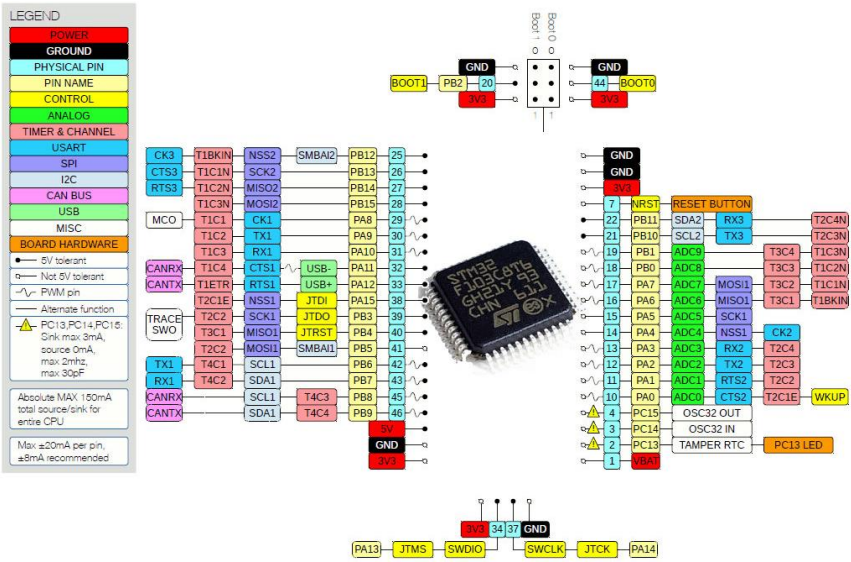



Рис. 1.15 – Назначение внешних линий микроконтроллера STM32F103C8T6

Особенности портов ввода-вывода:

1. Программно-управляемая структура, обеспечивающая согласование разных внешних устройств как по виду входного-выходного сигнала, так и по быстродействию (таблица 1.9).

Выходные линии могут быть настроены на режимы:

- **Двухтактного выхода (Push – pull)** Это основной режим, который используется для управления внешней нагрузкой (исполнительные устройства, реле, индикаторы и так далее).

- **Выхода с открытым стоком (Open – drain)**. Режим используется для согласования со схемами с открытым коллектором (стоком), при реализации шин $I^2C$, SMbus, TWI, 1-

Wire, в качестве усилителя мощности для управления различными устройствами (динамики, реле и так далее), обеспечения совместимости с нагрузкой, напряжение которой отличается от напряжения питания МК (например, 3,3 В и 5 В). Нагрузка включается между источником внешнего напряжения и выводом порта.

- **Альтернативного выхода**, при котором на линии выводятся сигналы внутренних периферийных устройств.

Таблица 1.9. Конфигурация линий ввода/вывода параллельного порта

| Configuration mode | | CNF1 | CNF0 | MODE1 | MODE0 | PxODR register |
|---|---|---|---|---|---|---|
| Выход общего назначения | Push-pull | 0 | 0 | 1 | | 0 или 1 |
| | Open-drain | | 1 | 10 | | 0 или 1 |
| Альтерна-тивные функции выхода | Push-pull | 1 | 0 | 11 | | любое |
| | Open-drain | | 1 | См. **Output MODE bits** | | любое |
| Вход | Analog | 0 | 0 | 0 | | любое |
| | Input floating | | 1 | | | любое |
| | Input pull-down | 1 | 0 | | | 0 |
| | Input pull-up | | | | | 1 |
| … | | | | | | |

| **Output MODE bits** | |
|---|---|
| MODE[1:0] | Функция |
| 0 | Зарезервировано |
| 1 | Макс. выходная частота 10 MHz |
| 10 | Макс. выходная частота 2 MHz |
| 11 | Макс. выходная частота 50 MHz |

Входные линии могут использоваться в режимах:

- **Традиционного цифрового входа** для приема сигналов внешних источников.

- **С подтягивающими резисторами** к питанию (**Input pull-up)** или к земле (**Input pull-down).** Подтягивающие резисторы используются как средство защиты от помех, если входы не подключены к источникам сигналов, уменьшения количества внешних резисторов (например, при подключении внешних кнопок или тумблеров), для согласования с внешними источниками сигналов. Номинал подтягивающих резисторов составляет (30÷50) кОм, что приводит к увеличению длительности фронта и среза входного сигнала. Поэтому необходимо контролировать их номиналы и величину входной емкости.
- **Аналогового входа** (**Analog)** при работе с АЦП, компараторами, усилителями.
- **Альтернативном**, в котором линии являются входами внутренних периферийных устройств.
- **Высокоимпедансного входа** (**Input floating).** В этом режиме (Hi-Z, плавающий) ток через вывод МК близок к нулю.

**Настройка линий ввода-вывода индивидуальная**.

Конфигурация линий портов производится в регистрах **GPIOx_CRL** (*Port configuration register low*) и **GPIOx_CRH** (*Port configuration register high*), где x –индекс порта. Младший регистр конфигурирует выводы порта с 0 по 7, а старший – с 8 по 15.

В этих регистрах для конфигурации линии используется по 4 бита: тип входа/выхода определяет поле **CNFn**, а направление передачи задается полем **MODEn** (00 – ввод, 01, 10, 11 – вывод с требуемой скоростью), где **n** – номер линии. Скорость ввода/вывода завышать не рекомендуется, так как это приводит к увеличению энергопотребления.

Конфигурирование в режиме альтернативных функций выполняется в соответствии с таблицами, приведенными в [17].

В таблице 1.10 в качестве примера приведена настройка линий ввода-вывода для USART:

Таблица 1.10. Конфигурирование линий для USART

| Линии USART | Режим | Конфигурация GPIO |
|---|---|---|
| USARTx TX | Дуплексный асинхронный режим | Альтернативная функция push-pull |
| | Полудуплексный синхронный режим | Альтернативная функция push-pull |
| USARTx RX | Дуплексный асинхронный режим | Плавающий вход / Вход pull-up |
| | Полудуплексный синхронный режим | Не используется. Может использоваться как обычная линия ввода-вывода |
| USARTx_CK | Синхронный режим | Альтернативная функция push-pull |
| USARTx_RTS | Аппаратное управление потоком | Альтернативная функция push-pull |
| USARTx_CTS | Аппаратное управление потоком | Плавающий вход / Вход pull-up |

1. Возможность блокировки выбранной конфигурации порта для защиты от несанкционированных изменений. Механизм блокировки повышает надежность работы как МК, так и модуля, в котором он установлен.

2. Бóльшая скорость сброса/установки бит порта по сравнению с традиционным способом («чтение-модификация-запись») за счет использования дополнительных аппаратных средств.

3. Механизм программной коммутации линий ввода/вывода к определенным внутренним периферийным устройствам (remapping). Это позволяет упростить печатную плату устройства, в котором используется МК, и устранить конфликты при использовании периферийных устройств, сигналы которых используют одни и те же линии порта. Коммутация выполняется в соответствии с таблицами, приведенными в [17]. Подобный механизм (программируемая коммутационная матрица) используется в 8-разрядных МК, системах на кристаллах SoC [50].

4. Возможность работы МК с сигналами 5 В при питании 3,3 В. Этим свойством обладают не все линии ввода/вывода, а только

**толерантные. При подключении внешней нагрузки необходимо контролировать их наличие.**

Схема одного разряда порта представлена на рисунке 1.16.

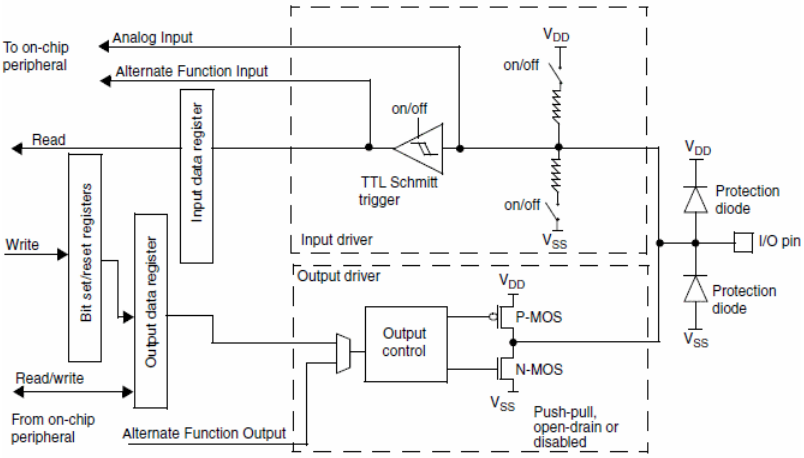



Рис. 1.16 – Схема разряда параллельного порта

При включении питания или после сброса входные линии устанавливаются в высокоимпедансное состояние.

**В режиме входа** выходной драйвер (**Output driver**) отключен, а структура входного драйвера (**Input driver**) определяется содержимым регистра конфигурации. При цифровом вводе сигнал поступает на вход триггера Шмитта, который является временным нормализатором входного сигнала – импульсы с различными фронтом и срезом на выходе триггера имеют унифицированный вид. С выхода триггера Шмитта данные поступают во **входной регистр IDR** или регистры периферийных устройств (при настройке на альтернативные функции).

В **режиме выхода** включается выходной драйвер (**Output driver**) на основе комплементарной пары транзисторов. В режиме **Push-pull** работают поочередно оба транзистора, а в **Open-drain –** только нижний. **Выходной регистр ODR** может работать в режиме записи и

чтения. В режиме выхода в него записываются выходные данные, а в режиме входа – фиксируется побитное состояние подключения подтягивающего резистора (к линии питания или земле).

При работе в альтернативном режиме на выход поступают данные из соответствующего периферийного устройства.

Значения выходных сигналов порта можно прочитать в регистре **IDR**, так как остается включенным триггер Шмитта.

Регистры сброса/установки бит (Bit set/reset registers) позволяют ускорить работу с битовыми данными по сравнению с традиционным способом «чтение-модификация-запись».

К входным линиям портов подключены 2 защитных диода, один из которых защищает от сигналов отрицательной полярности (нижний), а второй – ограничивает положительное напряжение на уровне напряжения питания входное Vdd. Для данного контроллера – это 3.3 В, а **входной ток не должен превышать ± 5 мА. Сигналы с напряжением, превышающим эти пределы, должны подключаться через ограничительные резисторы**. Некоторая часть входов МК допускает подключение напряжения 5 В – **толерантные входы FT (five volt tolerant**). Для таких выводов верхний защитный диод подключен не к питанию Vdd, а к ограничителю напряжения Vdd_ft (рисунок 1.17). Напряжение ограничивается до уровня (Vdd + 4 В) [78].

Порты GPIO работают в синхронном режиме. Для распознавания уровня нуля и единицы необходимо несколько тактов генератора. Поэтому для обеспечения функционирования порта необходимо контролировать частоту тактирования и параметры внешних сигналов. Порты тактируются с частотой периферийной шины APB2.

При работе в альтернативном режиме тактирование периферийных устройств в зависимости от требуемого быстродействия выполняется как от APB2, так и APB1.

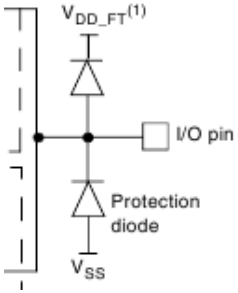



Рис. 1.17 – Схема толерантного входа

По умолчанию **тактирование APB2, APB1 выключено** с целью снижения энергопотребления. Поэтому необходимо разрешить тактирование периферийных устройств в соответствующих регистрах **RCC_APB2ENR** или **RCC_APB1ENR,** где каждому устройству соответствует свой бит разрешения тактирования, включая и параллельные порты IOPA-IOPG. Бит AFIO предназначен для включения тактирования при альтернативном использовании портов.

**Основные электрические параметры портов:**

**входные**

– напряжение низкого уровня – не более 0,35 VDD (при питании 3,3 В не более 1,16 В);

– напряжение высокого уровня – не менее 0,65 VDD (при питании 3,3 В не менее 2,15 В);

– ток утечки – ± 1 мкА;

– сопротивление подтягивающих резисторов – 30÷50 кОм, типовое значение 40 кОм;

– емкость – 5 пкФ.

Предельно-допустимые входные параметры портов:

– напряжение на толерантных входах – не менее – 0,3 В, не более Vdd + 4 В (при питании 3,3 В – 0,3 … 7,3 В);

– напряжение на остальных входах – не менее – 0,3 В, не более 4 В;

– максимальный втекающий ток через защитные диоды – ± 5 мА.

**выходные**

– напряжение низкого уровня при втекающем токе 8 мА – не более 0,4 В;

– напряжение низкого уровня при втекающем токе 20 мА – не более 1,3 В;

– напряжение высокого уровня при вытекающем токе 8 мА – не менее Vdd – 0,4 В;

– напряжение высокого уровня при вытекающем токе 20 мА – не менее Vdd – 1,3 В.

Предельно-допустимые выходные параметры портов:

– максимальный вытекающий ток вывода – 25 мА;

– максимальный втекающий ток вывода – 25 мА;

– общий вытекающий или втекающий ток портов – не более 150 мА.

Однако не рекомендуется подключать нагрузку до 25 мА, так как при этом уровень логического сигнала может выйти за допустимое значение.

Рекомендуется подключать не более 20 выводов с нагрузкой 8 мА или не более 8 выводов с 20 мА. Эти значения могут отличаться для различных типов корпусов МК [78].

Программная модель параллельного порта приведена в приложении П1.4

**Алгоритм настройки портов:**

1. Выбрать порт ввода/вывода данных для выполнения технического задания.

Это далеко не тривиальная задача, учитывая высокую степень совмещенности функций линий порта. В некоторых случаях облегчить этот выбор может помочь переназначение контактов (**ремапинг**). Ремапингом управляет группа регистров **AFIO** [17].

Если не используются альтернативные функции или их немного, то при выборе портов проблем не возникает. При выборе линий порта необходимо учитывать особенности внешних устройств: уровень входного сигнала, требуемый выходной ток, емкостную нагрузку и так далее.

2. Задать конфигурацию в регистрах **GPIOx_CRL, GPIOx_CRH** в соответствии с особенностями решаемой задачи.

3. В случае необходимости установить блокировку конфигурации порта или его отдельных линий в регистре **GPIOx_LCKR** (Port configuration lock register).

Каждому выводу порта соответствует бит блокировки **LCK0÷LCK15**. Установка бита в «1» запрещает изменение соответствующих битов в регистре конфигурации. После установки всех нужных битов регистра блокировки необходимо активировать защиту. Для этого в бит LCKK следует последовательно записать «1, 0, 1». **Изменение** защищенных битов конфигурации и режима возможно только **после сброса микроконтроллера**.

Проверить значение LCKK можно чтением этого бита. При активированной защите два подряд чтения бита должны дать результат 0, 1.

Запись/чтение порта выполняется в регистрах ввода и вывода данных. Регистр ввода данных порта **GPIOx_IDR** (Port input data register) **доступен только для чтения.** Регистр вывода данных порта **GPIOx_ODR (**Port output data register) **доступен как для записи, так и чтения.** Запись данных устанавливает состояние выводов порта **(ODR0÷ODR15),** а чтение разрешено в режиме ввода. При чтении **ODR** можно проверить место подключения подтягивающего резистора.

Для управления состоянием отдельных выводов можно воспользоваться специальными регистрами. Регистр установки/сброса битов **GPIOx_BSRR** (Port bit set/reset register)

позволяет установить соответствующие биты **(BS0÷BS15)** или сбросить **(BR0÷BR15)**. Регистр сброса битов **GPIOx_BRR** (Port bit reset register) работает аналогично разрядам **BR0÷BR15** предыдущего регистра.

4. Разрешить тактирование в регистре **RCC_APB2ENR**.

5. Выполнить обработку данных.

### 1.6.2 Блок временных событий

Функционирование любой вычислительной системы невозможно без специальных средств, обеспечивающих анализ внутренних и внешних временных событий, формирования управляющих временных сигналов различного вида.

В МК эти функции реализует блок временных событий (**БВС**), который должен решать следующие задачи:

- измерение временных интервалов, частоты, периода, скважности, фазового сдвига, числа импульсов и так далее;
- фиксацию момента появления тех или иных событий (фронта, среза, заданного количества импульсов и так далее);
- формирование интервалов времени для запуска внутренних и внешних событий;
- формирование последовательности импульсов заданной частоты и скважности;
- реализацию различных модуляторов: широтно-импульсных (ШИМ), фазо-частотных (ФЧМ) и так далее;
- контроль времени работы программы;
- выполнение функций часов реального времени [70].

В состав БВС STM32 входят: счетчики-таймеры, системный таймер, часы реального времени и сторожевой таймер. Если системный таймер является стандартным блоком архитектуры Cortex M3, то количество и типы других устройств зависят от типа МК.

**Несомненным достоинством БВС STM32 является стандартизация (унификация) программной модели и структуры блока таймеров, что обеспечивает переносимость программ между разными микроконтроллерами класса STM32 с незначительными доработками или без них, расширенная функциональность, реализуемая с широким использованием аппаратных средств.**

### 1.6.2.1 Счетчики-таймеры

В документации по МК STM32 технические средства для работы с частотно-временными сигналами определяются как таймеры и обозначаются **TIMx**. В отечественной технической литературе наиболее часто используется термин **счетчик-таймер (СТ)**. В **режиме таймера** сигналом синхронизации является **внутренний генератор МК**, в **режиме счетчика** – **внешний источник**. Поэтому будем пользоваться термином: счетчик-таймер.

**Количество счетчиков-таймеров СТ в STM32 определяется моделью микроконтроллера, а номер СТ определяет его принадлежность к тому или иному типу и функциональность.** В разных МК счетчики-таймеры с одинаковыми номерами обычно структурно и программно-совместимы. СТ в МК нумеруются не подряд, а выборочно в зависимости от типа микроконтроллера. Наличие СТ с 17 номером (TIM17) не означает, что имеются все 17 СТ (TIM1÷TIM17), так как СТ с промежуточными номерами могут отсутствовать.

**СТ делятся на три группы: базовые (basic timers), общего назначения (general-purpose timers) и с расширенным управлением (advanced-control timers). Функциональность СТ возрастает от базовых к расширенным, включая в себя функции предыдущих групп. Особенностью последних двух групп является минимальное участие процессорного блока в работе СТ.**

В качестве примера в таблице 1.11 приведён перечень счетчиков-таймеров, которые **могут быть** включены в МК типа STM32F101xx. – STM32F107xx.

Таблица 1.11. Состав таймеров в МК STM32F101xx. – STM32F107xx

| **Имя СТ** | **Тип СТ** |
|---|---|
| TIM1, TIM8 | Расширенное управление |
| TIM2, TIM3, TIM4, TIM5 | Общего назначения |
| TIM9, TIM10, TIM11, TIM12, TIM13, TIM14 | Общего назначения (сокращенные функции) |
| TIM6, TIM7 | Базовый |

В микроконтроллере STM32F103C8T6, установленном в отладочном модуле, используются TIM1, TIM2, TIM3, TIM4. **TIM1 подключен к скоростной периферийной магистрали APB2, а остальные – к APB1.** Технологию работы с СТ рассмотрим **на примере таймера общего назначения**.

### Основные характеристики СТ TIM2-TIM4

1. Базовым элементом СТ является счетчик **CNT (counter):**

**- Разрядность** – 16 бит (регистр **TIMx_CNT**). В других семействах STM32 может быть 32 бита.

**- Максимальная частота** (быстродействие). В режиме таймера определяется частотой внутренних источников, в режиме счетчика – внешних. Длительность уровней лог. «1» и «0» внешней частоты должна быть больше двух периодов CK_INT. **Частота CK_INT будет формироваться, если установлен бит разрешения в регистре RCC_APBnENR,** где **n** – номер периферийной шины.

**Источники синхроимпульсов:**

- генератор синхронизации соответствующей периферийной шины **APB** (внутренний сигнал **TIMx_CLK**);
- сигнал, возникающий при переполнении другого таймера (внутренние **ITR0÷ITR3)**;

- сигнал внешнего источника, поступающий на вход **ETR**;
- внешние сигналы входов захвата **TI1, TI2, TI3, TI4**;
- внешние сигналы энкодера.

**Источник синхронизации задается в регистрах TIMx_SMCR, TIMx_CCMR1, TIMx_CCER.**

**- Направление счёта** – в зависимости от настройки может быть суммирующим, вычитающим или реверсивным. В суммирующем счетчике содержимое увеличивается на 1 с приходом очередного импульса, в вычитающем – уменьшается на 1. **Особенностью реверсивного режим STM32** является сначала увеличение счета на 1 до достижения определенного значения, хранящегося в регистре перезагрузки **ARR,** а затем – уменьшение на 1 до нулевого значения. **Направление счета задается в регистре управления TIMx_CR1 (разряды CMS и DIR).**

**- Способ запуска** – программный или аппаратный. Аппаратный запуск используется для синхронизации работы МК с внешними устройствами. В этом случае начало работы счетчика задается внешним сигналом.

**- Наличие программируемого делителя частоты PSC** (15÷0) с коэффициентом деления до 65535 (предделитель). В отличие от 8-разрядных МК, у которых предделитель имеет фиксированные коэффициенты деления кратные степени 2, в **PSC можно устанавливать любой коэффициент из указанного диапазона.** Тактовый сигнал **CK_PSC** на вход **PSC** поступает со схемы управления. Входная частота счетчика **CNT** определяется следующим образом $F_{cnt} = F_{in}/(PSC + 1)$, где $\mathbf{F_{cnt}}$ – входная частота счетчика, $\mathbf{F_{in}}$ – входная частота предделителя, **PSC** – содержимое регистра, определяющее коэффициент деления.

Счетчик предделителя считает входные импульсы от 0 до значения, записанного в PSC. При равенстве кода счетчика и PSC счетчик сбрасывается и цикл повторяется. **Значение**

**коэффициента деления записывается в регистр TIMx_PSC**. Регистр предделителя имеет буфер. Поэтому **значение предделителя можно изменять динамически в процессе работы счетчика**. После записи нового значения в буферный регистр изменение коэффициента деления произойдет в момент переполнения счетчика. Наличие предделителя фактически увеличивает разрядность счетчика.

**Модуль пересчета счётчика** определяется содержимым регистра автоперезагрузки **ARR (Auto-Reload Register**). У **суммирующего** счетчика **в момент превышения значения**, записанного в **ARR, формируется сигнал (событие) переполнения**. **Вычитающий** счетчик **формирует этот сигнал**, когда его значение **станет равным нулю. Реверсивный** счетчик изменяет свое значение **от нуля до ARR**, а затем в обратном направлении **до нуля**. Этот режим используется для формирования ШИМ-сигнала с выравниванием по центру. У реверсивного счетчика сигнал переполнения формируется в зависимости от значения битов **CMS0, CMS1** регистра **TIMx_CR1** (при достижении 0, значения регистра перегрузки или в обоих случаях).

При переполнении происходит сброс счетчика. (в суммирующем счетчике – записывается ноль, в вычитающем – значение регистра ARR) и предварительного делителя, а далее работа продолжается.

Управление перезагрузкой задается битом **ARPE регистра TIMx_CR1.** Регистр перезагрузки имеет буферный регистр. В зависимости от значения **ARPE** регистр ARR обновляется сразу после записи в него нового значения или из буферного регистра после переполнения счетчика.

Возможен **однократный режим** счетчика, при котором после переполнения счет прекращается и происходит сброс разряда

разрешения счета **CEN** регистра **TIMx_CR1**. Однократный режим устанавливается битом **OPM TIMx_CR1**.

2. **Четыре канала захвата/сравнения**, каждый из которых работает в режиме с разделением времени, так как у них общие регистр захвата/сравнения и линии ввода/вывода.

В **режиме захвата** текущее значение счетчика по внешнему или внутреннему сигналу сохраняется в буферном регистре захвата. Частными случаями режима захвата являются **режимы измерения параметров входного широтно-импульсного сигнала ШИМ и энкодера.**

**Режим сравнения** используется для формирования выходных сигналов различного вида: генераторы программируемой частоты и скважности, импульсы заданной длительности и полярности и так далее. Частными случаями режима сравнения являются **режимы ШИМ и одновибратора (одиночного импульса).**

3. **Возможность синхронизированной работы нескольких СТ по принципу «ведущий-ведомый». Подобное каскадное соединение** возможно только с помощью **внутренней** коммутации. У каждого СТ для этих целей формируется выходной триггерный сигнал TRGO и имеется 4 входных сигнала ITR0, ITR1, ITR2, ITR3. Сигнал TRGO **ведущего СТ** может быть подключен к одному или нескольким входам **ведомых СТ**. Синхронизированная работа нескольких СТ используется, например, для увеличения разрядности счетчика за счет каскадного соединения, измерения частоты, временного интервала и так далее.

4. **Способы ввода/вывода** – программный, по прерыванию, в режиме ПДП. Управление способом ввода/вывода выполняется регистром разрешения прерываний и ПДП **TIMx_DIER.**

5. Возможность перекоммутации каналов таймеров на альтернативные линии GPIO (ремапинг, таблица 1.12) [17].

Таблица 1.12. Ремапинг каналов CT

| **TIM1** | | | |
|---|---|---|---|
| Альтернативные функции | TIM1_REMAP[1:0] = “00” (нет ремапинга) | TIM1_REMAP[1:0] = = “01” (частичный ремапинг) | TIM1_REMAP[1:0] = = “11” (полный ремапинг) (1) |
| TIM1_ETR | PA12 | | PE7 |
| TIM1_CH1 | PA8 | | PE9 |
| TIM1_CH2 | PA9 | | PE11 |
| TIM1_CH3 | PA10 | | PE13 |
| TIM1_CH4 | PA11 | | PE14 |
| TIM1_BKIN | PB12 (2) | PA6 | PE15 |
| TIM1_CH1N | PB13 | PA7 | PE8 |
| TIM1_CH2N | PB14 (2) | PB0 | PE10 |
| TIM1_CH3N | PB15 (2) | PB1 | PE12 |

| **TIM2** | | | | |
|---|---|---|---|---|
| Альтернативные функции | TIM2_REMAP[1: 0] = = “00” (нет ремапинга) | TIM2_REMAP [1: 0] = “01” (частичный ремапинг) | TIM2_REMAP[1: 0] = “10” (частичный ремапинг) (1) | TIM2_REMAP [1: 0] = “11” (полный ремапинг) (1) |
| TIM2_CH1_ETR(2) | PA0 | PA15 | PA0 | PA15 |
| TIM2_CH2 | PA1 | PB3 | PA1 | PB3 |
| TIM2_CH3 | PA2 | | PB10 | |
| TIM2_CH4 | PA3 | | PB11 | |

| **TIM3** | | | |
|---|---|---|---|
| Альтернативные функции | TIM3_REMAP [1:0] = “00” (нет ремапинга) | TIM3_REMAP[1:0] = = “10” (частичный ремапинг) | TIM3_REMAP[1:0] = = “11” (полный ремапинг) (1) |
| TIM3_CH1 | PA6 | PB4 | PC6 |
| TIM3_CH2 | PA7 | PB5 | PC7 |
| TIM3_CH3 | PB0 | | PC8 |
| TIM3_CH4 | PB1 | | PC9 |

| **TIM4** | | |
|---|---|---|
| Альтернативные функции | TIM4_REMAP = 0 | TIM4_REMAP = 1(1) |
| TIM4_CH1 | PB6 | PD12 |
| TIM4_CH2 | PB7 | PD13 |
| TIM4_CH3 | PB8 | PD14 |
| TIM4_CH4 | PB9 | PD15 |

На рисунке 1.18 приведена структура СТ [17].

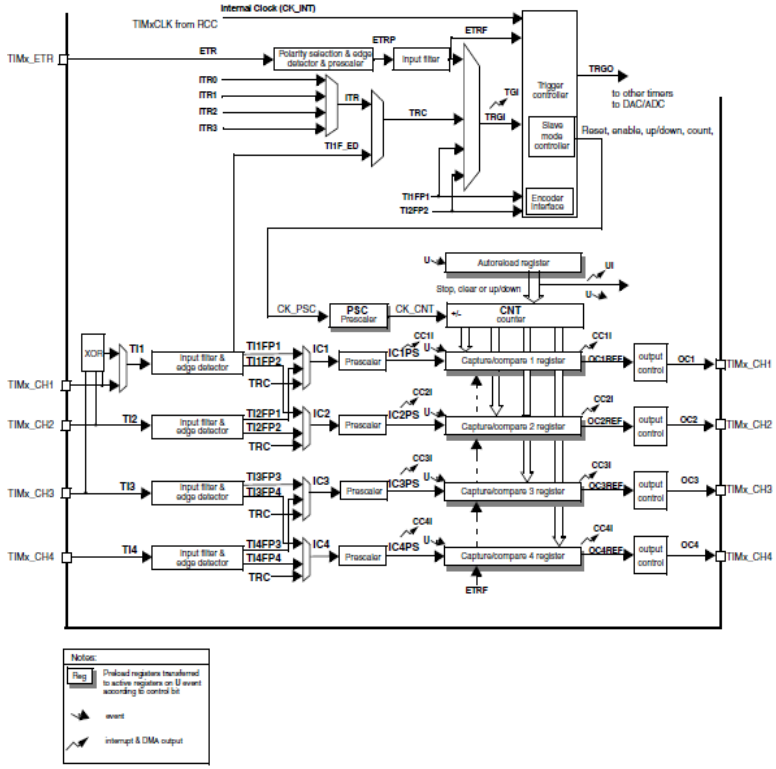



Рис. 1.18 – Структура счетчика-таймера общего назначения

**Особенностями таймера TIM1** с расширенным управлением является наличие:

- комплементарных выходов **TIM1CH1** и **TIM1CH1N** с программируемым мертвым временем (**Dead Time** – пауза между переключениями необходима для исключения сквозных токов при управлении силовыми ключами полу/полно мостовых схем);
- счетчика повторений для обновления регистров таймера только после заданного числа циклов;

- входа останова **BRK** для перевода сигналов таймера в состояние сброса или известное состояние;
- аппаратных средств для работы с датчиком Холла;
- средств защиты параметров таймера от несанкционированного доступа до следующего сброса МК.

### Режим счетчика

При работе в режиме счетчика внешние сигналы могут поступать на **вход ETR** (режим 2 внешней синхронизации) или на один из входов канала таймера **TIx** (режим 1 внешней синхронизации). При выборе этих режимов не следует забывать, что состояние входного импульса **определяется синхронно** с частотой синхронизации периферийного интерфейса, к которому подключен СТ. Поэтому уровень лог. «1» и лог. «0» должны быть не менее двух периодов синхрогенераторов. **Наличие входа ETR необходимо контролировать**. Например, **у TIM3 вход ETR отсутствует.**

Структура блока синхронизации ETR представлена на рисунке 1.19.

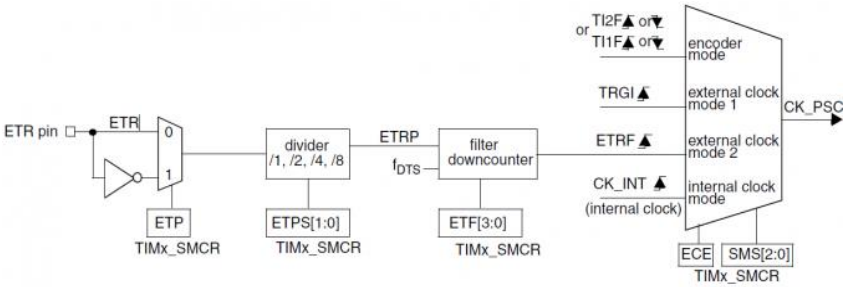



Рис. 1.19 – Структура блока внешней синхронизации ETR

При программировании помимо структуры требуется знать назначение и состав регистров СТ. Программная модель СТ приведена в приложении П1.5 [85, 17].

Особенностью TIM2 в режиме счетчика является объединение входов CH1и ETR. Поэтому эти входы одновременно использовать нельзя (только в режиме с разделением времени).

Основные настройки канала ETR выполняются в регистре **TIMx_SMCR**. Для этого нужно:

- Выбрать активный перепад сигнала **ETP**: (External trigger polarity) 0 – фронт сигнала, 1 – срез сигнала.
- Установить значение предделителя **ETPS[1:0]** (External trigger prescaler) – (1, 2, 4, 8). **Наличие предделителя позволяет выполнять внешнюю синхронизацию с частотой большей, чем CK_INT. Однако необходимо контролировать, чтобы частота на входе счетчика была в 4 раза меньше CK_INT.**
- Настроить фильтр в битовом поле **ETF[3:0]** (**N** – количество событий). **Фильтрация обеспечивает защиту от помех, длительность которых меньше значения, указанного в этом поле.** Параметры фильтрации определяются выбором источника синхронизации (**CK_INT** или **FDTS**), заданием значения частоты и установкой значения минимальной длительности **N**. Рисунок 1.20 иллюстрирует работу фильтра [68].

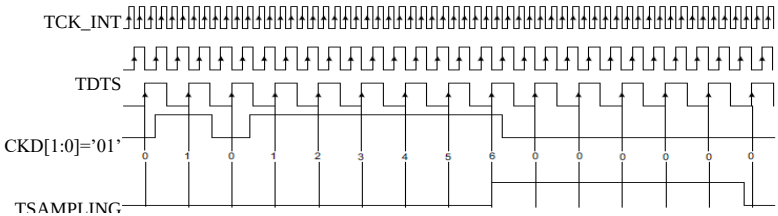



Рис. 1.20 – Фильтрация сигнала ETF

**FDTS** формируется на выходе делителя, на вход которого поступает **CK_INT.** При использовании **FDTS** значение частоты

устанавливается битами **CKD0, CKD1 регистра TIMx_CR1.** При FCK_INT = 1 МГц и CKD [1:0] = 01 частота $F_{dts} = F_{ck\_int}/2$ = 500 кГц. Если задать ETF [3:0] = 0100, то сигнал ETR на входе счетчика, длительность которого меньше T = 6*1/250 кГц = 24мкс не будет вызывать изменения счетчика.

- Задать режим внешней синхронизации от сигнала **ETR (ECE** = 1);
- установить бит разрешения формирования **CK_INT** в регистре **RCC_APBnENR,** где **n** – номер периферийной шины;
- включить таймер, установив **CEN** = 1, в регистре **TIMxCR1.**

**В режиме 1 внешнего тактирования** (external clock mode 1) может использоваться любой сигнал, подключенный к мультиплексору, на выходе которого формируется **TRGI**:

- сигнал **ETRF** с выхода фильтра **ETR**;
- внутренние сигналы синхронизации таймера (входы **ITR0÷ITR3**);
- сигнал **TI1_ED** на выходе первого канала таймера, который формирует импульсы по фронту и спаду входного сигнала;
- выходные сигналы фильтров **TI1FP1** (канал 1) и **TI2FP2** (канал 2).

Для настройки на этот режим необходимо сделать следующее:

- В регистре **TIMx_SMCR** установить «111» в разряды **SMS(2÷0),** а в разряд **ECE** – «0».
- Выбрать один из восьми источников сигнала с помощью разрядов TS0÷TS2 регистра **TIMx_SMCR** (для второго канала – 110).
- Настроить канал на режим ввода – разряды **CCxS(1÷0)** регистра **TIMx CCMR1** (например, 01).

- Задать параметры фильтра аналогично предыдущему с помощью битов **ICnF(3÷0) TIMx_CCMR1**, где **n** – номер канала. Например, ICnF = 0010 установит частоту тактирования (опроса входного сигнала) CK_INT, а состояние сигналов TI1 и TI2 должно быть стабильным в течении не менее 4 тактов сигнала CK_INT. Только в этом случае изменится состояние счетчика под действием сигналов TI1FP1, TI1FP2.

- Выбрать источник запуска (синхронизации) счетчика – биты **TS (2÷0)** регистра **TIMx_SMCR**. Например, 110 для 2 канала (выходной сигнал фильтра TI1FP2).

- Разрешить работу счетчика записью **CEN = 1** в **TIMx_CR1.**

На рисунке 1.21 представлена схема формирования сигнала для канала TI2.

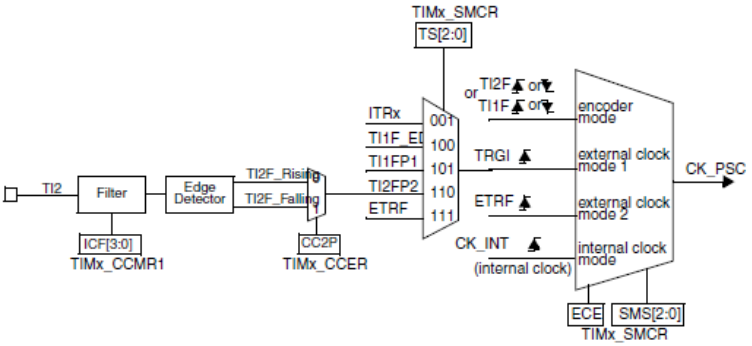



Рис. 1.21 – Формирование сигнала внешней синхронизации канала TI2

При использовании сигнала ETRF методика настройки аналогична вышеизложенной. Особенностью является установка «1» в разряде ECE регистра TIMx_SMCR, а канал ETR настраивается как в режиме 2.

**Отличия режима 1 и режима 2 внешнего тактирования**:

- **Обновление счетчика по обоим перепадам внешнего сигнала можно использовать только в режиме внешнего тактирования 1.** В режиме 2 это невозможно;

- При настройке на режим 2 таймер можно дополнительно конфигурировать на определенные режимы ведомого [69].

**Синхронизация от внешнего сигнала**

При использовании внешних синхронизирующих сигналов ETR, TI1, TI2B возможно задание нескольких дополнительных режимов: сброса, управления счетом, запуска, энкодера [29].

Настройка на требуемый режим выполняется в регистре **TIMx_SMCR** биты **SMS(2÷0).** В **режиме сброса (Reset mode)** счетчик и предварительный делитель СТ могут быть принудительно инициализированы при возникновении события на входе запуска. Кроме того, если сброшен разряд **URS** регистра **TIMxCR1**, может генерироваться событие обновления **UEV**. Тогда все регистры с предварительной загрузкой (ARR, PSC и др.) будут обновлены (рисунок 1.22).

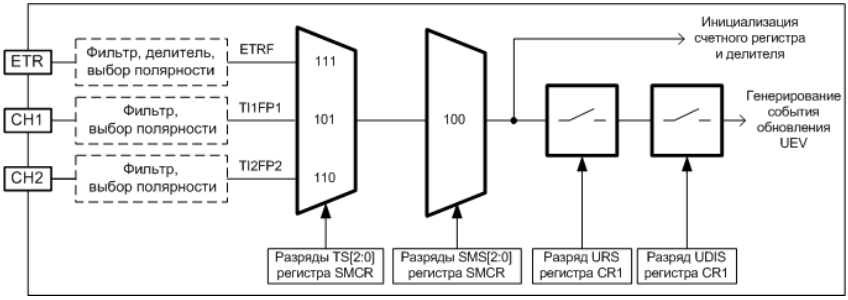



Рис. 1.22 – Управление СТ в режиме **Reset mode**

Источник сигнала задается разрядами **TS(2÷0)** регистра **TIMx_SMCR**. Приведенная ниже диаграмма (рисунок 1.23) иллюстрирует работу таймера в **режиме Reset Mode** (синхронизация от TI1, ARR = 0x36).

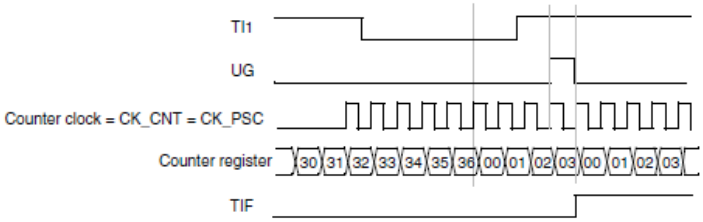



Рис. 1.23 – Временная диаграмма в режиме сброса

По фронту сигнала **TI1** с небольшой задержкой, определяемой внутренней тактовой частотой, устанавливается разряд **UG** регистра генерации событий **TIMx_EGR**, который через один такт **CK_PSC** автоматически сбрасывается. При сбросе **UG** формируется сигнал, который в зависимости от направления счета сбрасывает или записывает содержимое **TIMx_ARR** в счетчик и сбрасывает предварительный делитель. Возможно формирование события обновления **UEV,** если оно разрешено (см. выше). **TIF –** флаг прерывания, устанавливаемый в регистре состояния **TIMxSR.**

В **режиме управления счетом (Gated mode)**, подача тактовых импульсов на вход предварительного делителя счетчика **PSC зависит от уровня сигнала** на выбранном входе (рисунок 1.24).

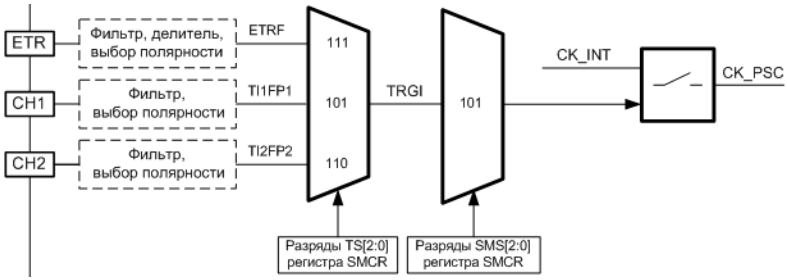



Рис. 1.24 – Управление СТ в режиме **Gated mode**

После его включения, тактовые импульсы **CK_PSC** будут поступать на вход **PSC** только при **TRGI = 1.** На рисунке 1.25 приведена диаграмма режима **Gated mode** для входа TI1 (CC1P = 1):

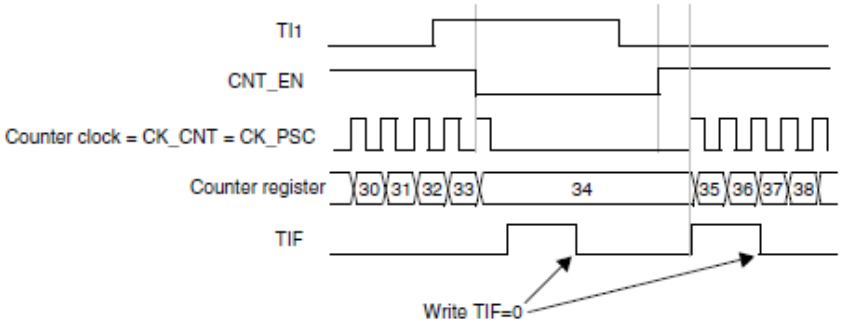



Рис. 1.25 – Временная диаграмма режима **Gated mode**

В приведенном примере разрешение счета происходит, когда TI1 = 0, а не «1». Это возможно потому, что разряд CC1P регистра установки режима работы входного сигнала TIMx_CCER установлен в единицу, поэтому происходит инверсия TI1, и при TI1 = 0 сигнал TRGI (а именно он управляет подачей тактовых импульсов в режиме Gated mode) равен единице.

**При остановке счетчика и начале его работы устанавливается флаг TIF.** В **режиме запуска (Trigger Mode)** счет импульсов начинается **при поступлении фронта** выбранного сигнала. **Содержимое счетчика** при этом **не перезагружается** (рисунок 1.26).

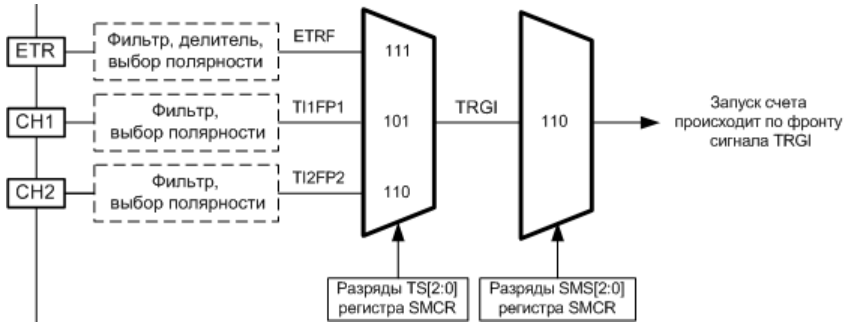



Рис. 1.26 – Управление СТ в режиме **Trigger Mode**

**Режим захвата**

Режим захвата можно использовать для синхронизации работы МК с различными внешними событиями (запуск периферийных устройств по фронту или срезу входного сигнала, измерение длительности импульса, периода входной частоты, фазового сдвига, измерения параметров ШИМ, энкодера и так далее) [50].

В момент захвата формируется флаг, который может использоваться при вводе-выводе в программном режиме, по прерыванию или режиме ПДП (DMA) для дальнейшей обработки содержимого регистра захвата.

Особенности режима захвата:

- Использование общих ресурсов с режимом сравнения (входы/выходы **TIMxCHi,** где i – номер канала, регистр захвата/сравнения **capture/compare register**, модуль фильтрации).
- Возможность настройки параметров входного сигнала – активный фронт или срез (**CCxP** регистра **TIMx_CCER**), фильтрация, программно-управляемый делитель (**prescaler**). Параметры фильтрации определяет содержимое **IC xF[3÷0]**, а коэффициента деления предделителя – **ICxPSC[1÷0]** регистров **TIMxCCMR1** (каналы 1 и 2) и **TIMxCCMR2** (каналы 3 и 4).
- Источником сигнала захвата может быть не только внешний сигнал, но и другой таймер (сигнал TRC на рисунке 1.18).
- Количество каналов – 4.
- Имеется возможность каналы 2 и 3 подключить к каналу 1 (**TI1S регистра TIMx_CR2)**.
- Предусмотрена возможность совместной работы CH1 и CH2 (режим энкодера, ШИМ) и CH2, CH3 (режим ШИМ). Имеется возможность канал 1 подключить к выводу CH2, а канал 2 – к выводу CH1; канал 3 – к выводу CH4, а канал 4 – к выводу CH3 (рисунок 1.18).
- Для каждого канала в регистре состояния **TIMxSR** предусмотрено два флага захвата: **CC1F÷CC4F** устанавливается

при обнаружении захвата, а **CCO1F÷CCO4F**. при повторном захвате (если произошел захват при установленном первом флаге).

На общей структуре (рисунок 1.18) **ICnPS –** сигнал, по которому происходит сохранение текущего значения счетчика в соответствующем регистре захвата **TIMxCCRn**.

Для настройки на этот режим следует произвести следующие действия:

- Включить режим захвата **TIMx_CCMR1 (биты CC1S [1÷0]).** После этого входные линии будут сконфигурированы как вход, а регистр **TIMx_CCR1** будет доступен только для чтения.
- Задать параметры фильтра: разряды **ICxF[3÷0]** и **ICxPSC[1÷0]** регистров **TIMxCCMR1** (каналы 1 и 2) и **TIMxCCMR2** (каналы 3 и 4). Эти параметры определяются особенностями внешних сигналов.
- Выбрать активный перепад входного сигнала (фронт или срез) бит **CCnP в TIMx_CCER**.
- Задать значение предделителя **ICnPSC**[1÷0] регистра **TIMxCCMR1** или **TIMxCCMR2**.
- Разрешить захват установкой бита **CCnE** в **TIMx_CCER**.
- В момент захвата формируются флаги **CCnIF** и **CCnOF** в регистре **TIMxSR**.
- При вводе /выводе по прерыванию и/или в режиме ПДП (DMA) следует настроить соответствующие режимы и разрешить работу в регистре **TIMx_DIER** (прерывания установкой бита **CCnIE,** а DMA установкой бита **CCnDE.** Прерывание и запрос DMA можно генерировать программно установкой соответствующего бита **CCnG** регистра **TIMx_EGR**.

Частными случаями режима захвата являются **режимы измерения параметров входного сигнала ШИМ и энкодера.**

**В режиме захвата ШИМ** используются два канала: один канал настраивается на активный фронт сигнала и сброс, а второй – на срез. В этом случае первый канал будет измерять период ШИМ, а второй – длительность **положительной** полуволны. Отношение этих величин позволяет определить **скважность входного сигнала.** Оба канала подключаются к одному и тому же физическому входу. Один из двух сигналов TIxFPn выбирается как вход запуска и контроллер подчиненного режима конфигурируется в режиме сброса. **Для измерения могут использоваться каналы 1, 2.** На рисунке 1.27 приведена временная диаграмма для каналов 1 и 2.

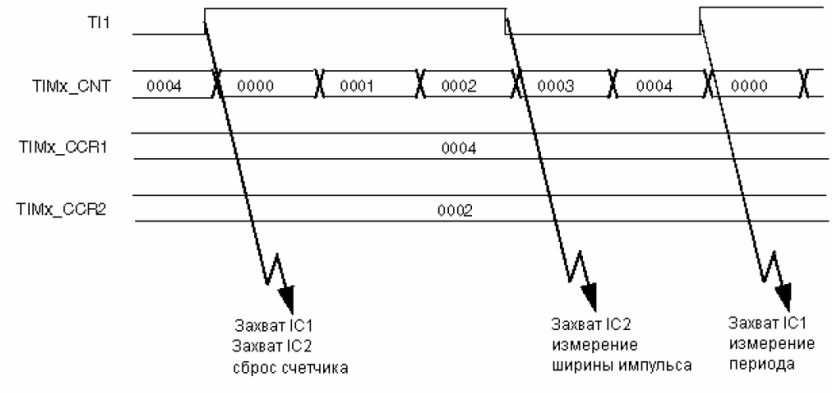



Рис. 1.27 – Временная диаграмма измерения параметров ШИМ

Для приведенного примера период будет зафиксирован в TIMx_CCR1, а длительность импульса – в TIMx_CCR2. **Значение кода в этих регистрах будет определяться частотой TIMxCNT, которая зависит от CK_INT и коэффициента предделителя PSC [76]**. Входной сигнал подключен к входу TI1.

Для измерения параметров ШИМ необходимо:

- выбрать активный вход TI1, записав в биты CC1S = 01 регистра TIMx_CCR1;

- настроить на фронт сигнала TI1FP1 – **CC1P = 0** регистра **TIMx_CCER**;
- выбрать активный вход для TI2, записав в CC2S = 10 регистра TIMx_CCMR1;
- настроить на срез сигнала TI1FP2, задав CC2P = 1 регистра **TIMx_CCER**;
- выбрать ведущий вход запуска – TS = 101 в регистре TIMx_SMCR (выбран TI1FP1);
- сконфигурировать ведомый таймер в режим сброса – SMS = 100 в регистре TIMx_SMCR;
- разрешить захваты – биты CC1E и CC2E регистра **TIMx_CCER** установить в 1.

**Режим подключения к внешнему энкодеру** используется для определения направления и угла поворота различных устройств. На основании этой информации можно рассчитать скорость (частоту вращения), ускорение и другие параметры.

В оценочном модуле используется электромеханический инкрементальный (квадратурный) энкодер. Он представляет собой устройство, которое с помощью двух контактов формирует на выходе две последовательности импульсов, смещенных по фазе (рисунок 1.28). Описание конструкции энкодера, используемого в оценочном модуле, приведенов разделе 3.1.4.

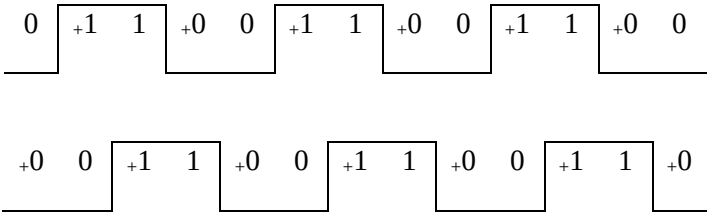



Рис. 1.28 – Временная диаграмма измерения параметров ШИМ

**Порядок чередования фаз указывает направление вращения, а число повторяющихся комбинаций – угол**

**поворота.** Для приведенного примера комбинация цифр в прямом направлении 00, 10, 11, 01, где первая цифра – логический уровень первого вывода энкодера, а вторая – второго вывода. Если, например, из состояния 11 выполнен переход к 01, то направление прямое, а 10 – обратное. Разрешающая способность энкодера определяется конструктивными особенностями – чем больше отверстий (больше количество повторяющихся комбинаций), тем точнее можно определить угол поворота (см. раздел 3.1.4). При изменении углового положения счетчик содержит количество импульсов, пропорциональных углу поворота. Переход на одну позицию формирует 4 импульса.

Если необходимо подсчитать число позиций, то значение счетчика следует разделить на 4. При известной разрешающей способности таким образом можно определить угол поворота (360 делят на содержимое счетчика, сдвинутое вправо на 2 разряда). Электрическая схема энкодера приведена на рисунке 1.29.

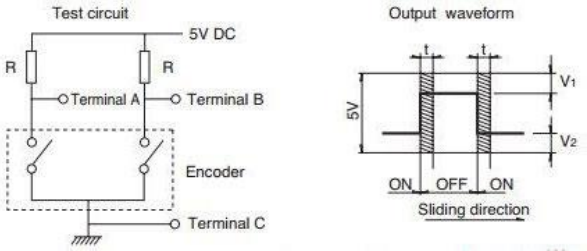



Рис. 1.29 – Электрическая схема энкодера

При электрическом согласовании с МК необходимо учитывать соотношение напряжения питания энкодера и МК. Если у энкодера Uпит = 5 В, то для STM32 подключать его выводы следует к толерантным входам GPIO или ограничивать входной ток с помощью резистора. При переключении механических контактов на фронте и срезе импульса возникает дребезг, с которым борются аппаратными или программными средствами [83].

**Программирование, как и в предыдущих режимах, состоит в инициализации элементов структуры от входа порта до счетчика (рисунок 1.18).**

Для работы энкодера с МК STM32 необходимо:

- Выбрать любой из счетчиков-таймеров общего назначения или продвинутых.
- Определить используемые линии GPIO каналов CH1 и CH2.
- Настроить выбранные линии на ввод с подтяжкой к плюсу, если отсутствуют внешние резисторы; при их наличии – обычный толерантный вход.
- Подключить вывод «С» на землю, а выводы «А» и «В» – ко входам CH1 и CH2.
- Разрешить тактирование СТ в регистре **RCC_APBnENR**, где n – номер периферийного интерфейса.
- Выбрать активный фронт или срез для **TI1** и **TI2** – биты **CC1P** и **CC2P** в регистре **TIMx_CCER**.
- Запрограммировать предварительный делитель **ICxPSC[1÷0]** и частоты входного фильтра **ICxF[3÷0]** регистра **TIMxCCMR1**. Эти параметры определяются временем дребезга контактов.
- Задать в регистре **TIMx_SMCR** один из режимов энкодера **SMS (2÷0)**. Если выбран счет по фронтам обоих входов **TI1** и **TI2**, то **SMS = 011** и счетчик будет тактироваться по фронту сигналов **TI1FP1** и **TI2FP2**.
- Записать значение, до которого будет считать СТ в регистр **TIMxARR**. Если необходимо определить угол в пределах **одного оборота** энкодера, то это значение равно количеству отверстий на диске умноженное на 4. Это значение должно быть увеличено в зависимости от числа оборотов.
- Включить счетчик **TIMxCR1**, установив бит **CEN**.
- Сбросить счетчик **TIMxCNT**.

Текущее значение счетчика можно прочитать в регистре **TIMxCNT**, используя события другого СТ, запрос ПДП (DMA), генерируемый часами реального времени, в режиме захвата по другому каналу, по прерыванию от системного таймера **SysTick** и события от других периферийных устройств.

На рисунке 1.30 представлена временная диаграмма, поясняющая работу энкодера в режиме 3, см. приложение П1.5 (**SMS = 011**) [21].

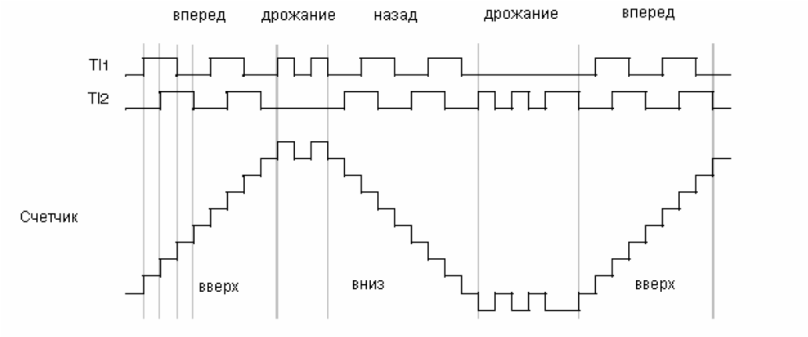



Рис. 1.30 – Временная диаграмма работы энкодера

**В этом режиме СТ работает как реверсивный счетчик (режим внешнего тактирования)**. Он считает от 0 до ARR или от ARR до 0 в зависимости от направления, где **ARR** – содержимое регистра автоперезагрузки **TIMx_ARR**.

Счетчик модифицируется в зависимости от направления вращения и угла поворота энкодера. **Направление вращения** указывает бит **DIR** регистра **TIMx_CR1**, который аппаратно изменяет свое значение по каждому переходу TI1 или TI2: 1 – значение счетчика уменьшается, 0 – увеличивается.

### Режим сравнения

Режим сравнения используется для формирования выходных сигналов различного вида: генераторы программируемой частоты и

скважности, одиночные импульсы заданной длительности и полярности, широтно-импульсные модуляторы и так далее. Не следует забывать, что каналы захвата/сравнения **используют общие ресурсы и могут работать только в режиме с разделением времени.**

Упрощенная структурная схема первого канала сравнения приведена на рисунке 1.31 [27].

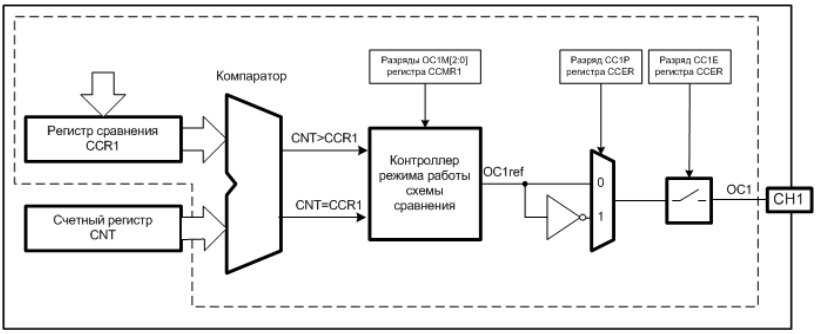



Рис. 1.31 – Упрощенная структурная схема канала сравнения 1

В регистр сравнения **TIMx_CCRn**, где **n** – номер канала, записывается значение, которое сравнивается со значением счетного регистра (счетчика) **CNT.** Когда значение счетчика станет равным или больше значения регистра сравнения на входе компаратора формируется сигнал, поступающий в контроллер режима работы схемы сравнения.

Управление контроллером выполняется разрядами **OCxM[2÷0]** регистров выбора режима сравнения **TIMxCCMR1** (каналы1 и 2) и **TIMxCCMR2** (каналы 3 и 4). Сравнение можно запретить, при сравнении можно формировать 0 или 1, инверсию предыдущего значения, установить режим ШИМ и так далее (см. форматы регистров в приложении 1).

Дальнейшее преобразование выходного сигнал контроллера **OCnref** выполняется под управлением регистра **TIMxCCER –**

разряд **CCnP** определяет активный уровень сигнала 0 или 1, а **CCnE –** разрешает или запрещает прохождение сигнала **OCn** на выход **CHn.**

Сигнал **OCnref** устанавливает флаги **CCnF** в регистре состояния **TIMxSR.** В зависимости от содержимого регистра **TIMx_DIER** может быть реализована процедура прерывания (биты **CCnIE**) или прямой доступ к памяти DMA (биты **CCnDE** в **TIMx_DIER**, и бит **CCDS** в **TIMx_CR2**).

Возможна **принудительная генерация сравнения** при установленных битах **CCnG** регистра **TIMxEGR.** Эта процедура может использоваться в том случае, если необходимо сформировать выходной сигнал **по результатам обработки программы**.

Временное разрешение – 1 такт счетчика. Временная диаграмма для канала 1, работающего в режиме инвертирования выходного сигнала, приведена на рисунке 1.32.

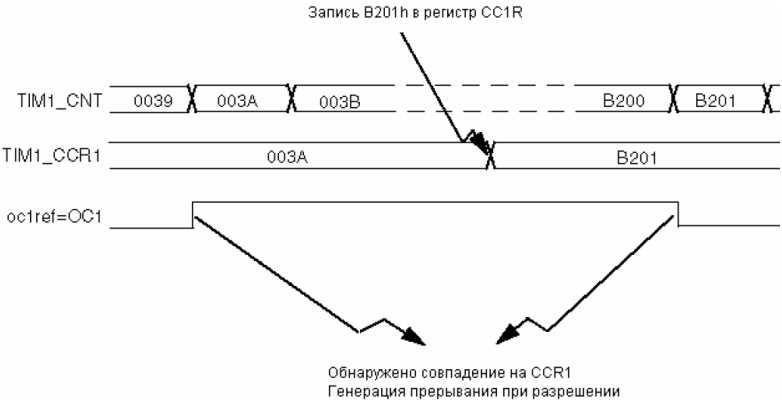



Рис. 1.32 – Временная диаграмма режима сравнения

Процедура инициализации заключается в следующем:

- Выбрать СТ и номер канала с учетом реализации требуемых альтернативных функций.

- Настроить выбранную линию на вывод в режим push-pull.
- Задать источник тактирования счетчика (внутренний или внешний) и значение предделителя в TIMx_ARR. В режиме внешнего тактирования следует разрешить прохождение сигнала ETR OCnCE в регистрах **TIMxCCMR1** и **TIMxCCMR2.**
- Задать режим сравнения биты **CCnS[1÷0]** в **TIMxCCMR1, TIMxCCMR2.**
- Записать в регистр сравнения **TIMx_CCRx** требуемое значение. Регистры TIMx_CCRx могут быть настроены на работу с предварительной буферизацией или без неё (бит **OCxPE** в **TIMx_CCMRn)**. Если буферизация выключена, то регистр **TIMx_CCRx** может быть перезаписан программно в любое время, в противном случае рабочий регистр будет обновлён только по событию **UEV.**
- Задать режим управления выходом в регистрах **TIMxCCMR1**, **TIMxCCMR2**, **TIMxCCER.**
- Определить способ ввода/вывода. При прерывании и/или ПДП выполнить инициализацию **(TIMx_DIER**, **TIMx_CR2).**
- Включить счетчик – бит **CEN** в регистре **TIMx_CR1.**

**Режим ШИМ** является частным случаем режима сравнения. Частота сигнала на выходе ШИМ определяется значением регистра автоперезагрузки TIMx_ARR, а скважность – содержимым регистра TIMx_CCRx.

Отличия при инициализации заключаются в следующем:

- Определить режим работы ШИМ записью в биты OCnM[2÷0] регистра TIMx_CCMRx значений «110» (режим 1 ШИМ) или «111» (режим 2 ШИМ). В режиме ШИМ (1 или 2) TIMx_CNT и TIMx_CCR всегда сравниваются на условие TIMx_CCRx ≤ TIMx_CNT или TIMx_CNT ≤ TIMx_CCRx (в зависимости от направления счетчика). Отличия этих режимов

состоит в полярности формируемых сигналов: режим1 – прямой, режим 2 – инверсный (см. формат регистра TIMx_CCMRx).

- Установить режим выравнивания ШИМ по фронту (срезу) или по центру (биты CSM[1÷0] регистра TIMxCR1).
- Разрешить работу с буферизацией регистрам сравнения TIMx_CCRx (бит OCnPE в TIMx_CCMRx) и автоперезагрузки TIMx_ARR (бит ARPE в регистре TIMxCR1).
- Установить UG в TIMx_EGR для принудительной генерации обновления счетчика формирования запросов ПДП и прерывания.

Работу ШИМ в режиме с **выравниванием по фронту** CSM[1÷0] = 0 (счетчик – суммирующий) иллюстрирует рисунок 1.33, где показано формирование сигналов для TIMx_ARR = 8 и различных значений регистра сравнения TIMx_CCRx.

При (TIMx_CNT) < (TIMx_CCRx) сигнал ШИМ OCxREF высокий, а в момент сравнения он переходит в низкий уровень. Если (TIMx_CCRx) ≥ (TIMx_ARR), то OCxREF = 1. При TIMx_CCRx = 0 OCxREF сбрасывается в 0.

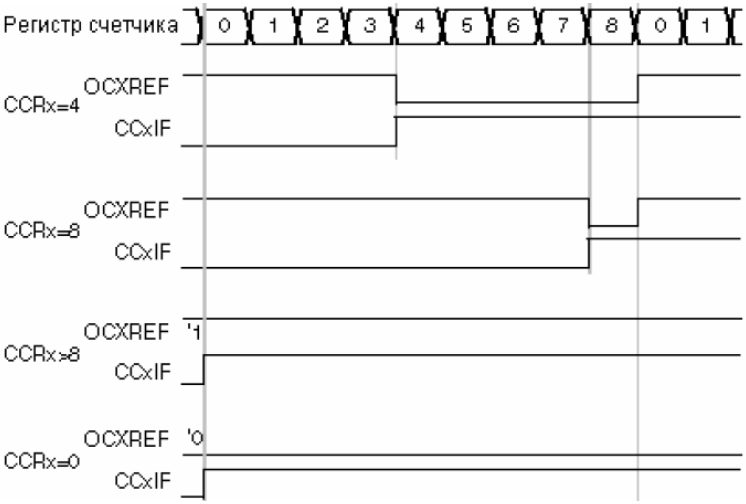



Рис. 1.33 – Режим выравнивания по фронту

При **выравнивании по центру** (в CSM[1÷0] любое значение, отличное от 0) счетчик работает в реверсивном режиме и изменяет свое состояние от 0 до TIMx_ARR, а затем – до 0. Для приведенного примера TIMx_ARR = 8. Выходной сигнал OCxREF изменяет свое состояние в момент равенства TIMx_CNT и TIMx_CCRx при прямом и обратном счете.

**При одной и той же частоте синхронизации и содержимом регистра сравнения длительность формируемого импульса и период будет в два раза больше, чем при выравнивании по фронту (рисунок 1.34).**

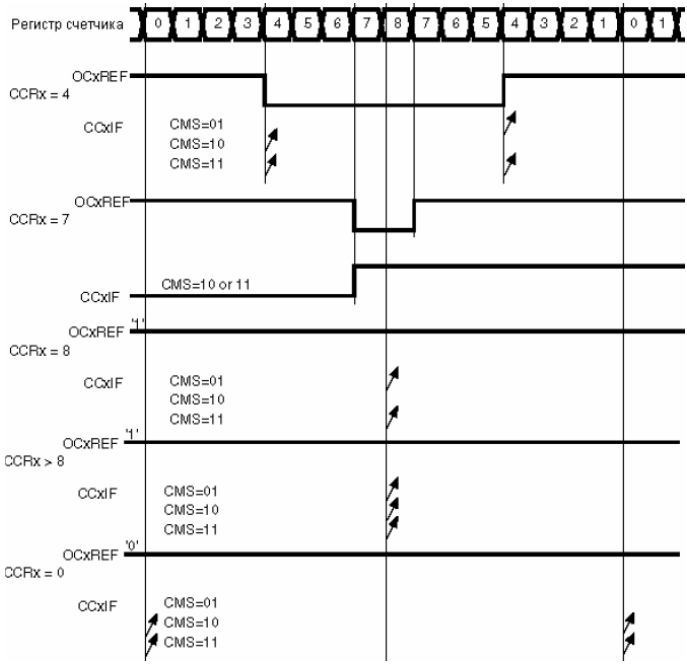



Рис. 1.34 – Режим выравнивания по центру

Флаг сравнения CCnIF устанавливается в регистре состояния TIMx_SR в зависимости от содержимого поля CMS регистра TIMxCR1 (совпадение при прямом, обратном счете или при любом направлении).

Бит направления DIR регистра TIMx_CR1 обновляется аппаратно и не должен изменяться программой.

**Выравнивание по центру обеспечивает большую разрешающую способность и точность фазировки** [76].

**Режим одиночного импульса (One-pulse Mode)** или режим одновибратора является частным случаем предыдущих режимов. При появлении события (внешнее воздействие или внутреннее) можно сформировать на выходе импульс, у которого программируется длительность и задержка его формирования. Генерация импульса может происходить в режиме сравнения или ШИМ. Если требуется минимальная задержка, то следует использовать режим ШИМ (1 или 2). Режим одиночного импульса устанавливается битом OPM в регистре TIMx_CR1. Тогда при появлении события UEV счётчик автоматически останавливается (сбрасывается бит CEN регистра TIMx_CR1).

Корректная работа в этом режиме возможна, если при прямом счете $CNT < CCRx \leq ARR$ (в частности, $0 < CCRx$), а при обратном – $CNT > CCRx$.

На рисунке 1.35 показан пример запуска одновибратора фронтом импульса, приходящего на вход TI2.

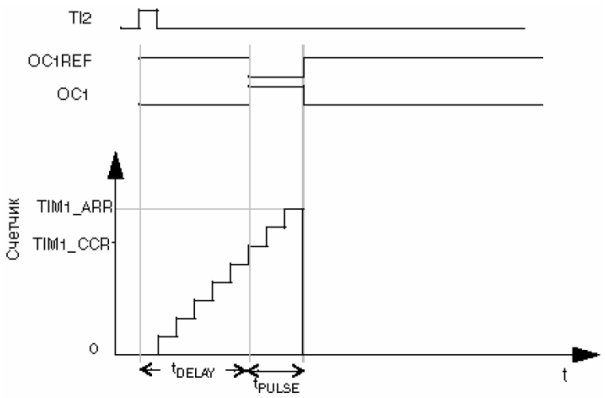



Рис. 1.35 – Режим одиночного импульса

Формирование импульса выполняется СТ1, работающим в режиме сравнения с внешним запуском. Аналогично реализуется этот режим и для каналов 2÷4.

**При заданной частоте синхронизации задержка $t_{DELAY}$ определяется содержимым регистра TIMx_ CCRx, а длительность импульса $t_{PULSE}$ – разностью между значениями, записанными в TIMx_ARR и TIMx_ CCRx.**

Алгоритм настройки изложен в [76]:

- настроить TI2 на вход и подключить его к сигналу TI2FP2 (CC2S[0÷1] = 01 регистр TIMx_CCMR1);
- задать активный фронт TI2FP2 (CC2P = 0, CC2NP = 0 в регистре TIMx_CCER);
- передать сигнал на выход мультиплексора TRGI (TS[2÷0] = = 110 в регистре TIMx_SMCR);
- настроить **режим ведомого с запуском по фронту** триггера TRGI (SMS[2÷0] = 110 в регистре TIMx_SMCR). С приходом фронта бит CEN регистра TIMx_CR1 будет устанавливаться в 1;
- включить ШИМ 2 (OC1M[2÷0] = 111 в регистре TIMx_CCMR1) для того, чтобы сформировать переход сигнала из 0 в 1 при достижении счётчиком значения TIMx_CCR1 и переход из 1 в 0 при достижении счётчиком значения TIMx_ARR;
- записать требуемые значения в регистры TIMx_CCR1 и TIMx_ARR, а также сформировать событие UEV, чтобы эти значения переписались в рабочие регистры;
- в случае необходимости включить и настроить предварительные делители, записав OC1PE = 1 (в регистре TIMx_CCMR1) и ARPE = 1 (в регистре TIMx_CR1);
- задать исходное значение на выходе равное 0 (CC1P = 1 в регистре TIMx_CCER);
- установить режим одиночного импульса (OPM = 1 в регистре TIMx_CR1);

- для приведенного примера биты DIR и SMS регистра TIMx_CR1 должны быть сброшены.

С приходом TI2 на выходе будет сформирован одиночный импульс. В режиме одиночного импульса при определении фронта на входе TI1 устанавливается бит CEN, который разрешает счетчик.

Приведенный алгоритм не обеспечивает быстрого формирования импульса, так как на включение счетчика и сравнение требуется определенное количество циклов (5 циклов) синхрогенератора. Для обеспечения минимальной задержки $\mathbf{t_{DELAY}}$ (3 цикла) следует установить бит OCxFE в регистре TIMx_CCMRx. В результате базовый выходной сигнал OCxREF (и соответственно выход OCx) будет переключаться сразу по событию от триггера, не принимая во внимание результат операции сравнения. **Минимальная задержка обеспечивается только в режиме ШИМ (1 или 2).**

### Каскадное соединение счетчиков

Каскадное соединение возможно только с помощью **внутренней** коммутации. У каждого СТ для этих целей формируется выходной триггерный сигнал **TRGO** и имеется 4 входных сигнала **ITR0, ITR1, ITR2, ITR3**. Сигнал TRGO **ведущего СТ** может быть подключен к одному или нескольким входам **ведомых СТ.** Инициатором формирования TRGO могут быть сигнал обновления счетчика, бит **UG** регистра **TIMx_EGR**, внешний сигнал синхронизации, момент захвата/сравнения (см. регистр **TIMxCR2**).

**Соотношение «ведущий-ведомый» может быть установлено между каналами одного СТ.** Например, выполнен захват на одном канале, сформировано событие, и на основании этого события СТ сам себе посылает команду [74].

Функции сигнала ведущего СТ **TRGO** определяют разряды **MMS** регистра **TIMxCR2**:

- **MMS = 000 – режим «Reset»**. Сигнал TRGO формируется при установке разряда **UG** регистра генерации событий **EGR**. Этот разряд устанавливается программно или внешним сигналом. Он приводит к формированию события переполнения.
- **MMS = 001 – режим «Enable»**. Сигнал TRGO формируется при возникновении сигнала CNT_EN, который управляет таймером (разрешает/запрещает счет). Можно использовать для одновременного запуска нескольких таймеров.
- **MMS = 010 – режим «Update»**. Сигнал TRGO формируется при возникновении события обновления UEV. В этом режиме можно использовать таймер-мастер в качестве предварительного делителя для подчиненного таймера.
- **MMS = 100÷111 – режим «Compare»**. Сигнал TRGO формируется, при возникновении сигнала сравнения в соответствующем канале (OC1Ref, OC2Ref, OC3Ref, OC4Ref) (рисунок 1.36) [29].

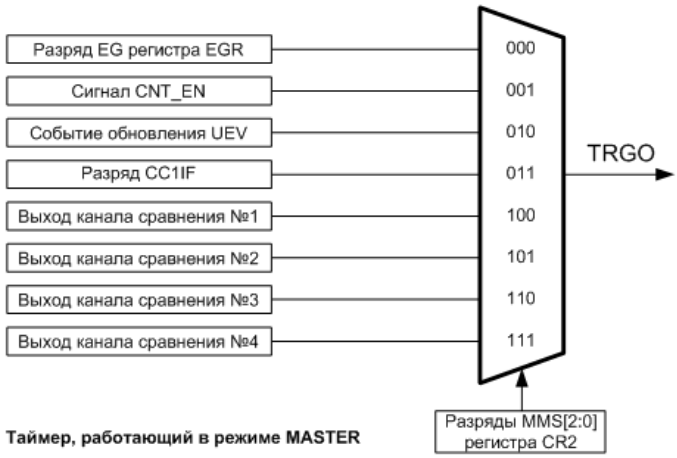



Рис. 1.36 – Источники запросов ведущего СТ

В **режиме ведомого** СТ синхронизируется от внешнего сигнала, в качестве которого может быть выбран:

- один из внутренних таймеров (ITR3-ITR0);
- сигнал на выводе контроллера (ETR, TI1 или TI2) (рисунок 1.37).

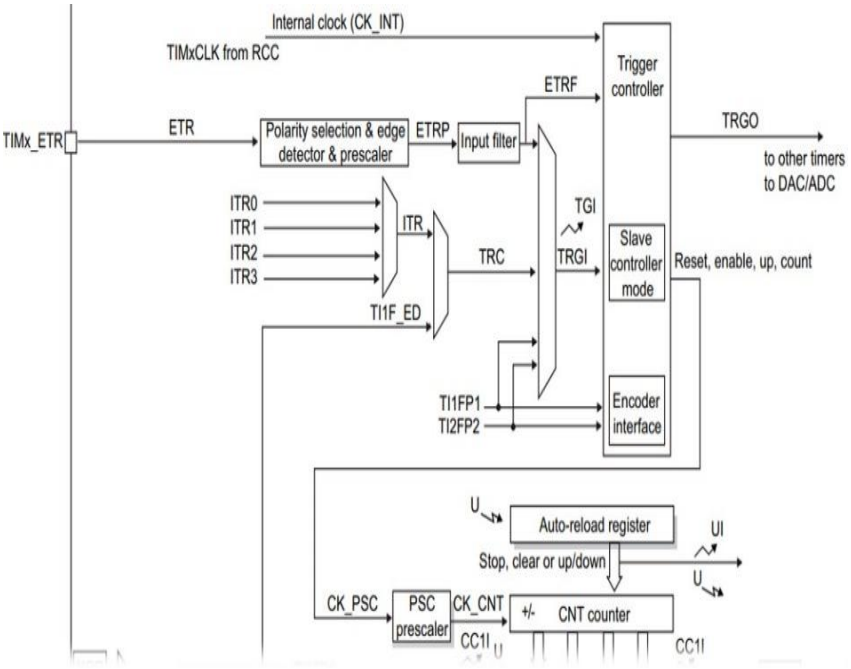



Рис. 1.37 – Источники синхронизации ведомого СТ

Активный вход синхронизации задается битами **TS2÷TS0 TIMx_SMCR.** Режимы работы ведомого описаны выше (синхронизация от внешнего сигнала). **Допустимые соединения** триггерных входов/выходов ведущего и ведомого отражены в таблице 1.13.

Например, если TIM3 – выбран ведомым, то ведущими могут быть TIM1, TIM2, TIM5, TIM4. Если в выбранной конфигурации МК таймер отсутствует (для STM32F103C8T6 – это TIM5), то эта

связь недоступна. Пример каскадного соединения ведущего (**TIM1)** и ведомого (**TIM2**) представлен на рисунке 1.38.

Таблица 1.13. Допустимые соединения между ведомым и ведущим СТ

| Ведомый TIM | ITR0 (TS = 000) | ITR1 (TS = 001) | ITR2 (TS = 010) | ITR3 (TS = 011) |
|---|---|---|---|---|
| TIM2 | TIM1 | TIM8 | TIM3 | TIM4 |
| TIM3 | TIM1 | TIM2 | TIM5 | TIM4 |
| TIM4 | TIM1 | TIM2 | TIM3 | TIM8 |
| TIM5 | TIM2 | TIM3 | TIM4 | TIM8 |

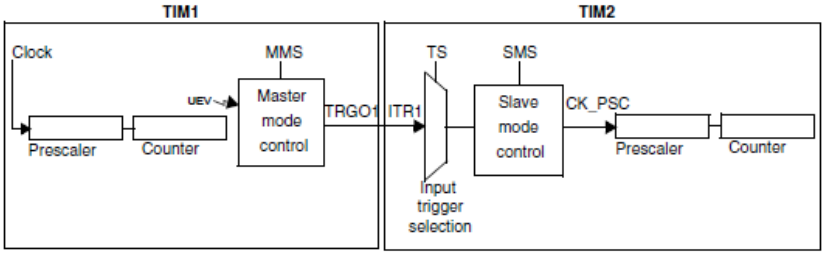



Рис. 1.38 – Каскадное соединение TIM1 и TIM2

Например, необходимо организовать 48-разрядный СТ на основе каскадного соединения TIM3, TIM2, TM4. Для этого следует настроить TIM3 на работу по событию переполнения (MMS0÷MMS2 регистра **TIMxCR2** должно быть равно 010), выход ITR0 таймера TIM3 подключить к входу ITR2 TIM2(010), а ITR0 TIM2 – к ITR1 TIM4 (001). Требуемые подключения задаются в битах TS0÷TS2 регистра **TIMx_SMCR**. Далее следует запустить счетчики **– бит CEN TIMxCR1**.

Преимущества каскадного соединения можно продемонстрировать на примере измерения длительности положительного временного интервала. Измерение выполняется в режиме захвата, при котором фиксируется момент прихода фронта входного сигнала t1 в регистре захвата, его сохранение, затем та же

процедура повторяется для среза – t2, а длительность определяется как T = t2 – t1 [50]. Для вычисления T требуется выполнение определенного алгоритма с участием процессорного блока.

При каскадном соединении можно реализовать режим «ведущий-ведомый» для 1 и 2 каналов одного таймера. Измеряемый сигнал можно подавать на один вход, так как 1 и 2 каналы СТ могут обмениваться сигналами. Первый канал настраивается на фронт, а второй – на срез. Ведущим является первый канал, который формирует **TI1FP1**, а второй – ведомый канал работает в режиме сброса **Reset mode**. С приходом фронта счетчик **CNT** сбрасывается, а по срезу – его содержимое сохраняется в регистре захвата второго канала. Дальнейшая обработка определяется решаемой задачей: программный ввод/вывод, прерывание, ПДП (рисунок 1.39) [74].

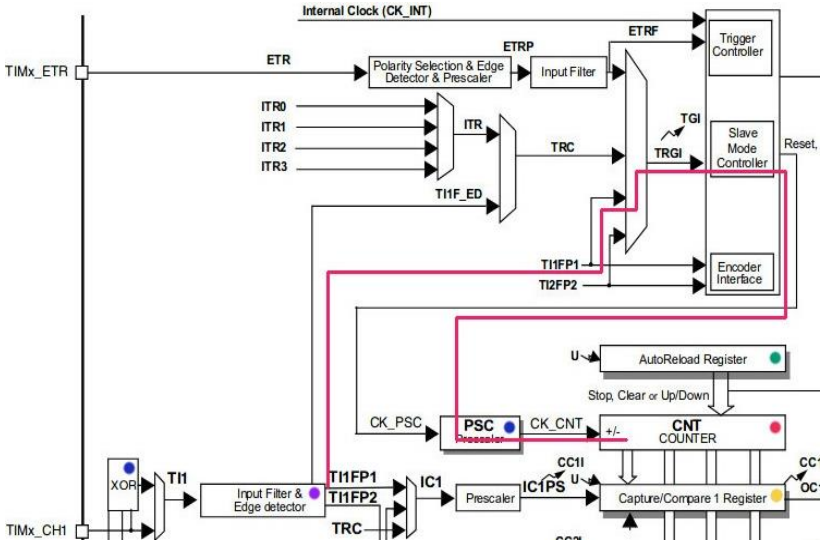



Рис. 1.39 – Прохождение сигнала при измерении временного интервала

Блок СТ, по нашему мнению, является одним из функционально насыщенных и при его использовании много нюансов, которые не нашли отражение в этом разделе.

При измерении и формировании сигналов с помощью СТ могут потребоваться повышенные метрологические характеристики. В этом случае необходимо учитывать задержки, возникающие в процессе преобразования, и использовать внешний кварцевый резонатор, так как стабильность внутренних генераторов составляет 1–2 %. Практическое использование возможностей СТ отражено в [69, 75, 74, 79].

### 1.6.2.2 Часы реального времени

Часы реального времени (англ. **RTC**) предназначены для отсчета секунд, минут, часов, дней недели, месяцев, годов. В 8-разрядных МК для этих целей использовали внешние **RTC** с последовательным интерфейсом (SPI, I2C).

В МК STM32 имеется внутренний блок **RTC**, функции которого определяются особенностями конкретной модели МК. В продвинутых МК большая часть функций отсчета времени и управления реализуется аппаратно.

В STM32F103C8T6 аппаратно реализуются функции:

- Делитель частоты с программируемым коэффициентом (два 16-разрядных регистра **RTC_PRLH**, **RTC_PRLL**).
- Двоичный счетчик времени, сек (два 16-разрядных счетных регистра **RTC_CNTH**, **RTC_CNTL).**
- Флаги внутренних событий**:** переполнение входного делителя (формирование секундного импульса) **SECF**, совпадение значений счетного и сигнального регистров **ALRF**, переполнение счетного регистра **OWF**. Флаги хранятся в регистре управления **RTC_CRL**. Эти флаги могут вызвать прерывание, если установлены биты разрешения в регистре **RTC_CRLH**.
- Формирование на выводе **TAMPER** тактового сигнала, поделенного на 64, секундных импульсов или сигнала тревоги (в

момент равенства счетного и сигнального регистров). Управление выводом **TAMPER** выполняется в регистре **BKP_RTCCR**.

Регистры часов реального времени находятся в зоне МК, питание которой может быть выполнено от резервного внешнего источника **Vbat** (например, батарейки). Кроме RTC в этой зоне находятся регистры, которые можно использовать по своему усмотрению. При отключении или сбое основного напряжения питания МК эта зона автоматически подключается к внешнему источнику, сохраняя заданные режимы и обеспечивая бесперебойную работу RTC и регистров. Структурная схема RTC приведена на рисунке 1.40.

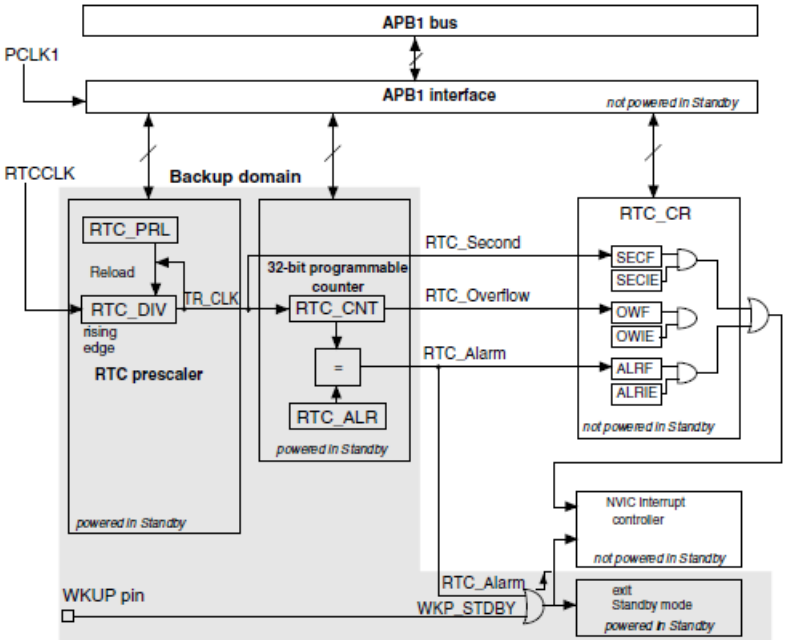



Рис. 1.40 – Структура блока часов реального времени

Входная частота **RTCCLK** может поступать от трех источников:

- **LSI** – низкочастотный внутренний RC-генератор на 40 KHz;
- **LSE** – внешний, «часовой» кварцевый резонатор на 32 768 Hz;
- **HSE**/128 – внешний высокочастотный кварцевый резонатор с предварительным делителем на 128.

Источник входной частоты задается в регистре **RCC_BDCR битами RTSEL(1÷0).** Деление частоты **RTCCLK** выполняет предделитель **RTC prescaler**. Коэффициент деления записывается в регистры старшего **RTC_PRLH** и младшего **RTC_PRLL** слова.

Частота импульсов на входе счетчика **RTC_CNT** определяется как **FTR_CLK = FRTCCLK / (PRL[19÷0] + 1).**

При частоте входных импульсов 32768Hz для получения секундных импульсов в PRL необходимо записать 0x7FFF.

Если необходимо контролировать определенное (допустимое) время, то следует использовать сигнальный регистр **RTC_ALR**, состоящий из двух 16-разрядных регистров **RTC_ALRH**, **RTC_ALRL**. В момент равенства **RTC_CNT** и **RTC_ALR формируется сигнал RTC Alarm** и устанавливается флаг **ALRF**. Аналогично устанавливаются и другие флаги (секундный **SECF**, переполнение счетчика **OWF)** в регистре **RTC_CRL**.

При реализации, например, часов-будильника, можно ежесекундно выводить текущее время на индикатор, а в момент сравнения **RTC_CNT** и **RTC_ALR –** включать звонок.

Если необходим календарь, то в STM32F103C8T6 определение дня, месяца, года выполняется программно на основании известных алгоритмов [31, 3]. В более поздних моделях STM32 эти функции реализованы аппаратно.

Взаимодействие **RTC** с другими блоками МК выполняется через периферийную шину **APB1**. Так как **RTC** и **APB1** работают **асинхронно** – необходимо **определенное время** для реализации операций записи/чтения регистров. Готовность к этим действиям указывают флаги **BSF**, **CNF**, **RTOFF** регистра **RTC_CRL.** Сигнал

синхронизации с внешними устройствами формируется на выходе **TAMPER**.

Программная модель часов реального времени приведена в приложении П1.6.

Алгоритм работы с часами реального времени [3, 80]:

- Включить тактирование RTC и резервных регистров.
- Разрешить доступ к резервным регистрам.
- Сбросить резервные регистры.
- Подключить LSE.
- Синхронизировать RTC с APB1.
- Установить входную частоту счетчика.
- Задать при необходимости значение сигнального регистра.
- Записать в резервные регистры какую-либо константу для проверки повторного включения. После первого включения микроконтроллер запустит функцию инициализации и включит **RTC**. Во время следующих запусков, функция будет включать только тактирование нужных модулей и доступ к области резервных данных **PWR_BackupAccessCmd (ENABLE)**. Это необходимо, чтобы получить доступ к счетчику **RTC**.
- Разрешить в случае необходимости работу с прерываниями.
- Разрешить работу RTC.
- Дождаться окончания синхронизации.
- Выполнить действия в соответствии с заданием.

Наиболее просто инициализировать **RTC** с применением код-генератора **CUBE MX** [18].

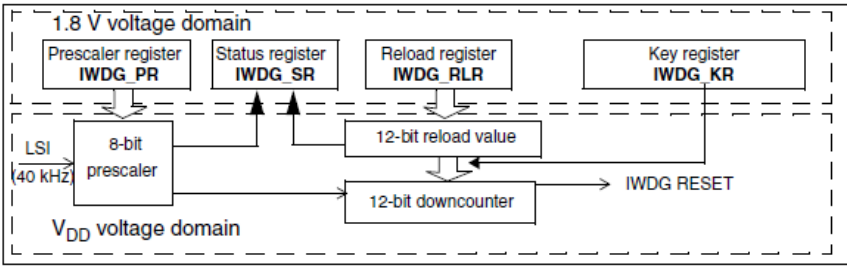
### 1.6.2.3 Сторожевой таймер

Сторожевой таймер СтТ (Watchdog) предназначен для контроля состояния процесса обработки и управления, выполняемого ОМК. Он предотвращает «зависание» ОМК при возникновении нештатных ситуаций.

На стадии инициализации в СтТ задается максимально допустимое время работы программы или её фрагмента. Если установленный временной интервал закончился, генерируется системный сброс. При нормальном ходе процесса обработки СтТ **должен программно перезагружаться через время меньшее установленного временного интервала**. Если сброс произошел из-за СтТ, то в специальном регистре будет установлен соответствующий флаг [50].

В состав МК входит два СтТ: независимый Independent watchdog (**IWDG**) и оконный Windowed Watchdog (**WWDG**). **Регистры сторожевого таймера расположены в зоне автономного (резервного) питания ВКР.**

Источником синхронизации **IWDG** является внутренний низкочастотный RC генератор (**LSI**), что обеспечивает его работоспособность даже при отказе системного тактового генератора. Следует отметить, что в различных МК реальная частота этого генератора может изменяться в пределах от 30 до 60 кГц (заявленная частота – 40кГц) и зависит от температуры (диапазон изменения ±9%) [72]. Структурная схема **IWDG** приведена на рисунке 1.41.




*The watchdog function is implemented in the $V_{DD}$ voltage domain that is still functional in Stop and Standby modes.*

Рис. 1.41 – Структура сторожевого таймера **IWDG**

Сигналы генератора **LSI** поступают на вход 8-разрядного предделителя, коэффициент деления которого определяется регистром **IWDG_PR**. Контролируемое время определяется содержимым регистров **IWDG_PR** и **IWDG_RLR**. После запуска СтТ содержимое **IWDG_RLR** переписывается в вычитающий счетчик (**downcounter**). По умолчанию этот регистр имеет значение **0xFFF**. **В момент достижения нуля формируется сигнал сброса IWDG RESET.** Флаг сброса устанавливается в регистре **RCC**.

Управление СтТ выполняется регистром **IWDG_KR,** в младшие 16 бит которого записываются определенные коды: **0xCCCC –** запуск таймера; **0x5555 –** разрешение доступа к регистрам **IWDG_PR** и **IWDG_PLR**, **0xAAAA** – перезагрузка счетчика.

В момент обновления содержимого регистров **IWDG_PR** и **IWDG_RLR** устанавливаются соответствующие флаги в регистре состояния **IWDG_SR**.

Для предотвращения сброса МК необходимо периодически перезагружать **IWDG_RLR**, записывая команду **0xAAAA** в регистр **IWDG_KR**.

Период таймера вычисляется следующим образом: **T = ((4*2^IWDG_PR) * IWDG_RLR)/40 [мс].**

Значение **T** для тактовой частоты 40 кГц можно определить с помощью таблицы **Table 94** [17], часть которой показана в таблице 1.14. Минимальное время T – 0,1 мс, максимальное – 26214,4 мс.

Таблица 1.14. Настройки периода сторожевого таймера

| Значение предделителя | PR[2:0] | Минимальный таймут (мс) RL[11:0] = 0x000 | Максимальный таймаут (мс) RL[11:0] = 0xFFF |
|---|---|---|---|
| /4 | 0 | 0.1 | 409.6 |
| /8 | 1 | 0.2 | 819.2 |
| /16 | 2 | 0.4 | 1638.4 |
| /32 | 3 | 0.8 | 3276.8 |
| /64 | 4 | 1.6 | 6553.6 |

**По умолчанию регистры IWDG_PR и IWDG_RLR защищены от записи. Прежде чем изменить их значения, необходимо записать код 0x5555 в регистр IWDG_KR.**

Программная модель сторожевого таймера приведена в приложении П1.7

**Алгоритм инициализации IWDG:**

- Включить генератор **LSI** (40 kHz);
- Разрешить доступ к регистрам (**IWDG_KR = 0x5555);**
- Задать значения **IWDG_PR** и **IWDG_RLR** в соответствии с требуемым временем **T;**
- Запустить таймер (**IWDG_KR = 0xCCCC**).

**Оконный сторожевой таймер WWDG** используется в том случае, если необходимо контролировать работу программы в определенном временном интервале (окне времени). Он позволяет обнаруживать не только превышение времени, но и запаздывание временных событий. Структурная схема **WWDG** приведена на рисунке 1.42.

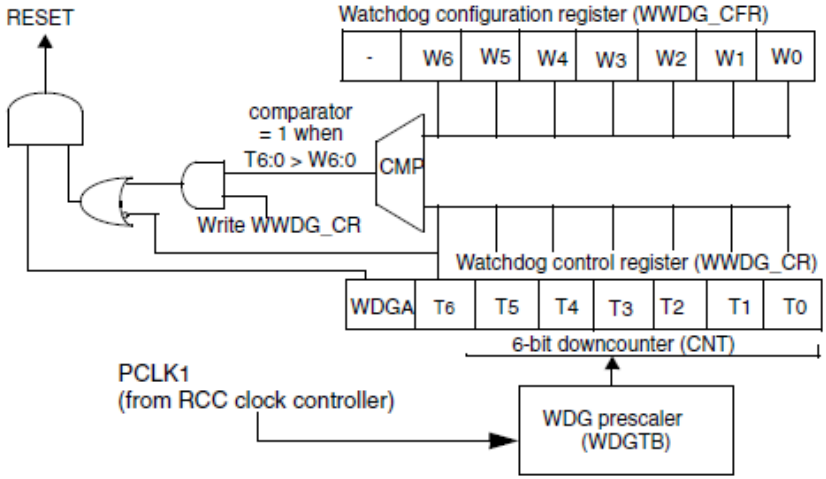



Рис. 1.42 – Структура сторожевого таймера **WWDG**

Особенности **WWDG:**

- синхронизация таймера сигналом PCLK1 (PCLK1 – частота периферийной шины **APB1**);
- использование входного предделителя **WDGTB**, который **аппаратно** делит входную частоту **на 4096** и позволяет **дополнительно программно** устанавливать **коэффициенты деления 1,2,4,8;**
- наличие семиразрядного вычитающего счетчика **(T6÷T0),** значение которого хранится в регистре **WWDG_CR**, и разрядов временного окна **(W6÷W0),** находящихся в регистре **WWDG_CFR**;
- формирование **двух сигналов сброса**:
    - при попытке записи в сторожевой таймер, когда **содержимое разрядов (T6÷T0) регистра WWDG_CR будет больше содержимого разрядов (W6÷W0)** регистра **WWDG_CFR** (**раньше требуемого времени**);
    - при переходе из состояния 0x40 в 0x3F (**позже требуемого времени**);
- использование прерывания для предупреждения о необходимости перезагрузки таймера.

Для инициализации СтТ необходимо записать исходные значения в счетчик и регистр временного окна.

Значение, записанное в **WWDG_CR,** должно обязательно **в разряде T6 содержать единицу** (любое число из диапазона 0x7F÷0x40), чтобы не произошло немедленного сброса.

Содержимое (**T5÷T0**) определяет максимальную временную задержку до сброса сторожевого таймера:

$$T_{wwdg} = T_{pclk1} * 4096 * 2^{WDGTB} * (T[5÷0] + 1),$$

где **WDGTB –** значение 7 и 8 разрядов регистра **WWDG_CFR**.

Разряды (**W6÷W0**) регистра **WWDG_CFR** задают время, в течении которого **перезагрузка таймера вызывает формирование сигнала сброса.**

Запуск СтТ выполняется установкой бита **WWDG_CR.WDGA**. **После запуска останов СтТ возможен только включением общего сброса.**

Счетчик начинает работу и в момент перехода из состояния 0x40 в 0x3F формируется сигнал сброса. **Флаг сброса устанавливается в бите WWDGRST регистра RCC_APB1RSTR**.

В процессе работы выполняется сравнение текущего значения счетчика и регистра окна схемой **CMP**. Если контролируемое событие выполнилось раньше установленного времени, то следует выполнить перезагрузку СтТ.

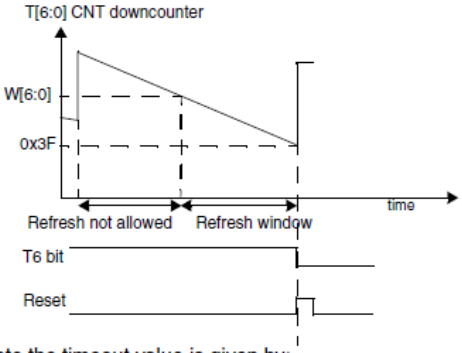
Перезагрузка допустима, когда счетчик находится в интервале **0xF3 < (T6:T0) < (W6:W0) (**рисунок 1.43). Если перезагрузка выполняется в момент **(T6:T0) > (W6:W0)**, то будет сформирован сигнал сброса.

Перезагрузка СтТ возможна **по запросу прерывания самого таймера**. В этом случае при достижении счетчиком значения 0x40 устанавливается бит EWIF в регистре состояния WWDG_SR. Если прерывание разрешено (**WWDG_CFR.EWI = 1**), то будет вызвано прерывание и соответствующая подпрограмма обработки перезагрузит СтТ.

Сброс запроса прерывания выполняется записью нуля в разряд EWIF регистра WWDG_SR.

При необходимости можно сгенерировать программный сброс процессора путём установки в единичное состояние бита **WDGA** и сбросив бит **T6** [34].

**Минимальный временной интервал при частоте APB1 36 МГц составляет 113 мкс, а максимальный – 58,25 мс.**




The formula to calculate the timeout value is given by:

$$T_{WWDG} = T_{PCLK1} \times 4096 \times 2^{WDGTB} \times (T[5:0] + 1) \quad \text{(ms)}$$

where:

$T_{WWDG}$: WWDG timeout

$T_{PCLK1}$: APB1 clock period measured in ms

**Min-max timeout value @36 MHz (PCLK1)**

| WDGTB | Min timeout value | Max timeout value |
|---|---|---|
| 0 | 113 µs | 7.28 ms |
| 1 | 227 µs | 14.56 ms |
| 2 | 455 µs | 29.12 ms |
| 3 | 910 µs | 58.25 ms |

Рис. 1.43 – Временная диаграмма работы **WWDG**

**Алгоритм инициализации WWDG:**

- Включить тактирование таймера на шине APB1 (бит **RCC_APB1ENR.WWDGEN = 1**).
- Задать требуемую частоту в регистре WWDG_CFR.
- Задать временное окно в регистре WWDG_CFR.
- Установить начальное значение счетчика в соответствии с требованиями и разрешить работу CтT в регистре WWDG_CR. Необходимо помнить, что **T6** должен быть равен 1.

Если предполагается использование прерывания, то

- разрешить прерывание в регистре **NVIC_ISER[0]**, установив нулевой разряд в «1» (номер вектора WWDG – 0);

- очистить флаг EWI (бит **WWDG_SR.EWIF = 0**);
- разрешить прерывание (бит **WWDG_CFR_EWI = 1**);
- написать подпрограмму перезагрузки таймера.

Для остановки СтТ необходимо отключить тактирование (бит **RCC_APB1ENR.WWDGEN = 0**). Особенности программирования сторожевого таймера с использованием **HAL** приведены в [20].

### 1.6.3 Последовательный порт SPI

SPI является популярным последовательным **синхронным** интерфейсом передачи данных между микроконтроллером и внешними периферийными устройствами. Используется в основном для внутриплатного или внутрикрейтового обмена. Особенностью этого интерфейса является несложная аппаратная реализация (приемник и передатчик представляют собой **удаленные** сдвиговые регистры, объединенные общими цепями сдвига), относительно высокая пропускная способность, отсутствие стандартизованного протокола обмена (регламентирован только физический уровень) [50].

Принцип обмена по интерфейсу поясняет рисунок 1.44.

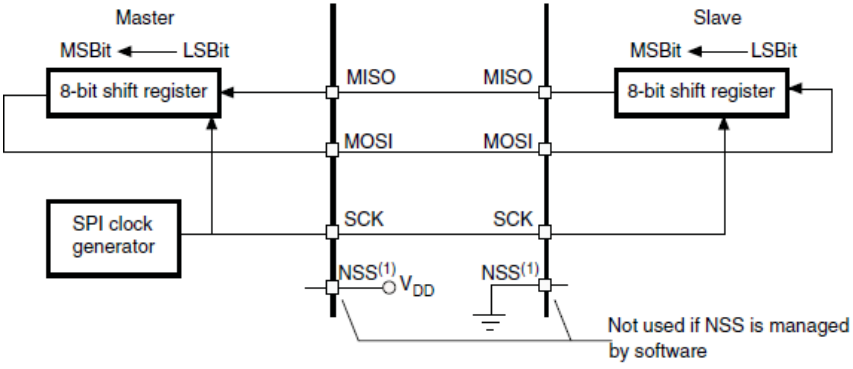



Рис. 1.44 – Схема соединения при дуплексном обмене по SPI

На рисунке изображены:

**Master** – ведущее устройство;

**Slave** – ведомое устройство;

**MISO** (Master In / Slave Out data) – вход в режиме Master и выход в режиме Slave;

**MOSI** (Master Out / Slave In data) – выход в режиме Master и вход в режиме Slave;

**SCK**– тактовый сигнал, формируемый Master;

**NSS** – входная линия, предназначенная для выбора **ведомого** устройства (подача на его вход логического нуля), а в режиме **Multimaster** с помощью этой линии можно переключать режимы **Master/Slave.** Программное управление **NSS** рассмотрено ниже.

Микроконтроллеры STM32 в зависимости от семейства и типа корпуса могут иметь до шести независимых блоков SPI. **В состав STM32F103xx входит два блока, один из которых (SPI1) присоединен к более скоростной шине APB2, а второй (SPI2) – к APB1.**

SPI1 может работать на частоте до 36 Мбит/с, а SPI2 – до 18 Мбит/с.

Основные функции:

- Режимы работы – **Master/Slave** (один ведущий –несколько ведомых), **Multimaster** (несколько ведущих).
- Возможность реализации дуплексного, симплексного или полудуплексного обменов. Дуплексный обмен представлен на рисунке 1.44 и требует для реализации 3 линии, симплексный – 2 линии, полудуплексный – 2 линии (линия **MOSI** ведущего устройства соединяется с линией **MISO** ведомого)**.**
- Программное или аппаратное управление выводом **NSS**.
- Длина кадра – 8 или 16 бит.
- Программное управление полярностью, фазой тактового сигнала и порядком следования бит. Допустимы два порядка

следования: наиболее значимыми битами вперед «most significant bit» (MSB-first) или наименее значимыми – «least significant bit» (LSB-first).

- Аппаратное формирование циклического избыточного кода CRC при передаче данных и его контроль при приеме. Использование CRC позволяет поддерживать базовые режимы работы с картами **SD/MMC**.
- Программное управление частотой синхронизации генератора от $f_{PCLK}/2$ до $f_{PCLK}/256$ (8 значений), где $f_{PCLK}$ – частота периферийной шины.
- Формирование флагов состояния процесса обмена и на их основе организация ввода-вывода в программном режиме, по прерыванию или в режиме ПДП.

В первом столбце таблицы 1.15 приведены события, которые формируют флаги в регистре состояния SPI_SR:

**Transmit buffer empty flag** – буфер передатчика свободен.

**Receive buffer not empty flag** – данные находятся в буфере приемника.

**Master Mode fault event** – ошибка режима. В режиме Master на входе NSS обнаружен сигнал низкого уровня, и он становится ведомым.

**Overrun error** – переполнение буфера приема (принятый байт не прочитан, а на его место записан следующий).

**CRC error flag** – ошибка контрольной суммы.

Таблица 1.15. Флаги состояния блока SPI

| Событие прерывания | Флаг события | Флаг разрешения прерываний |
|---|---|---|
| Transmit buffer empty flag | TXE | TXEIE |
| Receive buffer not empty flag | RXNE | RXNEIE |
| Master Mode fault event | MODF | ERRIE |
| Overrun error | OVR | ERRIE |
| CRC error flag | CRCERR | ERRIE |

Во втором столбце таблицы 1.15 приведены флаги этих событий, а в третьем – маски прерываний, находящиеся в регистре SPI_CR2. Однако **вектор прерываний** для всех перечисленных событий **один.**

- Возможность перекоммутации линий SPI (ремапинг для SPI1, таблица 1.16);

Таблица 1.16. Ремапинг блока SPI1

| Альтернативные функции | SPI1_REMAP = 0 | SPI1_REMAP = 1 |
|---|---|---|
| SPI1_NSS | PA4 | PA15 |
| SPI1_SCK | PA5 | PB3 |
| SPI1_MISO | PA6 | PB4 |
| SPI1_MOSI | PA7 | PB5 |

- Работа в режиме интерфейса **I2S,** который используется для передачи стереофонических данных между цифровыми аудиоустройствами.

Структура блока SPI представлена на рисунке 1.45.

Прием и передача информации в/из внешнего устройства выполняется в регистре сдвига (**Shift register**), а организацию взаимодействия с периферийной шиной обеспечивают буфер приемника (**Rx buffer**) и буфер передатчика (**Tx buffer**).

Частота на выходе **SCK** определяется частотой периферийной шины $f_{PCLK}$ и предделителем (**Baud rate generator**), коэффициент деления которого задается битами **BR2÷BR0** регистра управления **SPI_CR1**.

**Управление NSS** выполняется схемой мультиплексора **Communication control**, на вход которой поступают **внешний** сигнал **NSS** и бит **SPI_CR1.SSI**, а переключение этих сигналов осуществляет **SPI_CR1.SSM**. Если **SPI_CR1.SSM = 1** (программное управление), то значение **NSS** определяет бит **SPI_CR1.SSI** («0» – для ведомого, «1» – для ведущего). Если внешний вывод **NSS** остается свободным, то его можно использовать для других целей.

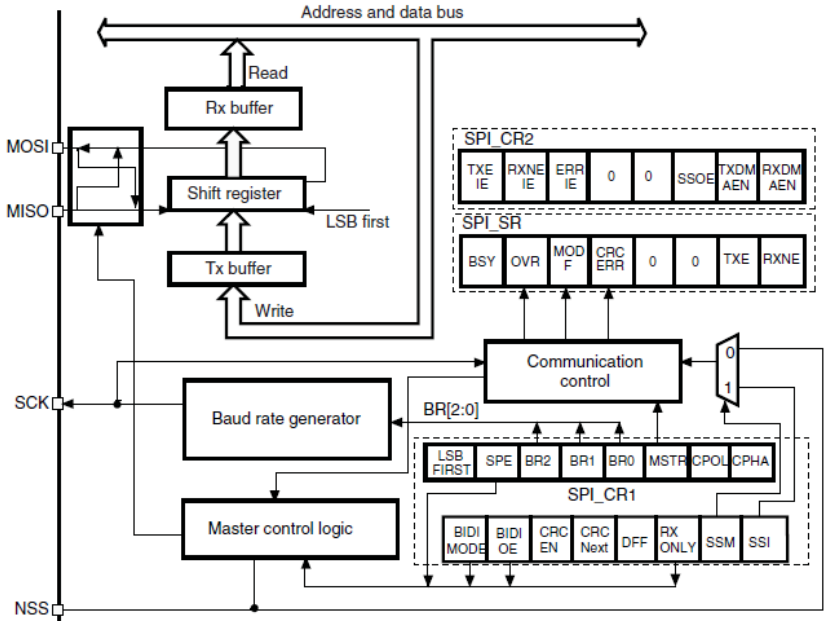



Рис. 1.45 – Структура блока SPI

При аппаратном управлении (**SPI_CR1.SSM = 0**) **NSS** определяется сигналом, поданным на внешний вывод (лог. «0» – для ведомого, лог. «1» – для ведущего).

В **мультипроцессорном режиме** при управлении **NSS** необходимо учитывать значение бита **SPI_CR2.SSOE**.

Основные функции управления блоком SPI реализует **Master control logic**. Принцип работы блока рассмотрим на примере передачи данных от ведущего контроллера к ведомому.

На стадии инициализации задаются:

- статус контроллера,
- основные параметры обмена,
- очищаются флаги в регистре состояния,
- выбирается способ ввода-вывода и выполняется запуск SPI.

Требуемые данные загружаются в регистр **SPI_DR**. При этом флаг занятости сдвигового регистра **BSY устанавливается**, а **TXE сбрасывается**, указывая, что буферный регистр передачи пока занят.

После загрузки данных в сдвиговый регистр начинается процедура сдвига, а **TXE = 1**, что указывает на возможность загрузки очередных данных. С каждым тактом **SCK** передается очередной бит на линию **MOSI** и принимается по линии **MISO**. Когда передан (а значит и принят) последний бит, содержимое сдвигового регистра перегружается в буфер приемника и устанавливается флаг **RXNE**. Если буфер передатчика пуст (не было загружено новое значение), то передача данных завершается и флаг **BSY** сбрасывается в ноль.

Однако производители **рекомендуют проверять готовность при приеме и передаче, анализируя флаги TXE и RXNE, а не BSY [**17**].** Это связано с отличиями в формировании сигнала **BSY** в ведущем и ведомом устройствах.

Прием/передача информации по SPI реализуются на основании единого алгоритма. Дополнительно при приеме можно выполнить анализ ошибок.

В зависимости от особенностей решаемых задач возможны различные схемы соединения устройств, подключенных к интерфейсу SPI.

Дуплексный обмен данными с одним ведущим и одним ведомым иллюстрирует рисунок 1.46.

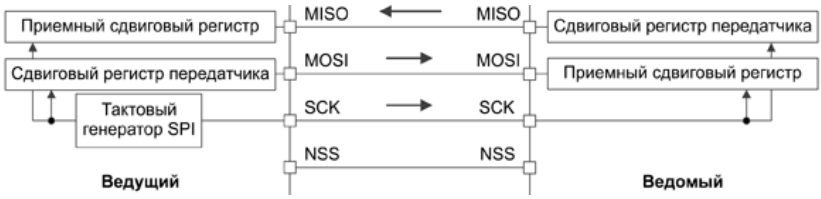



Рис. 1.46 – Схема подключения к SPI при дуплексном обмене

На рисунке начало обмена разрешает ведущее устройство сигналом NSS. В приведенной конфигурации линию NSS можно не использовать, а задать фиксированные сигналы (см. рисунок 1.44).

При реализации **симплексного** обмена можно использовать **одну из линий MOSI или MISO**.

Схемы соединения при различных режимах работы отражены в [54]. Схема соединения при большом числе ведомых устройств показана на рисунке 1.47.

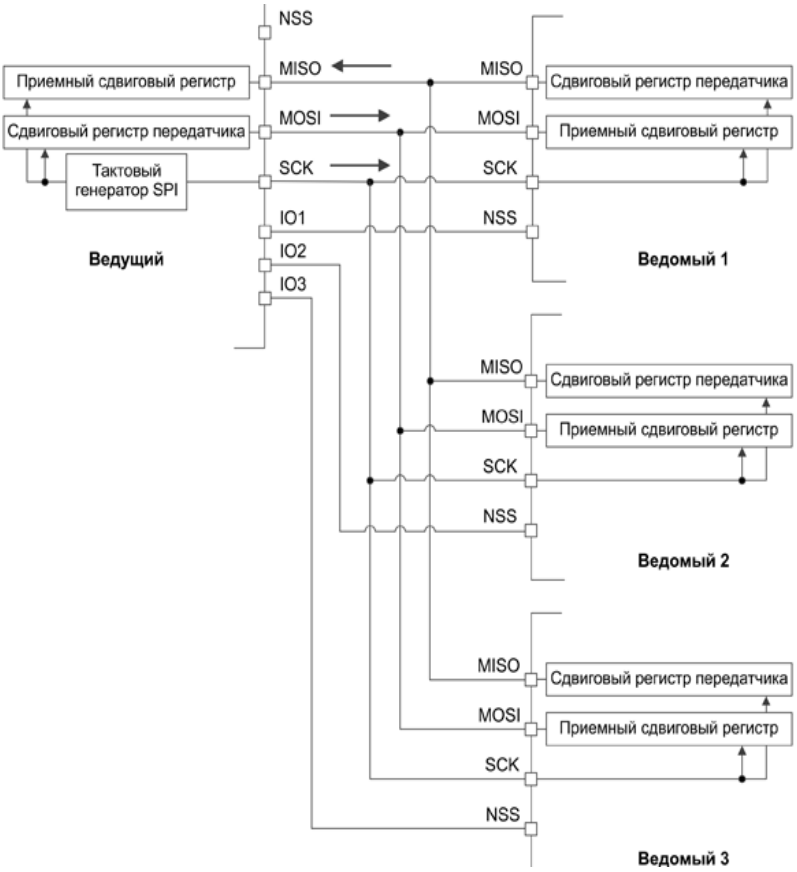



Рис. 1.47 – Схема подключения к SPI при обмене с несколькими ведомыми

В этом случае для выборки ведомых устройств используются свободные выводы параллельных портов GPIO.

Схема подключения при полудуплексном обмене представлена на рисунке 1.48.

Этот режим задается установкой бита **SPI_CR1.BIDIMODE**. Ведущее устройство генерирует SCK, и объединяются линии MOSI и MISO. Направление передачи (Input/Output) выбирается битом **SPI_CR1.BIDIOE**. Когда этот бит установлен в «1», линия данных становится выходом, иначе она работает как вход.

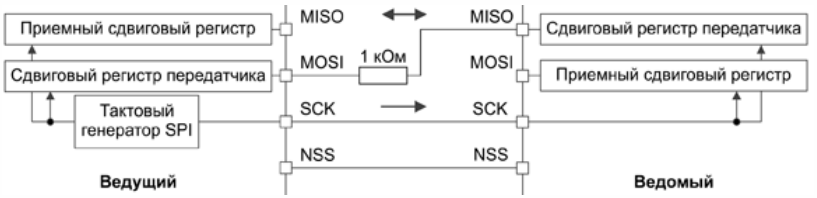



Рис. 1.48 – Схема подключения к SPI при полудуплексном обмене

Функции передатчика при подобном соединении может выполнять как ведущее, так и ведомое устройства. Для исключения аварийных ситуаций, когда оба устройства одновременно пытаются воспользоваться линией данных, MOSI и MISO соединяются через защитный резистор. Временная диаграмма полудуплексного режима представлена на рисунке 1.49.

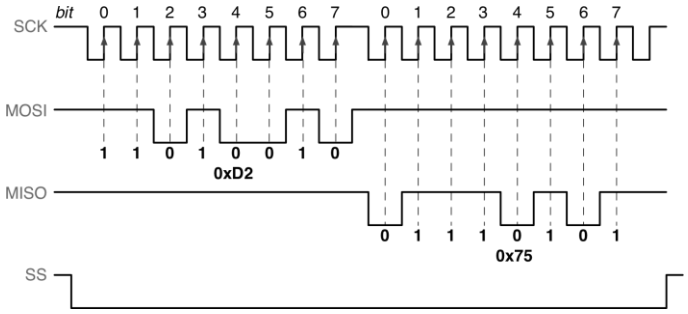



Рис. 1.49 – Временная диаграмма SPI при полудуплексном обмене

Возможна организация двунаправленного обмена между несколькими ведущими устройствами (рисунок 1.50). Изменение статуса устройства должно выполняться с помощью внешнего сигнала, формируемого на выходе GPIO.

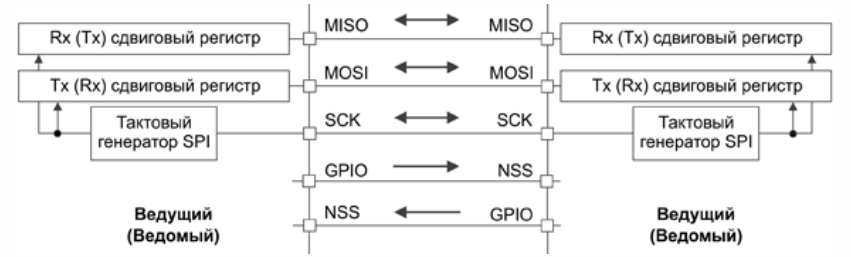



Рис. 1.50 – Обмен по SPI между двумя ведущими устройствами

Рекомендации по конфигурированию альтернативных функций приведены в таблице 1.17.

**Особенностью работы с CRC** является наличие двух полиномов для вычисления контрольной суммы. При приеме/передаче 8-разрядных данных используется CRC8, а 16-разрядных – CRC16. Тип полинома записывается в регистр SPI_CRCPR как в приемном, так и в передающем устройстве. **По умолчанию в регистре полинома записано 0x7 (x^8+x^2+x^1+1)**.

Вычисление CRC разрешается установкой бита CRCEN в регистре SPI_CR1. Это действие сбрасывает регистры вычисления CRC при приеме (SPI_RXCRCR) и передаче (SPI_TXCRCR).

При дуплексном обмене или в режиме только передачи необходимо установить бит SPI_CR1.CRCNEXT **во время записи последнего байта (или слова)** в SPI_DR. Тогда в процедуру передачи будет включено содержимое SPI_TXCRCR, которое вычисляется аппаратно.

В режиме приема бит CRCNEXT следует **установить после приема второй порции данных (или перед последней)**, чтобы подготовить SPI к входу в фазу контроля CRC в конце приема данных.

CRC, сформированная передатчиком, сравнивается со значением, вычисленным приемником, которое находится в регистре SPI_RXCRCR. Если значения не совпадают, то устанавливается флаг SPI_SR.CRCERR и будет вызвано прерывание при установленном бите ERRIE регистра SPI_CR2.

Таблица 1.17. Рекомендации по конфигурированию блока SPI

| Линии SPI | Режим работы | Конфигурирование GPIO |
|---|---|---|
| SPIx SCK | Ведущий | Альтернативная функция push-pull (двухтактный выход) |
| | Ведомый | Input floating (высокоимпедансный вход) |
| SPIx MOSI | Полный дуплекс/ведущий | Альтернативная функция push-pull |
| | Полный дуплекс/ведомый | Высокоимпедансный вход / вход с подтягивающим к питанию резистором (pull-up) |
| | Однонаправленная передача данных/ведущий | Альтернативная функция push-pull |
| | Однонаправленный прием данных/ведомый | Не используется. Можно использовать для других целей как линию GPIO |
| SPIx MISO | Полный дуплекс/ведущий | Input floating / Input pull-up |
| | Полный дуплекс/ведомый | Альтернативная функция push-pull |
| | Однонаправленный прием данных/ведущий | Не используется. Можно использовать для других целей как линию GPIO |
| | Однонаправленная передача данных/ведомый | Альтернативная функция push-pull |
| SPIx_NSS | Аппаратное переключение ведущий/ведомый | Высокоимпедансный вход / вход с подтягивающим к питанию резистором / вход с подтягивающим к земле резистором (Input pull-down) |
| | Аппаратный выход ведущего/выход NSS разрешен | Альтернативная функция push-pull |
| | Программное управление | Не используется. Можно использовать для других целей как линию GPIO |

Программная модель контроллера SPI приведена в приложении П1.8.

**Общий алгоритм работы блока:**

- Выбрать порт SPI и определить альтернативные линии ввода-вывода GPIO в соответствии с таблицей 1.16.

- Задать режимы работы линий ввода-вывода в соответствии с таблицей 1.17.
- Настроить схему синхронизации и включить тактирование выбранного порта.
- Задать требуемый коэффициент предделителя **SPI_CR1.BR[2:0]** (для ведущего порта).
- Задать полярность и фазу тактового сигнала разрядами **CPOL** и **CPHA** регистра **SPI_CR1**.
- Определить формат данных (8 или 16 бит) разрядом **SPI_CR1.DFF**.
- Задать порядок передачи данных (первым или последним битом вперед) разрядом **SPI_CR1.LSBFIRST**.
- Установить статус блока (ведущий/ведомый) битом **SPI_CR1.MSTR**.
- В ведомом SPI выбрать программное или аппаратное управление NSS (**SPI_CR1.SSM**) и, в зависимости от этого, подать на эту линию соответствующий уровень или задать значения в **SPI_CR1.SSI** и **SPI_CR2. SSOE**.
- Определить способ ввода/вывода.

При программном обмене контролировать флаги регистра SPI_SR в соответствии с логикой обмена.

Для обмена по прерыванию необходимо включить глобальное прерывание, разрешить прерывание от SPI и задать требуемый приоритет в NVIC, разрешить соответствующее прерывание в регистре **SPI_CR2**.

При вводе-выводе в режиме ПДП (DMA) необходимо настроить контроллер ПДП, включить его тактирование, установить бит ***SPI_CR2.TXDMAEN*** или ***SPI_CR2.RXDMAEN***.

- Разрешить работу битом **SPI_CR1.SPE**.
- Организовать прием/передачу данных в соответствии со схемой включения и требуемым алгоритмом.

**Ведомое устройство рекомендуется включать раньше ведущего, чтобы оно заранее было подготовлено к обмену.**

Особенностью полудуплексного режима является его настройка и управление битами **SPI_CR1.BIDIMODE**, **SPI_CR1.BIDIOE** и передача данных **младшими разрядами вперед**.

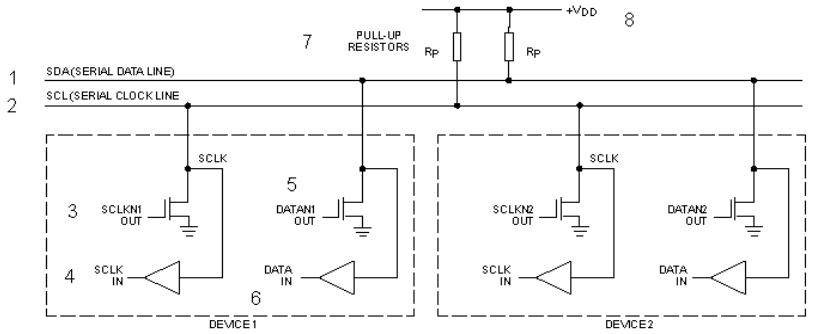
### 1.6.4 Последовательный порт $I^2C$

Область применения $I^2C$ аналогична SPI – в основном **внутриплатный** обмен между внешними периферийными устройствами различного назначения. Хотя на его основе были разработаны шины ACCESSbus (для подключения к персональному компьютеру типовой периферии), SMBus (для управления подсистемой питания, конфигурирования памяти и ряда других задач на системной плате ПЭВМ), PMBus (для регуляторов мощности и цифровых устройств управления питанием).

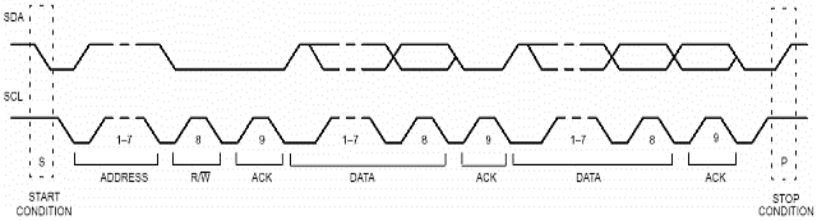
В отличие от SPI шина $I^2C$ 2-проводная полудуплексная с открытым стоком (коллектором), единицей обмена информации является 9 бит (8 бит данных и бит подтверждения приема АСК), число подключенных устройств определяется допустимой емкостной нагрузкой (не более 400 пФ), возможна работа с несколькими ведущими (режим мультимастер), обмен выполняется на основе стандартного кадра (рисунок 1.51) [50].

На рисунке 1.51 а) SDA – линия данных, SCL – линия синхронизации, SCLK – вход/выход синхронизации устройства, Data In, Data Out – соответственно вход/выход данных устройства, Rp – подтягивающий резистор.

На рисунке 1.51 б) START – стартовый бит, ADRESS – поле адреса, $R/\overline{W}$ (R/W*) – бит, определяющий направление передачи информации («1» – чтение, «0» – запись), $\overline{ACK}$ (ACK*) – бит подтверждения приема информации, DATA – данные, STOP – стоповый бит, * – признак инверсии.




а




б

Рис. 1.51 – Интерфейс I²C:
а – схема соединения устройств; б – временная диаграмма

**Блок I²C STM32F103xx присоединен к периферийной шине APB1 и обладает перечисленными ниже возможностями.**

- Может работать в режиме **Master** или **Slave**. В функции **ведущего** входит: формирование кадра и сигнала синхронизации, прием/передача данных, анализ сигнала АСК при передаче и формирование его при приеме, в случае необходимости формирование повторного старта, участие в процедуре арбитража. **Ведомое** устройство определяет начало и конец кадра, адрес (идентификатор) сообщения, направление передачи данных, выполняет запись/чтение данных, при приеме передает бит АСК, а при передаче – этот бит анализирует, при обмене имеет возможность вводить задержки. Перечисленные функции задаются при программировании. По умолчанию модуль находится в режиме

Slave. После генерации состояния START он автоматически переключается в режим Master.

- В **мультипроцессорном** режиме аппаратная реализация арбитража выполняется методом множественного доступа с контролем несущей и разрешением конфликтов (CSMA/CR) [50].
- Позволяет задавать частоту синхронизации – до 100 кГц (**standart mode**) и 400 кГц **(fast mode**). Более поздние модели STM32 обеспечивают работу на частотах 1 МГц **(fast mode plus**), 3.4 МГц **(high speed mode**) и выше.
- Программируемые частота синхронизации, скважность, длительность фронта и среза импульса.
- Программируемые длина поля адреса (7 или 10 бит), вид адресации (широковещательная 0x00 или адресная) и альтернативный адрес ведомого устройства.
- Ввод-вывод информации программно, по прерыванию, в режиме ПДП. Флаги событий формируются в регистрах состояния. Флаги 1÷13 находятся в **I2C_SR1**, а 14÷16 – в **I2C_SR2** (таблица 1.18).

Таблица 1.18. Флаги событий блока $I^2C$

| Событие | Флаг события | Маски прерываний |
|---|---|---|
| 1. Условие старта сформировано | SB | ITEVEN |
| 2. Передача/приема адреса | ADDR | |
| 3. Окончание приема/передачи байта | BTF | |
| 4. Передача старшей части 10-битового адреса | ADD10 | |
| 5. Получение STOP ведомым | STOPF | |
| 6. Прием данных завершен | RxNE | ITEVEN и ITBUFEN |
| 7. Буфер данных пуст | TxE | ITEVEN и ITBUFEN |
| 8. Ошибка шины | BERR | ITERREN |
| 9. Потеря прав ведущего при арбитраже | ARLO | |

*Окончание табл. 1.18*

| | | |
|---|---|---|
| 10. Отсутствие ACK | AF | |
| 11. Ошибка CRC | PECERR | |
| 12. Ошибка времени ожидания SCL | TIMEOUT | |
| 13. Режим работы | MSL | Только программно |
| 14. Шина занята | BUSY | |
| 15. Прием/передача устройства | TRA | |

При программном обмене флаги контролирует процессорный блок. **Для ввода/вывода по прерыванию предусмотрено два вектора**. Маски разрешения прерываний хранятся в **I2C_CR2**. Структура подсистемы прерывания представлена на рисунке 1.52.

В режим ПДП в контроллере резервируется буфер размером 1 байт. Включение режима ПДП выполняется битами **I2C_CR2.DMAEN** и **I2C_CR2.LAST**.

- Аппаратное формирование контрольной суммы CRC при передаче и проверка её при приеме.

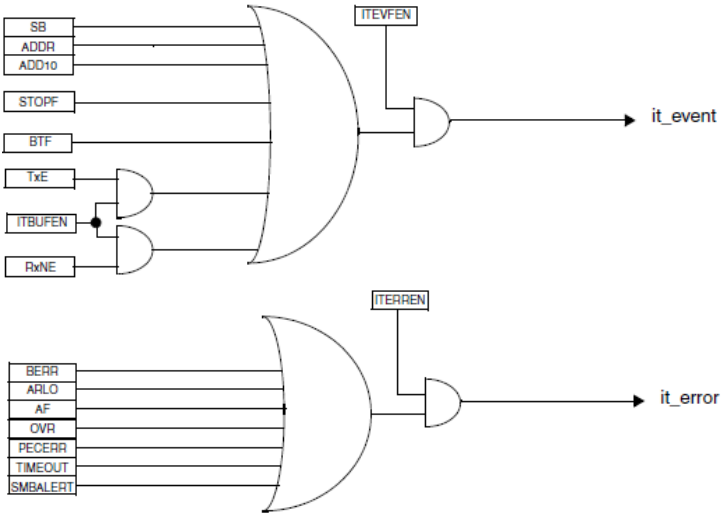



Рис. 1.52 – Структура подсистемы прерывания блока $I^2C$

Встроенный калькулятор CRC-PEC рассчитывает контрольную сумму по формуле C(x) = x8 + x2 + x + 1 (полином CRC-8). PEC – это CRC-8, рассчитанное для всех байтов сообщения, включая адреса и R/W биты. Расчет PEC разрешается установкой бита ENPEC в регистре I2C_CR1, а значение PEC сохраняется в регистре I2C_SR2.

**При передаче** установить бит I2C_CR1.PEC после события TxE, которое соответствует **последнему байту**. PEC будет передано вслед за последним байтом.

**При приеме** бит I2C_CR1.PEC следует **установить после события RxNE, которое соответствует последнему байту**, чтобы приемник послал NACK, если полученный байт не равен тому, что был получен в результате внутреннего расчета PEC.

В случае ведущего-приемника, NACK должен следовать за PEC, какой бы ни был результат проверки. Бит PEC должен ставиться до сигнала ACK при приеме текущего байта. В регистре I2C_SR1 также доступен флаг ошибки / прерывания PECERR.

Если есть разрешение на DMA и расчет PEC, то:

- **при передаче** после получения сигнала EOT (прерывание от события завершения обмена) от DMA контроллера PEC **автоматически пересылается после последнего байта**;
- при приеме, когда интерфейс $I^2C$ получает сигнал EOT от DMA контроллера, он автоматически будет рассматривать следующий байт как PEC, и будет проверять его. Запрос на DMA генерируется после получения PEC.

Чтобы разрешить передачу промежуточного PEC, существует управляющий бит в регистре I2C_CR2 (бит LAST), который определяет, действительно ли это последний обмен по DMA или нет. Если это последний запрос на DMA для ведущего приемника, то автоматически посылается NACK после получения последнего байта. Расчет PEC некорректен при потере прав на шину.

- Альтернативные линии ввода/вывода (Таблица 1.19).

Таблица 1.19. Ремапинг линий блока $I^2C$

| Альтернативные функции | I2C1_REMAP = 0 | I2C1_REMAP = 1 |
|---|---|---|
| I2C1_SCL | PB6 | PB8 |
| I2C1_SDA | PB7 | PB9 |

- Работа в режимах **$I^2C$** и **SMBus 2.0** (System Management Bus – шина системного управления).

Шина SMBus базируется в основном на процедурах обмена $I^2C$, но поддерживает идеологию физического, канального и сетевого уровней открытых систем OSI (Open Systems Interconnection). В этой связи имеются отличия как на физическом уровне (например, параметры сигналов, ограничения по частоте, энергопотребление), так и в протоколах обмена [82]. Структура блока $I^2C$ приведена на рисунке 1.53.

На стадии инициализации определяются основные параметры блока в соответствии со структурой кадра и режимом работы. Эти параметры записываются в регистры управления **I2C_CR1**, **I2C_CR2**, а адрес ведомого – в регистр основного адреса **I2C_OAR1 (Own address)** и/или альтернативного **– I2C_OAR2 (Dual address)**. В ведомом устройстве выполняется сравнение адреса, заданного в кадре, с содержимым регистра адреса с помощью компаратора (**Comparator**). В процессе приема/передачи может быть активирована функция формирования и анализа контрольной суммы CRC (**PEC calculation**) битом **I2C_CR1.ENPEC**. Значение CRC находится в регистре **I2C_SR2.PEC(7÷0)**.

Управление синхронизацией выполняет схема **Clock control**. В ведущем устройстве SCL является выходом, а в ведомом – входом. Параметры частоты синхронизации определяются значением, записанным в **I2C_CR2.(5÷0**), и содержимым регистров **I2C_CCR** (частота и скважность), **I2C_TRISE** (длительность фронта и среза).

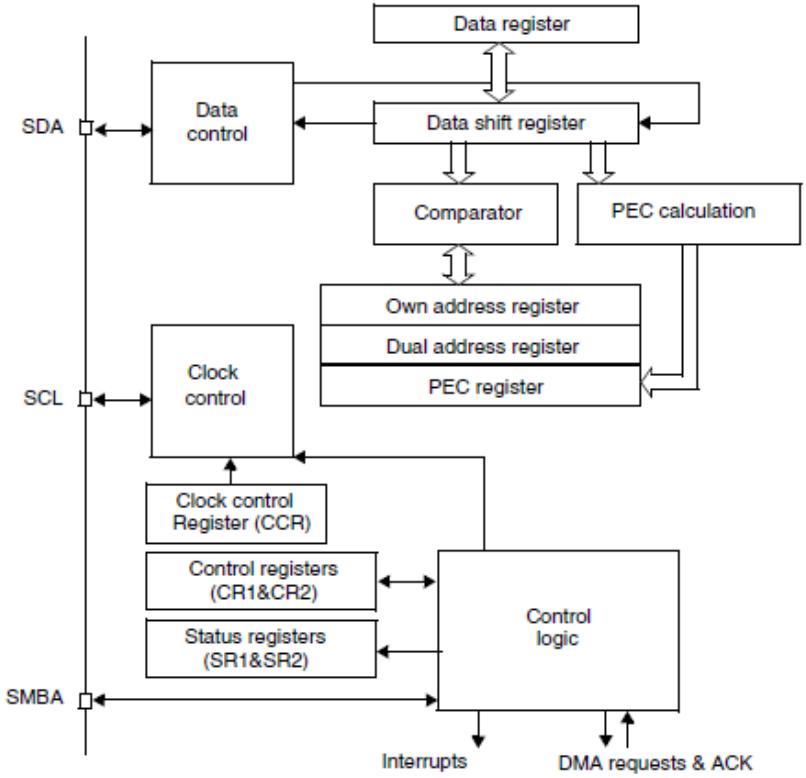



Рис. 1.53 – Структура блока $I^2C$

**В зависимости от режима обмена частота синхронизации APB1 должна быть не менее 2 МГц для стандартного режима и не менее 4 МГц для быстрого (fast mode). Выбор этого значения должен быть согласован с другими периферийными устройствами, подключенными к APB1.** Значение частоты **APB1** устанавливается соответствующими делителями системы синхронизации.

**Далее это значение записывается в регистр I2C_CR2.(5÷0) в виде целого числа в двоичном формате. Например, если частота APB1 равна 12 МГц, то 001100, 36 МГц – 100100**.

Значение частоты **SCL** определяется как 1/(Th + Tl), где Th – длительность высокого уровня, Tl – длительность низкого уровня.

Для стандартного режима скважность постоянная Th = CCR*$T_{pclk1}$; Tl = CCR*$T_{pclk1}$, где CCR – значение, заданное в регистре I2C_CCR.CCR(11÷0), $T_{pclk1}$ – частота APB1.

В быстром режиме скважность определяется I2C_CCR.DUTI. Поэтому при DUTI = 0 Th = CCR*$T_{pclk1}$, Tl = 2*CCR*$T_{pclk1}$; при DUTI = 1 Th = 9*CCR*Tpclk1; Tl = 16*CCR*Tpclk1.

Например, в **стандартном** режиме для частоты $I^2C$, равной 100 КГц (период 10 мкс, Th = Tl = 5мкс), и частоте PCLK1, равной 12 МГц, **значение CCR** = 5мкс*Fpclk = 60 (0x3C). Длительность фронта определяется как **(Tr max / TPCLK1) + 1**, где **Tr max** для стандартного режима равен **1000 нс**, а для быстрого – **300 нс, TPCLK1 – период PCLK1.** Вычисленное значение записывается в регистр **I2C_TRISE**. Для приведенного примера **TPCLK1 =** 1/12000000 = 83 нс, К = 1000/83 = 12 + 1. TRISE(5÷0) = 13 = = 0xD [36].

Начало обмена задает ведущее устройство. После обнаружения начала кадра ведомым устройством (Start) входные данные по линии **SDA** поступают в регистр сдвига **Data shift register**. В момент окончания приема байта выполняется аппаратная запись в регистр данных **Data register**. При этом формируется АСК и устанавливается флаг **I2C_SR1.RxNE**.

Если адрес не совпал, контроллер порта переходит в состояние ожидания до появления следующего стартового бита.

Для поддержания непрерывного потока при приеме необходимо прочитать из регистра данных **I2C_DR** до момента окончания приема очередного байта. При передаче контролируется состояние регистра данных и, если он свободен (**I2C_SR1.TxE = 1**), в этот регистр выполняется запись очередного байта. Основные флаги событий приведены выше (таблица 1.18).

При приеме данных возможен контроль ошибок. Количество байт в кадре не регламентируется. Однако, чем длиннее кадр, тем больше время доступа к интерфейсу. Поэтому в системах реального времени не рекомендуется использовать длинные кадры. Прием/передача завершается после обнаружения бита Stop.

В некоторых случаях возможно многократное повторение Start до формирования Stop-условия. Этот прием рекомендуется использовать: при изменении направления передачи к ведомому устройству (запись/чтение), при обращении по другому адресу в течение текущего кадра, для синхронизации нового пакета данных и так далее. Повторный старт увеличивает пропускную способность $I^2C$, исключая процедуру арбитража.

Основные действия по формированию и контролю кадра выполняет схема **Control logic.**

При подключении внешних устройств к порту $I^2C$ **рекомендуется использовать внешние подтягивающие резисторы** даже при наличии внутренних. Номиналы внешних резисторов должны быть (4,7÷10) кОм для обеспечения большей скорости передачи данных (значение сопротивления внутренних подтягивающих резисторов – около 20 кОм). Поэтому **линии GPIO, предназначенные для $I^2C$, должны быть сконфигурированы с открытым стоком** и к ним подключены внешние подтягивающие резисторы.

При проектировании модулей с несколькими устройствами $I^2C$ **следует обратить внимание на их адресацию**. Некоторые устройства имеют фиксированный адрес, заданный при изготовлении. В других устройствах (например, памяти) имеется возможность модификации трех младших разрядов при заданном базовом адресе. **При конфликте адресов может потребоваться использование дополнительного порта $I^2C$.**

Необходимо контролировать функциональные особенности внешних устройств (ведущий/ведомый или только ведомый).

Порт может работать в следующих режимах:

- ведущий передатчик;
- ведущий приемник;
- ведомый передатчик;
- ведомый приемник.

Алгоритмы работы режимов приведены в [17, рис. 270÷276]. Программная модель блока I2C приведена в приложении П1.9.

**Общий алгоритм управления портом состоит в следующем:**

- определить используемые линии GPIO (таблица 1.19);
- настроить альтернативные функции линий на выход с открытым стоком и частотой, соответствующей выбранному режиму (стандартный или быстрый);
- включить тактирование портов GPIO и порта $I^2C$;
- выполнить конфигурацию порта $I^2C$:

**для ведущего** – установить режим $I^2C$, разрешить или запретить формирование **АСК,** контрольной суммы **CRC**, широковещательной передачи (**I2C_CR1),** задать частоту синхронизации модуля и длительность фронта (**I2C_CR2, I2C_CCR**, **I2C_TRISE**);

**для ведомого** – установить режим $I^2C$, разрешить или запретить формирование АСК, контрольной суммы CRC, широковещательной передачи, задержку синхронизации, альтернативного адреса (**I2C_CR1**), задать длину поля адреса, основной и альтернативный адрес (**I2C_OAR1**, **I2C_OAR2**).

Определить способ ввода/вывода, задав соответствующие разряды в регистре **I2C_CR2**. Выполнить при необходимости процедуру инициализации подсистемы прерывания и/или канала ПДП.

Дальнейшее изложение относится к программному способу ввода/вывода, так как при любом способе необходимо контролировать флаги состояния и ошибок.

Включить порт (**I2C_CR1.PE**).

При работе, например, в **режиме ведущего передатчика** необходимо проверить флаг занятости шины (**I2C_CR2.BUSY**).

Сформировать стартовый бит (**I2C_CR1.START**).

Проверить установку стартового бита (**I2C_SR1.SB = 1**).

Сбросить его (сброс выполняется аппаратно, если после чтения **I2C_SR1** выполняется запись **I2C_DR**).

Если **I2C_SR1.TxE = 1**, записать в регистр данных **I2C_DR** адрес устройства (7 или 10 бит). **Последний бит поля адреса задает направление передачи** («0» – запись, «1» – чтение). **Признаком 10-разрядной адресации** является код 11110, расположенный в старших разрядах старшего байта.

Проверить флаг передачи адреса (**I2C_SR1.ADDR = 1**) и после проверки его сбросить (чтение **I2C_SR1 и I2C_SR2**). При передаче 10-разрядного адреса проверяется флаг посылки старшего байта (**I2C_SR1.ADD10 = 1**), который следует сбросить чтением **I2C_SR1** и записью в **I2C_DR** младшего байта адреса.

Далее выполняется запись всех данных аналогичным образом: проверка **TxE** перед записью, сброс этого флага, контроль окончания передачи байта (**I2C_SR1.BTF = 1**), очистка его (чтение **I2C_SR1** и запись в **I2C_DR**).

При разрешенном CRC (**I2C_CR1.PEC = 1**) последний байт будет содержать контрольную сумму.

Завершается кадр формированием стопового бита (**I2C_SR1.STOP = 1**).

На всех этапах при разрешенном АСК формирование сигнала подтверждения выполняется аппаратно, а равенство нулю соответствующего флага состояния означает отсутствие АСК. Отсутствие сигнала АСК можно проверить контролем бита **I2C_SR1.AF**.

При работе **в режиме ведомого приемника** необходимо контролировать состояние Start (**I2C_SR1.SB = 1**).

Принять адресный байт (или байты). Окончание записи можно проверить в бите **I2C_SR1.BTF**.

После окончания записи выполняется **аппаратное сравнение** принятого байта с адресом, записанным в регистре **I2C_OAR1**. Если адрес не совпал, порт переходит в состояние ожидания до прихода следующего состояния **Start**.

При совпадении адресов устанавливается бит **I2C_SR1.SB**, будет сформирован сигнал **АСК**, а флаг **TRA = 0** (режим ведомого приемника).

Для того, чтобы порт откликался на альтернативный адрес, следует в регистр **I2C_OAR2** записать значение адреса и установить разряд **ENDUAL**. При совпадении адресов будет установлен разряд **DUALF**.

Если порт настроен на широковещательную передачу (**I2C_CR1.ENGC = 1**), то при получении адреса 0x00 будет установлен бит **I2C_SR2. GENCALL**.

Проверить значение бита (**I2C_SR1.ADDR = 1**) и после проверки его сбросить (чтение **I2C_SR1** и **I2C_SR2**).

Проверить готовность приема данных **I2C_SR1.RxEN** и, если он установлен, выполнить чтение регистра данных. Процедуру чтения и последующей записи в память следует продолжать до конца кадра.

Если по какой-то причине регистр данных не был прочитан, то устанавливается флаг **I2C_SR1.OVR**.

При обнаружении **STOP** будет установлен бит **I2C_SR1.STOPF**, который затем надо сбросить. Для сброса нужно прочитать регистр **I2C_SR1** и произвести запись в **I2C_CR1**. Например, выполнить операцию **I2C_CR1.STOP = 0**.

При приеме имеется возможность контролировать и другие ошибки (таблица 1.18).

### 1.6.5 Последовательный порт USART

Блок USART (универсальный синхронно-асинхронный приёмопередатчик) используется для организации обмена информацией по последовательным интерфейсам, протоколы которых соответствуют формату кадра USART (RS-232) или содержат его в качестве базового элемента (RS-422, интерфейсы на основе RS-485, некоторые профили Profibus и другие). Наиболее часто электрическое согласование с интерфейсами требует применения внешних преобразователей уровня. Для связи с интерфейсами, требующими дополнительно преобразования форматов кадра (USB, $I^2C$, SPI, CANbus и так далее), используются внешние мосты.

Блоки USART STM32 по сравнению, например, с ATmega фирмы Atmel, обладают расширенными функциональными возможностями [62, 41].

**Основные характеристики USART:**

- Режимы обмена: синхронный, асинхронный.
- Направление обмена: дуплексный, симплексный, полудуплексный.
- Кодирование данных: NRZ (без возврата к нулю).
- Программируемый генератор скорости передачи данных – до 4.5 Мбит/с.
- В асинхронном режиме программируемая длина кадра:
  - количество бит данных (8 или 9);
  - контроль четности (нет, четность, нечетность);
  - количество стоп-битов (0.5, 1, 1.5, 2).
- Обработка прерываний от 10 источников запроса, **но с одним вектором.**
- Буферизация и передача данных в режиме ПДП (DMA).

- Мультипроцессорный режим работы с различными источниками пробуждения из «спящего» режима.

Особенности **USART:**

- программно управляемая частота опроса значения бита (умножение частоты передачи на 16 или 8, что позволяет при 8-кратном опросе увеличить вдвое скорость передачи). **В МК STM32F103xx доступно только умножение на 16;**
- задание коэффициента деления частоты передачи USART в виде числа, содержащего целую и дробную части. В этом случае возможна настройка скорости передачи с минимальной ошибкой **практически при любых значениях кварцевого резонатора**;
- аппаратные средства для согласования работы устройств, отличающихся по быстродействию (**линии управления модемом RTS, CTS**);
- **асинхронная однопроводная полудуплексная связь;**
- **возможность использования интерфейсов:**
  - **IrDA SIR кодек** для инфракрасной связи (поддерживается длительность бита 3/16 в нормальном режиме);
  - **LIN,** наиболее часто использующегося на нижнем уровне бортовых систем автомобиля для управления медленными исполнительными устройствами [13, 81];
  - **Smartcard,** который поддерживает стандарт ISO 7816-3 для работы с смарт-картами, «симками» и так далее [55, 30].

В зависимости от конкретной модели STM32 может быть до трех модулей USART (USART1, USART2, USART3) и до двух модулей UART (UART4, UART5) Функциональные особенности модулей отражены в таблице 1.20.

Таблица 1.20. Режимы работы блоков USART STM32

| Режим работы USART | USART1 | USART2 | USART3 | UART4 | UART5 |
|---|---|---|---|---|---|
| Асинхронный | X | X | X | X | X |
| Аппаратное управление потоком | X | X | X | NA | NA |
| Режим прямого доступа к памяти (DMA) | X | X | X | X | NA |
| Мультипроцессорный | X | X | X | X | X |
| Синхронный | X | X | X | NA | NA |
| Smartcard | X | X | X | NA | NA |
| Полудуплексный однопроводный режим | X | X | X | X | X |
| IrDA | X | X | X | X | X |
| LIN | X | X | X | X | X |

В состав микроконтроллера оценочного модуля входят USART1, USART2, USART3. **При настройке портов необходимо учитывать, что USART1 присоединен к APB2 (до 72 МГц), а USART2, USART3 – APB1 (до 36 МГц).** Структура блока USART приведена на рисунке 1.54. Принцип работы блока в синхронном и асинхронном режимах приведен во многих источниках (например, в [50]). На рисунке 1.54 используются следующие обозначения:

**TX** – выход передатчика.

**RX** – вход приемника.

**SW** – линия полудуплексного обмена, которая может быть использована для организации локальной сети.

**CK** – выход тактового сигнала при синхронной передаче данных.

**RTS** – готовность устройства к приему данных (выход). Если на выходе устанавливается значение лог. «0», то устройство готово принимать данные, при значении лог. «1» – передача должна быть остановлена.

**CTS** – разрешение передачи (вход). Если на CTS низкий уровень, то передача выполняется, при высоком уровне – передача останавливается после завершения кадра.

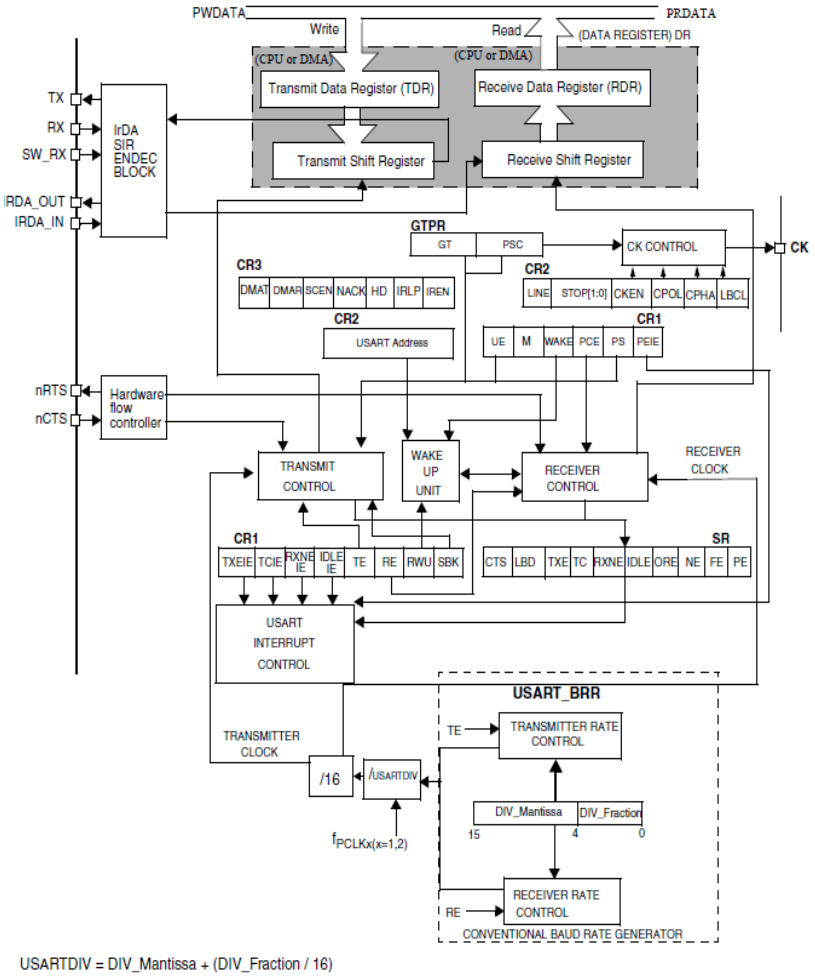



Рис. 1.54 – Структура блока USART

Сигналы RTS, CTS (сигналы управления модемом) обеспечивают аппаратное управление потоком данных. Если приемник информации не успевает обрабатывать входные данные, то на выходе RTS сформируется сигнал (RTS = 1), который, поступая на выход CTS, прервет передачу и возобновит её при RTS = 0 (рисунок 1.55).

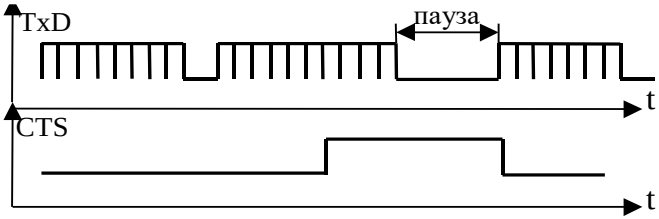
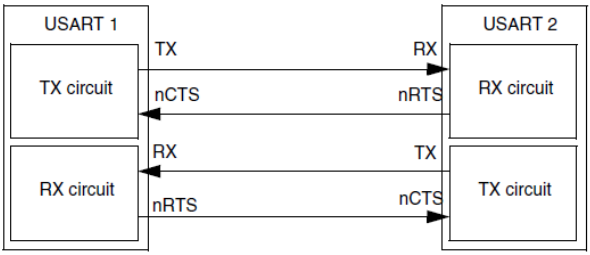



Рисунок. 1.55 – Временная диаграмма управления потоком данных USART

Дуплексную связь с использованием сигналов управления модемом иллюстрирует рисунок 1.56.




RTS and CTS flow control can be enabled independently by writing respectively RTSE and CTSE bits to 1 (in the USART_CR3 register).

Рис. 1.56 – Дуплексная связь с сигналами управления модемом

**DR** – виртуальный регистр данных. Физически регистр данных состоит из двух пар, работающих на прием (RDR и приемный сдвиговый регистр) и передачу (TDR и передающий сдвиговый регистр). Однако при программировании обращение происходит к регистру DR, который при записи переадресует информацию в TDR, а при чтении – содержимое RDR будет прочитано из DR.

**USART_BRR** – блок делителя частоты с регистром скорости передачи USART_BRR.

**CR1-CR3 –** регистры управления.

**GTPR** – регистр защитного интервала и предделитель, используемый в режимах IrDA и Smartcard.

**SR** – регистр состояния.

Внешние выводы USART должны быть соответствующим образом сконфигурированы с помощью регистра конфигураций **GPIOx_CRL**, **CRH**. Рекомендации по настройке приведены в таблице 1.21.

Таблица 1.21. Режимы работы линий ввода/вывода блока USART

| Линии USART | Режим работы | Конфигурирование GPIO |
|---|---|---|
| USARTx TX | Полный дуплекс | Альтернативная функция push-pull |
| | Полудуплексный синхронный | Альтернативная функция push-pull |
| USARTx RX | Полный дуплекс | Высокоимпедансный вход / вход с подтягивающим к питанию резистором (pull-up) |
| | Полудуплексный синхронный | Не используется. Может использоваться как обычный ввод-вывод GPIO |
| USARTx_CK | Синхронный | Альтернативная функция push-pull |
| USARTx_RTS | Аппаратное управление потоком | Альтернативная функция push-pull |
| USARTx_CTS | Аппаратное управление потоком | Высокоимпедансный вход / вход с подтягивающим к питанию резистором (pull-up) |

В зависимости от реализуемых функций микроконтроллера может возникнуть необходимость перекоммутации линий ввода/вывода USART на альтернативные линии GPIO (ремапинг). Это выполняется с помощью регистров настройки AFIO [17].

В таблице 1.22 указано подключение линий USART по умолчанию и возможности назначения линиям альтернативных функций (ремапинг). Возможности ремапинга ограничиваются характеристиками корпуса МК. **В используемом МК отсутствуют в полном объеме порты PD, PC**.

По умолчанию **тактирование APB1** и **APB2 выключено** с целью снижения энергопотребления. Поэтому для работы USART необходимо разрешить тактирование в соответствующем регистре **RCC_APBnEN**.

Таблица 1.22. Ремапинг линий USART

| USART1 | | | |
|---|---|---|---|
| Альтернативные функции | USART1_REMAP = 0 | USART1_REMAP = 1 | |
| USART1_TX | PA9 | PB6 | |
| USART1_RX | PA10 | PB7 | |
| USART2 | | | |
| Альтернативные функции | USART2_REMAP = 0 | USART2_REMAP = 1(1) | |
| USART2_CTS | PA0 | PD3 | |
| USART2_RTS | PA1 | PD4 | |
| USART2_TX | PA2 | PD5 | |
| USART2_RX | PA3 | PD6 | |
| USART2_CK | PA4 | PD7 | |
| USART3 | | | |
| Альтернативные функции | USART3_REMAP P[1:0] = “00” (нет ремапинга) | USART3_REMAP P[1:0] = “01” (частичный ремапинг) | USART3_REMAP P[1:0] = “11” (полный ремапинг) |
| USART3_TX | PB10 | PC10 | PD8 |
| USART3_RX | PB11 | PC11 | PD9 |
| USART3_CK | PB12 | PC12 | PD10 |
| USART3_CTS | PB13 | | PD11 |
| USART3_RTS | PB14 | | PD12 |

Программная модель блока USART приведена в приложении П1.10

**Алгоритм настойки USART в дуплексном режиме:**

1. Выбрать в качестве источника тактирования **внешний кварцевый резонатор**, так как в асинхронном режиме отличие в частотах приемника и передатчика должно быть не более определенной величины (в зависимости от скорости передачи 0,5÷1 %). **Внутренние частотно-задающие элементы могут не обеспечить требуемую стабильность.**

2. Выбрать свободный блок USART (1, 2 или 3). При выборе блока необходимо контролировать возможное использование линий USART другими периферийными устройствами. В случае

необходимости переключить линии в соответствии с таблицей 1.22 с помощью регистра **AFIO_MARP**.

3. Определить требуемый периферийный интерфейс (APB1 или APB2) и включить синхронизацию в соответствующем регистре **RCC_APBnEN**.

4. Задать рекомендуемые в таблице 1.21 режимы линий ввода-вывода в регистре **GPIOx_CRL** или **GPIOx_CRH.** В зависимости от особенностей внешнего устройства необходимо выбрать тип выхода обычный или толерантный.

5. Выбрать требуемый формат кадра: определить требуемое количество стоповых бит **(USART_CR2.STOP[1:0])**, выбрать вид паритета (или его отсутствие) и включить его контроль (**USART_CR1.PS, PCE**), указать длину поля данных (**USART_CR1.M**), выбрать частоту опроса бит (оверсэмплинг) (**USART_CR1.OVER8**). **В STM32F103 этот бит отсутствует**. При использовании бита паритета необходимо учитывать, что он располагается последним в поле данных. Поэтому при передаче байта поле данных должно иметь длину 9 бит.

6. Настроить скорость обмена. При работе с внешними устройствами рекомендуется использовать стандартный ряд скоростей 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600 бит/с. Для внутренних целей можно использовать любую требуемую скорость. Следует отметить: чем ниже качество линии связи, тем меньшую скорость передачи необходимо выбирать. Иначе, количество потерянных посылок будет увеличиваться (отбрасываться по таймауту), а информация будет передаваться не вся.

Скорость обмена определяется частотой работы выбранной периферийной шины и коэффициентом деления, записанным в регистре **USART_BRR**. Регистр скорости передачи **USART_BRR** состоит из 2 частей: **DIV_Mantissa** и **DIV_Fraction**.

Значение USART_BRR вычисляется следующим образом **BRR = (BUS_FREQ/BAUD)/16**, где BUS_FREQ – частота периферийной шины; BAUD – скорость передачи бит/с; 16 – количество опросов для определения значения бита (значение бита определяется на основании правила «2 из 3» в середине 16-битной последовательности).

Анализ начала кадра и его завершения позволяет обнаружить наличие помех в момент формирования стартового бита – **USART_SR.NE** и стоповых бит – **USART_SR.FE** (рисунок 1.57).

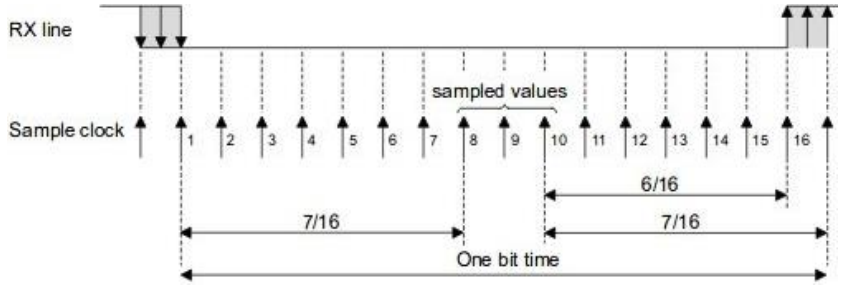



Рис. 1.57 – Принцип определения значения бита USART

Предположим, что частота периферийной шины 36 МГц, а требуемая скорость 19200 бит/с. Тогда

(36 000 000 / 19200) /16 = 117.18.

Целая часть коэффициента деления – 117, а дробную – надо восстановить, умножив на 16 и округлив до ближайшего целого:

0.18 * 16 = 2.88 -> 3

После преобразования каждой части в 16-ричную систему USART_BRR = 0x00000753.

Для расчета погрешности необходимо оценить степень приближения; 3 / 16 = 0.1875.

Тогда погрешность (117.18–117.1875) / 0.0075 = 0.0064%. Значения погрешностей для стандартных скоростей приведены в таблице 1.23.

7. Выбрать схему соединения USART с внешним устройством и, в случае необходимости, инициализировать работу с сигналами управления модемом RTS, CTS (**USART_CR3.CTSE**, **USART_CR3.RTSE**).

Таблица 1.23. Оценка погрешностей расчета скорости передачи USART

| Скорость | | fpclk = 36 MHz | | | fpclk = 72 MHz | | |
|---|---|---|---|---|---|---|---|
| № | Кбит/с | Фактическая | Содержимое регистра скорости обмена | Погрешность % | Фактическая | Содержимое регистра скорости обмена | Погрешность % |
| 1 | 2.4 | 2.4 | 937.5 | 0% | 2.4 | 1875 | 0% |
| 2 | 9.6 | 9.6 | 234.375 | 0% | 9.6 | 468.75 | 0% |
| 3 | 19.2 | 19.2 | 117.1875 | 0% | 19.2 | 234.375 | 0% |
| 4 | 57.6 | 57.6 | 39.0625 | 0% | 57.6 | 78.125 | 0.00% |
| 5 | 115.2 | 115.384 | 19.5 | 0.15% | 115.2 | 39.0625 | 0% |
| 6 | 230.4 | 230.769 | 9.75 | 0.16% | 230.769 | 19.5 | 0.16% |
| 7 | 460.8 | 461.538 | 4.875 | 0.16% | 461.538 | 9.75 | 0.16% |
| 8 | 921.6 | 923.076 | 2.4375 | 0.16% | 923.076 | 4.875 | 0.16% |
| 9 | 2250 | 2250 | 1 | 0% | 2250 | 2 | 0% |
| 10 | 4500 | NA | NA | NA | 4500 | 1 | 0% |

8. Включить USART, установив требуемые биты в регистре **USART_CR1 (UE, TE, RE, RWU)**.

9. **Выбрать способ ввода-вывода информации.**

**В программном режиме** выполняется анализ флагов в регистре **USART_SR**. Если **TXE** установлен, выполняется **запись** в регистр **USART_DR**, после которой **TXE аппаратно сбрасывается, а установится этот флаг после передачи TDR в регистр сдвига, при завершении передачи установится флаг TC.**

При **чтении** информации проверяется флаг готовности **RX**, при необходимости флаги ошибок **PE**, **FE**, **NF**, **ORE** и выполняется чтение **USART_DR**.

Для снижения энергопотребления возможен **переход в спящий режим (USART_CR1.RWU)**. Если **сброшен** бит **USART_CR1.WAKE**, то переход в активный режим будет выполнен при **USART_SR.IDLE = 1**.

**При вводе-выводе по прерыванию** необходимо:

- разрешить прерывание от нужных событий, учитывая, что вектор прерываний один, а источников – много. Флаги событий приведены в таблице 1.24, а на рисунке 1.58 показано формирование запросов прерывания. Разрешение прерываний устанавливается в соответствующих битах регистров **USART_CR1**, **USART_CR3**;
- настроить в регистрах **NVIC_IPRx** приоритет прерывания от USART;
- в регистрах **NVIC_ISERx** включить требуемое прерывание.

Таблица 1.24. Флаги событий блока USART

| Событие прерывания | Флаг события | Флаг разрешения прерываний |
|---|---|---|
| Transmit data register empty | TXE | TXEIE |
| CTS flag | CTS | CTSIE |
| Transmission complete | TC | TCIE |
| Received data ready to be read | RXNE | RXNEIE |
| Overrun error detected | ORE | |
| Idle line detected | IDLE | IDLEIE |
| Parity error | PE | PEIE |
| Break flag | LBD | LBDIE |
| Noise flag, Overrun error and Framing error in multibuffer communication | NE or ORE or FE | EIE |

**TXE –** регистр передачи данных пуст.

**CTS –** изменение состояния линии CTS. Сигнализирует о готовности приёмника к обмену данными.

**TC –** завершение передачи. Бит устанавливается автоматически по окончанию передачи данных и **TXE = 1.**

**RXNE –** полученные данные доступны для чтения.

**ORE** – обнаружение ошибки переполнения входного буфера.

**IDLE** – обнаружение свободной линии (состояние ожидания).

**PE** – ошибка чётности.

**LBD** – флаг обрыва. Используется в протоколе LIN.

**NE** – флаг обнаружения шумов.

**FE** – ошибка кадра.

Более подробное описание флагов приведено в приложении 1.

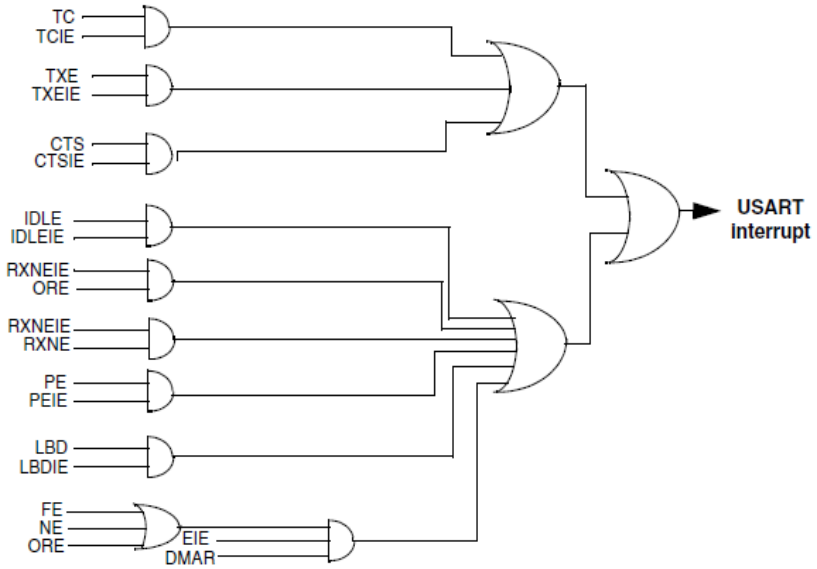



Рис. 1.58 – Формирование запросов прерывания блока USART

**При вводе-выводе в режиме прямого доступа** необходимо разрешить этот режим в регистре **USART_CR3 (биты DMAR,**

**DMAT)** и настроить блок ПДП. **Алгоритм работы с контроллером ПДП описан в разделе 1.4** [28].

**В полудуплексном режиме** данные передаются и принимаются **по одной линии** (**TX**) в режиме с разделением времени (линия RX и TX соединяются внутри микроконтроллера). В каждый момент одно из устройств является передатчиком, а другое – приемником. Поэтому **необходимо следить за направлением передачи и перестраивать в нужный момент линии ввода-вывода.** В остальном управление происходит как в обычном асинхронном режиме, включая использование прерывания и ПДП. Полудуплексный режим включается в регистре **USART_CR3.HDSEL**. При этом необходимо запретить работу в остальных режимах. Для этого следует сбросить биты **LINEN** и **CLKEN** в регистре **USART_CR2**, **SCEN** и **IREN** в регистре **USART_CR3**.

**Линию TX необходимо соединить через подтягивающий резистор (≈10кОм) к плюсу питания.**

**Мультипроцессорный режим** позволяет подключать несколько устройств к одной асинхронной шине в режиме «Master-Slave» (ведущий-ведомый).

Инициатором обмена является ведущее устройство, которое передает по шине информацию с девятибитовым полем данных, старший бит которого является идентификатором типа данных («1» – адрес, «0» – данные).

В исходном состоянии ведомые устройства следует настроить в 9-битовый режим (**USART_CR1.M**), отключить контроль паритета (**USART_CR1.PCE**), задать адреса (**USART_CR2.ADD[3:0]**), включить выход из спящего режима по значению адреса (**USART_CR1.WAKE**), включить спящий режим (**USART_CR1.RWU**). Остальные настройки как обычно (подключение, количество стоповых бит, скорость, способы ввода/вывода).

Ведущее устройство должно быть в активном режиме с теми же настройками, что и у ведомого (кроме работы с адресами).

В начале обмена **ведущее** устройство передает адресный байт, который анализируют все **ведомые** устройства. Устройство, у которого **адрес совпал** с заданным при программировании, переходит в активный режим и принимает **данные, следующие** за этим адресом. Остальные ведомые находятся в спящем режиме и аппаратно проверяют старший бит каждого байта. Если это данные, то кадр игнорируется, если адрес, то производится сравнение с заданным при программировании. Далее процесс повторяется под управлением ведущего. Новое устройство переходит в активный режим, а остальные – в спящий до появления соответствующего адреса. Этот процесс иллюстрирует рисунок 1.59.

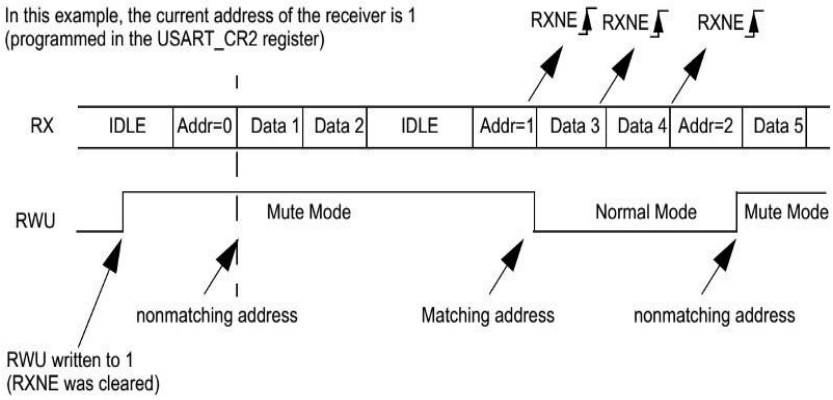



Рис. 1.59 – Процесс обмена информацией в мультипроцессорном режиме USART

На рисунке показана реакция **ведомого устройства** с адресом «1» на информацию, передаваемую по шине. При передаче данных устройствам с адресами «0» и «2» оно находится в спящем режиме (Mute Mode). При обнаружении своего адреса переходит в активный режим приема (Normal Mode) и записывает два байта (Data3, Data4).

При организации сети **выход Tx ведущего** соединяется со **входами Rx ведомых, а выходы Tx ведомых** со **входом Rx ведущего**.

При соединении одного ведущего и одного ведомого рекомендуется включить подтягивающий резистор к плюсу питания. Остальные приёмопередатчики становятся ведомыми устройствами, у которых соответствующие выходы TX объединены логической операцией конъюнкции (AND) и соединены со входом RX ведущего.

К однопроводной линии можно подключить несколько устройств (рисунок 1.60), которые будут образовывать сеть для передачи данных. Арбитраж в этой сети должен быть реализован программно.

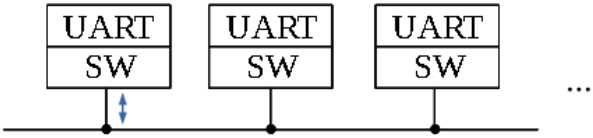



Рис. 1.60 – Однопроводное подключение модулей USART

Особенностью реализации **синхронного режима** является его совместимость с интерфейсом SPI, в котором **USART может быть только ведущим устройством**. Для его инициализации необходимо разрешить формирование сигнала синхронизации (**USART_CR2.CLKEN**), сбросить USART_CR2.LINEN и SCEN, HDSEL и IREN в регистре USART_CR3 задать длину кадра (**USART_CR1.M**), согласовать полярность и фазу тактового сигнала (**USART_CR2.CPOL**, **USART_CR2.CPHA**), разрешить или запретить формирование последнего импульса (**USART_CR2.LBCL**).

Временная диаграмма и схема соединения в этом режиме представлена на рисунке 1.61. Реализация ввода-вывода выполняется аналогично предыдущему.

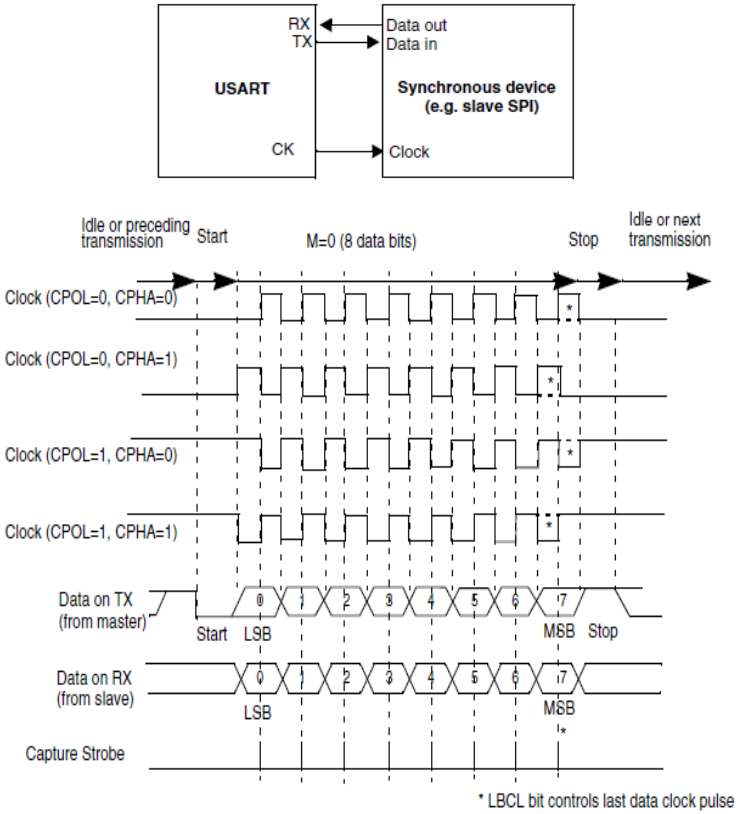



Рис. 1.61 – Схема соединения и временная диаграмма USART в режиме SPI

Пример программирования USART с помощью библиотеки HAL и код генератора CUBE MX приведен в [44].

### 1.6.6 Аналого-цифровой преобразователь

Аналого-цифровой преобразователь STM32 реализует принцип поразрядного уравновешивания (последовательного приближения) с использованием устройства выборки-хранения УВХ [50].

Количество АЦП в STM32 может быть от 1 до 3. **В микроконтроллере 32F103C8T6 используется ADC1.**

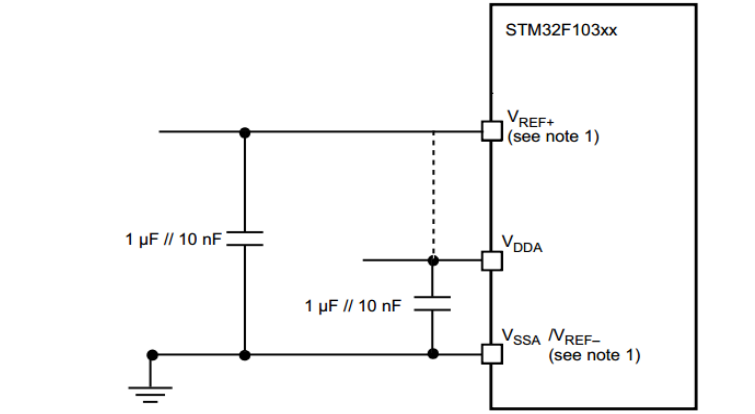
Основные характеристики АЦП:

- **Разрядность –** 12 бит. Этот параметр характеризует разрешающую способность. Но одним из основных параметров измерения является погрешность, которая для различных типов АЦП STM32 составляет величину от 2 до 5 младших разрядов, то есть около (0,05÷0,1) %.
- **Минимальное время преобразования** – 1 мкс. Основное время преобразования АЦП составляет To = 12.5 цикла, но для **каждого** канала время **Ti** можно задать программно (3 бита) в пределах, определяемых регистрами **ADC_SMPR1**, **ADC_SMPR2** (1,5÷239,5 циклов). Таким образом, **время преобразования канала определяется как T = To + Ti**.

Чтобы включить тактирование модуля ADCx нужно установить бит **ADCx** в регистре **RCC_APB2ENR** и задать значение предделителя (2, 4, 6, 8) битами **ADCPRE[1:0]** регистра **RCC_CFGR**. **Частота тактирования модулей ADC не должна превышать 14 МГц**. Минимальное время преобразования – 14 тактов, что на частоте 14МГц обеспечивает 1мкс. Из приведенных соотношений рассчитывается частота, поступающая с шины APB2 на АЦП. **Механизм тактирования позволяет включать в цикл опроса источники сигналов с различным быстродействием, если оно соответствует указанным диапазонам.**

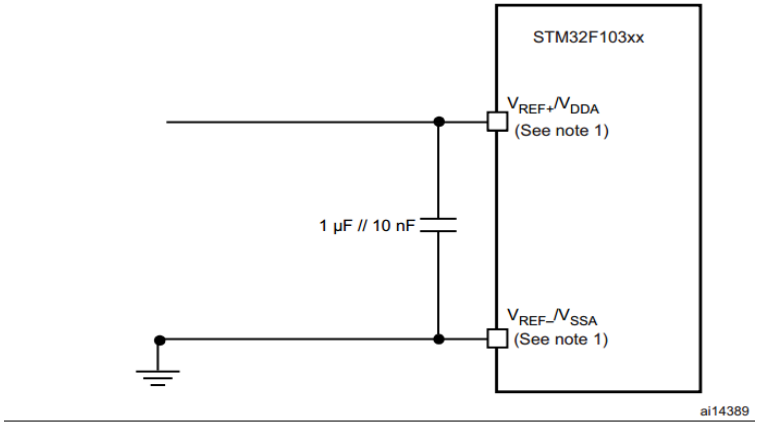
- **Диапазон входных сигналов по напряжению** определяется величиной опорного (эталонного) напряжения $U_{ref}$, которое может быть как внешним, так и внутренним. Диапазон $U_{ref}$ от 2,4 В до 3,6 В в зависимости от напряжения питания микроконтроллера. Лучшие метрологические характеристики обеспечивает внешний источник опорного напряжения (ИОН), но возможность его подключения зависит от модели STM322 и количества выводов корпуса. Схема подключения внешнего ИОН представлена на рисунке 1.62.

**В STM32F103C8T6 такая возможность отсутствует**. В качестве внутреннего ИОН обычно используется Vdda. Поэтому аналоговый источник питания должен обладать повышенными метрологическими характеристиками. Рекомендуемая схема включения приведена на рисунке 1.63.




$V_{REF+}$ and $V_{REF-}$ inputs are available only on 100-pin packages.

Рис. 1.62 – Схема подключения внешнего ИОН




$V_{REF+}$ and $V_{REF-}$ inputs are available only on 100-pin packages.

Рис. 1.63 – Схема включения внутреннего ИОН

При пониженных метрологических требованиях можно использовать источник питания микроконтроллера, но с дополнительными фильтрами. Значение преобразуемого напряжения $V_{in} = ((V_{ref+} - V_{ref-})/4096)*Ni$, где Ni – код, пропорциональный $V_{in}$, записанный в регистр данных АЦП. В STM32F103C8T6 дифференциальные каналы отсутствуют. Поэтому $V_{ref-}$ обычно соединяют с **аналоговой землей $V_{ssa}$.** Если диапазон измерения значительно меньше $V_{ref}$ или больше, то **следует использовать нормализаторы** сигналов (усилители или делители).

- **Количество каналов** – до 16 внешних и 2 внутренних (ИОН и температурный датчик). **Для STM32F103C8T6 – 10 внешних каналов.** При инициализации АЦП внешние линии GPIO необходимо сконфигурировать как аналоговые входы.

Каналы функционально разделены на две группы. **Регулярные каналы** опрашиваются и преобразуются с определённой периодичностью, а результат помещается **в общий 12-разрядный регистр ADC_DR**, т.е **до очередного** измерения результат **должен быть переписан** в ОЗУ или другой регистр. Количество **регулярных каналов –** от 1 до 16.

Особенностью **инжектированных каналов** является их более высокий приоритет по сравнению с регулярными. После их запуска **специальными сигналами** (см. **ADC_CR2**) приостанавливается работа регулярных каналов, выполняется измерение заданных инжектированных каналов, а затем восстанавливается измерение каналов регулярной группы. **Максимальное количество инжектированных каналов в группе – 4.**

**Каждый** инжектированный канал содержит регистры данных **ADC_JDRx** и смещения (компенсации) **ADC_JOFRx**, где x – номер канала. Значение регистра смещения **аппаратно** вычитается из регистра данных, а результат записывается в регистр данных.

Регистр смещения целесообразно использовать для вычитания **констант**, связанных с процессом измерения. Например, значение аддитивной составляющей при использовании токовой петли (4÷20) мА и так далее.

Инжектированные каналы используются в том случае, если в процессе функционирования регулярных каналов возникает необходимость приоритетного опроса какого-либо канала, проверка ИОН или температуры кристалла.

Если во время работы инжектированного канала происходит событие запуска регулярных каналов, то оно не прерывает его работу, а запуск регулярной последовательности выполняется сразу после окончания обработки инжекционной последовательности.

- **Способ запуска** каналов – программный, от таймеров или внешних источников (см. рисунок 1.65). Программирование источников запуска для регулярных и инжектированных каналов различное и определяется соответствующими разрядами регистра **ADC_CR2**. **При использовании внешнего запуска необходимо контролировать, чтобы интервал между событиями был больше, чем время преобразования каналов.**
- Регистры данных **ADC_DR** и **ADC_JDRx** могут быть выровнены по правой границе (данные записываются в разряды 11:0) или по левой (данные записываются в разряды 15:4). В остальные разряды записываются нули. Выравнивание определяется битом **ADC_CR2.ALIGN**.
- **Опрос каналов** может быть выполнен в режимах: сканирования, одиночного, непрерывного и прерывистого преобразований.

В **режиме одиночного преобразования** выполняется **однократное измерение канала**. Запуск выполняется программно установкой бита **ADC_CR2.ADON** (для регулярного канала) или аппаратно внешним запуском (для обоих типов каналов), который

программируется в том же регистре **ADC_CR2**. Разряд **ADC_CR2.CONT** должен быть равен нулю. Результат преобразования сохранится в соответствующем регистре данных **ADC_DR** или **ADC_JDR**, установятся флаги в регистре состояния **ADC_SR** и может быть вызвано прерывание, если установлены маски в регистре управления **ADC_CR1**.

В режиме **непрерывного преобразования (Continious Conversion Mode**) очередное преобразование начинается после завершения предыдущего и будет продолжаться до тех пор, пока ADC не будет выключен. Этот режим запускается от внешнего источника или путём установки разряда **ADON** регистра **ADC_CR2**. При этом разряд **CONT** регистра **ADC_CR2** должен быть равен единице. **При внешнем запуске необходимо установить соответствующие режимы в битах ADC_CR2 (15:10).**

**Внешний запуск** регулярных каналов разрешается установкой бита **EXTTRIG** регистра **ADC_CR2**, а тип события настраивается битами **EXTSEL[2:0]** регистра **ADC_CR2**. Источниками внешнего запуска могут быть сигналы определенных каналов таймеров или запросы внешнего прерывания. Запуск также может быть сформирован программно установкой бита **SWSTART** регистра **ADC_CR2**.

Аналогично запускаются инжектированные каналы: разрешение – установкой бита **JEXTTRIG**, тип события – битами **JEXTSEL[2:0]**, программный старт – **JSWSTART.** Эти биты находятся в регистре **ADC_CR2**.

Для непрерывного запуска инжектированных каналов следует **установить ADC_CR1.JAUTO**.

**Использовать одновременно оба режима, автоматический запуск и запуск внешним событием, для инжектированных каналов невозможно.**

Преобразование завершается как в предыдущем случае. Однако следует заботиться о сохранении регистра **ADC_DR** программно, по прерыванию или в режиме ПДП. Большее быстродействие обеспечивает применение ПДП.

В **режиме сканирования (Scan Conversion Mode)** выполняется поочерёдное преобразование группы каналов, заданных в регистрах **ADC_SQRx** для регулярных каналов или в регистре **ADC_JSQR** для инжектированных каналов Группа формируется в регистрах **ADC_SQR1 … ADC_SQR3**, в разряды **SQx[4:0]** которых записывается **последовательность номеров опрашиваемых каналов**, где **x** – номер позиции в последовательности. Количество измерений в группе определяет поле **L[3:0]** регистра **ADC_SQR1** и может достигать 16. Например, для регулярного опроса 5 каналов в последовательности 5, 1, 9, 1, 6 необходимо занести в регистр **ADC_SQR3** следующие значения: SQ1[4:0] = 5, SQ2[4:0] = 1, SQ3[4:0] = 9, SQ4[4:0] = 1, SQ5[4:0] = 6, а в регистр **ADC_SQR1.L[3:0]** записать число 5. Таким образом, можно создать **любую последовательность опроса каналов.**

Этот режим выбирается установкой разряда **SCAN** регистра **ADC_CR1**. При запуске инжектированных каналов необходимо **сбросить ADC_CR1.JAUTO**. Если разряд **CONT** установлен, то преобразование не останавливается на последнем канале, а вновь запускается с первого канала.

Организация групп и последовательность опроса выполняется аналогично регулярным каналам с помощью регистра **ADC_JSQR**.

В **прерывистом режиме** (**discontinuous mode**) имеется возможность разбить группы на подгруппы и выполнять их последовательный опрос.

Для регулярных и инжектированных каналов этот режим задаётся в регистре ADC_CR1 битами DISCEN и JDISCEN. Размер подгруппы для регулярных каналов задаётся битами

DISCNUM[2:0] регистра ADC_CR1, а для инжектированных он равен 1 (последовательный опрос).

Если приведенный выше список разделить на подгруппы, состоящие из трех каналов, то первый внешний запуск вызовет опрос 5, 1, 9 каналов, а второй – 1, 6 канала. Третий внешний запуск начнет цикл опроса сначала, если установлен бит **CONT**.

Для инжекционных каналов приход каждого импульса внешнего запуска вызовет опрос 1, 2, 3 каналов.

- **Способы ввода-вывода** – программный, по прерыванию, в режиме ПДП (DMA). Особенностью работы с прерываниями является **один вектор и три запроса** (по окончанию регулярного и инжектированного преобразований, по сигналу оконного компаратора). Маски прерываний находятся в регистре **ADC_CR1**. Запуск режима ПДП – **ADC_CR2**. **DMA. Запросы ПДП могут генерировать только ADC1 и ADC3.**

- АЦП имеет встроенный механизм **автоматической калибровки** аддитивной составляющей погрешности (смещение нуля), вызванной неоднородностью внутренних конденсаторов выборки и хранения сигналов. **Во время калибровки** вычисляется цифровое значение АЦП **для каждого конденсатора** в виде корректирующего кода. Во всех последующих преобразованиях этот код используется для компенсации погрешности. Калибровку рекомендуется проводить каждый раз после включения питания, но при этом **АЦП должен находиться в отключённом состоянии (ADON = 0), как минимум, в течение двух циклов работы АЦП.**

- Для запуска процедуры калибровки необходимо очистить регистры калибровки, установив в единицу бит **RSTCAL** регистра **ADC_CR2**, проверить окончание процедуры сброса (бит **RSTCAL** аппаратно сбросится в ноль), запустить калибровку, установив в единицу бит **CAL** в регистре **ADC_CR2**, проверить её окончание (бит **CAL** аппаратно сбросится в ноль). После завершения

калибровки коды калибровки сохраняются в **ADC_DR**. Запустить работу АЦП.

- Наличие **оконного компаратора (AWD),** входящего в состав АЦП, позволяет **аппаратно** (без привлечения ресурсов процессора) реализовать процедуру сравнения **Nmin ≤ Ni ≤ Nmax. (**рисунок 1.64).

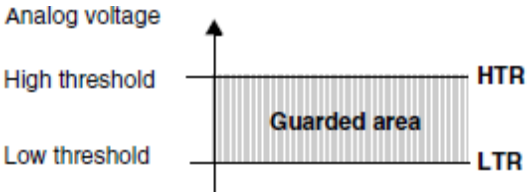



Рис. 1.64 – Границы оконного компаратора

Оконный компаратор может оказаться полезным для автоматического контроля определённого параметра. Например, допусковый контроль, который при отсутствии компаратора приходится выполнять программно, контроль датчика температуры (защита от перегрева микроконтроллера) и так далее.

Он имеет в своём составе два регистра для задания границ и цифровую схему сравнения. В регистр нижней границы **ADC_LTR** и регистр верхней границы **ADC_HTR** записываются соответствующие значения для контроля измеряемого сигнала **Ni**. При выходе за заданные границы компаратора автоматически устанавливается бит **AWD** в регистре **ADC_SR**, который можно контролировать программно или по прерыванию, если установлен бит **ADC_CR1.AWDIE**.

Компаратор можно подключить к одному или нескольким каналам. Каналы, к которым подключен компаратор, задаются в регистре **ADC_CR1** биты **AWDSGL** (выбор канала), **AWDEN** (подключение к регулярным каналам), **JAWDEN** (подключение к инжектированным каналам), **AWDCH0÷AWDCH4** (номер канала). Использование этих бит иллюстрирует таблица 1.25.

Таблица 1.25. Управление оконным компаратором

| Типы каналов | Биты регистра ADC_CR1 (x = безразлично) | | |
|---|---|---|---|
| | AWDSGL | AWDEN | JAWDEN |
| нет | x | 0 | 0 |
| Все инжектированные | 0 | 0 | 1 |
| Все регулярные | 0 | 1 | 0 |
| Все регулярные и инжектированные | 0 | 1 | 1 |
| Один инжектированный | 1 | 0 | 1 |
| Один регулярный | 1 | 1 | 0 |
| Один регулярный или инжектированный | 1 | 1 | 1 |

- Для измерения температуры микроконтроллера предназначен **датчик температуры АЦП**. Датчик целесообразно использовать **для контроля изменения** температуры, а **не для измерения абсолютных значений**, так как он обладает низкими метрологическими свойствами**.** Характеристика датчика – линейная, но в различных микроконтроллерах она может смещаться до 45 градусов, что связано с технологией изготовления. **З**начение температуры T[°C] = {(V25 – Vsense)/Avg_ Slope} + 25, где V25 – значение измерения при 25, имеющее типовое значение 1.41 В, Vsense – текущее измеренное значение, Avg_Slope – коэффициент из таблицы на кристалл, имеющий типовое значение 4.3 мВ/°С.

Датчик подключён к 16-му входу АЦП. Рекомендуемое время выборки при опросе датчика составляет 17,1 мкс. Когда датчик не используется, его можно отключить.

Для определения температуры необходимо:

1) выбрать 16-й канал;

2) задать время выборки не менее 17.1 мкс;

3) включить датчик, установив разряд **TSVREFE** в регистре **ADC_CR2**;

4) запустить преобразование АЦП установкой разряда **ADON** или внешним событием, прочитать результат преобразования;

5) преобразовать его в напряжение и на основании приведенной формулы рассчитать температуру.

Установка **TSVREFE** позволяет также **прочитать значение внутреннего ИОН, который присоединен к 17 каналу** [33].

- В МК, содержащих несколько АЦП, предусмотрены режимы каскадного соединения, при которых один АЦП (ведущий) может запускать другие, АЦП могут работать синхронно (или с временной задержкой) по опросу одноименных каналов и т.д. [53]. Структура блока АЦП приведена на рисунке 1.65 [23].

Входные сигналы поступают на входы **GPIO**, которые следует настроить как аналоговые. С выхода аналогового мультиплексора **AnalogMUX** сигнал подается на вход АЦП соответствующего канала. Частота синхронизации **ADCCLK** поступает с выхода предделителя.

При использовании внешнего запуска требуемый источник с помощью мультиплексоров задается программно для регулярных каналов (**EXTSEL[2:0]**) и инжектированных (**JEXTSEL[2:0**]).

Результаты измерений записываются в регистры инжектированных каналов (4 регистра) и регистр регулярных каналов.

Флаги готовности хранятся в регистре состояния и могут быть использованы для программного ввода-вывода, по прерыванию (**ADC Interrupt NVIC)**, в режиме ПДП (**DMA request**).

Оконный компаратор **Analog watchdog** содержит два регистра и схему формирования флага. Программная модель блока АЦП приведена в приложении П1.11

Особенности программирования с использованием библиотеки HAL и код генератора CUBE MX показаны в [2].

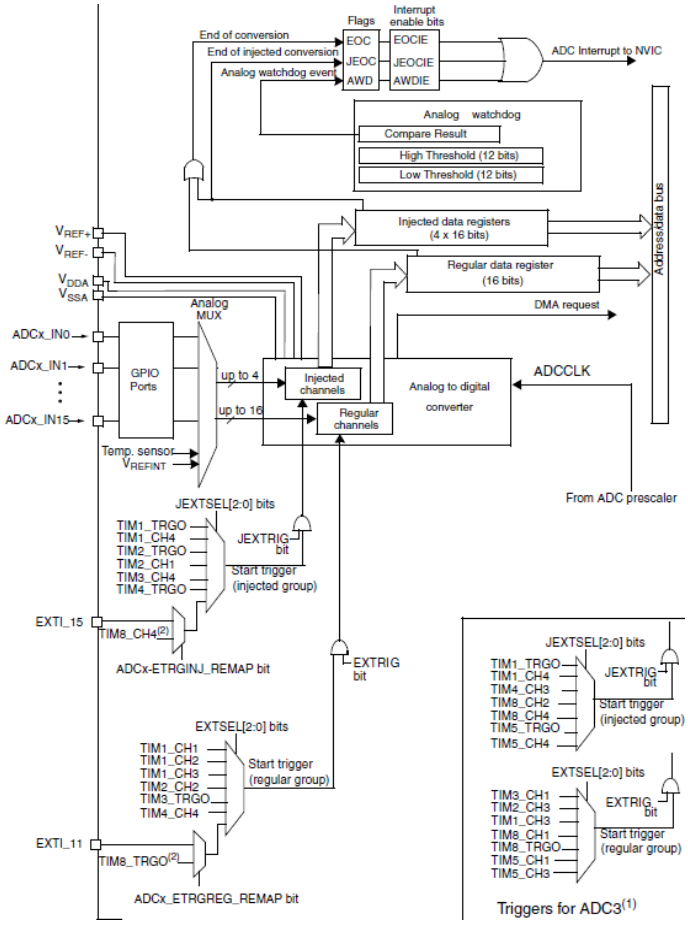



Рис. 1.65 – Структура блока АЦП

Общий алгоритм работы с АЦП состоит в следующем:

- определить линии GPIO, к которым подключены источники напряжения;
- настроить их как аналоговые входы, **GPIOx_CRL**, **GPIOx_CRH**;
- включить тактирование порта **RCC_APB2ENR** и предделителя **RCC_CFGR**;

- определить время преобразования (**ADC_ SMR1**, **ADC_ SMR2**);
- определить тип канала (регулярный или инжектированный) и способ запуска (внешний или программный) **ADC_CR2**;
- задать выравнивание результатов преобразования (вправо/влево) **ADC_CR2**;
- выбрать тип опроса (сканирование, одиночное преобразование, непрерывное, прерывистое) и подготовить соответствующие регистры: **ADC_CR1**, **ADC_CR2** и т.д.;
- определить количество каналов;
- при необходимости включить оконный компаратор (**ADC_CR1**, **ADC_HTR**, **ADC_LTR**);
- при использовании каналов измерения температур и ИОН включить их в опрос **ADC_CR2**;
- запустить калибровку **ADC_CR2**;
- включить АЦП **ADC_CR2**;
- в соответствии со способом ввода-вывода выполнить обработку, контролируя регистр состояния **ADC_SR**, маски в регистре **ADC_CR1**, регистры подсистем прерываний и ПДП;
- организовать сохранение результатов преобразования [22, 63].

## 1.7 Контрольные вопросы

### По разделу 1 Архитектура STM32F103C8T6

1. Анализ основных характеристик STM32F103C8T6
2. В чем особенности реализации Гарвардской архитектуры в STM32F103?
3. Какие проблемы возникают при работе с фрагментированными данными и как они решаются в МК?
4. Какие преимущества обеспечивает наличие битового пространства и как оно реализовано в STM32F103xx?

5. В чем особенности организации обращения к флеш-памяти? Как она реализована в STM32F103? **(см. «Структурную организацию» стр. 163)**

6. В чем особенности организации конвеера? Характеристики конвеера STM32F103.

7. В чем достоинства использования унифицированных блоков, структур, программных моделей? Как они реализованы в МК?

8. Какие характеристики МК улучшает аппаратная реализация базовых функций периферийных блоков?

9. Какие преимущества обеспечивает иерархическая организация магистралей? Как они реализованы в МК?

10. В чем особенность системы синхронизации STM32F103? На какие характеристики STM32F103 они влияют?

11. Какие решения в STM32F103 способствуют более простой компоновки печатных плат?

12. Какие решения в STM32F103 способствуют снижению энергопотребления?

13. Какие методы используются в STM32F103 для повышения быстродействия?

14. Какие задачи решает матрица шин? Как обеспечивается арбитраж матрицы шин?

15. В чем отличие в назначении периферийных шин APB1 и APB2? Какие преимущества это обеспечивает?

16. В чем преимущество трехадресных команд? Как они реализованы в STM32F103?

17. Какие команды появились в STM32F103 по сравнению с AVR? «См. [50]»

18. В чем особенности системы команд **Thumb-2?**

19. Какие преимущества обеспечивает наличие расширенного регистрового файла? Какие регистры (по назначению) используются в STM32F103?

20. Для каких целей используется R14? В чем преимущество по сравнению с AVR?

21. Какие особенности использования счетчика адреса команд в STM32F103 по сравнению с AVR?

22. В чем особенности регистра состояния STM32F103 по сравнению с AVR?

23. В чем отличия режимов потока и обработчика?

24. Какие преимущества обеспечивает наличие портов отладки? В чем отличие в организации **JTAG и SWB?** (см. [50], стр. 149–152)

### По разделу 1.2 Контроллер прерываний

1. В каких случаях целесообразно использование ввода-вывода по прерыванию?

2. Какие преимущества обеспечивает использование стандартизованного блока прерываний NVIC?

3. Какие основные характеристики блока прерываний STM32F103xx?

4. В чем особенность перехода к подпрограмме обработки прерываний STM32F103xx по сравнению с AVR?

5. Какие виды прерываний обрабатывает контроллер NVIC?

6. Как вызвать программное прерывание NVIC? В каком регистре?

7. Что такое исключение и какие исключения обрабатывает контроллер NVIC?

8. Какие события формируют запросы прерываний на входе NVIC? Сколько векторов выделяется на периферийное устройство?

9. Какие события формируют запросы прерываний на входе контроллера внешних прерываний EXTI? Почему важно контролировать перечисленные внутренние события?

10. В чем особенность работы с внешними прерываниями STM32F103xx по сравнению с AVR?

11. Какие требования предъявляются к внешнему сигналу запроса прерывания EXTI?

12. Какие ограничения существуют на количество внешних запросов прерываний EXTI? Состав линий.

13. Какие отличия в обработке внешних прерываний, подключаемых к линиям 0-4, 5-9, 10-15 портов?

14. Какие входы запросов внешних прерываний следует использовать для обеспечения максимального быстродействия? Почему?

15. Сколько **векторов прерываний** предназначено для обработки внешних запросов со входов порта?

16. Какие ограничения существуют при выборе линий порта для подключения внешних запросов? (см. GPIO)

17. Как задать линии портов, предназначенных для подключения внешних запросов? (см. GPIO)

18. Какой порт настраивается на прием внешних запросов прерывания при включении или сбросе?

19. Как задать требуемый приоритет запроса прерывания? В каком регистре?

20. Как разрешить/запретить прерывания? В каких регистрах?

21. Какое стандартное время доступа к подпрограмме обработки прерывания у STM32?

22. Какие методы используются для уменьшения времени реакции на запрос прерываний?

23. В чем особенность сохранения/восстановления контекста при вызове подпрограммы обработки прерываний в STM32F103xx? В чем отличие от AVR?

24. Какие методы используются для снижения времени обработки прерываний в STM32F103xx?

25. Как реализуется доступ к подпрограмме обработки после определения запроса с максимальным приоритетом?

26. В чем особенности обработки вложенных прерываний STM32F103xx по сравнению с AVR?

27. Как проверить состояние подпрограмм обработки прерываний?

28. В чем недостатки подсистемы прерываний STM32F103xx? В чем достоинства и недостатки подсистемы прерываний STM32F103xx по сравнению с AVR?

29. Как настроить в CUBE MX работу с внутренними прерываниями? внешними? Просмотрите диалог.

30. Какие функции HAL предназначены для обработки прерываний? Приведите примеры.

### По разделу 1.3 Системный таймер

1. Для каких целей предназначен системный таймер **SysTick**?
2. Почему не рекомендуется использовать системный таймер в программах пользователя?
3. Как задать частоту переполнения **SysTick?**
4. Как проверить переполнение счетчика **SysTick**
5. Для каких целей используется калибровка **SysTick** и как она реализуется?
6. Как контролируется текущее значение счетчика **SysTick**?
7. В чем особенность обращения к регистрам **SysTick?**

### По разделу 1.4 Контроллер прямого доступа к памяти

1. В каких случаях следует использовать режим ПДП?
2. В чем преимущества режима ПДП перед программным вводом/выводом? по прерыванию?
3. Технические характеристики контроллера ПДП (КПДП) STM32F103C8T6.
4. Какие факторы необходимо учитывать при оценке быстродействия в режиме ПДП?

5. Какие преимущества обеспечивает применение КПДП в структуре МК?

6. Какие функции выполняет контроллер прямого доступа к памяти КПДП?

7. На основании какой информации распределяются ресурсы КПДП между периферийными устройствами?

8. Как задать приоритет между каналами КПДП?

9. Как обеспечивается согласование форматов, передаваемых данных и памяти КПДП?

10. В каких случаях следует использовать режим цикличности КПДП? В чем его особенность?

11. В чем особенность обработки прерываний КПДП?

12. Как оценить быстродействие КПДП? Сравнить с вводом-выводом по прерыванию и программным режимом?

13. Как выполнить инициализацию канала КПДП?

14. Как реализуется обмен между периферийным устройством и КПДП?

15. Какие настройки необходимо выполнить в периферийном устройстве и контроллере ПДП для реализации обмена?

16. В чем особенности режима «память-память» КПДП? Как этот режим реализуется?

17. Какие особенности в распределении ресурсов матрицы шин в STM32?

18. Как настроить в CUBE MX работу с КПДП?

19. Какие функции HAL предназначены для работы с КПДП?

### По разделу 1.5 Блок синхронизации

1. В чем особенности блока синхронизации STM32 по сравнению с 8-разрядными МК, например, AVR?

2. В каких случаях следует использовать HSI? HSE? LSI? LSE?

3. Как контролировать корректность работы HSI?

4. Как обеспечивается синхронизация с внешними устройствами? Какими средствами?

5. Какие устройства подключены к APB1? APB2?

6. Как обеспечить минимальное энергопотребление МК средствами блока синхронизации?

7. Какие регистры обеспечивают управление блоком синхронизации?

8. Как задать частоту и включить тактирование устройства в CUBE MX?

### По разделам 1.6 Периферийные устройства Cortex M3 и 1.6.1 Параллельные порты

1. В чем преимущества периферийных устройств STM32F103 по сравнению с 8-разрядными МК?

2. Какие ограничения характерны для STM32F103C8T6?

### По разделу 1.6.1 Параллельные порты

3. Основные характеристики параллельных портов STM32F103C8T6.

4. Структурная организация порта. Дать рекомендации по использованию различных схем конфигурации.

5. В каких случаях следует использовать двухтактный выход (**Push-pull**)? Выход с открытым стоком (**Open-drain)?**

6. С какой целью используют входы с подтягивающими резисторами к питанию (**Input pull-up)** или к земле (**Input pull-down)?**

7. Какие преимущества обеспечивает применение высокоимпедансного входа (**Input floating)?**

8. Какие функции выполняет триггер Шмитта? В каких режимах работы параллельного порта он используется?

9. Как настроить порт на работу с аналоговыми сигналами? В чем отличие от цифровых?

10. Как задать конфигурацию порта? Используя программную модель, настройте 5 выход порта B на работу с открытым стоком и частотой не более 2 МГц; 10 вход порта A на с подтягивающим резистором к питанию.

11. В чем особенности конфигурирования альтернативных функций?

12. Какие преимущества обеспечивает ремапинг? Привести примеры.

13. Как обеспечивается защита конфигурации порта? Для чего это нужно?

14. В чем отличие в работе с битовыми данными по сравнению с AVR?

15. Как определить минимальную длительность входных сигналов параллельного порта? От каких параметров она зависит?

16. Как определить скорость ввода/вывода параллельного порта?

17. Как выполняется тактирование параллельного порта?

18. Электрические характеристики порта. Какая максимальная нагрузка допустима на одну линию порта? для МК?

19. Как обеспечить согласование МК при работе с 5-вольтовыми сигналами?

20. Как реализовать асинхронный/синхронный обмен с портом, если ведущим является внешнее устройство? Микроконтроллер? Какие настройки необходимо выполнить?

21. Как задать требуемый режим работы параллельного порта в CUBE MX?

22. Какие функции HAL используются для работы с параллельным портом? Перечислить или привести примеры использования функций HAL для реализации асинхронного/синхронного обменов.

### По разделу 1.6.2.1 Счетчики-таймеры

1. В чем отличия блока счетчиков-таймеров СТ STM32 от AVR?
2. Какие группы СТ используются в STM32?
3. Какие СТ используются в STM32F103C8T6? В чем их отличия?
4. Какая максимальная частота СТ в режиме таймер? Счетчика?
5. Какие источники синхроимпульсов в режиме таймер? Счетчика?
6. В каких случаях следует использовать суммирующий счет? Вычитающий? Реверсивный?
7. Приведите примеры использования аппаратного и программного запуска
8. Как задать коэффициент деления предделителя? В чем отличие от AVR?
9. Как задать модуль пересчета СТ? В чем отличие от AVR?
10. В какой момент формируется сигнал переполнения у суммирующего счетчика? Вычитающего? Реверсивного?
11. В чем особенность реализации режимов захвата/сравнения? В чем отличие от AVR?
12. Какие преимущества обеспечивает каскадное соединение СТ? В чем его особенность?
13. Как реализовать измерение частоты в STM32, используя каскадное соединение СТ? в AVR?
14. Какие возможности ремапинга в МК STM32F103C8T6?
15. Какие отличия в функциях TIM1 по сравнению с TIM2-TIM4?
16. Какие настройки необходимо выполнить, чтобы внешний сигнал поступил на вход счетчика**?**
17. Как настроить схему защиты от помех счетчика**?**
18. Какие сигналы являются входными в режим1 внешней синхронизации?

19. В чем отличие работы счетчика в режимах 1 и 2 внешней синхронизации? Рекомендации по использованию этих режимов.

20. В каких случаях следует использовать режим сброса **Reset mode**? Как он реализуется?

21. В каких случаях следует использовать режим управления счетом **Gated mode?** Как он реализуется?

22. В каких случаях следует использовать режим **запуска (Trigger Mode)?** Как он реализуется?

23. Для решения каких задач следует использовать режим захвата?

24. Принцип работы СТ в режиме захвата.

25. Какими возможностями обладает СТ STM32 в режиме захвата?

26. Как измерить параметры сигнала широтно-импульсного модулятора ШИМ?

27. Назначение энкодера. Как определить направление вращения? Угол поворота?

28. Как определить параметры энкодера в STM32?

29. В каких случаях следует использовать режим сравнения?

30. Принцип работы СТ в режиме сравнения.

31. Как вызвать принудительное формирование сигнала сравнения? Для каких целей?

32. В каких случаях необходимо формирование сигнала ШИМ?

33. Как настроить СТ на работу в режиме ШИМ?

34. В каких случаях следует использовать формирование одиночного импульса?

35. Какие параметры программируются в режиме одиночного импульса?

36. Как настроить СТ на режим одиночного импульса? Как задать требуемые параметры?

37. Как организовать каскадное соединение СТ?

38. Как задать функции ведущего СТ?

39. В чем особенности синхронизации ведомого СТ?

40. Выполните анализ и перечислите преимущества каскадного соединения СТ в приведенном примере.

41. Как выполнить инициализацию CN в CUBE MX?

42.Какие процедуры HAL можно использовать при работе с СТ?

### По разделу 1.6.2.2 Часы реального времени

1. Какие функции можно реализовать с помощью часов реального времени RTC?

2. Какие особенности RTCв МК STM32F103C8T6?

3. Как обеспечить бесперебойную (непрерывную) работу RTC?

4. Какие источники синхроимпульсов можно подключать к RTC? Какой предпочтительнее?

5. Как сформировать на входе счетчика RTC временной интервал заданной длительности?

6. В чем особенности взаимодействия RTC с другими блоками МК?

7. Как организовать на основе RTC часы-будильник?

8. Как выполнить инициализацию RTC в CUBE MX?

9. Какие процедуры HAL можно использовать при работе с RTC?

### По разделу 1.6.2.3 Сторожевой таймер

1. Для каких целей необходимо использовать сторожевой таймер СтТ? Принцип его работы.

2. Как настроить СтТ в независимый режим Independent watchdog (**IWDG**)?

3. Как задать время срабатывания СтТ в режиме **IWDG?**

4. Для каких целей используется оконный сторожевой таймер **WWDG?** Принцип его работы.

5. Как настроить СтТ в режим **WWDG?**
6. Как задать параметры оконного таймера?
7. Как выполнить инициализацию СтТ в CUBE MX?
8. Какие процедуры HAL можно использовать при работе СтТ?

### По разделу 1.6.3 Последовательный порт SPI

1. Почему **SPI** активно используется как интерфейс обмена?
2. Сколько портов **SPI** входит в состав STM32F103C8T6 и какие особенности?
3. Какие дополнительные режимы появились в **SPI** STM32F103xx по сравнению с AVR?
4. Как обеспечивается согласование регистра сдвига с периферийной шиной МК? Какие преимущества это обеспечивает?
5. В чем особенности управления сигналом NSS?
6. Принцип работы порта SPI. Как использовать флаги состояния в процессе обмена?
7. Как реализовать дуплексный обмен по SPI с несколькими устройствами? Симплексный?
8. В чем особенности полудуплексного обмена порта SPI? Как его настроить?
9. Как организовать мультипроцессорный режим порта SPI? Какие настройки необходимо выполнить?
10. Какие средства используются в порте SPI для обеспечения достоверности данных?
11. Как реализуется метод CRC? В чем особенности его реализации в порте SPI?
12. Как выполнить ввод/вывод порта SPI в программном режиме? Режиме ПДП? По прерыванию?
13. Как выполнить инициализацию порта SPI в CUBE MX?
14. Какие процедуры HAL можно использовать при работе с портом SPI?

### По разделу 1.6.4 Последовательный порт $I^2C$

1. В чем принципиальное отличие интерфейса $I^2C$ от SPI? Какой интерфейс обеспечивает большую эффективную пропускную способность? Почему?
2. Какими функциональными возможностями обладает порт $I^2C$ STM32F103xx?
3. Как настроить порт $I^2C$ в режим ведущего устройства? Ведомого?
4. Какие сигналы формирует ведущее устройство $I^2C$? Ведомое?
5. В каких случаях используется повторный старт?
6. Какие флаги следует проверять в ведущем устройстве? Ведомом?
7. Как реализуется арбитраж? См. [1].
8. Как ведущее устройство определяет, что оно стало ведомым?
9. Какие используются средства для повышения достоверности обмена информацией в порте $I^2C$?
10. Как передать контрольную сумму CRC?
11. Как проверить CRC при приеме?
12. Какие особенности в формировании сигнала синхронизации порта $I^2C$? Сравнить с AVR.
13. Как настроить порт $I^2C$ на требуемую частоту? Скважность? Длительность фронта/среза?
14. Почему необходимо использование внешних подтягивающих резисторов?
15. Какие параметры порта $I^2C$ ограничивают количество устройств, присоединенных к шине?
16. В чем особенности согласования адресов устройств, подключенных к шине $I^2C$?
17. Как выполнить широковещательную передачу?

18. Как выполнить инициализацию порта $I^2C$ в CUBE MX?

19. Какие функции HAL можно использовать при работе с $I^2C$?

### По разделу 1.6.5 Последовательный порт USART

1. В каких случаях следует использовать USART?

2. Какие дополнительные режимы работы по сравнению с AVR можно использовать в USART STM32? Перечислить.

3. Для работы с какими последовательными интерфейсами можно использовать USART STM32?

4. В чем особенности асинхронного режима USART STM32по сравнению с AVR?

5. Как определяется значение бита в процессе обмена USART STM32? На какие характеристики передачи влияет оверсемплинг?

6. Как задать требуемую скорость обмена в асинхронном режиме USART STM32? В чем отличие от AVR?

7. В каких случаях и как следует использовать сигналы управления модемом RTS, CTS?

8. Как контролировать состояние процесса обмена USART STM32? Какие новые флаги появились по сравнению с AVR? Какие преимущества это обеспечивает?

9. В чем особенность ввода/вывода по прерыванию USART STM32? Сравнить с AVR.

10. В каких случаях используется режим ПДП? Как настроить контроллер ПДП и USART STM32 на работу в этом режиме?

11. В чем особенность реализации асинхронной однопроводной полудуплексной связи USART STM32? Какие преимущества обеспечивает её использование?

12. Как реализуется синхронный режим USART STM32? В каких случаях его следует использовать? Сравнить с AVR.

13. В каких случаях следует использовать мультипроцессорный режим USART STM32?

14. В чем особенность реализации мультпроцессорного режима USART STM32? Сравнить с AVR.

15. Сколько USART входит в состав STM32F103C8T6? В чем отличие? На каких основаниях следует выбирать USART?

16. Какие преимущества обеспечивает «ремапинг» входов/выходов USART? Какие линии USART STM32F103C8T6 можно переключать?

17. Алгоритм работы порта USART STM32 в программном режиме? по прерыванию? в режиме ПДП.

18. Как задать режим работы USART STM32 в CubeMX?

19. Какие функции HAL используются для работы с портом? Перечислить или привести примеры реализации для USART STM32.

**По разделу 1.6.6 Аналого-цифровой преобразователь**

1. АЦП STM32. Основные характеристики АЦП

2. АЦП STM32. Какие средства предусмотрены для согласования по быстродействию с источниками напряжения? В чем отличие от AVR?

3. АЦП STM32. Как оценить быстродействие канала?

4. АЦП STM32. От каких характеристик зависит входной диапазон по напряжению АЦП?

5. АЦП STM32. В чем особенность источника опорного напряжения STM32F103C8T6?

6. АЦП STM32. Как определить значение входного напряжения в вольтах?

7. АЦП STM32. В чем особенность регулярных каналов? Инжектированных?

8. АЦП STM32. В каких случаях следует использовать инжектированные каналы?

9. АЦП STM32. Какие способы запуска каналов используются? Дать рекомендации по их применению.

10. АЦП STM32. Источники внешнего запуска. Какие факторы необходимо учитывать при использовании внешнего запуска?

11. АЦП STM32. Какое количество каналов может обрабатывать АЦП STM32F103C8T6?

12. АЦП STM32. В каких случаях следует использовать режим одиночного преобразования? непрерывного? Прерывистый режим? Режим сканирования? В чем их особенности?

13. В каких случаях следует использовать ввод/вывод АЦП в программном режиме? По прерыванию? В режиме ПДП? В чем особенность их реализации в STM32F103C8T6?

14. АЦП STM32. Какие преимущества обеспечивает использование калибровки АЦП? Как она реализуется?

15. АЦП STM32. В чем отличие калибровки от учета смещения в инжектированных каналах?

16. АЦП STM32. Какие функции выполняет аналоговый сторожевой таймер (оконный компаратор)? Какие преимущества он обеспечивает?

17. АЦП STM32. В чем отличие реализации допускового контроля с помощью АЦП в AVR и STM32?

18. Назначение датчика температур. Как определить значение?

19. АЦП STM32. Какие преимущества обеспечивает ремапинг?

20. Особенности режима каскадного соединения АЦП STM32?

21. АЦП STM32. В чем особенности подключения питания АЦП?

22. АЦП STM32. Как выполнить инициализацию АЦП?

23. Сравнительные характеристики АЦП AVR и STM32.

# 2 Средства программирования и отладки

Разработка программного обеспечения обычно предусматривает следующие этапы:

1. Анализ технического задания.
2. Выбор необходимых периферийных устройств и их режимов работы.
3. Выбор алгоритмов обработки данных.
4. Выбор микроконтроллера.
5. Распределение ресурсов и линий ввода/вывода.
6. Формирование системы синхронизации, задание режимов работы периферийных устройств, портов ввода/вывода и порта отладки.
7. Создание программы в соответствии с алгоритмом.
8. Компиляция программы.
9. Запись программы в память микроконтроллера.

В настоящее время имеется большой выбор как отдельных инструментов в виде редакторов, компиляторов, отладчиков кода, так и интегрированных сред разработки (IDE) для программирования и отладки микроконтроллеров.

В большинстве случаев может быть использован любой из доступных для семейства микроконтроллеров STM32 компиляторов. Каждый компилятор разрабатывался, исходя из определенных особенностей используемых компонентов системы. Опытные разработчики осуществляют выбор компилятора не только на основе анализа возможностей, но и по результатам тестирования конечного продукта по выбранным критериям (объем памяти, скорость работы и т.д.) [68].

Для создания кода достаточно иметь навыки программирования на языке C/C++, написать программу в любом текстовом редакторе и скомпилировать её с помощью, так называемого Make-файла. Данный файл содержит описание того,

как системе сборки необходимо проводить компиляцию: из какой папки начать сборку проекта, каким компилятором сформировать код и т.д. В результате исполнения make-файла создается bin- или hex-файл с программой, которая с помощью программатора записывается в память микроконтроллера.

Современные высокопроизводительные микроконтроллеры характеризуются широким набором возможностей, обеспечиваемых усложнением аппаратной и программной архитектуры, что, в свою очередь, обуславливает усложнение задачи создания встраиваемого программного обеспечения устройств и систем на базе микроконтроллеров. В связи с чем указанный способ разработки программ для микроконтроллеров применяется опытными разработчиками, в основном с целью оптимизации кода.

Упростить создание эффективно функционирующего кода на основе оптимальной конфигурации аппаратных ресурсов микроконтроллера позволяют специальные программные инструменты (среды) для разработчиков.

## 2.1 Среды разработки и отладки

Для микроконтроллеров семейства ARM STM32 обычно используются следующие среды разработки (IDE) [48]:

- Keil MDK-ARM (ARM C/C++ Compiler);
- CooCox CoIDE (GCC Compiler);
- IAR Embedded Workbench for ARM (IAR C/C++ Compiler);
- RealView Development Suite (ARM C/C++ Compiler);
- IAR Embedded Workbench for ARM (IAR C/C++ Compiler);
- MULTI IDE for ARM (Green Hills C/C++ Compiler);
- TASKING VX-toolset for ARM (Altium C/C++ Compiler);
- Sourcery CodeBench (GCC Compiler);
- Rowley CrossWorks for ARM (GCC Compiler);

- Atollic TrueSTUDIO (GCC Compiler);
- RAISONANCE Ride7 IDE for ARM (GCC Compiler);
- mikroC for ARM (MikroElektronika C Compiler).

Данный список IDE согласно сведениям из [48] составлен по результатам опроса пользователей. Названия IDE в списке указаны в порядке частоты использования (от наибольшей). При этом CooCox CoIDE распространялся бесплатно, тогда как Keil MDK-ARM в бесплатной версии имеет ограничение на размер кода в 32 килобайта. В опросе не уточнялось, какие именно версии Keil MDK-ARM используются шире. Возможно, для большинства пользователей достаточным объемом кода является 32 килобайта и использовалась именно бесплатная версия. Вместе с тем, высок риск возникновения потенциальных проблем при модернизации и поддержке уже реализованных с помощью Keil MDK-ARM продуктов. Это связано не только с лицензированием, но и обусловлено прочими особенностями организации составных частей проекта MDK-ARM [4].

Несмотря на то, что опрос фокусной группы проводился не так давно, к настоящему моменту появилось множество иных сред разработки и некоторые из них явились результатом совместной работы производителей микроконтроллеров и разработчиков IDE.

Учитывая концепцию настоящей работы, заключающуюся в предоставлении сведений для широкой массы пользователей с различными финансовыми возможностями, рассмотрению подлежат IDE, распространяемые на бесплатной основе.

На сайте STMicroelectornics в разделе, отражающем поддерживаемые IDE [25], в качестве полностью бесплатно распространяемых указаны следующие IDE:

- System Workbench for STM32;
- STM32CubeIDE;

- winIDEA Open;
- CooCox CoIDE;
- Atollic TrueSTUDIO.

При этом отмечено, что CooCox CoIDE и Atollic TrueSTUDIO в настоящее время не рекомендованы для разработки новых продуктов, они могут использоваться только для уже реализуемых проектов. В связи с чем, данные IDE далее не рассматриваются.

Сред разработки, позволяющих реализовывать проекты для микроконтроллеров много, поэтому при выборе IDE разработчик должен самостоятельно оценить преимущества и недостатки различных систем и выбрать подходящие. На взгляд авторов оптимальными являются IDE, рекомендованные производителем микроконтроллеров. Однако, если разработчик привык пользоваться какой-либо средой разработки он может установить для нее дополнение (если оно есть), компилятор ARM архитектуры (GNU Arm Embedded Toolchain), подключить отладчик OpenOCD (Open On-Chip Debugger) и разрабатывать приложения с помощью привычных инструментов. Следует заметить, что такие решения часто являются платными и требуют дополнительных усилий и навыков в процессе конфигурирования/ настройки системы. Зачастую они изначально не были рассчитаны на разработку программ для микроконтроллеров. В качестве примера приведем несколько подобных систем и дадим их краткие характеристики:

- Embedded ARM Toolchain – пакет программ для компиляции кода МК, имеющих ARM архитектуру. Бесплатный.
- OpenOCD (Open On-Chip Debugger) – открытое ПО для программирования и отладки МК STM32 (и других ARM), имеет консольный интерфейс, может использоваться как модуль для других сред разработки, бесплатная.

- Eclispe – среда для разработки кода, поддерживает множество языков программирования, вместе с OpenOCD и ARM Toolchain может использоваться для разработки на C/C++ под STM32 и другие ARM. Бесплатная.
- CLion – среда для разработки кода на языках C/C++, вместе с OpenOCD и ARM Toolchain может использоваться для разработки под STM32 и другие ARM. Платная, есть 30-дневная пробная версия.
- Visual Studio 2019. Изначально не содержит средств разработки для МК, требует дополнительной установки плагина Visual GDB, который в автоматическом режиме (при установке) скачивает компилятор ARM архитектуры GNU Arm Embedded Toolchain. Утилита ST-LINK Utility предназначена для программирования микроконтроллеров STM8 и STM32 через программатор ST-LINK или ST-LINK/V2. Также необходим отладчик – OpenOCD.

Стоимость разработки программного обеспечения является основным фактором в индустрии встраиваемых решений, при этом этап создания программы включает в себя отладку программы, проводить которую удобно с использованием описанных сред разработки. Несмотря на то, что микроконтроллеры от STM отличаются по мощности, скорости и иным параметрам, внутри у них много схожего. Для упрощения работы программистов в случае перехода с одного чипа на другой существует ряд абстракций.

**Первая абстракция CMSIS** – стандарт на интерфейс ПО микроконтроллеров Cortex (англ. Cortex Microcontroller Software Interface Standard).

Стандартизируя интерфейсы всех продуктов производителей микроконтроллеров с ядром Cortex-M, можно сократить процесс создания нового проекта или переноса старого на микроконтроллер другого производителя. ARM CMSIS – это независимый от производителя уровень абстракции для серии ядер Cortex-M, а

также интерфейс отладчика. CMSIS предоставляет последовательные и простые интерфейсы для ядра, его периферии и операционных систем реального времени.

Библиотека **CMSIS включает** в себя перечисленные ниже **компоненты**.

- **CMSIS-CORE**: API для ядра Cortex-M и периферии. Эта часть предоставляет стандартизированный интерфейс для Cortex-M0, Cortex-M3, Cortex-M4, SC000, и SC300. Включает в себя также дополнительные SIMD-инструкции (англ. single instruction, multiple data – одиночный поток команд, множественный поток данных) для Cortex-M4.
- **CMSIS-Driver**: определяет основные драйверы интерфейсов периферии. Содержит API для операционных систем реального времени (ОСРВ, или англ. Real-Time Operating Systems – RTOS) и соединяет микроконтроллер с промежуточным ПО, таким как стек коммуникации, файловая система или графический интерфейс.
- **CMSIS-DSP**: коллекция из более чем 60 функций для различных типов данных (относятся к обработке сигналов, DSP – Digital Signal Processor): с фиксированной точкой (q7, q15, q31) и с плавающей точкой (32 бита). Библиотека доступна для Cortex-M0, Cortex-M3, и Cortex-M4. Реализация библиотеки для Cortex-M4 оптимизирована с использованием SIMD-инструкций.
- **CMSIS-RTOS API**: общий API для систем реального времени. Эта часть предоставляет стандартизированный программный интерфейс, позволяющий большинству программных шаблонов, промежуточного ПО, библиотек и других компонентов RTOS-систем работать на ядре Cortex.
- **CMSIS-DAP** (DebugAccessPort): стандартизованное программное обеспечение для отладчика (DebugUnit), которое подключает CoreSightDebugAccessPort. CMSIS-DAP поставляется

отдельно и используется для интеграции (на отладочных платах). Далее будет использоваться только CMSIS-CORE.

**CMSIS** регламентирует соглашения об именах, т.е. в ней описано как обращаться к регистрам, портам, в каком порядке. Есть несколько системных функций, связанных с функциями ядра – настройка приоритетов прерываний и тактового генератора.

**Вторая абстракция SPL** (англ. Standard Peripherals Library) – библиотека, созданная компанией STMicroelectronics на языке Си для своих микроконтроллеров семейства STM32xx. Содержит функции, структуры и макросы, позволяющие облегчить работу с периферией микроконтроллера. SPL на базе имен CMSIS создает функции для работы с различной периферией, включает функции инициализации периферии. Однако, SPL не нашел широкую поддержку в профессиональном сообществе, сейчас не поддерживается и не развивается.

**Третья абстракция HAL** (Hardware Abstraction Layer, слой аппаратных абстракций) – слой абстрагирования, реализованный в программном обеспечении, находящийся между физическим уровнем аппаратного обеспечения и программным обеспечением. Это дальнейшее развитие концепции, заложенной в библиотеке SPL: компания ST Microelectronics стремится дать разработчику единый инструмент для работы со всеми чипами STM32. HAL позволяет абстрагироваться от работы с регистрами и значительно облегчить написание кода. В настоящее время активно развивается кроссплатформенная утилита с графическим интерфейсом STM32CubeMX (общим для всех микроконтроллеров семейства STM32), основанная на фреймворк HAL STM32Cube. Фреймворк STM32Cube/HAL для каждой линейки чипов свой (также, как и в SPL). Такой подход позволяет минимизировать время, которое разработчик тратит на конфигурирование системы, распределение ресурсов и проверку конфликтов. Используя графический

интерфейс STM32CubeMX, разработчик настраивает программную модель устройства, далее с помощью код-генератора и фреймворк конкретной модели микроконтроллера, генерирует код.

STM32CubeMX – позволяет сгенерировать проект на основе CMSIS+HAL под различные IDE, что помогает избежать многих ошибок, на начальном этапе освоения микроконтроллеров семейства STM32. Включает в себя драйверы различной периферии. Имеется возможность подключить HAL-библиотеку, сконфигурировать ее на нужный контроллер и, например, дать команды «оправить/принять байт по UART», «передать байт через прямой доступ к памяти ПДП» (англ. DMA) и т.д. Все это реализуется с использованием стандартизованных библиотек и имен, что позволяет менять контроллеры и переносить проекты (мигрировать) с одного на другой.

**Описание HAL функций можно найти в руководстве пользователя соответствующего семейства контроллеров, которое доступно на сайте st.com (для микроконтроллера STM32F103C8T6 используется STM32F1HAL_User_ manual.pdf).**

### 2.1.1 Конфигуратор STM32CubeMX

STM32CubeMX представляет собой графический инструмент, позволяющий конфигурировать микроконтроллеры и микропроцессоры STM32, а также генерировать код на языке C для ядра Arm® Cortex®-M [35]. Процесс конфигурации осуществляется пошагово. На первом шаге выбирается микроконтроллер или микропроцессор STMicroelectronics STM32, который соответствует требуемому набору периферийных устройств.

На втором шаге конфигурируются GPIO и настраиваются таймеры системы, назначаются периферийные устройства как на Arm® Cortex®-M, так и на Cortex®A. При этом обеспечивается

разрешение конфликтов, возникших при настройке линий, осуществляется помощь в настройке тактирования, энергопотребления и настройке периферийных устройств.

В конце пользователь запускает генерацию кода, соответствующего выбранному варианту конфигурации. На этом этапе формируется код инициализации микроконтроллера и периферии на языке C для Arm® Cortex®-M, готовый к использованию.

В состав STM32CubeMX входят программные библиотеки для отдельных серий МК, загружаемые с сервера производителя при создании первого проекта [5]. Они сохраняются на диске локального компьютера в специальном каталоге под названием репозиторий для дальнейшего их использования в пользовательских приложениях. Такие библиотеки включают STM32Cube HAL (уровень абстракции встроенного ПО, обеспечивающий максимальную взаимозаменяемость МК семейства STM32), а также согласованный набор компонентов библиотечного ПО MiddleWare (RTOS, USB, TCP/IP и графика, рисунок 2.1).

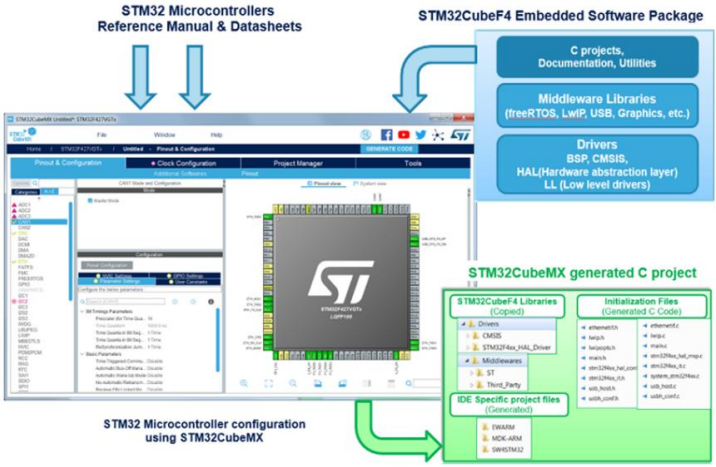



Рис. 2.1 – Процесс генерации C кода в STM32CubeMX

При создании нового проекта (File → New Project …) STM32CubeMX предлагает сделать выбор контроллера/ отладочного модуля. Удобная система фильтрации позволяет ограничивать отображаемые списки рамками определенных семейств микроконтроллеров или заданными параметрами, включая поддержку необходимой периферии: количества каналов ЦАП и АЦП, Ethernet, USB, CAN, объем встроенной Flash-памяти, RAM, EEPROM и т.д.

STM32CubeMX обеспечивает интуитивно понятный, визуальный графический интерфейс с четырьмя закладками для настройки используемого в проекте контроллера: «Pinout&Configuration», «Clock Configuration», «Project Manager» и «Tools». В окне конфигурации STM32CubeMX отображаются параметры всех программных компонентов: порты ввода/вывода (GPIO), связь с периферией, межплатформенное ПО (MiddleWare). (рисунок 2.2).

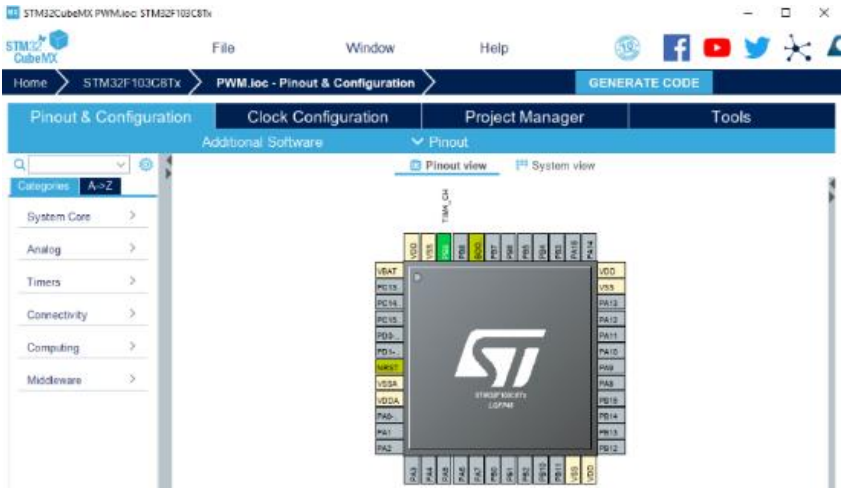



Рис. 2.2 – Интерфейс код-генератора STM32CubeMX

В процессе работы пользователь может конфигурировать и настраивать используемое встроенное ПО. Для этого предлагаются: утилита конфигурирования выводов, утилита настройки схемы тактирования, калькулятор энергопотребления, утилита конфигурации периферийных модулей микроконтроллера (GPIO, USART и так далее), набор библиотек (USB, TCP/IP и другие).

Задействованные в проекте линии микроконтроллера отображаются в разделе **GPIO → Configuration** (рисунок 2.3). На данной вкладке имеется возможность конфигурирования каждой линии. Для этого необходимо выбрать соответствующую вкладку (в примере на рисунке 2.3 – вкладка «SYS», «RCC» или «GPIO»), после чего установить необходимые параметры, определяющие режим работы (в примере на рисунке 2.3 – показаны параметры для линии **PC13**).

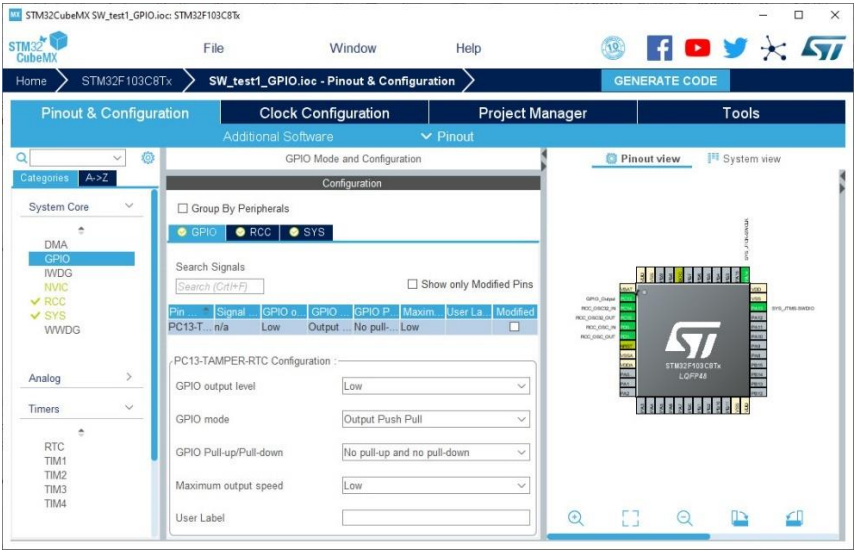



Рисунок 2.3 – Окно определения параметров линий

Линии, задействованные CT_TIM4, показаны на рисунке 2.4.

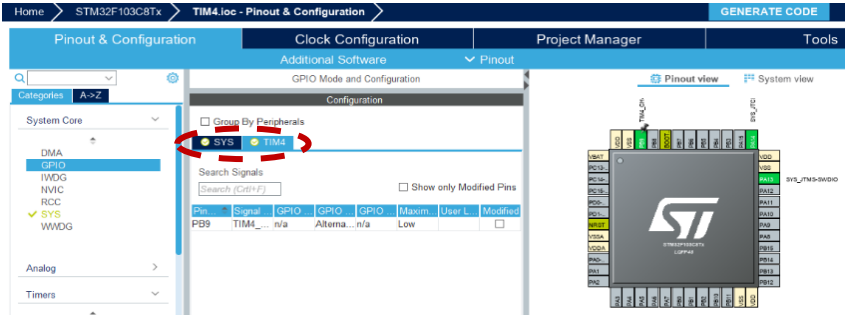



Рис. 2.4 – Конфигурация линий TIM4

Выявить и устранить конфликты, присутствующие в проекте, помогает цветовая подсветка отдельных элементов, меняющаяся в соответствии с выбором и активацией функций пользователем. Зеленый цвет свидетельствует об отсутствии проблем, желтым цветом окрашивается ограниченная функциональность, а красный свидетельствует о недоступности настроек (рисунок 2.5).

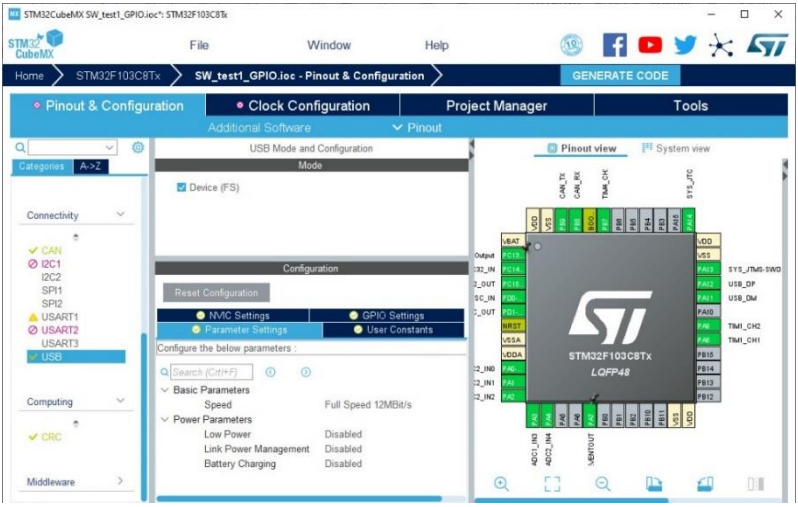



Рис. 2.5 – Цветовая подсветка CubeMX

На вкладке «Clock Configuration» показана конфигурация системы тактирования и часов реального времени в зависимости от

выбранных разработчиком источников и требуемой частоты выходного сигнала (рисунок 2.6).

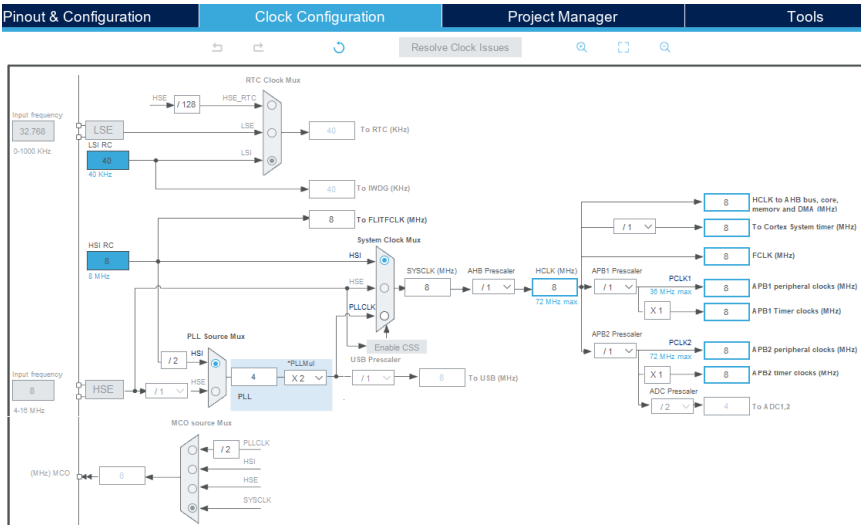



Рис. 2.6 – Конфигурация системы тактирования в STM32CubeMX

Если нет особых требований, конфигурацию тактирования можно оставить по умолчанию – внутренний RC-генератор на 8 МГц (HSI RC).

Для подключения генераторов с внешним кварцевым резонатором (HSE, LSE) нужно выполнить настройки во вкладке **Pinout&Configuration → System Core**, раздел RCC (reset clock control). Мастер конфигурации тактирования автоматически выполнит необходимые настройки в схеме синхронизации внутренних регистров и периферии микроконтроллера. При необходимости используется умножитель частоты PLL (Phase Locked Loop, фазовая автоподстройка частоты).

Настройка на максимальную частоту МК обеспечивается подключением генератора HSE с внешним кварцевым резонатором (рисунок 2.7) и установкой требуемого множителя PLL.

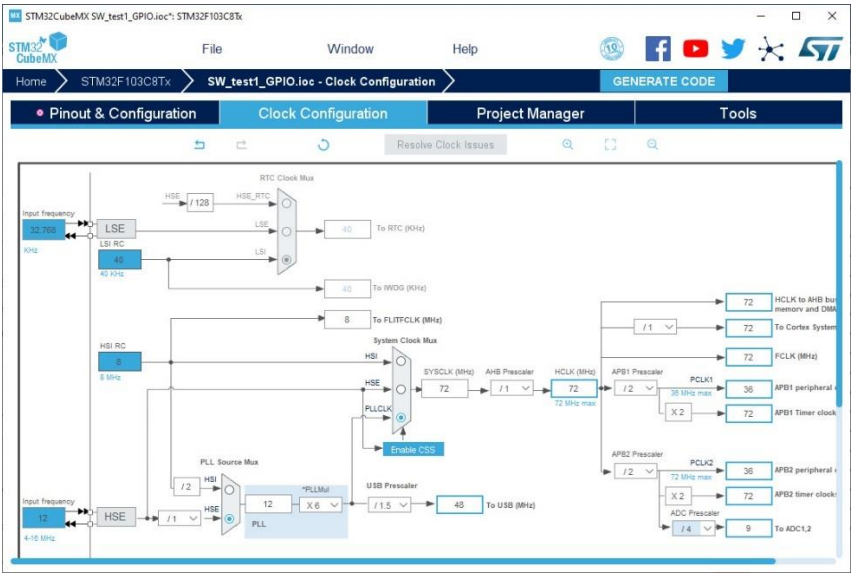



Рис. 2.7 – Конфигурация системы тактирования от генераторов с внешним кварцевым резонатором

В отладочном модуле установлен кварцевый резонатор на 12 МГц. Для обеспечения максимальной тактовой частоты STM32F103C8T6 в 72 МГц множитель должен быть равным «x6». На этой частоте может работать ядро МК и периферийная шина APB2. Однако максимальная частота APB1 равна 36 МГц. Поэтому необходимо установить предделитель частоты «APB1 Prescaler» на 2 или меньшее значение.

Снижение потребляемой устройством энергии является важной задачей при создании большинства встраиваемых систем и автономных устройств. Чтобы обеспечить минимальное энергопотребление, STM32CubeMX предлагает калькулятор потребляемой мощности (рисунок 2.8). Следует помнить, что ток потребления МК зависит от частоты синхронизации и числа включенных периферийных устройств (см. таблицы 1.7, 1.8 раздела 1.5).

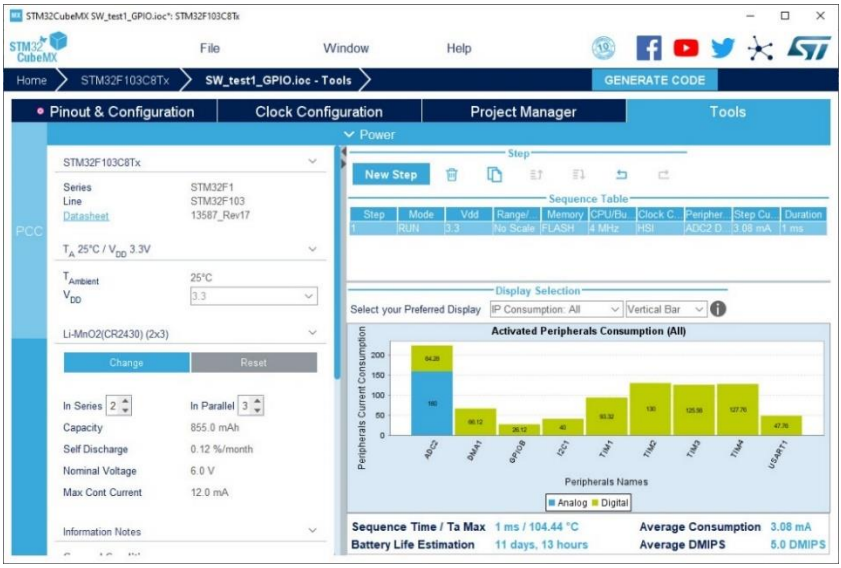



Рис. 2.8 – Калькулятор энергопотребления в STM32CubeMX

Вкладка калькулятора потребляемой мощности позволяет оценить среднее энергопотребление и длительность работы от встроенных батарей в зависимости от режима использования контроллера, выбранного напряжения питания, температуры окружающей среды, емкости и типа батарей.

Power Consumption Calculator (PCC) позволяет задавать режимы работы контроллера проектируемого устройства. При этом в каждом случае можно указать активную периферию (рисунок 2.9). В отдельном списке предлагается выбрать тип используемой батареи из имеющихся в базе данных или задать параметры другой батареи. Таким образом, можно создать полный профиль энергетической конфигурации устройства, сформировать отчет и сохранить его в одном из наиболее удобных форматов. Этот инструмент позволяет быстро оценить энергетические затраты разрабатываемой системы и степень ее автономности, критически важные в условиях энергосбережения.

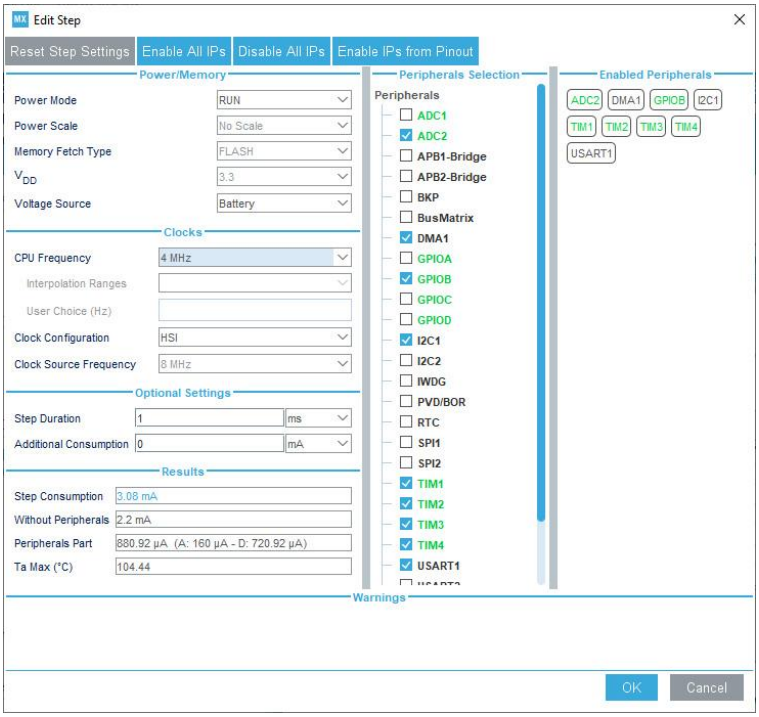



Рис. 2.9 – Калькулятор энергопотребления в STM32CubeMX

После завершения настройки проекта разработчик генерирует на базе созданной конфигурации, программный Си-код. Выбор среды IDE доступен через вкладку «Project Manager» в разделе «Toolchain/IDE». При использовании System Workbench for STM32, необходимо указать SW4STM32. В этом же окне задается название проекта (Project Name) и место его расположения.

Исходное место расположения копируемых в проект библиотек (так называемый репозиторий) по умолчанию находится на диске C в папке «Мои документы». Репозиторий можно переместить в любое место, не забыв при этом сделать соответствующие изменения в настройках программы STM32CubeMX через меню **Help → Updater Settings**. Место

расположения рабочего пространства в SW4STM32 можно изменить в меню **File → Switch Workspace**.

В случае необходимости последующей отладки и прошивки в МК, необходимо разрешить работу с программатором через последовательный интерфейс Serial Wire Debug (SWD). Для этого на вкладке **Pinout&Configuration → System Core**, раздел «SYS» в окне «Mode» необходимо перейти к полю «Debug» и выбрать интерфейс отладки «Serial Wire» (рисунок 2.10). При этом выходы PA13, PA14 микропроцессора автоматически конфигурируются на работу в режиме SWD.

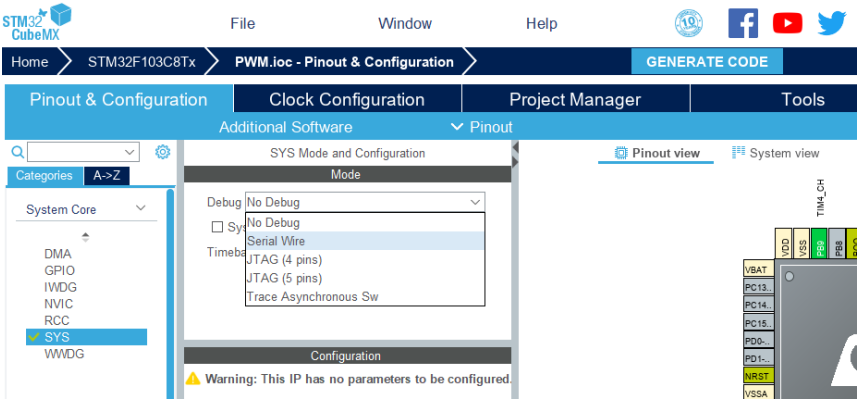



Рис. 2.10 – Настройка отладки

Код сконфигурированного проекта генерируется по нажатию кнопки «Generate Code» либо сочетанию клавиш **Ctrl+Shift+G**. Если при конфигурировании ошибок допущено не было, код генератор предложит открыть проект в IDE для дальнейшего редактирования.

Более подробную информацию по работе с STM32CubeMX можно найти в соответствующем руководстве пользователя, представленном в [43]. В приложении 3 приведен пример настройки портов ввода-вывода общего назначения.

### 2.1.2 Среда разработки System Workbench for STM32

System Workbench for STM32, кратко именуемая SW4STM32, представляет собой бесплатную среду разработки программного обеспечения на основе IDE Eclipse, которая поддерживает всё семейство микроконтроллеров STM32 и связанных с ними плат [37]. SW4STM32 можно загрузить с веб-сайта www.openstm32.org, на котором также доступны форумы, блоги, тренинги, форумы технической поддержки.

Основные характеристики SW4STM32:

- комплексная поддержка микроконтроллеров STM32, плат STM32 Nucleo, наборов Discovery и оценочных плат, а также прошивки STM32 (стандартная периферийная библиотека или STM32Cube HAL);
- компилятор GCC C / C ++;
- нет ограничений по размеру кода;
- отладчик на основе GDB;
- Eclipse IDE с управлением командной работой совместим с плагинами Eclipse;
- поддержка программатора ST-LINK.

На рисунке 2.11 схематично представлен состав компонентов SW4STM32 и принцип их взаимодействия [5].

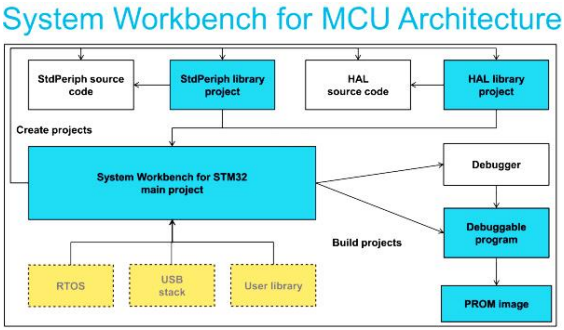



Рис. 2.11 – Архитектура System Workbench for STM 32

SW4STM32 включает в себя все инструменты, необходимые для разработки и может интегрироваться с плагинами для Eclipse. Работа с периферией возможна благодаря библиотекам SPL и HAL, могут использоваться пользовательские библиотеки, стек USB и RTOS. В составе SW4STM32 имеется встроенный отладчик. Необходимо учесть, что **при отладке проекта с помощью программатора ST-Link, может потребоваться установка драйвера**.

В процессе установки и/или работы с System Workbench for STM32 постоянный доступ в интернет не требуется.

Однако, на начальном этапе создания проекта, соединение с сетью может понадобиться для обновления или загрузки программной платформы (framework) используемого контроллера или отладочного модуля. **Загрузка осуществляется с сервера компании производителя www.st.com.**

При запуске SW4STM32 в качестве рабочей среды (workspace) необходимо указать директорию, которую IDE Eclipse будет использовать для хранения временных и конфигурационных файлов проекта (рисунок 2.12). **Путь не должен содержать пробелов и кириллических символов.**

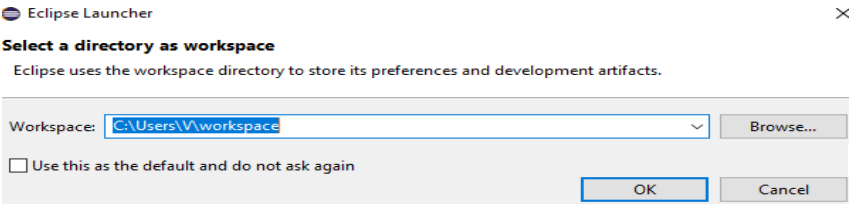



Рис. 2.12 – Задание директории для рабочей среды

После выбора директории появляется экран приветствия, закрыв который пользователь видит разделенный на несколько рабочих зон графический интерфейс пользователя (рисунок 2.13). При этом интерфейс может быть настроен с учетом предпочтений разработчика.

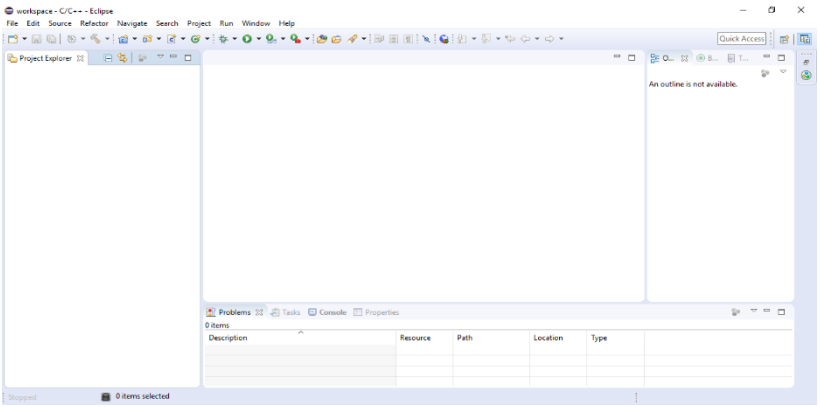



Рис. 2.13 – Рабочий экран SW4STM32 по умолчанию

Слева на вертикальной панели окна отображается структура рабочих проектов. Элементы текущего активного проекта (переменные, функции, константы, заголовочные файлы, список задач) отображаются на вертикальной панели справа.

Функции, реализованные в подключенных к проекту файлах, можно посмотреть на вкладке «Outline» (см. рисунок 2.14). Для просмотра необходимо, чтобы в центральной рабочей зоне был открыт интересующий файл, после чего требуется перейти к нему.

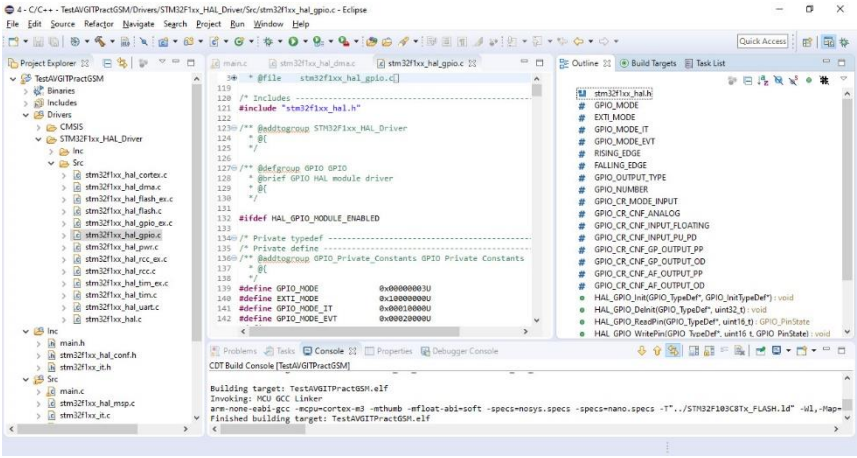



Рис. 2.14 – Функции, реализованные в файле stm32f1xx_hal_gpio.c

Центральная рабочая зона пользовательского интерфейса предназначена для редактирования исполняемого программного кода. В правом верхнем углу на панели инструментов есть кнопка «Open Perspective», позволяющая добавить (в дополнение к основному окну – перспектива «C/C++») ряд вкладок, включая кнопку для переключения рабочего экрана в режим отладки – перспектива «Debug» (рисунок 2.15).


Рис. 2.15 – SW4STM32 в режиме отладки

В нижней горизонтальной панели имеется ряд информационных вкладок, включая вкладку «Problems», в которой выводится дополнительная информация о возникающих в процессе работы ошибках и конфликтах. В этом поле также может отображаться информация о скомпилированном файле.

На рисунке 2.16. показан пример сгенерированного код-генератором проекта. Видна его структура, а также используемые библиотеки HAL-функций. Следует заметить, что к проекту подключаются библиотеки HAL, используемые для работы с подключенным и сконфигурированным в STM32CubeMX оборудованием.

На основе созданного в SW4STM32 проекта можно сгенерировать файл с двоичным кодом, который записывается в МК. Сборка проекта запускается через **Project → Build All** или с помощью сочетания клавиш «Ctrl+B». В случае успешного завершения этой операции в структуре проекта появляется дополнительная ветка «Binaries» с двоичным кодом проекта, имеющим расширение «.elf» (Executable and Linkable Format).

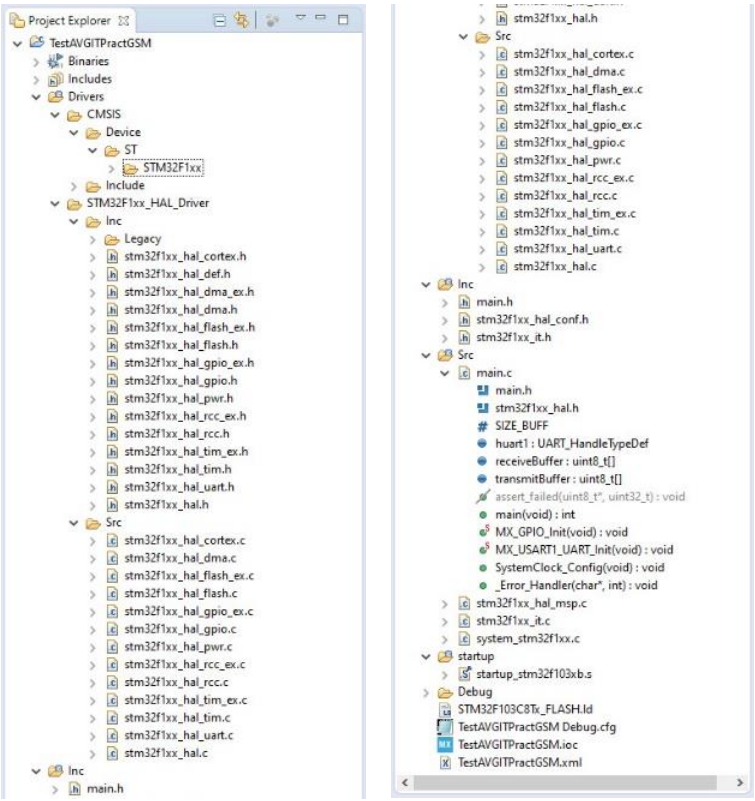



Рис. 2.16 – Список заголовочных файлов HAL, подключенных к проекту

Перед переходом к процессу запуска/отладки модуль PINBOARD II с контроллером STM32 необходимо подключить к USB порту компьютера через программатор ST-Link. Важно убедиться в том, что программатор появился в списке диспетчера устройств компьютера.

Существует несколько вариантов запуска процесса отладки: пункт меню **Run → Debug** кнопка  на панели инструментов или «горячая» клавиша **F11**. Однако, предварительно требуется определить конфигурацию отладки. Сделать это можно через

свойства проекта (меню **Project → Properties**), в котором необходимо выбрать «Run/Debug Settings» (рисунок 2.17).

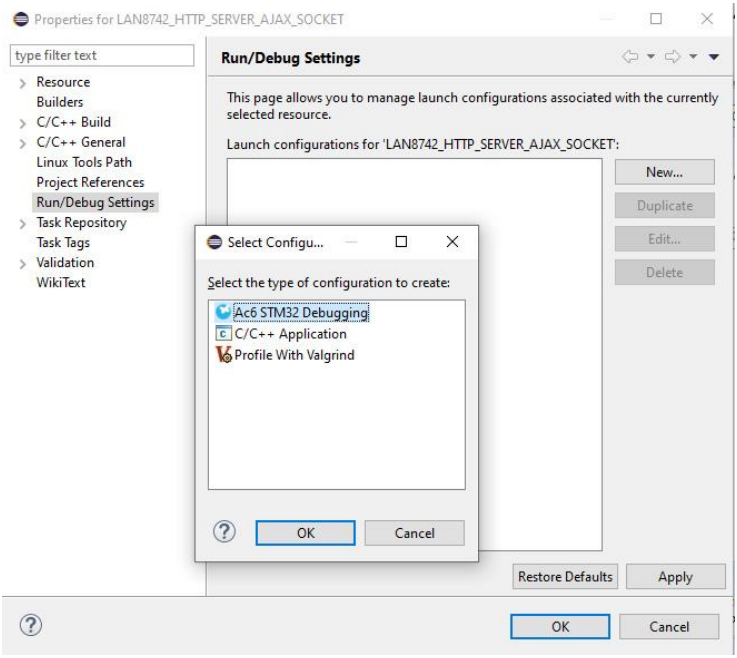



Рис. 2.17 – Настройка конфигурации отладки

Далее необходимо нажать кнопку «New», выбрать из предложенного списка конфигурацию «As6 STM32 Debugging» и подтвердить выбор, нажав кнопку «OK». Если проект к этому времени был скомпилирован, то соответствующий двоичный файл с кодом подключается к конфигурации отладки.

После сборки проекта и запуска процесса отладки (**Run → Debug**  ), система наверняка выдаст сообщение о невозможности прошивки и перезапуске контроллера т.к. требуется аппаратный сброс. Исправить ошибку можно, запустив процесс отладки при нажатой кнопке «Reset» на отладочном модуле, либо, задав в настройках отладки режим программного сброса. Для этого

необходимо выбрать используемую конфигурацию отладки и нажать кнопку «Edit» для перехода в раздел редактирования конфигурации (рисунок 2.18).

На вкладке «Debugger» нажать кнопку «Show generator options…» (название кнопки сменится на «Hide generator options...»). Появится раздел «Generator options», в области «ModeSetup» необходимо выбрать режим сброса (ResetMode) «Software system reset» (рисунок 2.19) и сохранить конфигурацию.

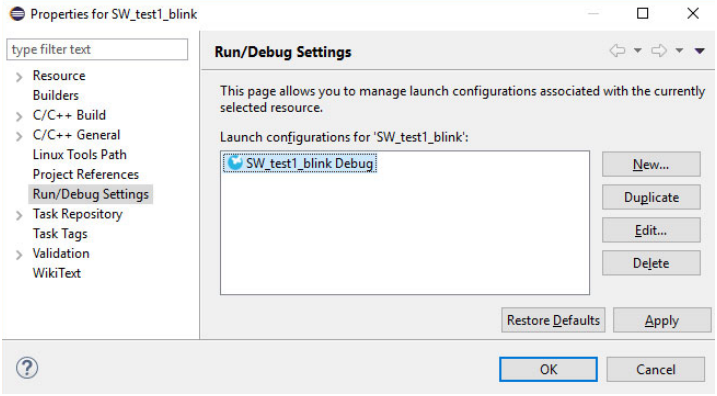



Рис. 2.18 – Редактирование конфигурации

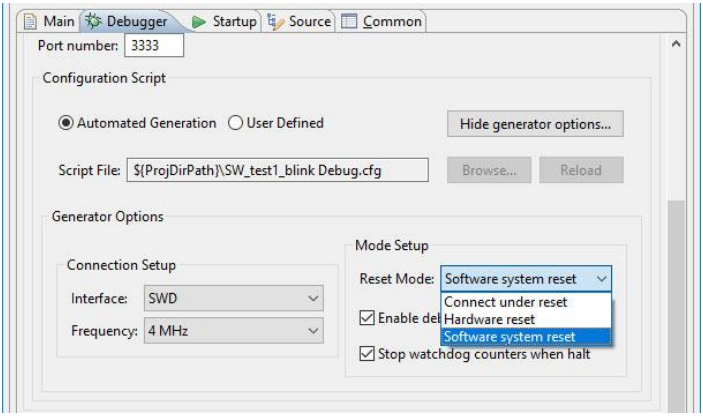



Рис. 2.19 – Выбор программного режима сброса микроконтроллера

По завершении настройки отладчика можно приступать к процессу отладки.

Перспектива (Perspective) «Debug»  (рисунок 2.16) позволяет управлять запуском и отладкой программы. В ней сверху-слева отображается структура стека для приостановленных потоков отлаживаемой программы. Каждый поток отображается как отдельный узел в общей структуре. Окно справа предназначено для контроля состояния переменных, точек прерываний и регистров ввода/вывода. Ниже расположено окно с несколькими вкладками для редактирования файлов проекта.

В самом нижней области экрана отладки размещены вкладки консоли для отображения хода процесса, имеющихся задач, выявленных ошибок и исполняемых файлов (рисунок 2.20).

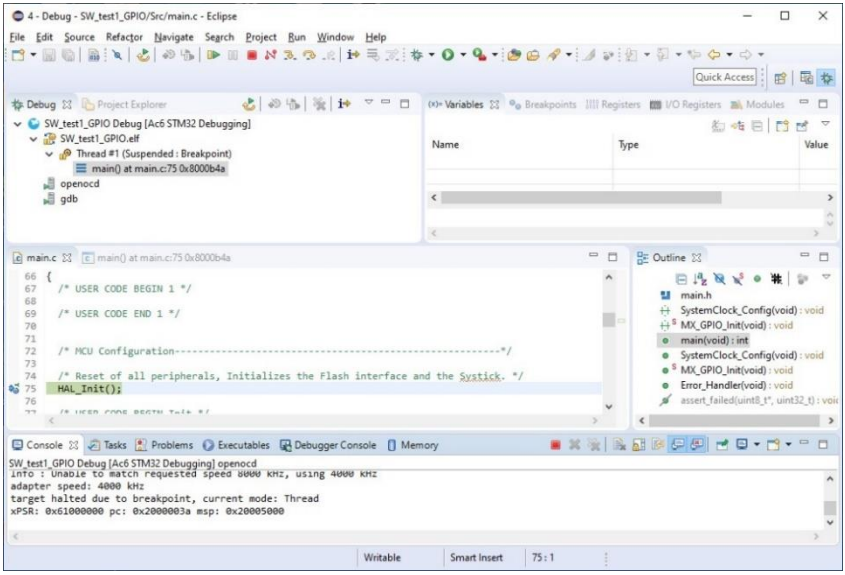



Рисунок 2.20 – Перспектива «Debug»

В режиме отладки содержание различных регистров можно посмотреть во вкладках «Registers», «I/O Registers» и т.д. (рисунок 2.21). Значения переменных или выражений можно просматривать в реальном времени во вкладке «Live Expressions». Чтобы добавить переменную для просмотра, необходимо нажать на «Add new expression» и ввести имя переменной.

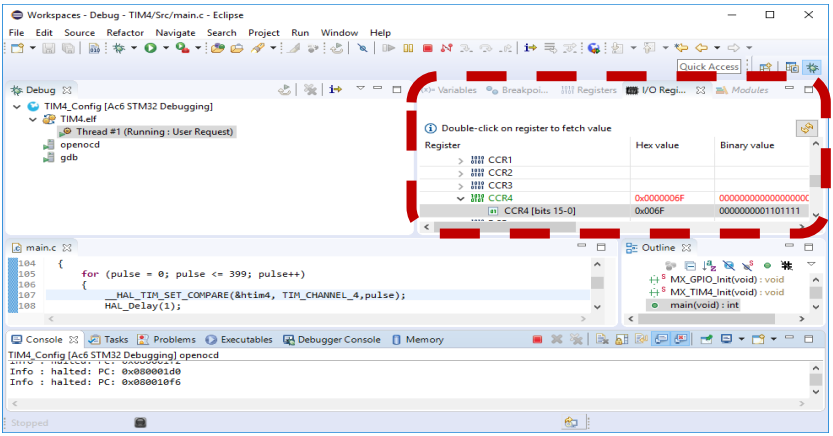



Рис. 2.21 – Окно отладки

Прежде чем продолжить изучение структуры проекта и начать написание первой программы, необходимо знать фундаментальные основы языка Си и основы программирования микроконтроллеров Cortex. Базовые понятия, необходимые для программирования отладочных модулей с процессорами ARM Cortex от компании STM32, приведены в приложении 2.

Программирование и запуск работы микроконтроллера в SW4STM32 мало чем отличается от процесса отладки: необходимо создать конфигурацию запуска или воспользоваться конфигурацией, созданной при отладке (рисунок 2.17).

Запуск процесса программирования микроконтроллера производится нажатием сочетания клавиш **Ctrl+F11** (пункт меню **Run → Run**) или щелчком мыши на соответствующей пиктограмме

 (рисунок 2.20). При этом процесс отладки должен быть остановлен.

Необходимо отметить, что программирование микроконтроллеров предполагает не только написание кода, но и предварительную настройку микроконтроллера и его периферии. Существенно облегчить эту задачу позволяет код-генератор STM32CubeMX, поддерживающий работу с различными IDE (включая SW4STM32).

STM32CubeMX и System Workbench for STM32 позволяют ускорить и упростить процесс отладки, однако являются отдельными инструментами, что накладывает определенные неудобства при изменении или частичной модернизации проекта. Данного недостатка лишена среда разработки STM32CubeIDE.

### 2.1.3 Среда разработки STM32CubeIDE

STM32CubeIDE представляет собой C/C++ платформу разработки для микроконтроллеров и микропроцессоров STM32, обладающую возможностью конфигурации периферии, функциями генерации, компиляции кода и отладки. Основана на инфраструктуре ECLIPSE™/CDT и GCC для разработки, также GDB для отладки. Это позволяет интегрировать сотни существующих плагинов, которые дополняют функции ECLIPSE™ IDE [9].

STM32CubeIDE объединяет все функциональные возможности STM32CubeMX, предлагая универсальный инструмент и экономя время на установку и разработку. После выбора микроконтроллера создается проект и генерируется код инициализации. В любой момент во время разработки пользователь может вернуться к настройке периферийных устройств, чтобы повторно сгенерировать код инициализации.

Код-генератор, в процессе формирования файлов проекта, расставляет комментарии вида: /* USER CODE BEGIN 1 */ … /*

USER CODE END 1 */. Во избежание потери изменений (при повторной генерации), внесенных пользователем, код должен находиться между комментариями.

Код-генератор STM32CubeIDE содержит инструмент анализа содержимого стека, который предоставляет пользователю полезную информацию о состоянии проекта и требованиях к памяти. STM32CubeIDE также включает в себя стандартные и расширенные функции отладки: просмотр регистров ядра ЦП, оценка текущего состояния памяти/периферийных регистров, а также просмотр значений переменных в реальном времени, интерфейс Serial Wire Viewer или инструмент анализа неисправностей.

Основные особенности STM32CubeIDE.

- Интеграция функционала STM32CubeMX обеспечивает возможность:
- выбирать STM32 микроконтроллер/микропроцессор в интегрированной среде разработки;
- осуществлять конфигурацию линий, таймеров, периферии и стандартных протоколов;
- создавать проект и генерировать код инициализации.
- Расширены функции отладки, в том числе:
- мониторинг периферийных регистров и памяти;
- просмотр значений переменных в реальном времени;
- анализ системы и отслеживание ее состояния в реальном времени (SWV);
- инструмент анализа ошибок процессора;
- поддержка отладочных средств ST-LINK (STMicroelectronics) и J-Link (SEGGER);
- возможность импорта проектов из Atollic® TrueSTUDIO® и AC6 System Workbench for STM32 (SW4STM32).

STM32CubeIDE является бесплатно распространяемой средой разработки, загрузить которую можно с сайта производителя [9]. На том же ресурсе можно найти подробное руководство пользователя. Далее описана работа в STM32CubeIDE версии 1.1.0. Среда разработки регулярно обновляется – функционал и интерфейс может отличаться.

При запуске STM32CubeIDE появится окно выбора рабочей среды (рисунок 2.22), также, как и в STM32CubeMX.

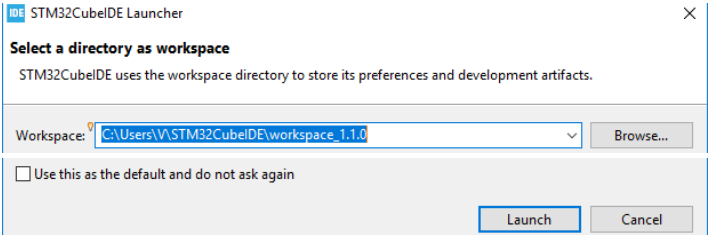



Рис. 2.22 – Окно выбора рабочей среды

После этого откроется стартовая страница (рисунок 2.23), на которой можно создать новый проект или импортировать готовый, созданный в SW4STM32 или Atollic TrueSTUDIO.

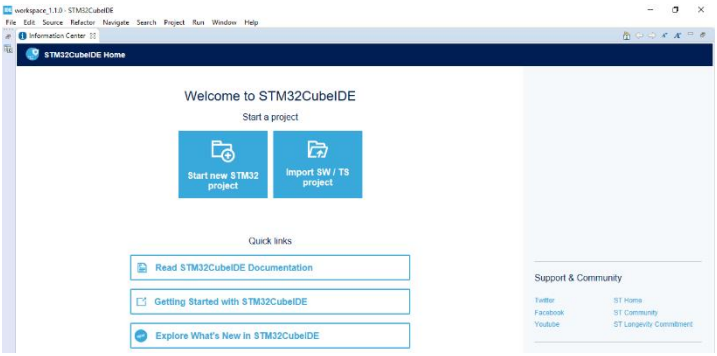



Рис. 2.23 – Стартовая страница STM32CubeIDE

При создании нового проекта («Start new STM32 project») появится окно выбора микроконтроллера (рисунок 2.24).

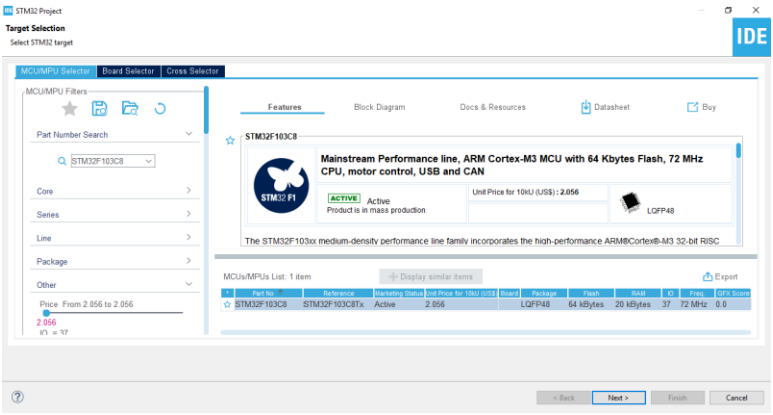



Рис. 2.24 – Окно выбора контроллера в STM32CubeIDE

После окончания работы в сервисе выбора микроконтроллера, появится окно со стандартными настройками проекта. На следующей странице необходимо определить название проекта и указать язык программирования (рисунок 2.25).

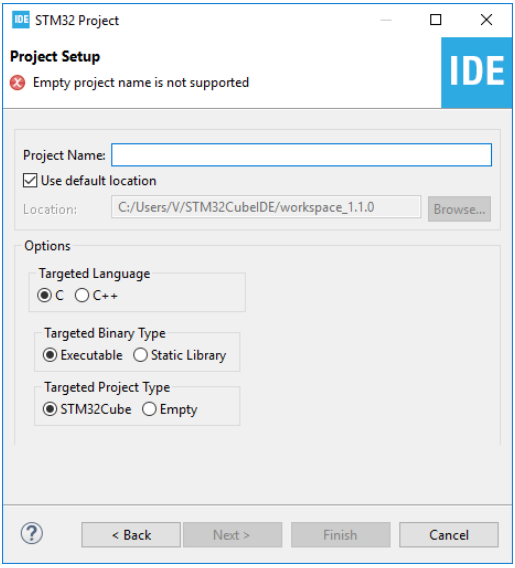



Рис. 2.25 – Начальная настройка проекта в STM32CubeIDE

После завершения начальной настройки будет выдано сообщение о том, что данный тип проектов связан с перспективой в STM32CubeMX (рисунок 2.26).

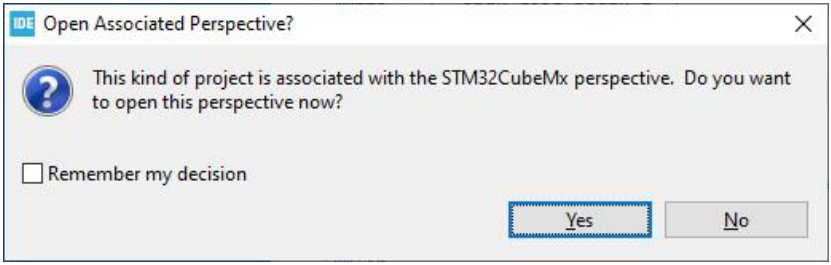



Рис. 2.26 – Окно открытия ассоциированной перспективы

Необходимо выбрать «Yes», чтобы отобразилась перспектива «Device Configuration Tool», интерфейс IDE будет аналогичен интерфейсу редактирования проекта в STM32CubeMX. Однако в отличие от STM32CubeMX интерфейс STM32CubeIDE в правой части дополнительно содержит обозреватель всех открытых проектов («Project Explorer») рабочего пространства, в верхней части располагается панель инструментов (рисунок 2.27).

Тип проекта, созданного мастером проектов, зависит от семейства устройств. Проект содержит все необходимые элементы встроенного приложения, готовые для программирования и запуска.

Если в рабочей области существует несколько проектов, их можно закрыть, щелкнув правой кнопкой мыши на имя проекта в представлении «Project Explorer», а затем выбрать «Close Project». Открыть проект можно аналогично, выбрав **File → Open Projects from File System…**.

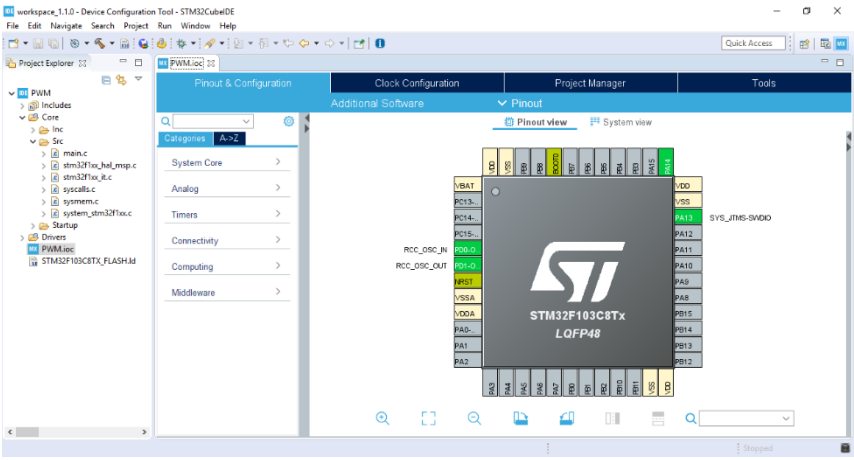



Рис. 2.27 – Начальный вид проекта в STM32CubeIDE

Чтобы сгенерировать проект можно воспользоваться инструментом  («Device Configuration Tool Code Generation»), доступным из панели управления при активной перспективе «Device Configuration Tool» (рисунок 2.28).


Рис. 2.28 – Перспектива «Device Configuration Tool»

На рисунке 2.29 показан пример сгенерированного код-генератором проекта. Видна его структура, а также используемые библиотеки HAL-функций. Следует заметить, что к проекту подключаются библиотеки HAL, используемые для работы с подключенным и сконфигурированным оборудованием.

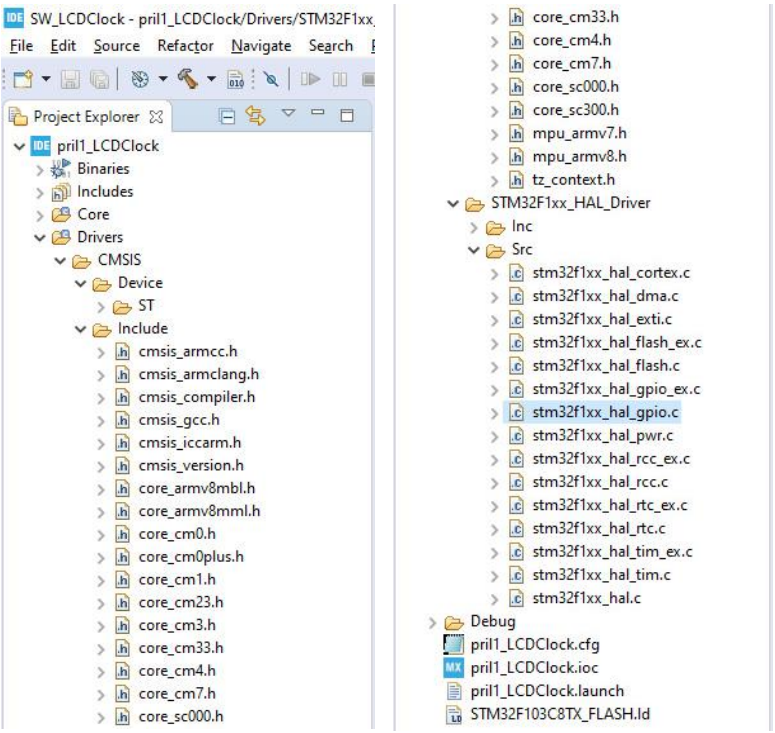



Рис. 2.29 – Список заголовочных файлов HAL, подключенных к проекту

Функции, реализованные в файлах, можно посмотреть на вкладке «Outline» (см. рисунок 2.31). При этом должна быть активна перспектива «C/C++» (рисунок 2.30).


Рис. 2.30 – Перспектива «C/C++»

Для просмотра функций, реализованных в библиотеке, необходимо чтобы в центральной рабочей зоне был открыт интересующий файл.

Если в списке «Outline» выбрать определение или функцию, среда разработки спозиционирует курсор на реализации функции в библиотеке. Для примера на рисунке 2.31 показана реализация функции **HAL_GPIO_ReadPin**:

```
/**
  * @brief  Reads the specified input port pin.
  * @param  GPIOx: where x can be (A..G depending on device used)
to select the GPIO peripheral
  * @param  GPIO_Pin: specifies the port bit to read.
  *         This parameter can be GPIO_PIN_x where x can be (0..15).
  * @retval The input port pin value.
  */
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx,
uint16_t GPIO_Pin)
```

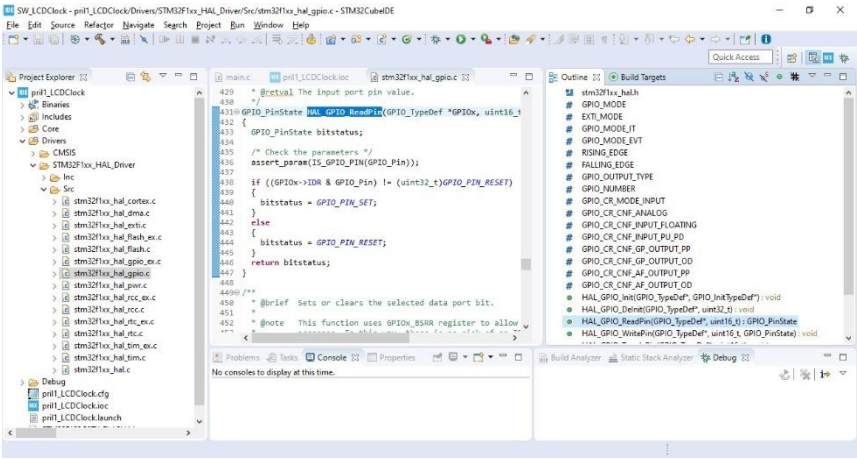
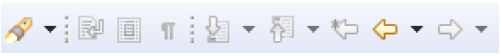



Рис. 2.31 – Функции, реализованные в файле stm32f1xx_hal_gpio.c

Из описания видно, что функция отвечает за чтение состояния указанного разряда заданного порта. В качестве параметров в функцию передаются: GPIOx – порт (возможные варианты: GPIOA … GPIOG) и GPIO_Pin – номер вывода (варианты: GPIO_PIN_0 …

GPIO_PIN_15). В качестве результата функция возвращает PinState – состояние вывода. Состояние цифрового вывода может принимать значения: GPIO_PIN_SET – высокий уровень; GPIO_PIN_RESET – низкий уровень.

Все значки элементов панели управления имеют всплывающие подсказки, которые появляются при наведении указателя мыши. Значки являются ярлыками для функций, которые можно активировать из пунктов главного меню. Эти значки управляют определенными функциями, относящимися к редактированию кода, созданию и управлению проектами:

 Значок фонарика запускает различные поисковые утилиты; стрелки позволяют перемещаться между недавно посещенными местами в проекте («Search» и «Navigate» в опциях главного меню).

 Используйте этот значок для создания нового модуля исходного кода C, файлов заголовков или нового объекта, такого как проект, библиотека или репозиторий (**File → New** из параметров главного меню).

 Чтобы собрать свой проект используйте этот значок.

 Нажав на стрелку, можно запустить определенную конфигурацию отладки или настроить её (функции можно активировать в пункте «Run» в главном меню).

В качестве зонда (средства программирования и отладки) («Debug probe», см. рисунок 2.32) среда предоставляет: ST-LINK (ST-LINK GDB server), SEGGER J-LINK и ST-LINK (OpenOCD). **Для используемого программатора (ST-Link V2) необходимо выбрать ST-LINK (OpenOCD). Прошивку программатора обновлять не рекомендуется, поскольку это может привести к потере работоспособности устройства.**

В качестве интерфейса отладки рекомендуется использовать интерфейс Serial Wire Debug (SWD). Он должен быть включен в диалоговом окне конфигурации отладки. Чтобы перейти к настройкам SWD, нажмите на вкладку «Debugger» в соответствующей конфигурации отладки (**Run → Debug Configurations**), и включите SWD (см. рисунок 2.32). Если внесены изменения в настройки (диалоговое окно свойств конфигурации запуска), их необходимо сохранить, нажав кнопку «Apply».

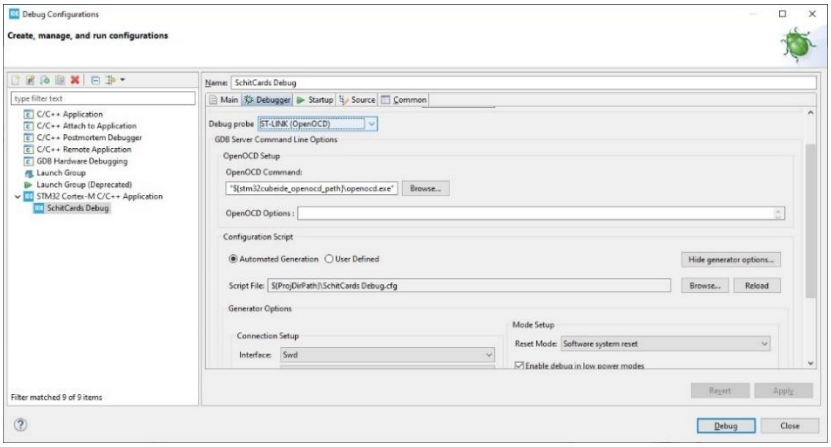



Рис. 2.32 – Окно выбора интерфейса отладки Serial Wire Debug

В начале сеанса отладки запускается драйвер отладчика и сервер OpenOCD, а также приложение и вспомогательные модули, вызываемые в сценарии запуска. На этом этапе приложение обычно останавливается в первой строке функции main ().

Основные значки управления отладчиком:

 Возобновить выполнение приложения (выделено серым цветом при запуске).

 Остановить выполнение (выделено серым цветом при остановке).

 Шаг в функцию, через функцию или из функции.

 Переключение между C и пошаговой инструкцией.

 Перезагрузить чип и перезапустить выполнение.

 Завершить сеанс отладки.

Чтобы установить точку останова, нажмите на синюю горизонтальную полосу рядом с номером строки в коде. Основной исполняемый код вносится в файл «main.c», располагаемый в папке **Core → Src** (справедливо в случае автоматической генерации проекта код-генератором). Для просмотра областей памяти используются соответствующие окна.

По умолчанию при компиляции создается elf-файл. Для формирования hex-файла необходимо перейти в меню **Project → Properties**, далее в древовидной структуре выбрать раздел **C/C++ Build → Settings**, во вкладке «Tool Settings» в разделе «MCU Post build outputs» выбрать опцию «Convert to Intel Hex file (-O ihex)» и нажать кнопку «Apply and Close» (рисунок 2.33). После выполнения указанных настроек при компиляции дополнительно к выбранным будет формироваться hex-файл.

В режиме отладки содержание регистров ядра отражается во вкладке «Registers», а регистров специального назначения (Special Function Registers) во вкладке SFRs (рисунок 2.34). Принудительное чтение содержания выбранного регистра осуществляется по нажатию кнопки «RD».

Значения переменных или выражений можно видеть в реальном времени во вкладке «Live Expressions» (рисунок 2.35). Чтобы добавить переменную для просмотра, необходимо нажать на «Add new expression» и ввести имя переменной.

При этом существует возможность анализа используемой памяти в задачах оптимизации ресурсов с помощью «Build

Analyzer», который доступен через меню **Window → Show View → Build Analyzer**.

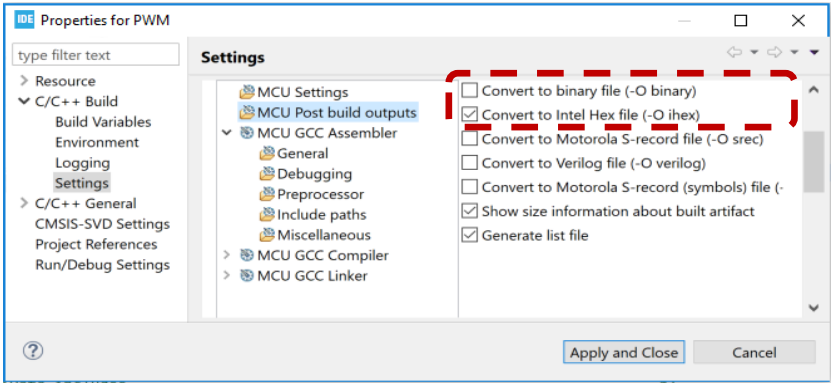



Рис. 2.33 – Настройка результатов компиляции

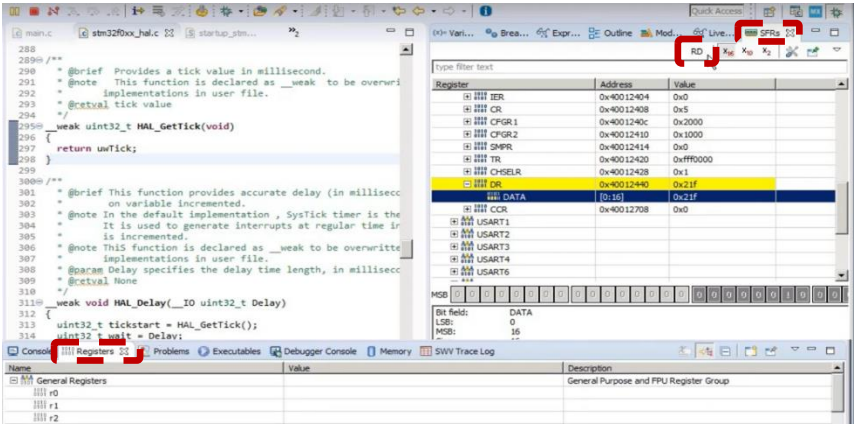


Рис. 2.34 – Просмотр содержимого регистров

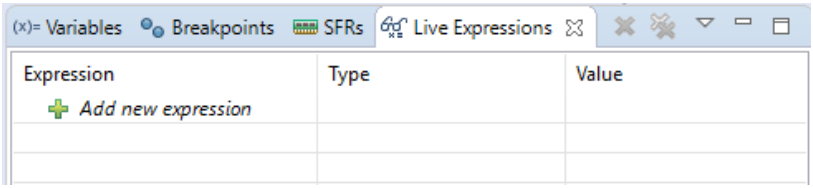



Рис. 2.35 – Просмотр значений переменных

Таким образом, работа в STM32CubeIDE мало чем отличается от работы в SW4STM32 + STM32CubeMX. При этом пользователю не нужно переключаться между двумя инструментами и все операции проводятся в одной IDE, что позволяет сократить время разработки и снизить количество ошибок.

По этой же причине в настоящем пособии не рассматривается поддерживаемая STMicroelectronics среда разработки winIDEA Open, т.к. она, так же как и SW4STM32, является отдельным инструментом. Вместе с тем, читатель может самостоятельно бесплатно загрузить и ознакомиться с данной IDE, обратившись к ресурсам [46, 10].

В отличие от Keil [8] доступна версия STM32CubeIDE для установки в операционной системе Linux, что позволяет снизить риск блокировки работы при ограничениях, связанных с географическим расположением региона, в котором ведется разработка продукта.

Несмотря на указанные преимущества, STM32CubeIDE является новой средой разработки (первая версия стала доступна весной 2019). В связи с этим, данной среде, как и остальным новым программным продуктам, присущи ограниченный функционал и периодически возникающие в процессе работы ошибки. Данные обстоятельства явились основанием для использования SW4STM32 в качестве основной среды разработки при рассмотрении примеров программирования микроконтроллера в настоящем пособии.

### 2.1.4 Proteus Design Suite

Proteus Design Suite – пакет программ, разработанный компанией Labcenter Electronics, предназначенный для автоматизированного проектирования электронных схем. Это система схемотехнического моделирования, базирующаяся на моделях электронных компонентов, принятых в PSpice (Personal Simulation Program with Integrated Circuit Emphasis – программа симуляции аналоговой и цифровой логики).

Отличительной чертой пакета PROTEUS VSM является обширная библиотека моделей электронных и электромеханических устройств (десятки тысяч) [66]. В Proteus реализована концепция сквозного проектирования, при которой изменения в логике работы схемотехники приводят к изменениям в системе трассировки. Дополнительно в пакет PROTEUS VSM входит система проектирования печатных плат. Пакет Proteus состоит из двух частей, двух подпрограмм: ISIS – средства разработки и отладки электронных схем в режиме реального времени и ARES – программа разработки печатных плат [16].

Последние версии пакета программ Proteus (версия 8.7 или более новая) содержат модели микроконтроллеров STM32. На момент написания учебного пособия доступно несколько микропроцессоров из серии STM32F103 и, как показало тестирование, в моделях присутствуют ошибки.

Наиболее близок по своим характеристикам к STM32F103C8T6, который используется в отладочном модуле, микроконтроллер STM32F103C6xx.

На официальном сайте компании Labcenter Electronics (http://www.labcenter.com) доступна демоверсия продукта, однако, она имеет существенные ограничения: отсутствует опция сохранения проекта, в реальном времени симулируются лишь примеры из прилагаемой папки «Samples».

В качестве примера проектирования и отладки проекта, созданного с помощью STM32CubeIDE и PROTEUS DESIGN SUITE 8.10 SP2, рассмотрим задачу измерения напряжения 4 каналов с помощью АЦП и передачу результатов через последовательный порт USART на скорости 19200 бит/сек в программном асинхронном режиме.

Для создания проекта необходимо нажать «New Project» в панели управления или выполнить пункт меню **File → New Project** (рисунок 2.36).

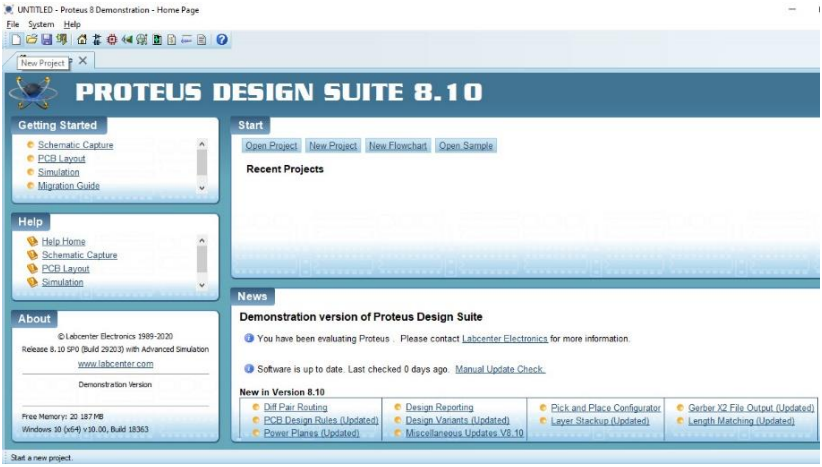



Рис. 2.36 – Стартовое окно Proteus

В открывшемся окне задать имя проекта и место его расположения (рисунок 2.37).

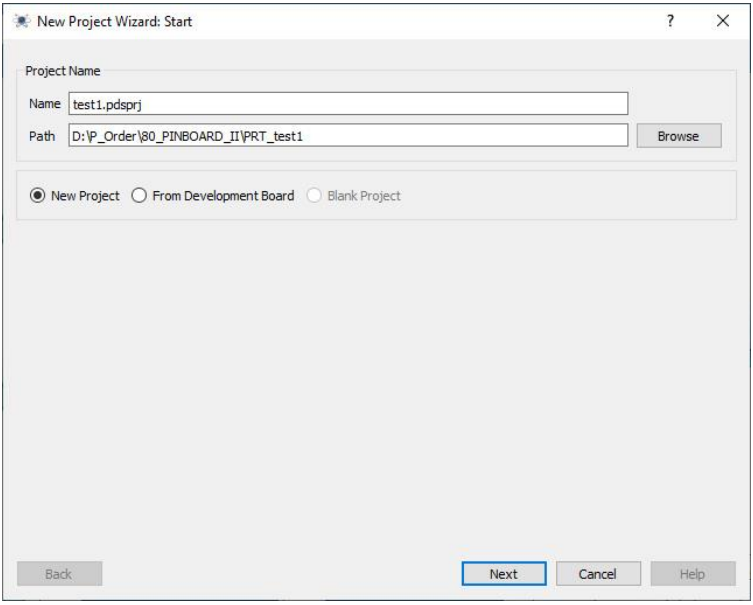



Рис. 2.37 – Указание имени проекта

Далее задается формат эскизного проекта (рисунок 2.38).

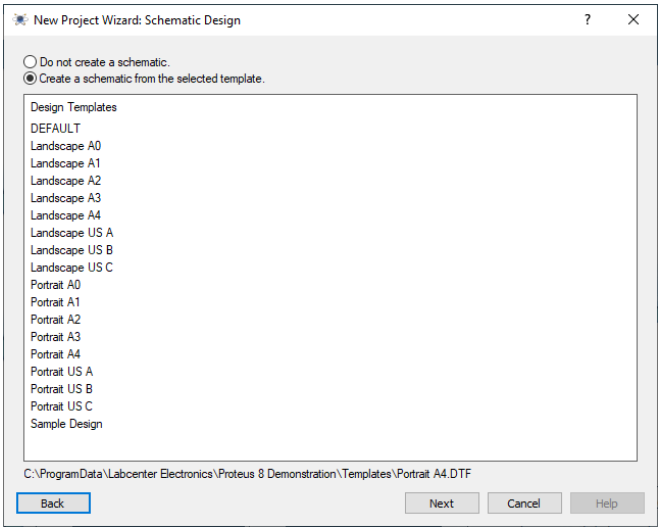



Рис. 2.38 – Формат эскизного проекта

Макет печатной платы создавать не нужно (рисунок 2.39).

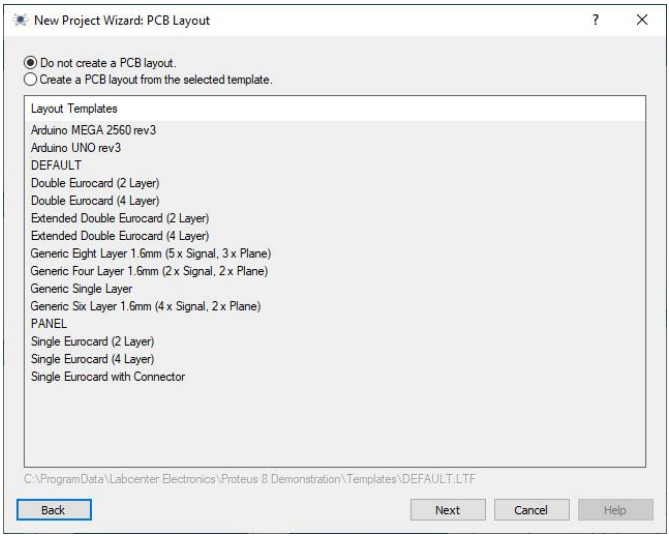



Рис. 2.39 – Отказ от разработки печатной платы

В окне Firmware необходимо выбрать контроллер STM32F103C6 семейства Cortex-M3, в параметре «Compiller» указать компилятор «GCC for ARM (not configured)» (рисунок 2.40).

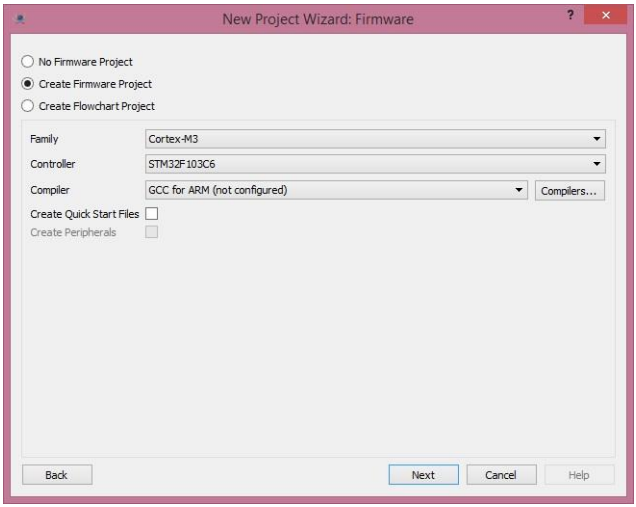



Рис. 2.40 – Выбор типа микроконтроллера и компилятора

При первом запуске необходимо скачать и установить компилятор, нажав на кнопку «Comilers» (рисунок 2.41).

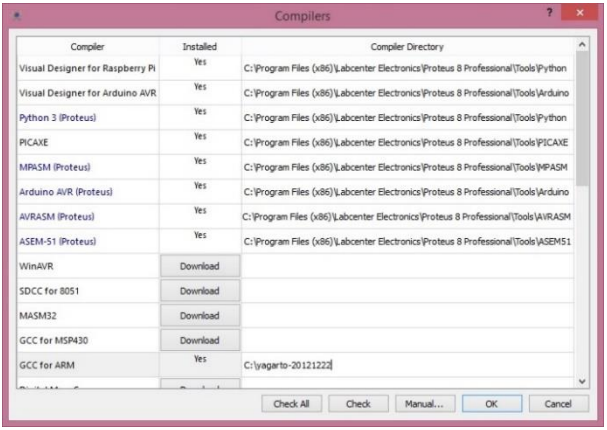



Рис. 2.41 – Установка компилятора

После успешной установки компилятора параметр «Compiller» изменит название на «GCC for ARM» (рисунок 2.42).

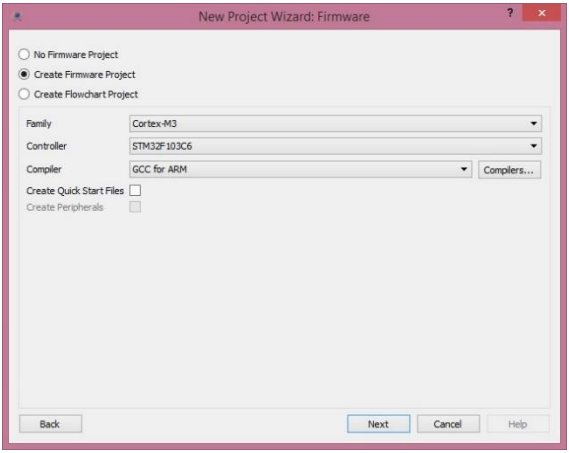



Рис. 2.42 – Настройка проекта

Необходимо снять выделение с параметра «Create Quick Start Files» (рисунок 2.42). На этом процесс создания проекта завершается и можно нажать кнопку «Finish» (рисунок 2.43). На следующем этапе необходимо разработать электрическую схему (рисунок 2.44).

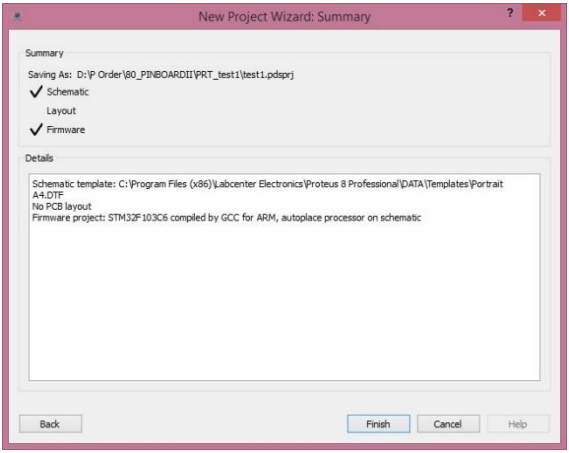



Рис. 2.43 – Окончание процесса создания проекта

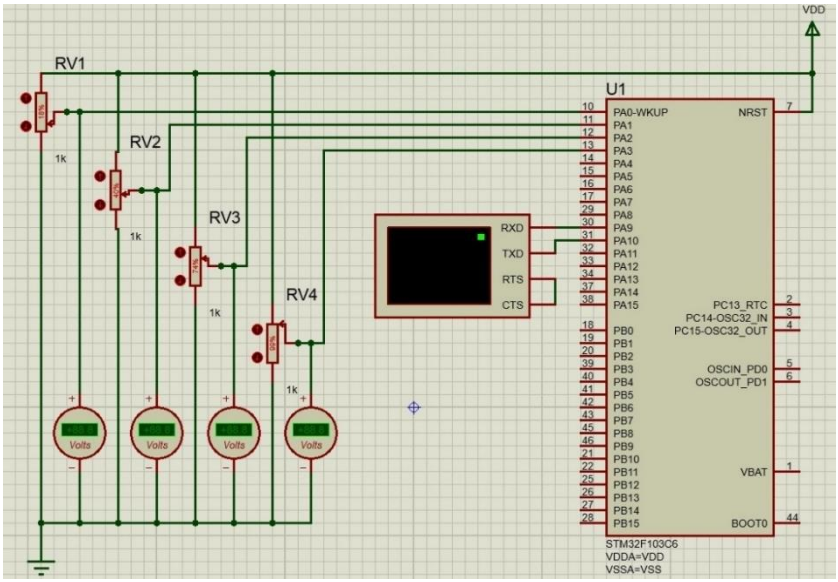



Рис. 2.44 – Электрическая схема проекта

Схема собрана, далее необходмо создать проект в STM32CubeIDE или другой среде разработки (см. раздел 2.13).

При создании проекта из перечня контроллеров необходимо выбрать STM32F103C6Tx и нажать «Next» (рисунок 2.45).

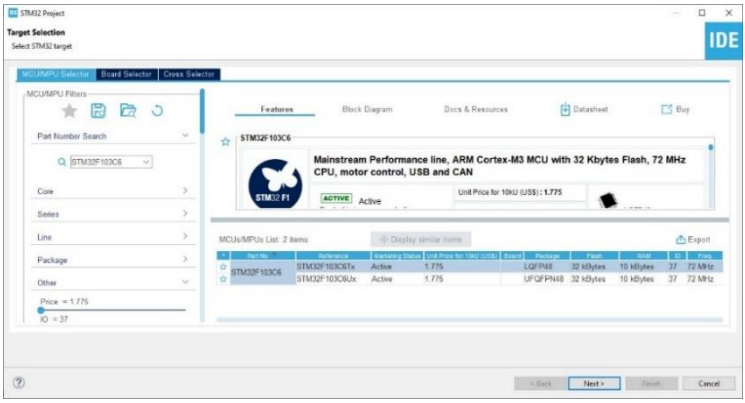



Рис. 2.45 – Выбор микроконтроллера

Необходимо определить название проекта и указать язык программирования (рисунок 2.46).

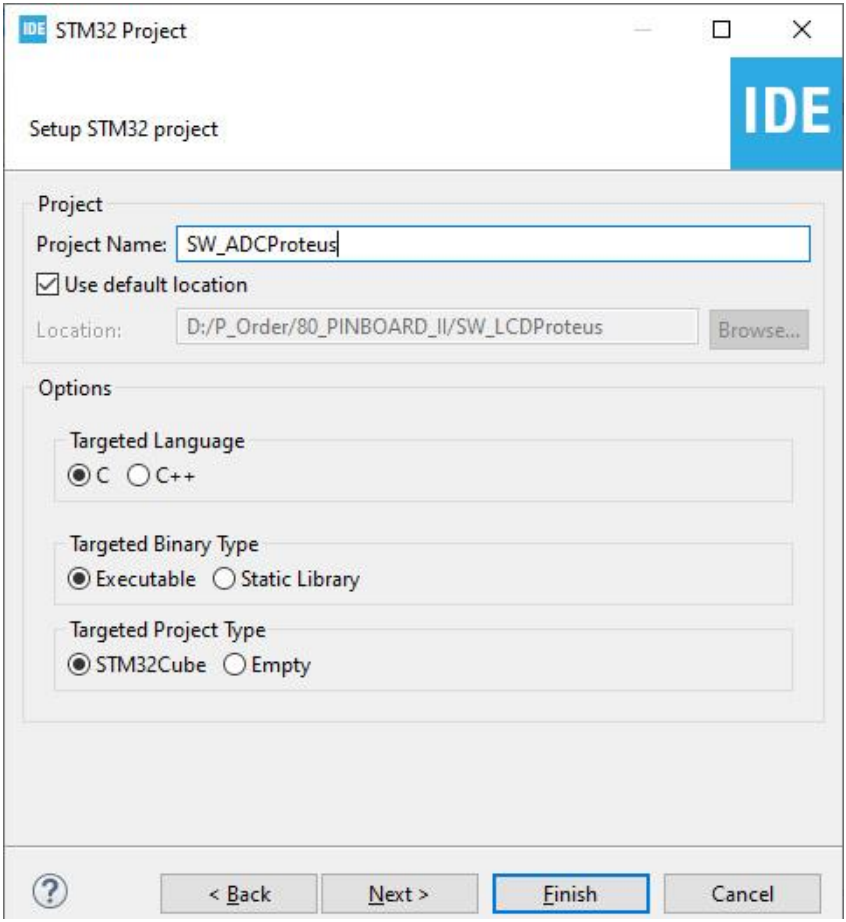



Рис. 2.46 – Выбор микроконтроллера

Во вкладке «Pinout & Configuration» необходимо задать параметры АЦП (ADC1): выбрать 4 первых канала и выставить параметры как показано на рисунке 2.47.

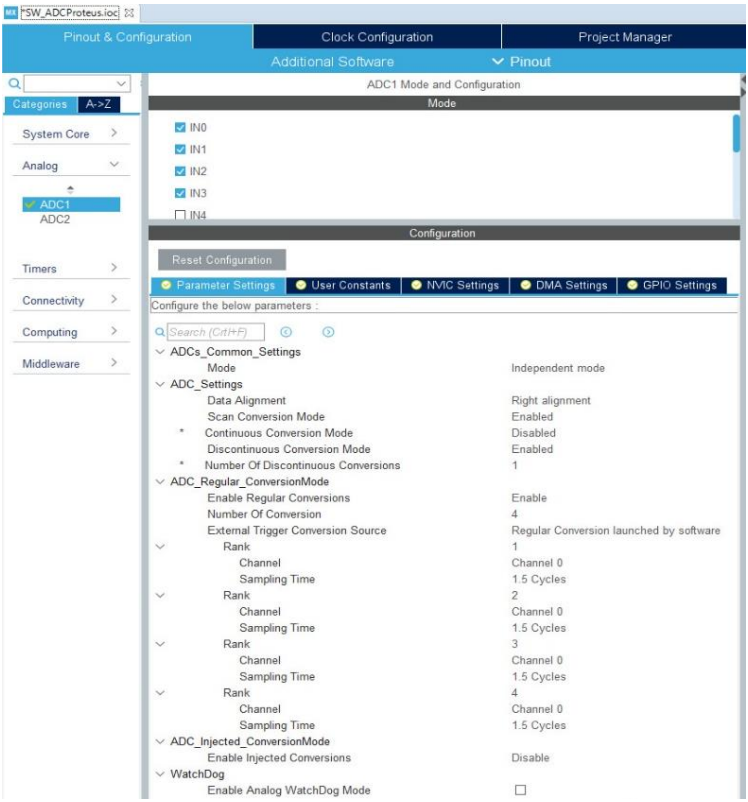



Рис. 2.47 – Настройка параметров АЦП

Параметры АЦП означают следующее (см. раздел 1.6.6):

**Mode** – режим: независимый или каскадный (ведущий – ведомый).

**Data Alignment** – выравнивание данных по левому/правому краю регистра данных.

**Scan Conversion Mode** – режим сканирования. Становится активным, когда опрашиваются несколько каналов на одном АЦП (см. параметр **Number Of Conversions**).

**Continious Conversion Mode** – опрос канала/каналов производится циклически (перезапуск не требуется).

**Discontinuous Conversion Mode** – режим позволяет сканировать несколько каналов (на одном АЦП) так, чтобы опрос происходил не по всем каналам, а по заранее заданным группам каналов. **Number Of Discontinuous Conversions** – количество каналов в одной группе.

**Enable Regular Conversions** – каналы настроены как регулярные (Enable – активировать параметр).

**Number Of Conversions** – количество каналов для опроса.

**External Trigger Conversion Source** – событие, которое будет запускать АЦП (выбран программный запуск преобразования).

**Rank** – порядок опроса и индивидуальное время преобразования каналов Sampling Time, которые указываются в тактах частоты генератора АЦП.

**Channel** – номер канала.

Далее необходимо настроить модуль USART (раздел 1.6.5) в разделе «Connectivity», как показано на рисунке 2.48.

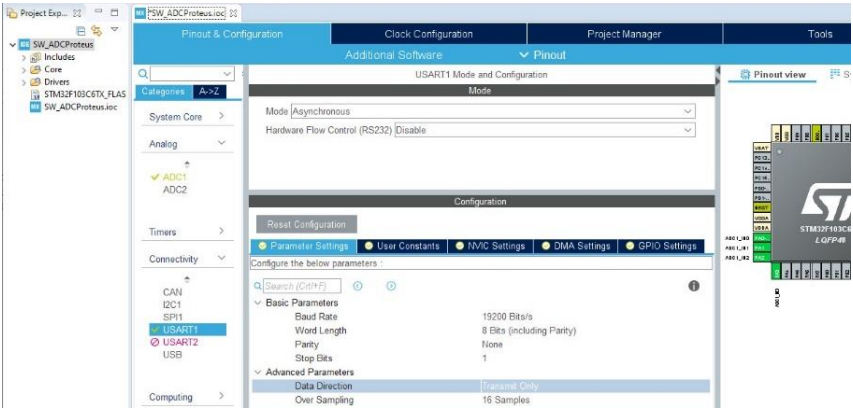



Рис. 2.48 – Настройка параметров USART

Во вкладке «Clock Configuration» необходимо настроить тактирование АЦП (рисунок 2.49). Частота тактирования АЦП не должна превышать 14 МГц.

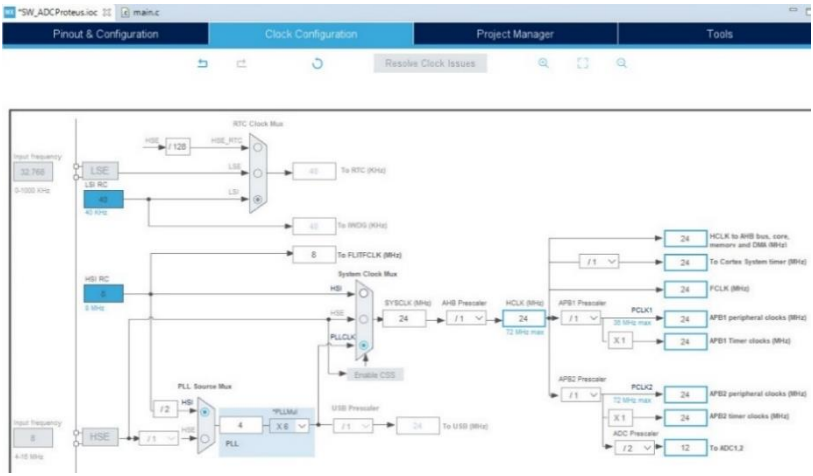



Рис. 2.49 – Тактирование периферии

После генерации проекта в программу следует добавить необходимые заголовочные файлы:

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
#include "string.h"
/* USER CODE END Includes */
```

Объявить переменные:

```
/* USER CODE BEGIN PV */
char strTransmit[64] = {0,};
volatile uint16_t adcCodes[4] = {0,}; //массив для
хранения измеренных значений
/* USER CODE END PV */
```

В разделе /* USER CODE BEGIN 2 */ вызвать функцию калибровки (реализована в файле stm32f1xx_hal_adc_ex.c):

```
  /* USER CODE BEGIN 2 */
  HAL_ADCEx_Calibration_Start(&hadc1);
  /* USER CODE END 2 */
```

В основном цикле программы реализовать опрос АЦП и вывод результатов через USART:

```
    /* USER CODE BEGIN 3 */
    // Канал 1
       // запуск АЦП
       HAL_ADC_Start(&hadc1);
       //дождаться окончания преобразования в группе
регулярных каналов
       HAL_ADC_PollForConversion(&hadc1, 100); // 100 -
таймаут, мс
       //зафиксировать измеренное значение
       adcCodes[0] = HAL_ADC_GetValue(&hadc1);
       // Канал 2
       HAL_ADC_Start(&hadc1);
       HAL_ADC_PollForConversion(&hadc1, 100);
       adcCodes[1] = HAL_ADC_GetValue(&hadc1);
       // Канал 3
       HAL_ADC_Start(&hadc1);
       HAL_ADC_PollForConversion(&hadc1, 100);
       adcCodes[2] = HAL_ADC_GetValue(&hadc1);
       // Канал 4
       HAL_ADC_Start(&hadc1);
       HAL_ADC_PollForConversion(&hadc1, 100);
       adcCodes[3] = HAL_ADC_GetValue(&hadc1);
       // останов АЦП
       HAL_ADC_Stop(&hadc1);

       // отформатировать строку
       snprintf(strTransmit, 63, "An1: %d An2: %d An3: %d
An4: %d\r\n", adcCodes[0], adcCodes[1], adcCodes[2],
adcCodes[3]);
       // вывести результирующую строку через usart
       HAL_UART_Transmit(&huart1, (uint8_t*)strTransmit,
strlen(strTransmit), 1000);
       HAL_Delay(500);
```

**Для генерации корректного elf-файла, который можно использовать при расширенной отладке в Proteus, перед сборкой проекта необходимо произвести настройку компилятора.**

Для этого в контекстном меню «Project Explorer» выбрать раздел «Properties». Далее перейти к «C/C++ Build» → «Settings», затем во вкладке «Tool Settings» открыть «MCU GCC Compiler» → «Miscellaneous». После необходимо добавить опцию -gdwarf-2 –fmessage-length=0 в параметр «Other Flags» и нажать кнопку «Apply» (рисунок 50).

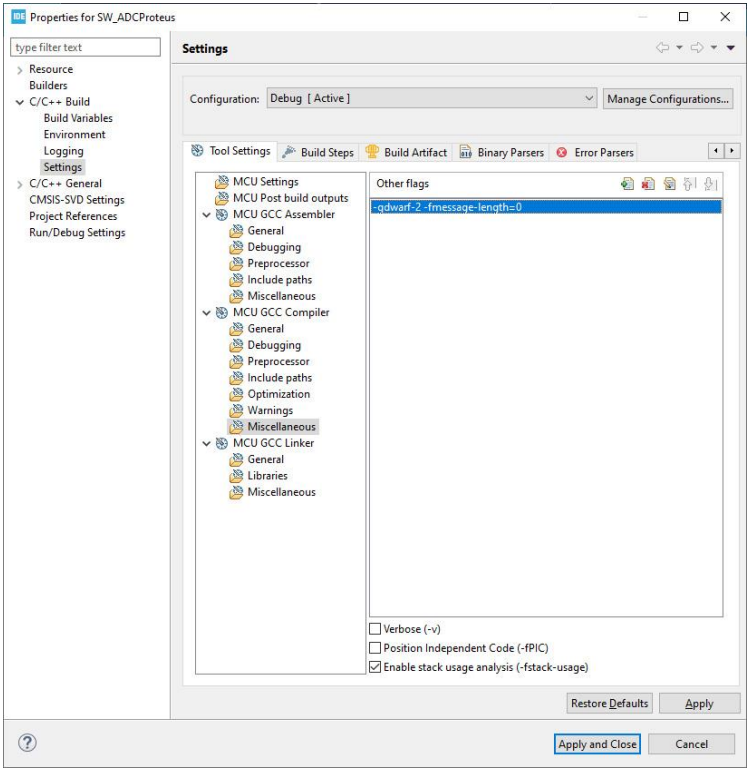



Рис. 2.50 – Опция компилятора для работы в Proteus

Результат компиляции представлен на рисунке 2.51.

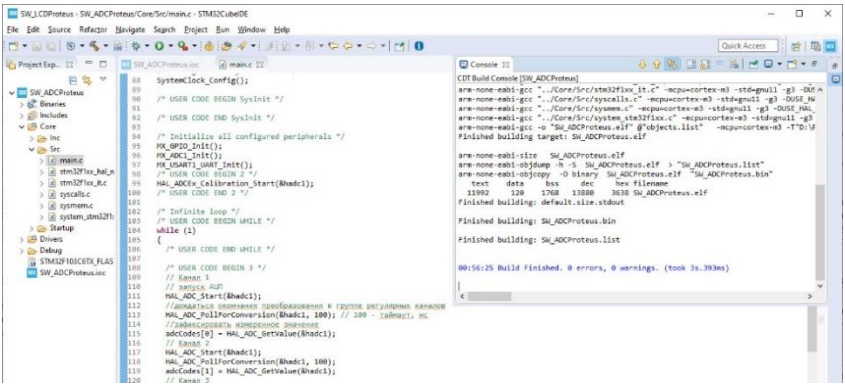


Рис. 2.51 – Результат компиляции для работы в Proteus

Выполнив компиляцию проекта (**Project → Build All**), необходимо в Proteus загрузить elf-файл. Для этого требуется кликнуть правой кнопкой мыши на микроконтроллере, в контекстном меню выбрать пункт «Edit Properties». В окне «Edit Component» задать параметр «Program File» (рисунок 2.52).

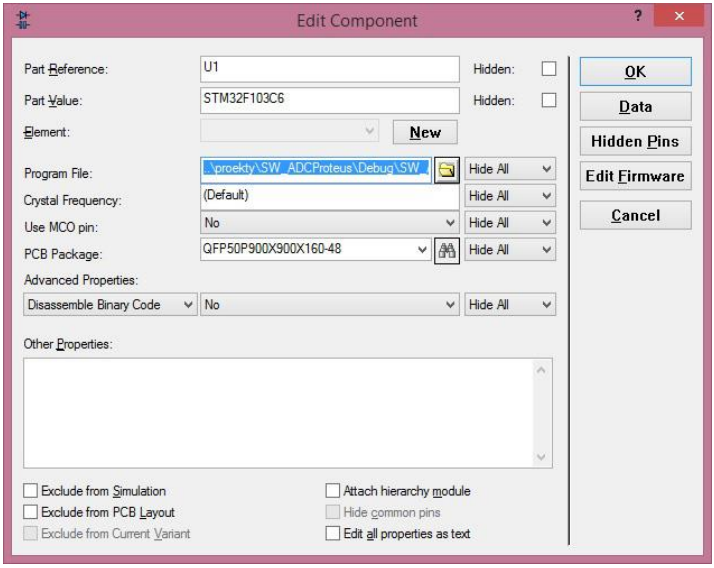



Рис. 2.52 – Запись программы в микроконтроллер

В терминале USART задать параметры, соответствующие заданию (скорость обмена, длину поля данных, количество стоповых бит, особенности использования бита паритета, сигналы управления модемом). Установить на входах значения напряжения, соответствующие диапазону измерения АЦП.

Далее в Proteus следует сохранить проект и запустить отладку, нажав **Debug → Start VSM Debugging.**

Отобразить на экране необходимые для отладки внешние устройства и внутренние ресурсы микроконтроллера: память программ, данных, регистры общего назначения, регистры периферийных устройств (окно Watch Window).

После этого нажатием на кнопку **Debug → Run Simulation** запустить симуляцию.

При обнаружении ошибок в проекте переключиться в пошаговый режим и контролировать состояние программы до устранения ошибок.

Результаты выполнения программы представлены на рисунке 2.53.

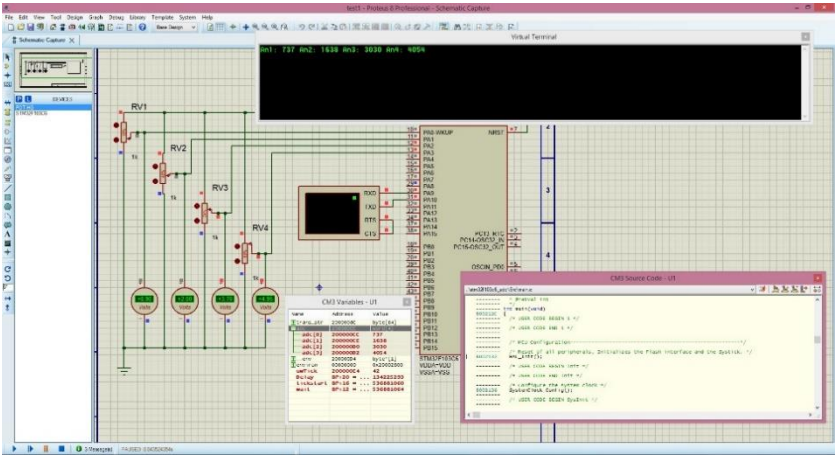



Рис. 2.53 – Окно отладки Proteus

Диапазон преобразуемого аналогового напряжения определяется значением Vdda, так как используемая модель микроконтроллера не допускает присоединения внешнего источника опорного напряжения.

## 2.2 Операционные системы реального времени

В большинстве случаев для программирования микроконтроллера разработчику достаточно внести необходимый код в бесконечно исполняемый цикл (режим «Flat») [52]. Однако данный способ не всегда эффективен и удобен, поэтому часто используются многозадачные операционные системы реального времени (ОСРВ, RTOS).

Операционная система реального времени ОСРВ – это тип операционной системы, способный обеспечить **требуемое (предсказуемое)** время обработки поступающих событий на конкретном оборудовании.

Микроконтроллер STM32F103C8T6 разработан с учетом возможности работы под управлением различными ОСРВ, занимающими небольшой объем в памяти. Использование ОСРВ дает более широкие возможности повторного использования программ, а также более простое управления проектом.

Ядром Cortex-M3 является 32-битное Центральное Процессорное Устройство (ЦПУ), поддерживающее два режима работы: потоковый (Thread) и режим обработчика (Handler). Для каждого режима можно сконфигурировать свои собственные стеки [52]. Эта возможность обычно используется ОСРВ, которые могут выполнять свой «системный» код в защищенном режиме. У двух стеков ЦПУ имеются собственные наименования: основной стек и стек процесса. Благодаря этому, появляется возможность разработки более интеллектуального программного обеспечения и поддержки операционных систем реального времени.

ЦПУ запускается в режиме Thread при непрерываемом, фоновом выполнении инструкций и переключается в режим Handler при обработке исключительных ситуаций. Кроме того, ЦПУ Cortex может выполнять код программы в привилегированном или непривилегированном режимах. В привилегированном режиме ЦПУ имеет доступ ко всему набору инструкций, а в непривилегированном режиме некоторые инструкции отключаются [52].

Сразу после сброса процессор Cortex запускается в конфигурации «Flat» [52]. В обоих режимах, Thread и Handler, инструкции выполняются в привилегированном режиме. Таким образом, какие-либо ограничения на доступ к процессорным ресурсам отсутствуют. В режимах Thread и Handler используется основной стек. Чтобы начать выполнение инструкций достаточно процессору указать вектор сброса и стартовый адрес стека, после чего можно выполнять код программы. Однако, если используется ОСРВ или выполняется разработка критичного к безопасности приложения, процессор может использоваться в более специфичной конфигурации: режим Handler (используется при обработке исключительных ситуаций и операционными системами реального времени) с привилегированными операциями и использованием основного стека; режим Thread при исполнении прикладного кода с непривилегированным доступом и использованием стека процесса. При таком подходе, весь код программы разделяется на системный и прикладной и, поэтому, ошибки прикладного кода не вызывают сбоев ОСРВ.

Существует большое число операционных систем реального времени: Salvo, uClinux, VxWorks, RIOT, Mbed OS, SafeRTOS, FreeRTOS. Одной из популярных ОСРВ с открытым исходным кодом является FreeRTOS, распространяемая бесплатно (https://freertos.org). **Необходимо отметить, что в отличие от**

**сертифицированных, данную ОСРВ нельзя использовать в критически важных областях**, например, при разработке программ для оборудования, применяемого в медицине или авиации. Подробная информация об особенностях работы и возможностях FreeRTOS представлена в руководстве пользователя FreeRTOS [40].

Для настройки работы FreeRTOS в STM32CubeMX предусмотрен специальный раздел, переход к которому осуществляется через слой Middleware. Конфигурирование микроконтроллера на работу с данной операционной системой производится выбором FREERTOS и установкой режима CMSIS_V1 (рисунок 2.54). Выбор версии CMSIS_V1 в качестве стандарта программного интерфейса микроконтроллеров Cortex (Cortex Microcontroller Software Interface Standard) обусловлен поддержкой данной версии большинством микроконтроллеров.

В разделе «Configuration» на вкладке «Config parameters» производится настройка параметров операционной системы. Справочная информация о выбранном параметре становится доступной при щелчке по пиктограмме «i» (рисунок 2.54).

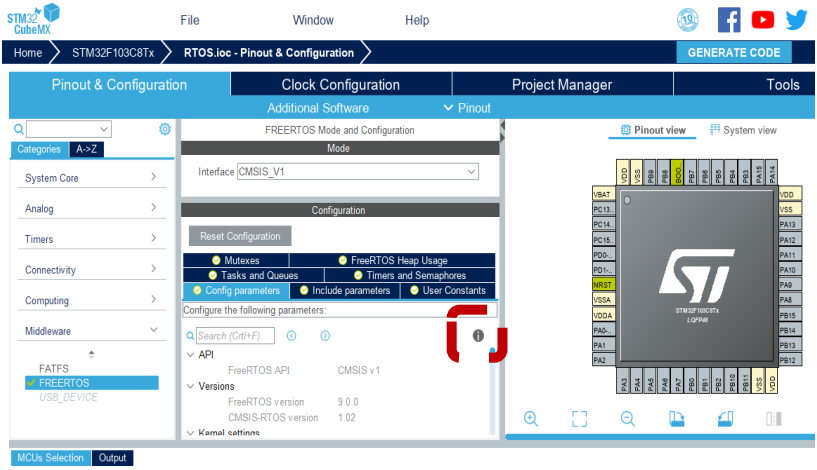



Рис. 2.54 – Окно настройки FREERTOS

Память для операционной системы выделяется из RAM микроконтроллера. Параметр TOTAL_HEAP_SIZE определяет, какой объем памяти доступен операционной системе.

Параметр Memory Management scheme определяет модель организации памяти. Различают следующие модели [12]:

**heap_1** – предназначена только для создания задач, возможность их удаления отсутствует, невозможно повторно использовать область памяти, невозможно дефрагментировать. Модель используется в критически важных областях: медицина, космос и т.д. В этих областях фрагментация памяти неприемлема;

**heap_2** – является устаревшей моделью, используется для совместимости с ранними версиями;

**heap_3** – не выделяет в памяти «кучу», выделение памяти осуществляется разработчиком самостоятельно;

**heap_4** – современная версия heap_2, позволяющая совершать все операции, включая удаление задач. По умолчанию в STM32CubeMX установлена данная модель.

**heap_5** – эта схема использует те же алгоритмы первоначального формирования и объединения памяти, что и heap_4, и позволяет «куче» охватывать несколько несмежных областей памяти.

Параметр **TICK_RATE_HZ** определяет частоту переключения задачи. Может принимать значения от 1 до 1000. Подробнее о переключении задач описано в руководстве по FreeRTOS [19].

Активация параметра **USE_TICK_HOOK** приведет к тому, что в каждый отсчет времени будет вызываться функция vApplicationTickHook(void), которая используется для определения задачи, выполняемой в данный момент. Например, чтобы посмотреть, какая задача выполняется в данный момент.

Активация параметра **USE_IDLE_HOOK** позволит диспетчеру задач вызывать функцию vApplicationIdleHook(void) в

моменты, когда процессор не задействован для выполнения какой-либо задачи. Иногда данную функцию используют для перевода микропроцессора в режим сна или отключения периферии с целью снижения энергопотребления. Эту функцию часто используют для отслеживания загруженности процессора, чтобы узнать справляется ли процессор с поставленными задачами: если, например, затрачиваемое время на **холостую задачу (Idle)** невелико, есть повод задуматься об увеличении тактовой частоты.

Если активировать параметр **USE_TICKLESS_IDLE**, диспетчер задач будет вызывать две функции: PreSleepProcessing и PostSleepProcessing. Данные функции предусмотрены для решения задачи снижения энергопотребления.

Все параметры, указанные на данной вкладке, подробно описаны в руководстве пользователя FreeRTOS [6]. Вместе с тем, во вкладке «Include parameters» можно отключать некоторые функции FreeRTOS, например, с целью снижения размера кода.

В разделе «Configuration» во вкладке «Tasks and Queues» по умолчанию создана задача, в дополнение к которой разработчик может создавать свои задачи. Задача добавляется с помощью кнопки «Add», в появившемся окне задаются параметры новой задачи (рисунок 2.55).

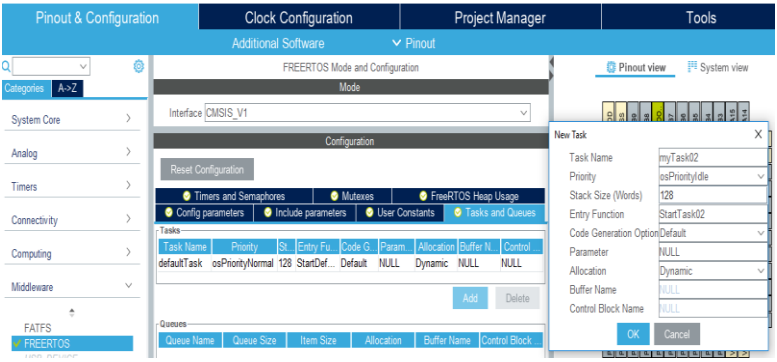



Рис. 2.55 – Окно добавления задачи

Для хранения параметров задачи предусмотрен стек. Размер стека определяется в словах (4 байта) при создании задачи. Из общей кучи выделяется часть памяти для размещения блока управления задачей (Task Control Block, TCB) и стека.

Как и во всех остальных случаях для работы микроконтроллера необходимо произвести настройку тактирования, например, на внешний кварц **RCC → RCC Mode and Configuration → Mode → High Spped Clock (HSE)**, необходимо выбрать «Crystal / Ceramic Resonator». В разделе «Clock Configuration» необходимо настроить систему тактирования (см. рисунок 2.56).

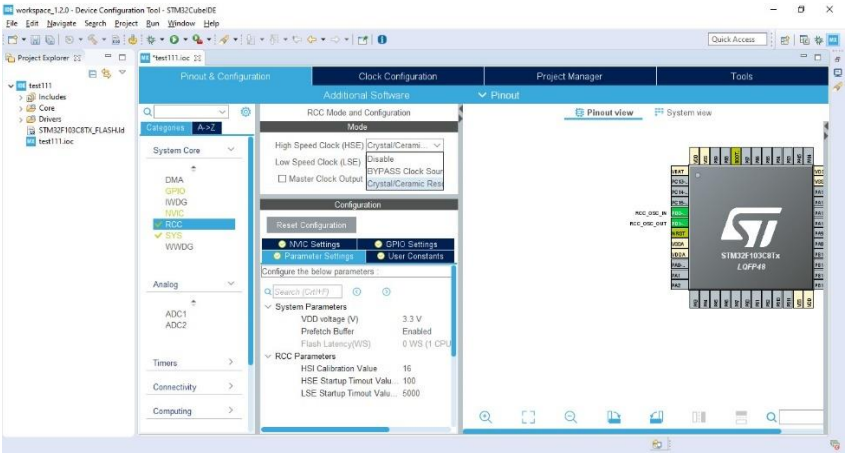



Рис. 2.56 – Задание режима тактирования

Для формирования шкалы времени ОСРВ или генерации периодических прерываний можно использовать системный таймер SysTick (см. раздел 1.3) [52].

Однако, в случае генерации кода в среде STM32CubeIDE будет выдано предупреждение о том, что при использовании RTOS рекомендуется вместо SysTick использовать иной источник импульсов (рисунок 2.57).

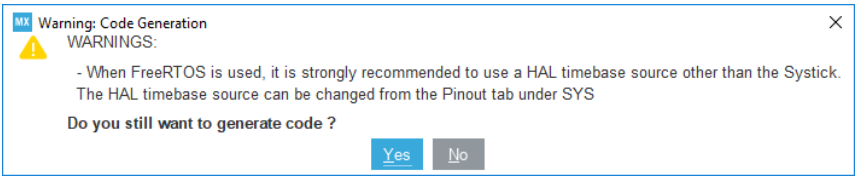



Рис. 2.57 – Предупреждение об использовании SysTick в библиотеке HAL

Предупреждение вызвано тем, что таймер SysTick занят ресурсами библиотеки HAL, в которой реализовано много функций, использующих системный таймер. Данное обстоятельство можно отнести к недостаткам использования HAL.

Изменить источник импульсов можно в разделе **SYS → Mode**, задав параметр «Timebase Source». Например, выбрав незадействованный таймер, который будет порождать прерывания, вызывающие диспетчер задач. Диспетчер будет переключать выполнение задач, предварительно ранжируя их по приоритету. Необходимо помнить, что подключение периферии приведет к дополнительному потреблению энергии.

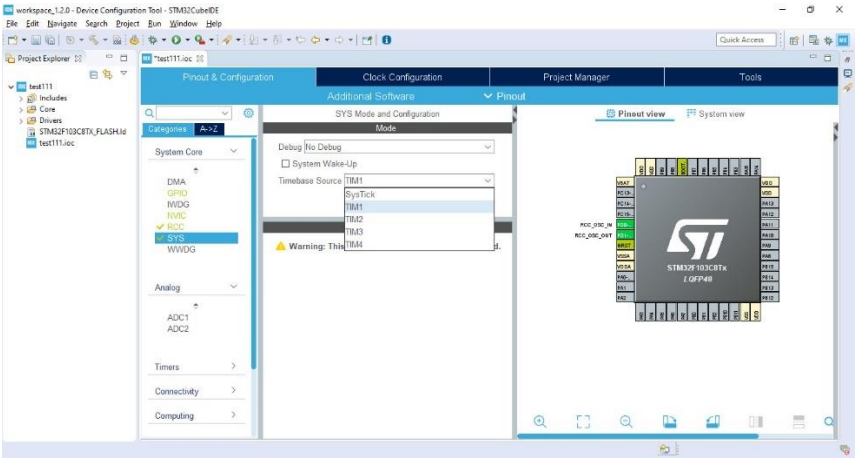



Рис. 2.58 – Выбор источника импульсов

После генерации кода с помощью код-генератора STM32CubeMX в среде разработки сгенерируется проект, «main.c» которого содержит:

- включение заголовочного файла «cmsis_os.h»:

```
#include "cmsis_os.h";
```

- в разделе «Private variables» указываются обработчики задач:

```
osThreadId defaultTaskHandle;
osThreadId Task02Handle;
```

- в разделе «Private function prototypes» объявлены функции:

```
static void MX_GPIO_Init(void);
void StartDefaultTask(void const * argument);
void Task2_Init(void const * argument);
```

- в раздел «Initialize all configured peripherals» осуществляется вызов функции инициализации GPIO:

```
MX_GPIO_Init();
```

- раздел «Create the thread(s)», в котором определяются и создаются сконфигурированные задачи. Изначально задача не инициализирована (не создана), есть лишь скомпилированный код, расположенный во flash-памяти и известен его адрес. В нужный момент задача создается.

```
/* definition and creation of defaultTask */
osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

/* definition and creation of Task02 */
osThreadDef(Task02, Task2_Init, osPriorityNormal, 0, 128);
Task02Handle = osThreadCreate(osThread(Task02), NULL);
```

В результате выполнения, приведенного выше кода, под каждую задачу выделяется часть памяти, задается свой стек, и она запускается.

- Раздел «Start scheduler», содержит функцию osKernelStart(), в которой запускается диспетчер (планировщик) задач. Планировщик перехватывает на себя управление и, как следствие, следующий за данным разделом бесконечный цикл while(1) исполняться не будет.

- Реализации функций для сконфигурированных задач, в которых разработчик реализует необходимый программный код (алгоритм):

```
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_Task2_Init */
/**
* @brief Function implementing the Task02 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_Task2_Init */
void Task2_Init(void const * argument)
{
  /* USER CODE BEGIN Task2_Init */
  /* Infinite loop */
  for(;;)
  {
```

```
    osDelay(1);
  }
  /* USER CODE END Task2_Init */
}
```

Функция osDelay() сообщает диспетчеру операционной системы о том что задаче, из которой был осуществлен вызов функции osDelay(), можно не выделять процессорное время. То есть задачу можно пропустить и в течение времени, указанного в качестве параметра, не вызывать. Интервал времени задается в миллисекундах. При использовании osDelay() планировщик задач выгружает контекст текущей задачи и загружает другую задачу, если такая имеется. Планировщик постоянно осуществляет переключения между задачами. При вызове osDelay() диспетчер определяет, что задаче в течении указанного времени не нужно передавать управление, а ОС меняет статус задачи на заблокированный.

Функция HAL_Delay() существенно отличается от osDelay(). В ней пауза выполняется внутри самой задачи, за счет цикла опроса текущего времени. При этом остальные задачи в системе продолжают выполняться, но не образуется свободного (невостребованного) времени. Если одна или несколько задач сообщат диспетчеру, что есть свободное время, он не поделит его на другие задачи, а потратит на вызов Idle процесса.

Задача может находиться в одном из перечисленных состояний (см. рисунок 2.59):

- **READY**. Задача запущена и готова принять на себя управление. Как только подойдет ее очередь, сразу перейдет в режим RUN.
- **RUN**. Диспетчер переключил управление на задачу, процессор работает непосредственно по её коду.
- **BLOCKED**. Задача ожидает событие, на которое она должна отреагировать. При этом диспетчер не переключается на

нее, процессорное время не тратится. Как только ожидаемое событие произойдет, RTOS назначит задаче состояние READY.

- **SUSPEND**. Выключено, то есть задача не выгружена из памяти, но она не активна и все её данные сохранены. Ни на какие события задача не реагирует, сама из этого состояния выйти не сможет. Для изменения состояния можно использовать API команду ОС.

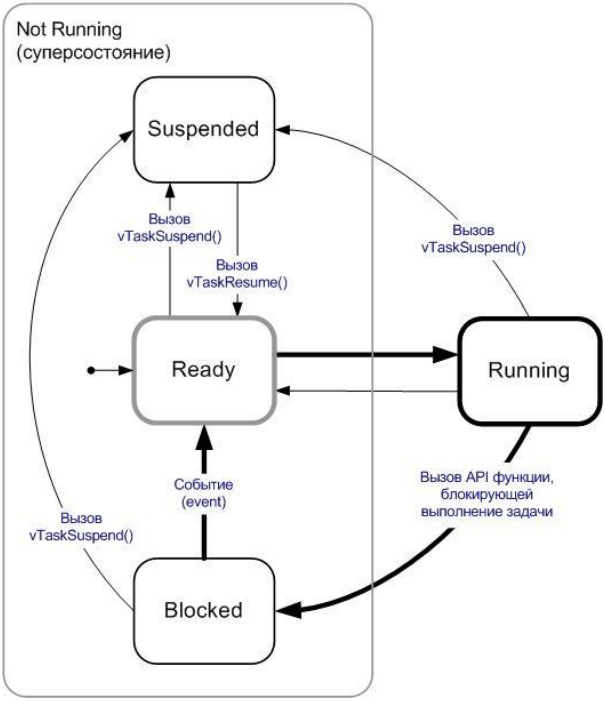



Рис. 2.59 – Граф состояния задач

Задачу можно удалить специальной командой, при этом она освободит RAM, а ее текущее состояние и локальные переменные будут потеряны. Впоследствии задачу можно запустить вновь.

Приоритет определяет в каком порядке будут выполняться задачи. Если есть две задачи в статусе READY, но у одной приоритет

выше другой, то задача с низким приоритетом не получит управление до тех пор, пока высокоприоритетная задача не перейдет в состояние BLOCKED. Диспетчер для исполнения выбирает ту задачу в состоянии READY, у которой приоритет выше.

Определить, достаточен ли размер стека для задачи, позволяет функция vTaskList(char *pcWriteBuffer), которой в качестве параметра передается символьный буфер. В этот буфер функция вернет список задач с указанием имени, состояния, приоритета, минимального объема свободной области стека, который был у задачи с момента старта диспетчера задач [38], а также уникальный номер, присвоенный каждой задаче.

Также доступна функция xPortGetFreeHeapSize(), которая возвращает значение, соответствующее объему свободного места в куче. Указанные функции крайне полезны на этапе отладки.

В связи с тем, что задачи в указанных условиях выполняются асинхронно и диспетчер может прервать каждую из них в любой момент, может возникнуть ситуация, когда задача прерывается в момент записи данных в глобальную переменную. При этом данные переменной будут записаны не полностью и по сути ошибочны.

Вместе с тем, другая задача, для которой эти данные предназначались, прочтёт их и начнет обработку. Результат такой обработки не будет иметь ценности, т.к. обработаны некорректные данные.

В случае если в задаче имеется часть кода, приостанавливать исполнение которого нельзя (например, из-за передачи диспетчером задач управления от одной задачи к другой), целесообразно использовать функции taskENTER_CRITICAL() – перед началом части критического кода и taskEXIT_CRITICAL() – по его окончании.

Чтобы не запрещать прерывания и не останавливать диспетчер, вместо глобальных переменных предусмотрен обмен данными

между задачами с помощью так называемых очередей, работу с которыми также организует планировщик.

Диспетчер следит за тем, чтобы из очереди нельзя было прочитать до того, как в нее все корректно запишется. Также он следит за тем, чтобы в очередь нельзя было записать новые данные если она переполнена. Если очередь пуста/переполнена, задача, которая хочет считать/записать данные, переводится в состояние BLOCKED, и диспетчер вернет ее в состояние READY, когда очередь будет готова отдать/принять данные.

Попутно решается вопрос приоритетов: при разблокировке готовыми к исполнению станут ожидающие очереди задачи и наиболее приоритетные из них выполнятся в первую очередь.

Перед использованием очереди ее необходимо создать, а также указать количество и тип элементов. Если очередь стала не нужна, ее можно удалить, освободив память. Список основных API функций для работы с очередями представлен в [40].

Очереди (Queues) для обмена данными между задачами можно создавать в STM32CubeMX во вкладке «Tasks and Queues» (рисунок 2.60) или непосредственно в коде программы.

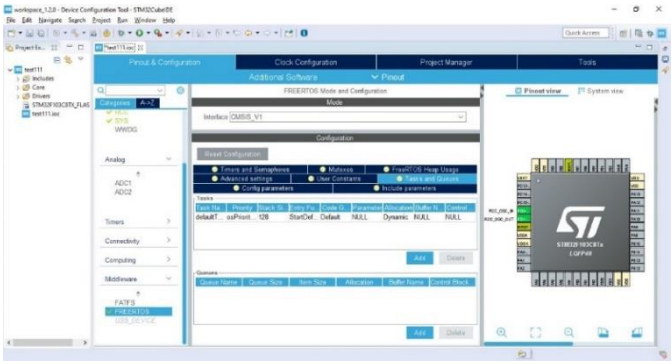



Рис. 2.60 – Окно состояния задач

При создании очереди операционная система выделяет место в памяти, достаточное для хранения определенного количества

сообщений. Помещая сообщение в очередь или читая из очереди, операционная система выдает указатель на текущую структуру. Специальный механизм в операционной системе следит, чтобы данные были правильными. Функции для работы с очередями представлены в [11]. Остальные вкладки в разделе «Configuration» (рисунок 2.60), предназначены для конфигурирования доступа к ресурсам и подробно описаны в [52].

Задачами, очередями, мьютексами и семафорами управляет диспетчер. Диспетчер вызывается по срабатыванию таймера, и пока он не будет вызван, ничего в системе не меняется. Однако в FreeRTOS доступны различные варианты реализации многозадачности. Выше описана вытесняющая многозадачность: когда диспетчер каждый системный тик, никого не спрашивая, останавливает задачу и передает управление другой, обеспечивая тем самым псевдопараллельное выполнение.

В зависимости от конфигурации ядра FreeRTOS многозадачность может быть кооперативной, при ней диспетчер не может отобрать у задачи управление (диспетчер не вызывается каждым системным тиком), а вызывается вручную программистом. Данный режим может привести к тому, что высокоприоритетная задача не сможет взять управление у низкоприоритетной до тех пор, пока та сама не отдаст управление. Однако, грамотное использование кооперативной многозадачности позволяет экономить память, т.к. ОС переключается в конкретные, заранее известные моменты, а значит необходимо сохранять меньше данных.

## 2.3 Контрольные вопросы

1. Можно ли разработанный в SW4STM32 или STM32CubeIDE проект, скомпилировать в *.bin файл и зашить в память микроконтроллера?

2. Какие утилиты позволяют осуществлять программирование МК, проверять корректность прошивки микропрограмм, очищать память программ?

3. Как выбрать МК в CubeMX?

4. Как выбрать необходимые линии ввода/вывода и определить требуемые функции?

5. Как выбрать в CubeMX требуемое периферийное устройство и настроить его на необходимый режим работы?

6. Как определить наличие конфликтов при конфигурировании в CubeMX?

7. Как задать параметры синхронизации в CubeMX?

8. В каких случаях необходимо выбирать внутренний высокочастотный генератор **HSI RC**, в каких – генератор с внешним кварцевым резонатором **HSE**, в каких – внутренний RC-генератор на 40 кГц **LSI**, в каких – генератор с внешним кварцевым резонатором 32,768 кГц **LSE?**

9. Как обеспечить работу МК на максимальной частоте?

10. Почему необходимо выбирать минимально возможные частоты синхронизации?

11. Как оценить энергопотребление МК CubeMX?

12. Можно ли оценить энергопотребление МК, работающего в различных режимах с подключенными периферийными устройствами?

13. Как в CubeMX задать используемое прикладное ПО (IDE)?

14. Какие основные задачи STM32CubeMX позволяет решать пользователю? С какими IDE совместима эта утилита?

15. Как в CubeMX определить интерфейс отладки проекта?

16. Как в CubeMX задать имя проекта и его расположение?

17. Как задать рабочее пространство проекта System Workbench?

18. Какие разделы кода будут неизменны в System Workbench, если проект заново сгенерировать в CubeMX?

19. Как в System Workbench и CubeIDE установить программный сброс при записи программы в память микроконтроллера?

20. Как настроить генерацию hex- и bin-файлов при компиляции проекта в System Workbench и CubeIDE?

21. Как настроить внешний вид среды разработки System Workbench в режиме отладки? Какие ресурсы микроконтроллера доступны для анализа при отладке?

22. Как начать и остановить отладку в System Workbench?

23. Как в CubeIDE при отладке пошагово пройти по инструкциям внутри функции, на которой установлен указатель текущей точки останова?

24. Каковы преимущества и недостатки HAL-функций?

25. С какой целью применяются RTOS, какие преимущества и недостатки их использования?

26. Каковы особенности моделей организации памяти при использовании FREERTOS?

27. С какой частотой переключаются задачи при использовании FREERTOS?

28. В каких случаях вызывается холостая (Idle) задача при использовании FREERTOS?

29. Допускается ли в качестве источника импульсов при использовании FREERTOS использовать таймер SysTick?

30. В каких состояниях может быть задача во FREERTOS, как задача переходит из одного состояния в другое?

# 3 Отладочный модуль PinBoard II R3

При разработке микропроцессорных систем на основе микроконтроллеров (МК) наиболее эффективным способом изучения и приобретения практических навыков является использование отладочных модулей (комплексов), включающих плату с МК, набор внешних периферийных устройств (ПУ), а также соответствующее программное обеспечение (ПО).

Программа, отлаженная на ЭВМ с помощью виртуальной среды моделирования (ВСМ), по каналу связи загружается в отладочный модуль (ОМ), а далее производится отладка с учетом реальных характеристик ОМК и периферийных устройств.

Отладочные модули могут использоваться в качестве автономных устройств или локальных контроллеров автоматизированных систем.

Основными характеристиками ОМ являются:

- типы используемых МК;
- состав и количество внешних периферийных устройств, которые должны соответствовать специфике решаемых задач;
- наличие стандартных последовательных интерфейсов, обеспечивающих взаимодействие как с ЭВМ, так и с внешними ПУ;
- модульный принцип организации;
- возможность оперативной модификации структуры ОМ;
- используемые средства отладки и программирования;
- количество и номиналы источников питания, энергопотребление;
- наличие и полнота документации на ОМ, МК и ПО;
- доступность (бесплатное или условно бесплатное) программного обеспечения;
- авторское сопровождение ОМ;
- стоимость.

Существует большое количество ОМ с различными техническими характеристиками. Авторами выбран оценочный комплекс (ОК) PinBoard II R3 AVR + STM32, имеющий основные характеристики, соответствующие целям и задачам обучения студентов по направлениям 09.03.01 – «Информатика и вычислительная техника», 10.03.01 – «Информационная безопасность», 10.05.01 – «Компьютерная безопасность» и 10.05.03 – «Информационная безопасность автоматизированных систем».

ОК состоит из базовой платы и комплектов модулей для изучения семейств микроконтроллеров MegaAVR и STM32. В данной комплектации используются микроконтроллеры ATMega16A и STM32F103C8T6.

AVR ATMega16A – популярный 8-разрядный микроконтроллер на ядре Atmel AVR8. 16 КБ памяти, 1 КБ ОЗУ, тактовая частота до 16 МГц. Относительно простая организация периферии семейства Mega делают этот микроконтроллер хорошим полигоном для изучения принципов управления типовыми периферийными блоками любого МК (USART, SPI, $I^2C$, параллельные порты, АЦП, аналоговый компаратор, счетчики-таймеры, подсистема прерываний и так далее).

STM32F103C8T6 – 32-разрядный микроконтроллер на ядре ARM Cortex M3 с 64 КБ памяти, 20 КБ ОЗУ и тактовой частотой до 72 МГц. МК является типичным представителем многочисленного семейства ARM Cortex-M3, на примере которого можно проследить тенденции развития не только архитектуры МК, но и периферийных устройств.

Модульный принцип организации позволяет использовать этот ОМ для различных типов МК или ПЛИС (FPGA) [56].

## 3.1 Структурная организация отладочного модуля

Структурная организация ОМ представлена на рисунке 3.1.

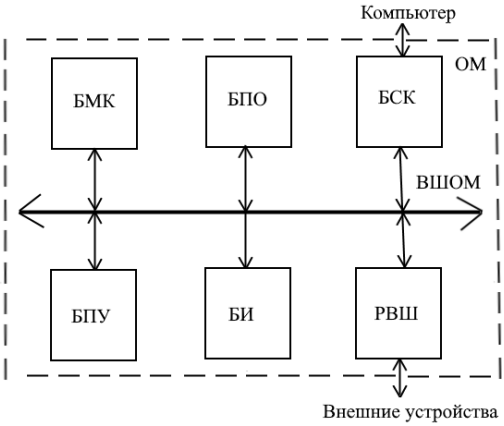



Рис. 3.1 – Структура отладочного модуля

Основным центром обработки и управления является блок микроконтроллера (БМК), который представляет собой набор сменных плат с соответствующими МК или ПЛИС, устанавливаемых в гнездо контактов, имеющих унифицированные линии ввода/вывода.

Взаимодействие БМК с компьютером выполняет блок связи с компьютером (БСК), в состав которого входят средства, обеспечивающие обмен информацией по интерфейсу USB.

Блок программирования и отладки (БПО) обеспечивает запись программ и контроль их работоспособности в МК с помощью соответствующих инструментальных средств, находящихся в ЭВМ.

Блок периферийных устройств (БПУ) содержит набор компонентов, необходимых для изучения функциональных возможностей МК и организации взаимодействия оценочного модуля с внешними устройствами.

Контроль работы МК при вводе/выводе информации осуществляет блок индикации (БИ), состоящий из набора светодиодов, 7-сементного и алфавитно-цифрового индикаторов и зуммера.

Общий вид базовой платы представлен на рисунке 3.2 (вид сверху и снизу).

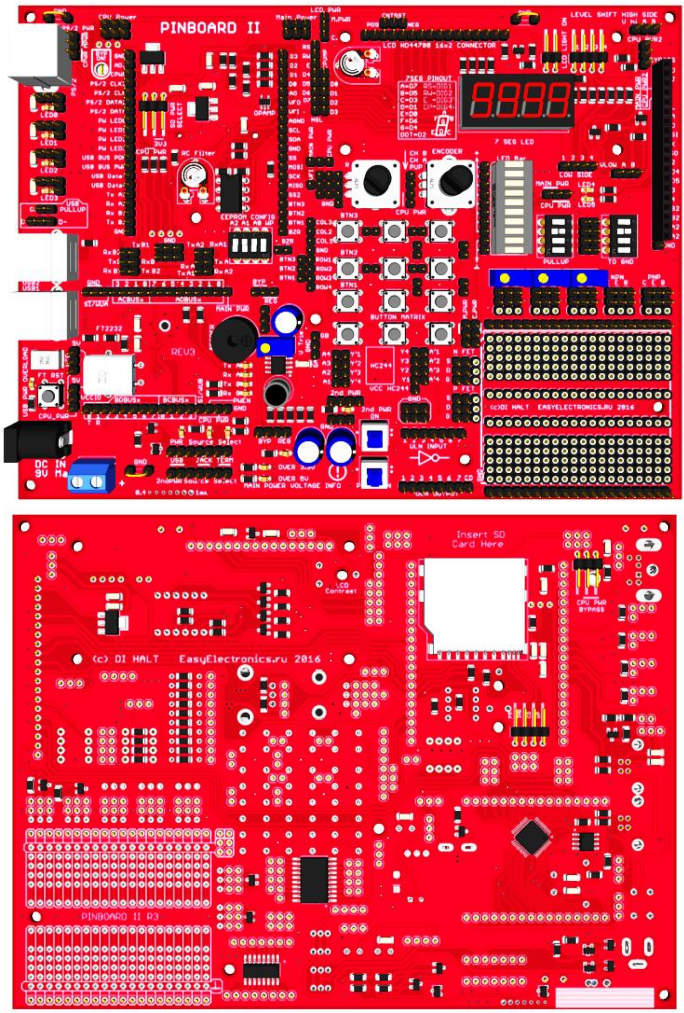



Рис. 3.2 – Общий вид базовой платы

Расширитель внутренней шины (РВШ) позволяет присоединять к оценочному модулю дополнительные модули

пользователя, ориентированные на использование последовательных интерфейсов PS/2, USART, SPI, $I^2C$.

Блоки объединяются на основе внутренней шины оценочного модуля (ВШОМ), которая состоит из линий ввода/вывода БМК. Часть элементов блоков соединены с ВШОМ жестко, а основная часть – с помощью перемычек и проводной (шнуровой) коммутации.

На стадии проектирования определяется требуемый состав внутренних и внешних периферийных устройств, выполняется соединение ВШОМ с блоками оценочного модуля, выбираются алгоритмы обработки и управления и на их основе разрабатывается программа с помощью соответствующих инструментальных средств (IDE). После компиляции с помощью БПО программа загружается в БМК. Результаты работы контролируются БИ и/или терминалом ПЭВМ.

### 3.1.1 Блок микроконтроллеров

Плата сменного модуля БМК содержит определенный тип МК или ПЛИС с требуемым набором внешних элементов и линий питания. Линии ввода/вывода МК или ПЛИС соединены со штырями, которые могут быть скоммутированы с помощью соединительных проводов или перемычек с ВШОМ и другими элементами оценочного модуля. Это обеспечивает возможность произвольного соединения БМК с базовой платой.

**Дальнейшее изложение предполагает использование микроконтроллера STM32F103C8T6.**

- Плата модуля микроконтроллера STM32F103C8T6 представлена на рисунке 3.3 Она содержит МК, элементы «обвязки», кнопку сброса **RST**, кварцевые резонаторы на 12 МГц и 32 768 Гц («часовой кварц»), переключатели, задающие режим работы памяти программ **BOOT1** и **BOOT0,** штыри для подключения порта **JTAG** и батарейного питания **UBAT.** На плате обозначены выводы портов МК.

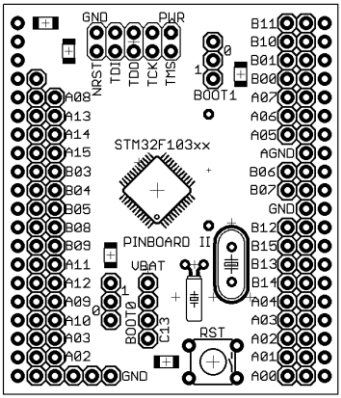



Рис. 3.3 – Модуль микроконтроллера STM32F103C8T6

Состав линий БМК представлен на рисунке 3.4 и в таблице 3.1.

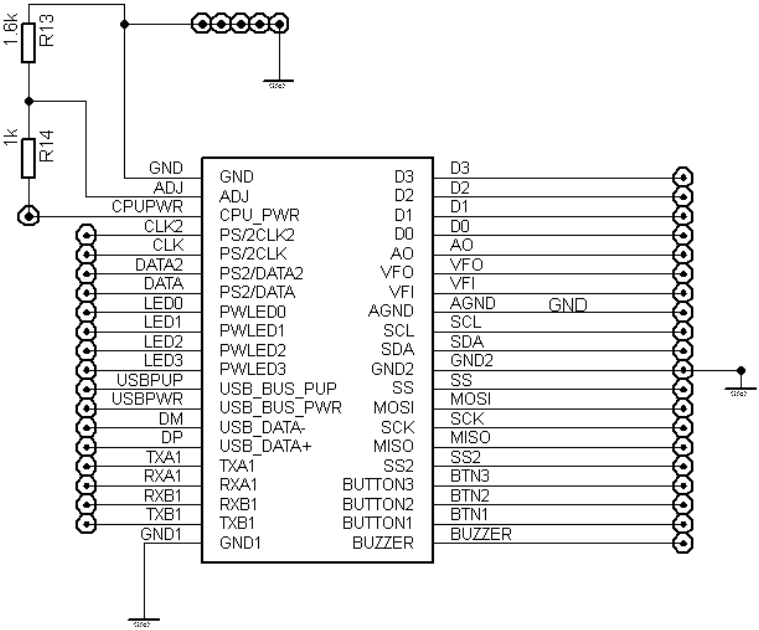



Рис. 3.4 – Состав линий блока микроконтроллеров

Таблица 3.1. Цоколевка внешних выводов модуля

| Левая сторона | Правая сторона |
|---|---|
| **GND** – шина земли | **D3…D0** – вход ЦАП по схеме R2R |
| **ADJ** – вывод ADJ LM317 для задания напряжения | **AO** – от Analog Output – выход R2R ЦАП. Поступает с операционного усилителя |
| **CPU_POWER** – напряжение питания процессора | **VFO** – Voltage Filter Out – выход RC фильтра. |
| **PS/2CLK2** – CLOCK 2 вывод разъема PS/2 | **VFI** – Voltage Filter In – вход RC фильтра. |
| **PS/2CLK** – CLOCK вывод разъема PS/2 | **AGND** – Аналоговая земля |
| **PS/2DATA2** – DATA 2 вывод разъема PS/2 | **SCL** – строб линия шины $I^2C$. Соединена подтягивающим резистором с CPU POWER |
| **PS/2DATA** – DATA вывод разъема PS/2 | **SDA** – линия данных шины $I^2C$. Соединена подтягивающим резистором с CPU POWER |
| **PWLED0** – вывод на светодиод и фильтр для сглаживания ШИМ сигнала. Канал 0 | **GND** – земля. |
| **PWLED1** – вывод на светодиод и фильтр для сглаживания ШИМ сигнала. Канал 1 | **SS** – линия выбора устройства, подключенного к шине SPI. Этот вывод подключен также SD карте |
| **PWLED2** – вывод на светодиод и фильтр для сглаживания ШИМ сигнала. Канал 2 | **MOSI** – Master Output Slave Input – линия данных шины SPI |
| **PWLED3** – вывод на светодиод и фильтр для сглаживания ШИМ сигнала. Канал 3 | **SCK** – линия синхронизации шины SPI |
| **USB BUS PUP** – подтяжка шины D на USB. Подача на этот вывод пяти вольт подтягивает шину D и обеспечивает обнаружение устройства на шине USB | **MISO** – Master Input Slave Output – линия данных шины SPI |

*Окончание табл. 3.1*

| **Левая сторона** | **Правая сторона** |
|---|---|
| **USB DATA + – Шина данных USB** | **BTN3 – линия может быть подключена к кнопке BTN3** |
| **USB DATA - – Шина данных USB** | **BTN2 – линия может быть подключена к кнопке BTN2** |
| **USB BUS PWR** – пять вольт с шины USB пропущенные через резистор в 100 кОм. Позволяют отследить наличие питание на шине USB | **SS2** – дополнительная линия выбора устройства, подключенного к шине SPI. Выведен на интерфейс пользователя, позволяя адресовать внешнее устройство SPI |
| **TXA1** – Вывод TX канала A (соединяется с линией TX микроконтроллера! Линии RX и TX перекрещиваются на колодке коммутатора) | **BTN1** – линия может быть подключена к кнопке BTN1 |
| **RXA1** – Вывод RX канала A (соединяется с линией RX микроконтроллера! Перекрещивание RX и TX осуществляется на колодке коммутатора) | **BZR** – вывод для управления зуммером (Buzzer) |
| **TXB1** – Вывод TX канала B (соединяется с линией TX микроконтроллера! Перекрещивание RX и TX осуществляется на колодке коммутатора) | — |
| **RXB1** – Вывод RX канала B (соединяется с линией RX микроконтроллера! Перекрещивание RX и TX осуществляется на колодке коммутатора) | — |
| **GND** – земля | — |

На базовой плате каждое гнездо имеет соответствующую надпись. На рисунке 3.5 приведена цоколевка STM32F103C8T6.

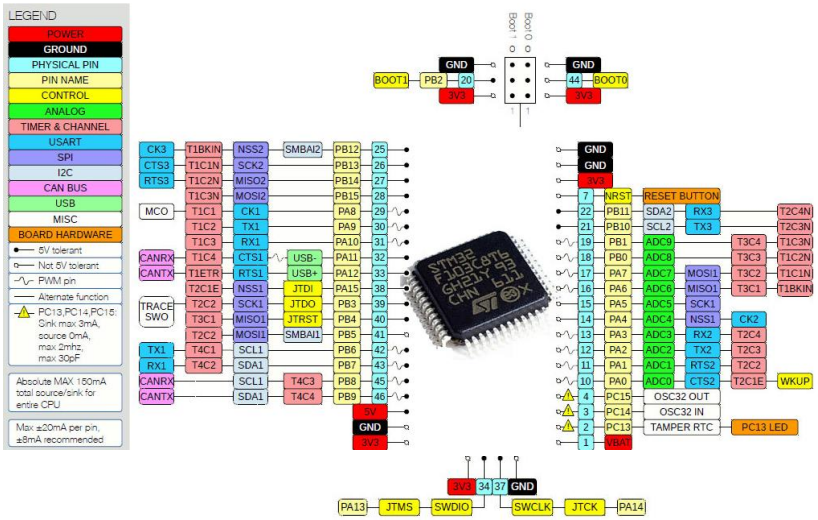



Рис. 3.5 – Цоколевка STM32F103C8T6

### 3.1.2 Блок связи с компьютером

Связь с компьютером осуществляется через два разъема **USB.**

**USB1** используется в основном для обмена информацией между компьютером и ресурсами оценочного модуля через параллельные и последовательные каналы, формируемые мостом FT2232, при программировании модулей AVR (например, ATmega 16).

**USB2** служит для обмена информацией с **устройствами, имеющими «на борту» порты USB,** например, **компьютером, микроконтроллерами.**

**Разъем USB1** соединен с двухканальным универсальным мостом FT2232 фирмы FTDI. Место установки моста – два горизонтально расположенных штыревых разъема (рисунок 3.2).

**Основная функция FT2232 в модуле – преобразование сигналов порта USB в два коммутируемых канала ввода-вывода с протоколом USART.**

На базовой плате мост FT2232D представляет собой автономный блок, линии ввода-вывода которого выведены под штыревые соединения, что позволяет его использовать в качестве демоплаты для изучения возможностей этой микросхемы. Режимы программируются с помощью микросхемы памяти EEPROM, расположенной на плате.

Электрическая схема моста FTDI представлена на рисунке 3.6.

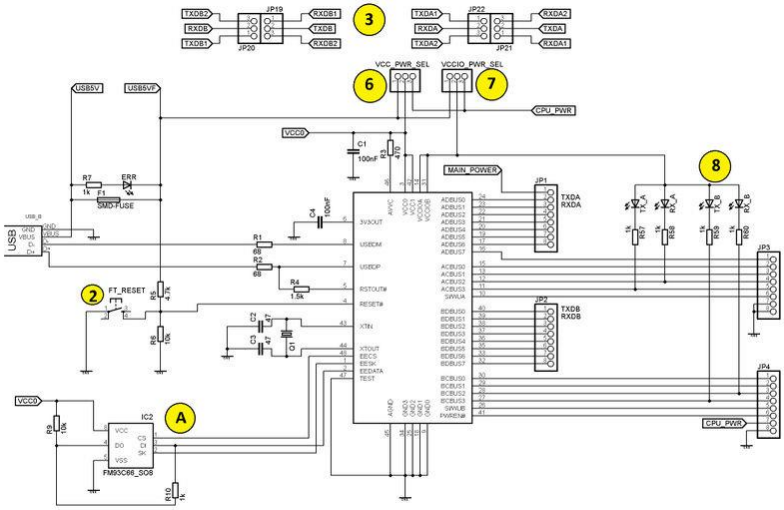



Рис. 3.6 – Электрическая схема моста FTDI

Буквой «А» отмечена EEPROM память, в которую можно записать конфигурацию микросхемы. Шины данных и управления моста выведены на штыри (справа на схеме), что позволяет устанавливать дополнительные платы пользователя.

2 – кнопка сброса моста.

3 – коммутатор портов USART (рисунок 3.7).

С помощью джамперов, средние выводы которых соединены с каналами A и B FTDI, можно подключиться к БМК (4)

(RxDA1:TxDA1, RxDB1:TxDB1), разъему расширения (5) (RxDA2:TxDA2, RxDB2:TxDB2) или выполнить «эхо контроль», замкнув линии Rx и Tx (рисунок 3.9).


Рис. 3.7 – Коммутатор портов USART

6 – джампер выбора питания FTDI. Позволяет выбрать в качестве источника питания USB или шину CPU Power. **Рекомендуется напряжение 5 В**.

7 – джампер выбора питания портов ввода-вывода.

8 – микросхемы FTDI. Можно выбирать между 5 В и шиной CPU Power. **Рекомендуется присоединять к CPU Power**.

9 – блок индикации состояния сигналов Rx, Tx.

Однако возможности моста не ограничиваются только этими функциями [7, 65].

FT2232 может работать в следующих режимах:

**Multi-Protocol Synchronous Serial Engine (MPSSE).** В этом режиме мосты могут эмулировать работу различных последовательных протоколов. Компания FTDI в качестве примера предоставляет готовые библиотеки, позволяющие использовать мосты в режимах USB-SPI, USB-$I^2C$ или USB-JTAG.

**Synchronous Bit-Bang Mode** – синхронного управления портами. В этом режиме можно использовать порт USB для управления линиями ввода-вывода FT2232 в соответствии с программой, находящейся в компьютере. Скорость передачи при таком способе обычно не очень высокая, но позволяет реализовать любой протокол.

**CPU-Style FIFO Interface Mode** – режим FIFO с эмуляцией процессорной шины.

**MCU Host Bus Emulation Mode** – режим эмуляции хостом шины микроконтроллера.

**Fast Opto-Isolated Serial Interface Mode** – режим быстрого последовательного интерфейса с оптоизоляцией.

**Разъем USB2** (10 на рисунке 3.9) обеспечивает взаимодействие с внешними устройствами на основе спецификации USB 2.0 в режимах Low Speed (до 1, 5 Мбит/с) и Full Speed (до 12 Мбит/с). Технические возможности блока приведены в [51].

Электрическая схема соединения разъема USB2 и микроконтроллера приведена на рисунке 3.8.

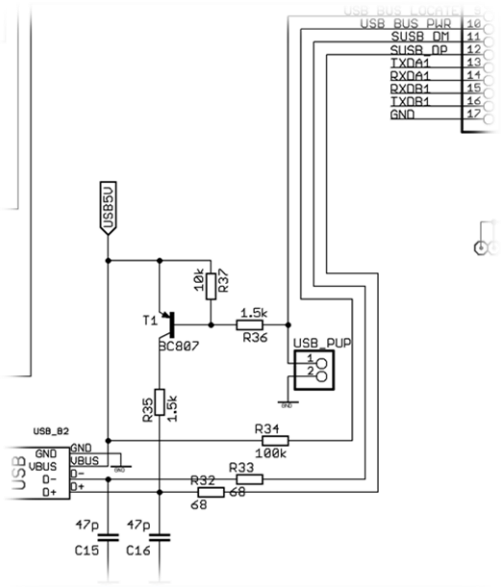



Рис. 3.8 – Электрическая схема соединения разъема USB2 и БМК

Схема содержит стандартную обвязку и управляемый транзистором T1 подтягивающий резистор R35, который сигнализирует компьютеру о наличии подключенного устройства. Транзистор T1 может быть включен по сигналу БМК (Вывод USB BUS PDN) или перемычкой USB PULLUP (рисунок 3.6).

Перед подключением БМК к внешним линиям USB следует сформировать идентификатор скорости обмена. В режиме Full Speed подтягивающий резистор подключается к линии «D+», а в Low Speed – к линии «D-». Это выполняется джампером в секции «USB Pull UP».

Подключение питания к внешним линиям USB2 можно проверить, контролируя сигнал USB PWR.

Шина данных USB соединяется с **USB Data+** и **USB Data-** БМК.

Расположение элементов электрических схем на плате отладочного модуля иллюстрирует рисунок 3.9.

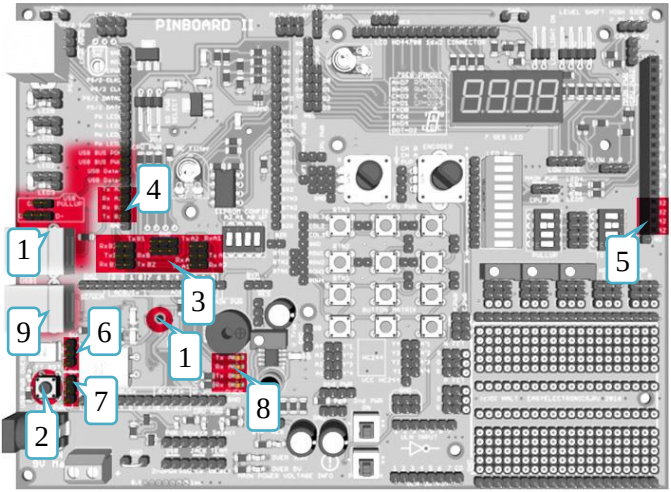



Рис. 3.9 – Расположение элементов БСК

### 3.1.3 Блок программирования и отладки

Блок программирования и отладки БПО включает определенный набор средств, зависящих от типа электронного устройства (МК, ПЛИС) и его структуры.

Наиболее эффективным средством отладки и программирования для микроконтроллеров является внутрисхемный отладчик JTAG.

Технология JTAG широко применяется для тестирования электронных устройств на основе микроконтроллеров, CPU, CPLD и/или FPGA. JTAG также позволяет выполнять аппаратную отладку, чтение/запись памяти, управление линиями ввода-вывода, анализ производительности разработанной программы. Особенностью JTAG является возможность контроля и программирования не только микроконтроллера (или ПЛИС), но и подключенной к его выводам периферии [50].

Отладочный модуль с МК **STM32F103C8T6** может использовать программаторы **CoLink ARM Cortex JTAG** и **ST-LINK V2**.

**CoLink ARM Cortex JTAG** предназначен для работы с микроконтроллерами, поддерживающими интерфейс **JTAG** и ориентирован на использование среды разработки **CooCox IDE,** которая не рекомендуется к использованию (рисунок 3.10).

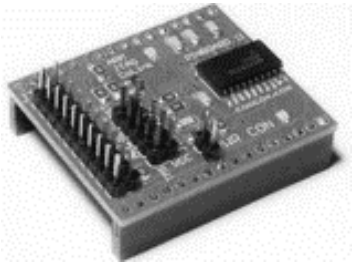


Рис. 3.10 – Внешний вид модуля **CoLink ARM Cortex JTAG**

Связь модуля **CoLink ARM Cortex JTAG** с компьютером осуществляется с помощью моста FT2232. Модуль устанавливается на штыри моста (рисунок 3.11).

Одним из недостатков **JTAG** является относительно большое количество сигнальных линий (4-5).

**ST-LINK V2 –** это адаптер STMicroelectronics, предназначенный для **программирования** микроконтроллеров STM8 и STM32.

Адаптер поддерживает интерфейсы JTAG, SWD и SWIM (применяется для STM8).

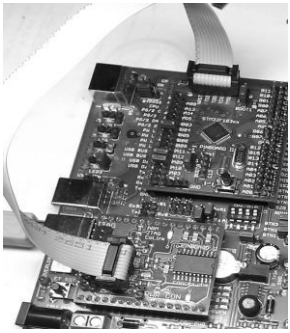


Рис. 3.11 – Соединение программатор **CoLink ARM Cortex JTAG** с микроконтроллером

В лабораторных работах используется **ST-LINK V2 в алюминиевом корпусе.** Он поддерживает интерфейсы SWD и SWIM, но **не поддерживает JTAG** [47].

Технология **SWD** (Serial Wire Debug) – это более современная версия JTAG, требующая для работы только две сигнальных линии ввода/вывода (SWCLK – линия синхронизации, SWDIO – двунаправленная линия данных). Однако, JTAG и SWD обладают аналогичными функциональными возможностями с разными вариациями, зависящими от управляющего ПО и аппаратного обеспечения. Соединение БМК с адаптером показано на рисунке 3.12.

При загрузке программы **во внутреннюю флеш-память** МК необходимо перед программированием на плате модуля микроконтроллера установить перемычки **BOOT1 = 1**, **BOOT0 = 0 (рисунок 3.12,** таблица 3.2).

Затем соединить USB-разъем ST-LINK V2 с компьютером непосредственно или с помощью кабеля. Все драйверы в компьютере должны установиться автоматически. Диспетчер устройств должен отразить подключение устройства (см. рисунок 3.13).

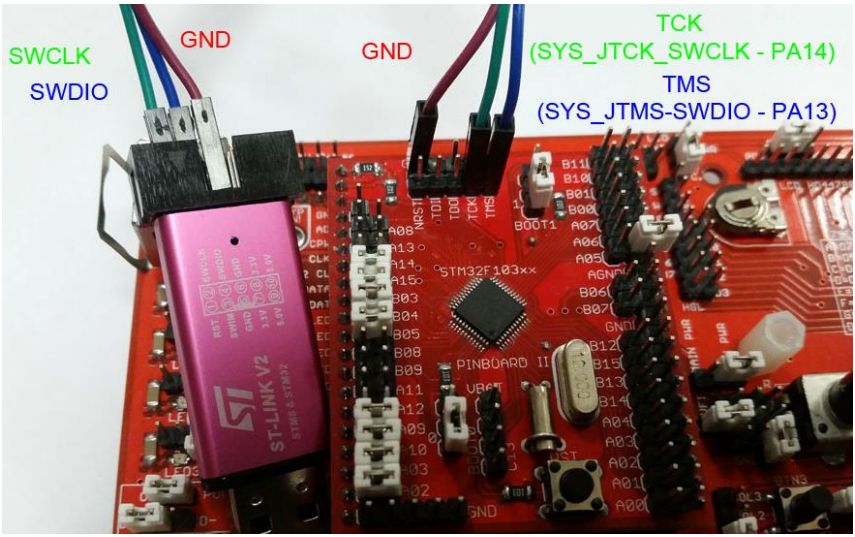



Рис. 3.12 – Соединение **ST-LINK V2** с платой микроконтроллера

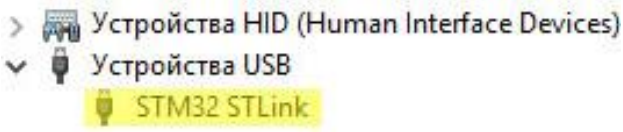



Рис. 3.13 – Подключение адаптера ST-LINK V2

Если что-то пошло не так и устройство не обнаружено – необходимо пройти по ссылке, скачать [86] и установить утилиту **STM32 ST-LINK utility**.

**Примечание:** на компьютерах под управлением ОС Windows 8, перед проведением вышеописанных действий необходимо сделать так: «**Параметры**» → «**Изменение параметров компьютера**» → «**Общие**» → «**Особые варианты загрузки**» и выбрать параметр «**Отключение проверки подписи драйверов**».

При запуске программы из **системной памяти** (**BOOT1 = 0**, **BOOT0 = 1)** используется уже зашитый загрузчик (bootloader), что позволяет использовать в качестве программатора USART,

управляемый специальными драйверами. В этом случае связь с компьютером выполняется через разъем USB1 и мост FT2232D.

Загрузка **из ОЗУ** (**BOOT1 = 1**, **BOOT0 = 1)** используется на этапе отладки для сохранения ресурса программирования FLASH-памяти.

Таблица 3.2. Режимы запуска микроконтроллера

| BOOT1 | BOOT0 | Режим запуска программы |
|---|---|---|
| 0 | 0 | Внутренняя FLASH |
| 1 | 0 | |
| 0 | 1 | Системная память |
| 1 | 1 | Внутреннее ОЗУ |

### 3.1.4 Блок периферийных устройств

Блок периферийных устройств (БПУ) содержит набор компонентов, позволяющих изучать особенности проектирования контроллеров различного назначения на основе микроконтроллера STM32F103C8T6. В состав БПУ входят: микросхема EEPROM 24C64WP, гнездо для SD-карты, энкодер, формирователи входных и выходных напряжений.

**Микросхема 24C64WP** является энергонезависимым постоянным запоминающим устройством с возможностью перепрограммирования (EEPROM).

Основные технические характеристики: объем – 8196 байт, напряжение питания – (2,7÷5,5) В, интерфейс – $I^2C$, предусмотрено изменение трех младших битов адреса, частота обмена – не более 400 КГц. Остальные характеристики приведены в [1].

Младшие биты адреса (A0-A2) и режим защиты от записи (WP) EEPROM задаются переключателем EEPROM CONFIG (рисунок 3.14).

Линии интерфейса выведены на разъем расширения. Расположение микросхемы и переключателей на плате показано цифрой «1» на рисунке 3.15.

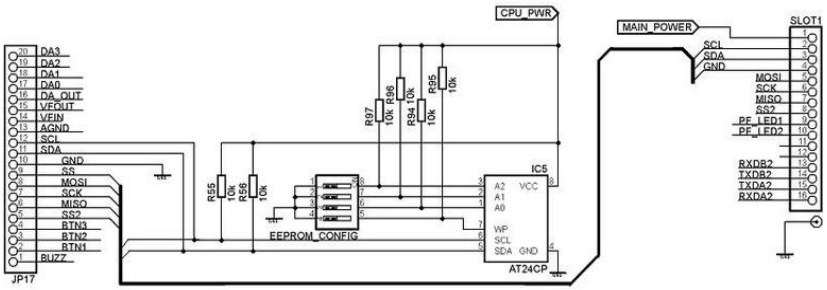



Рис. 3.14 – Электрическая схема подключения 24C64WP

**SD карта** (или Secure Digital Memory Card – безопасная цифровая карта памяти) представляет собой микроконтроллер и микросхему FLASH памяти. Реализация SD Memory Card SPI использует подмножество набора команд протокола SD Memory Card [59, 57]. Электрическая схема подсоединения гнезда SD карты приведена на рисунке 3.16.

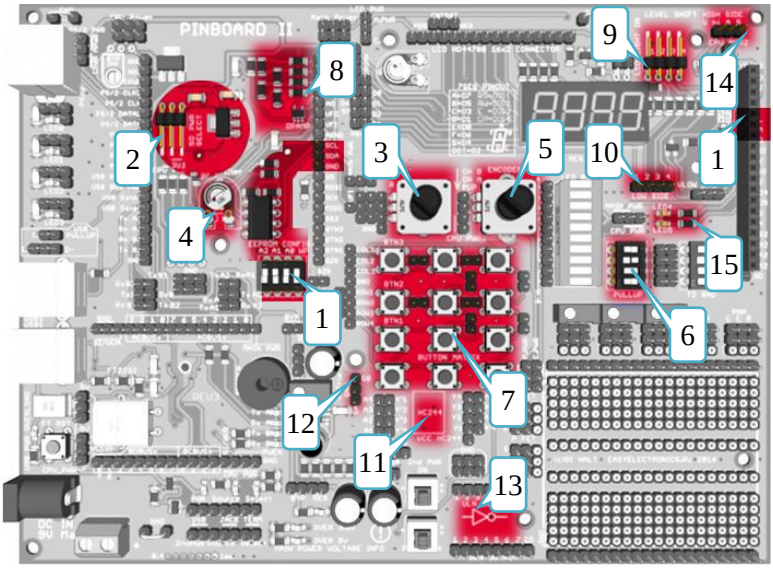



Рис. 3.15 – Расположение периферийных устройств

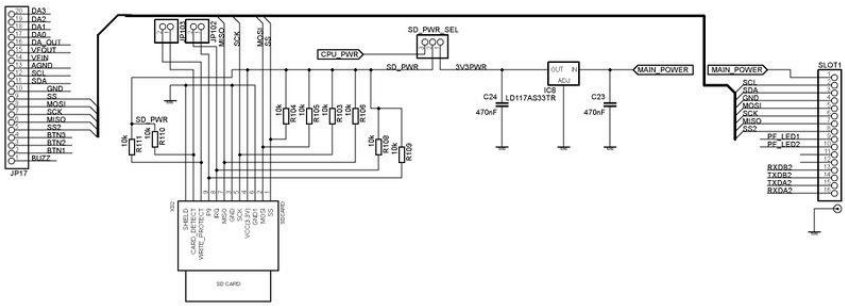



Рис. 3.16 – Электрическая схема соединений SD карты

Часть линий SD карты подключена к интерфейсу SPI, а остальные – выведены на штыри, расположенные снизу платы. **Для выбора карты используется линия SS**.

Питание карты может быть подключено к CPU PWR или стабилизатору IC8 с выходным напряжением 3.3 В и током до 1 А. Рекомендуемая нагрузка – не более 500 мА. Переключение напряжения выполняется селектором SD PWR SEL, расположение которого показано на рисунке 3.16. Выход стабилизатора отмечен цифрой «2».

Гнездо SD карты разведено полностью, но **на ВШОМ выведены только линии SPI.** Если нужны остальные выводы, то к ним можно подключиться через штыри снизу платы (рисунок 3.17).

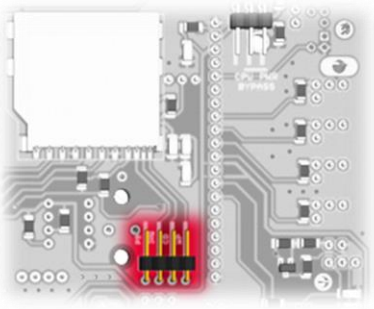


Рис. 3.17 – Гнездо SD карты

Интерфейс SPI выведен также на разъем расширения, а **для адресации внешнего модуля используется линия SS2.**

**Источник аналогового напряжения** для АЦП может быть реализован с помощью потенциометра и RC-фильтра (рисунок 3.18).

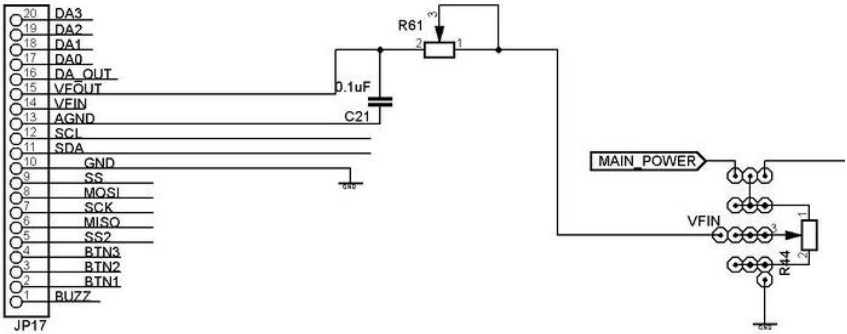



Рис. 3.18 – Электрическая схема источника аналогового напряжения

Переменный резистор R44 (10 кОм) («3» на рисунке 3.15) с помощью переключателей может быть подключен к линиям питания (MAIN POWER и CPU POWER), земле и на вход RC-фильтра **VFIN**. Выход фильтра **VFOUT** соединен с БМК.

Низкочастотный RC – фильтр собран на конденсаторе в 100nF и переменном резисторе R61 в 10 кОм. **Переменный резистор находится под процессорным модулем и доступен через отверстие на обратной стороне платы отладочного модуля** («4» на рисунке 3.15).

**Энкодер или датчик угла поворота** – это электромеханическое устройство, предназначенное для преобразования углового положения вала (или оси) в электрические сигналы. Существует два основных типа энкодеров – абсолютные и инкрементные. Более функциональный и дорогой абсолютный энкодер для каждой позиции своего вала выдает уникальный код. Этот тип энкодеров применяется в промышленном оборудовании – робототехнике, станках, конвейерных линиях [71].

В оценочном комплексе используется электромеханический инкрементный энкодер. При вращении вала формируются импульсы, число которых пропорционально углу поворота. Количество этих импульсов относительно начального положения позволяет определить величину угла поворота вала энкодера.

**Этот тип энкодеров не формирует выходные импульсы, когда его вал находится в состоянии покоя.**

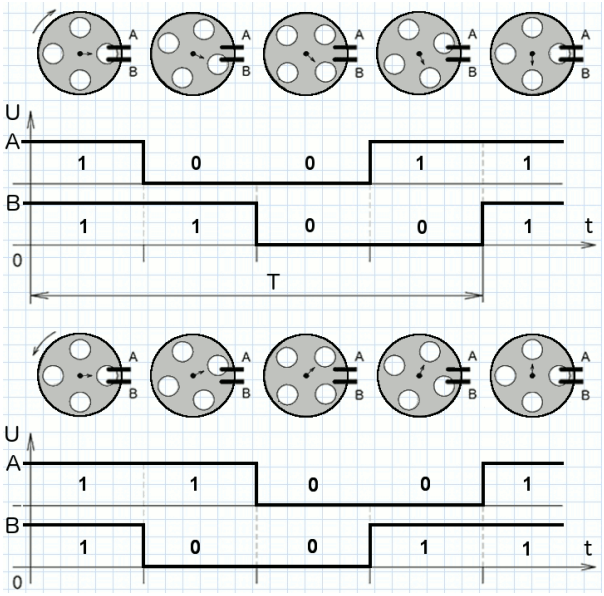



Рис. 3.19 – Принцип действия электромеханического энкодера

В качестве примера рассмотрим энкодер (рисунок 3.19). Он представляет собой металлический диск с 4 отверстиями. При нахождении контактов A и B на диске они замыкаются, а в момент прохождения отверстия – размыкаются [84].

Если контакты подключить к схеме, то на выходе получаем уровни сигналов, анализ которых позволяет определить угол поворота и направление вращения. Выводы A и B подтягиваются

резисторами к напряжению питания. Нулевой уровень соответствует замкнутому состоянию, а высокий – разомкнутому.

Если оба контакта попадают в отверстие, тогда на них будет высокий уровень напряжения. При повороте по часовой стрелке первым замыкается на землю контакт A, а затем – контакт B. Достигнув следующего отверстия в диске, контакт A разомкнётся первым и сформируется высокий уровень, вторым будет контакт B. После этих перемещений контакты вернутся в исходное состояние и при дальнейшем вращении эта диаграмма будет циклически повторяться.

В приведенном примере последовательность двоичных кодов за один «щелчок» ручкой энкодера будет 11, 01, 00, 10 при повороте направо, а в обратном направлении – 11, 10, 00, 01. Зная последовательность чисел, поступающую с энкодера, можно определить направление вращения энкодера, подсчитывая количество измененных состояний, можно определить угол поворота.

Число отверстий на диске характеризует разрешающую способность энкодера.

В общем случае, конструкции дисков энкодеров могут быть различными: металлические, диэлектрические, оптические, магнитные и так далее. В зависимости от этого появляются отличия в виде и полярности выходного сигнала в электрической схеме.

В лабораторных работах используется энкодер EC12E24404A6 («5», рисунок 3.15), который имеет 24 отверстия, максимальный рабочий ток 5 мА, ресурс – 15000 циклов. По сравнению с приведенным выше примером формирование импульсов будет инверсным: в исходном состоянии 00, а затем 10, 11, 01. За один полный оборот ручки его выводы изменят состояние от 00 до 11–24 раза, то есть за один щелчок диск энкодера поворачивается на 15 градусов.

Электрическая схема энкодера представлена на рисунке 3.20.

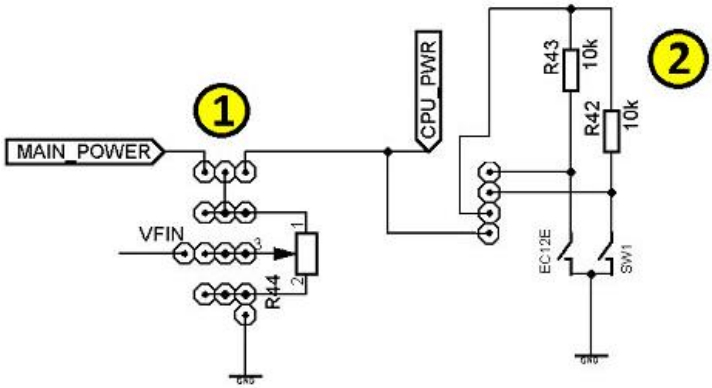



Рис. 3.20 – Электрическая схема соединений энкодера

На схеме рядом с разъемом энкодера находится линия питания CPU POWER, что позволяет одним джампером установить в качестве подтяжки напряжение с CPU POWER.

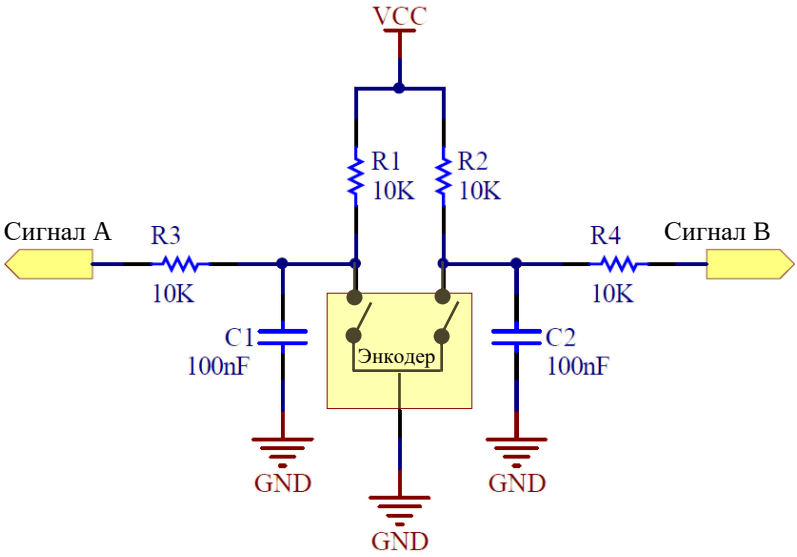



Рис. 3.21 – Электрическая схема энкодера

При переключении механических контактов возникает дребезг (многократное замыкание/размыкание контактов и последующее формирование электрических импульсов). Для устранения его влияния используют на выходе энкодера RC-фильтры (рисунок 3.21) или вводят программные задержки на время, достаточное для завершения процесса дребезга (20–30 мс).

**Четырехразрядный блок переключателей** («6» на рис. 3.15) может использоваться для формирования цифровых управляющих сигналов (рисунок 3.22). Переключателем SW14 можно установить высокий уровень, а SW15 – низкий. Две линии этого блока выведены на разъем расширения.

**Конфигурируемая матричная клавиатура** («7» на рисунке 3.15) состоит из 12 кнопок. В зависимости от решаемых задач кнопки могут использоваться автономно или соединены перемычками с помощью коммутации, например, в матричную клавиатуру 4x3 (рисунок 3.23).

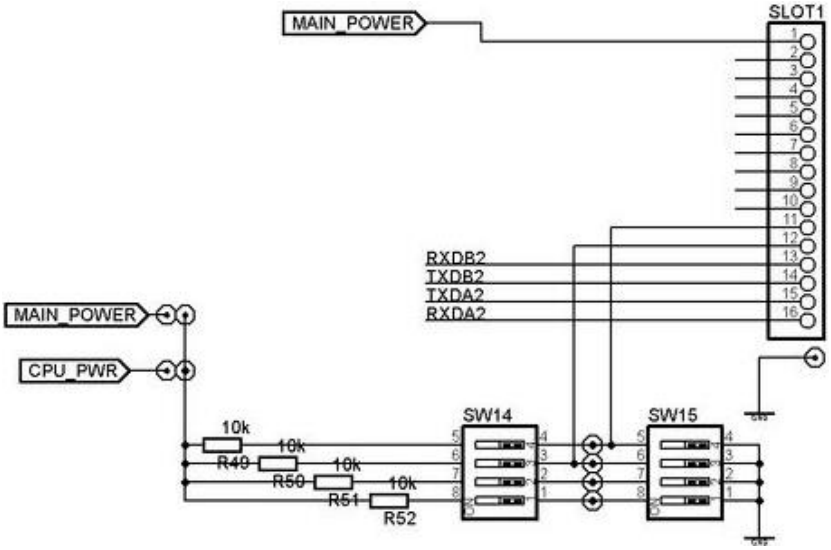



Рис. 3.22 – Электрическая схема переключателей

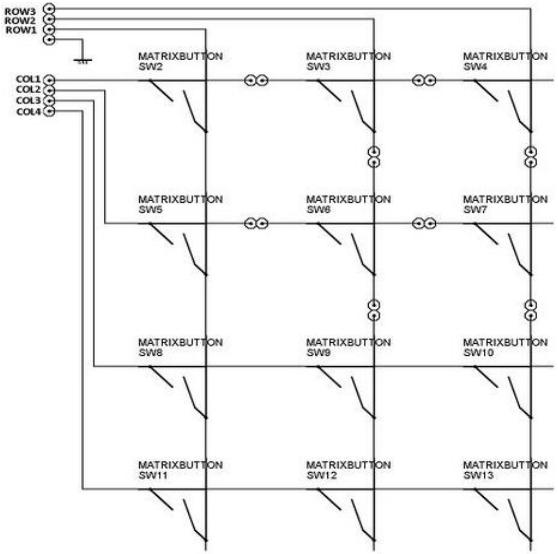



Рис. 3.23 – Схема конфигурируемой матричной клавиатуры

Рядом с выводом левого столбца ROW1 находится линия GND, что позволяет в случае необходимости перемычками присоединить 4 кнопки этого столбца к земле.

Выводы от кнопок, отмеченных на плате как BTN1 – BTN3, соединяются с БМК. Для присоединения этих кнопок к требуемым выводам микроконтроллера используют шнуровую коммутацию, а с клавиатурой соединяют перемычками.

Остальные кнопки, коммутируя перемычками, можно использовать произвольным образом.

**Цифроаналоговый преобразователь ЦАП с резисторной матрицей R2R** на четыре разряда («8» на рисунке 3.15) можно использовать как источник напряжения при изучении принципов программирования АЦП МК. Для метрологических целей ЦАП не пригоден, так как не обеспечиваются стабильность номиналов матрицы и питающего напряжения. Возможны и другие применения.

Двоичный код с выходов DA0÷DA3 БМК поступает на вход матрицы **R2R.** На выходе повторителя напряжения DA1, выполненного на операционном усилителе LMV321, формируется напряжение, пропорциональное двоичному коду (рисунок 3.24).

Выход DA1 подключен к линии DAOUT БМК.

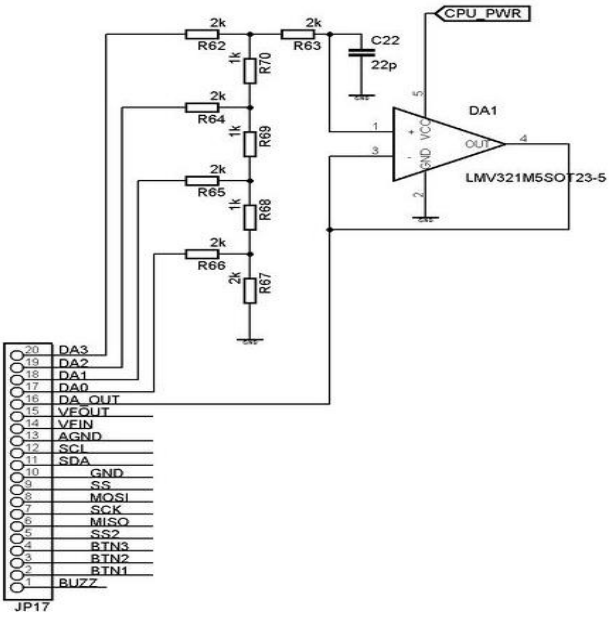



Рис. 3.24 – Электрическая схема ЦАП

Для **преобразования 5 В в 3,3 В** можно использовать **четырехканальный делитель напряжения**. Цифрой **«9»** обозначена сторона высокого напряжения (**High Side**), а цифрой **«10»** низкого (**Low Side**) (рисунок 3.15) и («3», «4» рисунок 3.25).

Связь шин с различными значениями питающего напряжения, работающие по принципу «монтажное И» с подтягивающим резистором ($I^2C$, TWI, 1-Wire), выполняет **двухканальный транзисторный конвертер** логических уровней («5», «6» рисунок 3.25) и («14», «15» рисунок 3.15). Для его использования кроме

линий шины A и B нужно подключить к выводам $V_{hi}$ и $V_{low}$ шины питания высокой и низкой стороны (рисунок 3.25).

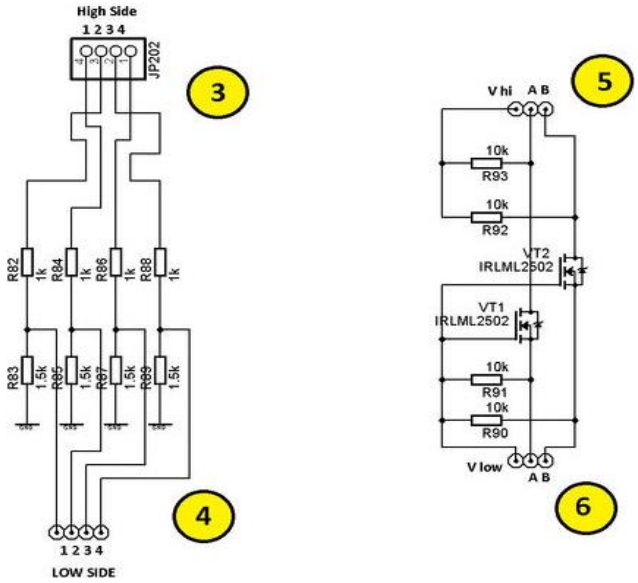



Рис. 3.25 – Схема делителей напряжения и конвертеров

**Шинный формирователь** 74HC244 («11» на рисунке 3.15) может использоваться в качестве магистрального усилителя, шинного мультиплексора и др.

Он состоит из двух однонаправленных неинвертирующих четырехразрядных формирователя с тремя состояниями (рисунок 3.26). Выходной ток – 35 мА, напряжение питания – 2÷6 В. Управление выходом выполняется по линии **G**. Выводы управления выведены на штыри слева от микросхемы и обозначены на плате как GA и GB. Между ними находится штырь земли, что позволяет с помощью джампера управлять линиями A и B («12» на рисунке 3.15).

**Микросхема привязана только к общей земле. Использование остальных линий определяется характером решаемых задач.**

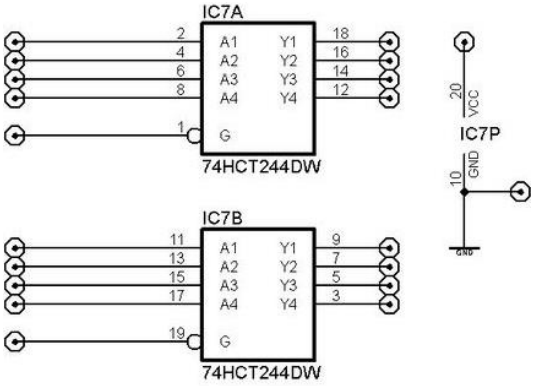



Рис. 3.26 – Шинный формирователь 74HC244

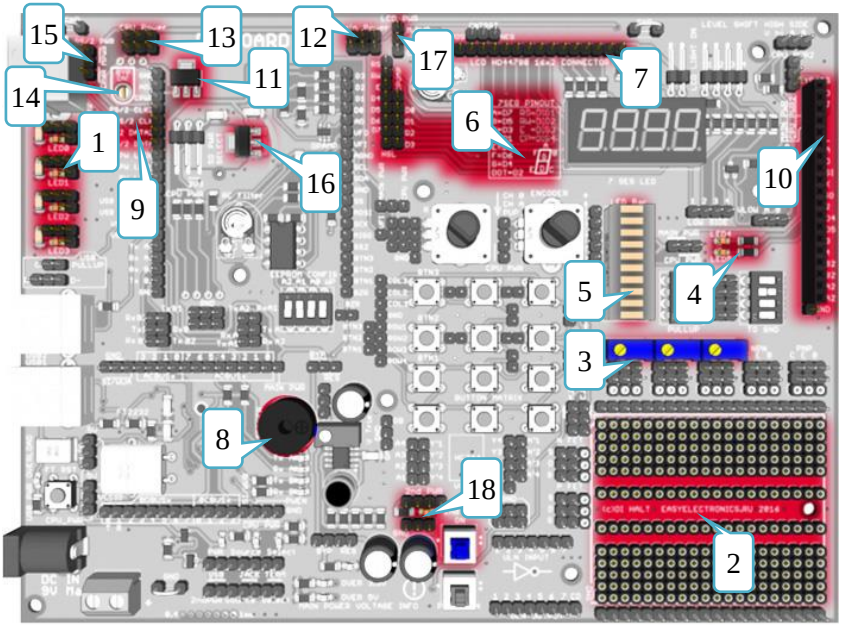



Рис. 3.27 – Расположение периферийных устройств.

**Семиканальная сборка на составных транзисторах ULN2003** («13» на рисунке 3.15). Микросхема ULN2003a – это транзисторная сборка по схеме Дарлингтона с выходными ключами

повышенной мощности, имеющая на выходах диоды, которые предназначены для защиты управляющих электрических цепей от обратного выброса напряжения при индуктивной нагрузке (рисунок 3.28).

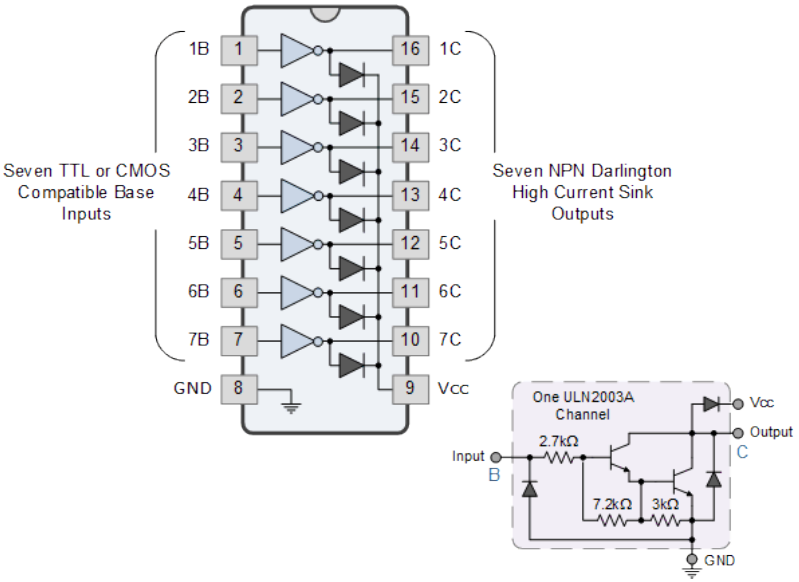



Рис. 3.28 – Электрическая схема ULN2003.

**Может использоваться для управления нагрузкой значительной мощности, включая электромагнитные реле, двигатели постоянного тока, электромагнитные клапаны, в схемах управления различными шаговыми двигателями и так далее.**

Общий вывод шунтирующих диодов («9» на рисунке 3.28) расположен на стороне выхода. Номинальный ток коллектора одного ключа – 0,5А, максимальное напряжение на выходе до 50 В. Рекомендуемое значение выходного тока (300–400) мА. В ТТЛ-логике при величине входного напряжения не более 3 В ток коллектора выходного транзистора составляет 300 мА. Время

переключения выходных транзисторов лежит в пределах от 0,25 до 1 мкс, что позволяет работать на достаточно высоких частотах.

Для **контроля выходного сигнала широтно-импульсного модулятора ШИМ** можно использовать **четырехканальный** блок светодиодов с фильтрами, который позволяет формировать различные варианты подключения с помощью джамперов.

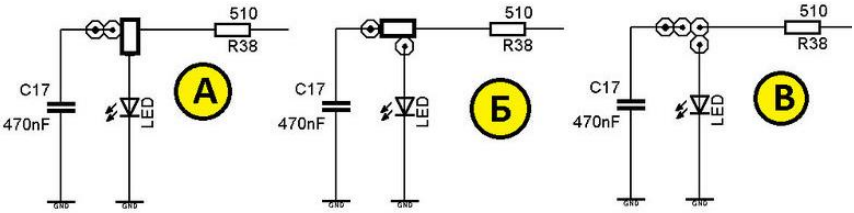



Рис. 3.29 – Варианты подключения светодиода и фильтра

На рисунке 3.29 показана схема коммутации одного канала, где:

**А**. Подключен светодиод. В режиме ШИМ можно изменять яркость светодиодов.

**Б.** Подключен конденсатор, образуя низкочастотный RC-фильтр, позволяющий преобразовать выходной сигнал ШИМ в аналоговое напряжение. Светодиод при этом отключается.

**В.** Свободный выход через резистор.

Режим ШИМ можно реализовать программно или с помощью таймера МК. Связь с микроконтроллером выполняется с помощью шнуровой коммутации.

Расположение блока светодиодов с фильтрами – 1 на рисунке 3.27 (выше разъема USB2).

Макетирование недостающих узлов отладочного модуля можно реализовать с помощью **монтажной панели с резисторами и транзисторами** (рисунок 3.30). Это небольшая цанговая матрица (2 на рисунке 3.27), нижний горизонтальный ряд которой соединен с землей, а к верхнему – можно подключить питание, расположенное чуть левее. Сверху находятся три многооборотных

потенциометра (3 на рисунке 3.27), разведенных на гнезда и штыри, а также два транзистора PNP и NPN структур. **В качестве PNP используется BC858**, а **NPN – BC817**. В базовых выводах уже впаяны резисторы по 10кОм. Слева стоит 2 N-канальных маломощных MOSFET IRLML2502L.

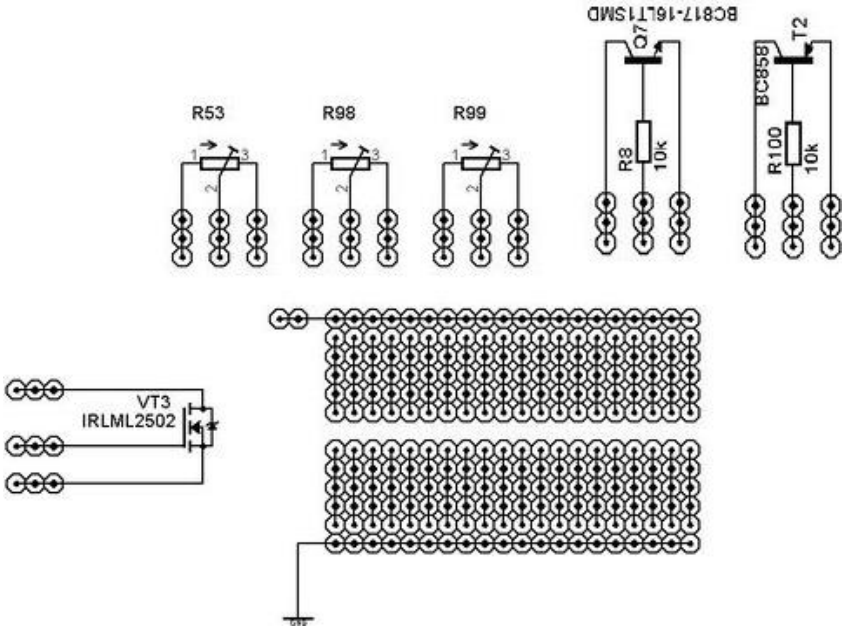



Рис. 3.30 – Монтажная панель

### Блок индикации

В состав блока индикации входят светодиодные индикаторы, семисегментный и алфавитно-цифровой индикаторы, звуковой индикатор.

**Светодиодные индикаторы** отражают включение/выключение линий питания и могут использоваться для отображения состояния реализуемых процессов. Они располагаются в различных местах платы отладочного модуля (рисунок 3.31).

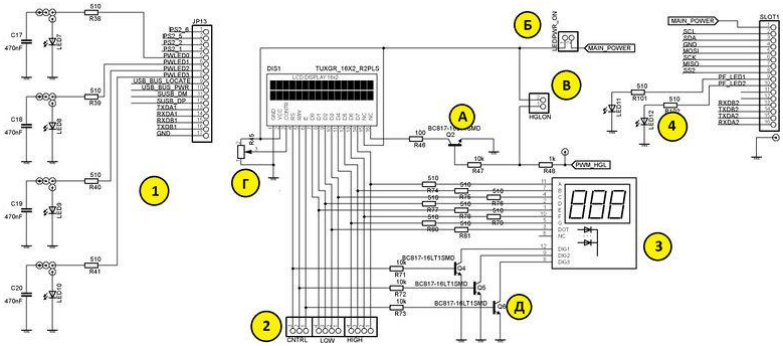



Рис. 3.31 – Электрическая схема блока индикации

Для задач пользователя доступны:

- **Светодиоды LED7÷LED10**, схема соединения которых показана на рисунке 3.29 (А). На плате отладочного модуля они обозначаются как **LED0÷LED3** («1» на рисунке 3.27).
- **Светодиоды LED11, LED12** («4» на рисунке 3.31) и («4» на рисунке 3.27), которые присоединены к разъему расширения. Обозначение на плате – **LED4**, **LED5**.
- **Bar индикатор на 10 светодиодов** (рисунок 3.32). Выводы индикатора подключены к штырям и могут использоваться для любых целей. На плате они обозначены – **LED Bar** («5» на рисунке 3.27).

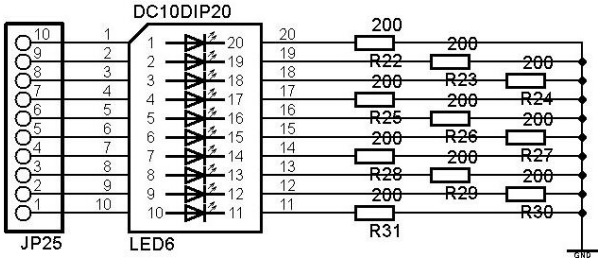



Рис. 3.32 **–** Электрическая схема индикатора

Для отображения алфавитно-цифровой информации служат полупроводниковый **четырехразрядный 7-сегментный индикатор**

**с общим катодом 7-SEG Led** («3» на рисунке 3.31) и **модуль жидкокристаллического индикатора LCD WH1602-YYH-CTK** с контроллером HD44780 (рисунок 3.33). Связь индикаторов с БМК выполняется через общий разъем («2» на рисунке 3.31). Поэтому **одновременная работа этих индикаторов невозможна.**

На плате («6» на рисунке 3.27) показан этот разъем и нанесена таблица соответствия выводов модуля индикатора и сегментов.

Выбор нужного разряда семисегментного индикатора осуществляется подачей напряжения на транзисторы (**«Д»** на рисунке 3.31), подключающие катоды индикатора к земле. Разряды именуются слева направо как DIG1 (RS), DIG2 (RW), DIG3 (E), DIG4 (CP) (рисунок 3.31).

Управление модулем **LCD** выполняется с помощью линий:

- RS – идентификация данные/команда;
- RW – выбор чтения или записи;
- E – синхронизация;
- D0…D7 – данные HD44780;
- NC, NC – включение подсветки. Управление подсветкой выполняется транзистором (**«А»**, рисунок 3.31), на базу которого следует подать положительное напряжение джампером HGL ON (**«В»**, рисунок 3.31) или внешним сигналом PWM_HGL;
- CONTR – напряжение смещения, управляющее контрастностью. Регулировка контрастности выполняется потенциометром (**«Г»**, рисунок 3.31).

Питание на LCD подключается джампером LCD PWR ON (**«Б»**, рисунок 3.31).

Модуль индикатора устанавливается на колодку в верхней части платы («7» на рисунке 3.27), рядом с которой находится соответствующая надпись (рисунок 3.33).

На рисунке 3.33 показано соединение модуля **LCD** с микроконтроллером: RS → PA12, R/W → PA13, E → PA14, D4 →

D7 соответственно PA8 → PA11. Особенности использования модуля приведены в разделе 3.2.2.

Для **звуковой сигнализации** событий предназначен **зуммер электромагнитного типа HCM (BEEPER)** (**«8»** на рисунке 3.27). Он характеризуется интенсивностью звука (75 дБ, 80 дБ, 85 дБ), частотой резонанса (2048 Гц÷2400 Гц), рабочим напряжением (1,5 В÷12 В) и силой тока (12 мА÷70 мА) [49]. Для управления зуммером необходимо с помощью микроконтроллера сформировать генератор с указанной выше частотой на выходе **BZR** БМК. Согласование по мощности выполняется транзистором Q3 (рисунок 3.34).

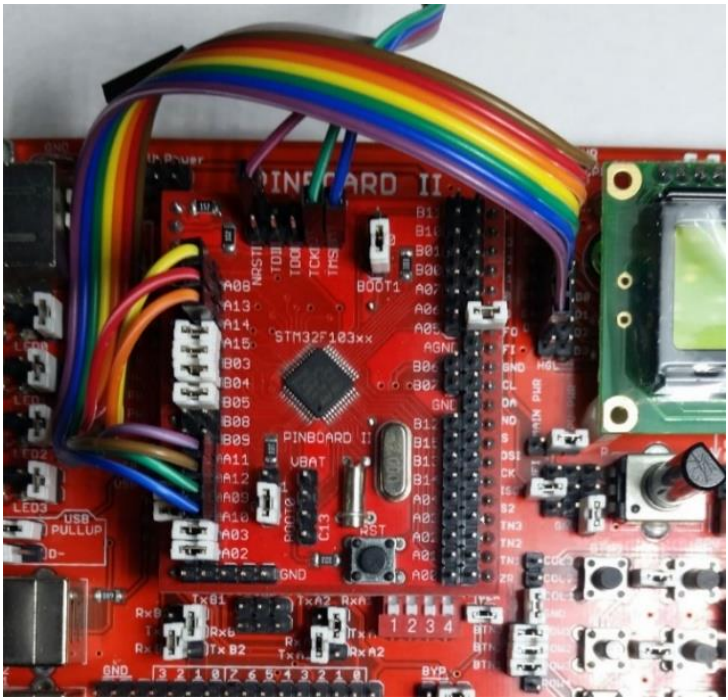



Рис. 3.33 – Соединение модуля индикации с микроконтроллером

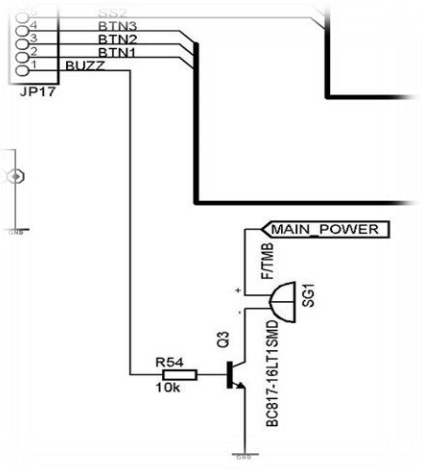



Рис. 3.34 – Электрическая схема зуммера

### Расширитель внутренней шины

Расширитель внутренней шины предназначен для подключения внешних устройств. Он состоит из двух разъемов.

**Разъем интерфейса PS/2** можно использовать для подключения клавиатуры, мышки, поддерживающих этот стандарт, или как разъем общего применения. Физически интерфейс PS/2 включает четыре проводника: землю, питание, линию данных и линию синхронизации. С помощью джампера можно установить питание 5 В или 3,3 В. Размещение на печатной плате показано цифрой «9» на рисунке 3.27, а схема соединений с БМК – на рисунке 3.35.

**Разъем расширения** пользователя позволяет подключать внешние устройства с интерфейсами SPI, $I^2C$, USART или использовать эти линии БМК для реализации других функций. На базе перечисленных интерфейсов разработаны датчики различного назначения, часы реального времени, блоки клавиатуры и индикации и так далее. В частности, к этому разъему можно подключить **модуль Ethernet на базе сетевого адаптера ENC28J60 с интерфейсом SPI.**

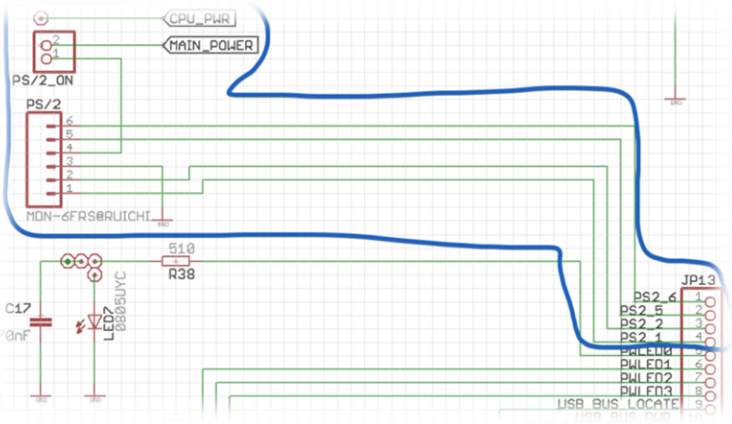



Рис. 3.35 – Схема соединения разъема **PS/2** с БМК

Дополнительно на этот разъем выведены линии питания, земли, пара светодиодов и переключателей. Разъем расширения обозначен цифрой «10» на рисунке 3.27.

### 3.1.5 Блок питания

Блок питания подробно описан в [56]. В данном разделе отмечены только основные сведения, необходимые пользователю.

Обобщенная структура системы питания приведена на рисунке 3.36.

Питание на плату может быть подключено от одного из трех внешних источников: компьютера (линии питания **USB** – 5 В), штекера **JACK** (не более 12 В), клеммной колодки **TERM** (5 В). Используемый источник питания с помощью перемычки, установленной на селекторе источника (**PWR Source Select**), коммутируется на кнопку **POWER ON** включения питания. Подключение источников питания индицируется соответствующими светодиодами, рядом с которыми на плате имеются надписи (рисунок 3.37).

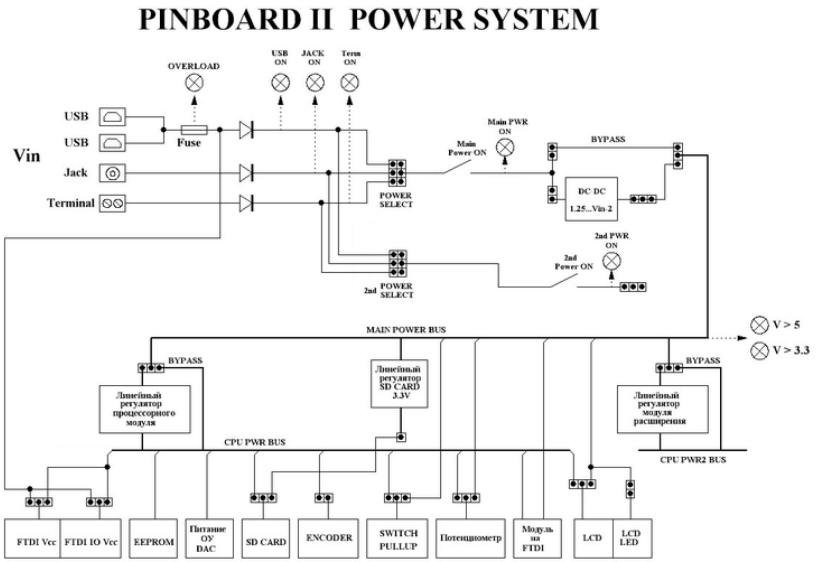



Рис. 3.36 – Структура блока питания

**В лабораторных работах используется питание USB компьютера, которое поступает на плату с одного из разъемов USB1 или USB2. Одновременно эти разъемы включать нельзя.**

На плате формируется несколько шин питания.

Шина **MainPower** служит для подключения основных периферийных устройств и выведена на разъем расширения. Значение напряжения на этой шине определяется состоянием **входного селектора** стабилизатора **DC-DC** (**«1»** на рисунке 3.37).

В положении **(А) BYP** (BYPASS – обход) напряжение будет соответствовать подключенному внешнему источнику. Если установить перемычку в положение **(Б) REG** (Regulator), то входное напряжение поступает на вход стабилизатора, а на выходе оно будет **приблизительно на 2 В меньше, чем входное.** Коммутация этих двух состояний на выходном селекторе выполняется с помощью контактов «**Г**», «**Д**»**,** средние штыри которых подключены к **MainPowe**r.

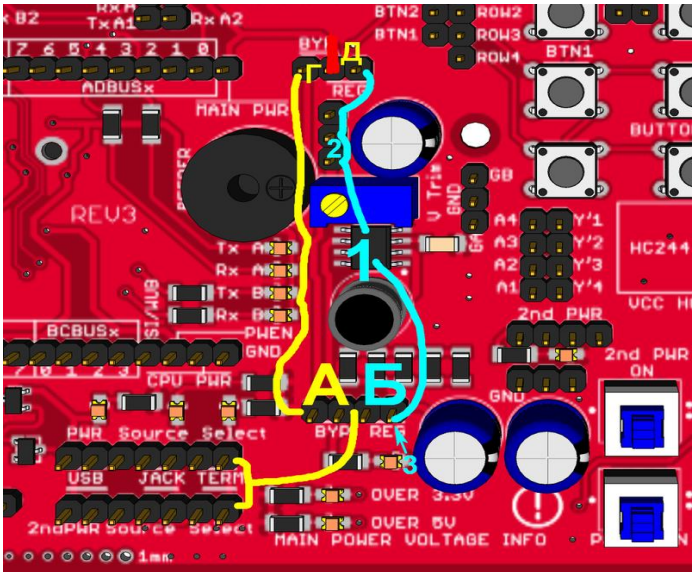



Рис. 3.37 – Формирование напряжения шины **MainPower**

Если перемычки установить в положения «**А**» и «**Г**» (**BYP**), то на шине будет напряжение питания USB, а в положении «**Б**» и «**Д**» – выходное напряжение стабилизатора. Выходное напряжение стабилизатора можно регулировать потенциометром (синий квадрат) и контролировать на **контактах «2»**. **Регулировать выходное напряжение стабилизатора рекомендуется при отключенной перемычке «Д».**

Напряжение питания **микроконтроллера** задается с помощью стабилизатора «**11**» (рисунок 3.27).

На выходе стабилизатора формируется **шина CPU PWR**, к которой могут быть подключены и периферийные устройства модуля. Плавной регулировки выходного напряжения нет, а резисторы, задающие напряжение шины **CPU PWR**, находятся на плате блока контроллера**. Поэтому нельзя включать питание платы без блока контроллера, так как это может привести к неконтролируемому повышению напряжения**.

**Шину CPU PWR рекомендуется использовать для работы с внешними подтягивающими резисторами, потенциометрами, энкодером, чтобы их выходные сигналы были в диапазоне питания микроконтроллера.**

Рядом с процессорным блоком находятся два **набора контактов питания Main Power** («**12**», рисунок 3.27) и **CPU Power** («**13**», рисунок 3.27), напряжения с которых можно использовать для задач пользователя.

На входе стабилизатора шины **CPU PWR** установлен **селектор** («**14**», рисунок 3.27), работающий аналогично описанному выше. В положении **BYP** шины **Main Power и CPU Power объединяются, минуя стабилизатор.** Этот режим можно использовать с процессорным блоком AVR при питании от USB. **С процессорным модулем STM32 этот режим не рекомендуется, так как, по неосторожности можно «сжечь» микроконтроллер. Состояние селектора можно увидеть через отверстие платы (рисунок 3.38).**

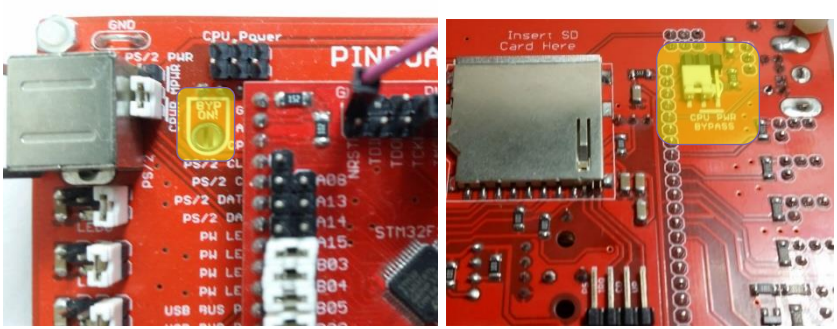



Рис. 3.38 – Селектор на выходе стабилизатора шины CPU PWR

Джампер подачи питания **на PS/2 («15»**, рисунок 3.27**)** позволяет коммутировать шины **Main Power** и **CPU Power**.

Для работы с **SD картой** предусмотрено питание 3.3 В, которое формируется **стабилизатором** («**16**», рисунок 3.27). **Селектор SD карты должен быть установлен в положение 3V3.**

**При подключении ЖКИ (LCD) (см. «17»** на рисунке 3.27) **необходимо учитывать, что питание контроллера ЖКИ должно находится в диапазоне (6÷3,7) В, а подсветки – (6÷4) В. Подсветка ЖКИ LCD LED может быть подключена только к шине Main Power.**

**Для модуля STM32** при подаче питания с выхода стабилизатора (**положение REG на входном и выходном селекторах стабилизатора**) максимальное значение, которое можно выставить регулировкой потенциометра, на **шине Main Power – 3,12 В, а на CPU Power – 2,26 В.** Для этого режима **питание ЖКИ можно взять на входном селекторе с контакта BYP (внизу платы) или 5V рядом с FTDI и присоединить к средней ножке селектора ЖКИ LCD PWR («17»**, рисунок 3.27**).**

**В положении BYP напряжение питания ЖКИ должно быть подключено к шине Main Power. В этом положении напряжение на шине Main Power – 4,9 В, а на CPU Power – 3,3 В.**

**Шина вторичного питания 2nd PWR** предназначена для подключения дополнительного источника с требуемым напряжением, если в этом есть необходимость. Селектор **2nd Power Select** находится в нижней части платы и коммутируется аналогично Power Select («18», рисунок 3.27). Кнопка включения вторичного питания 2nd Power On находится выше кнопки **Power On.**

Расположение перемычек при питании отладочного модуля USB-портом компьютера на рисунке 3.39 выделено желтым цветом.

В отладочном модуле PINBOARD II rev.3 предусмотрена индикация питания перегрузки по току, а также индикация включенного первичного и вторичного питания. Данные индикаторы позволяют визуально оценить уровень напряжения в системе главного контура питания, чтобы не вывести из строя устройства, расположенные на отладочном модуле.

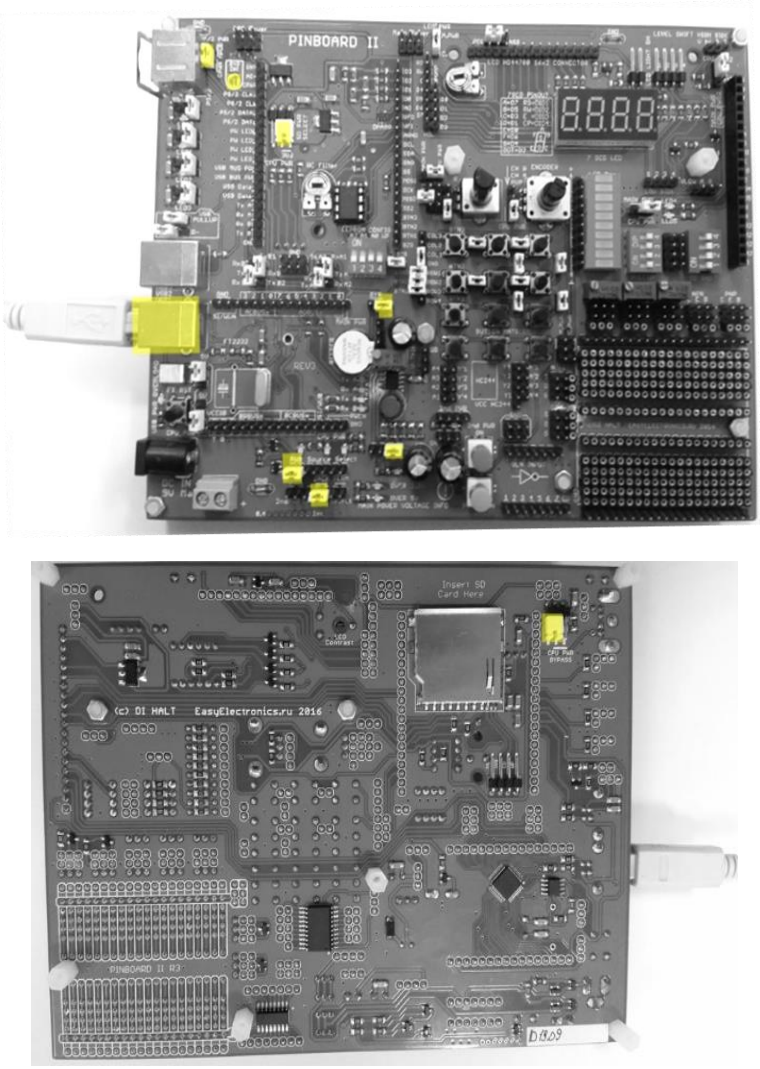



Рис. 3.39 – Перемычки системы питания

Интенсивность свечения светодиода «OVER 3.3V» зависит от напряжения и чем оно выше, тем ярче горит светодиод: при 3 В свечение светодиода тусклое, а при превышении 3.3 В – яркое.

Второй светодиод «OVER 5V» работает от 4.8 В и, также как светодиод «OVER 3.3V», свечением сигнализирует о превышении допустимых значений напряжения (рисунок 3.40).

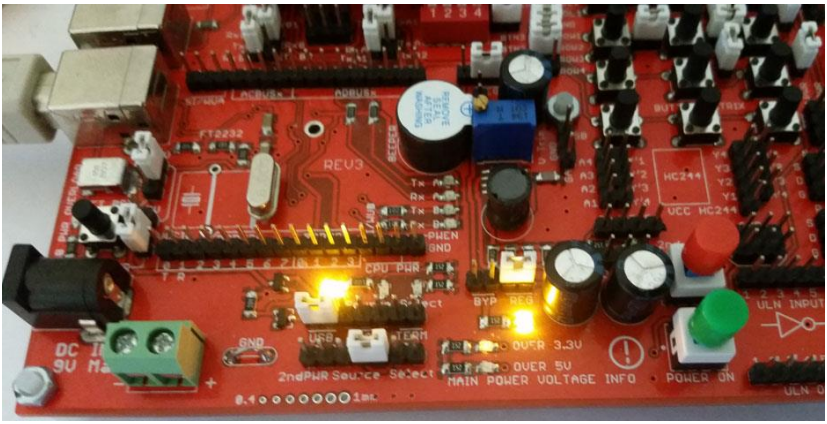



Рис. 3.40 – Индикация питания и превышения допустимых значений

## 3.2 Особенности работы с внешними устройствами

### 3.2.1 Модуль индикации на базе HD-44780

Основные характеристики модуля: формат экрана (количество символов, количество строк). размер символа, размер модуля, параметры жидкого кристалла, режим работы, тип и цвет подсветки, цвет жидкого кристалла, варианты знакогенератора, температурный диапазон, напряжение питания, ток потребления, интерфейс контроллера [102]

В лабораторных работах используется модуль WH1602-YYH-CTK, который поставляется в составе набора отладочных модулей PINBOARD II rev.3. Внешний вид модуля представлен на рисунке 3.41.


Рис. 3.41 – Внешний вид модуля WH1602-YYH-CTK

Основные характеристики модуля индикации:

- формат экрана 16x2;
- вид символа – матричный, размер 5x8 или 5x11 точек;
- тип дисплея – LCD STN (super-twisted nematic display);
- экран – позитивный, желто-зеленый;
- подсветка – LED (светодиодная);
- алфавит – русский, английский;
- интерфейс – параллельный (8 или 4 разряда);
- напряжение питания (4,5÷5,5) В;

- рабочая температура от –20 до +70 °С;
- размер модуля 80х36 мм, видимая область 66х16 мм;
- размер символа 2.95х5.55 мм.

Назначения выводов модуля приведены в таблице 3.3.

Таблица 3.3 Назначение контактов модуля

| № выхода | Обозначение | Функция | | Примечание |
|---|---|---|---|---|
| 1 | VSS | Power Supply | 0V | |
| 2 | VDD | | +5V | |
| 3 | VEE | | For LCD | переменное |
| 4 | RS | Register Select (H: Data L: Instruction) | | |
| 5 | R/W | L: MPU to LCM H: LCM to MPU | | |
| 6 | E | Enable | | |
| 7 | DB0 | Data Bit 0 | | |
| 8 | DB1 | Data Bit 1 | | |
| 9 | DB2 | Data Bit 2 | | |
| 10 | DB3 | Data Bit 3 | | |
| 11 | DB4 | Data Bit 4 | | |
| 12 | DB5 | Data Bit 5 | | |
| 13 | DB6 | Data Bit 6 | | |
| 14 | DB7 | Data Bit 7 | | |
| 15 | A | Anode of LED Unit | | |
| 16 | K | Catode of LED Unit | | |

1. **VSS** – общий провод или «земля».
2. **VDD** – напряжение питания.
3. **VEE** –регулировка контрастности. Контрастность зависит от значения напряжения, поданного на этот вход. Как правило, для задания напряжения (регулировки контрастности дисплея),

используется переменный резистор 10 кОм, подключенный к линиям земли и питание, средний вывод которого соединен с VEE.

4. **RS** – сигнал, указывающий вид информации, передаваемой по шине данных: RS = 0 –команда; RS = 1 – данные.

5. **RW** –сигнал чтения/записи (запись – RW = 0, чтение – RW = 1).

6. **E** – сигнал синхронизации, по срезу (переход из «1» в «0») которого выполняется запись/чтение данных.

7. **Входы/выходы DB0÷DB7** – шина данных.

8. **A (NC+), K** (**NC-**) – анод и катод для подачи напряжения питания светодиодной подсветки дисплея. Рекомендуется подключать к источнику напряжения 5 В через токоограничивающий резистор 100 Ом.

Упрощенная структура модуля индикации приведена на рисунке 3.42.

Процессорный блок **ПБ** является специализированным контроллером, обеспечивающим

- организацию взаимодействия с МК;
- прием и дешифрацию команд;
- запись отображаемых символов в видеопамять;
- формирование изображения;
- управление жидкокристаллическим индикатором.

**ПБ** обеспечивает взаимодействие блока связи с интерфейсом БСИ с внутренними ресурсами контроллера: регистром данных **РгД (DR),** регистром команд **РгК** (**IR)**, видеопамятью **ВП** (**DDRAM)**, знакогенераторами **ОЗУ ЗГ (CGRAM)** и **ПЗУ ЗГ (CGROM)**, счетчиком адреса **СчА,** флагом готовности **BF.**

Текущее значение счетчика **СчА** и флага готовности **BF** хранится в программно доступном регистре **IR.**

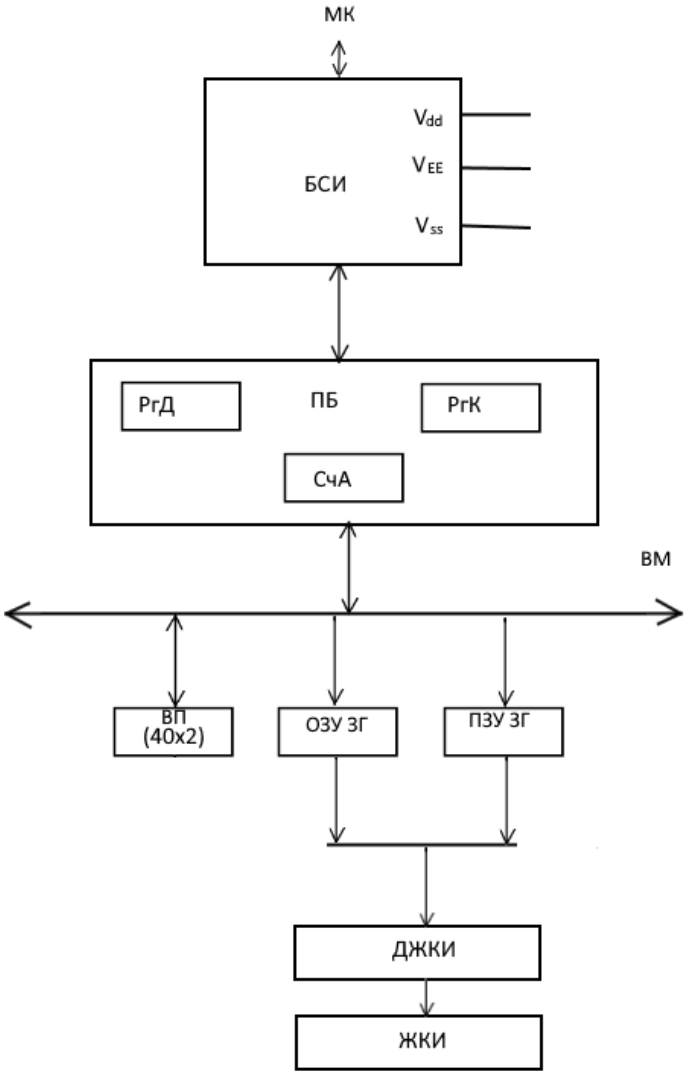



Рис. 3.42 – Упрощенная структура модуля индикации с контроллером HD44780

Видеопамять **ВП** является основным местом хранения **выводимой информации. Объем памяти 80 байт (40 – первая**

**строка, 40 – вторая). Отображаемые символы должны быть записаны в ASCII коде.**

**На экране ЖКИ отображается только часть видеопамяти.** Для используемого модуля – это 16 байт в первой и второй строках.

**Доступ к другим элементам ВП обеспечивается изменением содержимого текущего значения счетчика адреса СчА.** Это позволяет заранее записывать требуемые символы по различным адресам, а затем, изменяя содержимое **СчА**, вызывать требуемую информацию.

**Место вывода следующего символа указывает курсор, который может быть включен или отключен, отображаться в виде подчеркивания снизу или мерцающего знакоместа.**

**Преобразование ASCII символов в изображение выполняется с помощью таблицы, хранящейся в ПЗУ ЗГ. Содержимое ячеек видеопамяти является адресом ПЗУ ЗГ.**

**ПЗУ ЗГ формируется при изготовлении модуля на заводе, и пользователь изменить его не может.** Поэтому при выборе модуля необходимо контролировать наличие требуемого алфавита.

Англо-русская таблица ПЗУ ЗГ представлена на рисунке 3.43 [88].

Традиционно первая половина ПЗУ с адресами 00÷7Fh содержит начертания цифр, знаков препинания, а также заглавных и строчных букв латинского алфавита. Вторая половина содержит начертания символов национального алфавита.

Для выбора соответствующего символа необходимо указать в команде его адрес, который формируется из номера столбца (старшая тетрада) и номера строки (младшая тетрада). Например, код 0x47 соответствует символу «G», а код 0xA1 – букве «Г». **Кодирование русских букв выполняется только по адресу таблицы, английские буквы можно отображать (‘d’), а текст помещать в кавычки (“Hello, student@”).**

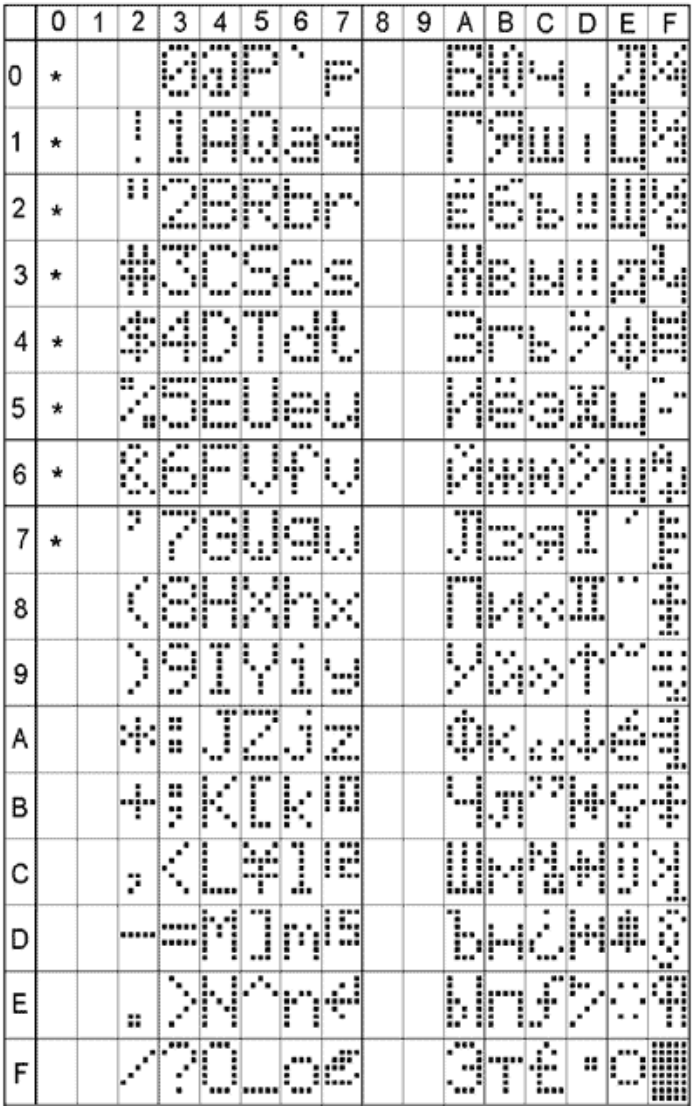


Рис. 3.43 – Таблица ASCII-кодов и символов контроллера HD44780

При отсутствии требуемых символов в ПЗУ ЗГ их можно синтезировать в ОЗУ ЗГ, объем которого 64 байта.

Для программирования доступны 8 переопределяемых символов в режиме с матрицей 5x7 точек и 4 с матрицей 5x10 (в режиме 5x10 переопределяемые символы адресуются кодами **ВП** через один 00h, 02h, 04h, 06h).

Чтобы определить собственный символ, необходимо установить СчА на адрес начала матрицы требуемого символа в ПЗУ ЗГ – 00h, 08h, 10h и т.д. (00h, 10h, 20h для режима 5x10 точек) и произвести перезапись всех байтов матрицы, начиная с верхней строки.

Для кодирования матрицы 5x7 используются 8 байт, пять младших бит которых несут информацию о рисунке (1 – сегмент включен, 0 – сегмент выключен), старшие три бита не используются. Матрица программируемых символов допускает использование полной высоты строки (8 строчек для режима 5x7 и 11 строчек для режима 5x10), но **в области подчеркивающего курсора (нижняя строка) размещать информацию не рекомендуется.**

На рисунке 3.44 показана схема формирования символов в памяти ОЗУ ЗГ

| EPROM Address | | | | | | | | | | | Data | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | O4 | O3 | O2 | O1 | LSB O0 |
| | | | | | | | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| | | | | | | | | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | | | | | | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | | | | | | | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| | | | | | | | | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| | | | | | | | | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| | | | | | | | | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Cursor position

Character code

Line position

Рис. 3.44 – Схема формирования символов в ОЗУ ЗГ

Информация с ПЗУ ЗГ и ОЗУ ЗГ поступает на драйверы ДЖКИ, формирующие определенную временную диаграмму для управления жидкокристаллическим индикатором ЖКИ (рисунок 3.42).

**Блок связи с интерфейсом БСИ** обеспечивает согласование работы модуля с микроконтроллером МК. На вход модуля поступают сигналы RS, WR, E, DB0-DB7, формируемые микроконтроллером. Подключение к МК может быть выполнено на основе асинхронного обмена с использование флага готовности **BF**, синхронного обмена, с 8- или 4-разрядной шиной данных [102].

Временная диаграмма взаимодействия микроконтроллера с HD44780 по 8-битовой шине представлена на рисунке 3.45.

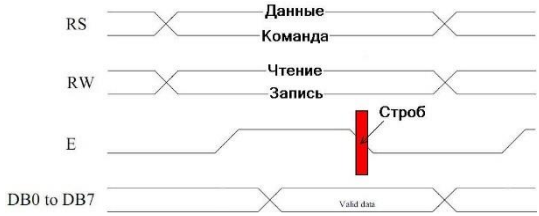



Рис. 3.45 – Временная диаграмма передачи команд и данных по 8-битовой шине

Особенностью формирования временной диаграммы является необходимость согласования работы внутреннего генератора модуля, работающего с частотой (250 ±50) КГц, и системного генератора МК, частота которого определяется типом микроконтроллера и его режимом работы.

Поэтому для реализации взаимодействия с МК требуется выполнение определенных временных соотношений.

**Длительность уровней нуля и единицы, формируемых МК, должна быть больше периода внутреннего генератора модуля.**

**Очередная команда должна быть передана после выполнения предыдущей.**

**При асинхронном обмене** программа МК должна проверить значение флага готовности **BF**, находящегося в старшем разряде **СчА**.

**При синхронном обмене** пауза между командами должна быть **больше времени выполнения самой медленной команды** (см. таблицу 3.4).

**Наиболее рационально использование синхронного режима с 4-битной шиной данных.** Его особенностью является:

- минимальное количество линий связи с МК (RS, E, BD7-BD4, WR соединяется с землей модуля);
- передача команд и данных выполняется в режиме с разделением времени (сначала старшая тетрада, а затем – младшая);
- использование постоянной временной задержки межу командами.

Пример подключения в этом режиме модуля индикации к МК STM32F103C6 приведен на рисунке 3.46.

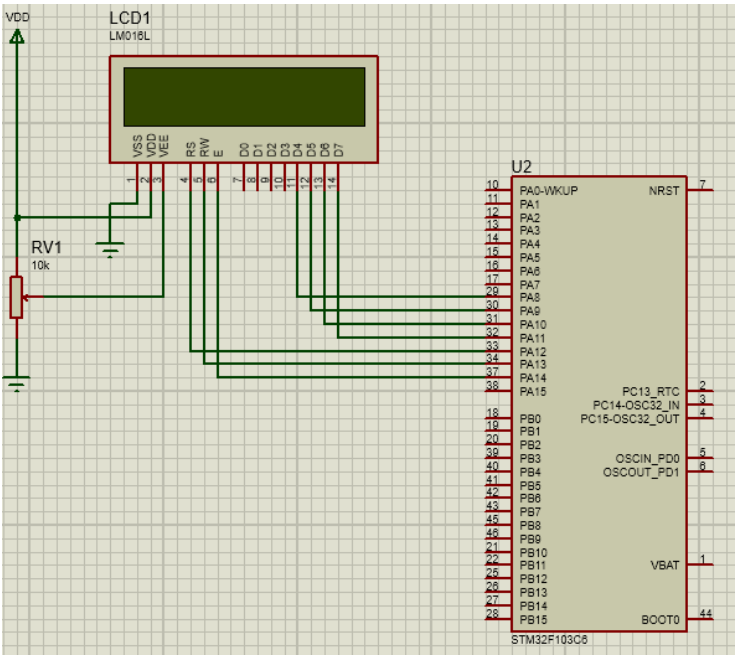



Рис. 3.46 – Подключение контроллера HD44780 к STM32F103C6

Вход RW можно заземлить, а не присоединять МК. Временная диаграмма, реализуемая при синхронном обмене информацией через 4-битную шину данных, представлена на рисунке 3.47 [87].

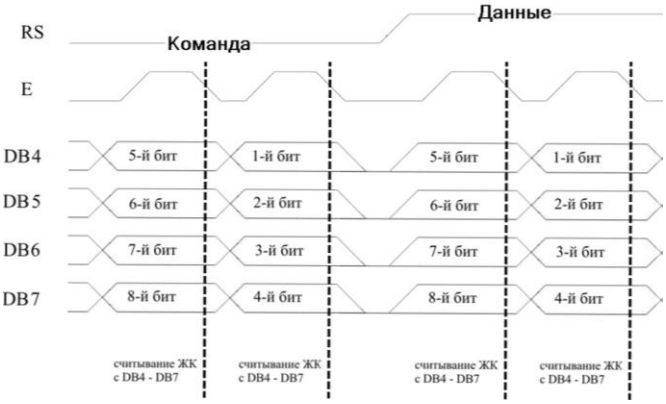



Рис. 3.47 – Временная диаграмма синхронной передаче информации по 4-разрядной шине

**Электрическая схема и пример соединения HD44780 с блоком микроконтроллера оценочного модуля приведены в разделе 3.1.4 (рисунок 3.31, рисунок 3.33).**

Таблица 3.4 Система команд модуля с **HD44780**

| Код | | | | | | | | | | Описание команды | Время выполнения |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **RS** | **R/W** | **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** | | |
| 0 | 0 | 0 | 0 | 1 | **D/L** | **N** | **F** | * | * | Определение параметров развертки и ширины шины данных | 40 мкс |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | **D** | **C** | **B** | Вкл./выкл. дисплея; управление курсором | 40 мкс |

*Окончание табл. 3.4*

| Код | | | | | | | | | | Описание команды | Время выполнения |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **RS** | **R/W** | **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **I/D** | **S** | Направление сдвига курсора или дисплея | 40 мкс |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Очистка дисплея, **СчА (AC) = 0**, адресация **АС** на **ВП (DDRAM)** | 82 мкс – 1.64 мс |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ***** | Возврат в начальное положение. **AC = 0**, адресация на **DDRAM**, сброшены сдвиги | 40 мкс – 1.6 мс |
| 0 | 0 | 0 | 0 | 0 | 1 | **S/C** | **R/L** | ***** | ***** | Сдвиг курсора/дисплея | 40 мкс |
| 0 | 0 | 0 | 1 | **$A_{CG}$** | | | | | | Установка адреса **ПЗУ ЗГ (CGRAM)** | 40 мкс |
| 0 | 0 | 1 | **$A_{DD}$** | | | | | | | Установка адреса **ОЗУ ЗГ (DDRAM)** | 40 мкс |
| 0 | 1 | **BF** | **AC** | | | | | | | Чтение флага готовности **BF** и счетчика адреса **AC** | 40 мкс |
| 1 | 0 | Данные | | | | | | | | Запись данных в **CGRAM** или **DDRAM** | 40 мкс |
| 1 | 1 | Данные | | | | | | | | Чтение данных из **CGRAM** или **DDRAM** | 40 мкс |

Программирование модуля выполняется с помощью команд, представленных в таблице 3.4, а назначение флагов, используемых в командах, отражается в таблице 3.5.

Таблица 3.5. Флаги, управляющие работой модуля с **HD44780**

| Флаг | Назначение | Значение Флага | |
|---|---|---|---|
| | | **0** | **1** |
| **I/D** | Режим смещения счетчика адреса **AC** | Уменьшение | Увеличение |
| **S** | Режим сдвига содержимого дисплея (*) | Сдвиг дисплея не производится | После записи в **DDRAM** очередного кода дисплей сдвигается в направлении, определяемым флагом **I/R**: 0 – вправо; 1 – влево |
| **S/C** | Производит вместе с флагом **R/L** операцию сдвига содержимого дисплея или курсора. Определяет объект смещения (*) | Сдвигается курсор | Сдвигается дисплей |
| **R/L** | Производит вместе с флагом **S/C** операцию сдвига дисплея или курсора. Уточняет направление сдвига | Влево | Вправо |

*Окончание табл. 3.5*

| Флаг | Назначение | Значение Флага | |
|---|---|---|---|
| | | **0** | **1** |
| **D/L** | Флаг, определяющий ширину шины данных | 4 разряда | 8 разрядов |
| **N** | Режим развертки изображения на ЖКИ | Одна строка | Две строки |
| **F** | Размер матрицы символов | 5x8 символов | 5x10 символов |
| **D** | Наличие изображения | Выключено | Включено |
| **C** | Курсор в виде подчерка | Выключен | Включен |
| **B** | Курсор в виде мерцающего знакоместа | Выключен | Включен |

При сдвиге не производится изменение содержимого **DDRAM**, изменяются только внутренние указатели расположения видимого начала строки **DDRAM**. После подачи питания модулю флаги принимают значения, указанные в таблице 3.6.

Таблица 3.6. Значения управляющих флагов после подачи питания

| Флаг | Значение | Режим |
|---|---|---|
| **I/D** | 1 | Режим увеличения счетчика на 1 |
| **S** | 0 | Без сдвига изображения |
| **D/L** | 1 | 8-разрядная шина данных |
| **N** | 0 | Режим развертки одной строки |
| **F** | 0 | Символы с матрицей 5x8 точек |
| **D** | 0 | Отображение выключено |
| **C** | 0 | Курсор в виде подчерка выключен |
| **B** | 0 | Курсор в виде мерцающего знакоместа выключен |

На основании этих таблиц можно настроить модуль на требуемый режим работы. Так как на момент включения контроллера **HD44780** дисплей ничего не отображает (флаг **D = 0**), то для того, чтобы вывести какой-либо текст необходимо, как минимум, включить изображение, установив флаг **D = 1**.

Некоторые типовые операции модуля приведены в таблице 3.7.

Таблица 3.7 Типовые операции модуля

| Команда ЖКИ | HEX-код | Выполняемые действия | Время выполнения, мкс |
|---|---|---|---|
| Очистка дисплея | 0x01 | Пустой экран, очистка памяти, курсор в левой верхней позиции | 1640 |
| Возврат курсора в начало | 0x02 | Курсор в левой верхней позиции, память не очищается | 1640 |
| Сдвиг курсора влево | 0x04 | После вывода очередного символа курсор автоматически сдвигается на одно знакоместо влево | 40 |
| Сдвиг курсора вправо | 0x06 | После вывода очередного символа курсор автоматически сдвигается на одно знакоместо вправо | 40 |
| Выключение дисплея | 0x08 | Полное отсутствие изображения на экране ЖКИ | 40 |
| Выключение курсора | 0x0C | Разрешается вывод изображения, но курсор не виден | 40 |

*Окончание табл. 3.7*

| Команда ЖКИ | HEX-код | Выполняемые действия | Время выполнения, мкс |
|---|---|---|---|
| Прямоугольная форма курсора | 0x0D | Разрешается вывод изображения, курсор в виде темного мигающего прямоугольника | 40 |
| Линейная форма курсора | 0x0E | Разрешается вывод изображения, курсор в виде нижней подстрочной немигающей линии | 40 |
| Комплексная форма курсора | 0x0F | Разрешается вывод изображения, курсор в виде мигающего прямоугольника с подчеркиванием | 40 |
| Интерфейс 4 бита, 1 строка | 0x20 | Связь с однострочным ЖКИ через 4 линии шины данных (ШД) | 40 |
| Интерфейс 4 бита, 2 строки | 0x28 | Связь с двухстрочным ЖКИ через 4 линии ШД | 40 |
| Интерфейс 8 бит, 1 строка | 0x30 | Связь с однострочным ЖКИ через 8 линий шины данных | 40 |
| Интерфейс 8 бита, 2 строки | 0x38 | Связь с двухстрочным ЖКИ через 8 линий шины данных | 40 |
| Доступ к ОЗУ знакогенератора | 0x40÷0x7F | Запись данных по этим адресам позволяет создать 16 своих символов | 40 |

Различают команды прямого и косвенного действия. Первые из них занимают адреса 0x01÷0x3F и не требуют передачи данных. За вторыми (диапазон выше 0x3F) обязательно следует передача одного или нескольких байт информации. На рисунке 3.51 показаны временные диаграммы выполнения команды 0x80 (установка курсора в первое знакоместо верхней строки экрана и индикация в нем цифры «4» пересылкой кода данных 0x34).

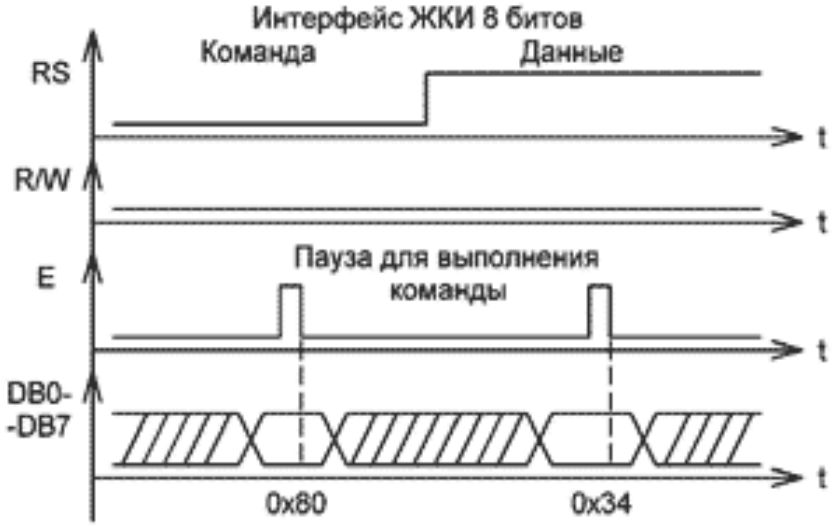



Рис. 3.48 – Временные диаграммы вывода символа «4»
на экран с 8-битной шиной

Формировать диаграммы, показанные на рисунке 3.48, должен МК с учетом задержек из таблицы 3.4, необходимых контроллеру HD44780 для выполнения команд.

Работа с модулем начинается с процедуры инициализации, которая определяется режимом работы.

Согласно [87] для 4-разрядной шины в режиме синхронного обмена процедура инициализации осуществляется по алгоритму, представленному на рисунке 3.49.

В таблице 3.8 представлена организация знакомест дисплея в памяти ВП (DDRAM), которая может быть полезна при написании функции позиционирования изображения.

Таблица 3.8. Организация знакомест дисплея в памяти DDRAM

| Display position | 1-1 | 1-2 | 1-3 | 1-4 | 1-5 | 1-6 | 1-7 | 1-8 | 1-9 | 1-10 |
|---|---|---|---|---|---|---|---|---|---|---|
| DDRAM address | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |
| Display position | 1-11 | 1-12 | 1-13 | 1-14 | 1-15 | 1-16 | | | | |
| DDRAM address | 0A | 0B | 0C | 0D | 0E | 0F | | | | |
| Display position | 2-1 | 2-2 | 2-3 | 2-4 | 2-5 | 2-6 | 2-7 | 2-8 | 2-9 | 2-10 |
| DDRAM address | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| Display position | 2-11 | 2-12 | 2-13 | 2-14 | 2-15 | 2-16 | | | | |
| DDRAM address | 4A | 4B | 4C | 4D | 4E | 4F | | | | |

**Вторая строка в области видеопамяти начинается с адреса 0x40** (2÷1).

**В четырехстрочном дисплее** адреса первой и второй строки остаются прежними (0 и 40h), а адреса третьей строки – 14h, четвертой – 54h.

**Алгоритм работы**

Определить линии МК, используемые для связи с модулем и подключить его. Модуль индикации устанавливается на разъем LCD CONNECTOR (рисунки 3.50, 3.51), затем с помощью шнуровой коммутации соединяется с МК через разъем, расположенный слева от модуля, на котором обозначены линии ввода (RS, RW, E и так далее) (рисунок 3.33).

Схема подключения интерфейсных линий, описание предварительной настройки проекта в STM32CubeMX, а также листинг программы примера работы с LCD приведены в приложении 5.

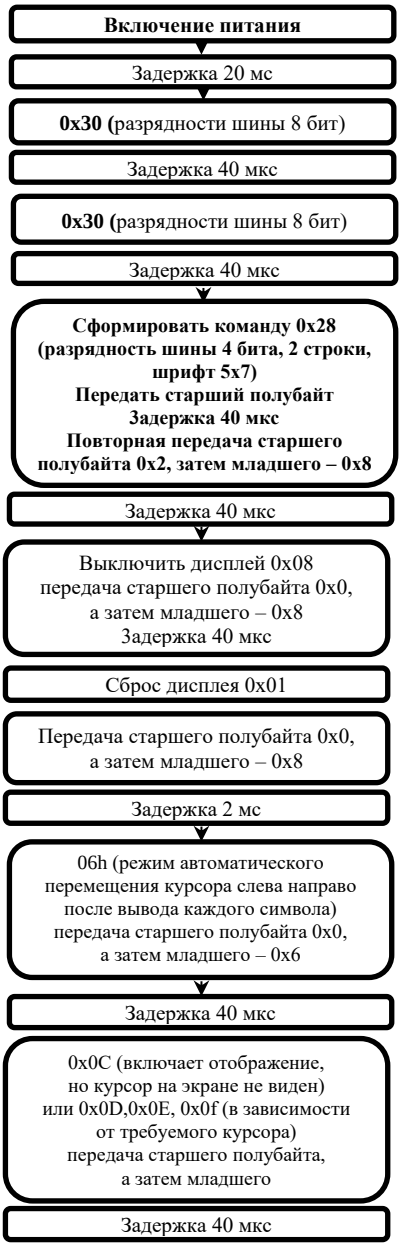



Рис. 3.49 – Алгоритм инициализации 4-битного синхронного режима работы

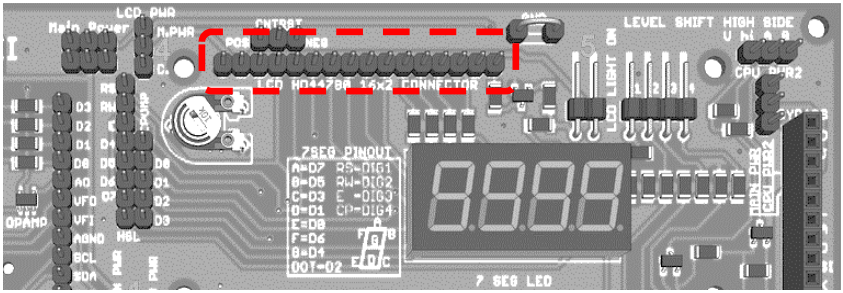



Рис. 3.50 – Место установки модуля индикации

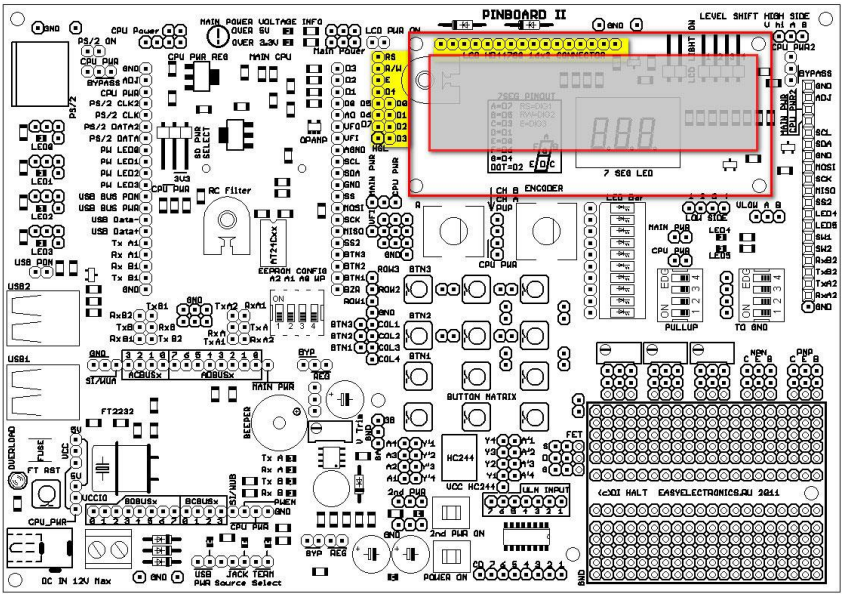



Рис. 3.51 – Оценочная плата с установленным модулем индикации

- Разработать подпрограммы (процедуры) для реализации требуемых временных диаграмм. Рекомендуемая длительность E – 4÷8 мкс, а пауза между командами – 2÷3 мс. В зависимости от стабильности генератора контроллера и условий эксплуатации (температура, номинал и вариации напряжения питания, длина

линий управления и так далее параметры временной диаграммы могут отличаться от приведенных. В системах моделирования параметры временной диаграммы определяются особенностями программной модели контроллера HD44780. Например, в Протеусе минимальная длительность E = 6 мкс, а пауза 800 мкс (при частоте МК 1 МГц). При передаче информации сначала передается старшая тетрада, а затем – младшая.

- Выполнить инициализацию модуля.
- Для вывода информации в модуль необходимо передать команду записи адреса выводимого символа и записать по этому адресу код символа. Для первой строки команда вывода имеет вид (0x80 + n), а для второй – (0xC0 + n), где n(0÷15) – номер знакоместа. Тип передаваемых данных определяется битом **RS. Кодирование русских букв выполняется только по адресу таблицы, английские буквы можно отображать (‘d’), а текст помещать в кавычки (“Hello, student”).**
- При выводе «бегущей» строки необходимо записать текст в видеопамять модуля и командами 0x18 (сдвиг строки влево), 0x1C (сдвиг строки вправо) реализовать сдвиг текста на экране. При выводе необходимо подобрать задержку для комфортного чтения текста.
- Для вывода символов, не содержащихся в ПЗУ ЗГ, выполнить следующее:
- создать таблицу отображаемого символа и разместить её в памяти команд МК. Таблица должна начинаться с верхней строки символа;
- задать адрес размещения сформированного символа в ОЗУ ЗГ. Для режима с размером символа 5x7 – адреса 00, 08, 10h, 18h и так далее;
- задать соответствующее значение счетчика СчА– 0x40, 0x48, 0x50, 0x58 и так далее;

- с выбранного адреса переписать таблицу в ОЗУ ЗГ, начиная с верхней строки;
- записать созданный символ на требуемое знакоместо, указав начальный адрес строки и номер знакоместа (начальный адрес первой строки – 0x80, второй – 0xC0);
- вывести символ на экран командой вывода данных. Параметром является адрес ASCII –таблицы. Для символов 5x7 первый символ имеет адрес 00, второй – 01, третий – 02 и т.д. Для символов 5x10 адреса соответственно 00, 02, 03 и т.д. Пример программирования символа приведен в приложении 5 [89].
- Разработать процедуру вывода требуемой символьной информации.

### 3.2.2 Подключение виртуального терминала

Собственные средства ввода/вывода оценочного модуля не всегда являются достаточными при приеме/передаче большого объема цифровой и текстовой информации.

Более универсальным является использование виртуального терминала компьютера, с помощью которого можно передавать исходные данные или параметры настройки отладочного модуля и контролировать результаты его работы.

В данном разделе рассматривается использование порта USB микроконтроллера для связи с терминальной программой компьютера.

**Связь оценочного модуля с компьютером выполняется через разъем USB2, а разъем USB1 должен быть отключен**.

Особенности подключения **USB2** отражены в разделе 3.1.2 Принципы обмена информацией по USB изложены в [45].

Установку необходимых перемычек для работы USB в режиме High Speed иллюстрирует рисунок 3.52.

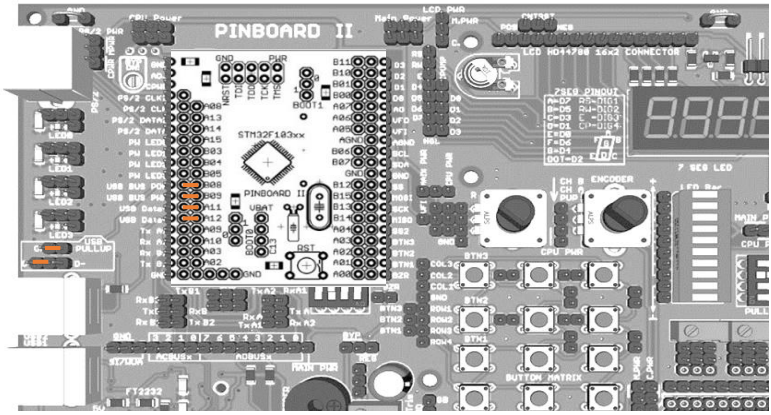



Рис. 3.52 – Подключение разъема USB2 к микроконтроллеру

С целью организации обмена данными между МК и компьютером необходимо в STM32CubeMX выполнить следующие действия:

1. Установить режим «Device full-speed» в разделе **USB → Device** (FS). Контроллер поддерживает USB 2.0 full-speed (до 12 Мбит/с), поэтому этот вариант выбран по умолчанию (рисунок 3.53).

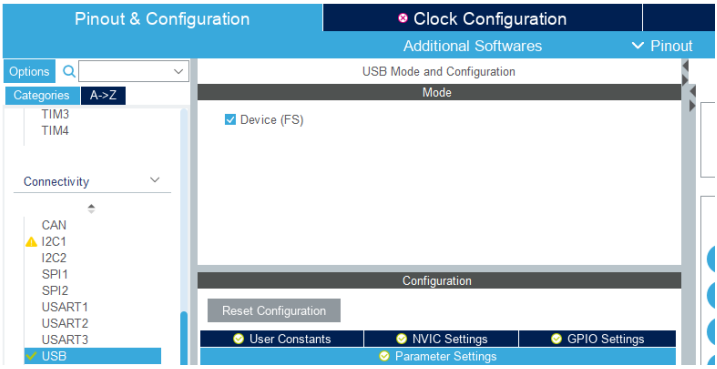



Рис. 3.53 – Окно выбора режима USB

2. В разделе **Middleware → USB_DEVICE** (рисунок 3.54) во вкладке «Mode» для параметра «Class For FS IP» выбрать «Communication Device Class (Virtual Port Com)».

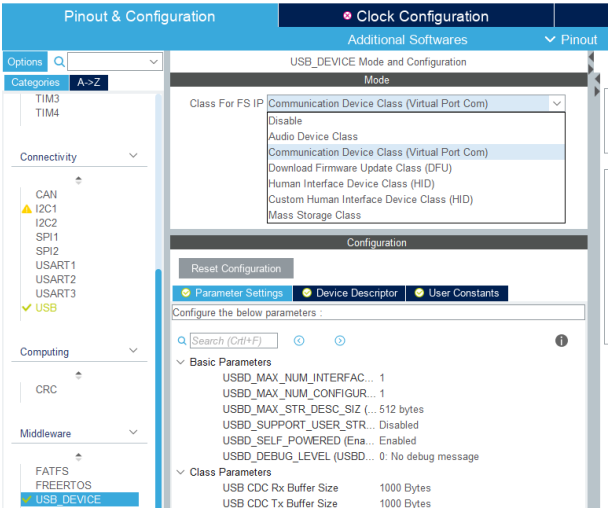



Рис. 3.54 – Настройка интерфейса USB

Ввод/вывод по прерыванию USB, как правило, включается автоматически (рисунок 3.55).

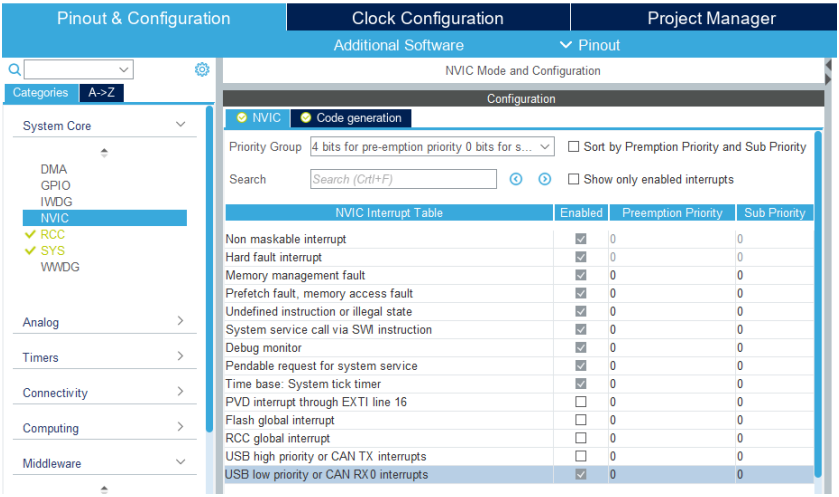



Рис. 3.55 – Настройка прерывания на USB

3. После того, как выбраны входы/выходы микроконтроллера, необходимо настроить тактирование, выбрать закладку «Clock Configuration» и, при желании, воспользоваться автоматическим помощником (рисунок 3.56).

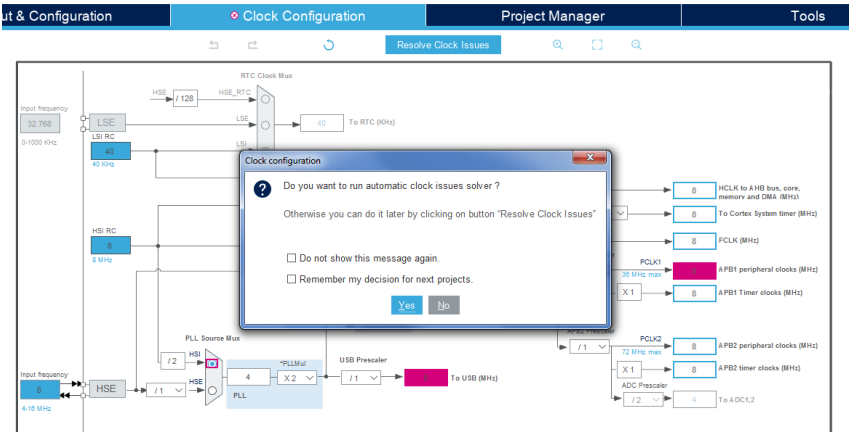



Рис. 3.56 – Автоматическая настройка тактирования

Важно, чтобы USB тактировался частотой 48 МГц. Для этого необходимо, чтобы делители были выставлены как на рисунке 3.57.

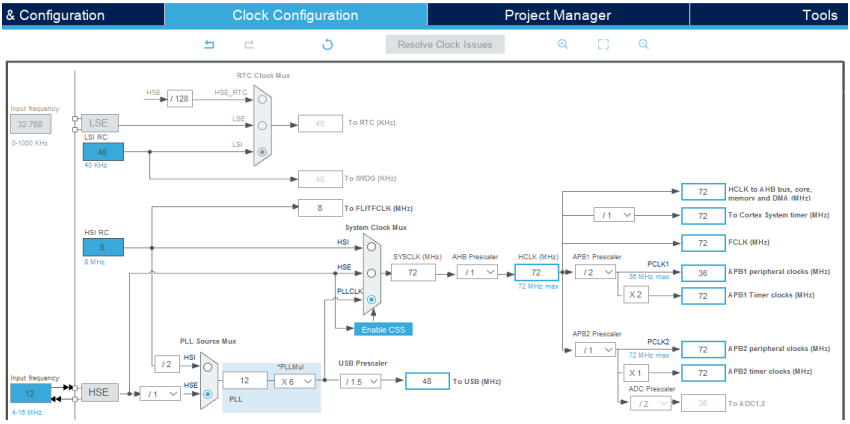



Рис. 3.57 – Настройка тактирования для обмена по USB

Проект настроен. Теперь необходимо его сгенерировать и написать программу. В примере средой разработки является IDE SW4STM32.

Далее представлен пример программы, позволяющий передавать из контроллера через USB значение, которое постоянно изменяется в контроллере (переменная count).

Если контроллер получает по USB сообщение, в котором первый символ «0», переменная count примет значение ноль, если в сообщении первый символ «1» – значение единицы.

```
/* USER CODE BEGIN 2 */
char buf[25];
uint32_t count=0;
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
	sprintf(buf,"count: [%ld]\n", count);
	CDC_Transmit_FS((uint8_t *)buf, strlen(buf));
	HAL_Delay(100);

	if (count < 1000)
		count++;
	else
		count = 0;
	if(receiveBufLen > 0)// если получены данные от ПК
	{
		HAL_Delay(250);
```

```
                switch(UserRxBufferFS[0]) // выбор
определённого действия
                {
                        case '0':
                          count = 0;
                        break;
                        case '1':
                          count = 1;
                        break;
                }

                receiveBufLen = 0; // сброс получения
        }
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
```

Созданный проект позволяет принимать и отправлять данные через USB. Отправка данных осуществляется функцией CDC_Transmit_FS(), в качестве параметров передаются буфер с данными и количество элементов буфера. Данная функция описана в файле «usbd_cdc_if.c», созданном STM32CubeMX автоматически при генерации кода проекта.

При приеме данных по USB происходит переход в функцию CDC_Receive_FS(), и в ней можно напрямую оперировать с буфером UserRxBufferFS с принятыми данными.

CDC_Receive_FS() – это, так называемая callback-функция, ее не надо вызывать вручную (она вызывается по прерыванию). Возможен прием сразу нескольких байт: один из аргументов

функции CDC_Receive_FS() – это количество принятых байт. В эту функцию можно добавлять свой код. Например, для управления флагом или семафором (при использовании RTOS), с целью определения события «данные приняты».

В приведенном коде в качестве флага получения данных используется переменная receiveBufLen, которую можно определить в файле «usbd_cdc_if.c».

```
/* USER CODE BEGIN PRIVATE_VARIABLES */
uint32_t receiveBufLen = 0;
/* USER CODE END PRIVATE_VARIABLES */
```

Значение данной переменной присваивается непосредственно в теле функции CDC_Receive_FS().

```
static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)
{
    /* USER CODE BEGIN 6 */
    USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
    USBD_CDC_ReceivePacket(&hUsbDeviceFS);

    receiveBufLen = *Len;

    return (USBD_OK);
    /* USER CODE END 6 */
}
```

Важно отметить, что при получении пакета данных необходимо подтвердить обработку пакета, для чего в тело функции CDC_Receive_FS() код-генератором CubeMX автоматически добавлена функция USBD_CDC_ReceivePacket(),

которая осуществляет подготовку выхода конечного устройства на прием. В более ранних версиях CubeMX указанная функция автоматически не добавлялась и это у неопытных пользователей приводило к сбою работы.

Приемный буфер UserRxBufferFS также определен в файле «usbd_cdc_if.c»:

```
/** Received data over USB are stored in this buffer      */
uint8_t UserRxBufferFS[APP_RX_DATA_SIZE];
```

Данный буфер чистить от мусора не нужно, он очистится автоматически. Чтобы иметь к нему доступ из «main.c», необходимо в заголовочном файле «usbd_cdc_if.h» указать следующее:

```
/* USER CODE BEGIN EXPORTED_DEFINES */
#define APP_RX_DATA_SIZE  1000
/* USER CODE END EXPORTED_DEFINES */

/* USER CODE BEGIN EXPORTED_VARIABLES */
extern uint8_t UserRxBufferFS[APP_RX_DATA_SIZE];
extern uint32_t receiveBufLen;
/* USER CODE END EXPORTED_VARIABLES */
```

При этом, необходимо убрать из файла «usbd_cdc_if.c» определение размера буфера принимаемых данных APP_RX_DATA_SIZE.

Сформированный код необходимо скомпилировать и записать его в микроконтроллер с помощью программатора (например, ST-LINK V2).

При подключении устройство будет распознано, например, как «STMicroelectronic Virtual COM Port (Com11)» (рисунок 3.58) и в этом можно убедиться через «Диспетчер устройств».

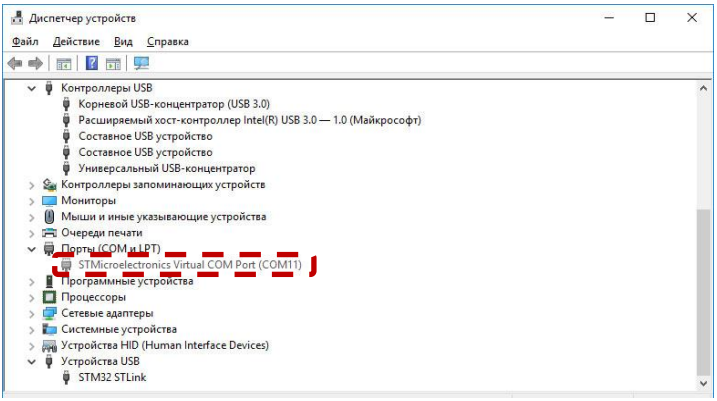



Рис. 3.58 – Определение подключенного устройства

Описание проектируемого устройства, которое отображается в диспетчере устройств, доступно для изменения в CubeMX при настройке проекта (рисунок 3.59): раздел **Middleware → USB_DEVICE → Configuration** вкладка «Device Descriptor» поле «PRODUCT_STRING (Product Identifier)».

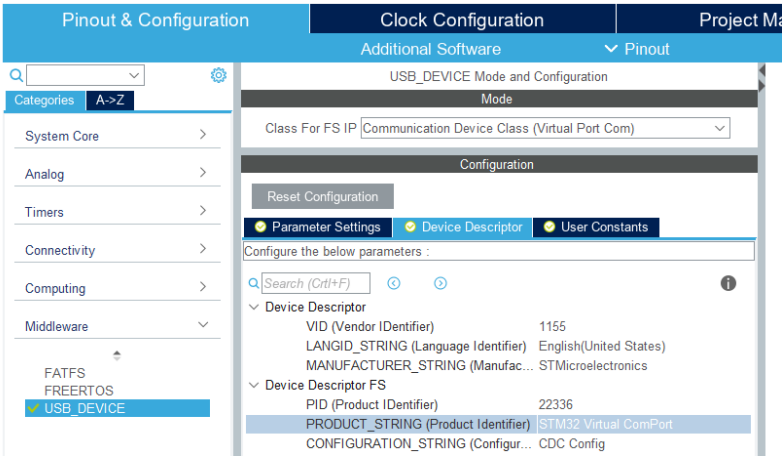



Рис. 3.59 – Описание подключенного устройства

Необходимо отметить, что при подключении МК через USB к компьютеру, **может возникнуть ситуация, когда порт**

**отображается в диспетчере, но не функционирует** (в свойствах устройства при этом обычно указано: **«не удается запустить устройство (код 10)»** и при попытке обращения к этому порту, выдается ошибка, что такого порта нет). Данная проблема обычно решается либо увеличением объема выделяемого стека и кучи (Stack_Size и Heap_Size), либо изменением максимального размера пакета данных, которые доступны в файле «usbd_def.h»:

```
#define USB_HS_MAX_PACKET_SIZE      512
#define USB_FS_MAX_PACKET_SIZE      64
```

Однако, если проект в CubeMX сгенерировать вновь, то измененный максимальный размер пакета данных будет выставлен по умолчанию, т.к. файл «usbd_def.h» не предполагает наличие пользовательских неизменяемых разделов. Следовательно, для исправления ошибки изменять параметры рекомендуется через код-генератор с последующей перегенерацией кода проекта.

Также в файл «main.c» в соответствующий раздел необходимо добавить **#include “usbd_cdc_if.h”**, чтобы при компиляции проекта не возникло предупреждения: **implicit declaration of function 'CDC_Transmit_FS' [-Wimplicit-function-declaration].**

**Ошибка 28** при подключении к компьютеру в большинстве случаев обусловлена некорректной установкой драйвера. В этом случае для загрузки драйвера необходимо обратиться на сайт производителя.

Вместе с тем, при настройке обмена данными нередки ситуации, когда внешний кварц не дает нужную частоту. В этом случае можно попробовать осуществить тактирование от внутреннего кварца и посмотреть, будут ли изменения. Проверку следует произвести как в задачах с USB, так и в проектах с таймерами.

Для наблюдения за передаваемыми данными через виртуальный Com-порт можно воспользоваться терминальной

программой (например, «CoolTermWin»), распространяемой бесплатно. После её установки и запуска появится окно, внешний вид которого представлен на рисунке 3.60.

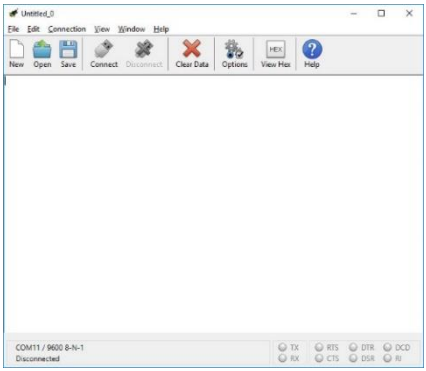



Рис. 3.60 – Внешний вид утилиты CoolTermWin

Перед обменом информацией необходимо настроить параметры «Options», в частности, в разделе «Terminal» определить режим «Line Mode» (рисунок 3.61), после чего станет доступна строка ввода передаваемого от компьютера сообщения.

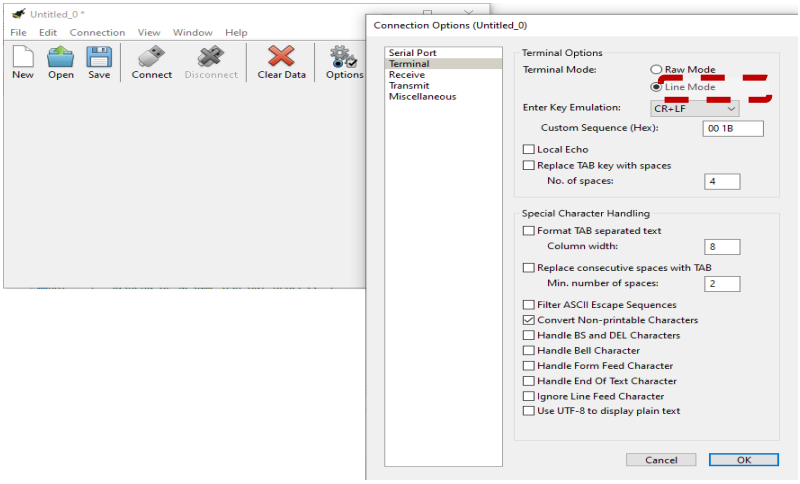



Рис. 3.61 – Настройка параметров терминала в CoolTermWin

Далее необходимо подключиться к порту, нажав кнопку «Connect». После чего можно наблюдать обмен данными. Для отправки сообщения от компьютера необходимо ввести его в соответствующей строке (рисунок 3.62) и нажать клавишу «Enter».

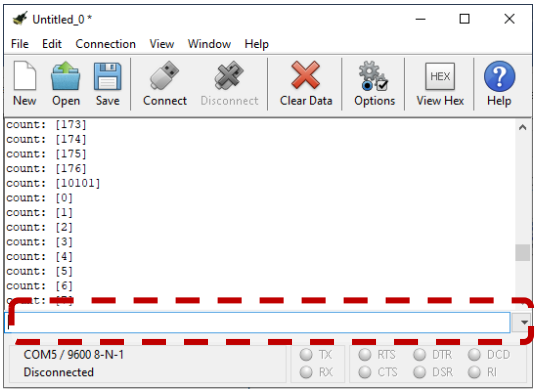



Рис. 3.62 – Строка ввода в CoolTermWin

Необходимо отметить, что настройка устройства на работу с USB иногда требует анализа на предмет наличия конфликта между используемой в проекте периферией. В приложении 4 приведен пример настройки USB и ШИМ.

### 3.2.3 Обработка сигналов энкодера

Конструкция, принцип работы, электрическая схема и особенности обработки сигналов энкодера приведены в разделах 1.6.2.1 и 3.1.4.

Для работы с энкодером, входящим в состав отладочного модуля, необходимо:

1. Выбрать напряжение питания энкодера.

   При напряжении 5 В выходные сигналы энкодера следует подавать на толерантные входы МК.

   Если напряжение < 3,6 В, выходные сигналы можно подавать на любой вход МК.

2. Определить алгоритм обработки сигналов.

Основными алгоритмами являются:

- Программный – анализ предыдущего и последующего состояния на входе порта укажет направление вращения. Например, если текущее состояние кода 11, а последующее 01, то вращение по часовой стрелке, при 10 – против часовой. Угол поворота можно определить, например, подсчитывая количество переходов через 00.
- По прерыванию – в момент среза импульса одного канала, который подается на вход внешнего прерывания, анализировать состояние второго.
- Аппаратный – с помощью счетчика-таймера, работающего в режиме измерения параметров внешнего энкодера (см. раздел 1.6.2.1).

3. Соединить выходы энкодера с МК.
4. Разработать программу, в которой следует предусмотреть средства отображения конечных и промежуточных результатов.

В качестве примера рассмотрим следующую задачу: определить текущий номер позиции энкодера и записать его в виртуальный терминал компьютера. При вращении ручки энкодера по часовой стрелке значение переменной увеличивается, против часовой – уменьшается. При переходе через нулевое состояние изменяется свечение светодиода LED3 отладочного модуля.

Для решения этой задачи выбрано напряжение питание энкодера 5 В.

Меньшую сложность программы обеспечит использование счетчика-таймера СТ в режиме захвата (измерение параметров внешнего энкодера) (раздел 1.6.2.1). В качестве СТ выбран TIM4 и толерантные входы PB6, PB7.

Связь с компьютером выполняется через разъем USB2. **Следует напомнить, что разъем USB1 должен быть отключен от компьютера**.

Схема коммутации оценочного модуля представлена на рисунке 3.63.

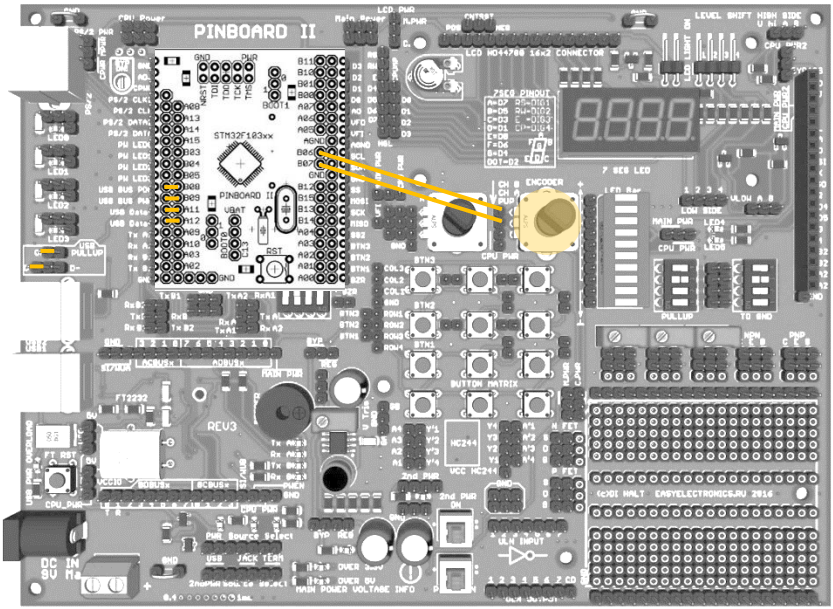



Рис. 3.63 – Схема соединений для решения задачи

Разработка программного обеспечения начинается с создания проекта в CubeMX: выбрать соответствующий микроконтроллер, настроить режим отладки (в поле «Debug» выбрать значение «Serial Wire») и тактирование процессора (в поле «High Speed Clock» выбрать «Crystal/Ceramic Resonator»).

Далее выполняется настройка TIM4 на соответствующий режим (рисунок 3.64).

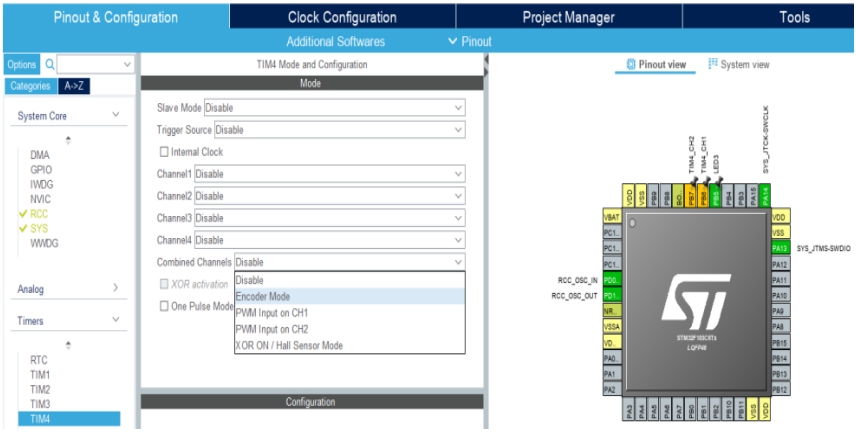



Рис. 3.64 – Настройка TIM4 на режим энкодера

В разделе «Configuration» таймера TIM4 (рисунок 3.65) производится настройка «Encoder Mode» [24].

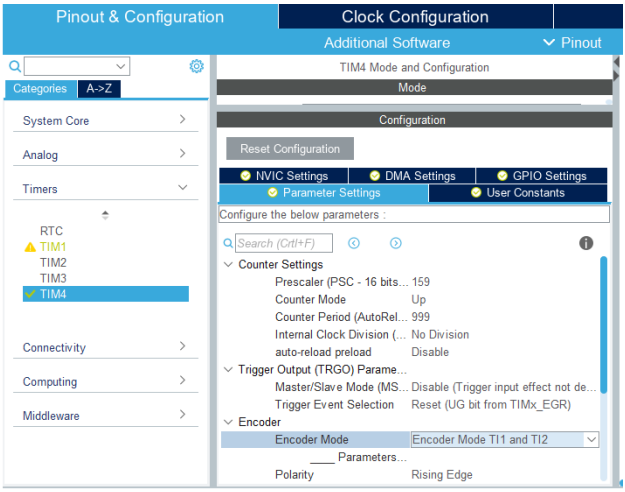



Рис. 3.65 – Выбор режима энкодера

Настройка обмена данными по USB описана в разделе 3.2.2.

Проект настроен. Теперь необходимо сгенерировать код в выбранной среде разработки и добавить необходимый код.

```
/* USER CODE BEGIN 2 */

  HAL_TIM_Encoder_Start(&htim4, TIM_CHANNEL_ALL);
// буфер для вывода строки в USB
  char buf[25];
// capture - текущее значение счетчика таймера TIM4
// capture_prev - предыдущее значение счетчика таймера TIM4
// encoder – разница между предыдущим и текущим значением
счетчика таймера TIM4
  int32_t capture=0, capture_prev=0, encoder=0;

  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
// зафиксировать текущее значение счетчика
        capture = TIM4->CNT;
// определить показания энкодера
        encoder += capture - capture_prev;
// условие смены состояния светодиодом
        if (abs(capture-capture_prev) > 32767) {
              encoder += (capture < capture_prev ? 65535 : -
65535);
// инвертировать состояние светодиода
              HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_5);
        }
// зафиксировать предыдущее захваченное значение
        capture_prev = capture;
// подготовить строку-сообщение для передачи через USB
        sprintf(buf,"count: [%ld]\n", encoder);
```

```
// передать строку в компьютер
        CDC_Transmit_FS((uint8_t *)buf, strlen(buf));
// задержка между измерениями
        HAL_Delay(100);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

Код написан, теперь необходимо его скомпилировать и записать в контроллер.

После подключения оценочного модуля к компьютеру устройство будет определено, например, как «STMicroelectronic Virtual COM Port (Com11)» (рисунок 3.66).

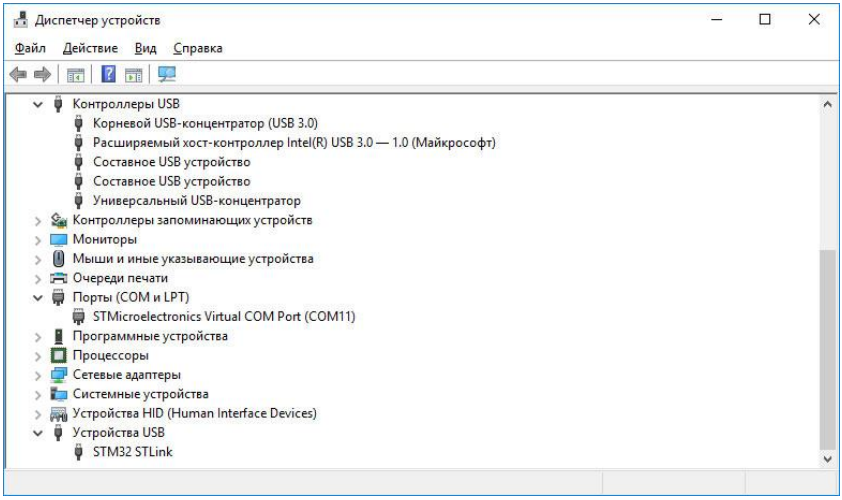



Рис. 3.66 – Диспетчер устройств

Для наблюдения за показаниями энкодера через виртуальный Com-порт необходимо запустить терминал «CoolTermWin» [90]).

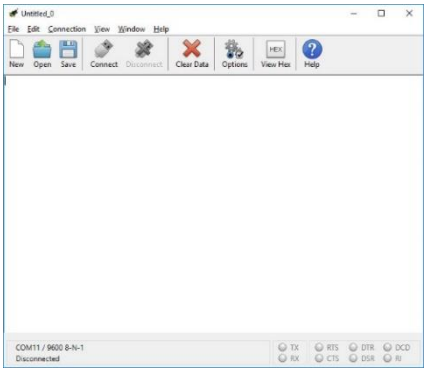



Рис. 3.67 – Интерфейс программы «CoolTermWin»

Настроить параметры подключения:

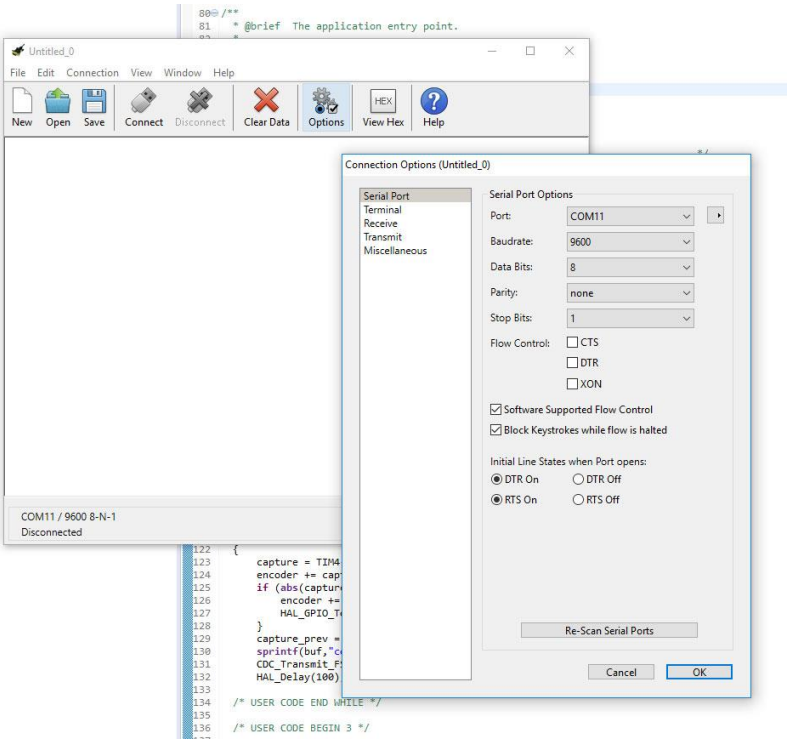



Рис. 3.68 – Параметры подключения «CoolTermWin»

Подключиться к порту.

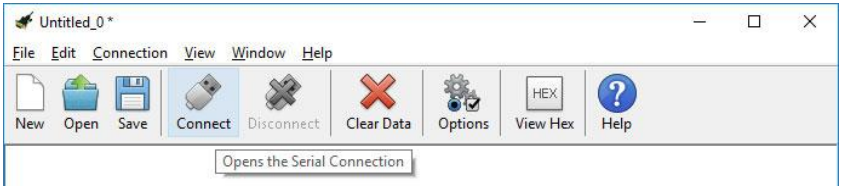



Рис. 3.69 – Подключение к порту

После подключения к последовательному порту можно наблюдать изменение переменной, вызванное вращением рукоятки энкодера (рисунок 3.70), при переходе через «0» на отладочном модуле изменяется свечение LED3.

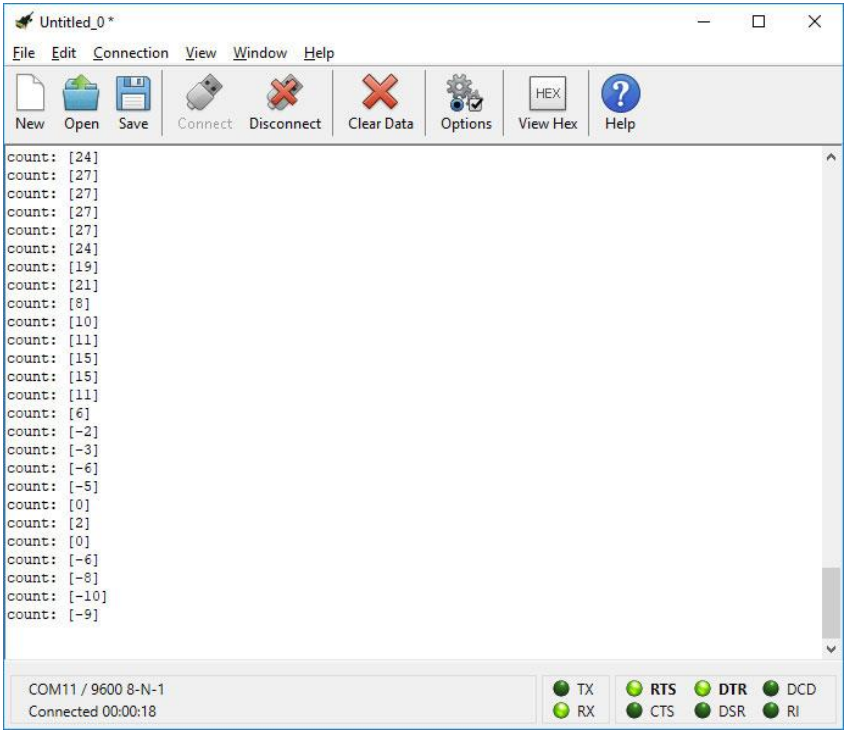



Рис. 3.70 – Результат работы

## 3.3 Контрольные вопросы

### По разделу 3. Отладочный модуль PinBoard II R3

1. Для каких целей используется отладочный модуль ОМ?
2. Какие основные характеристики ОМ?
3. Принцип работы ОМ.
4. Как формируется структура ОМ?
5. Какие режимы работы можно устанавливать на плате микроконтроллера?
6. В каких случаях следует использовать разъем **USB1?**
7. Каким образом информация с **USB1** поступает на МК?
8. Назначение и функции моста FT2232.
9. Как передать информацию с USB на USART1? USART2? Выполнить эхо контроль?
10. Для каких целей используется разъем USB2? Какую спецификацию поддерживает порт USB микроконтроллера?
11. Какие установки следует выполнить в ОМ для обмена через USB2?
12. Если необходимо реализовать обмен **данными** между компьютером и ОМ, какой разъем выбрать? Почему?
13. Почему нельзя одновременно подключать к компьютеру USB1 и USB2?
14. Какие функции реализует внутрисхемный отладчик?
15. В чем преимущества SWD по сравнению JTAG?
16. Как изменяются условия запуска микроконтроллера в зависимости от установки BOOT1, BOOT0? В какое положение их надо установить при программировании флеш-памяти?
17. Как присоединить программатор **ST-LINK V2** к плате микроконтроллера?
18. В чем особенность подключения микросхемы перепрограммируемой памяти 24C64WP?
19. Как присоединить гнездо SD-карты к микроконтроллеру?

20. Как присоединить источник аналогового напряжения к входу АЦП микроконтроллера помощью фильтра? Без фильтра?

21. Какое напряжение питания следует подавать на потенциометр, если установлен микроконтроллер STM32F103xx? Почему?

22. Для каких целей используют энкодеры?

23. В чем отличие инкрементного энкодера от абсолютного?

24. Какими характеристиками обладает энкодер EC12E24404A6?

25. Какие методы используются для определения параметров энкодера? Какой бы вы выбрали? Почему?

26.Какие методы можно применять для устранения дребезга контактов?

27. Как согласовать выходной сигнал энкодера с микроконтроллером?

28. Как сформировать с помощью блока переключателей уровень 1? Уровень 0? Как сформированный уровень подключить к микроконтроллеру?

29. Как соединить кнопки матрицы для организации клавиатуры 3x4?

30. Какие соединения с микроконтроллером необходимо выполнить для того, чтобы проверить состояние копки SWi?

31. Для каких целей можно использовать ЦАП? В чем его недостатки?

32. Какие средства предусмотрены для согласования уровней напряжения?

33. В каких случаях необходим магистральный усилитель? Шинный мультиплексор? Какую максимальную нагрузку можно подключать?

34. Какие соединения надо выполнить и как подключить к микроконтроллеру 8-разрядный магистральный усилитель? Шинный мультиплексор 2x4?

35. Какие средства предусмотрены для согласования с исполнительными устройствами повышенной мощности? Их технические характеристики.

36. Как обеспечить работу транзисторной сборки ULN2003 при повышенных уровнях напряжения?

37.Для каких целей используется широтно-импульсный модулятор ШИМ?

38. Как на базе ШИМ построить преобразователь постоянного напряжения? Какие соединения необходимо выполнить для подключения внешнего фильтра ШИМ?

39. Какие соединения необходимо выполнить для управления яркостью светодиода?

40. Для каких целей можно использовать светодиоды? BAR индикаторы?

41. В каких случаях целесообразно применение полупроводниковых 7-сегментных индикаторов? Алфавитно-цифровых индикаторов с контроллером HD44780? В чем отличия в управлении отображением информации?

42. Объяснить назначение элементов на электрической схеме блока индикации.

43. Как подключить к микроконтроллеру зуммер? Как сформировать требуемую частоту?

44. В каких случаях следует использовать расширитель внутренней шины?

45. Какие источники напряжения могут использоваться для питания отладочного модуля? Как выполнить их коммутацию?

46. Для каких целей предназначена шина **MainPower?** Какие варианты значений напряжений на ней возможны? Чем они отличаются?

47. Для каких целей предназначена шина **CPU PWR?** Какие варианты значений напряжений на ней возможны? Чем они отличаются?

48. Как подключить питание к SD-карте? энкодеру? потенциометру? 7-сегментному индикатору? Алфавитно-цифровому индикатору?

49. В каких случаях требуется шина вторичного питания? К каким устройствам её можно подключать?

### По разделу 3.2 Особенности работы с внешними устройствами и 3.2.1 Модуль индикации на основе контроллера HD-44780

1. Для каких целей предназначен контроллер жидкокристаллических индикаторов КЖКИ HD44780?
2. Какие функции выполняет процессорный блок?
3. Для каких целей предназначена видеопамять? ОЗУ знакогенератора? ПЗУ знакогенератора?
4. Какие параметры ПЗУ ЗГ следует контролировать при выборе типа КЖКИ HD44780?
5. Какова структура видеопамяти? Как выполнить адресацию к строкам в 2-строчном дисплее? 4-строчном?
6. С помощью каких сигналов выполняется управление КЖКИ?
7. Какой способ связи с микроконтроллером является более рациональным? Почему?
8. В чем особенности формирования временной диаграммы при работе с 4-разрядной синхронной шиной.
9. Какие требования предъявляются к параметрам сигналов, формируемых микроконтроллером?
10. Какие особенности при выборе линии микроконтроллера для управления модулем индикации?
11. Как выполнить инициализацию модуля индикации, работающего в синхронном режиме с 4-разрядной шиной?
12. Какие команды необходимо выполнить, чтобы произвольное число вывести в произвольную позицию дисплея модуля?

13. Какие команды необходимо выполнить чтобы включить/выключить курсор? Перечислите регистры, которые необходимо задействовать.

14. Какие команды необходимо выполнить, чтобы вернуть курсор в нулевое положение без очистки выведенной ранее информации?

15. Какие команды необходимо выполнить, чтобы вывести сообщения справа-налево на дисплей модуля?

16. Какие команды необходимо выполнить, чтобы включить/выключить дисплей модуля?

17. Какова последовательность создания и структура проекта, а также схема подключения модулей PINBOARD II rev.3 при реализации двухстороннего обмена информацией с компьютером?

**По разделу 3.2.2 Подключение виртуального терминала**

1. Какие задачи позволяет решать виртуальный терминал?
2. Как подготовить отладочный модуль для работы с виртуальным терминалом?
3. Какие параметры необходимо установит в CUBE MX для настройки порта USB микроконтроллера?
4. Как установить виртуальный терминал CoolTermWin?
5. Какие настройки CoolTermWin необходимо выплнить для работы с микроконтроллером?
6. Как организовать обмен данными между виртуальным терминалом и микроконтроллером?

**По разделу 3.2.3 Обработка сигналов энкодера**

1. Для решения каких задач следует использовать энкодер?
2. Какие особенности необходимо учитывать при выборе питания энкодера?

3. Какие факторы необходимо учитывать при выборе алгоритма обработки?

4. В чем преимущество счетчика-таймера при измерении параметров энкодера?

5. Какие соединения необходимо выполнить в отладочном модуле для взаимодействия энкодера с виртуальным терминалом компьютера?

6. Как настроить счетчик-таймер для работы с энкодером?

7. Как настроить виртуальный терминал для работы с энкодером?

8. Как организовать передачу данных о состоянии энкодера в компьютер?

# 4 Утилита программирования микроконтроллера

Рассмотренные в разделе 2 среды разработки позволяют программировать микроконтроллеры. Однако, функционал сред разработки для программирования МК в большинстве случаев крайне ограничен, и опытные разработчики используют специализированные приложения. Данным приложениям достаточно бинарного или hex-файла с программой для загрузки в память контроллера. Одним из таких приложений является утилита STM32 ST-Link Utility.

## 4.1 Утилита STM32 ST-Link Utility

Утилита STM32 ST-Link Utility распространяется бесплатно, доступна для загрузки с сайта ST Microelectronics [26] и обладает следующими возможностями:

- загрузка, редактирование и сохранение исполняемых файлов и файлов данных, созданных Assembler/Linker или C компиляторами; поддержка HEX и двоичных форматов;
- удаление, программирование, просмотр и проверка содержимого флеш-памяти устройства;
- автоматизация программирования STM32 (удаление, проверка, программирование, настройка байтов опций, вычисление контрольной суммы);
- прошивка однократно-программируемой памяти;
- поддержка программирования и настройки опционных байтов.

Для программирования микроконтроллера с помощью ST-Link Utility необходимо выполнить ряд действий:

1. Установить связь с микроконтроллером, выбрав в меню **Target** пункт **Connect** (рисунок 4.1). После этого утилита подключается к целевому устройству и отображает тип устройства,

идентификатор устройства и объем флеш-памяти в информационной зоне.

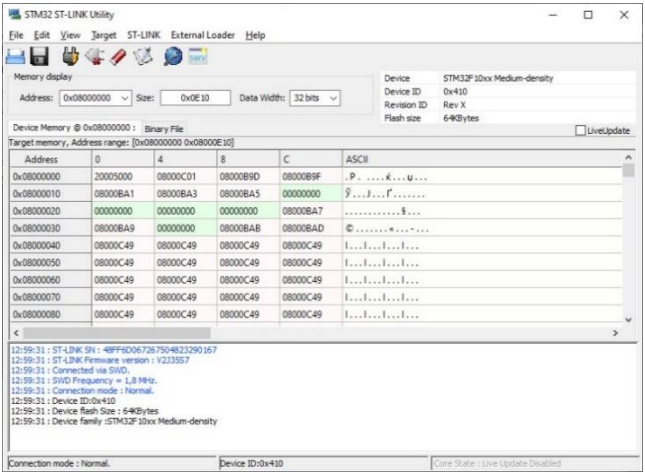



Рис. 4.1 – Интерфейс утилиты ST-Link Utility

2. Осуществить настройку соединения, выбрав в меню **Target** пункт **Settings…** (рисунок 4.2).

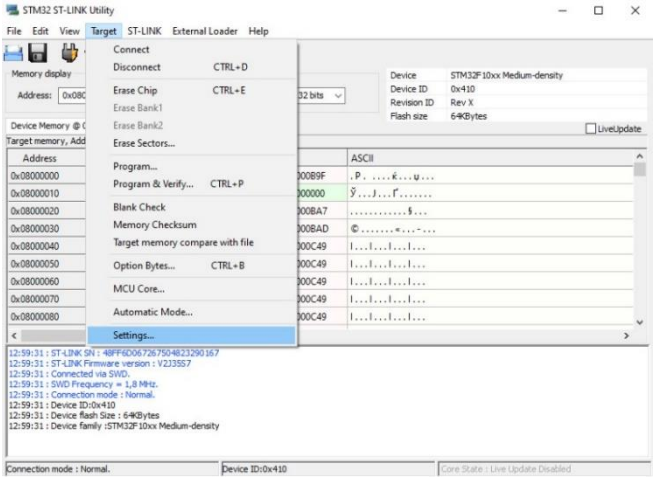



Рис. 4.2 – Пункт меню для доступа к настройкам

В появившемся диалоговом окне (рисунок 4.3) можно, например, выбрать программный сброс «Software System Reset».

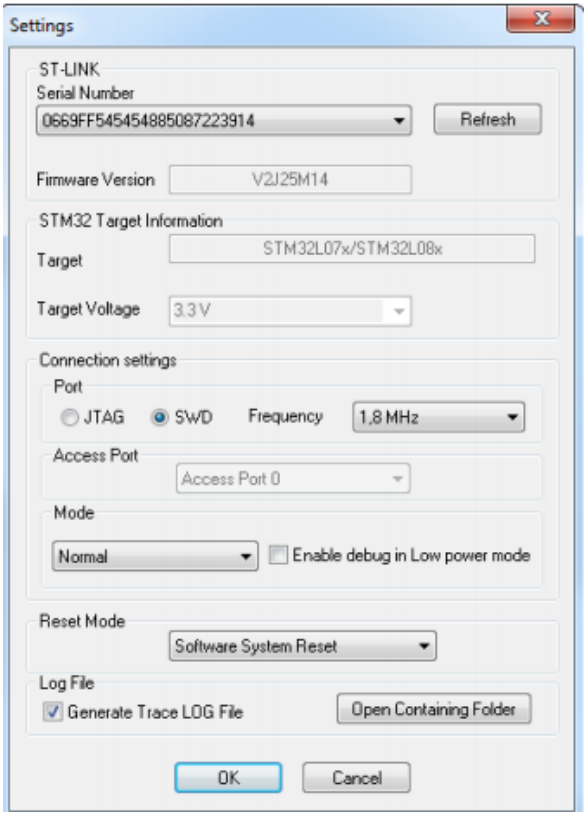



Рис. 4.3 – Окно настройки соединения

3. В меню **Target** выбрать пункт **Program...** (или **Program & Verify...**, если необходимо проверить записанные данные) и в открывшемся диалоговом окне выбрать необходимый для записи в контроллер файл.

4. Указать адрес, с которого нужно начать программирование (рисунок 4.4, это может быть флеш-память или адрес ОЗУ) и нажать кнопку Start для программирования.

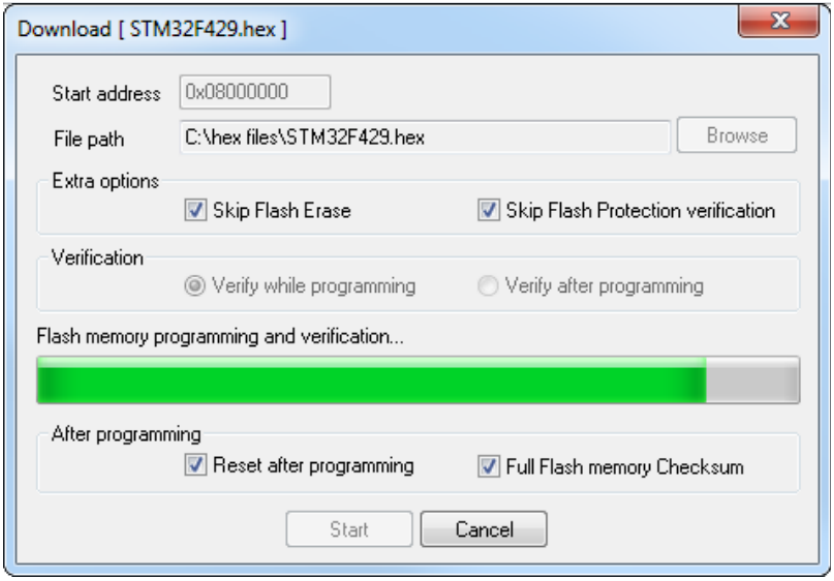



Рис. 4.4 – Окно программирования устройства

Необходимо отметить, что в окне программирования (рисунок 4.4) доступны следующие опции:

**Skip Flash Erase** – пропустить операцию удаления флеш-памяти (если устройство уже очищено);

**Skip Flash Protection verification** – пропустить проверку защиты флеш-памяти (если устройство не защищено);

**Verify while programming** – быстрый метод проверки на чипе, при котором сравнивается содержимое буфера программы (часть файла) с содержимым флеш-памяти;

**Verify after programming** – медленный, но надежный метод проверки, при котором вся запрограммированная зона памяти после завершения работы программы сравнивается с содержимым файла;

**Reset after programming** – после программирования выполняется сброс MCU;

**Full Flash memory Checksum** – после операции программирования рассчитывается контрольная сумма полной флеш-памяти.

При использовании ST-LINK напряжение питания MCU должно быть указано в настройках (рисунок 4.3), чтобы можно было запрограммировать устройство в правильном режиме. При использовании ST-LINK/V2 или ST-LINK-V3 напряжение питания определяется автоматически.

Если некоторые страницы флеш-памяти защищены от записи, защита отключается во время программирования, а затем восстанавливается. Если устройство защищено от чтения, защита отключается.

## 4.2 Очистка памяти

Вместе с тем, при отладке устройств начинающими разработчиками часто возникают ситуации, когда контроллер не определяется программатором, что лишает возможности запрограммировать микроконтроллер. SW4STM32 при этом выдает соответствующее сообщение (рисунок 4.5).

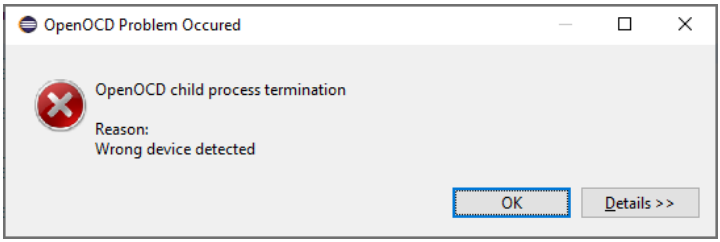



Рис. 4.5 – Сообщение об ошибке подключения к устройству

Это может быть связано с тем, что в контроллер записан код, не предполагающий подключение интерфейса SWD, через который производится отладка. То есть в контроллер загружена прошивка, не предполагающая его перепрограммирование.

Данную проблему можно устранить, например, очистив память микроконтроллера. Для этого в STM32 ST-Link Utility предусмотрено два типа очистки флеш-памяти:

• **полная очистка** (удаление всех секторов флеш-памяти подключенных устройств). Данное действие осуществляется по нажатию в меню **Target** пункта **Erase Chip**;

• **очистка сектора флеш-памяти** (удаление выбранных секторов флеш-памяти). Выбрав в меню **Target** пункт **Erase Sectors...**, откроется диалоговое окно, отражающее флеш-память, в котором пользователь может выбрать сектор(-ы) для удаления, как показано на рисунке 4.6.

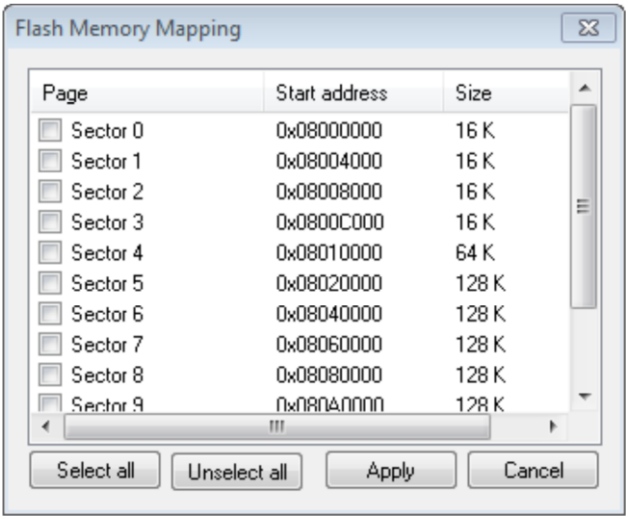



Рис. 4.6 – Окно страниц флеш-памяти

Кнопка «Select all» выбирает все страницы флеш-памяти.

Кнопка «Deselect all» отменяет выбор всех выбранных страниц.

Кнопка «Cancel» отменяет операцию удаления, даже если выбраны некоторые страницы.

Кнопка «Apply» стирает все выбранные страницы.

## 4.3 Контрольные вопросы

1. С помощью каких программных продуктов можно загрузить программу в память микроконтроллера?
2. С помощью каких программных продуктов можно осуществить предварительную настройку необходимой периферии?
3. Как подготовленный ранее *.hex файл можно прошить во флеш-память микроконтроллера?
4. Можно ли очистить прошивку микроконтроллера, если флеш-память устройства защищена?
5. Какие действия необходимо выполнить для того, чтобы код программы, зашитый в микроконтроллер, было невозможно копировать на другие устройства?
6. С помощью каких программных продуктов можно производить отладку программы с возможностью просмотра содержимого памяти и регистров микроконтроллера?

## Список использованных источников


1. 24C64WP datasheet: [сайт]. 2020. – URL: https://www.alldatasheet.com/datasheet-pdf/pdf/45746/SIEMENS/24C 64.html (дата обращения: 20.09.2020).
2. ADC HAL stm32: [сайт]. 2020. – URL: https://istarik.ru/blog/stm32/113.html#startadctim (дата обращения: 20.09.2020).
3. AN2821 Application note: [сайт]. 2020. – URL: https://www.st.com/content/ccc/resource/technical/document/application_note/b0/34/9f/35/17/88/43/41/CD00207941.pdf/files/CD00207941.pdf/jcr:content/translations/en.CD00207941.pdf (дата обращения: 20.09.2020).
4. ARM: How to archive your MDK project: [сайт]. 2020. – URL: http://www.keil.com/3739.htm (дата обращения: 20.09.2020).
5. CubeMX и Workbench: создание проекта на базе STM32 с помощью бесплатного ПО: [сайт]. 2020. – URL: https://www.compel.ru/lib/75317 (дата обращения: 20.09.2020).
6. Customisation: [сайт]. 2020. – URL: https://www.freertos.org/a00110.html (дата обращения: 20.09.2020).
7. FT2232D-Breakout: плата для гальванической развязки USB – URL: http://microsin.net/adminstuff/hardware/ft2232d-with-galvanic-usb-isolation.html (дата обращения: 20.09.2020).
8. GENERAL: IS THE LINUX OPERATING SYSTEM SUPPORTED?: [сайт]. 2020. – URL: http://www.keil.com/support/docs/1456.htm (дата обращения: 20.09.2020).
9. Integrated Development Environment for STM32: [сайт]. 2020. – URL: https://www.st.com/content/st_com/en/products/ development-tools/software-development-tools/stm32-software-development-tools/stm32-ides/stm32cubeide.html#overview (дата обращения: 20.09.2020).
10. iTAG: инструментарий отладки и тестирования: [сайт]. 2020. – URL: http://microsin.net/programming/arm/itag-debug-and-testing.html (дата обращения: 20.09.2020).
11. Kernel > API Reference > QueuesQueue Management [API]: [сайт]. 2020. – URL: https://freertos.org/a00018.html (дата обращения: 20.09.2020).
12. Kernel > Developer Docs > Heap Memory Management Memory Management: [сайт]. 2020. – URL: https://freertos.org/a00111.html#heap_1 (дата обращения: 20.09.2020).

13. LIN интерфейс: [сайт]. 2020. – URL: http://www.gaw.ru/html.cgi/txt/interface/lin/index.htm (дата обращения: 20.09.2020).

14. Mainstream Performance line, ARM Cortex-M3 MCU with 64 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN: [сайт]. 2020. – URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html#overview (дата обращения: 20.09.2020).

15. PR0056. Programming manual. STM32F10xxx/20xxx/21xxx/L1xxx Cortex-M3: [сайт]. 2020. – URL: https://www.st.com/content/ccc/resource/technical/document/programming_manual/5b/ca/8d/83/56/7f/40/08/CD00228163.pdf/files/CD00228163.pdf/jcr:content/translations/en.CD00228163.pdf (дата обращения: 20.09.2020).

16. Proteus (система автоматизированного проектированя): [сайт]. 2020. – URL: Proteus_(система_автомати зированного_проектирования) (дата обращения: 20.09.2020).

17. RM0008 Reference manual: [сайт]. 2020. – URL: https://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf (дата обращения: 20.09.2020).

18. RTC HAL stm32: [сайт]. 2020. – URL: https://istarik.ru/blog/stm32/ 116.html (дата обращения: 20.09.2020).

19. Scheduling [RTOS Fundamentals]: [сайт]. 2020. – URL: https://freertos.org/a00005.html (дата обращения: 20.09.2020).

20. STM Урок 138. Independent watchdog (IWDG). Часть 1.: [сайт]. 2020. – URL: http://narodstream.ru/stm-urok-138-independent-watchdog-iwdg-chast-1/ (дата обращения: 20.09.2020).

21. STM Урок 147. LL. Таймеры. Часть 1: [сайт]. 2020. – URL: https://nardstream.ru/tamery-cst1/ (дата обращения: 20.09.2020).

22. STM32 ADC Примеры использования. Шаг 1 – URL:http://mycontroller.ru/old_site/stm32-adc-primeryi-ispolzovaniya-shag-1/default.htm (дата обращения: 20.09.2020).

23. STM32 ADC. Описание работы модуля: [сайт]. 2020. – URL: http://mycontroller.ru/old_site/stm32-adc-opisanie-rabotyi-modulya/default.htm (дата обращения: 20.09.2020).

24. STM32 blink++ или читаем данные инкрементального энкодера: [сайт]. 2020. – URL: http://othermedia.info/?p=1692 (дата обращения: 20.09.2020).

25. STM32 IDEs: [сайт]. 2020. – URL: https://www.st.com/en//stm32-ides.html#products (дата обращения: 20.09.2020).

26. STM32 ST-LINK utility: [сайт]. 2020. – URL: https://www.st.com/en/ development-tools/stsw-link004.html (дата обращения: 20.09.2020).

27. STM32 TIMER general purpose. Режим сравнения.: [сайт]. 2020. – URL: http://mycontroller.ru/old_site/stm32-timer-general-purpose-rezhim/default.htm (дата обращения: 20.09.2020).

28. STM32 USART. Использование DMA: [сайт]. 2020. – URL: http://mycontroller.ru/default.htm (дата обращения: 20.09.2020).

29. STM32. TIMER general purpose. Внешняя синхронизця.: [сайт]. 2020. – URL: http://mycontroller.ru/old_site/general-purpose-vneshnyaya-sinhronizatsiya/default.htm (дата обращения: 20.09.2020).

30. STM32. Подключаем смарт-карты стандарта ISO7816: [сайт]. 2020. – URL: https://habr.com/ru/post/257279/ (дата обращения: 20.09.2020).

31. STM32. Программирование STM32F103. RTC.: [сайт]. 2020. – URL: http://www.avislab.com/blog/stm32-rtc/ (дата обращения: 20.09.2020).

32. STM32. Программирование STM32F103. Тактирование: [сайт]. 2020. – URL:https://blog.avislab.com/stm32/ (дата обращения: 20.09.2020).

33. STM32: аналого-цифровой преобразователь: [сайт]. 2020. – URL: http://blog.myelectronics.com.ua/stm32-%D0%B0%D0%BD%D0%B0%D0%BB%D0%BE%D0%B3%D0% BE-%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0% BE%D0%B9-%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C/ (дата обращения: 20.09.2020).

34. STM32: сторожевые таймеры (WDT): [сайт]. 2020. – URL: http://blog.myelectronics.com.ua/stm32-%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%B6%D0%B5%D0%B2%D1%8B%D0%B5-%D1%82%D0%B0%D0%B9%D0%BC%D0%B5%D1%80%D1%8B-wdt/ (дата обращения: 20.09.2020).

35. STM32Cube initialization code generator: [сайт]. 2020. – URL: https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stm32cubemx.html (дата обращения: 20.09.2020).

36. STM32L-Интерфейс I2C. – URL: http://ziblog.ru/2011/12/14/stm32l-interfeys-i2c.html (дата обращения: 20.09.2020).

37. System Workbench for STM32: free IDE on Windows, Linux and OS X: [сайт]. 2020. – URL: https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development/sw4stm32.html (дата обращения: 20.09.2020).

38. Task Utilities [API]: [сайт]. 2020. – URL: https://freertos.org/a00021.html#vTaskList (дата обращения: 20.09.2020).

39. Tasks [More about tasks…]: [сайт]. 2020. – URL: https://freertos.org/ RTOS-task-states.html (дата обращения: 20.09.2020).

40. The FreeRTOS™Reference Manual: [сайт]. 2020. – URL: https://freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf (дата обращения: 20.09.2020).

41. UART и USART. COM-порт. Часть 1.: [сайт]. 2020. – URL: http://www.rotr.info/mcu/ (дата обращения: 20.09.2020).

42. UM0892User manualSTM32 ST-LINK utility software description: [сайт]. 2020. – URL: https://www.st.com/resource/en/user_manual/ cd00262073-stm32microelectronics.pdf (дата обращения: 20.09.2020).

43. UM1718 User manual. STM32CubeMX for STM32 configurationand initialization C code generation: [сайт]. 2020. – URL: https://www.st.com/content/ccc/resource/technical/document/user_manual/10/c5/1a/43/3a/70/43/7d/DM00104712.pdf/files/DM00104712.pdf/jcr:content/en.DM00104712.pdf (дата обращения: 20.09.2020).

44. USART stm32 HAL STM32: [сайт]. 2020. – URL: https://istarik.ru/blog/stm32/120.html (дата обращения: 20.09.2020).

45. USB in a NutShell – путеводитель по стандарту USB (начало): [сайт]. 2020. – URL: http://microsin.ru/44 (дата обращения: 20.09.2020).

46. WinIDEA OpenFree IDE: [сайт]. 2020. – URL: https://www.isystem.com/winidea.html (дата обращения: 20.09.2020).

47. Адаптеры JTAG с поддержкой SWD.: [сайт]. 2020. – URL: http://microsin.net/jtag-adapters.html (дата обращения: 20.09.2020).

48. Бесплатные инструменты разработчика для микроконтроллеров STM32, рекомендуемые STMicroelectronics: [сайт]. 2020. – URL: http://robotosha.ru/stm32/free-ide-stm32.html (дата обращения: 20.09.2020).

49. Звуковой зуммер серии HCM: [сайт]. 2020. – URL https://asenergi.com/catalog/akustika/zummer-hcm.html#more (дата обращения: 20.09.2020).

50. Иоффе, В. Г. Структурная организация однокристальных микроконтроллеров: [учеб. пособие] / В. Г. Иоффе; М-во образования и науки Рос. Федерации, Самар. нац. исслед. ун-т им. С. П. Королева (Самар. ун-т). – Самара: Изд-во Самар. ун-та, 2017. URL: http://repo.ssau.ru/handle/Metodicheskie-materialy/Strukturnaya-organizaciya-odnokristalnyh-mikrokontrollerov-Elektronnyi-resurs-ucheb-posobie-70944

51. Контроллер USB STM32 серий STM32F103xx и STM32F102xx: [сайт]. 2020. – URL: http://microsin.net/programming/arm-working-with-usb/stm32f103xx-stm32f102xx-full-speed-usb.html (дата обращения: 20.09.2020).

52. Мартин М. Инсайдерское руководство по STM32: [сайт]. 2020. – URL: https://istarik.ru (дата обращения: 20.09.2020).

53. Мартин Т. Инсайдерское руководство по STM32: [сайт]. 2020. – URL: https://docplayer.ru/25793154-Martin-m-insayderskoe-rukovodstvo.html (дата обращения: 20.09.2020).

54. Микроконтроллеры STM32F7. Часть 3. Периферия и таймеры: [сайт]. 2020. – URL: https://www.compel.ru/lib/125145 (дата обращения: 20.09.2020).

55. Обмен данными со смарт-картой ISO 7816-3: [сайт]. 2020. – URL: http://www.hackersrussia.ru/Cards/ASyncro/ISO7816-3.php (дата обращения: 20.09.2020).

56. Описание отладочного модуля PinBoard II R3 AVR + STM32: [сайт]. 2020. – URL: http://wiki.easyelectronics.ru/index.php/Pinboard_II_REV3 (дата обращения: 20.09.2020).

57. Подключаем sd карту памяти к stm32 по spi: [сайт]. 2020. – URL: https://alex-exe.ru/radio/stm32/connect-sd-card-stm32-spi/ (дата обращения: 20.09.2020).

58. Подключение инкрементального энкодера к микроконтроллеру: [сайт]. 2020. – URL: https://hubstub.ru/programming/63-podklyucheniye.html (дата обращения: 20.09.2020).

59. Подключение карт SD через SPI (упрощенное описание стандарта): [сайт]. 2020. – URL: http://microsin.net/programming/file-systems/sd-specifications-part-1-physical-layer-simplified-specification-ver-200-spi-mode.html (дата обращения: 20.09.2020).

60. Подтягивающий резистор: [сайт]. 2020. – URL: https://ru.wikipedia.org/wiki/Подтягивающий_резистор (дата обращения: 20.09.2020).

61. Прерывания в STM32. Регистры: [сайт]. 2020. – URL: https://mcucpu.ru/index.php/platformy-32-bit/stm32/150-preryvaniya-v-stm32 (дата обращения: 20.09.2020).

62. Программирование ARM-контроллеров STM32 на ядре Cortex-M3. Часть 12. Работа с модулями USART и UART: [сайт]. 2020. – URL: https://radiohlam.ru/stm32_12/ (дата обращения: 20.09.2020).

63. Программирование ARM-контроллеров STM32 на ядре Cortex-M3. Часть 13. Работа с модулями ADC: [сайт]. 2020. – URL: https://radiohlam.ru/stm32_13/#2.6 (дата обращения: 20.09.2020).

64. Проектирование микропроцессорных устройств на базе однокристальных микроконтроллеров: [метод. указания] / М-во образования и науки Рос. Федерации, Самар. гос. аэрокосм. ун-т им. С. П. Королева; сост. В. Г. Иоффе. – Самара, 2015. URL: http://91.222.128.30/handle/Metodicheskie-materialy/Proektirovanie-mikroprocessornyh-ustroistv-na-baze-odnokristalnyh-Elektronnyi-resurs-metod-ukazaniya-70959 (дата обращения: 20.09.2020).

65. Работа с микросхемой FTDI FT2232 в режиме BitBang: [сайт]. 2020. – URL: http://easyelectronics.ru/rabota-s-mikrosxemoj-ftdi-ft2232-v-rezhime-bitbang.html (дата обращения: 20.09.2020).

66. Разработка и отладка микропроцессорных устройств в виртуальной среде моделирования Proteus: [метод. указания] / М-во образования и науки Рос. Федерации, Самар. нац. исслед. ун-т им. С. П. Королева (Самар. ун-т); сост. В. Г. Иоффе. – Самара: Изд-во Самар. ун-та, 2017. URL: http://repo.ssau.ru/handle/Metodicheskie-materialy/Razrabotka-i-otladka-mikroprocessornyh-ustroistv-v-virtualnoi-srede-modelirovaniya-Proteus-Elektronnyi-resurs-metod-ukazaniya-70958.

67. Ревич Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. – 2-е изд., испр. – СПб.: БХВ-Петербург, 2011. – 352 с: ил.

68.: [сайт]. 2020. – URL: (дата обращения: 20.09.2020). Руководство по использованию обычных таймеров STM32: [сайт]. 2020. – URL:http://microsin.net/programming/arm/an4776-general-purpose.html (дата обращения: 20.09.2020).

70. Система тактирования: [сайт]. 2020. – URL: http://dimoon.ru/obuchalka/stm32f1/uroki-stm32f103-chast-4-nastroyka-rcc.html (дата обращения: 20.09.2020).

71. Сопрягаем энкодер и микроконтроллер: [сайт]. 2020. – URL: http://chipenable.ru/index.php/how-connection/9-sopryagaem-enkoder-i-mikrokontroller.htm (дата обращения: 20.09.2020).

72. Сторожевой таймер: [сайт]. 2020. – URL: https:// themagicsmoke.ru/courses/stm32/watchdog.html (дата обращения: 20.09.2020).

73. Таймер SysTick, реализация задержек в программе: [сайт]. 2020. – URL: http://microsin.net/programming/arm/stm32-systick.html (дата обращения: 20.09.2020).

74. Таймеры stm32 HAL – часть вторая: [сайт]. 2020. – URL: https://istarik.ru/blog/stm32/121.html (дата обращения: 20.09.2020).

75. Таймеры stm32 HAL – часть первая: [сайт]. 2020. – URL: https://istarik.ru/blog/stm32/118.html (дата обращения: 20.09.2020).

76. Таймеры улучшенного управления (TIM1) : [сайт]. 2020. – URL: http://pro-interes.com/wp-content/uploads/2018/12/ RM0041_ 12_Tim1.pdf (дата обращения: 20.09.2020).

77. Тест Си компиляторов под Windows: [сайт]. 2020. – URL: https://habr.com/ru/post/107664/ (дата обращения: 20.09.2020).

78. Урок 6. Порты ввода-вывода STM32: [сайт]. 2020. – URL: http://mypractic.ru/urok-6-porty-vvoda-vyvoda-stm32.html (дата обращения: 20.09.2020).

79. Частотомер на микроконтроллере stm32: [сайт]. 2020. – URL: https://istarik.ru/blog/stm32/124.html#cut (дата обращения: 20.09.2020).

80. Что такое часы реального времени? : [сайт]. 2020. – URL: https://stm32.chrns.com/post/150532640859/whatisrtc (дата обращения: 20.09.2020).

81. Что такое шина LIN: [сайт]. 2020. – URL: http://canhacker.ru/%D1%88%D0%B8%D0%BD%D0%B0-lin/%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA% D0%BE%D0%B5-%D1%88%D0%B8%D0%BD%D0%B0-lin/ (дата обращения: 20.09.2020).

82. Шина SMBus: [сайт]. 2020. – URL: https://wm-help.net/ lib/b/book/3065756330/150 (дата обращения: 20.09.2020).

83. Энкодер и шкала: [сайт]. 2020. – URL: http://robocraft.ru/blog/electronics/590.html (дата обращения: 20.09.2020).

84.: [сайт]. 2020. – URL: https://trolsoft.ru/ru/articles/encoder (дата обращения: 20.09.2020).

85. Ю. Джозеф. Ядро Cortex-M3 компании ARM. Полное руководство / Джозеф Ю.; пер. с англ. А.В. Евстифеева. – М.: Додека – XXI, 2012. – 552 с.: ил.

86. ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32: [сайт]. 2020. – URL: https://www.st.com/en/development-tools/st-link-v2.html (дата обращения: 20.09.2020).

87. Работаем с LCD дисплеем на основе микроконтроллера – HD44780 (ч.1) : [сайт]. 2020. – URL: http://s-engineer.ru/rabotaem-s-lcd-displeem-na-osnove-mikrokontrollera-hd44780-ch-1/ (дата обращения: 20.09.2020).

88. Подключение LCD(HD44780) по 4-х разрядной шине. Русификация LCD: [сайт]. 2020. – URL: https://radioparty.ru/prog-avr/program-c/268-lcd-avr-lesson2 (дата обращения: 20.09.2020).

89. Рисуем свой символ на LCD дисплее 1602A: [сайт]. 2020. – URL: https://hubstub.ru/display/73-risuem-svoy-simvol-na-lcd-displee-1602a.html (дата обращения: 20.09.2020).

90. Roger Meier's Freeware: [сайт]. 2020. – URL: http://freeware.the-meiers.org (дата обращения: 20.09.2020).

91. Самоучитель по Си с нуля: [сайт]. 2020. – URL: http://itrobo.ru/programmirovanie/samouchitel-po-si-s-nulja.html (дата обращения: 20.09.2020).

92. STM32 TIMER basic. Введение и описание регистров: [сайт]. 2020. – URL: http://mycontroller.ru/old_site/stm32-timer-basic-vvedenie/default.htm (дата обращения: 20.09.2020).

93. Защита прошивки от копирования: [сайт]. 2020. – URL: http://stm32.chrns.com/153904682324/ (дата обращения: 20.09.2020).

94. PM0063 Programming manual STM32F100xx value line Flash programming: [сайт]. 2020. – URL: https://www.st.com/content/ccc/resource/technical/document/programming_manual/33/85/c1/48/49/54/43/15/CD00246875.pdf/files/CD00246875.pdf/jcr:content/translations/en.CD00246875.pdf (дата обращения: 20.09.2020).

95. Исправление кодировки в среде STM32CubeIDE: [сайт]. 2020. – URL: http://www.mcu4you.ru/ispravlenie-kodirovki-v-srede-stm32cubeide/ (дата обращения: 20.09.2020).

96. Выводим собственные символы на LCD с HD44780: [сайт]. 2020. – URL: https://www.youtube.com/watch?v=JrFLn_G6gPk (дата обращения: 20.09.2020).

97. Микроконтроллеры STM32: использование встроенных RTC: [сайт]. 2020. – URL: https://eax.me/stm32-rtc/] (дата обращения: 20.09.2020).

98. STM32: example of usage of internal RTC: [сайт]. 2020. – URL: https://github.com/afiskon/stm32-rtc-example (дата обращения: 20.09.2020).

99. Mastering STM32 book: [сайт]. 2020. – URL: https:// github.com/cnoviello/mastering-stm32 (дата обращения: 20.09.2020).

100. Stm32 HAL OLED WH1602: [сайт]. 2020. – URL: http://www.cyberforum.ru/arm/thread2229703.html (дата обращения: 20.09.2020).

101. Преобразуем в строку. Часть 1. Целые числа.: [сайт]. 2020. – URL: http://we.easyelectronics.ru/blog/Soft/2400.html (дата обращения: 20.09.2020).

102. Отображение информации на жидкокристаллических индикаторах с контроллером HD44780: [метод. указания] / М-во образования и науки Рос. Федерации, Самар. нац. исслед. ун-т им. С.П. Королева (Самар. ун-т); сост. В. Г. Иоффе. – Самара: Изд-во Самар. ун-та, 2017. URL:http://repo.ssau.ru/bitstream/Metodicheskie-materialy/Otobrazhenie-informacii-na-zhidkokristallicheskih-indikatorah-s-kontrollerom-HD44780-Elektronnyi-resurs-metod-ukazaniya-70955.pdf

103. Анализ рынка микроконтроллеров. Прошлое и настоящее: [сайт]. 2020. – URL: https://commarketru.com/mikro kontrollery-proshloe-i-nastoyashhee/ (дата обращения: 20.09.2020).

# ПРИЛОЖЕНИЕ 1. Программная модель периферийных устройств микроконтроллера STM32F103C8T6

## П1.1. Контроллер прерываний

Распределение прерываний в регистрах

**Table 41. Mapping of interrupts to the interrupt variables**

| Interrupts | CMSIS array elements(1) | | | | |
|---|---|---|---|---|---|
| | Set-enable | Clear-enable | Set-pending | Clear-pending | Active Bit |
| 0-31 | ISER[0] | ICER[0] | ISPR[0] | ICPR[0] | IABR[0] |
| 32-63 | ISER[1] | ICER[1] | ISPR[1] | ICPR[1] | IABR[1] |
| 64-67 | ISER[2] | ICER[2] | ISPR[2] | ICPR[2] | IABR[2] |

1. Each array element corresponds to a single NVIC register, for example the element ICER[1] corresponds to the ICER1 register.

В связи с большим количеством векторов прерываний, в состав Cortex-M3 входит три регистра каждого типа с номерами **x = 0, 1, 2,** которые имеют один и тот же формат. Ниже приведена информация для регистра с номером 0.

**NVIC_ISERx Регистр разрешения прерываний**
Адрес смещения: 0x00 – 0x0B
Состояние после сброса: 0x0000 0000

Требуемая привилегия: Привилегированный

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SETENA[31:16] | | | | | | | | | | | | | | | |
| rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SETENA[15:0] | | | | | | | | | | | | | | | |
| rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs |

Биты 31:0 **SETENA[31:0]**: Биты разрешения соответствующих векторов прерываний.

**Запись**:0 – нет эффекта, 1– разрешение прерывания

**Чтение**:0 – прерывание запрещено, 1– прерывание разрешено.

**NVIC_ICERx Регистр запрещения прерываний**
Адрес смещения: 0x00 – 0x0B
Состояние после сброса: 0x0000 0000
Требуемая привилегия: Привилегированный

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CLRENA[31:16] | | | | | | | | | | | | | | | |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CLRENA[15:0] | | | | | | | | | | | | | | | |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

Биты 31:0 **CLRENA[31:0]**: Биты запрещения соответствующих векторов прерываний.
**Запись**: 0 – нет эффекта, 1– запрет прерывания
**Чтение**: 0 – прерывание запрещено, 1 – прерывание разрешено.

**NVIC_ISPRx Регистр установки прерывания в очередь на обработку**
Адрес смещения: 0x00 – 0x0B
Состояние после сброса: 0x0000 0000
Требуемая привилегия: привилегированный

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SETPEND[31:16] | | | | | | | | | | | | | | | |
| rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SETPEND[15:0] | | | | | | | | | | | | | | | |
| rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs | rs |

Биты 31:0 **SETPEND[31:0]**: Биты установки прерывания в очередь.
**Запись**: 0 – нет эффекта, 1 – перевод прерывания в очередь.
**Чтение**: 0 – прерывание не находится в очереди, 1 – перевод прерывания в очередь.

**NVIC_ICPRx Регистр удаления прерывания из очереди на обработку**
Адрес смещения: 0x00÷0x0B.
Состояние после сброса: 0x0000 0000.
Требуемая привилегия: привилегированный.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CLRPEND[31:16] | | | | | | | | | | | | | | | |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CLRPEND[15:0] | | | | | | | | | | | | | | | |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

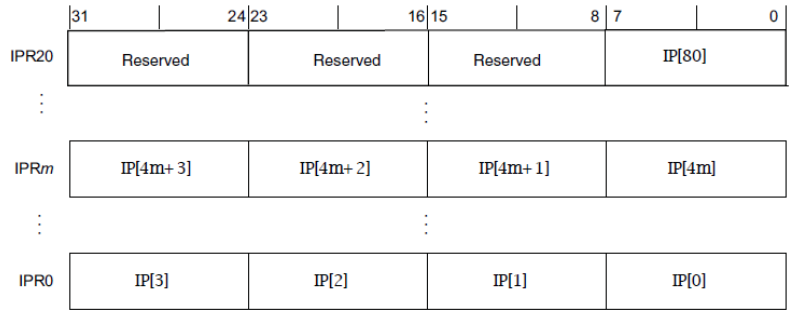
Биты 31:0 **CLRPEND[31:0]**: биты удаления прерывания из очереди.
**Запись**: 0 – нет эффекта, 1 – удаление прерывания из очереди.
**Чтение**: 0 – прерывание не находится в очереди,1 – прерывание ждет обработки.

**NVIC_IABRx Регистр активных прерываний**
Адрес смещения: 0x00÷0x0B.
Состояние после сброса: 0x0000 0000.
Требуемая привилегия: привилегированный.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ACTIVE[31:16] | | | | | | | | | | | | | | | |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ACTIVE[15:0] | | | | | | | | | | | | | | | |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |

Биты 31:0 **ACTIVE[31:0]**: биты прерывания.
Чтение: 0 – прерывание не активно, 1 – прерывание активно или находится в очереди.

**NVIC_IPRx Регистр установки приоритетов прерываний**
адрес смещения: 0x00÷0x0B;
состояние после сброса: 0x0000 0000;
требуемая привилегия: привилегированный.
Число регистров **NVIC_IPRx** определяется типом МК. В STM32F103C6T8 используется 17 регистров (**IPR0÷IPR16**). Приоритет задают числом от 0 до 255. Каждый регистр содержит значения приоритетов четырех прерываний.
**В микроконтроллере STM32F103C6T8 приоритеты задаются в старшей тетраде (16 уровней).**




Чтобы найти число IPR и смещение байта для прерывания N, надо:

- Соответствующий IPR номер M = N DIV 4
- Байтовое смещение для необходимого поля приоритета в этом регистре определяется как N MOD 4, где:
  - смещение байта 0 относится к битам регистра [7: 0];

– смещение байта 1 относится к битам регистра [15: 8];
– смещение байта 2 относится к битам регистра [23: 16];
– смещение байта 3 относится к битам регистра [31: 24].

**NVIC_STIR Регистр программного вызова прерываний**
Адрес смещения: 0xE00.
Состояние после сброса: 0x0000 0000.
Требуемая привилегия: когда бит USERSETMPEND в SCR установлен в «1», то код программы в непривилегированном режиме может обратиться к STIR, см. раздел 4.3.5: «Регистр управления системой (SCB_SCR)». Только привилегированный код может разрешить непривилегированному доступу к STIR.

Для вызова программного прерывания необходимо записать в регистр номера IRQ. Доступ к регистру определяется режимом работы МК. На **непривилегированном уровне** доступ возможен, если **установлен бит USERSETMPEND в регистре SCB_CCR**.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | | INTID[8:0] | | | | | | | | |
| | | | | | | | w | w | w | w | w | w | w | w | w |

Биты 7:0 **INTID[7:0]**: номер вызываемого прерывания.

## Регистры EXTI

**EXTI_IMR Регистр маски внешних прерываний**
Адрес смещения: 0x00.
Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | MR19 | MR18 | MR17 | MR16 |
| | | | | | | | | | | | | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MR15 | MR14 | MR13 | MR12 | MR11 | MR10 | MR9 | MR8 | MR7 | MR6 | MR5 | MR4 | MR3 | MR2 | MR1 | MR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:20 Зарезервировано, должно сохраняться при сбросе значения (0).
Биты 19:0 MRx: Маскирование прерывания на линии x
0: Запрос прерывания от линии x запрещен
1: Запрос прерывания от линии x разрешен
**Замечание.** бит 19 используется только в семействе Connectivity Line и зарезервирован у других семейств.

**EXTI_EMR Регистр маски внешних событий**

Адрес смещения: 0x04

Состояние после сброса: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | MR19 | MR18 | MR17 | MR16 |
| | | | | | | | | | | | | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MR15 | MR14 | MR13 | MR12 | MR11 | MR10 | MR9 | MR8 | MR7 | MR6 | MR5 | MR4 | MR3 | MR2 | MR1 | MR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:20 зарезервированы, при записи всегда 0.

Биты 19:0 MRx: маскирование события на линии x.

0: Запрос события от линии x запрещен.

1: Запрос события от линии x разрешен.

**Замечание.** Бит 19 используется только в семействе Connectivity Line и зарезервирован у других семейств.

**EXTI_RTSR Регистр выбора срабатывания прерывания/события по фронту**

Адрес смещения: 0x08.

Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | TR19 | TR18 | TR17 | TR16 |
| | | | | | | | | | | | | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TR15 | TR14 | TR13 | TR12 | TR11 | TR10 | TR9 | TR8 | TR7 | TR6 | TR5 | TR4 | TR3 | TR2 | TR1 | TR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:20 Зарезервированы, при записи всегда 0.

Биты 19:0 TRx: конфигурирует биты запуска линии x по фронту:

0 – запуск по фронту сигнала на соответствующей линии запрещен (для события и прерывания);

1 – запуск по фронту сигнала на соответствующей линии разрешен (для события и прерывания).

**Замечание.** Бит 19 используется только в семействе «Connectivity Line» и зарезервирован у других семейств.

**Замечание.** Внешние линии пробуждения запускаются по перепаду, никаких помех не должно допускаться на этих линиях. Если фронт сигнала произойдет на внешней линии прерывания во время записи регистра EXTI_RTSR, то бит запроса прерывания не будет установлен. Оба разрешения на запуск от фронта и среза можно устанавливать для одной линии. В этой конфигурации оба перепада генерируют условие запуска.

**EXTI_FTSR Регистр выбора срабатывания прерывания/события по срезу**

Адрес смещения: 0x0C.

Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | TR19 | TR18 | TR17 | TR16 |
| | | | | | | | | | | | | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TR15 | TR14 | TR13 | TR12 | TR11 | TR10 | TR9 | TR8 | TR7 | TR6 | TR5 | TR4 | TR3 | TR2 | TR1 | TR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:20 зарезервированы, при записи всегда «0».

Биты 19:0 TRx конфигурирует биты запуска линии x по срезу:

0 – запуск по срезу сигнала на соответствующей линии запрещен (для события и прерывания);

1 – запуск по срезу сигнала на соответствующей линии разрешен (для события и прерывания).

**Замечание.** Бит 19 используется только в семействе «Connectivity Line» и зарезервирован у других семейств.

**Замечание.** Внешние линии пробуждения запускаются по перепаду, никаких помех не должно допускаться на этих линиях. Если фронт сигнала произойдет на внешней линии прерывания во время записи регистра EXTI_FTSR, то бит запроса прерывания не будет установлен. Оба разрешения на запуск от фронта и среза можно устанавливать для одной линии. В этой конфигурации оба перепада генерируют условие запуска.

**EXTI_SWIER** Регистр программного вызова прерывания/события:

Адрес смещения: 0x10.

Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | SWIER 19 | SWIER 18 | SWIER 17 | SWIER 16 |
| | | | | | | | | | | | | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWIER 15 | SWIER 14 | SWIER 13 | SWIER 12 | SWIER 11 | SWIER 10 | SWIER 9 | SWIER 8 | SWIER 7 | SWIER 6 | SWIER 5 | SWIER 4 | SWIER 3 | SWIER 2 | SWIER 1 | SWIER 0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:20 зарезервированы, при записи всегда «0».

Биты 19:0 SWIERx: программное прерывание на линии x:

Запись «1» в этот бит, когда он был равен «0», ставит соответствующий флаг запроса в **EXTI_PR**. Если

разрешено прерывание от этой линии в регистрах **EXTI_IMR** и **EXTI_EMR**, то будет сгенерирован запрос на прерывание.

Сброс осуществляется записью «1» в соответствующий бит **EXTI_PR**.

**Замечание.** Бит 19 используется только в семействе Connectivity Line и зарезервирован у других семейств.

**EXTI_PR Регистр текущего прерывания/события и сброса.**

Адрес смещения: 0x14.

Состояние после сброса: неопределенное.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | PR19 | PR18 | PR17 | PR16 |
| | | | | | | | | | | | | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PR15 | PR14 | PR13 | PR12 | PR11 | PR10 | PR9 | PR8 | PR7 | PR6 | PR5 | PR4 | PR3 | PR2 | PR1 | PR0 |
| rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 | rc_w1 |

Биты 31:20 зарезервированы, при записи всегда «0».

Биты 19:0 PRx: Бит запроса:

«0» – нет запроса;

«1» – сброс запроса прерывания.

Этот бит ставится, когда на внешнюю линию прерывания поступило выбранное условие запуска. Этот бит очищается записью в него «1» или изменением чувствительности детектора перепада.

**Замечание.** Бит 19 используется только в семействе Connectivity Line и зарезервирован у других семейств. При настройке внешних прерываний необходимо использовать регистры **AFIO.**

**AFIO_EXTICR1 Конфигурационный регистр 1 внешних прерываний:**

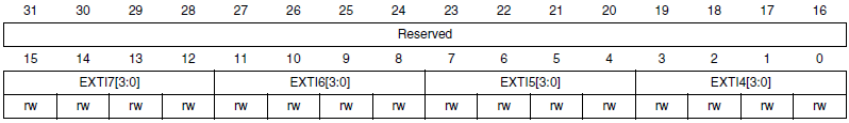
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EXTI3[3:0] | | | | EXTI2[3:0] | | | | EXTI1[3:0] | | | | EXTI0[3:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

**EXTIx[3:0]:** Конфигурация мультиплексора канала *x EXTI*:

- 0000: Выбор пина PA[x];
- 0001: Выбор пина PB[x];
- 0010: Выбор пина PC[x];

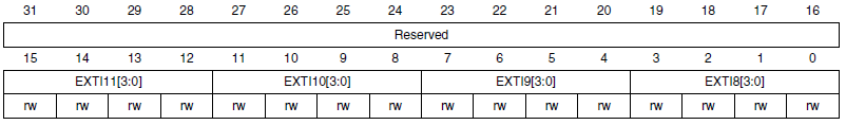
- 0011: Выбор пина PD[x];
- 0100: Выбор пина PE[x];
- 0101: Выбор пина PF[x];
- 0110: Выбор пина PG[x].

**AFIO_EXTICR2 Конфигурационный регистр 2 внешних прерываний:**



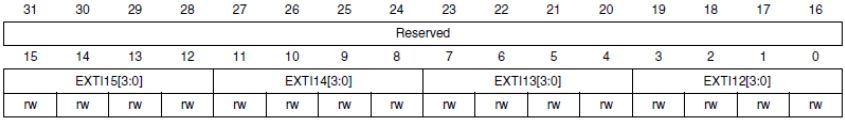

Тут все то же самое, что и для *AFIO_EXTICR1*, только для каналов *EXTI4..7*.

**AFIO_EXTICR3 Конфигурационный регистр 3 внешних прерываний:**




Конфигурация каналов *EXTI8..11*.

**AFIO_EXTICR4 Конфигурационный регистр 4 внешних прерываний:**



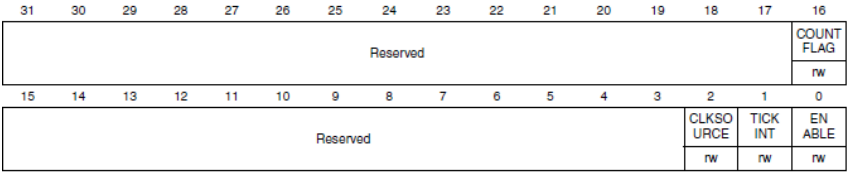

Конфигурация каналов *EXTI12..15.*

## П1.2. Системный таймер

**STK_CTRL Регистр управления и состояния**
Адрес смещения: 0x00.
Состояние после сброса: 0x0000 0004.

Required privilege: Privileged

The SysTick CTRL register enables the SysTick features.




Бит 0 – ENABLE – разрешение счета:

«1» – запуск таймера,

«0» – останов.

При ENABLE = 1 в счетный регистр SysTick_VAL автоматически загружается содержимое регистра SYST_RVR и начинается счет.

Бит 1 – TICKINT – разрешение прерывания:

«1» – разрешение,

«0» – запрет.

Прерывание возникает, если установлен бит COUNTFLAG и TICKINT = 1.

Бит 2 – CLKSOURCE – выбор источника синхронизации.

«0» – синхронизировать частотой f HCLK /8,

«1» – частотой ядра f HCLK.

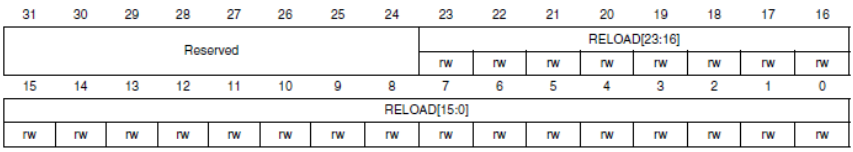
Бит 16 – COUNTFLAG – флаг переполнения. Устанавливается в «1» автоматически при переполнении счётчика (попытка перехода из состояния «0» в состояние – «1»). Сбрасывается в «0» программно.

**STK_LOAD Регистр перезагрузки**

Адрес смещения: 0x04.

Состояние после сброса: 0x0000 0000.

Required privilege: Privileged




Содержит данные, загружаемые в SysTick_VAL, при запуске системного таймера и переполнении.

**STK_VAL Счетчик системного таймера**
Адрес смещения: 0x08.
Состояние после сброса: 0x0000 0000.

Required privilege: Privileged

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | CURRENT[23:16] | | | | | | | |
| | | | | | | | | rw | rw | rw | rw | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CURRENT[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Текущее значение счетчика. При записи в STK_VAL происходит его сброс и перезагрузка SySTick_LOAD, но флаг COUNTFLAG не устанавливается.

**STK_CALIB Регистр калибровки**
Адрес смещения: 0x0C.
Состояние после сброса: 0x0000 2328.

Required privilege: Privileged

The CALIB register indicates the SysTick calibration properties.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NO REF | SKEW | Reserved | | | | | | TENMS[23:16] | | | | | | | |
| r | r | | | | | | | r | r | r | r | r | r | r | r |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TENMS[15:0] | | | | | | | | | | | | | | | |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |

Данный регистр доступен только для чтения и содержит информацию об особенностях калибровки.

Бит 31 – NOREF указывает на наличие (NOREF = 0) или отсутствие (NOREF = 1) эталонной частоты HCLK/8 на входе системного таймера.

Бит 30 – SKEW – характеризует точность калибровочного значения в поле TENMS («0» – значение точное, «1» – значение неточное или не задано).

Биты (23:0) – TENMS – содержит значение калибровки, когда счетчик SysTick работает от тактов HCLK max/8. Реальное значение зависит от конкретного чипа и определяется в процессе производства. Когда HCLK запрограммирован на максимальную частоту, период таймера SysTick равен 1 мс.

# П1.3. Контроллер ПДП (DMA1)

**DMA_ISR Регистр состояния прерываний ПДП**
Адрес смещения: 0x00.
Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | TEIF7 | HTIF7 | TCIF7 | GIF7 | TEIF6 | HTIF6 | TCIF6 | GIF6 | TEIF5 | HTIF5 | TCIF5 | GIF5 |
| | | | | r | r | r | r | r | r | r | r | r | r | r | r |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TEIF4 | HTIF4 | TCIF4 | GIF4 | TEIF3 | HTIF3 | TCIF3 | GIF3 | TEIF2 | HTIF2 | TCIF2 | GIF2 | TEIF1 | HTIF1 | TCIF1 | GIF1 |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |

Биты 31:28 зарезервировано, всегда читается как «0».
Биты 27, 23, 19, 15, 11, 7, 3 **TEIFx:** флаг **ошибки** обмена в канале x (x = 1 .. 7). Устанавливается аппаратно. Сбрасывается программно записью «1» в соответствующий бит регистра **DMA_IFCR**.

«0» – нет ошибки обмена (TE) в канале x.

«1» – есть ошибка обмена (TE) в канале x.

Биты 26, 22, 18, 14, 10, 6, **HTIFx:** флаг **завершения половины** обмена в канале x (x = 1 .. 7). Устанавливается аппаратно. Сбрасывается программно записью «1» в соответствующий бит регистра **DMA_IFCR**.

«0» – нет события завершения половины обмена (HT) в канале x.

«1» – есть событие завершения половины обмена (HT) в канале x.

Биты 25, 21, 17, 13, 9, 5, 1 **TCIFx:** Флаг з**авершения** обмена в канале x (x = 1 ..7). Устанавливается аппаратно. Сбрасывается программно записью «1» в соответствующий бит регистра **DMA_IFCR**.

«0» – нет события завершения обмена (TC) в канале x.

«1» – есть событие завершения обмена (TC) в канале x.

Биты 24, 20, 16, 12, 8, 4, 0 **GIFx: Общий** флаг прерывания в канале x (x = 1 .. 7). Устанавливается аппаратно. Сбрасывается программно записью «1» в соответствующий бит регистра DMA_IFCR.

«0» – нет ни одного события прерывания TE, HT или TC в канале x.

«1» – есть одно из событий прерывания TE, HT или TC в канале x.

**DMA_IFCR Регистр очистки флагов прерываний ПДП**
Адрес смещения: 0x04.
Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | CTEIF7 | CHTIF7 | CTCIF7 | CGIF7 | CTEIF6 | CHTIF6 | CTCIF6 | CGIF6 | CTEIF5 | CHTIF5 | CTCIF5 | CGIF5 |
| | | | | w | w | w | w | w | w | w | w | w | w | w | w |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CTEIF4 | CHTIF4 | CTCIF4 | CGIF4 | CTEIF3 | CHTIF3 | CTCIF3 | CGIF3 | CTEIF2 | CHTIF2 | CTCIF2 | CGIF2 | CTEIF1 | CHTIF1 | CTCIF1 | CGIF1 |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

Биты 31:28 зарезервировано, всегда читается как «0».
Биты 27, 23, 19, 15, 11, 7, 3 **CTEIFx:** очистка флага ошибки обмена в канале x (x = 1 .. 7). Этот бит устанавливается и сбрасывается программно.

«0» – нет эффекта.

«1» – очистка соответствующего TEIF флага в регистре DMA_ISR.
Биты 26, 22, 18, 14, 10, 6, 2 **CHTIFx:** очистка флага завершения половины обмена в канале x (x = 1 .. 7). Этот бит устанавливается и сбрасывается программно.

«0» – нет эффекта.

«1» – очистка соответствующего HTIF флага в регистре DMA_ISR.
Биты 25, 21, 17, 13, 9, 5, 1 **CTCIFx:** очистка флага завершения обмена в канале x (x = 1 .. 7). Этот бит устанавливается и сбрасывается программно.

«0» – нет эффекта.

«1» – очистка соответствующего TCIF флага в регистре DMA_ISR.
Биты 24, 20, 16, 12, 8, 4, 0**CGIFx:** глобальная очистка флагов прерывания в канале x (x = 1 .. 7). Этот бит устанавливается и сбрасывается программно.

«0» – нет эффекта.

«1» – очистка флагов GIF, TEIF, HTIF и TCIF в регистре DMA_ISR.

**DMA_CCRx Регистр конфигурации канала x ПДП** (x = 1..7, где x – номер канала)
Адрес смещения: 0x08 + 0d20 × (номер канала – 1).
Состояние после сброса: 0x0000 0000.

**Все биты этого регистра устанавливаются и сбрасываются программно.**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Res. | MEM2 MEM | PL[1:0] | | MSIZE[1:0] | | PSIZE[1:0] | | MINC | PINC | CIRC | DIR | TEIE | HTIE | TCIE | EN |
| | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:15 Зарезервировано, всегда читается как «0».
Бит 14 **MEM2MEM:** режим «память-память».
0: режим «память-память» запрещен.
1: режим «память-память» разрешен.
Биты 13:12 **PL[1:0]:** уровень приоритета канала:
00: низкий;
01: средний;
10: высокий;
11: очень высокий.
Биты 11:10 **MSIZE[1:0]:** размер элемента данных памяти.
00: 8 бит;
01: 16 бит;
10: 32 бита;
11: зарезервирован.
Биты 9:8 **PSIZE[1:0]:** размер элемента данных периферии.
00: 8 бит;
01: 16 бит;
10: 32 бита;
11: зарезервирован.
Бит 7 **MINC:** режим инкремента указателя памяти.
0: режим инкремента указателя памяти запрещен.
1: режим инкремента указателя памяти разрешен.
Бит 6 **PINC:** режим инкремента указателя периферии.
0: режим инкремента указателя периферии запрещен.
1: режим инкремента указателя периферии разрешен.
Бит 5 **CIRC:** режим цикличности.
0: режим цикличности запрещен.
1: режим цикличности разрешен.
Бит 4 **DIR:** Направление обмена данных.
0: чтение из периферии.
1: чтении из памяти.
Бит 3 **TEIE:** Разрешение прерывания от ошибки обмена.
0: прерывание от TE запрещено.
1: прерывание от TE разрешено.

Бит 2 **HTIE:** разрешение прерывания от события завершения половины обмена.

0: прерывание от HT запрещено.

1: прерывание от HT разрешено.

Бит 1 **TCIE:** разрешение прерывания от события завершения обмена.

0: прерывание от TC запрещено.

1: прерывание от TC разрешено.

Бит 0 **EN:** разрешение канала.

0: канал запрещен.

1: канал разрешен.

**DMA_CNDTRx Регистр размера данных канала x ПДП** (x = 1..7, где x = номер канала)

Адрес смещения: 0x0C + 0d20 × (номер канала – 1).

Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NDT | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:16 зарезервировано, всегда читается как «0».

Бит 15:0 **NDT[15:0]:** количество данных обмена.

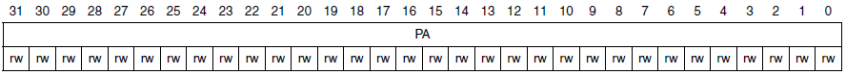
Количество данных, которые будут переданы (от 0 до 65535). В этот регистр можно производить запись только тогда, когда канал выключен. Как только канал будет разрешен, этот регистр можно будет только читать, чтобы определить число байт, оставшихся для обмена. Регистр декрементируется после каждой DMA транзакции.

Как только обмен завершен, регистр может остаться в нуле или автоматически перезагрузиться ранее запрограммированным значением, если канал сконфигурирован в режиме автоперезагрузки (режим цикличности).

Если значение этого регистра равно нулю, никакая транзакция не может быть обслужена независимо от того, разрешен канал или нет.

**DMA_CPARx Регистр адреса периферии для канала x ПДП** (x = 1..7, где x = номер канала)

Адрес смещения: 0x10 + 0d20 × (номер канала – 1).
Состояние после сброса: 0x0000 0000.
В этот регистр нельзя производить запись, когда канал разрешен.



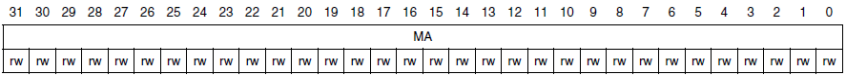

Биты 31:0 **PA[31:0]:** адрес периферии.

Регистр базового адреса данных в периферии, из/по которому будут читаться/записываться данные.

Если PSIZE = 01 (элемент 16-битный), бит PA[0] игнорируется. Доступ автоматически выравнивается по адресу полуслова.

Если PSIZE = 10 (элемент 32-битный), биты PA[1:0] игнорируются. Доступ автоматически выравнивается по адресу слова.

**DMA_CMARx Регистр адреса в памяти для канала x ПДП (x = 1..7, где x = номер канала)**
Адрес смещения: 0x14 + 0d20 × (номер канала – 1).
Состояние после сброса: 0x0000 0000.
**В этот регистр нельзя производить запись, когда канал разрешен.**




Биты 31:0 **MA[31:0]:** адрес в памяти.

Регистр базового адреса данных в памяти, из/по которому будут читаться/записываться данные.

Если MSIZE = 01 (элемент 16-битный), бит MA[0] игнорируется. Доступ автоматически выравнивается по адресу полуслова.

Если MSIZE = 10 (элемент 32-битный), биты MA[1:0] игнорируются. Доступ автоматически выравнивается по адресу слова.

## П1.4. Параллельный порт GPIO STM32

**GPIOx_CRL** Младший регистр конфигурации портов (*Port configuration register low*):

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNF7[1:0] | | MODE7[1:0] | | CNF6[1:0] | | MODE6[1:0] | | CNF5[1:0] | | MODE5[1:0] | | CNF4[1:0] | | MODE4[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CNF3[1:0] | | MODE3[1:0] | | CNF2[1:0] | | MODE2[1:0] | | CNF1[1:0] | | MODE1[1:0] | | CNF0[1:0] | | MODE0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

**GPIOx_CRH** Старший регистр конфигурации портов (*Port configuration register high*):

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNF15[1:0] | | MODE15[1:0] | | CNF14[1:0] | | MODE14[1:0] | | CNF13[1:0] | | MODE13[1:0] | | CNF12[1:0] | | MODE12[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CNF11[1:0] | | MODE11[1:0] | | CNF10[1:0] | | MODE10[1:0] | | CNF9[1:0] | | MODE9[1:0] | | CNF8[1:0] | | MODE8[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

На каждый вывод отводится 4 бита, которые делятся на 2-битные поля режима и конфигурации (см.1.6.1). Поле режима (биты Mode):

| Биты режима MODE [1 : 0] | Режим |
|---|---|
| 0 0 | Вход |
| 0 1 | Выход, синхронизация 10 мГц |
| 1 0 | Выход, синхронизация 2 мГц |
| 1 1 | Выход, синхронизация 50 мГц |

Поле конфигурации:

| Конфигурация | | Биты конфигурации CNF [1÷0] | Биты режима MODE [1÷0] | Бит в регистре вывода данных PxODR |
|---|---|---|---|---|
| Выход общего назначения | Активный выход | 00 | 01<br>10<br>11 | 0 или 1 |
| | Открытый сток | 01 | | 0 или 1 |
| Выход альтернативной функции | Активный выход | 10 | | — |
| | Открытый сток | 11 | | — |
| Вход | Аналоговый вход | 00 | 00 | — |
| | Свободный вход | 01 | | — |
| | Вход, подтяжка к земле | 10 | | 0 |
| | Вход, подтяжка к питанию | | | 1 |

**GPIOx_LCKR** Регистр блокировки конфигурации порта (*Port configuration lock register*):

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | LCKK |
| | | | | | | | | | | | | | | | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LCK15 | LCK14 | LCK13 | LCK12 | LCK11 | LCK10 | LCK9 | LCK8 | LCK7 | LCK6 | LCK5 | LCK4 | LCK3 | LCK2 | LCK1 | LCK0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Установка бита LCKn в «1» запрещает изменение соответствующих битов режима и конфигурации, где n – номер линии порта.
Бит LCKK – включение блокировки последовательной записью «1», «0», «1». Проверка установки режима защиты выполняется двукратным последовательным чтением LCKK. При установленной защите результат должен быть «0», «1».

**GPIOx_IDR** Регистр ввода данных порта (*Port input data register*):

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IDR15 | IDR14 | IDR13 | IDR12 | IDR11 | IDR10 | IDR9 | IDR8 | IDR7 | IDR6 | IDR5 | IDR4 | IDR3 | IDR2 | IDR1 | IDR0 |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |

Биты IDR0 – IDR15 содержат состояние соответствующих выводов порта.
Доступен только для чтения.

**GPIOx_ODR** Регистр вывода данных порта (*Port output data register*):

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ODR15 | ODR14 | ODR13 | ODR12 | ODR11 | ODR10 | ODR9 | ODR8 | ODR7 | ODR6 | ODR5 | ODR4 | ODR3 | ODR2 | ODR1 | ODR0 |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Запись данных во второй регистр устанавливает состояние выводов порта (ODR0 – ODR15).
Если **задан режим вывода – вход**, то состояние соответствующего бита ODR0 – ODR15 определяет, куда подключен подтягивающий резистор, к шине питания или земле.

**GPIOx_BSRR** Регистр установки/сброса битов (*Port Бит set/reset register*):

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BR15 | BR14 | BR13 | BR12 | BR11 | BR10 | BR9 | BR8 | BR7 | BR6 | BR5 | BR4 | BR3 | BR2 | BR1 | BR0 |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BS15 | BS14 | BS13 | BS12 | BS11 | BS10 | BS9 | BS8 | BS7 | BS6 | BS5 | BS4 | BS3 | BS2 | BS1 | BS0 |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

Запись «1» в биты BS0÷BS15 устанавливает соответствующие выводы в состояние высокого уровня.

Запись «1» в биты BR0÷BR15 переводит соответствующие выводы в состояние низкого уровня.

**GPIOx_BRR** Регистр сброса битов (*Port Bum reset register*):

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BR15 | BR14 | BR13 | BR12 | BR11 | BR10 | BR9 | BR8 | BR7 | BR6 | BR5 | BR4 | BR3 | BR2 | BR1 | BR0 |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

Запись 1 в биты BR0 – BR15 переводит соответствующие выводы в состояние низкого уровня.

## П1.5. Счетчик-таймер TIM2-TIM5

**TIMx_CR1 Управляющий регистр 1**

Адрес смещения: 0x00.

Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | CKD[1:0] | | ARPE | CMS | | DIR | OPM | URS | UDIS | CEN |
| | | | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 9:8**: CKD (Clock division)** – настройки делителя. Это поле определяет период сигнала DTS, который используется для фильтрации сигнала при внешней синхронизации таймера $t_{CK\_INT}$ период тактовой частоты таймера:

**00:** $t_{DTS} = t_{CK\_INT;}$

**01:** $t_{DTS} = 2 * t_{CK\_INT;}$

**10:** $t_{DTS} = 4 * t_{CK\_INT;}$

**11:** Зарезервировано.

Бит 7**: ARPE (Auto-reload preload enable)** – включение предварительной загрузки регистра ARR:

**«0»** – буферизация регистра TIMx_ARR отключена;

**«1»** – буферизация регистра TIMx_ARR включена.

Биты 6÷5: **CMS (Center-aligned mode selection)** – выбор режима выравнивания по краю или центру:

**00**: режим с выравниванием по краю: счёт ведётся вверх (режим суммирующего счётчика) или вниз (режим вычитающего счётчика) в зависимости от значения бита направления DIR;

**01**: режим «1» с выравниванием по центру: счёт ведётся вверх и вниз поочерёдно, но при счёте вниз происходит установка флага

прерывания при сравнении для каналов, которые сконфигурированы как выходы (CCxS = 00 в регистре TIMx_CCMRx);

**10**: режим «2» с выравниванием по центру: счёт ведётся вверх и вниз поочерёдно, но при счёте вверх происходит установка флага прерывания при сравнении для каналов, сконфигурированных как выходы (CCxS = 00 в регистре TIMx_CCMRx);

**11**: режим 3 с выравниванием по центру: счёт ведётся вверх и вниз поочерёдно, установка флага прерывания при сравнении для каналов, сконфигурированных как выходы (CCxS = 00 в регистре TIMx_CCMRx) происходит как при счёте вверх, так и при счёте вниз.

Бит 4**: DIR (Direction)** – бит направления счёта:

**«0»** – счёт вверх (суммирующий);

**«1»** – счёт вниз (вычитающий).

Бит 3**: OPM (One-pulse mode)** –режим одиночного импульса:

**«0»** – при появлении события обновления счётчик не останавливается;

**«1»** – при появлении события обновления счётчик останавливается.

Бит 2**: URS (Update request source)** – бит источника запроса на обновление:

**«0»** – к генерации прерывания обновления или запроса DMA (если разрешено) привело любое из следующих событий:

- переполнение счётчика;
- установка бита UG регистра генерации событий TIMx_EGR;
- обновление от подчиненного таймера, если организован режим каскадного соединения.

**«1»** – только переполнение таймера.

Бит 1**: UDIS (Update disable)** – бит устанавливается и очищается программно для разрешения/запрета генерации события обновления (UEV):

**«0»** – разрешена генерация UEV: случаи, в которых генерируется UEV, определяются значением бита URS;

**«1»** – генерация UEV отключена: теневые регистры сохраняют свои значения неизменными (ARR, PSC, CCRx), но счётчик и

предделитель сбрасывается при установке бита UG или при получении сигнала аппаратного сброса от контроллера подчинённого режима.

Бит 0: **CEN (Counter enable)** – бит включения счётчика:

**«0»** – счётчик выключен;

**«1»** – счётчик включен.

### TIMx_CR2 Управляющий регистр 2

Адрес смещения: 0x04.

Состояние после сброса: 0x0000.


Бит 7**: TI1S (TI1 selection)** – управление входом TI1 блока захвата №1:

**«0»** – к входу TI1 подключается вывод TIMx_CH1 микроконтроллера;

**«1»** – на вход TI1 подаётся XOR (исключающее ИЛИ) от сигналов с выводов TIMx_CH1, CH2 и CH3 (TI1 = CH1 xor CH2 xor CH3).

Биты 6:4**: MMS2:MMS0 (Master mode selection)** – управление выходом TRGO **ведущего** таймера:

**000:** Reset – сброс, в качестве сигнала TRGO используется бит UG регистра TIMx_EGR;

**001:** Enable – включение, в качестве сигнала TRGO используется сигнал запуска счёта CNT_EN. Режим может использоваться, например, для одновременного запуска нескольких таймеров или для управления «окном», в течении которого включён подчинённый таймер;

**010:** Update – событие обновления UEV. В этом режиме можно использовать один таймер в качестве предварительного делителя другого таймера;

**011:** Compare Pulse – импульс TRGO формируется при установлении флага CC1IF регистра TIMx_SR (даже если он уже содержал значение «1»), это происходит при захвате или при срабатывании схемы сравнения;

**100:** Compare – сигнал OC1REF используется в качестве TRGO;

**101:** Compare – сигнал OC2REF используется в качестве TRGO;

**110:** Compare – сигнал OC3REF используется в качестве TRGO;
**111:** Compare – сигнал OC4REF используется в качестве TRGO.

**OCnREF** – сигнал, формируемый соответствующим каналом захвата/сравнения.

Бит 3: **CCDS (Capture/compare DMA selection)** – выбор источника запроса DMA:

**«0»** – CCx запрос DMA формируется, когда происходит событие CCx;

**«1»** – CCx запрос DMA формируется, когда происходит событие обновления.

**TIMx_SR Регистр состояния**

Адрес смещения: 0x10.

Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | CC4OF | CC3OF | CC2OF | CC1OF | Reserved | | TIF | Res | CC4IF | CC3IF | CC2IF | CC1IF | UIF |
| | | | rc_w0 | rc_w0 | rc_w0 | rc_w0 | | | rc_w0 | | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 |

Биты 12:9: **CC4OF:CC1OF (Capture/Compare 4÷1 overcapture flag)** – Переполнения флагов в режиме захвата. Устанавливаются аппаратно в случае, если происходит фиксация значения счётчика в данном канале (фиксируемое значение помещается в регистр TIMx_CCRy, y = 1..4) и при этом флаг фиксации CCyIF у этого канала уже установлен. Сброс бита выполняется программно путём записи в него значения «0».

«0» – переполнения флага не происходило;

«1» – обнаружено переполнение флага.

Бит 6: **TIF (Trigger interrupt flag)** – Флаг прерывания запуска. Устанавливается аппаратно триггерным событием (trigger event), которое возникает под действием активного фронта сигнала на входе TRGI при включенном ведомом таймере во всех режимах, кроме стробирующего. Сбрасывается программно.

«0» – событие запуска не происходило;

«1» – обнаружено прерывание запуска.

Биты 4:1**: CC4IF (Capture/Compare 4…1 interrupt flag)** – флаги прерывания захвата/сравнения в соответствующем канале:

«0» – захвата/сравнения в канале не обнаружено;

«1» – в канале произошел захват/сравнение.

Бит 0: **UIF (Update interrupt flag)** – флаг прерывания при возникновении события обновления. Устанавливается аппаратно, сбрасывается программно:

«0» – событие обновления не происходило;

«1» – обнаружено событие обновления.

**TIMx_EGR Регистр генерации событий**

Адрес смещения: 0x14.

Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | TG | Res. | CC4G | CC3G | CC2G | CC1G | UG |
| | | | | | | | | | w | | w | w | w | w | w |

Биты регистра работают только на запись.

Бит 6: **TG (Trigger generation)** – генерация триггерных событий (запуска), устанавливается программно:

«0» – событие не генерируется;

«1» – генерация триггерного события (имитируется сигнал TRGI), устанавливается флаг TIF в регистре TIMx_SR. Возможна генерация прерывания и запроса DMA (если они разрешены).

Бит 4:1**: CC4G, CC3G, CC2G, CC1G (Capture/compare 4÷1 generation)** – принудительная генерация события захвата/сравнения. Устанавливаются программно для генерации события (Capture/Compare) соответствующего канала таймера (1÷4), сброс бита происходит аппаратно:

«0» – событие не генерируется;

«1» – генерация события Capture/Compare в соответствующем канале таймера (1÷4).

**В режиме сравнения (выход)** устанавливается флаг CCnIF и вызываются прерывание или запрос DMA (если они разрешены).

**В режиме захвата (вход)** текущее значение счетчика записывается в регистр TIMx_CCRn. Устанавливается флаг CCnIF, при разрешении посылается соответствующее прерывание или запрос DMA. Флаг CCnOF устанавливается если флаг CCnIF уже был высоким.

Бит 0**: UG (Update generation)** – принудительная генерация события обновления. Данный бит устанавливается программно, сбрасывается аппаратно.

«0» – событие не генерируется;

«1» – генерация события обновления: происходит инициализация счётчика и обновляется содержимое буферизируемых регистров, сбрасывается счётчик предделителя (коэффициент деления не изменяется). В режиме с выравниванием по центру, счётчик таймера сбрасывается в 0 при DIR = 0 (во время счёта вверх) и инициализируется значением из регистра TIMx_ARR при DIR = 1 (счёт вниз).

**TIMx_ SMCR Регистр управления в режиме ведомого таймера**
Адрес смещения: 0x08.
Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ETP | ECE | ETPS[1:0] | | ETF[3:0] | | | | MSM | TS[2:0] | | | Res. | SMS[2:0] | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | | rw | rw | rw |

Бит 15**: ETP (External trigger polarity)** – выбор активного состояния внешнего сигнала ETR:

«0» – ETR не инвертирован, активным является высокий уровень или фронт;

«1» – ETR инвертирован, активным является низкий уровень или срез.

Бит 14: **ECE (External clock enable)** – включение режима 2 внешней синхронизации:

«0» – режим 2 отключен;

«1» – режим 2 включён.

Биты 13:12**: ETPS (External trigger prescaler)** – **ETPS[1:0]**. Предделитель внешнего запуска:

00 – предделитель отключен;

01– частота ETR делится на 2;

10– частота ETR делится на 4;

11– частота ETR делится на 8.

Биты 11÷8: **ETF3:ETF0 (External trigger filter)** – частота фильтрации $f_{SAMPLING}$ и количество выборок N.

**0000:** фильтр не используется, частота сэмплирования равна fDTS;

**0001:** $f_{SAMPLING} = f_{CK\_INT}$, N=2;

**0010:** $f_{SAMPLING} = f_{CK\_INT}$, N=4;

**0011:** $f_{SAMPLING} = f_{CK\_INT}$, N=8;

**0100:** $f_{SAMPLING} = f_{DTS}/2$, N=6;

**0101:** $f_{SAMPLING} = f_{DTS}/2$, N=8;
**0110:** $f_{SAMPLING} = f_{DTS}/4$, N=6;
**0111:** $f_{SAMPLING} = f_{DTS}/4$, N=8;
**1000:** $f_{SAMPLING} = f_{DTS}/8$, N=6;
**1001:** $f_{SAMPLING} = f_{DTS}/8$, N=8;
**1010:** $f_{SAMPLING} = f_{DTS}/16$, N=5;
**1011:** $f_{SAMPLING} = f_{DTS}/16$, N=6;
**1100:** $f_{SAMPLING} = f_{DTS}/16$, N=8;
**1101:** $f_{SAMPLING} = f_{DTS}/32$, N=5;
**1110:** $f_{SAMPLING} = f_{DTS}/32$, N=6;
**1111:** $f_{SAMPLING} = f_{DTS}/32$, N=8.

Бит 7**: MSM (Master/Slave mode)** – включение режима ведущий/ведомый:

**«0»** – режим отключен;

**«1»** – режим включён. Действие события по входу запуска (TRGI) задерживается, чтобы совершить синхронизацию между ведущим таймером и ведомыми (через TRGO). Это полезно, если необходимо синхронизировать несколько таймеров по одному внешнему событию.

Биты 6:4: **TS (Trigger selection)** – выбор источника синхронизации счётчика:

**000:** Internal Trigger 0 (ITR0) – внутренний сигнал 0;

**001:** Internal Trigger 1 (ITR1) – внутренний сигнал 1;

**010:** Internal Trigger 2 (ITR2) – внутренний сигнал 2;

**011:** Internal Trigger 3 (ITR3) – внутренний сигнал 3;

**100:** TI1 Edge Detector (TI1F_ED) – детектор фронта сигнала TI1;

**101:** Filtered Timer Input 1 (TI1FP1) – сигнал TI1 (после цифрового фильтра);

**110:** Filtered Timer Input 2 (TI2FP2) – сигнал TI2 (после цифрового фильтра);

**111:** External Trigger input (ETRF) – внешний сигнал ETR.

Биты 2:0: **SMS (Slave mode selection)** – выбор режима ведомого:

**000:** режим ведомого отключён (в этом случае при CEN = 1 на предделитель поступает внутренний тактовый сигнал);

**001:** режим 1 энкодера – счёт вверх/вниз по фронту на TI2FP1: направление счёта определяется уровнем сигнала TI1FP2;

**010:** режим 2 энкодера – счёт вверх/вниз по фронту на TI1FP2 (направление счёта определяется уровнем сигнала TI2FP1);

**011:** режим 3 энкодера – счёт вверх/вниз по фронту на любом из двух входов TI1FP1, TI2FP2: направление счёта при прохождении импульса на одном входе определяется уровнем сигнала в этот момент на другом входе;

**100:** Reset Mode, режим сброса – по фронту сигнала, определяемого битами TS0-TS2, сбрасывается счётчик и обновляются регистры;

**101:** Gated Mode, стробирующий режим – таймер выполняет счёт при высоком уровне сигнала на триггерном входе (TRGI): счётчик останавливается (но не сбрасывается) при переходе сигнала на запускающем входе к низкому уровню. Таким образом, внешний сигнал управляет как запуском, так и остановкой счётчика;

**110:** Trigger Mode, триггерный режим – при поступлении фронта, выбранного битами TS0-TS2 сигнала, счётчик запускается (но не сбрасывается). В отличие от стробирующего режима (Gated Mode) контролируется только запуск счётчика;

**111:** External Clock Mode 1, режим 1 внешней синхронизации. Синхронизация выполняется фронтом выбранного сигнала (TS0-TS2).

**TIMx_DIER Регистр разрешения запросов DMA и прерываний**

Адрес смещения: 0x0C.

Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Res. | TDE | Res | CC4DE | CC3DE | CC2DE | CC1DE | UDE | Res. | TIE | Res | CC4IE | CC3IE | CC2IE | CC1IE | UIE |
| | rw | | rw | rw | rw | rw | rw | | rw | | rw | rw | rw | rw | rw |

Бит 14: **TDE (Trigger DMA request enable)** – Разрешение запроса запуска DMA:

«0» – запрещен;

«1» – разрешен.

Бит 12: **CC4DE (Capture/Compare 4 DMA request enable)** – разрешения запроса DMA по сравнению/захвату канала 4:

«0» – запрещен;

«1» – разрешен.

Бит 11**: CC4DE (Capture/Compare 3 DMA request enable)** – разрешения запроса DMA по сравнению/захвату канала 3:

«0» – запрещен;

«1» – разрешен.

Бит 10**: CC4DE (Capture/Compare 2 DMA request enable)** – разрешения запроса DMA по сравнению/захвату канала 2:

«0» – запрещен;

«1» – разрешен.

Бит 9: **CC4DE (Capture/Compare 1 DMA request enable)** – разрешения запроса DMA по сравнению/захвату канала 1:

«0» – запрещен;

«1» – разрешен.

Бит 8: **UDE (Update DMA request enable)** – разрешение запроса DMA по обновлению счетчика:

«0» – запрещен;

«1» – разрешен.

Бит 6**: TIE (Trigger interrupt enable)** – разрешение прерывания по триггерному сигналу TRGI:

«0» – запрещено;

«1» – разрешено.

Бит 4**: CC4IE (Capture/Compare 4 interrupt enable)** – разрешение прерывания по сравнению/захвату канала 4:

«0» – запрещено;

«1» – разрешено.

Бит 3**: CC3IE (Capture/Compare 3 interrupt enable)** – разрешение прерывания по сравнению/захвату канала 3:

«0» – запрещено;

«1» – разрешено.

Бит 2: **CC2IE (Capture/Compare 2 interrupt enable)** – разрешение прерывания по сравнению/захвату канала 2:

«0» – запрещено;

«1» – разрешено.

Бит 1: **CC1IE (Capture/Compare 1 interrupt enable)** – разрешение прерывания по сравнению/захвату канала 1:

«0» – запрещено;

«1» – разрешено.

Бит 0: **UIE (Update interrupt enable)** – разрешение прерывания по обновлению счетчика:

«0» – запрещено;

«1» – разрешено.

**TIMx_CCMR1 Регистр 1 режима захвата/сравнения:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OC2CE | OC2M[2:0] | | | OC2PE | OC2FE | CC2S[1:0] | | OC1CE | OC1M[2:0] | | | OC1PE | OC1FE | CC1S[1:0] | |
| IC2F[3:0] | | | | IC2PSC[1:0] | | | | IC1F[3:0] | | | | IC1PSC[1:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

**TIMx_CCMR2 Регистр 2 режима захвата/сравнения:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OC4CE | OC4M[2:0] | | | OC4PE | OC4FE | CC4S[1:0] | | OC3CE | OC3M[2:0] | | | OC3PE | OC3FE | CC3S[1:0] | |
| IC4F[3:0] | | | | IC4PSC[1:0] | | | | IC3F[3:0] | | | | IC3PSC[1:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Эти регистры имеют аналогичную структуру и отличаются номерами каналов.

Так как каналы захвата/сравнения используются в режиме с разделением времени, то их функции определяются содержимым поля CCnS.

**CCnS**: Выбор захвата/сравнения **канала n:**

00: Канал CCn настроен на выход (режим сравнения);

01: Канал CCn настроен на вход (режим захвата), IC1 отображается на TI1;

10: Канал CCn настроен на вход (режим захвата), IC1 отображается на TI2;

11: Канал CCn настроен на вход (режим захвата), IC1 отображается на TRC. Этот режим доступен, если в регистре TIM_SMRC установлен бит TS.

Биты OCxx определяют особенности работы в режиме сравнения, а ICxx – в режиме захвата.

**Режим сравнения:**

**OCnCE**: Разрешение внешнего сигнала ETR **канала** n:

«0» – вход ETRF не влияет на OC1Ref;

«1» – OC1Ref очищается, как только обнаружен высокий уровень на входе ETRF.

**OCnM [2:0]**: Выбор режима сравнения:

Эти биты определяют поведение выходного сигнала OC1REF из которого получаются OC1 и OC1N. OCREF активный высокий, тогда как активный уровень OC1 и OC1N зависит от битов CC1P и CC1NP.

000: заморожен – сравнение между выходным регистром TIMx_CCRn и счетчиком TIMx_CNT не влияет на выходы (этот режим используется для генерации временной базы);

001: при сравнении TIMx_CNT и TIMx_CCRn OCnREF устанавливается в «1»;

010: при сравнении TIMx_CNT и TIMx_CCRn OCnREF сбрасывается в «0»;

011: при сравнении TIMx_CNT и TIMx_CCRn OCnREF инвертируется;

100: сброс OCnREF в «0»;

101: установка OCnREF в «1»;

110: режим ШИМ 1 – при TIMx_CNT < TIMx_CCRn неактивное состояние (OCnREF = 0). При TIMx_CNT > TIMx_ CCRn – активное (OCnREF = 1).

111: Режим ШИМ 2 – при TIMx_CNT > TIMx_CCRn – неактивное (OCnREF = 0). При TIMx_CNT < TIMx_CCRn активное состояние (OCnREF = 1).

**Примечание:**

1. Эти биты должны быть сконфигурированы **до установки режима** сравнения (CCnS = 00).

2. В режиме ШИМ 1 и 2 уровень OCnREF изменяется только в момент сравнения или при переключении из состояния «заморожен» в один из режимов ШИМ n.

**OC**n**PE**: Разрешение вносить изменения в регистр сравнения:

«0» – запрещено вносить изменения в регистр сравнения;

«1» – разрешено вносить изменения в регистр сравнения.

**Примечание:**

1. Эти биты должны быть сконфигурированы **до установки режима** сравнения (CCnS = 00).

2. В режиме ШИМ 1 и 2 уровень OCnREF изменяется только в момент сравнения или при переключении из состояния «заморожен» в один из режимов ШИМ.

**OC**n**FE**: Быстрый доступ к выходу канала n:

«0» – обычный доступ. Минимальная задержка от появления фронта на входе запуска и до активации выхода– 5 циклов;

«1» – ускоренный доступ. Задержка уменьшается до 3 циклов. Этот режим действует, если канал сконфигурирован в режиме ШИМ 1 или ШИМ 2.

**Режим захвата**

**ICnF[3:0]** Частота фильтрации ($f_{SAMPLING}$) и кол-во выборок (N):

0000: без фильтрации $f_{SAMPLING} = f_{DTS}$, N=1.
0001: $f_{SAMPLING} = f_{CK\_INT}$, N=2.
0010: $f_{SAMPLING} = f_{CK\_INT}$, N=4.
0011: $f_{SAMPLING} = f_{CK\_INT}$, N=8.
0100: $f_{SAMPLING} = f_{DTS}/2$, N=6.
0101: $f_{SAMPLING} = f_{DTS}/2$, N=8.
0110: $f_{SAMPLING} = f_{DTS}/4$, N=6.
0111: $f_{SAMPLING} = f_{DTS}/4$, N=8.
1000: $f_{SAMPLING} = f_{DTS}/8$, N=6.
1001: $f_{SAMPLING} = f_{DTS}/8$, N=8.
1010: $f_{SAMPLING} = f_{DTS}/16$, N=5.
1011: $f_{SAMPLING} = f_{DTS}/16$, N=6.
1100: $f_{SAMPLING} = f_{DTS}/16$, N=8.
1101: $f_{SAMPLING} = f_{DTS}/32$, N=5.
1110: $f_{SAMPLING} = f_{DTS}/32$, N=6.
1111: $f_{SAMPLING} = f_{DTS}/32$, N=8 ($f_{DTS}$ – см.TIMx_CR1).

ICnPSC[1:0] Выбор предварительного деления частоты ICx:

00: ICnPS=ICx;
01: ICnPS=ICx/2;
10: ICnPS=ICx/4;
11: ICxPS=ICx/8.

**TIMx_CCER** Регистр установки режима работы выходного сигнала каналов захвата/сравнения:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | CC4P | CC4E | Reserved | | CC3P | CC3E | Reserved | | CC2P | CC2E | Reserved | | CC1P | CC1E |
| | | rw | rw | | | rw | rw | | | rw | rw | | | rw | rw |

**CCxnP**: Выбор полярности активного уровня на выходе OCxn:

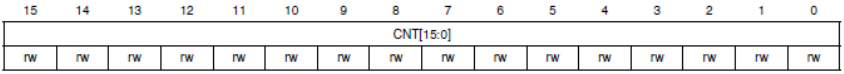
«0» – активный уровень высокий («1»).
«1» – активный уровень низкий («0»).

**CCxnE:** Состояние выхода OCxn:

«0» – Сигнал OC1n пассивен;

«1» – OC1n активен и его состояние при активном OCxREF определяется битами CCxNP – полярность активного сигнала.

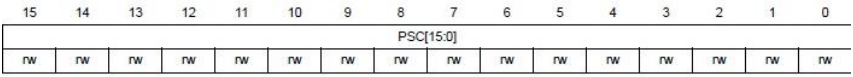
**TIMxCNT** Регистр счетчика/таймера:




**TIMxARR Регистр автоматической перезагрузки ARR**
Адрес смещения: 0x2C.
Состояние после сброса: 0x0000.

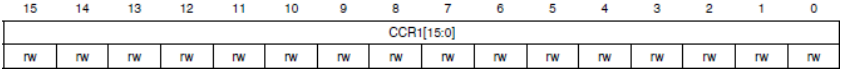

Регистр автоматической перезагрузки счётчика.

**TIMx PSC Регистр предделителя PSC**
Адрес смещения: 0x28.
Состояние после сброса: 0x0000.



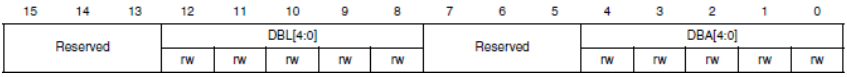

Регистр содержит значение коэффициента деления предделителя.

**TIMxCCRn Регистр сравнения (n – номер канала)**:




В регистре хранится переменная, которая сравнивается с содержимым регистра счётчика таймера TIMx_CNT.

**TIMx_DCR Регистр базового адреса и длины обмена DMA:**




Биты 4:0 **DBA (4:0)** Базовый адрес DMA:

Каждый регистр таймера имеет свой базовый адрес (не путать с «Address offset» – смещение адреса), например:

00000: TIMx CR1;

00001: TIMx CR2,

00010: TIMx SMCR и т.д. (см. документацию выбранного микроконтроллера).

При запросе DMA запись или чтение начнётся с регистра, базовый адрес которого указан в этих битах.

Биты 12:8 **DBL(4:0)** Длина обмена DMA. Указывается количество трансляций, подлежащих передаче/приёму через контроллер DMA ( 00000: 1 трансляция, 00001: 2 трансляции, 00010: 3 трансляции и т.д.) Размер трансляции определяется в настройках канала DMA (1 байт, 2 или 4 байта).

Так, например, для передачи содержимого Tl Mx CR2 и Tl Mx SMCR необходимо указать 0001 (2 трансляции).

**TIMx_DMAR Регистр адреса быстрого доступа DMA:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DMAB[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Адреса регистров, к которым обращается DMA должны лежать в диапазоне адреса, записанного в TIMx → DMAR: TIMx_DMAR = (адрес TlMx_CR1) + TIMx_DBA + TIMx_DBL.

Например, для передачи содержимого TIMx_CR2 и TIMx_SMCR необходимо установить значения: **(адрес TlMx_CR1) = 0**; TIMx_DBA = 1; TIMx_DBL = 1; TIMx_DMAR = 0+1+1 = 2 (0x0002).

## П1.6. Часы реального времени

**RTC _CRH Регистр управления (старший)**

Адрес смещения: 0x00.

Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | OWIE | ALRIE | SECIE |
| | | | | | | | | | | | | | rw | rw | rw |

Бит 0 **SECIE** – разрешение прерывания от секундных импульсов.

Бит 1 **ALRIE** – разрешение прерывания от сигнального регистра (будильника). Этот бит устанавливается аппаратно, когда 32-битный программируемый счетчик достигает порога, установленного в регистре RTC_ALR.

Бит 2 **OWIE** – разрешение прерывания по переполнению счетного регистра.

Биты 15: 3 зарезервированы, принудительно установлены на 0.

**RTC_CRL Регистр управления (младший)**

Адрес смещения: 0x04.

Состояние после сброса: 0x0020.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | RTOFF | CNF | RSF | OWF | ALRF | SECF |
| | | | | | | | | | | r | rw | rc_w0 | rc_w0 | rc_w0 | rc_w0 |

Биты 15÷6 зарезервированы, принудительно установлены в «0».
Первые три разряда – это флаги соответствующих событий:

Бит 0 **SECF** – переполнение входного делителя (секундный импульс). Этот бит устанавливается аппаратно при переполнении 32-битного программируемого предделителя, увеличивая счетчик RTC. Следовательно, этот флаг обеспечивает периодический сигнал с периодом, соответствующим разрешению, запрограммированному для счетчика RTC (обычно одна секунда);

Бит 1 **ALRF** – значение счетного и сигнального регистра равны. Этот бит устанавливается аппаратно, когда 32-битный программируемый счетчик достигает порога, установленного в регистре **RTC_ALR**;

Бит 2 **OWF** – переполнение счетного регистра. Этот бит устанавливается аппаратно при переполнении 32-битного программируемого счетчика;

Бит 3 **RSF –** флаг синхронизации. Этот бит устанавливается аппаратно каждый раз, когда регистры RTC_CNT и RTC_DIV обновляются и очищаются программным обеспечением. Перед любой операцией чтения после сброса APB1 или остановки тактового сигнала APB1 этот бит должен быть очищен программным обеспечением, и пользовательское приложение должно подождать, пока оно не будет установлено, чтобы убедиться, что регистры RTC_CNT, RTC_ALR или RTC_PRL синхронизированы («0» – регистры еще не синхронизированы, «1» – регистры синхронизированы);

Бит 4 **CNF –** конфигурирование регистров RTC_CNT, RTC_ALR или RTC_PRL разрешено. Чтобы позволить записывать новые значения в регистры RTC_CNT, RTC_ALR или RTC_PRL, этот бит должен быть установлен программным обеспечением для входа в режим конфигурации. Операция записи выполняется только тогда, когда бит CNF сбрасывается программным обеспечением после того, как он был установлен («0» – выйти из режима конфигурации (запустить обновление регистров RTC), «1» – войти в режим конфигурации);

Бит 5 **RTOFF –** запись регистров RTC_CNT, RTC_ALR или RTC_PRL завершена. Этот бит только для чтения:

«0» – последняя операция записи в регистры RTC все еще продолжается;

«1» – последняя операция записи в регистры RTC прекращена.

**RTC_PRLH/RTC_PRLL Регистр коэффициента деления предделителя RTC**

Операция записи в регистры разрешается, если значение RTC_CRL.RTOFF равно «1»:

**RTC prescaler load register high (RTC_PRLH)**

Address offset: 0x08
Write only (see *Section 17.3.4 on page 450*)
Reset value: 0x0000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | PRL[19:16] | | | |
| | | | | | | | | | | | | w | w | w | w |

Биты 15: 4 зарезервированы, принудительно установлены в «0».
Биты 3:0 **PRL [19:16]**: старшие биты предделителя RTC.

**RTC prescaler load register low (RTC_PRLL)**

Address offset: 0x0C
Write only (see *Section 17.3.4 on page 450*)
Reset value: 0x8000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PRL[15:0] | | | | | | | | | | | | | | | |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

Биты 15:0 **PRL[15: 0]**: Младшие биты предделителя RTC.

**RTC_DIVH/RTC_DIVL Регистр делителя входной частоты RTC**:

**RTC prescaler divider register high (RTC_DIVH)**

Address offset: 0x10
Reset value: 0x0000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | RTC_DIV[19:16] | | | |
| | | | | | | | | | | | | r | r | r | r |

Биты 15: 4 зарезервированы.

Биты 3:0 **RTC_DIV[19:16]** – старший регистр делителя тактовой частоты RTC.

**RTC prescaler divider register low (RTC_DIVL)**

Address offset: 0x14
Reset value: 0x8000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_DIV[15:0] | | | | | | | | | | | | | | | |
| r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |

Биты 15:0 **RTC_DIV[15:0]** – младший регистр делителя тактовой частоты RTC.

**RTC_CNTH/RTC_CNTL Регистр счетчика RTC:**
Адрес смещения: 0x18.
Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_CNT[31:16] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 15:0 **RTC_CNT[31:16]**: старший регистр счетчика RTC:
Адрес смещения: 0x1C.
Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_CNT[15:0] | | | | | | | | | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 15:0 **RTC_CNT** [15:0]: младший регистр счетчика RTC.

**RTC_ALRH / RTC_ALRL Сигнальный регистр RTC:**

**RTC alarm register high (RTC_ALRH)**

Address offset: 0x20
Write only (see *Section 17.3.4 on page 450*)
Reset value: 0xFFFF

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_ALR[31:16] | | | | | | | | | | | | | | | |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

Биты 15:0 **RTC_ALR [31:16]**: старшие разряды сигнального регистра RTC:

**RTC alarm register low (RTC_ALRL)**

Address offset: 0x24
Write only (see *Section 17.3.4 on page 450*)
Reset value: 0xFFFF

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_ALR[15:0] | | | | | | | | | | | | | | | |
| w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

Биты 15:0 **RTC_ALR[15:0]**: младшие разряды сигнального регистра RTC.

## П1.7. Сторожевой таймер

**Независимый сторожевой таймер IWDG**
**IWDG_KR Регистр ключа:**
Адрес смещения: 0x00.
Состояние после сброса: 0x0000 0000 (сбрасывается в режиме «Standby»).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | KEY[15:0] | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w | w |

Биты 15:0 **KEY[15:00]** Ключ управления:
0x0000AAAA – Сброс (перезагрузка) IWDG;
0x00005555 – Открытие доступа для модификации регистров IWDG PR и IWDG RLR;
0x0000CCCC – Запуск IWDG.

**IWDG_PR Регистр предделителя тактовой частоты**
Адрес смещения: 0x04.
Состояние после сброса: 0x0000 0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | Reserved | | | | | | | | | | | | | | PR[2:0] | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | rw | rw | rw |

Биты 2:0 Значение предделителя:
000: $f_{IWDG} = f_{LSI}$ /4;
001: $f_{IWDG} = f_{LSI}$ /8;
010: $f_{IWDG} = f_{LSI}$ /16;
011: $f_{IWDG} = f_{LSI}$ /32;
100: $f_{IWDG} = f_{LSI}$ /64;
101: $f_{IWDG} = f_{LSI}$ /128;

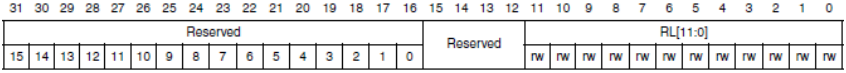
110: $f_{IWDG} = f_{LSI} /256$;

111: $f_{IWDG} = f_{LSI} /256$, где $f_{LSI}$ – частота генератора LSI; $f_{IWDG}$ – частота на выходе предделителя.

**IWDG_RLR Регистр перезагрузки счетчика IWDG**

Адрес смещения: 0x08.

Состояние после сброса: 0x0000 0FFF (сбрасывается в режиме «Standby»).

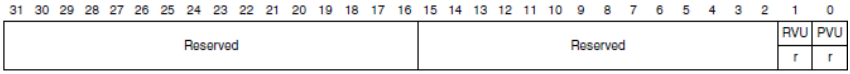



Биты 0:11 **RL[11:0]** – значение, перезагружаемое в счётчик сторожевого таймера при записи в регистр IWDG_KR ключа 0x0000AAAA.

**IWDG_SR Регистр состояния**

Адрес смещения: 0x0C.

Состояние после сброса: 0x0000 0000 (сбрасывается в режиме «Standby»).




Бит 0 **PVU = 1** Коррекции регистра **IWDG_PR** выполнена.

Бит 1 **RVU = 1** Коррекции регистра **IWDG_RLR** выполнена.

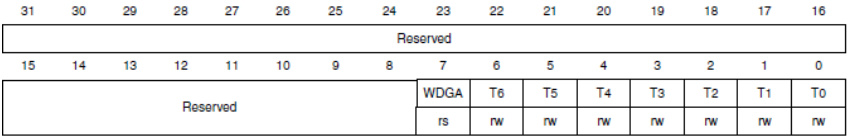
Биты устанавливаются и сбрасываются аппаратно.

**Оконный сторожевой таймер WWDG**

**WWDG_CR Регистр управления.**

Адрес смещения: 0x00.

Состояние после сброса: 0x7F.




Биты 6:0 **(T6:T0)** – исходное значение вычитающего счетчика.

Бит7 **WDGA** – запуск оконного сторожевого таймера.

**WWDG_CFR Регистр конфигурации:**

Адрес смещения: 0x04.

Состояние после сброса: 0x7F.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | EWI | WDG TB1 | WDG TB0 | W6 | W5 | W4 | W3 | W2 | W1 | W0 |
| | | | | | | rs | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 6:0 **W6-W0** – верхняя граница временного окна.
Биты 8:7 **WDGTBl÷WDGTBO** – значение предделителя частоты синхронизации (1, 2, 4, 8).
Бит 9 **EWI = 1** – разрешение прерывание при достижении счётчиком значения 0x40.

**WWDG_SR Регистр состояния:**
Адрес смещения: 0x08.
Состояние после сброса: 0x00.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | | | | | | | | | | EWIF |
| | | | | | | | | | | | | | | | rc_w0 |

EWIF – Флаг прерывания пробуждения процессора. Бит принимает единичное значение аппаратно, когда счётчик достигает **значения 0x40** и прерывание выключено. Очищается программно, записью в него нулевого значения. Запись в этот разряд значения «1» эффекта не даст.

## П1.8. Последовательный порт SPI

**SPI_CR1 Регистр 1 управления SPI (не используется в режиме $I^2S$):**
Адрес смещения: 0x00.
Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIDI MODE | BIDI OE | CRC EN | CRC NEXT | DFF | RX ONLY | SSM | SSI | LSB FIRST | SPE | BR [2:0] | | | MSTR | CPOL | CPHA |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Бит 15 **BIDIMODE.** Разрешение двунаправленного режима работы:
«0» – однонаправленная передача данных по **двум линиям MOSI, MISO**;

«1» – двунаправленная передача данных по **одной** линии.

Бит 14 **BIDIOE.** Направление обмена при двунаправленной работе по одной линии:

«0» – приём данных (выход отключен);

«1» – передача данных (выход включен).

Бит 13 **CRCEN.** Аппаратное вычисление CRC:

«0» – отключено;

«1» – включено.

Бит 12 **CRCNEXT –** указывает, что при следующей передаче будет передаваться CRC:

«0» – этап передачи данных;

«1» – передача из регистра CRC.

**Примечание:** Этот бит должен устанавливаться после последней передачи данных в регистр SPI_DR.

Бит 11 **DFF.** Длина формата данных:

«0» – 8 бит;

«1» – 16 бит.

Бит 10 **RXONLY.** Направление передачи в двухпроводном режиме:

«0» – Full duplex – передача и прием;

«1» – Output disabled – только прием.

Бит 9 **SSM.** Программное управление NSS:

«0» – сигнал с входа NSS (аппаратное управление);

«1» – значение бита SSI (программное управление).

Бит 8 **SSI.** Программный выбор ведомого. Если бит SSM = 1, то значение бита SSI управляет выбором ведомого **вместо сигнала** на внешнем выводе NSS;

Бит 7 **LSBFIRST.** Формат кадра:

«0» – старшим битом вперед MSB;

«1» – младшим битом вперед LSB.

Бит 6 **SPE.** Включение SPI:

«0» – отключен;

«1» – включен.

Биты 5:3 **BR[2:0].** Выбор скорости передачи:

000: $f_{PCLK}/2$;

001: $f_{PCLK}$ /4;

010: $f_{PCLK}$ /8;

011: $f_{PCLK}$ /16;
100: $f_{PCLK}$ /32;
101: $f_{PCLK}$ /64;
110: $f_{PCLK}$ /128;
111: $f_{PCLK}$ /256.

Бит 2 **MSTR.** Выбор режима работы SPI:
«0» – Slave (ведомый);
«1» – Master (ведущий).

Бит 1 **CPOL.** Выбор полярности синхросигнала SCK:
«0» – низкий уровень в режиме ожидания;
«1» – высокий уровень в режиме ожидания.

Бит 0 **CPHA.** Выбор активной фазы SCK:
«0» – выборка данных производится по фронту SCK;
«1» – выборка данных производится по срезу SCK.

**SPI_CR2 Регистр 2 управления SPI.**

Адрес смещения: 0x04.

Состояние после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | TXEIE | RXNEIE | ERRIE | Reserved | | SSOE | TXDMAEN | RXDMAEN |
| | | | | | | | | rw | rw | rw | | | rw | rw | rw |

Бит 7 **TXEIE** (Tx buffer empty interrupt enable) – Разрешение прерывания буфера передачи:
«0» – запретить;
«1» – формировать запрос прерывания при установке флага TXE (буфер передачи свободен).

Бит 6 **RXNEIE** (RX buffer not empty interrupt enable) – Разрешение прерывания буфера приёма:
«0» – запретить;
«1» – формировать запрос прерывания при установке флага RXNE (в буфере находятся данные).

Бит 5 **ERRIE** (Error interrupt enable) – разрешение прерывания при возникновении ошибки (установлены биты CRCERR, OVR, MASTER MODF):
«0» – запретить;
«1» – формировать запрос прерывания.

Бит 2 **SSOE** (SS output enable) – управление выходом SS (NSS) в режиме Master:

«0» – выход SS отключен, допустима работа в Multimaster – системе;

«1» – выход SS задействован, однако нельзя работать в Multimaster-системе. Разрешается использовать вывод NSS в качестве выхода и порт SPI управляет выводом NSS, но при включении порта SPI (SPE = 1) на NSS появляется низкий уровень и сохраняется, пока модуль не будет выключен (SPE = 0), что для большинства случаев не подходит. Поэтому используют GPIO, а на NSS в режиме Master программно выставляют единицу (SSM = 1, SSI = 1).

Бит 1 **TXDMAEN** (Tx buffer DMA enable). Разрешение запроса DMA буфером передачи:

«0» – запретить;

«1» – формировать запрос DMA при установке флага TXE.

Бит 0 **RXDMAEN** (Rx buffer DMA enable). Разрешение запроса DMA буфером приёма:

«0» – запретить;

«1» – формировать запрос DMA при установке флага RXNE.

**SPI_SR Регистр состояния**

Адрес смещения: 0x08.

Состояние после сброса: 0x0002.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | BSY | OVR | MODF | CRC ERR | UDR | CHSIDE | TXE | RXNE |
| | | | | | | | | r | r | r | rc_w0 | r | r | r | r |

Бит 7 **BSY:** Флаг занятости. Устанавливается и сбрасывается аппаратно:

«0» – шина SPI свободна или:

- в режиме Slave в паузах между передачами;
- в режиме Master при двунаправленном приёме по одной линии (в этих режимах для контроля занятости SPI лучше использовать флаги RXEN и TXE).

«1» – шина SPI занята (происходит приём или передача).

Бит 6 **OVR.** Переполнение буфера приема:

«0» – переполнение не произошло;

«1» – произошло переполнение.

Бит 5 **MODF** (Mode fault) – ошибка режима. Устанавливается, если в режиме Master на вход NSS поступает сигнал низкого уровня, и он становится Slave. Сброс выполняется устранением причины ошибки.

Бит 4 **CRCERR:** Ошибка контрольной суммы CRC. Этот флаг устанавливается аппаратно и сбрасывается программно записью нуля:

«0» – принятое значение CRC совпало со значением регистра SPI_RXCRCR;

«1» – принятое значение CRC не совпало со значением регистра SPI_RXCRCR.

Бит 3 **UDR (Underrun flag) [**17**]:** флаг недостаточного заполнения буфера (данные не готовы к передаче):

«0» – неполного заполнения буфера не обнаружено;

«1» – буфер заполнен не полностью. Флаг устанавливается при появлении первых тактов передачи, если при этом программа еще не загрузила никакого значения в SPI_DR. Этот флаг устанавливается аппаратно и сбрасывается программно. Актуален только для режима $I^2S$. Доступен, когда бит I2SMOD регистра SPI_I2SCFGR установлен. Прерывание может быть сгенерировано, если установлен бит ERRIE в регистре SPI_CR2. Очищается чтением регистра SPI_SR.

**Примечание:** этот бит не используется в режиме SPI.

Бит 2 **CHSIDE (Channel side) [**17**]:** флаг канала.

«0» – левый канал должен быть передан или получен;

«1» – правый канал должен быть передан или получен.

**Примечание:** не актуален в режиме SPI и не имеет смысла в режиме PCM.

Бит 1 **TXE.** Состояние буфера передатчика:

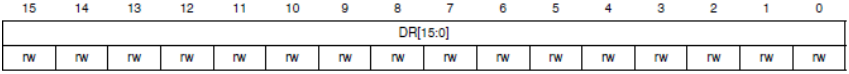
«0» – буфер содержит данные, которые не успели передать;

«1» – буфер пуст.

Бит 0 **RXNE.** Состояние буфера приемника:

«0» – Буфер не содержит принятых данных;
«1» – Данные из буфера еще не прочитаны.

**SPI_DR Регистр данных**
Адрес смещения: 0x0C.
Состояние после сброса: 0x0000.



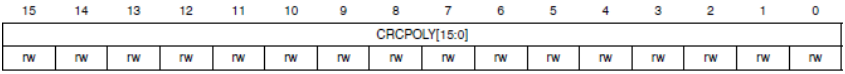

При передаче/приёме 8-битных фреймов младшими битами вперёд используются разряды регистра DR[7:0].

При передаче/приёме 8-битных фреймов старшими битами вперёд используются разряды регистра DR[15:8].

При передаче/приёме 16-битных фреймов используются все разряды регистра DR[15:0].

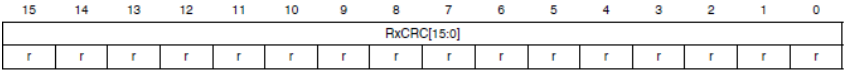
Действия выполняются аппаратно.

**SPI_CRCPR Регистр полинома для расчёта CRC:**
Адрес смещения: 0x10.
Состояние после сброса: 0x0007.



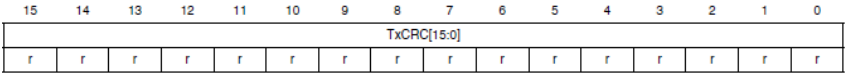

Биты 15:0 **SPI_RXCRCR** Регистр CRC принятых данных.

**SPI_RXCRCR Регистр полинома для расчёта CRC**
Адрес смещения: 0x14.
Состояние после сброса: 0x0000.




Биты 15:0 **SPI_TXCRCR Регистр CRC передаваемых данных**.

**SPI_TXCRCR Регистр полинома для расчёта CRC:**
Адрес смещения: 0x18.
Состояние после сброса: 0x0000.



Биты 15:0 **TXCRCR** Регистр CRC передаваемых данных.

## П1.9. Последовательный порт I2C

**I2C_CR1 Регистр управления1**
Смещение адреса: 0x00.
Значение после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SWRST | Res. | ALERT | PEC | POS | ACK | STOP | START | NO STRETCH | ENGC | ENPEC | ENARP | SMB TYPE | Res. | SMBUS | PE |
| rw | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | | rw | rw |

Бит 15 **SWRST**. Программный сброс:
«0» – состояние работы;
«1» – состояние сброса.
**Примечание:** Этот бит можно использовать в случае, когда стоит бит BUSY и на шине не обнаружено состояние СТОП.
Бит 14 Зарезервирован, аппаратно ставится в «0».
Бит 13 **ALERT**: Сигнал «Внимание» на шине SBus.
Бит 12 **PEC**: Проверка контрольной суммы CRC:
Устанавливается программно, очищается аппаратно после передачи значения CRC, в случае установки битов START, STOP, выключения контроллера $I^2C$ (PE = 0):
«0» – запретить передачу и анализ CRC;
«1» – разрешить передачу и анализ CRC.
Бит 11 **POS**: Позиция Acknowledge/PEC в режиме Master (при приеме) – позиция CRC:
«0» – нет эффекта;
«1» – байт, следующий за принятым, рассматривать как CRC-код.
**Примечание:** В случае выключения PE = 0 сбрасывается в «0» аппаратно.
Бит 10 **ACK**. Разрешение формирования сигнала АСК:
Этот бит устанавливается и сбрасывается программно. Сбрасывается аппаратно, если PE = 0.
«0» – не формировать АСК;
«1» – после приема байта формировать АСК.
Бит 9 **STOP.** Формирование сигнала окончания кадра СТОП:
Бит устанавливается и сбрасывается программно. Аппаратно сбрасывается, если обнаружен бит СТОП, и

устанавливается, если обнаружена ошибка тайм-аута (превышение допустимого времени ожидания).

В режиме ведущего:

«0» – СТОП не формируется;

«1» – СТОП формируется после передачи текущего байта или сигнала СТАРТ.

В режиме ведомого:

«0» – СТОП не формируется;

«1» – освобождает линии SCL и SDA после передачи текущего байта.

Бит 8 **START**. Формирование сигнала СТАРТ:

Бит устанавливается и сбрасывается программно. Аппаратно сбрасывается после формирования СТАРТ или PE = 0.

В режиме ведущего:

«0» – СТАРТ не формируется;

«1» – формирование повторного СТАРТа.

В режиме ведомого:

«0» – СТАРТ не формируется;

«1» – формирование СТАРТ, если шина свободна.

Бит 7 **NOSTRETCH**.

Разрешение задержки формирования сигнала SCL. Если **ведомому** устройству необходимо дополнительное время на обработку данных, оно может приостановить формирование SCL **ведущим,** притянуть линию SCL к земле. Ведомое устройство будет удерживать SCL в нуле до тех пор, пока не будут сброшены биты ADDR или BTF в регистре состояния I2C_SR1.

«0» – задержка разрешена;

«1» – задержка запрещена.

Бит 6 **ENGC**. Разрешение общего вызова:

«0» – общий вызов отключен. В ответ на адрес 0x00 посылается NACK;

«1» – общий вызов разрешен. В ответ на адрес 0x00 посылается ACK.

Бит 5 **ENPEC**. Разрешение формирования контрольной суммы блоком PEC:

«0» – PEC отключен;

«1» – PEC разрешен.

Бит 4 **ENARP**. Разрешение использования резервного адреса устройства ARP:

«0» – запретить ARP;

«1» – разрешить ARP.

Если SMBTYPE = 0, то распознается адрес устройства SMBus. При SMBTYPE = 1 распознается адрес хоста SMBus.

Бит 3 **SMBTYPE**. Тип SMBus устройства:

«0» – ведомый SMBus;

«1» – ведущий SMBus.

Бит 2 Зарезервирован, аппаратно ставится в «0».

Бит 1 **SMBUS**. Режим работы:

«0» – Режим $I^2C$;

«1» – Режим SMBus.

Бит 0 **PE**. Включение порта $I^2C$:

«0» – Выключен;

«1» – Включен.

**I2C_CR2 Регистр управления 2**

Address offset: 0x04.

Значение после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | LAST | DMA EN | ITBUF EN | ITEVT EN | ITERR EN | Reserved | | FREQ[5:0] | | | | | |
| | | | rw | rw | rw | rw | rw | | | rw | rw | rw | rw | rw | rw |

Биты 15÷13 зарезервированы, аппаратно ставятся в «0».

Бит 12 **LAST**: последний обмен DMA. Этот бит используется в **режиме ведущего приемника**, чтобы позволить генерацию NACK по **последнему принятому** байту:

«0» – EOT (сигнал окончания передачи) следующего байта не является концом обмена;

«1» – EOT следующего DMA означает конец обмена.

Бит 11 **DMAEN**. Разрешение запроса DMA:

«0» – запретить запрос DMA;

«1» – разрешить запрос DMA, если TxE = 1 или RxNE = 1.

Бит 10 **ITBUFEN**. Разрешение прерывания от буфера:

«0» – TxE = 1 или RxNE = 1 не генерируют прерывание;

«1» – TxE = 1 и RxNE = 1 генерируют запрос на прерывание при любом состоянии бита DMAEN.

Бит 9 **ITEVTEN**. Разрешение прерывания от события:

«0» – прерывание от события запрещено;

«1» – прерывание от события разрешено.

Это прерывание генерируется в случае одного из событий:

- SB = 1 (Master);
- ADDR = 1 (Master/Slave);
- ADD10 = 1 (Master);
- STOPF = 1 (Slave);
- BTF = 1 без наличия события TxE или RxNE);
- TxE = 1 если ITBUFEN = 1
- RxNE = 1 если ITBUFEN = 1.

Бит 8 **ITERREN**. Разрешение прерывания при возникновении ошибки:

«0» – прерывание от ошибки запрещено;

«1» – прерывание от ошибки разрешено.

Это прерывание генерируется, когда в регистре I2C_SR1 установлен один из флагов:

- BERR = 1;
- ARLO = 1;
- AF = 1;
- OVR = 1;
- PECERR = 1;
- TIMEOUT = 1;
- SMBAlert = 1.

Биты 7:6 зарезервированы, аппаратно ставятся в «0».

Биты 5:0 **FREQ[5:0]**. Частота синхронизации порта $I^2C$:

Допустимый диапазон от 2 до 36 мГц, при внутренней частоте 46 MHz:

000000: значение недопустимо;

000001: значение недопустимо;

000010: 2 MHz;

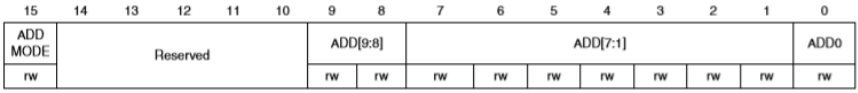
100100: 36 MHz;

Значение выше 100100 недопустимо.

**I2C_OAR1 Регистр основного собственного адреса в режиме ведомого**

Смещение адреса: 0x08.

Значение после сброса: 0x0000.




Бит 15 **ADDMODE**. Режим адресации:

«0» – 7-битный режим адресации;

«1» – 10-битный режим адресации.

Бит 14: при конфигурации всегда ставится в «1».

Биты 13:10 зарезервированы, аппаратно ставятся в «0».

Биты 9:0 **ADD[9:0]**: значение 10-битного адреса.

Биты 7:1 **ADD[7:1]**: значение 7-битного адреса.

**I2C_OAR2 Регистр собственного дублирующего адреса в режиме ведомого**

Смещение адреса: 0x0C.

Значение после сброса: 0x0000.


Биты 15:8 зарезервированы, аппаратно ставятся в «0».

Биты 7:1 **ADD2[7:1]:** 7-битный дублирующий адрес.

Бит 0 **ENDUAL**: разрешение режима двойной адресации:

«0» – только OAR1 в режиме 7-битного адреса;

«1» – используются оба регистра OAR1 и OAR2 в режиме 7-битного адреса.

**I2C_DR Регистр данных**

Смещение адреса: 0x10.

Значение после сброса: 0x0000.


Биты 15:8 зарезервированы, аппаратно ставятся в «0».

Биты 7:0 **DR[7:0]** 8-битный регистр данных.

**I2C_SR1 Регистр состояния 1**

Смещение адреса: 0x14.

Значение после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SMB ALERT | TIME OUT | Res. | PEC ERR | OVR | AF | ARLO | BERR | TxE | RxNE | Res. | STOPF | ADD10 | BTF | ADDR | SB |
| rc_w0 | rc_w0 | | rc_w0 | rc_w0 | rc_w0 | rc_w0 | rc_w0 | r | r | | r | r | r | r | r |

Бит 15 **SMBALERT**: запрос SMBus.

Бит 14 **TIMEOUT**: ошибка превышения времени ожидания шины SMBus.

Бит 13 зарезервирован, аппаратно ставится в «0».

Бит 12 **PECERR**. Ошибка PEC при приеме:

«0» – нет ошибки (приемник формирует ACK после приема PEC);

«1» – ошибка (приемник формирует NACK после приема PEC).

Сбрасывается программно (записью нуля) или аппаратно (когда PE = 0).

Бит 11 **OVR**. Флаг переполнения данных в режиме ведомого. Происходит при несвоевременном записи или чтении регистра данных:

«0» – нет переполнения;

«1» – переполнение.

Сбрасывается программно (записью нуля), или аппаратно (PE = 0).

Бит 10 **AF**. Отсутствие подтверждения ACK:

«0» – нормальное подтверждение (ACK);

«1» – отсутствие подтверждения (NACK).

Устанавливается аппаратно, сбрасывается программно (записью нуля), или аппаратно (PE = 0).

Бит 9 **ARLO**. Флаг потери режима ведущего при арбитраже:

«0» – режим ведущего сохранен;

«1» – потеря прав на шину. Ведущим становится другое устройство.

Устанавливается аппаратно, сбрасывается программно (записью нуля) или аппаратно (когда PE = 0). После события ARLO порт автоматически переключается в режим ведомого (M/SL = 0).

Бит 8 **BERR**. Ошибка шины. Устанавливается в случае возникновения сигнала **START** или **STOP** в неположенный момент:

«0» – нет ошибки;

«1» – обнаружена ошибка.

Сбрасывается программно (записью нуля) или аппаратно (когда PE = 0).

Бит 7 **TxE**. Флаг передачи данных:

«0» – регистр данных заполнен;

«1» – регистр данных пуст и готов к записи очередного байта данных.

Сбрасывается программно (записью нуля) или аппаратно (после состояния СТАРТ, СТОП или при PE = 0). TxE не устанавливается, если принят NACK или следующий передаваемый байт является PEC (PEC = 1).

Бит 6 **RxNE**. Флаг приема данных:

«0» – нет события;

«1» – принятый байт данных находится в регистре **I2C_DR**.

RxNE не ставится на фазе адресации и при наличии события ARLO. Очищается программно, чтением/записью в регистр DR или аппаратно, если PE = 0.

Бит 5 зарезервирован, аппаратно ставится в «0».

Бит 4 **STOPF.** Флаг получения сигнала СТОП в режиме ведомого:

«0» – сигнал СТОП не обнаружен;

«1» – сигнал СТОП обнаружен.

Устанавливается аппаратно, сбрасывается программно чтением регистра I2C_SR1 с последующей записью в регистр I2C_CR1.

Бит 3 **ADD10**. Флаг передачи старшего байта 10-битного адреса (в режиме ведущего):

«0» – событие передачи старшего байта не обнаружено;

«1» – передача старшего байта выполнена и получен АСК.

Устанавливается аппаратно, сбрасывается программно чтением регистра I2C_SR1 с последующей записью в регистр I2C_DR второго байта адреса.

Бит 2 **BTF**. Флаг окончания приема/передачи байта:

«0» – прием/передача данных не завершена;

«1» – байт данных принят/передан успешно и получен сигнал АСК.

Устанавливается аппаратно, когда NOSTRETCH = 0.

При приеме флаг устанавливается, когда принят новый байт, а I2C_DR еще не прочитан (RxNE = 1).

При передаче флаг устанавливается, когда надо передавать новый байт, а в I2C_DR он еще не записан (TxE = 1).

Сбрасывается программно чтением регистра I2C_SR1 с

последующим чтением/записью в регистр I2C_DR или аппаратно (после состояния СТАРТ или СТОП при передаче или при PE = 0).

**Примечание.** Бит BTF не устанавливается после получения NACK или, если следующий передаваемый байт является PEC (TRA = 1 в регистре I2C_SR2 и PEC = 1 в регистре I2C_CR1).

Бит 1 **ADDR**. Флаг посылки/приема адреса.

**В режиме ведущего:**

«0» – нет события;

«1» – конец передачи адреса.

При 10-битной адресации бит устанавливается после получения ACK для второго байта.

При 7-битной адресации бит устанавливается после получения ACK.

Бит ADDR не ставится после получения NACK.

**В режиме ведомого:**

«0» – адрес не передавался или он не совпадает с адресом ведомого;

«1» – полученный адрес корректен.

Устанавливается аппаратно, если передаваемый адрес равен содержимому I2C_OAR1, является адресом общего вызова или опознан один из адресов SMBus.

Бит сбрасывается программно чтением регистра I2C_SR1 с последующим чтением регистра I2C_SR2 или аппаратно, если PE = 0.

Бит 0 **SB**: флаг СТАРТ (используется в режиме ведущего):

«0» – СТАРТ не сформирован;

«1» – сформирован сигнал СТАРТ.

Сбрасывается программно. Для сброса необходимо прочитать регистр I2C_SR1 и затем записать значение в регистр I2C_DR. Аппаратный сброс выполняется при PE = 0.

**I2C_SR2 Регистр состояния 2**

Смещение адреса: 0x18.

Значение после сброса: 0x0000.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PEC[7:0] | | | | | | | | DUALF | SMB HOST | SMBDE FAULT | GEN CALL | Res. | TRA | BUSY | MSL |
| r | r | r | r | r | r | r | r | r | r | r | r | | r | r | r |

Биты 15:8 **PEC[7:0]** Регистр контрольной суммы кадра. Содержит формируемый код CRC, если I2C_CR1.ENPEC = 1.

Бит 7 **DUALF**. Флаг дублирующего адреса (в режиме ведомого):

«0» – полученный адрес соответствует I2C_OAR1;

«1» – полученный адрес соответствует I2C_OAR2.

Сбрасывается аппаратно после состояния СТОП, повторного состояния СТАРТ или когда PE = 0.

Бит 6 **SMBHOST**: флаг управления SMBus.

Бит 5 **SMBDEFAULT**: флаг обращения к ведомому устройству на шине SMBus.

Бит 4 **GENCALL**: принят адрес общего вызова на шине SMBus.

Бит 3 зарезервирован, аппаратно ставится в «0».

Бит 2 **TRA**: флаг передатчика/приемника:

«0» – прием данных;

«1» – передача данных.

Состояние флага зависит от направления передачи данных, которое определяется значением бита R/W, передаваемым вместе с адресом.

Сбрасывается аппаратно после состояния СТОП при СТАРТ, потери арбитража (ARLO = 1) или PE = 0.

Бит 1 **BUSY**. Флаг занятости шины:

«0» – шина свободна;

«1» – шина в состоянии обмена.

Устанавливается аппаратно при фиксировании низкого уровня на SDA или SCL. Сбрасывается аппаратно после сигнала СТОП.

Бит 0 **MSL**. Флаг режима работы:

«0» – режим ведомого;

«1» – режим ведущего.

Устанавливается аппаратно при переходе в режим ведущего (I2C_SR1.SB = 1). Сбрасывается аппаратно после состояния СТОП, потери прав на шину (I2C_SR1.ARLO = 1) или при PE = 0.

**I2C_CCR Регистр управления тактовым сигналом**

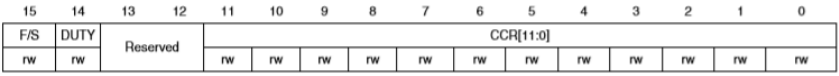
Смещение адреса: 0x1C.

Значение после сброса: 0x0000.

**Примечание.**

1. $f_{PCLK1}$ кратно 10 МГц, чтобы возможно было генерировать частоту 400 kHz в быстром режиме.

2. Регистр CCR можно менять только тогда, когда $I^2C$ отключена (PE = 0).



| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F/S | DUTY | Reserved | | CCR[11:0] | | | | | | | | | | | |
| rw | rw | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Бит 15 **F/S**. Выбор скорости для режима ведущего порта $I^2C$:

«0» – стандартный режим «**Standart mode»**;

«1» – быстрый режим **«Fast mode».**

Бит 14 **DUTY**. Скважность в быстром режиме:

«0» – Fast Mode $T_{Low}$ / $T_{High}$ = 2;

«1» – Fast Mode $T_{Low}$ / $T_{High}$ = 16/9.

Биты 13:12 зарезервированы, аппаратно ставятся в «0».

Биты 11:0 **CCR[11:0]**. Управление тактовым сигналом SCL:

Стандартный режим или SMBus: $T_{High}$ = CCR * $T_{PCLK1}$; $T_{Low}$ = = CCR * $T_{PCLK1}$.

Быстрый режим:

Если DUTY = 0: $T_{High}$ = CCR * $T_{PCLK1}$; $T_{Low}$ = CCR * $T_{PCLK1}$ * 2.

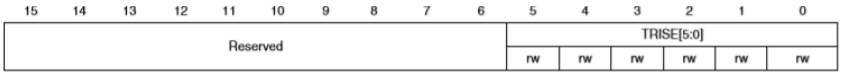
Если DUTY = 1 (для частоты 400 kHz): $T_{High}$ = 9 * CCR * $T_{PCLK1}$; $T_{Low}$ = 16 * CCR * $T_{PCLK1}$.

Содержимое регистра CCR можно изменять, если порт $I^2C$ отключен (PE = 0).

**I2C_TRISE Регистр допустимой длительности фронта**

Смещение адреса: 0x20.

Значение после сброса: 0x0002.



| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | TRISE[5:0] | | | | | |
| | | | | | | | | | | rw | rw | rw | rw | rw | rw |

Биты 15:6 Зарезервированы, аппаратно ставятся в «0».

Биты 5:0 **TRISE[5:0]**. Максимальное время фронта в режимах Fast/Standard (в режиме ведущего): **T = TrMAX / TPCLK1 + 1**.

TrMAX для режима «Standard» составляет 1000 наносекунд, а для режима «Fast» – 300.

Содержимое TRISE[5:0] можно изменять, если порт $I^2C$ отключен (PE = 0).

## П1.10. Последовательный порт USART

**USART_SR Регистр состояния**

Смещение адреса: 0x00.
Значение после сброса: 0x00C0.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | CTS | LBD | TXE | TC | RXNE | IDLE | ORE | NE | FE | PE |
| Res. | | | | | | rc_w0 | rc_w0 | r | rc_w0 | rc_w0 | r | r | r | r | r |

Биты 31:10 зарезервированы.
Бит 9 **CTS**. CTS flag – флаг изменения состояния CTS.

Устанавливается аппаратно, когда происходит переключение сигнала на входе CTS. Если установлен бит CTSIE (USART_CR3.CTSIE = 1), то при установке флага генерируется прерывание. Флаг сбрасывается программно записью «0».
Бит 8 **LBD**. LIN break detection flag – флаг приёма посылки Break **интерфейса LIN**.

Устанавливается аппаратно при обнаружении посылки Break на входе. Если установлен бит LBDIE (USART_CR3.LBDIE = 1), то генерируется прерывание. Флаг сбрасывается программно записью «0».
Бит 7 **TXE**. Transmit data register empty – флаг устанавливается аппаратно, когда содержимое регистра данных TDR пересылается в регистр сдвига (доступ к TDR осуществляется путём записи в регистр USART_DR). Если установлен бит TXEIE (USART_CR1.TXEIE = 1), генерируется прерывание. Флаг сбрасывается путём записи в регистр USART_DR.

Бит 6 **TC**. Transmission complete – флаг завершения передачи.

Устанавливается аппаратно, если передача кадра завершена, и флаг TXE установлен (т.е. регистр передаваемых данных пуст, больше нет данных для передачи). Если USART_CR1.TCIE = 1, то при установке флага генерируется прерывание.

Флаг сбрасывается программно последовательностью действий: чтение регистра USART_SR, затем запись в USART_DR. Также бит может быть сброшен записью в него «0».

5 бит **RXNE**. Read data register not empty – регистр данных для чтения не пуст.

Флаг устанавливается аппаратно, когда содержимое сдвигового регистра приемника передается в регистр данных RDR. Если USART_CR1.RXNEIE = 1, при этом генерируется прерывание.

Флаг сбрасывается чтением из регистра USART_DR или записью в него «0».

Бит 4 **IDLE**. IDLE line detected – линия свободна.

Флаг устанавливается аппаратно, если обнаружена свободная линия. Признаком свободной линии является кадр, состоящий из единиц. При этом генерируется прерывание, если USART_CR1.IDLEIE = 1.

Флаг сбрасывается программно последовательностью действий: чтение регистра USART_SR с последующим чтением из регистра USART_DR.

**Замечание.**

IDLE бит не будет установлен снова до тех пор, пока не произойдёт установка флага RXNE (т.е. следующее освобождение линии будет обнаружено только после приёма данных).

Бит 3 **ORE**. Overrun error – ошибка переполнения.

Устанавливается, если данные в сдвиговом регистре приемника данные готовы к передаче, но из регистра RDR не прочитан предыдущий результат (RXNE = 1). Если USART_CR1.RXNEIE = 1, то при установке флага генерируется исключение.

Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR с последующим чтением из USART_DR.

Бит 2 **NE.** Noise error flag – флаг обнаружения шумов.

Устанавливается аппаратно при обнаружении шума в кадре.

Сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение из регистра USART_DR.

Бит 1 **FE**. Framing error – ошибка кадра.

Флаг устанавливается аппаратно в случае нарушения синхронизации, чрезмерного шума в линии, при обнаружении символа Break.

Сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение из регистра USART_DR.

Бит 0 **PE**. Parity error – ошибка чётности.

Флаг устанавливается аппаратно, когда в принятом кадре обнаружена ошибка чётности (если контроль чётности включён). Если USART_CR1.PEIE = 1, то генерируется прерывание.

Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение либо запись регистра USART_DR. Перед сбросом флага программа должна дождаться установки флага RXNE (регистр данных для чтения не пуст).

**USART_DR Регистр данных:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | | DR[8:0] | | | | | | | | |
| Res. | | | | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 8:0 **DR**. Data value – виртуальный регистр данных. При записи в **DR** информация будет передана в регистр данных передатчика TDR, а при чтении – получена из регистра данных приемника RDR.

Если используется контроль четности (USART_CR1.PCE = 1), то при записи в DR значение старшего бита будет игнорироваться, так как при передаче он будет заменен битом четности. При приеме с включенным контролем четности старший бит будет содержать бит четности.

**USART_BRR Регистр скорости**

Смещение адреса: 0x08.

Значение после сброса: 0x0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DIV_Mantissa[11:0] | | | | | | | | | | | | DIV_Fraction[3:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 15:0 **DIV_Mantissa [11:0]:** целая часть коэффициента деления делителя частоты.

Биты 4:0 **DIV_Fraction [3:0]:** дробная часть коэффициента деления.

**USART_CR1 Регистр управления 1**

Смещение адреса: 0x0C.

Значение после сброса: 0x0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | UE | M | WAKE | PCE | PS | PEIE | TXEIE | TCIE | RXNEIE | IDLEIE | TE | RE | RWU | SBK |
| Res. | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Бит 13 **UE**. USART enable – включение USART:

«0» – предделители USART и его выходы отключены;

«1» – USART включен.

Бит 12 **M**. Word length – длина поля данных:

«0» – 1 старт-бит, 8 бит данных, n стоп-битов (количество стоп-битов программируется с помощью USART_CR2.STOP);

«1» – 1 старт-бит, 9 бит данных, n стоп-битов.

**Замечание:** бит чётности считается битом данных.

Бит 11 **WAKE**. Wakeup method – метод пробуждения USART:

«0» – пробуждение по освобождению линии (Idle line);

«1» – пробуждение по совпадению адресной метки.

Бит 10 **PCE**. Parity control enable – включение аппаратного контроля чётности. Бит чётности помещается в позицию старшего бита данных (9-й или 8-й бит в зависимости от выбранной длины слова):

«0» – контроль чётности отключён;

«1» – контроль чётности включён.

Бит 9 **PS**. Parity selection – тип контроля чётности:

«0» – контроль чётности;

«1» – контроль нечётности.

Бит 8 **PEIE**. PE interrupt enable – разрешение прерывания при ошибке контроля четности:

«0» – прерывание запрещено;

«1» – генерируется прерывание, когда USART_SR.PE = 1.

Бит 7 **TXEIE**. TXE interrupt enable – разрешение прерывания от TXE.

«0» – прерывание запрещено;

«1» – генерируется прерывание от USART, когда USART_SR.TXE = 1.

Бит 6 **TCIE**. Transmission complete interrupt enable – разрешение прерывания после завершения передачи:

«0» – прерывание запрещено;

«1» – генерируется прерывание от USART, когда USART_SR.TC = 1.

Бит 5 **RXNEIE**. RXNE interrupt enable – разрешение прерывания от RXNE:

«0» – прерывание запрещено;

«1» – генерируется прерывание от USART, когда USART_SR.ORE = 1 или USART_SR.RXNE = 1.

Бит 4 **IDLEIE**. IDLE interrupt enable – разрешение прерывания при обнаружении свободной линии (Idle line):

«0» – прерывание запрещено;

«1» – генерируется прерывание от USART, когда USART_SR.IDLE = 1.

Бит 3 **TE**. Transmitter enable – включение передатчика USART (TE = 1).

Бит 2 **RE**. Receiver enable – включение приёмника USART (RE = 1).

Бит 1 **RWU**. Receiver wakeup – выбор режима работы приемника USART:

«0» – активный режим;

«1» – спящий режим.

Устанавливается и сбрасывается программно. Может сбрасываться аппаратно при обнаружении пробуждающей (стартовой) последовательности.

Бит 0 **SBK**. Send break – отправить Break посылку LIN.

**USART_CR2. Регистр управления 2:**

Смещение адреса: 0x10.

Значение после сброса: 0x0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Res. | LINEN | STOP[1:0] | | CLK EN | CPOL | CPHA | LBCL | Res. | LBDIE | LBDL | Res. | ADD[3:0] | | | |
| | rw | rw | rw | rw | rw | rw | rw | | rw | rw | rw | rw | rw | rw | rw |

Бит 14 **LINEN**: LIN mode enable – включение режим LIN.

Биты 13:12 **STOP**[1:0]. STOP биты – количество стоповых битов:

00: 1 стоп-бит;

01: 0.5 стоп-бита;

10: 2 стоп-бита;

11: 1.5 стоп-бита.

Бит 11 **CLKEN**. Clock enable – включение выхода SCLK в синхронном режиме:

«0» – вывод отключен;

«1» – вывод включен.

Бит 10 **CPOL**. Clock polarity – выбор полярности тактового сигнала SCLK в синхронном режиме. Этот бит используется совместно с CPHA битом:

«0» – «0» в отсутствие передачи данных;

«1» – «1» в отсутствие передачи данных.

Бит 9 **CPHA**. Clock phase – выбор фазы тактового сигнала SCLK в синхронном режиме:

«0» – передача данных происходит по **первому** перепаду тактового сигнала;

«1» – передача данных происходит по **второму** перепаду тактового сигнала.

Бит 8 **LBCL**. Last Бит clock pulse – разрешение тактового импульса SCLK для последнего бита передаваемых данных в синхронном режиме:

«0» – тактовый импульс не формируется;

«1» – тактовый импульс формируется.

Бит 6 **LBDIE**. LIN break detection interrupt enable – разрешение прерывания при обнаружении Break посылки LIN.

Бит 5 **LBDL**. LIN break detection length – длина детектируемой Break посылки LIN.

Биты 3:0 **ADD** [3:0]. Address of the USART node – адрес модуля USART в мультипроцессорном режиме. Используется для пробуждения приемника по совпадению адреса в спящем режиме.

**USART_CR3 Регистр управления 3**

Смещение адреса: 0x14.

Значение после сброса: 0x0000.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | CTSIE | CTSE | RTSE | DMAT | DMAR | SCEN | NACK | HDSEL | IRLP | IREN | EIE |
| Res. | | | | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Бит 10 **CTSIE**. CTS interrupt enable – разрешение прерывания от CTS:

«0» – прерывание запрещено;

«1» – прерывание разрешено при USART_SR.CTS = 1.

Бит 9 **CTSE**. CTS enable – включение режима аппаратного управления передачей данных:

«0» – отключить режим аппаратного управления;

«1» – включить режим аппаратного управления.

Данные будут передаваться при nCTS = 0.

Бит 8 **RTSE**. RTS enable – включение аппаратного управления сигналом RTS:

«0» – выход RTS отключен;

«1» – RTS включен, включено соответствующее прерывание. Если данные могут быть приняты, то на выходе RTS должен быть «0». Этот бит недоступен для UART4 и UART5.

Бит 7 **DMAT.** Включение DMA передатчика:

«0» – выключен;

«1» – включен.

Бит 6 **DMAR.** Включение DMA приемника:

«0» – выключен;

«1» – включен.

Бит 5 **SCEN**. Smartcard mode enable – включение режима смарт-карт (установкой бита в «1»).

Бит 4 **NACK**. Smartcard NACK enable – включение в режиме смарт-карт передачу NACK в случае ошибки чётности (включается установкой бита в «1»).

Бит 3 **HDSEL**. Half-duplex selection – включение полудуплексного режима:

«0» – выключен;

«1» – включен.

Бит 2 **IRLP**. IrDA low-power – выбор между нормальным режимом и режимом низкой мощности IrDA:

«0» – нормальный режим;

«1» – режим низкой мощности.

Бит 1 **IREN**. IrDA mode enable – включить режим IrDA (включается установкой бита в «1»).

Бит 0 **EIE**. Error interrupt enable – разрешения прерывания по возникновению ошибки:

«0» – прерывания запрещены;

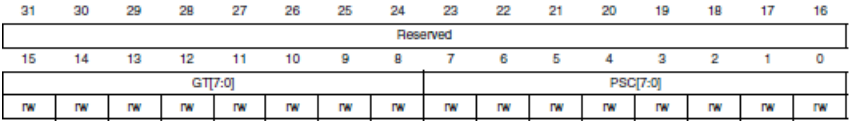
«1» – прерывание генерируется при USART_CR3.DMAR = 1, а в регистре USART_SR хотя бы один из флагов FE, ORE или NF установлен в «1».

**USART_GTPR. Регистр контрольного времени и предварительного делителя частоты:**

Используется в режимах IrDA и SmartCard. Этот регистр недоступен в устройствах UART4 и UART5.

Смещение адреса: 0x18.

Значение после сброса: 0x0000.




Биты 15:18 **GT [7:0**]: Guard time value – защитный интервал времени. Это битовое поле используется в режиме смарт-карт и задаёт время задержки перед установкой флага «Transmission Complete». Величина задаётся в количестве тактов с частотой, равной скорости передачи данных.

Биты 7:0 **PSC [7:0]:** Prescaler value – значение для предварительного делителя частоты. Используется в режимах IrDA и смарт-карт. Восьмибитовое значение определяет коэффициент деления для предварительного делителя частоты, который делит частоту системного тактового сигнала. Значение «0» зарезервировано и не должно использоваться.

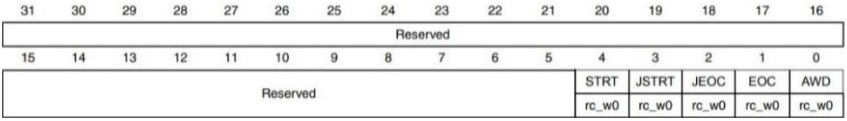
- **Значение в обычном режиме IrDA:** значения больше 1 используются в режимах с малым потреблением.
- **В режиме смарт-карт:** используется для обеспечения требуемой тактовой частоты. В этом режиме делитель определяется 5 младшими битами поля, значение делителя получается умножением поля на 2 (значение поля 1 соответствует делителю 2, 2 соответствует делителю 4 и т.д).

## П1.11. Аналого-цифровой преобразователь

**ADC_SR Регистр состояния АЦП**

Смещение адреса: 0x00.

Значение после сброса: 0x0000 0000.




Биты 31:5 зарезервированы.

Бит 4 **STRT.** Начало преобразования для регулярных каналов:

«0» – регулярное преобразование не запускалось;

«1» – Регулярное преобразование было запущено.

Бит устанавливается аппаратно, когда запускается преобразование из регулярной группы. Сбрасывается программно.

Бит 3 **JSTRT.** Начало преобразования для инжектированных каналов:

«0» – инжектируемое преобразование не запускалось;

«1» – инжектируемое преобразование было запущено.

Бит устанавливается аппаратно, когда происходит запуск преобразования из инжектируемой группы. Сбрасывается программно.

Бит 2 **JEOC.** Конец преобразования инжектированной группы:

«0» – преобразование не завершено;

«1» – преобразование завершено.

Бит устанавливается аппаратно после завершения всех преобразований из инжектируемой группы в сканирующем режиме или после каждого инжектируемого преобразования в других режимах. Если этот бит устанавливается, то EOC тоже устанавливается. Сбрасывается программно.

Бит 1 **EOC.** Конец преобразования:

«0» – преобразование не завершено;

«1» – преобразование завершено.

Бит устанавливается аппаратно после завершения всех преобразований в группе (регулярной или инжектируемой) в сканирующем режиме или после каждого преобразования в других режимах. Сбрасывается программно записью 0 в бит или чтением регистра ADC_DR.

Бит 0 **AWD (**Analog watchdog flag). Оконный компаратор:

«0» – не происходило срабатывания сторожевой схемы;

«1» – произошло срабатывание сторожевой схемы.

Бит устанавливается аппаратно, если результат преобразования на одном из контролируемых каналов выходит за пределы

значений, заданных в регистрах ADC_LTR и ADC_HTR. Сбрасывается программно.

**ADC_CR1 Регистр управления 1**

Смещение адреса: 0x04

Значение после сброса: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | AWDEN | JAWDEN | Reserved | | DUALMOD[3:0] | | | |
| | | | | | | | | rw | rw | | | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DISCNUM[2:0] | | | JDISCEN | DISCEN | JAUTO | AWDSGL | SCAN | JEOCIE | AWDIE | EOCIE | AWDCH[4:0] | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:24 зарезервированы.

Бит 23 **AWDEN**. Подключение оконного компаратора к регулярным каналам:

«0» – компаратор отключен от каналов регулярной группы.

«1» – компаратор подключен к каналам регулярной группы.

Бит устанавливается и сбрасывается программно.

Бит 22 **JAWDEN**. Подключение оконного компаратора к инжектированным каналам:

«0» – компаратор отключен от каналов инжектированной группы;

«1» – компаратор подключен к каналам инжектированной группы.

Бит устанавливается и сбрасывается программно.

Биты 21:20 зарезервированы.

Биты 19:16 **DUALMOD[3:0]**: определяют режим совместной работы двух АЦП (в **STM32F103C8T6 используется один АЦП**).

Биты 15:13 **DISCNUM[2:0]**. Количество каналов регулярного преобразования, которые будут выполняться в прерывистом режиме.

000: 1 канал;

001: 2 канала;

...

111: 8 каналов.

Бит 12 **JDISCEN**. Прерывистый режим преобразования инжектированных каналов:

«0» – прерывистый режим отключён;

«1» – прерывистый режим включён.

Бит устанавливается и сбрасывается программно для включения и отключения прерывистого режима преобразования в группе инжектированных каналов.

Бит 11 **DISCEN**. Прерывистый режим преобразования регулярных каналов Бит устанавливается и сбрасывается программно для включения и отключения прерывистого режима преобразования в группе регулярных каналов:

«0» – прерывистый режим отключён;

«1» – прерывистый режим включён.

Бит 10 **JAUTO**. Непрерывное преобразование инжектированных каналов:

«0» – режим непрерывного преобразования отключён;

«1» – режим непрерывного преобразования включён.

Бит устанавливается и сбрасывается программно для включения и отключения режима непрерывного преобразования.

Бит 9 **AWDSGL.** Режим сканирования оконного компаратора:

«0» – контролирует все каналы;

«1» – контролирует только один канал, заданный битами **AWDCH[4:0]**.

Бит 8 **SCAN.** Разрешение сканирования каналов, заданных регистрами ADC_SQRx, ADC_JSQRx:

«0» – режим сканирования запрещен;

«1» – режим сканирования разрешен.

Бит устанавливается и сбрасывается программно для включения и отключения режима сканирования.

**Примечание.** Установка флага завершения преобразования EOC или JEOC произойдёт только после выполнения последнего преобразования в группе.

Бит 7 **JEOCIE.** Разрешение прерывания по окончании инжектированного преобразования:

«0» – запретить прерывание при установке флага JEOC;

«1» – разрешить прерывание при установке флага JEOC.

Бит устанавливается и сбрасывается программно.

Бит 6 **AWDIE.** Разрешение прерывания по сигналу оконного компаратора:

«0» – запретить прерывание;

«1» – разрешить прерывание.

Бит устанавливается и сбрасывается программно.

Бит 5 **EOCIE.** Разрешение прерывания по окончании преобразования:

«0» – запретить прерывание при установке флага EOC;

«1» – разрешить прерывание при установке флага EOC.

Бит устанавливается и сбрасывается программно.

Биты 4:0 **AWDCH[4:0]**: выбор канала для оконного компаратора (при **AWDSGL = 1**). Битовое поле содержит номер канала, контролируемого компаратором:

00000: канал №0;

00001: канал №1;

…

01111: канал №15;

10000: канал №16;

10001: канал №17;

В STM32F103C8T6 доступны 10 каналов. Каналы 16, 17 подключены к датчику температуры микроконтроллера и встроенному источнику опорного напряжения.

**ADC_CR2 Регистр управления 2**

Смещение адреса: 0x08

Значение после сброса: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | TSVREFE | SWSTART | JSWSTART | EXTTRIG | EXTSEL[2:0] | | | Res. |
| | | | | | | | | rw | rw | rw | rw | rw | rw | rw | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| JEXTTRIG | JEXTSEL[2:0] | | | ALIGN | Reserved | | DMA | Reserved | | | | RSTCAL | CAL | CONT | ADON |
| rw | rw | rw | rw | rw | Res. | | rw | | | | | rw | rw | rw | rw |

Биты 31:24 зарезервированы.

Бит 23 **TSVREFE**. Включение измерения температуры и источника опорного напряжения:

«0» – датчик температуры и источник опорного напряжения отключены;

«1» – датчик температуры и источник опорного напряжения включены.

Бит устанавливается и сбрасывается программно.

Бит 22 **SWSTART**. Запуск преобразования регулярных каналов:

«0» – сброшенное состояние;

«1» – запуск регулярного преобразования.

Бит устанавливается программно для запуска преобразования и сбрасывается программно либо аппаратно сразу же после старта преобразования. Этот способ запуска работает только в том случае, если в качестве внешнего сигнала запуска EXTSEL[2:0] выбран вариант запуска SWSTART.

Бит 21 **JSWSTART**. Запуск преобразования инжектированных каналов:

«0» – сброшенное состояние;

«1» – запуск инжектированного преобразования.

Бит устанавливается программно для запуска преобразования и сбрасывается программно либо аппаратно сразу же после старта преобразования. Этот способ запуска работает только в том случае, если в качестве внешнего сигнала запуска JEXTSEL[2:0] выбран вариант запуска JSWSTART.

Бит 20 **EXTTRIG**. Запуск регулярных каналов преобразования внешним сигналами:

«0» – запуск запрещен;

«1» – запуск разрешён.

Бит устанавливается и сбрасывается программно.

Биты 19:17 **EXTSEL[2:0]**. Выбор источника сигнала внешнего запуска регулярных каналов:

000: Timer 1 CC1 event (совпадение в канале 1);

001: Timer 1 CC2 event (совпадение в канале 2);

010: Timer 1 CC3 event (совпадение в канале 3);

011: Timer 2 CC2 event (совпадение в канале 2);

100: Timer 3 TRGO event (триггер TRGO);

101: Timer 4 CC4 event (совпадение в канале 4);

110: EXTI line 11 (внешний запрос прерывания);

111: SWSTART (разрешение программного запуска регулярных каналов).

Бит 16 Зарезервирован.

Бит 15 **JEXTTRIG**. Разрешение запуска преобразования инжектированных каналов внешним сигналом:

«0» – запретить запуск внешним сигналом;

«1» – разрешить запуск внешним сигналом.

Бит устанавливается и сбрасывается программно.

Биты 14:12 **JEXTSEL[2:0]**. Выбор внешнего источника запуска инжектированных каналов:

000; Timer 1 TRGO event (триггер TRGO);
001: Timer 1 CC4 event (совпадение в канале 4);
010: Timer 2 TRGO event (триггер TRGO);
011: Timer 2 CC1 event (совпадение в канале 1);
100: Timer 3 CC4 event (совпадение в канале 4);
101: Timer 4 TRGO event (триггер TRGO);
110: EXTI line15 (запрос внешнего прерывания);
111: JSWSTART (программный запуск инжектированных каналов).

Бит 11 **– ALIGN**. Выравнивание данных:
«0» – выравнивание вправо;
«1» – выравнивание влево.
Бит устанавливается и сбрасывается программно.

Биты 10:9 Зарезервированы.

Бит 8 **DMA**. Разрешение работы DMA:
«0» – режим DMA запрещен;
«1» – режим DMA разрешен.
Бит устанавливается и сбрасывается программно.

Биты 7:4 Зарезервированы.

Бит 3 **RSTCAL**. Сброс значений калибровки:
«0» – инициализация регистров калибровки завершена;
«1» – сбросить калибровку ADC.
Бит устанавливается программно для сброса калибровки, сбрасывается аппаратно после завершения инициализации регистров калибровки.

**Примечание.** Если бит RSTCAL устанавливается в то время, когда происходит процесс преобразования, потребуются дополнительные циклы для сброса регистров калибровки.

Бит 2 **CAL**. Запуск калибровки:
«0» – калибровка завершена;
«1» – запустить калибровку.
Бит устанавливается программно для запуска калибровки. Сбрасывается аппаратно после завершения калибровки.

Бит 1 **CONT.** Режим преобразования:
«0» – режим однократного преобразования;
«1» – режим непрерывного преобразования.
Бит устанавливается и сбрасывается программно. Если бит установлен, преобразование будет выполняться в непрерывном режиме до сброса данного бита.

Бит 0 **ADON**. Включение АЦП:

«0» – перевод АЦП в режим пониженного энергопотребления и прекращение преобразования/калибровки;

«1» – разрешение работы АЦП.

Бит устанавливается и сбрасывается программно.

**Примечание**. Если изменяется любой бит регистра ADC_CR2, не считая ADON, то преобразование не запускается. Это сделано для того, чтобы можно было модифицировать настройки АЦП, управляемые регистром ADC_CR2, не вызывая ложных запусков АЦП.

**ADC_SMPR1. Регистр времени выборки 1:**
Смещение адреса: 0x0C
Значение после сброса: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | SMP17[2:0] | | | SMP16[2:0] | | | SMP15[2:1] | |
| | | | | | | | | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SMP 15_0 | SMP14[2:0] | | | SMP13[2:0] | | | SMP12[2:0] | | | SMP11[2:0] | | | SMP10[2:0] | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:24 Зарезервированы.
Биты 23:0 **SMPx[2:0].** Модифицируемые программно битовые поля, которые в индивидуальном порядке определяют время выборки в тактах для каждого канала. Должны оставаться неизменными во время циклов выборки. x – номер канала АЦП.

**ADC_SMPR2 Регистр времени выборки 2**
Смещение адреса: 0x10
Значение после сброса: 0x0000 0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | SMP9[2:0] | | | SMP8[2:0] | | | SMP7[2:0] | | | SMP6[2:0] | | | SMP5[2:1] | |
| Res. | | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SMP 5_0 | SMP4[2:0] | | | SMP3[2:0] | | | SMP2[2:0] | | | SMP1[2:0] | | | SMP0[2:0] | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Биты 31:30 Зарезервированы.
Биты 29:0 **SMPx[2:0]**. Аналогично предыдущему.

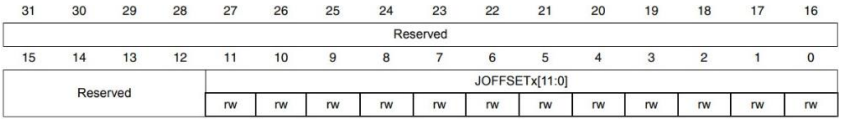
Значение поля **SMPx[2:0]**:

000: 1.5 такта;
001: 7.5 такта;
010: 13.5 такта;
011: 28.5 такта;
100: 41.5 такта;
101: 55.5 такта;
110: 71.5 такта;
111: 239.5 такта.

**ADC_JOFRx Регистр смещения инжектированных каналов (x = 1..4)**

Смещение адреса: 0x14÷0x20

Значение после сброса: 0x0000 0000



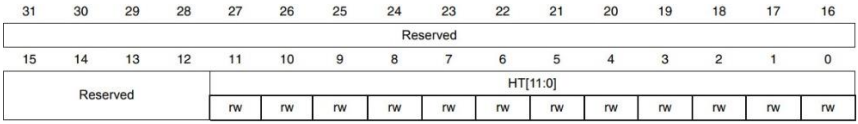

Биты 31:12 Зарезервированы.

Биты 11:0 **JOFFSETx[11:0]**: смещение данных для инжектированного канала. Это поле модифицируется программно и содержит значение, которое вычитается из результатов преобразований для инжектированных каналов перед сохранением в соответствующем регистре данных ADC_JDRx.

**ADC_HTR Регистр верхней границы оконного компаратора**

Смещение адреса: 0x24

Значение после сброса: 0x0000 0FFF



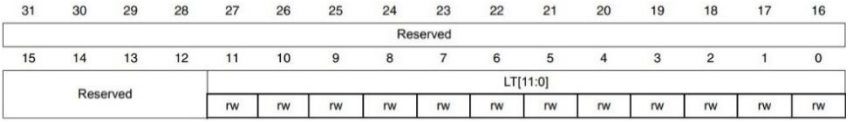

Биты 31:12 Зарезервированы.

Биты 11:0 **HT[11:0]**: верхняя граница оконного компаратора. Данное битовое поле модифицируется программно.

**ADC_LTR Регистр нижней границы оконного компаратора**

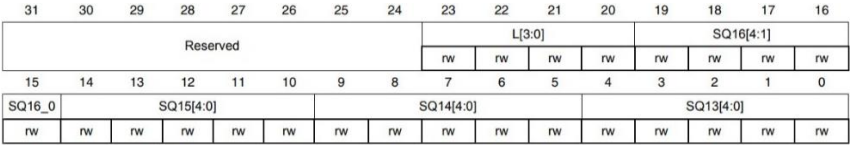
Смещение адреса: 0x28

Значение после сброса: 0x0000 0000




Биты 31:12 Зарезервированы.
Биты 11:0 **LT[11:0]**: нижняя граница оконного компаратора. Данное битовое поле модифицируется программно.

**ADC_SQR1 АЦП Регистр управления последовательностью опроса регулярных каналов 1**
Смещение адреса: 0x2C
Значение после сброса: 0x0000 0000



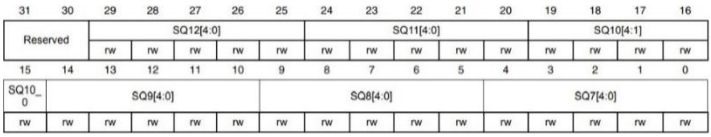

Биты 31:24 Зарезервированы.
Биты 23:20 **L[3:0]**: длина регулярной последовательности:

0000: 1 преобразование;
0001: 2 преобразования;
.....
1111: 16 преобразований.

Биты 19:0 **–** номера каналов в очереди **SQx[4:0], где x – номер очереди канала.** Например: если L[3:0] = 0011, а SQl = 00100, SQ2 = 000l1, SQ3 = 1000l, SQ4 = 01001, то последовательность обработки будет – 4-й канал, 3-й канал, 17-й канал, 9-й канал.

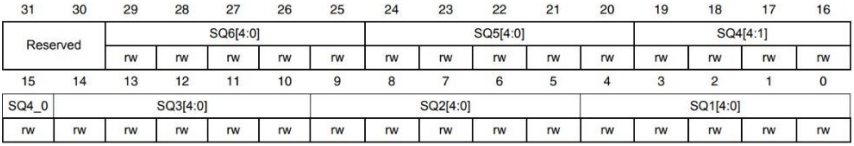
Данное битовое поле модифицируется программно и определяет общее количество преобразований в регулярной последовательности.

**ADC_SQR2 Регистр управления последовательностью опроса регулярных каналов 2**
Смещение адреса: 0x30
Значение после сброса: 0x0000 0000



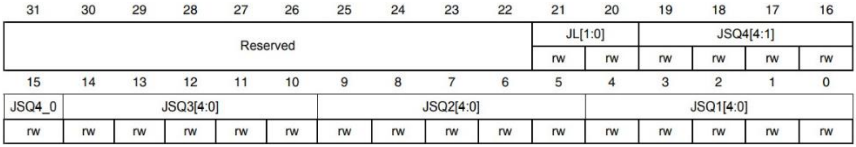
Биты 31:30 Зарезервированы.
Биты 29:0 – номера каналов в очереди **SQx[4:0], где x – номер очереди канала**. Данное битовое поле модифицируется программно и задаёт номер канала.

**ADC_SQR3 Регистр управления последовательностью опроса регулярных каналов 3**
Смещение адреса: 0x34
Значение после сброса: 0x0000 0000




Биты 31:30 Зарезервированы, должны храниться в значении сброса.
Биты 29:0 – номера каналов в очереди **SQx[4:0], где x – номер очереди канала.**

**ADC_JSQR Регистр управления последовательностью опроса инжектированных каналов**
Смещение адреса: 0x38
Значение после сброса: 0x0000 0000



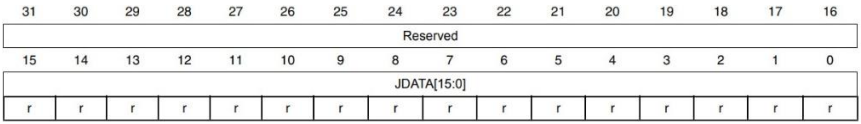

Биты 31:22 Зарезервированы.
Биты 21:20 **JL[1:0]**. Длина инжектированных последовательностей:
00: 1 преобразование;
01: 2 преобразования;
10: 3 преобразования;
11: 4 преобразования.
Данное битовое поле модифицируется программно и определяет общее количество преобразований в инжектированной последовательности.
Биты 19:0 **JSQx[4:0]**: номера инжектированных каналов в очереди **SQx[4:0], где x –номер очереди канала**. Поле задаётся программно.

**Примечание.** В случае инжектируемой группы имеются отличия по сравнению регулярной группой в способе определения и порядке выполнения преобразований. Если JL[1:0] меньше 4, то выполнение начнётся с элемента (4-JL).

Например, при значении регистра ADC_JSQR[21:0] = 10 00011 00011 00111 00010 **JSQ1 будет проигнорировано** (4 – 2 = 2) и последовательность использования каналов будет такая: 7, 3, 3.

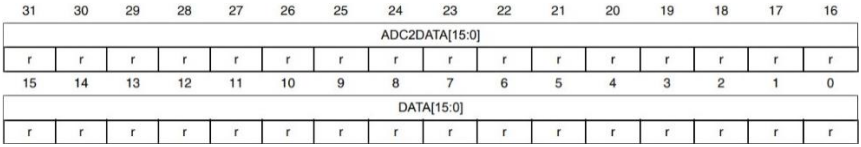
При JL[1:0] = 11 последовательность будет начинаться с канала, записанного в **JSQ1.**

**ADC_JDRx Регистр данных инжектированных каналов (x = 1..4)**
Смещение адреса: 0x3C÷0x48
Значение после сброса: 0x0000 0000




Биты 31:16 Зарезервированы.
Биты 15:0 **JDATA[15:0]**: данные инжектированных каналов. Биты только для чтения.

**ADC_DR Регистр данных регулярных каналов**
Смещение адреса: 0x4C
Значение после сброса: 0x0000 0000




Биты 31:16 **– ADC2DATA[15:0]:** данные АЦП2. При каскадном соединении АЦП1 и АЦП2 эти биты содержат данные АЦП2 (**в STM32C8T6 не используются)**.
Биты 15:0 **DATA[15:0]:** данные регулярных каналов. Биты только для чтения.

# ПРИЛОЖЕНИЕ 2. Базовые понятия для программирования на языке Си

Проект состоит из файлов с расширениями *.c и *.h. В первых находятся реализации функции, во-вторых, объявления используемых функций и констант [91]. Основной файл, в котором находится код программы, называется «main.c». При необходимости использования функций, написанных сторонними разработчиками, можно подключать библиотеки с этими функциями. Подключаются они с помощью директивы компилятора ***#include "название_библиотеки"***, в самом начале файла. Библиотеки предварительно должны быть добавлены в проект.

Функции – это часть программы. Программа состоит из одной или нескольких функций. Функция имеет вид:

```
тип_возвращаемой_переменной имя_функции
(тип_переменной)
{
        Тело функции
}
```

В функцию можно передать переменную, функция её обработает и вернёт значение. Как правило, функции используют для реализации повторяющихся действий. Перед тем, как использовать функцию, её необходимо объявить в самом начале файла в следующем виде:

```
тип_возвращаемой_переменнойимя_функции
(тип_переменной);
```

В конце каждой строки должна быть точка с запятой. Если функция ничего не возвращает (например, временная задержка), то тип указывают **void**.

При запуске программы, написанной на языке Си, первой всегда выполняется функция **main()** – точка входа программы.

Переменные, используемые в программе, могут быть различных типов:

**int** – переменная этого типа может быть только целым числом от –2147483648 до 2147483647;

**float** – переменная этого типа число с точностью до 7 разрядов от $\pm 1{,}5*10^{45}$ до $\pm 3{,}4*10^{33}$;

**double** – число с точностью до 16 разрядов от $\pm 5*10^{-324}$ до $\pm 1{,}7*10^{306}$;

**ulong** – тоже целое число, но от 0 до 18446744073709551615;

**long** – целое от -9223372036854775808 до 9223372036854775807;

**char** – один символ;

**bool** – логическая переменная, она может принимать одно из двух возможных значений: истина (true) или ложь (false);

**Строку** (слово, предложение) можно представить как массив из символов типа **char**.

Например:

***char strMessage[5] = "Слово";***

В квадратных скобках – количество символов в строке, «strMessage» – название массива.

Перед использованием переменной её обязательно нужно объявить (указать тип переменной и имя).

**Операторы** – символы при помощи которых производятся какие-либо операции над переменными:

**+** – сложение;

**-** – вычитание;

***** – умножение;

**/** – деление;

**=** – присвоение переменной значения.

Например, выражение a = b + c означает необходимость присвоить переменной a результат вычисления суммы значений переменных b и c.

**++** – инкремент (увеличение значения переменной на 1);

**--** – декремент (уменьшение значения переменной на 1);

Например, выражение a++ указывает на необходимость увеличить значение переменной на 1 (то же самое, что и a = a + 1).

**==** – сравнение, знак «равно» (не путать с присвоением);
**!=** – сравнение, знак «не равно»;
**<** – сравнение, знак «меньше»;
**<=** – сравнение, знак «меньше или равно»;
**>** – сравнение, знак «больше»;
**>=** – сравнение, знак «больше или равно»;

Например, выражение a < b становится истинным, если значение переменной «a» меньше значения переменной «b» и ложным, если значения равны или «a» больше «b». Выражение a == b истинно, если «a» равно «b» и ложно, если «a» не равно «b». Однако, выражение a = b истинно всегда, потому что это не сравнение, это присвоение переменной «a» значения переменной «b».

**%** – остаток от деления;

Например, если a = 5 и b = 3, то результат вычисления выражения a % b будет равно 2 (т.к. 5 / 3 = 1, а остаток 2).

**<<** – побитовый сдвиг влево;
**>>** – побитовый сдвиг вправо;
**&** – логическое «И»;
**|** – логическое «ИЛИ»;
**~** – инвертирование.

Циклы с предпроверкой условий (основной цикл программы).

```
while(условие)
{
        тело цикла
}
```

Тело цикла (всё, что в фигурных скобках) выполняется, когда условие истинно (пока условие не станет ложным).

Цикл со счетчиком выполняется определенное количество раз с определенным шагом переменной.

```
for (начальное_значение; цикл_выполняется_до; шаг)
{
        тело цикла  ,
}
```

где:

**начальное_значение** – начальное значение счётчика;

**цикл_выполняется_до** – указывается до достижения какого значения выполняется цикл;

**шаг** – с каким шагом счетчик считает.

Например:

```
for (i = 0; i < 10; i++)
{
        тело цикла
}
```

Здесь начальное значение переменной i равно 0, цикл выполняется, пока значение переменной i меньше 10, в каждой итерации цикла значение i увеличивается на 1. Так же значение переменной можно изменять прямо в цикле.

Условный переход.

```
if (условие)
{
        тело 1
} else
{
        тело 2
}
```

При условном переходе «тело 1» выполняется, если условие истинно, иначе (если условие ложно) выполняется «тело 2». Существует также следующий вариант условного перехода:

```
if (условие 1)
{
        тело 1
} elseif (условие 2)
{
        тело 2
}
```

В этом случае «тело 1» выполняется, если истинно «условие 1», «тело 2» выполняется, если истинно «условие 2». Таких условий может быть сколько угодно много, так же может быть одно else (иначе).

Условия могут быть простыми и составными: простые – одно логическое выражение, а составные – несколько логических выражений, соединённых знаком & (логическое «И») или | (логическое «ИЛИ»).

Код, для повышения его читаемости, рекомендуется снабжать комментариями. Комментарии помогут разобраться в проекте. Строки можно комментировать знаками // и до конца строки. Блок текста (несколько срок) можно комментировать символами **/***, ставится в начале комментируемого блока, и ***/** – в конце комментируемого блока. На размер программы комментарии не влияют.

# ПРИЛОЖЕНИЕ 3. Пример настройки портов ввода-вывода общего назначения GPIO

В данном приложении показан пример создания и настройки проекта для управления светодиодом, установленным на плате отладочного модуля (см. раздел 3.1.4, рисунок 3.29).

Для разработки программы использовались STM32CubeMX и SW4STM32.

Создание проекта в STM32CubeMX происходит в несколько этапов. На первом этапе создается новый проект (рисунок П 3.1) и выбирается нужный микроконтроллер (рисунок П. 3.2), далее приведены копии экранных форм для STM32CubeMX версии 5.2.1).

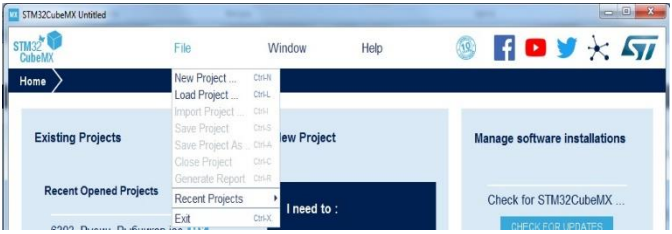



Рис. П 3.1 – Окно создания проекта

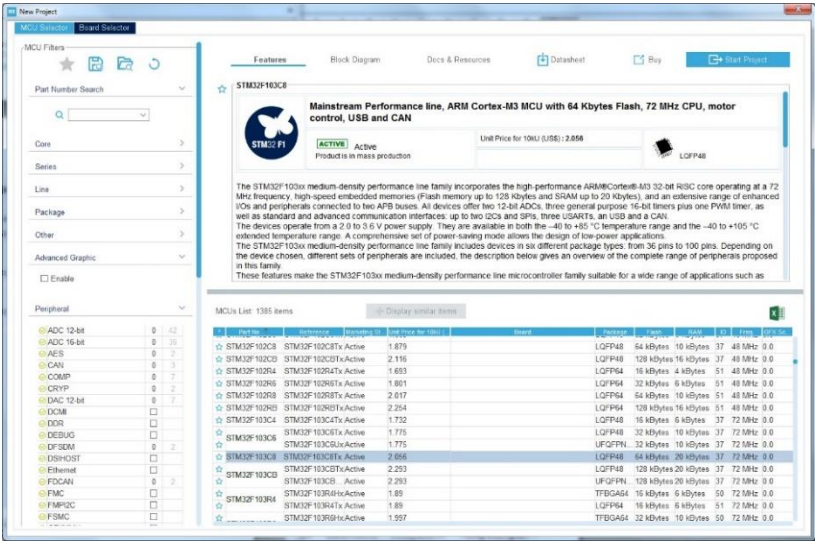



Рис. П 3.2 – Окно выбора микроконтроллера

Тип микроконтроллера выбирается из списка «MCUs List», поддерживаемых код-генератором. При поиске микроконтроллера есть возможность настроить различные фильтры «MCU Filters».

В рассматриваемом примере выбран микроконтроллер STM32F103C8Tx, используемый в отладочном модуле (рисунок П 3.3).

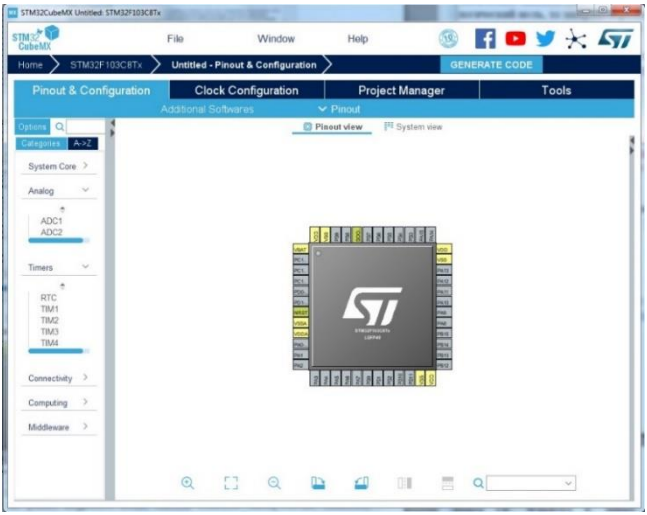



Рис. П 3.3 – Линии ввода/вывода контроллера STM32F103C8T6

Имеется возможность указать любую линию микроконтроллера, выбрать режим ее работы. При этом, в случае необходимости, система отметит линию символом, предупреждающим об использовании альтернативных функций и потенциальном конфликте (рисунок П 3.4).

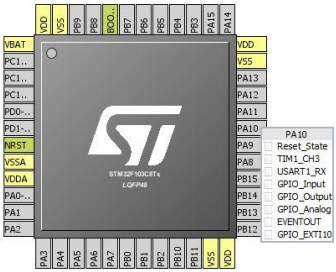



Рис. П 3.4 – Выбор функций линии PB10 контроллера

Сконфигурировать тактовую частоту, выставив переключатели и задав параметры делителей/умножителей частоты в нужные значения. Если в процессе настройки будут допущены ошибки, система предупредит об этом (рисунок П 3.5).

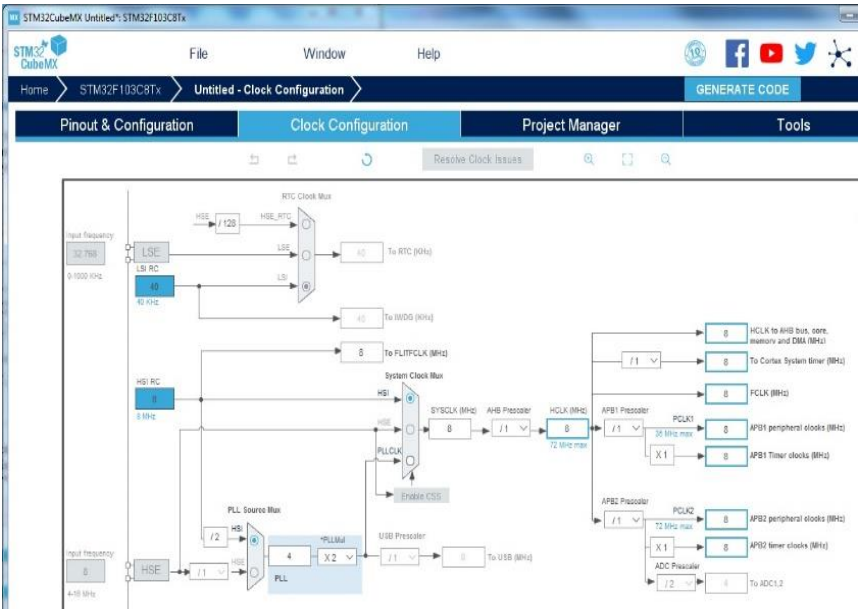



Рис. П 3.5 – Окно схемы тактирования STM32F103C8Tx

Более тонкие настройки параметров периферии, используемой в проекте, можно произвести на вкладке «Configuration» (рисунок П 3.6).

По окончании процесса конфигурирования проект следует сгенерировать для выбранной среды разработки (IDE).

**Настройка параметров проекта.**

Для корректной работы проекта, созданного код-генератором, его необходимо сконфигурировать. Графический интерфейс STM32CubeMX позволяет это сделать наглядно (рисунок П 3.7).

В первую очередь, важно настроить интерфейс отладки SWD (Serial Wire Debug), через который будет производиться запись новых программ в память контроллера, а также отслеживание текущего состояния процесса выполнения, поиска и исправления ошибок.

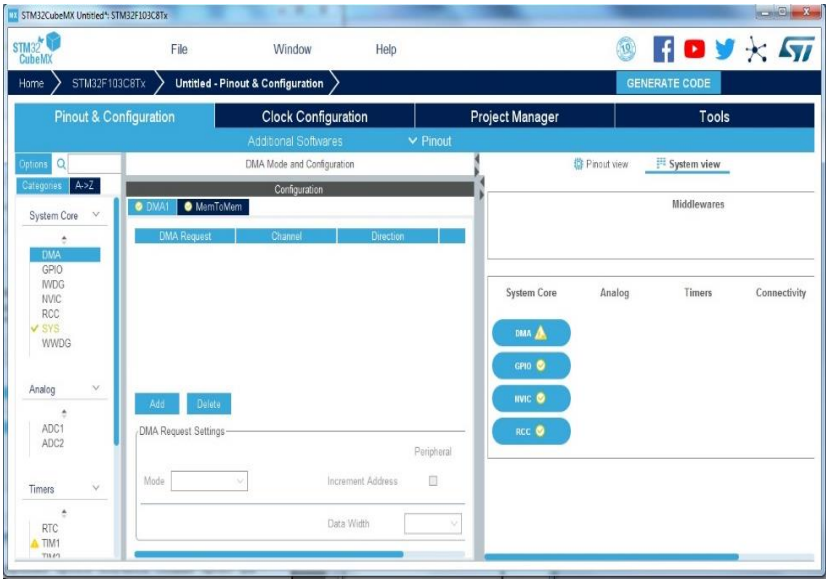



Рис. П 3.6 – Окно конфигурации периферии STM32F103C8Tx

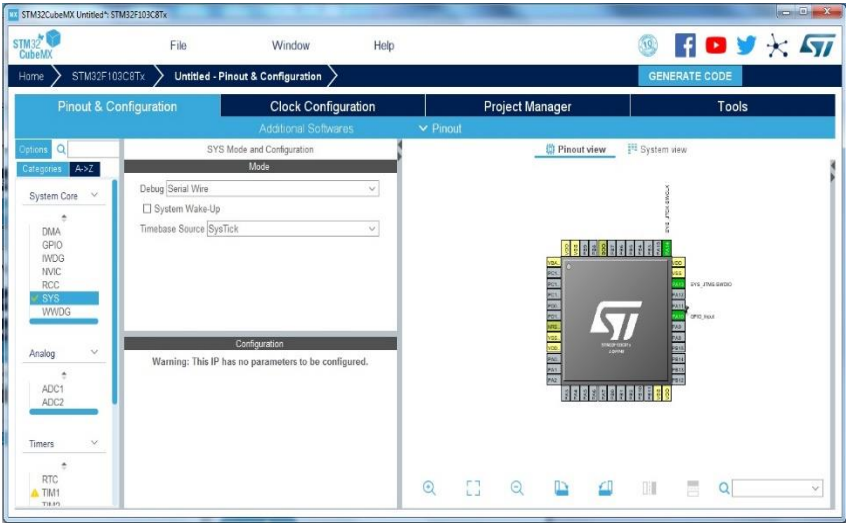



Рис. П 3.7 – Графический интерфейс STM32CubeMX

**После выбора интерфейса SWD, используемого для отладки, автоматически конфигурируются выходы PA13 и PA14 микропроцессора (**рисунок П 3.7**). Если этого не сделать, отладка может быть не доступна.**

**В данной задаче выполняется управление светодиодом LED3, который соединяется перемычкой с выходом PB5**. Размещение светодиодов на плате отладочного модуля показано на рисунке П 3.8.

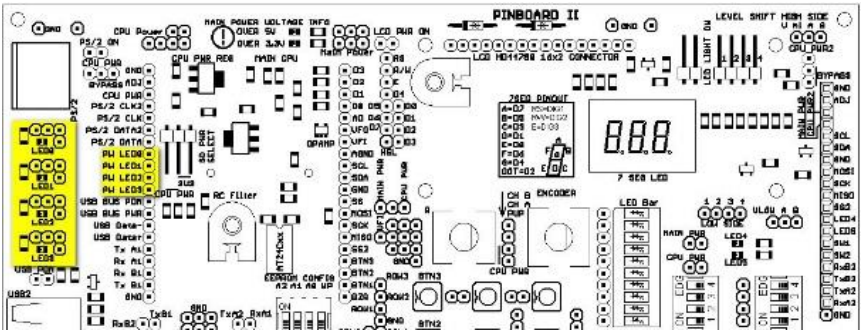



Рис. П 3.8 – Блок светодиодов отладочного комплекта

В код-генераторе необходимо выбрать порт и задать режим GPIO_Output (рисунок П 3.9).

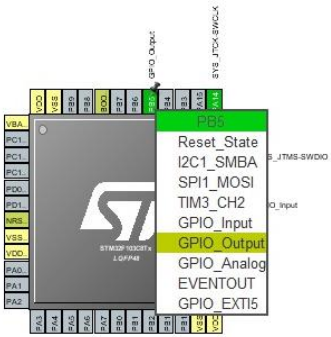



Рис. П 3.9 – Интерфейс конфигурирования порта для управления светодиодом

На следующем этапе конфигурируются выводы кварцевого генератора. В качестве источника частоты можно выбрать внешний кварц «Crystal/Ceramic Resonator» (рисунок П 3.10).

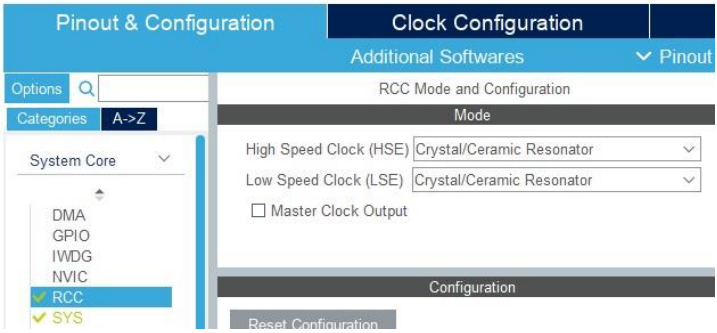



Рис. П 3.10 – Окно задания источника частоты

Далее необходимо перейти во вкладку «Clock configuration» для конфигурирования тактовой частоты, часовой кварц оставить без изменений (32.768 кГц), а кварц основной установить на 12 МГц (рисунок П 3.11).

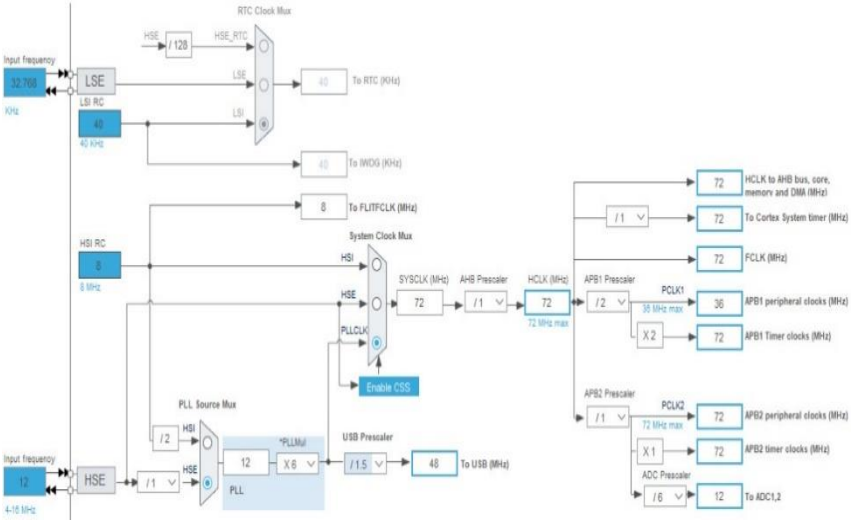



Рис. П 3.11 – Схема конфигурации тактовой частоты
на вкладке «Clock configuration»

Теперь необходимо поставить множитель PLL (Phase-Locked Loop): переключить тактирование на кварц HSE (англ. High Speed External oscillator, внешний генератор тактовой частоты) и подобрать параметры так, чтобы «HCLK» приняло значение 72

МГц, т.е. чтобы контроллер работал на максимальной частоте. Если задать «*PLLMul» равным «x6» и переключатель «SystemClockMux» установить в значение PLLCLK, возникнет ошибка в настройке частоты «APB1 Prescaler». Исправить ошибку можно задав значение предделителя частоты «APB1 Prescaler» равным «/2».

Таким образом, микроконтроллер настроен для работы на максимальных частотах. Далее необходимо перейти к конфигурации и произвести настройку выбранных интерфейсов и модулей микропроцессора. В частности, свойству «Maximum output speed» вывода PB5 присвоено значение «Low», необходимо задать значение скорости «Medium» (рисунок П 3.12).

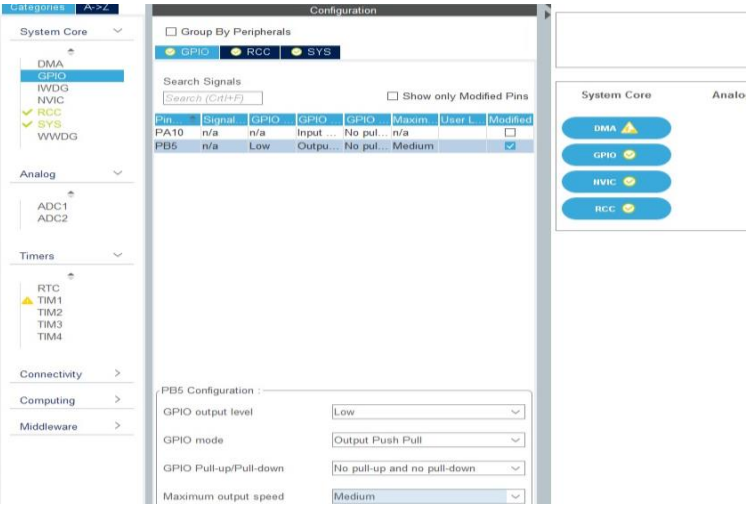



Рис. П 3.12 – Окно свойств вывода PB5

После того, как микроконтроллер сконфигурирован, необходимо сгенерировать проект. Для этого нужно выбрать пункт меню «Project Manager» или «Project/Settings…» в зависимости от версии программы. В открывшемся окне необходимо указать путь для создания проекта и его имя, а также в списке «Toolchain/IDE» указать среду, в которой продолжится создание проекта. Если необходимо, можно увеличить минимальный размер кучи «MinimumHeapSize» и минимальный размер стека «MinimumStackSize» (рисунок П 3.13).

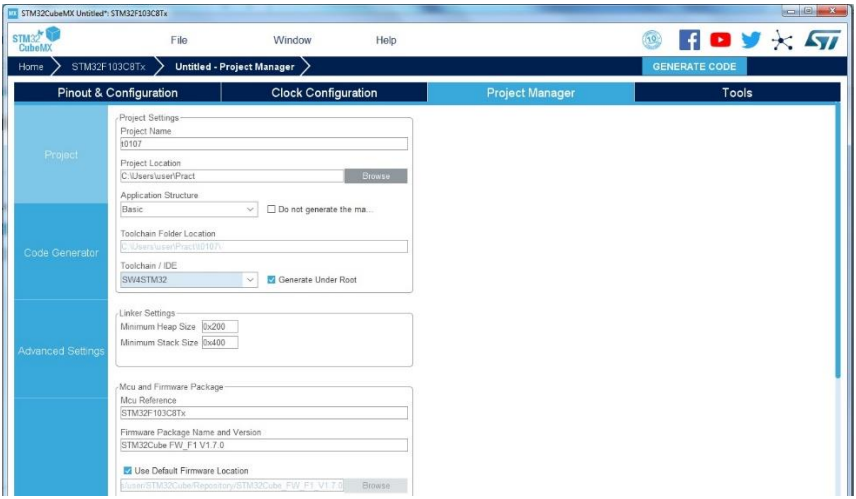



Рис. П 3.13 – Окно параметров проекта

В качестве среды разработки выбрана IDE SystemWorkbench for STM32 (в списке «Toolchain/IDE» доступна под названием «SW4STM32»).

Создание проекта в выбранной среде разработки осуществляется выбором в STM32CubeMX пункта меню «Generate code» или нажатием сочетание клавиш **Ctrl + Shift + G**.

После генерации проекта, система предложит его открыть в проводнике или в той среде разработки, которую пользователь указал при конфигурировании (рисунок П 3.14).

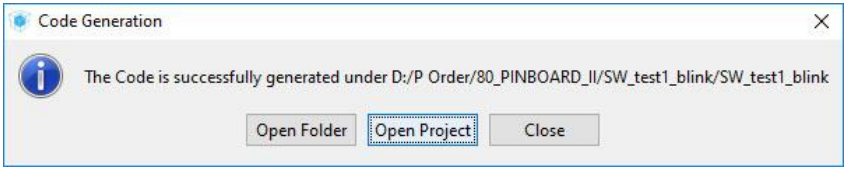



Рис. П 3.14 – Окно выбора действий после генерации кода проекта

Структура проекта в среде разработки и проводнике представлена на следующем рисунке (рисунок П 3.15).

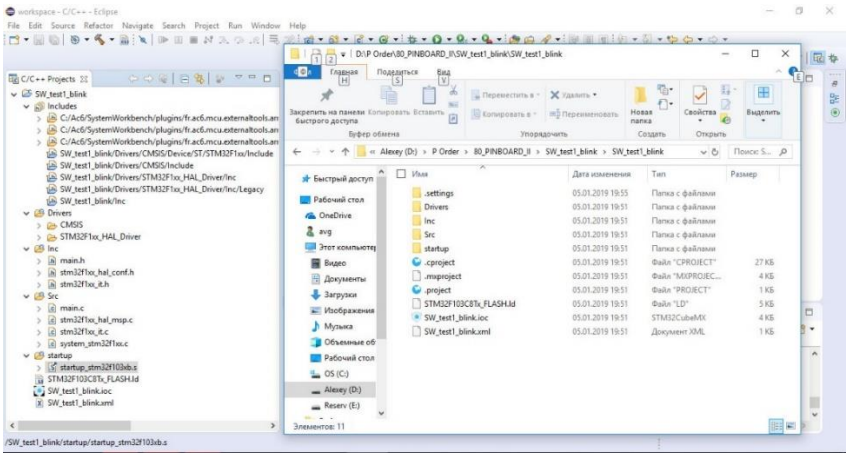


Рис. П 3.15 – Структура проекта в проводнике

Из структуры проекта видно, что драйверы и библиотеки HAL и CMSIS уже подключены. Главная программа реализуется в файле «main.c»; «stm32f1xx_it.c» – содержит обработчики прерываний (англ. interruption) и исключений (англ. exception); в «system_stm32f1xx.c» реализована начальная инициализация микроконтроллера.

***Прежде чем приступить к изучению структуры проекта и написанию первой программы, необходимо выучить/вспомнить фундаментальные основы языка Си (см. приложение 2) и базовые понятия программирования микроконтроллеров ARM Cortex от компании STM32.***

***Структура проекта***

В основной программе «main.c» подключены заголовочные файлы, сконфигурированы системные часы (функция *SystemClock_Config()*), сконфигурирован интервал периодических прерываний *HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);*. В функции *MX_GPIO_Init()* сконфигурированы используемые входы/выходы (GPIO) (рисунок П 3.16).

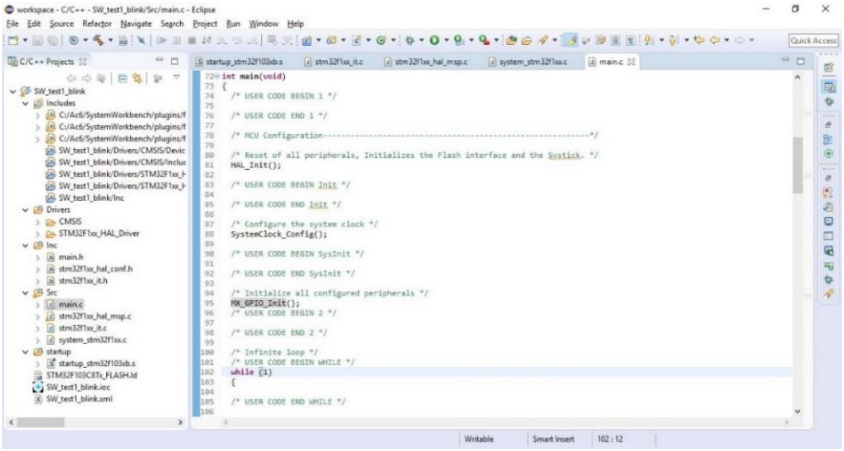


Рис. П 3.16 – Окно файла программы в редакторе

В главной функции программы ***int main(void)*** вызываются конфигураторы HAL ***HAL_Init()***, системных часов – ***SystemClock_Config()***, цифровых вводов/выводов – ***MX_GPIO_Init()***. Далее вызывается бесконечный цикл, где реализуется основной код программы:

```
/* Infiniteloop */
/* USER CODE BEGIN WHILE */
while (1)
  {

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

}
/* USERCODEEND 3 */
```

**Необходимо обратить внимание на секции *USER_CODE_BEGIN* и *USER_CODE_END*.** Весь пользовательский код должен находиться внутри этих секций. Иначе, при обновлении проекта CubeMX, код пропадет (на это указано в документации, однако, на практике случается так, что пользовательский код пропадает независимо от того в какой секции он написан).

Для управления портами ввода/вывода имеются стандартные HAL функции, их детальное описание можно посмотреть в документе «UM 1850 User manual. Description of STM32F1 HAL and Low-layer drivers», представленном на сайте https://www.st.com в разделе технической документации.

В соответствии с описанием HAL-функций программа мигания светодиодом будет выглядеть следующим образом:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
  {
        // выключитьсветодиод
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5,
GPIO_PIN_RESET);
        // задержка выключенного состояния на 200 мс
        HAL_Delay(200);
        // включитьсветодиод
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5,
GPIO_PIN_SET);
        // задержка включенного состояния на 400 мс
        HAL_Delay(400);
/* USERCODEENDWHILE */

/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Далее необходимо скомпилировать код, выбрав пункт меню Project → BuildAll, нажав сочетание клавиш **Ctrl+B** или нажав на кнопку панели инструментов (рисунок П 3.17).

После того как проект успешно скомпилируется, его можно запускать в отладочном модуле. Для этого необходимо выбрать пункт меню **Run → Debug**, нажать горячую клавишу **F11** или кнопку панели инструментов (рисунок П 3.18).

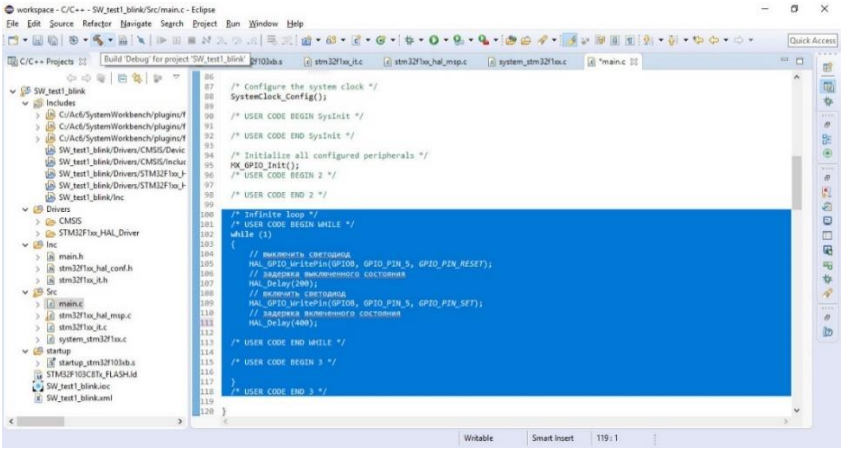


Рис. П 3.17 – Кнопка «Build» компиляции в панели инструментов


Рис. П 3.18 – Кнопка «Debug» отладки в панели инструментов

Также через панель инструментов, нажав на стрелку возле иконки «Debug», можно выбрать конфигурацию отладки (рисунок П 3.19):

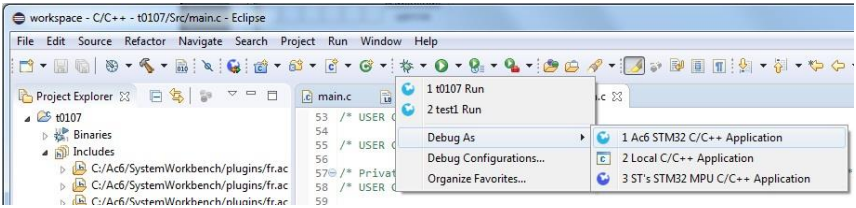



Рис. П 3.19 – Кнопка «Debug» отладки в панели инструментов

После осуществления указанных действий среда разработки не переходит в режим отладки, при этом появляется сообщение об ошибке (рисунок П 3.20).

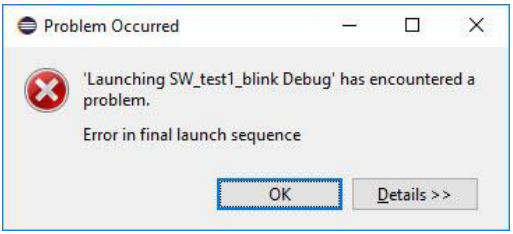



Рис. П 3.20 – Окно ошибки перехода в режим отладки

Для того, чтобы иметь возможность производить отладку, необходимо выполнить настройку среды разработки. Открыть окно «Project/Properties» (рисунок П 3.21).

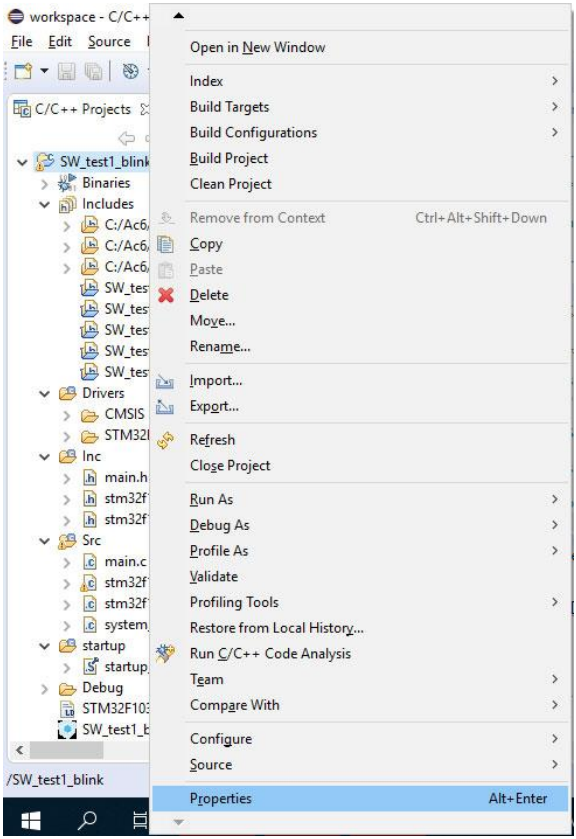



Рис. П 3.21 – Пункт меню для доступа к свойствам проекта

Перейти в раздел «Run/DebugSettings» (рисунок П 3.22).

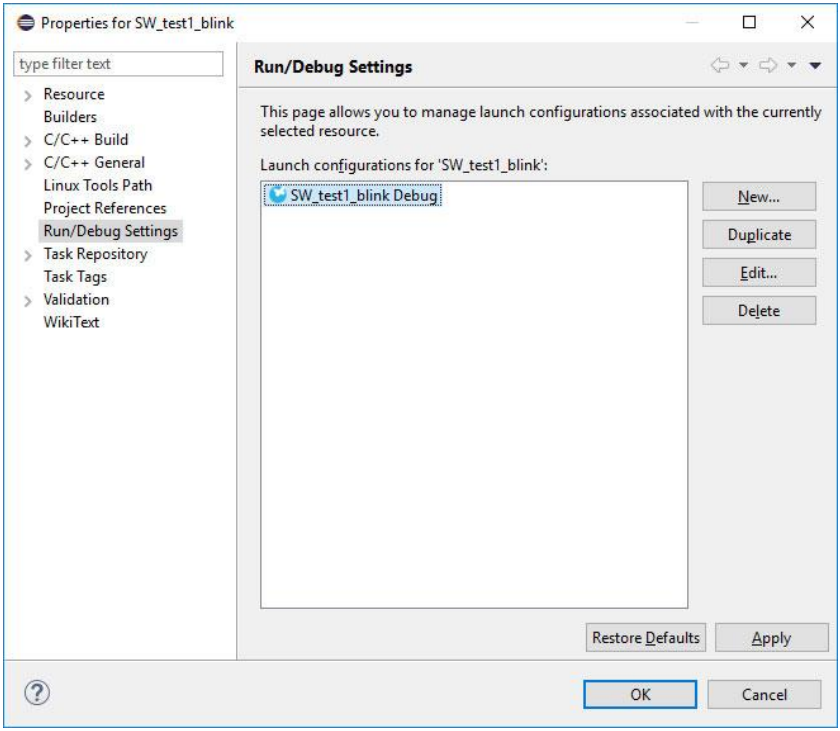



Рис. П 3.22 – Окно свойств проекта

Начать редактирование конфигурации запуска проекта (рисунок П 3.23).

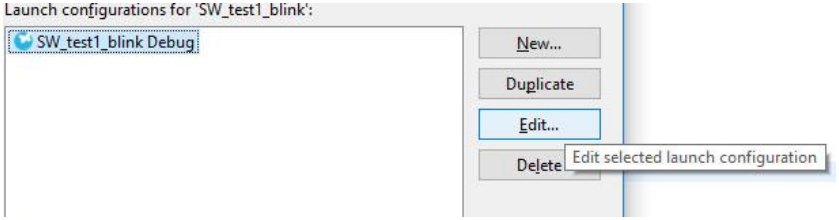



Рис. П 3.23 – Кнопка конфигурации процесса запуска проекта

На вкладке «Debugger» в появившемся окне необходимо нажать на кнопку «Show generator options…» (рисунок П 3.24).

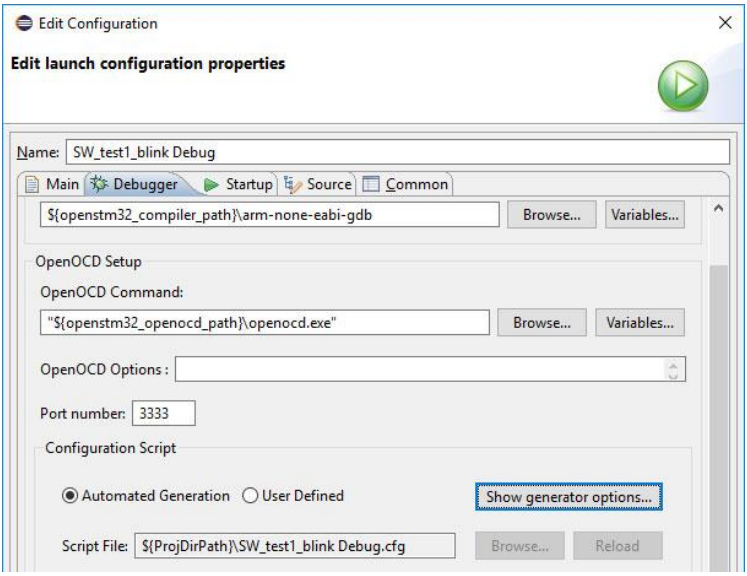



Рис. П 3.24 – Окно параметров генератора

В появившемся разделе «Generator options», в подразделе «ModeSetup» необходимо выбрать режим сброса «ResetMode» и указать «Software system reset» (Программный сброс) (рисунок П 3.25).

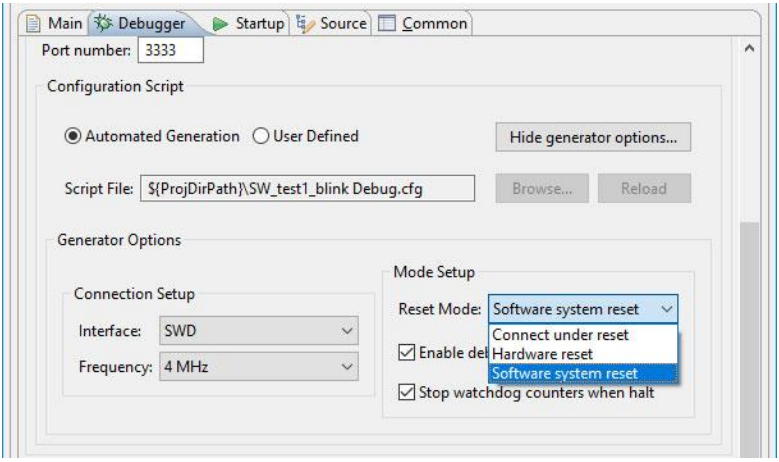



Рис. П 3.25 – Окно задания режима установки программы

После внесения указанных изменений в конфигурацию проекта можно приступать к процессу отладки. Если необходимо запустить приложение, минуя процесс отладки, можно нажать комбинацию клавиш **Ctrl + F11** или выбрать пункт меню **Run → Run**. При этом необходимо убедиться, что во вкладке Main в поле «C/C++ Application» указан правильный elf-файл (рисуноку П 3.26).

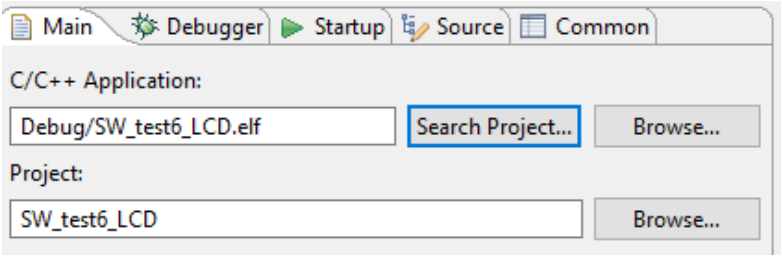



Рис. П 3.26 – Основное окно настройки конфигурации запуска

**Поиск неисправностей**

Бывают случаи, когда контроллер не отвечает и его не получается запрограммировать. Это может быть связано с тем, что в контроллер был записан проект, с отключенным SWD. Напомним, что через интерфейс SWD производится отладка. Недоступность контроллера может быть связана с включенной зашитой прошивки, не предполагающей его программирование.

Устранить возникшие трудности можно с помощью утилиты STM32 ST-LinkUtility (см. раздел 4). Для этого в CubeMX необходимо создать проект, как показано выше, с настроенными выходами SWD (активированным последовательным интерфейсом отладки), после чего с помощью утилиты прошить *.hex файл в контроллер.

# ПРИЛОЖЕНИЕ 4. Формирование сигнала ШИМ под управлением компьютера

В данном приложении представлен пример программирования микроконтроллера для решения следующей задачи:

Формирование сигнала ШИМ с заданной частотой и скважностью.

Управление работой ШИМ терминальной программой компьютер. (пуск, останов, сохранения текущего значения длительности импульса). Пуск – сообщение «1», останов – сообщение «0».

Связь с компьютером осуществляется через *USB-порт* микроконтроллера, работающего в режиме full-speed по протоколу *USB 2.*

Задача должна быть решена средствами отладочного модуля PinBoard II R3.

Для реализации ШИМ предлагается использовать счетчик-таймер TIM4 (выход PB6), работающий в режиме ШИМ (раздел 1.6.2.1 стр. 73–75).

В качестве исполнительного устройства, иллюстрирующего функционирование ШИМ–модулятора, используется светодиод LED3, яркость которого будет изменяться в зависимости от параметров ШИМ.

Электрическая схема подключения LED3 для этого случая представлена на рисунке П 4.1.

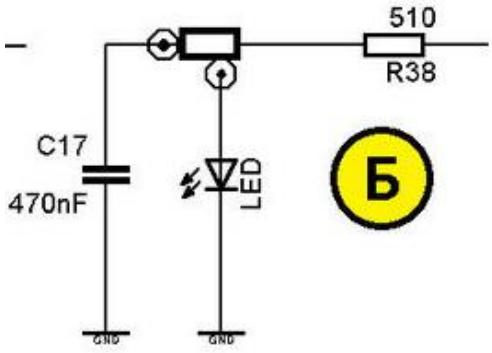



Рис. П 4.1 – Схема соединения светодиода для контроля ШИМ

Схема соединений отладочного модуля для данной задачи представлена на рисунке П 4.2.

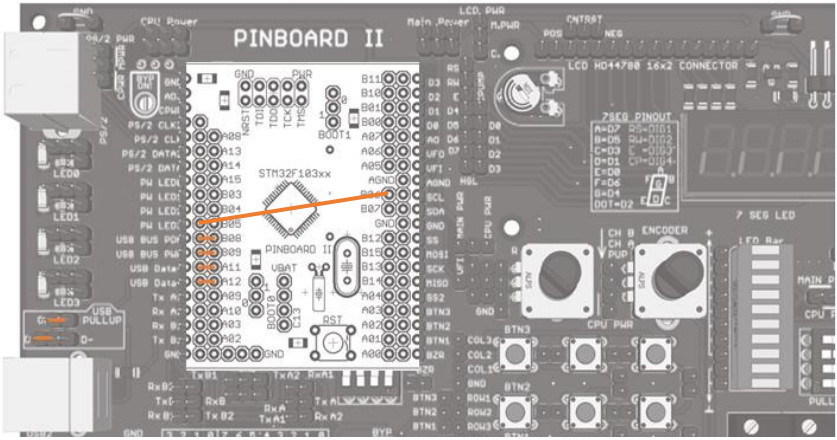



Рис. П 4.2 – Схема соединений отладочного модуля

Частота сигнала на выходе ШИМ определяется значением регистра автоперезагрузки TIMx_ARR, а длительность импульса (скважность) – содержимым регистра сравнения TIMx_CCRx. (раздел 1.6.2.1 стр. 73–75). Диаграмма сигнала ШИМ представлена на рисунке П 5.3, где t – длительность импульса, а T – период сигнала.

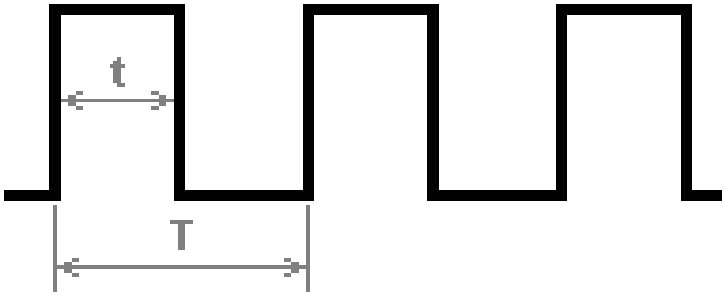



Рис. П 4.3 – Сигнал ШИМ

Для определения значений, записываемых в эти регистры, предположим, что частота ШИМ равна 50 Гц, длительность

импульса изменяется в диапазоне от 10 мс с установленным шагом, а частота шины APB1, к которой подключен TIM4, равна 8МГц.

Частота генерации импульсов при частоте шины APB1 равной 8 МГц, определяется как

$$f_{\text{Имп}} = \frac{f_{APB1}}{PSC_{TIM4} \cdot N_{\text{Имп}}},$$

где $N_{\text{Имп}}$ – количество тактов счетчика в одном импульсе;
$PSC_{TIM4}$ – значение предделителя частоты работы таймера 4.

Значение $N_{\text{Имп}}$ выбирают исходя из требований к шагу дискретизации длительности импульса. Если выбрать количество тактов счетчика в одном импульсе сигнала равным 1000, **то значение $PSC_{TIM4}$ будет равно 160**.

При настройке блока синхронизации необходимо определить источник тактирования (внешний HSE или внутренний HSI) и соответственно установить делители/умножители частоты.

При выборе источник тактирования следует учесть, что генератор USB должен быть настроен на частоту 48 МГц.

Для данной задачи, параметры, задаваемые при настройке в CUBE MX, будут равны

- Prescaler = 159;
- Counter Period = 999;
- Pulse = 499.

Значения Prescaler, Counter Period и Pulse установлены равными соответственно (160-1), (1000-1), и (500-1) в связи с тем, что счет ведется с нуля.

Программирование микроконтроллера осуществляется с помощью STM32CubeMX (версия 5.2.1) и System Workbench for STM32 (версия 4.6.3).

**Перед началом программирования соединить разъем USB2 отладочного модуля кабелем с компьютером и подключить программатор (см. раздел 3.2.3).**

Для решения поставленной задачи в STM32CubeMX необходимо выполнить следующие действия.

1. Создать новый проект, выбрать соответствующий контроллер, задать альтернативную функцию PB6.

2. Определить источник тактирования (выбран внешний кварцевый резонатор 12 МГц) (рисунок П 4.4).

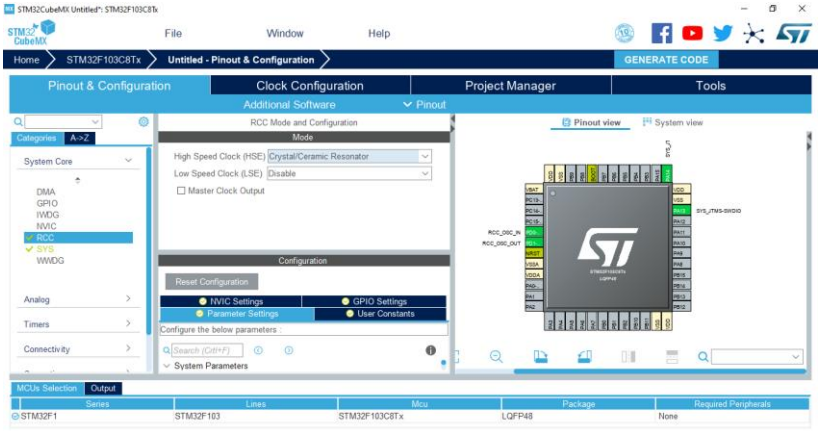



Рис. П 4.4 – Настройка источника тактирования

3. Настроить порт программирования (рисунок П 4.5).

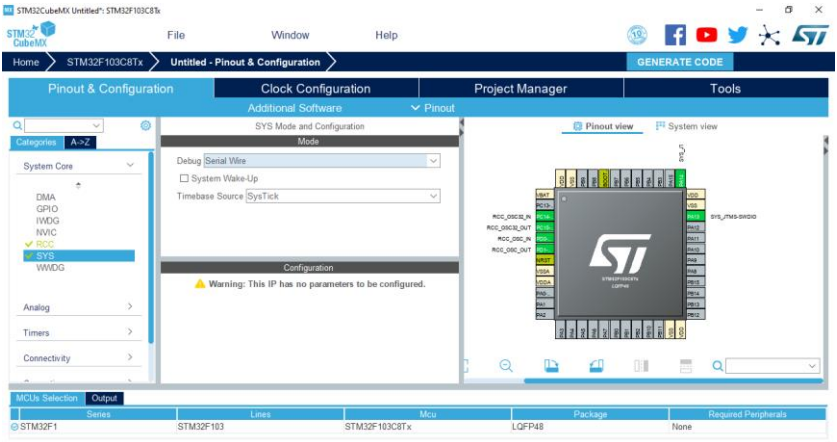



Рис. П 4.5 – Настройка режима отладки

4. Для соединения с компьютером через интерфейс USB выбрать режим максимальной скорости «Full Speed» (FS) (рисунок П 4.6).

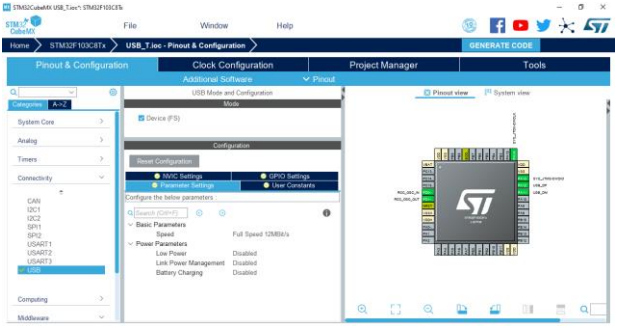



Рис. П 4.6 – Настройка параметров USB

5. Для интерфейса USB выбрать режим виртуального Com-порта «Communication Device Class (Virtual Port Com)» (рисунок П 4.7).

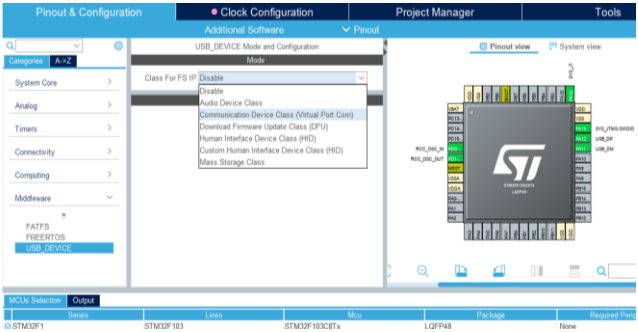



Рис. П 4.7 – Задание режима виртуального Com-порта

6. Настроить режим ШИМ TIM4 (внутреннее тактирование «Internal Clock», канал 1, режим «PWM Generation CH1») (рисунок П 4.8).

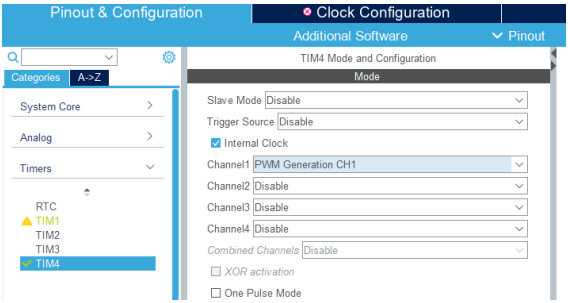



Рис. П 4.8 – Настройка таймера на режим ШИМ

7. Задать параметры синхронизации (рисунок П 4.9).

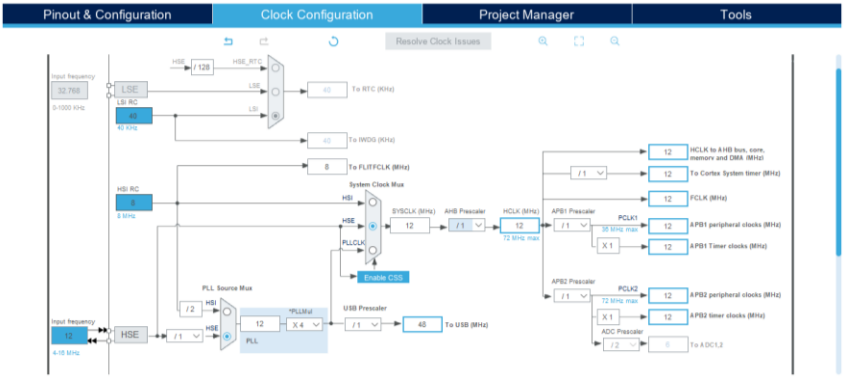



Рис. П 4.9 – Настройка параметров синхронизации

8. Настроить параметры таймера в разделе «Configuration» вкладка «Parameter Settings». (Prescaler = 159, Counter Period = 999, Pulse = 499 (рисунок П 4.10).

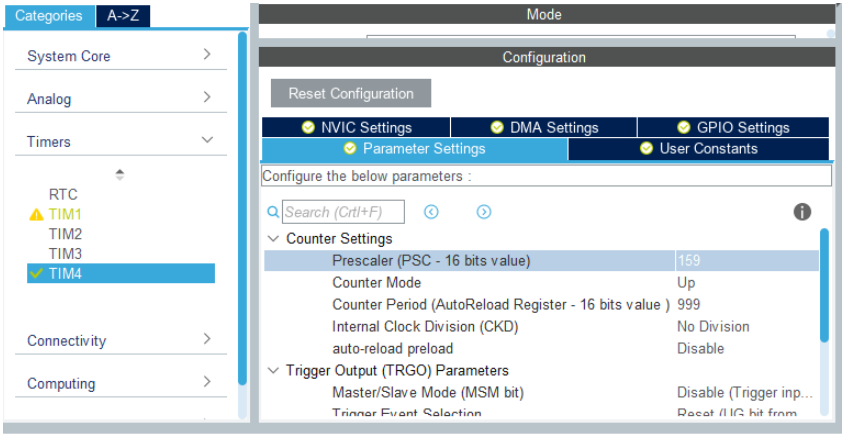



Рис. П 4.10 – Настройка параметров таймера

9. Сгенерировать проект для выбранной среды разработки, предварительно указав наименование и расположение проекта. **В названиях следует использовать только латинские символы и не допускать пробелов** (рисунок П 4.11).

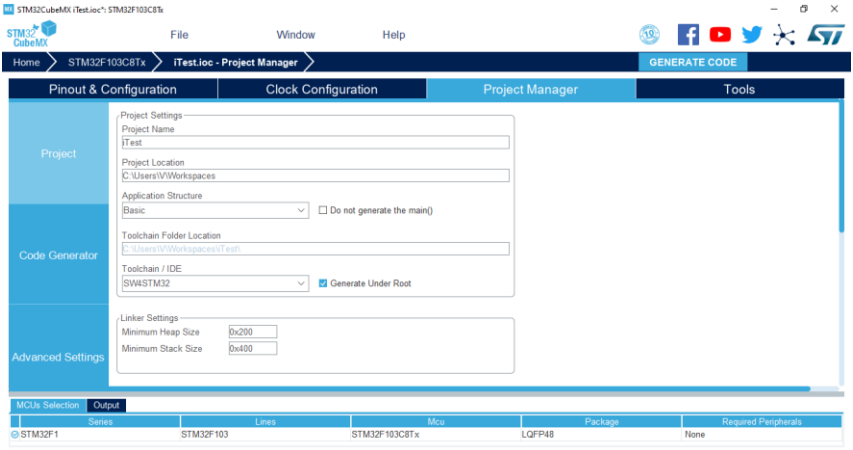



Рис. П 4.11 – Окно настройки параметров проекта

10. После того, как код сгенерирован в CUBE MX и открыт в среде разработки SW4STM32, необходимо убедиться, что выбрано рабочее пространство, соответствующее расположению открытого проекта (рисунок П 4.12).

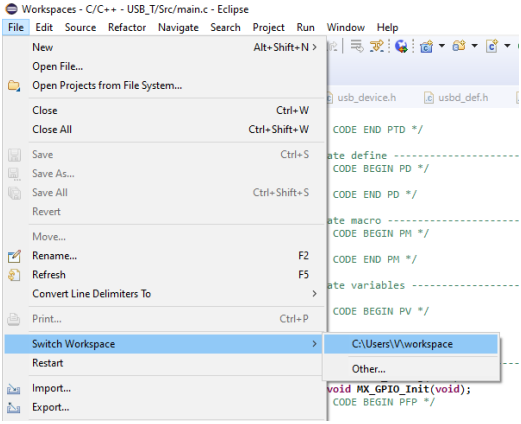



Рис. П 4.12 – Пункт меню, позволяющий сменить рабочее пространство среды разработки

Если оно не соответствует расположению проекта, определить его можно через меню **File → Switch Workspace → Other** (рисунок П 4.13).

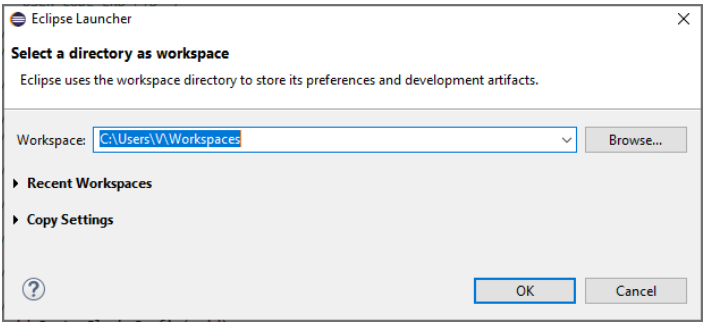



Рис. П 4.13 – Окно выбора рабочего пространства

Для передачи данных через USB используется функция CDC_Transmit_FS(), описанная в файле usbd_cdc_if.c, созданном STM32CubeMX при генерации проекта. Данный файл располагается в папке «Src». В этом же файле описана функция CDC_Receive_FS(), вызываемая по прерыванию при приеме данных.

В качестве флага получения данных используется переменная receiveBufLen, которую можно определить в соответствующем разделе файла usbd_cdc_if.c

```
/* USER CODE BEGIN PRIVATE_VARIABLES */
	uint32_t receiveBufLen = 0;
/* USER CODE END PRIVATE_VARIABLES */
```

Значение данной переменной можно изменять непосредственно в теле функции CDC_Receive_FS().

```
static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)
{
	/* USER CODE BEGIN 6 */
	USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
	USBD_CDC_ReceivePacket(&hUsbDeviceFS);

	receiveBufLen = *Len;

	return (USBD_OK);
	/* USER CODE END 6 */
}
```

В файле usbd_cdc_if.c определен также приемный буфер UserRxBufferFS. Чтобы к нему иметь доступ из main.c, необходимо

в заголовочном файле usbd_cdc_if.h в соответствующих разделах указать следующее:

```
/* USER CODE BEGIN EXPORTED_DEFINES */
#define APP_RX_DATA_SIZE  1000
/* USER CODE END EXPORTED_DEFINES */

/* USER CODE BEGIN EXPORTED_VARIABLES */
extern uint8_t UserRxBufferFS[APP_RX_DATA_SIZE];
extern uint32_t receiveBufLen;
/* USER CODE END EXPORTED_VARIABLES */,
```

а в файле main.c в соответствующем разделе указать включение файла usbd_cdc_if.h.

```
/* USER CODE BEGIN Includes */
#include "usbd_cdc_if.h"
/* USER CODE END Includes */.
```

Необходимо убрать из файла usbd_cdc_if.c продублированное определение размера буфера принимаемых данных APP_RX_DATA_SIZE.

Для передачи данных по USB и изменения скважности ШИМ-сигнала необходимо в файле main.c определить соответствующие переменные и запустить генерацию ШИМ.

```
 /* USER CODE BEGIN 2 */

 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);

 uint32_t pulse=0;
 int r = 1; // направление счета (1 - увеличение; 0 - снижение)

 char buf[25]; // передаваемая информация

 /* USER CODE END 2 */
```

Для решения поставленной задачи необходимо в бесконечно исполняемый цикл добавить следующий код, после чего скомпилировать проект.

```
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
        sprintf(buf,"pulse: [%ld]\n", pulse);
        CDC_Transmit_FS((uint8_t *)buf, strlen(buf));
        HAL_Delay(1);

        if (r == 1)
        {
            if (pulse < 399)
                pulse++;
            else
                r = 0;
        }
        else
        {
            if (pulse > 0)
                pulse--;
            else
                r = 1;
        }
        __HAL_TIM_SET_COMPARE(&htim4,
TIM_CHANNEL_1,pulse);
        HAL_Delay(1);

        if(receiveBufLen > 0)// если получены данные от ПК
        {
            HAL_Delay(250);

            switch(UserRxBufferFS[0]) // выбор определённого
действия
            {
                case '0':
                    HAL_TIM_PWM_Stop(&htim4,
TIM_CHANNEL_1);
                break;
                case '1':
```

```
                                HAL_TIM_PWM_Start(&htim4,
TIM_CHANNEL_1);
                        break;
                    }

                    receiveBufLen = 0; // сброс получения
            }
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Установка значения pulse выполняется функцией __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, pulse), третий параметр (является целочисленным). Таким образом, изменяя значение pulse, на линии PB6 будет формироваться ШИМ-сигнал с различными значениями скважности.

Все настройки выполнены, программный код написан. Далее необходимо скомпилировать проект и настроить конфигурацию запуска. Конфигурация запуска настраивается через меню **Run → Run Configurations** (рисунок П 4.14).

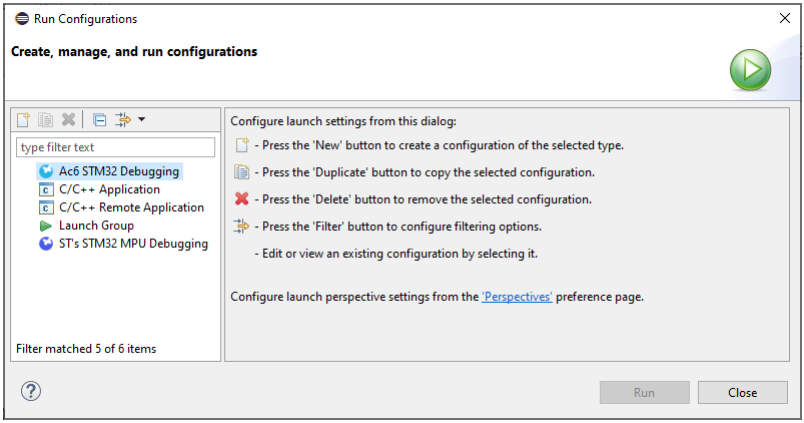



Рис. П 4.14 – Окно создания конфигурации запуска проекта

В появившемся окне необходимо дважды кликнуть левой кнопкой мыши на разделе «Ac6 STM32 Debugging». В результате сформируется новая конфигурация запуска программы. Если в

среде ведется одновременная разработка нескольких проектов, то в поле «Name:» можно указывать название созданной конфигурации для каждого проекта (рисунок П 4.15).

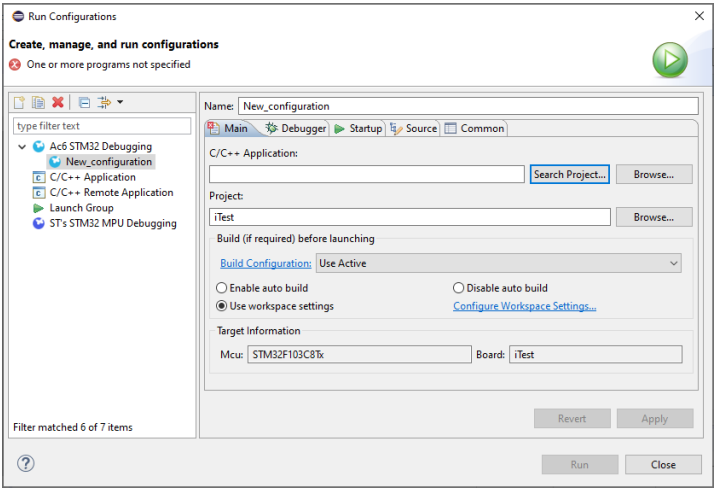



Рис. П 4.15 – Окно параметров конфигурации запуска проекта

По нажатию кнопки «Search Project…» во вкладке «Main» становится доступным выбор elf-файла проекта, сформированного в результате компиляции, который необходимо определить для созданной конфигурации запуска (рисунок П 4.16).

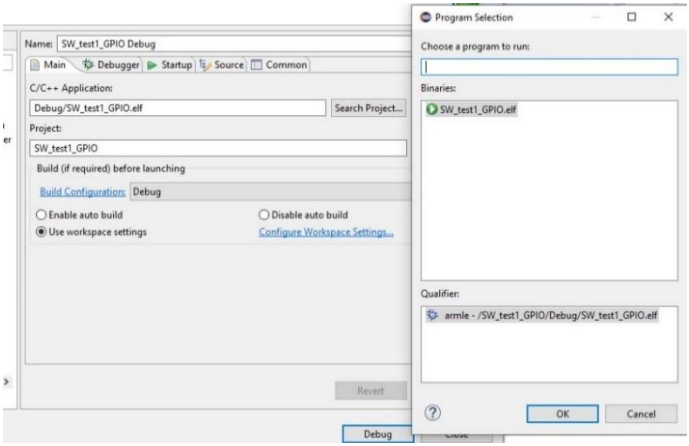



Рис. П 4.16 – Окно выбора *.elf файла проекта

Чтобы настроить программный сброс контроллера, на вкладке «Debugger», необходимо нажать кнопку «Show generator options…» (рисунок П 4.17).

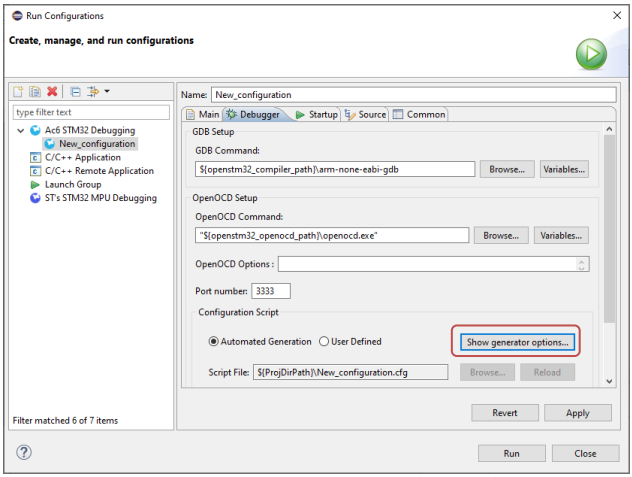



Рис. П 4.17 – Окно отображения дополнительных параметров

В появившейся области настройки «Reset Mode:» выбрать «Software system reset» и нажать кнопку «Run». Созданная программа запустится на микроконтроллере (рисунок П 4.18).

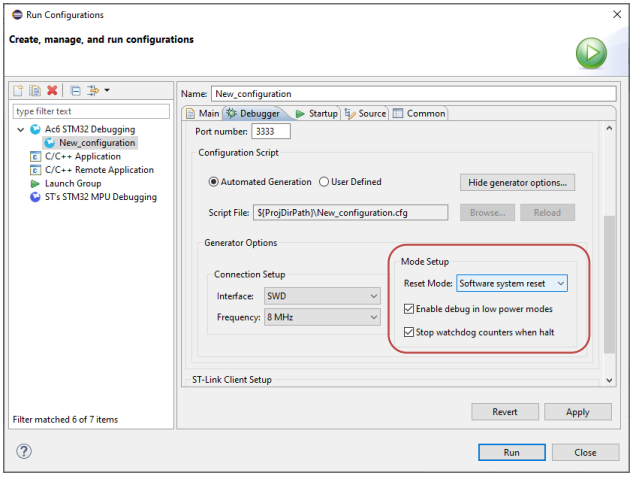



Рис. П 4.18 – Область настройки программного сброса контроллера

# ПРИЛОЖЕНИЕ 5. Программирование модуля индикации

Типовыми процедурами при выводе информации на LCD-дисплей являются

- вывод сообщения на русском и/или английском языках, используя ASCII-таблицу ПЗУ знакогенератора,
- формирование и вывод произвольного символа, отсутствующего в ASCII-таблице,
- анимация символов.

Разработку этих процедур иллюстрирует приложение 5.

Структура и программная модель модуля индикации приведена в разделе 3.2.2, а установка модуля на оценочной плате – на рисунке 3.5.

При выборе линий GPIO для подключения модуля индикации необходимо учитывать состав используемых периферийных устройств и наличие свободных толерантных входов (допускающие работу с напряжением 5 В).

Если не требуется высокое быстродействие, то не следует завышать как частоту тактового генератора, так и устанавливать большую скорость линий GPIO. Зависимость энергопотребления от тактовой частоты и количества включенных периферийных устройства приведены в таблицах 1.1, 1.7, 1.8.

В приведенном примере показано соединение модуля с блоком микроконтроллера и процедура инициализации в CUBE MX для 8-разрядной шин и высокого быстродействия. **Программа же написана для 4-разрядной синхронной шины, в которой D3-D0 не используются.**

Перед началом работы отладочного модуля необходимо настроить напряжение питания, установив соответствующие перемычки (раздел 3.1.5, рисунок 3.39) и подключить питание модуля индикатора (рисунок П 5.1). Значения напряжений можно проверить тестером.

Убедившись, что все перемычки выставлены правильно, система электропитания настроена на допустимые для микроконтроллера и дисплея номиналы напряжения, можно приступать к созданию проекта.

Программное обеспечение разрабатывается в STM32CubeIDE.

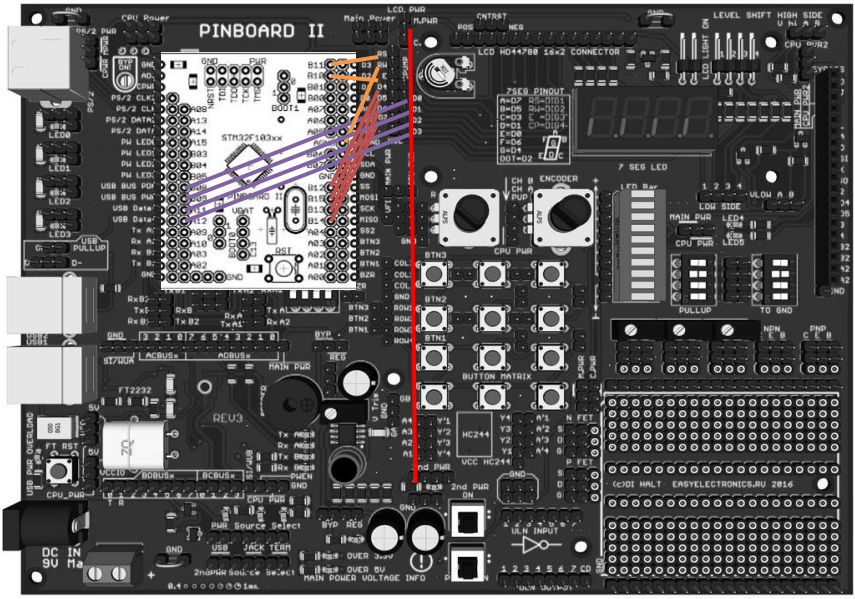



Рис. П 5.1 – Схема подключения модуля индикации

Необходимо запустить STM32CubeIDE и выбрать рабочее пространство (рисунок П 5.2), начать новый проект (в появившемся окне нажать на кнопку «Start new STM32 project») (рисунок П 5.3), задать используемый микроконтроллер STM32F103C8Tx (рисунок П 5.4).

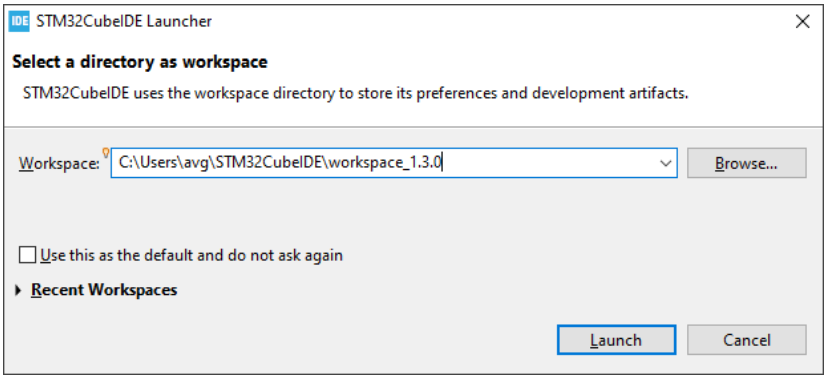



Рис. П 5.2 – Выбор рабочего пространства (workspace)

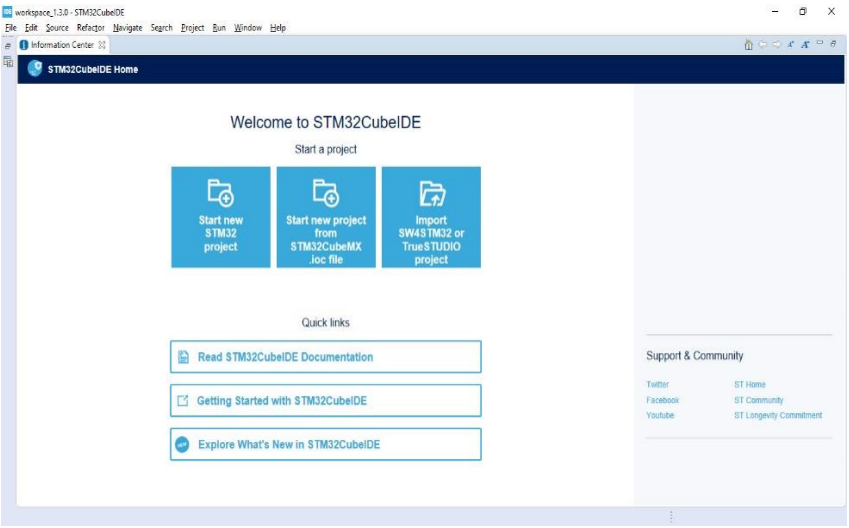



Рис. П 5.3 – Начало создания проекта

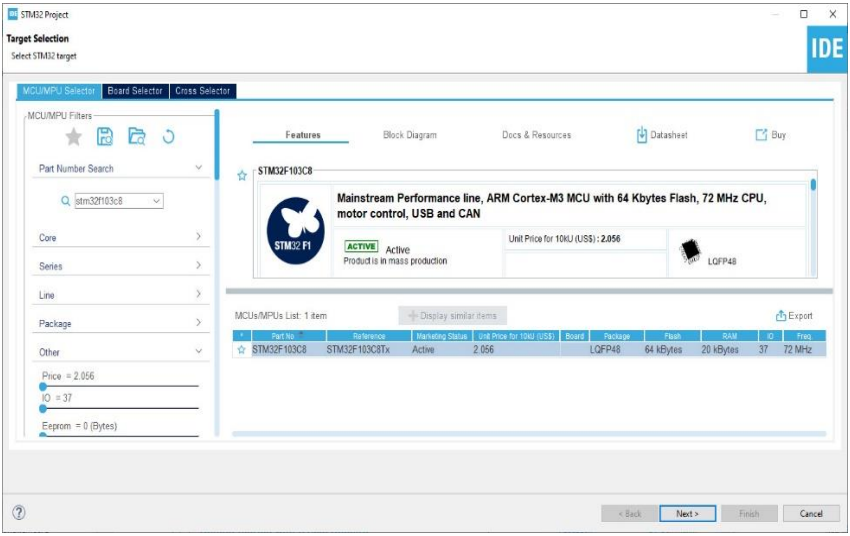



Рис. П 5.4 – Выбор микроконтроллера

Далее в мастере создания проекта необходимо задать параметры проекта и его опции (указать язык программирования и тип проекта, рисунок П 5.5).

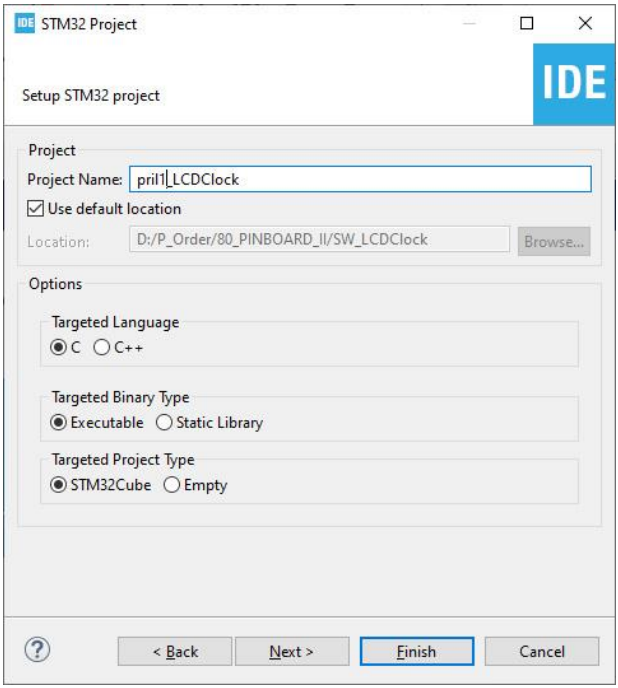



Рис. П 5.5 – Окно параметров проекта

Согласиться ассоциировать проект с STM32CubeMX, чтобы воспользоваться возможностями, предоставляемыми код-генератором (рисунок П 5.6).

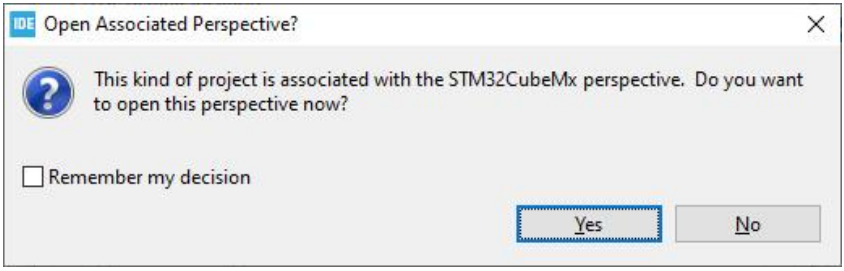



Рис. П 5.6 – Окно согласия открыть проект в STM32CubeMX

Созданный проект откроется в среде разработки. Проект готов к конфигурированию в код-генераторе (рисунок П 5.7).

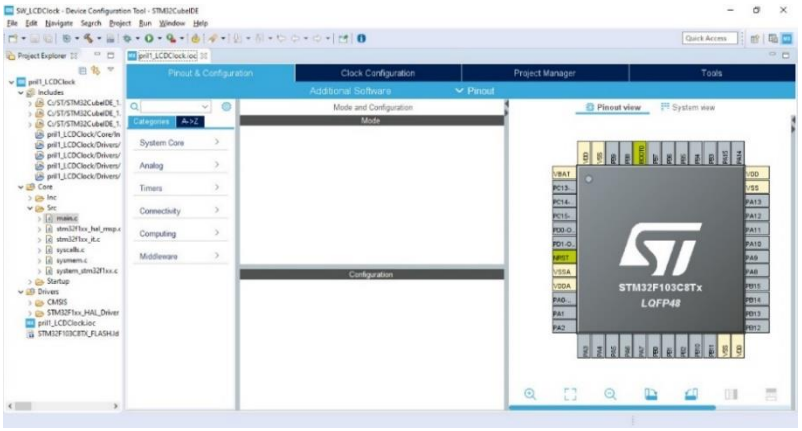



Рис. П 5.7 – Окно проекта в STM32CubeIDE

Для того, чтобы в последующем иметь возможность перепрограммировать микроконтроллер, необходимо выбрать режим отладки через последовательный интерфейс: в разделе **Pinout & Configuration → SYS → Mode** параметр «Debug» задать равным «Serial Wire» (рисунок П 5.8).

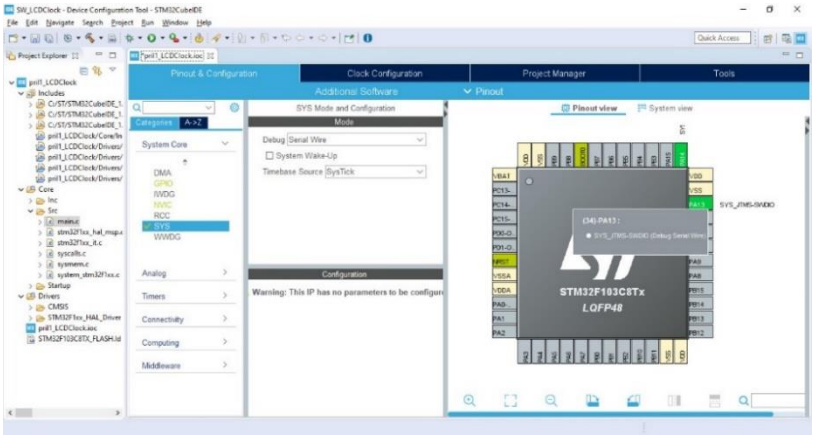



Рис. П 5.8 – Выбор параметров поддержки режима отладки

Следует задать тактирование от внешнего тактового генератора, в разделе **Pinout & Configuration → RCC → Mode** указав «High Speed Clock (HSE)» равным «Crystal/Ceramic Resonator») (рисунок П 5.9).

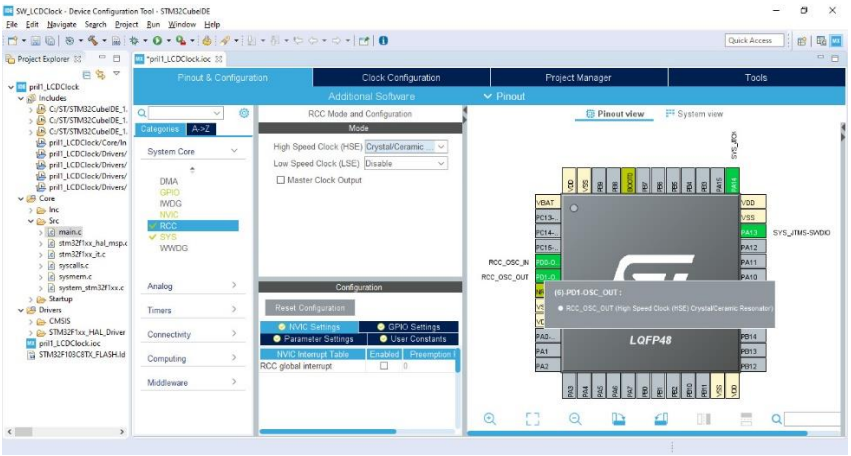



Рис. П 5.9 – Окно выбора режима тактирования от внешнего тактового генератора

Для работы с LCD-дисплеем выбрать входы/выходы контроллера (рисунок П 5.1). Необходимо помнить, что для работы с дисплеем нужно выбирать толерантные входы микроконтроллера.

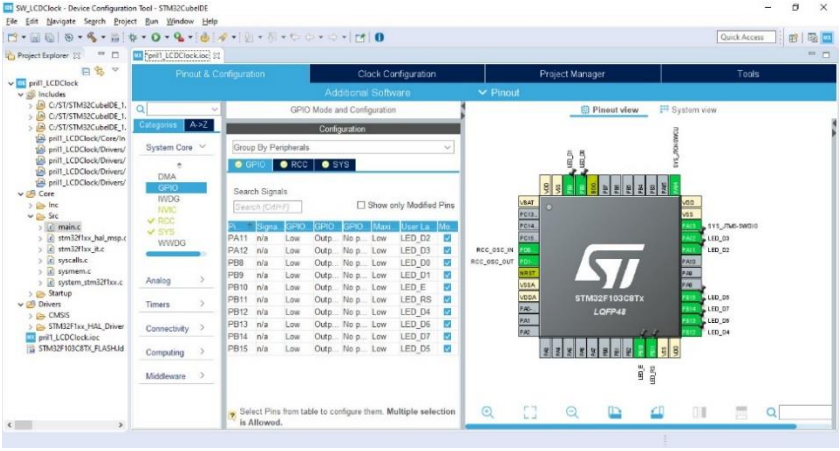



Рис. П 5.10 – Выбор и настройка линий портов GPIO

В качестве примера тактирование настраивается на максимальные частоты допустимые для микроконтроллера – 72 МГц. Параметры схемы тактирования показаны на рисунке П 5.11.

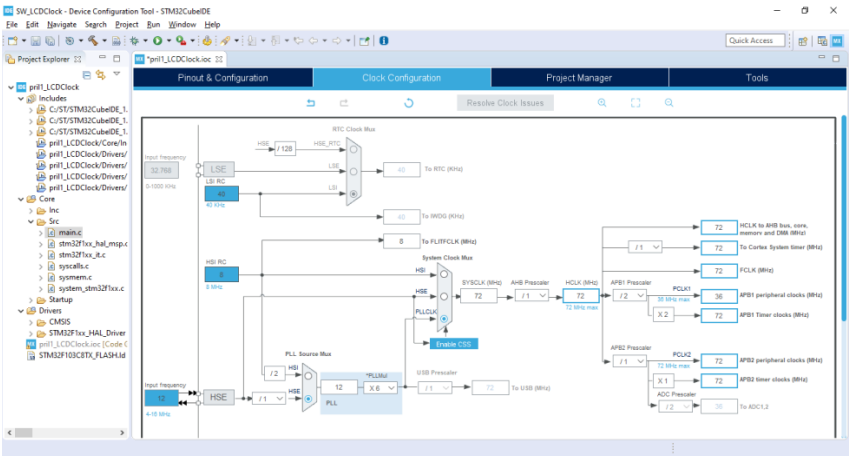



Рис. П 5.11 – Настройки системы тактирования

Для всех выходов настроим максимальную скорость работы (рисунок П 5.12).

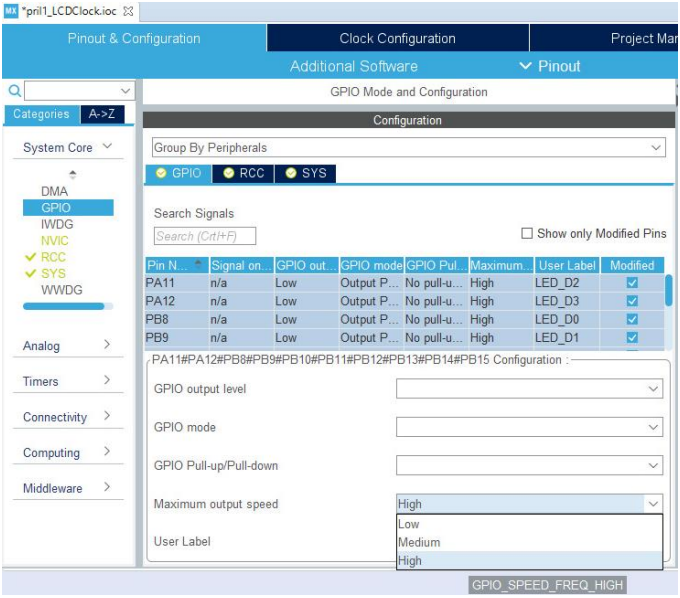



Рис. П 5.12 – Скорость работы выходов

Настроим параметры проекта (рисунок П5.13).

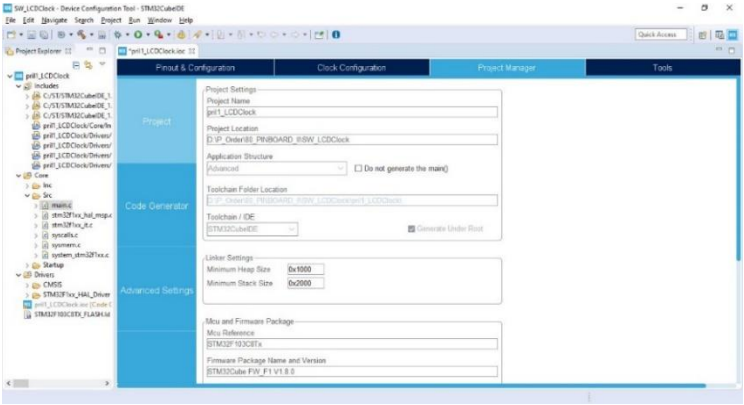



Рис. П 5.13 – Параметры проекта

Далее необходимо сгенерировать проект (рисунок П 5.14) и открыть файл main.c (рисунок П 5.15).

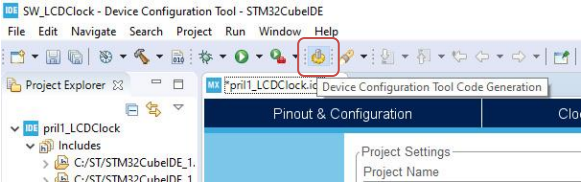



Рис. П 5.14 – Сгенерировать проект

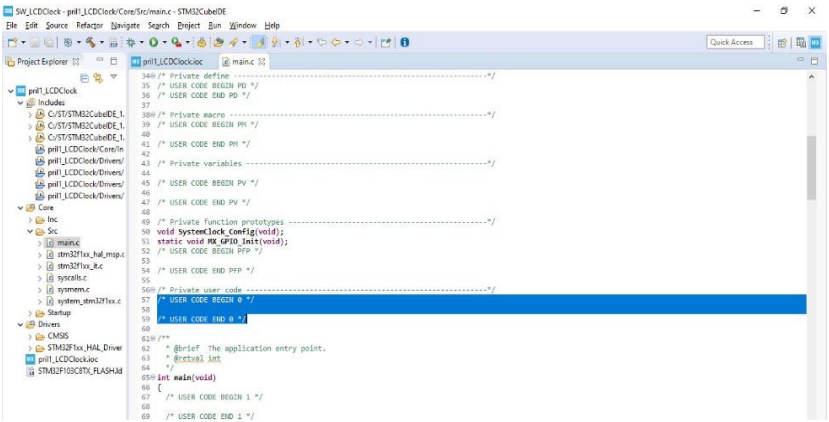



Рис. П 5.15 – Редактор файла main.c

**При использовании среды программирования STM32CubeIDE часто возникает проблема с выводом сообщений на русском языке.** Это связано с несовпадением кодировок. В среде STM32CubeIDE по умолчанию установлена кодировка UTF-8, а большинство программ и библиотек рассчитаны на работу с кодировкой CP1251. **Заметим, что такие проблемы не возникают при работе в SW4STM32.** В рассматриваемом проекте планируется использование кириллических символов, поэтому для их поддержки необходимо выполнить дополнительные настройки среды разработки [95] в окне персональных настроек.

Окно «Preferences» (персональных настроек) доступно через пункт меню **Window → Preferences** (см. рисунок П 5.16).

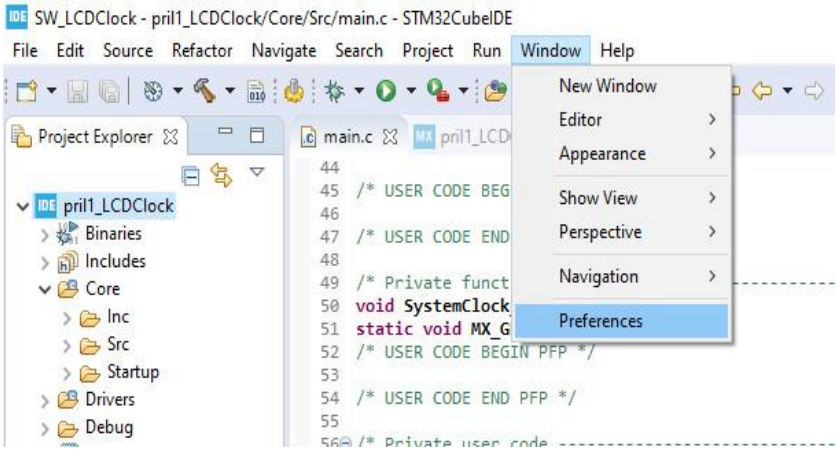



Рис. П 5.16 – Окно персональных настроек

Далее нужно в параметре **General → Workspace → Text file encoding** задать необходимую кодировку «CP1251», которую среда разработки будет использовать в текстовых файлах проекта (рисунок П 5.17). Для сохранения и применения изменений необходимо нажать на кнопку «Apply».

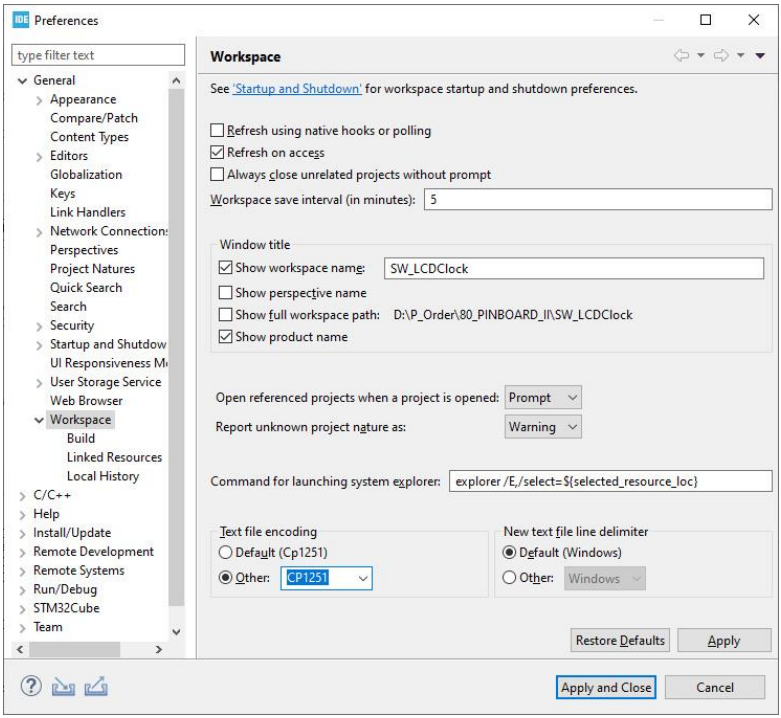



Рис. П 5.17 – Вызов окна персональных настроек

Перейдем к написанию программы. На этом этапе необходимо реализовать взаимодействие с дисплеем и обеспечить вывод произвольного сообщения на русском и английском языках (в том числе обеспечить вывод произвольного символа).

**Задача: инициализировать дисплей и подготовить к приему данных.**

1. Включить дисплей.
2. Очистить содержимое.
3. Сдвинуть курсор на одну позицию.
4. Вывести символ «1».

**Решение (последовательность команд):**

Сначала необходимо инициализировать дисплей – настроить взаимодействие с контроллером HD44780. Некоторые виды

контроллеров имеют состояние по умолчанию (шина 8 бит, курсор в 0), для них не требуется первоначальная инициализация, достаточно подать команду «включить дисплей». Однако, инициализацию рекомендуется делать всегда.

**001**11000 Шина 8 бит, 2 строки

**00000001** Очистка экрана

**00000**110 Инкремент адреса. Экран не движется

**00001**100 Включить дисплей (D=1)

**00000001** Очистить дисплей. Указатель установлен на DDRAM

**0001**0100 Сдвинуть курсор (S/C=0) вправо (R/L=1)

00110001 – записать данные (сигнал RS=1) код символа «1» равен 0x31

Жирным шрифтом выделен идентификатор команды, остальное – параметры команды по таблице 3.4 раздела 3.2.1.

В разделе «/* USER CODE BEGIN 0 */» (рисунок П 5.15) необходимо объявить структуру для работы с дисплеем, а также ряд определений:

```
/* USER CODE BEGIN 0 */
// структура для работы с контроллером HD44780
typedef struct {
        uint8_t DisplayControl;
        uint8_t DisplayFunction;
        uint8_t DisplayMode;
        uint8_t Rows;
        uint8_t Cols;
        uint8_t currentX;
        uint8_t currentY;
} HD44780_Options_t;

// переменная
static HD44780_Options_t HD44780_Opts;

// объявим команды
#define HD44780_CLEARDISPLAY      0x01
#define HD44780_RETURNHOME        0x02
```

```
#define HD44780_ENTRYMODESET       0x04
#define HD44780_DISPLAYCONTROL     0x08
#define HD44780_CURSORSHIFT        0x10
#define HD44780_FUNCTIONSET        0x20
#define HD44780_SETCGRAMADDR       0x40
#define HD44780_SETDDRAMADDR       0x80

// объявим флаги режимов дисплея
#define HD44780_ENTRYRIGHT         0x02
#define HD44780_ENTRYLEFT          0x00
#define HD44780_ENTRYSHIFTINCREMENT 0x00
#define HD44780_ENTRYSHIFTDECREMENT 0x01

// объявим флаги управления дисплеем и курсором
(включение/выключение)
#define HD44780_DISPLAYON          0x04
#define HD44780_CURSORON           0x02
#define HD44780_BLINKON            0x01

// флаги сдвига курсора/дисплея
#define HD44780_DISPLAYMOVE        0x08
#define HD44780_CURSORMOVE         0x00
#define HD44780_MOVERIGHT          0x04
#define HD44780_MOVELEFT           0x00

// флаги активации необходимого функционала, режима
отображения
#define HD44780_8BITMODE           0x10
#define HD44780_4BITMODE           0x00
#define HD44780_2LINE              0x08
#define HD44780_1LINE              0x00
#define HD44780_5x10DOTS           0x04
#define HD44780_5x7DOTS            0x00

// 4-х битный режим
// здесь переопределяются выходы контроллера, которые
используются для управления дисплеем
// проверить правильность подключения дисплея в соответствии со
```

```
схемой
// выходы управления
// RS - Register select pin
#ifndef HD44780_RS_PIN
#define HD44780_RS_PORT                    GPIOB
#define HD44780_RS_PIN                     GPIO_PIN_11
#endif
// E - Enable pin
#ifndef HD44780_E_PIN
#define HD44780_E_PORT                     GPIOB
#define HD44780_E_PIN                      GPIO_PIN_10
#endif
// выходы данных
// D4 - данные 4 бит
#ifndef HD44780_D4_PIN
#define HD44780_D4_PORT                    GPIOB
#define HD44780_D4_PIN                     GPIO_PIN_12
#endif
// D5 - данные 5 бит
#ifndef HD44780_D5_PIN
#define HD44780_D5_PORT                    GPIOB
#define HD44780_D5_PIN                     GPIO_PIN_15
#endif
// D6 - данные 6 бит
#ifndef HD44780_D6_PIN
#define HD44780_D6_PORT                    GPIOB
#define HD44780_D6_PIN                     GPIO_PIN_13
#endif
// D7 - данные 7 бит
#ifndef HD44780_D7_PIN
#define HD44780_D7_PORT                    GPIOB
#define HD44780_D7_PIN                     GPIO_PIN_14
#endif

// объявления для включения выхода порта
#define d4_set() HAL_GPIO_WritePin(HD44780_D4_PORT,
HD44780_D4_PIN, GPIO_PIN_SET)
#define d5_set() HAL_GPIO_WritePin(HD44780_D5_PORT,
```

```
HD44780_D5_PIN, GPIO_PIN_SET)
#define d6_set() HAL_GPIO_WritePin(HD44780_D6_PORT,
HD44780_D6_PIN, GPIO_PIN_SET)
#define d7_set() HAL_GPIO_WritePin(HD44780_D7_PORT,
HD44780_D7_PIN, GPIO_PIN_SET)

// объявления для выключения выхода порта
#define d4_reset() HAL_GPIO_WritePin(HD44780_D4_PORT,
HD44780_D4_PIN, GPIO_PIN_RESET)
#define d5_reset() HAL_GPIO_WritePin(HD44780_D5_PORT,
HD44780_D5_PIN, GPIO_PIN_RESET)
#define d6_reset() HAL_GPIO_WritePin(HD44780_D6_PORT,
HD44780_D6_PIN, GPIO_PIN_RESET)
#define d7_reset() HAL_GPIO_WritePin(HD44780_D7_PORT,
HD44780_D7_PIN, GPIO_PIN_RESET)

// объявления для включения/выключения управляющих линий
#define e1    HAL_GPIO_WritePin(HD44780_E_PORT,
HD44780_E_PIN, GPIO_PIN_SET) // установка линии E в 1
#define e0    HAL_GPIO_WritePin(HD44780_E_PORT,
HD44780_E_PIN, GPIO_PIN_RESET) // установка линии E в 0
#define rs1   HAL_GPIO_WritePin(HD44780_RS_PORT,
HD44780_RS_PIN, GPIO_PIN_SET) // установка линии RS в 1
(данные)
#define     rs0     HAL_GPIO_WritePin(HD44780_RS_PORT,
HD44780_RS_PIN, GPIO_PIN_RESET) // установка линии RS в 0
(команда)
```

В том же разделе напишем функцию инициализации дисплея в соответствии с алгоритмом инициализации 4-битного режима работы дисплея, показанном на рисунке 3.52.

```
// функция инициализации дисплея
void LCD_ini(void)
{
        // задержка HAL уровня в миллисекундах
        // ждать выход LCD дисплея на рабочий режим после
подключения напряжения VDD
```

```
	HAL_Delay(100);

	// указать дисплею, что на шину данных выставляется команда
	rs0;
	// при настройке рабочих параметров дисплея указать 8-и битный интерфейс
	// Внимание! Для применения данной настройки необходимо передать один байт в контроллер дисплея,
	//		этим объясняется использование здесь функуции записи данных "LCD_WriteData" при передаче команды.
	LCD_WriteData((HD44780_FUNCTIONSET | HD44780_8BITMODE) >> 4);

	// сформировать синхросигнал e1;
	delay(); // HAL_Delay формирует задержку мс. Для небольшой задержки можно использовать пустой цикл
	e0;
	// подождать, чтобы LCD-дисплей отработал команду
	HAL_Delay(4);

	// повторить команду согласно алгоритма инициализации
	rs0;
	LCD_WriteData((HD44780_FUNCTIONSET | HD44780_8BITMODE) >> 4);
	// сформировать строб записи команды
	e1;
	delay();
	e0;
	// задержка после команды
	HAL_Delay(2);

	// повторить команду
	rs0;
	LCD_WriteData((HD44780_FUNCTIONSET | HD44780_8BITMODE) >> 4);
	// сформировать строб записи команды
	e1;
```

```
        delay();
        e0;
        // задержка после команды
        HAL_Delay(2);

        // сформировать команду настройки рабочих параметров
дисплея (Function set)
        // указать, что дисплей использует:
        //                                          - 4-х битную
шину данных/адресов
        //                                          - обе строки
        //                                          - шрифт 5x7
пикселей
        LCD_Command(HD44780_FUNCTIONSET |
HD44780_4BITMODE | HD44780_2LINE | HD44780_5x7DOTS);
        HAL_Delay(2);

        // повторить посылку команды
        LCD_Command(HD44780_FUNCTIONSET |
HD44780_4BITMODE | HD44780_2LINE);
        HAL_Delay(2);

        // управление дисплеем: включить дисплей, включить
курсор подчеркивания и закрашивающий все знакоместо
        LCD_Command(HD44780_DISPLAYCONTROL |
HD44780_DISPLAYON | HD44780_CURSORON |
HD44780_BLINKON);
        HAL_Delay(2);

        // очистить дисплей
        LCD_Command(HD44780_CLEARDISPLAY);
        HAL_Delay(3);

        // настроить режим ввода (Entry Mode Set), при котором
после вывода очередного символа,
        // курсор автоматически сдвигается на одно знакоместо
вправо
        LCD_Command(HD44780_ENTRYMODESET |
```

```
HD44780_ENTRYRIGHT | HD44780_ENTRYSHIFTINCREMENT);
        HAL_Delay(2);

        // вернуть курсор в левый-верхний угол дисплея, память не очищать (Return Home)
        LCD_Command(HD44780_RETURNHOME);
        HAL_Delay(200);
}
```

Основные функции программы приведены ниже:

```
// задержка в несколько тактов микроконтроллера
void delay(void)
{
        uint16_t i;
        for(i=0;i<10;i++)
        {
                //...
        }
}
```

Также встречаются функции записи команд и данных.

```
// записать команду в контроллер LCD-дисплея
void LCD_Command(uint8_t dt)
{
        // указать дисплею, что на шину данных выставляется команда
        rs0;

        // передать старшую тетраду
        LCD_WriteData(dt>>4);
        delay();
        // сформировать строб записи команды
        e1;
        delay();
        e0;

        // передать младшую тетраду
        LCD_WriteData(dt);
        delay();
```

```
        // сформировать строб записи команды
        e1;
        delay();
        e0;
}

// записать данные в контроллер LCD-дисплея
void LCD_WriteData(uint8_t dt)
{
        // из данных, подготовленных для передачи, необходимо правильно выделить биты и сформировать соответствующие сигналы на шине данных
        if(((dt >> 3)&0x01)==1) {d7_set();} else {d7_reset();}
        if(((dt >> 2)&0x01)==1) {d6_set();} else {d6_reset();}
        if(((dt >> 1)&0x01)==1) {d5_set();} else {d5_reset();}
        if((dt&0x01)==1) {d4_set();} else {d4_reset();}
}
```

В разделе «*/* USER CODE BEGIN 2 */*» функции «*main()*» вызывается функция инициализации дисплея.

```
  /* USER CODE BEGIN 2 */

  // ожидание после подключения напряжения VDD к дисплею (включение LCD-дисплея)
  HAL_Delay(100);
  LCD_ini();
  HAL_Delay(100);
```

Вернемся в раздел «/* USER CODE BEGIN 0 */» и продолжим добавлять в конец раздела функции и переменные. Для вывода символов напишем реализацию функций:

```
// функция вывода символа
void LCD_SendChar(char ch)
{
        LCD_Data((uint8_t) ch);
}
```

```
// функция вывода байта
void LCD_SendByte(uint8_t bt)
{
	LCD_Data(bt);
}
```

Для вывода сообщений на русском языке используется массив и функция:

```
// ASCII таблица соответствия кириллических символов (русский язык)
const uint8_t russian[]={ 0x41, 0xA0, 0x42, 0xA1, 0xE0, 0x45,
0xA3, 0xA4, 0xA5,0xA6, 0x4B, 0xA7, 0x4D, 0x48, 0x4F, 0xA8,
0x50,0x43,
0x54, 0xA9, 0xAA, 0x58, 0xE1, 0xAB, 0xAC, 0xE2, 0xAD,0xAE,
0x62,
0xAF, 0xB0, 0xB1, 0x61, 0xB2, 0xB3, 0xB4, 0xE3, 0x65, 0xB6, 0xB7,
0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0x6F, 0xBE, 0x70,
0x63,0xBF,
0x79, 0xE4, 0x78, 0xE5, 0xC0, 0xC1, 0xE6, 0xC2,0xC3, 0xC4, 0xC5,
0xC6, 0xC7 };

// вывод строки с кириллическими символами (русский язык)
void LCD_Write_Str(uint8_t *STRING)
{
	// очередной символ строки
	char c;
	while (c=*STRING++)
	{
		if(c >= 192)
			LCD_SendChar(russian[c-192]);
		else
			LCD_SendChar(c);
	}
}
```

Вывод данных осуществляется с помощью функции:

```
// функция вывода данных
void LCD_Data(uint8_t dt)
{
	// передать старшую тетраду
	rs1;
	LCD_WriteData(dt>>4);
	HAL_Delay(30);
	// строб записи
	e1;
	delay();
	e0;

	// передать младшую тетраду
	LCD_WriteData(dt);
	HAL_Delay(30);
	// строб записи
	e1;
	delay();
	e0;
}
```

Команда очистки дисплея:

```
// команда очистки дисплея
void LCD_Clear(void)
{
	//LCD_Command(0x01);
	LCD_Command(HD44780_CLEARDISPLAY);
	HAL_Delay(2);
}
```

Для вывода символов в определенную позицию необходимо создать функцию, позволяющую выводить символы в видеопамять ВП (DDRAM ) с заданным адресом. Первая строка начинается с адреса 0x00, вторая – с адреса 0x40. Очередной символ будет выводиться со смещением относительно начала строки. Функция позиционирования (столбец, строка) имеет вид:

```
// установить курсор в заданную позицию
```

```
void LCD_SetPos(uint8_t x, uint8_t y)
{
	switch(y) {
		case 0:
			LCD_Command(x |
HD44780_SETDDRAMADDR);
			HAL_Delay(1);
			break;

		case 1:
			LCD_Command((0x40+x) |
HD44780_SETDDRAMADDR);
			HAL_Delay(1);
			break;
	}
}
```

Теперь можно вывести строку на экран.

```
 /* USER CODE BEGIN 3 */
		LCD_Write_Str("Здравствуйте!!!");
		HAL_Delay(200);
		LCD_SetPos(2, 1);
		HAL_Delay(200);
		LCD_Write_Str("fors.gtechs.ru");

		HAL_Delay(2000);
		LCD_Clear();
```

Подключим отладочный модуль через программатор к компьютеру. Схема подключения аналогична приведенной ранее на рисунке П 5.18.

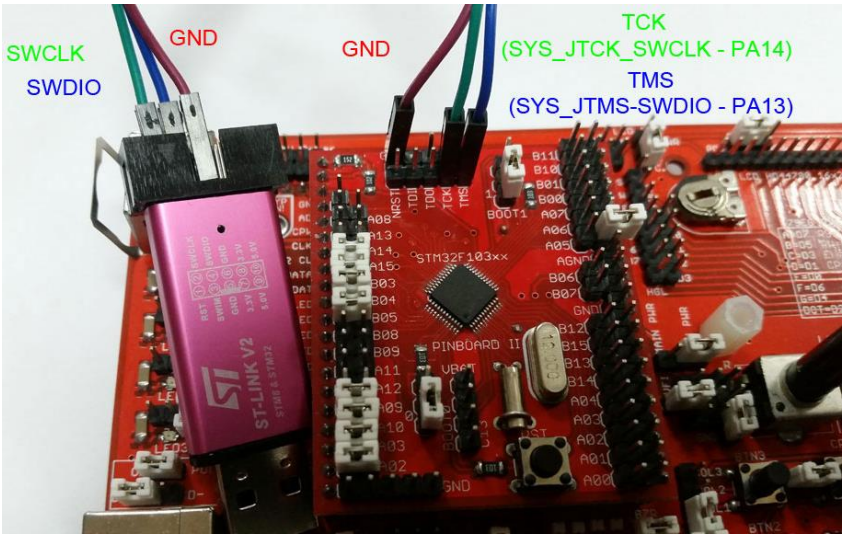



Рис. П 5.18 – Подключение программатора ST-LINK V2

Для того, чтобы иметь возможность запрограммировать контроллер, необходимо с помощью кабеля USB подключиться к компьютеру, все драйверы должны установиться автоматически. В диспетчере устройств должно отобразиться подключенное устройство (раздел 3.1.3, рисунок 3.13).

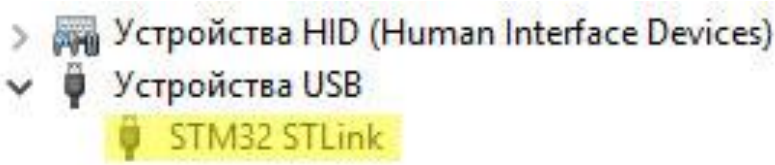



Рис. П 5.19 – Подключение адаптера ST-LINK V2

Если что-то пошло не так, см. раздел 3.1.3.

Далее необходимо скомпилировать проект (рисунок П 5.20).

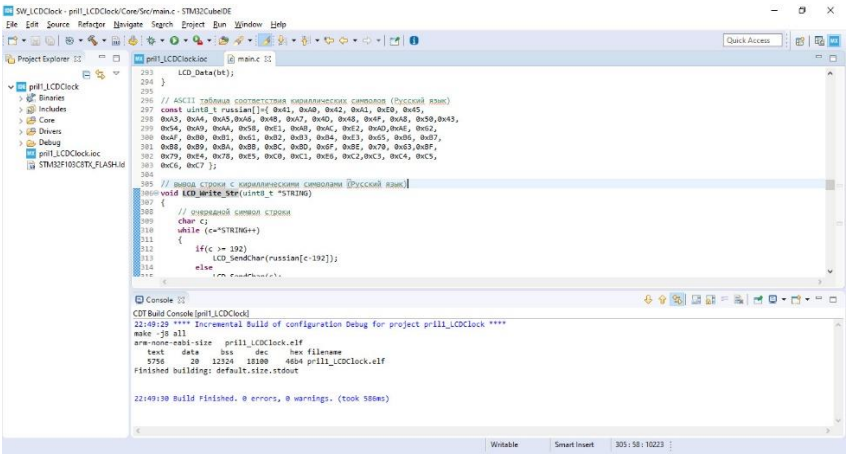


Рис. П 5.20 – Компиляция проекта

Проект скомпилировался успешно, ошибок не обнаружено. Скомпилированный файл **pril1_LCDClock.elf** можно зашить в микроконтроллер с помощью утилиты STM32 ST-LINK Utility или прошить и запустить на исполнение или отладку в STM32CubeIDE. Для запуска проекта в STM32CubeIDE следует нажать кнопку «Run» в панели управления (рисунок П 5.21).

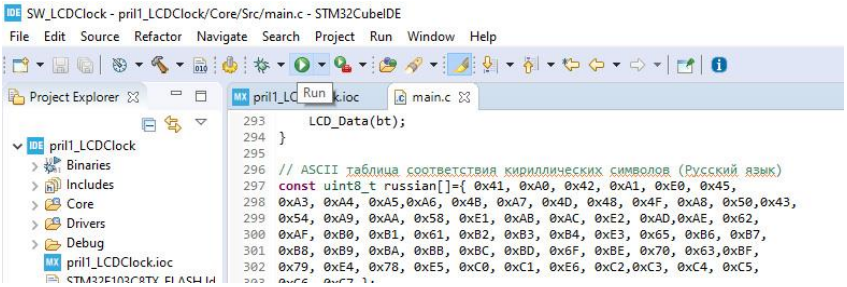



Рис. П 5.21 – Запуск проекта «Run»

Откроется окно редактирования конфигурации «Edit Configuration» (рисунок П 5.22).

По умолчанию предполагается, что программирование микроконтроллера осуществляется при нажатой кнопке «RST» на модуле микроконтроллера.

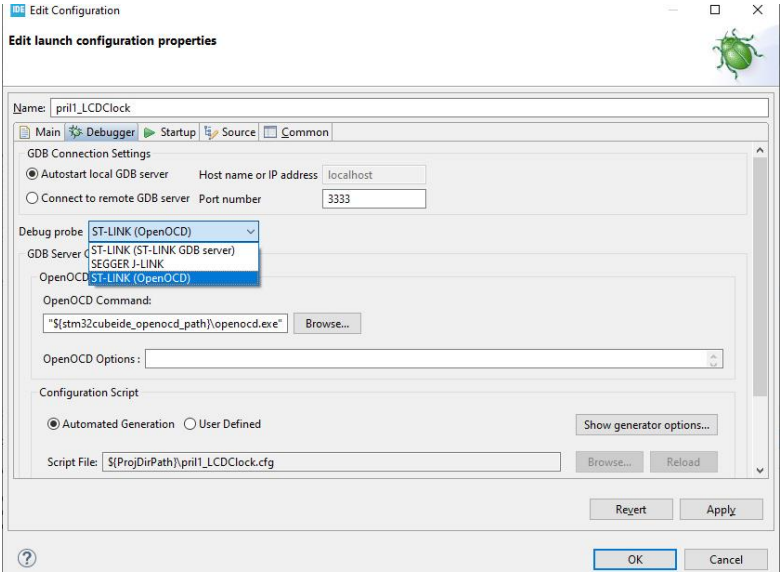



Рис. П 5.22 – Окно редактирования конфигурации

Изменить конфигурацию, чтобы микроконтроллер сбрасывался программно, можно нажав в разделе «Configuration Script» на кнопку «Show generation options…» и задав параметру «Reset Mode» значение «Software system reset» (рисунок П 5.23).

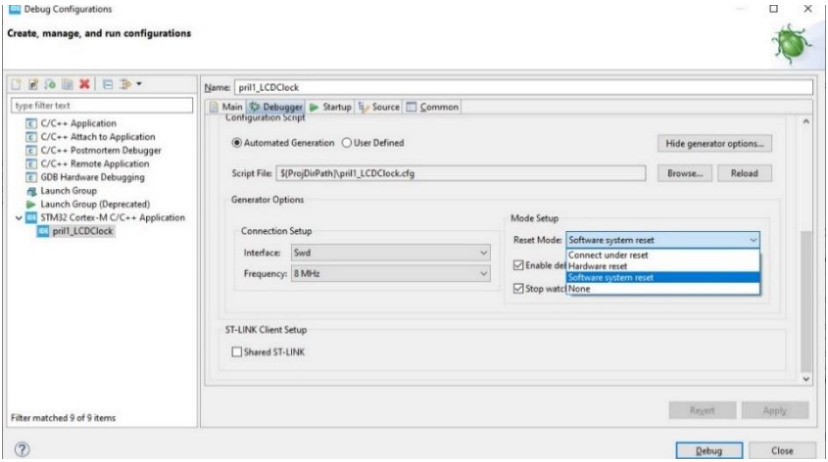



Рис. П 5.23 – Настройка программного сброса микроконтроллера

Программа записывается в микроконтроллер и запускается на исполнение.

Если запустить проект на отладку, нажав кнопку «Debug» в панели управления  , то появится диалоговое окно. Необходимо нажать «Switch» (рисунок П 5.24) и дать свое согласие переключиться в режим отладки.

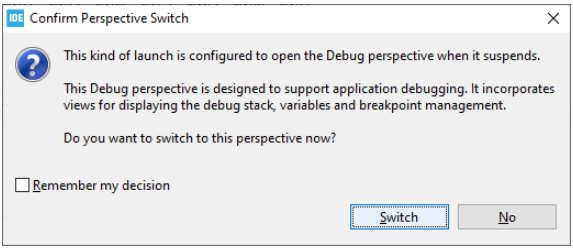



Рис. П 5.24 – Переключиться в режим отладки

Среда разработки переключится в режим отладки, а процесс отладки остановится на первой строке программы (рисунок П5.25).

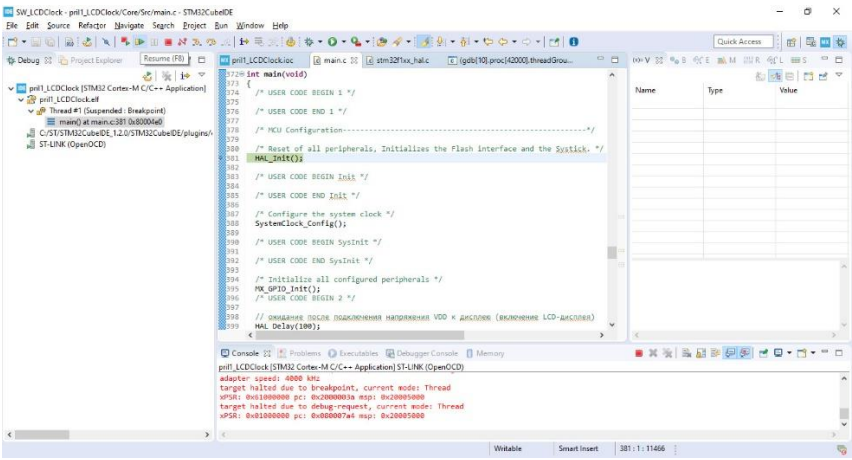


Рис. П 5.25 – Среда в режиме отладки

Кнопка «Resume» (F8) запускает проект на исполнение в режиме отладки и, если программа написана правильно, на LCD дисплее отображается сообщение.

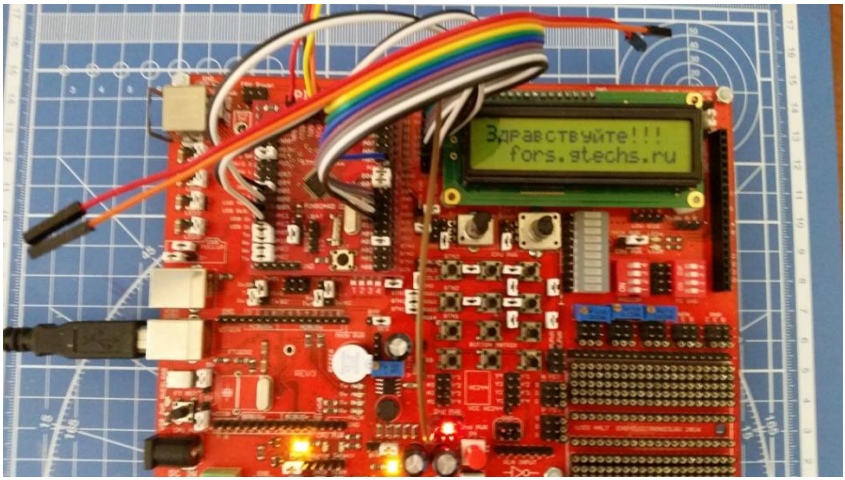



Рис. П 5.26 – Результат вывода сообщения на дисплей

**Следующая задача: создать произвольный символ «√» с кодом 01 и вывести его на экран.**

Считаем, что дисплей у нас уже инициализирован и готов к приему данных.

**Решение (последовательность команд):**

**01**001000 Выбрать в видеопамяти CGRAM адрес 0x08 – начало второго символа (важно обратить внимание, что один символ занимает 8 байт)

Для того, чтобы нарисовать значок «галочка – √», необходимо использовать младшие 5 бит, старшие три бита не используются:

00000001 Это пошли 8 байт данных (RS=1)
00000001
00000001
00000010
00010010
00010010
00001100
00001100 Последний байт, кодирующий произвольный символ

**1**0000000 Команда переключения адреса на DDRAM и указатель на адрес 0000000 – первый символ в первой строке.

00000001 Теперь запишем наш символ на экран (RS=1), код 01 – это номер произвольного символа, нарисованного символа, который необходимо отобразить.

Умение выводить произвольные символы позволяет создавать интересный интерфейс пользователя системы. Примеры показаны в [96].

Для реализации этой задачи в раздел «/* USER CODE BEGIN 0 */» необходимо добавить заготовки символов и анимации, а также реализацию функции вывода произвольного символа:

```
// заготовки символов и анимации
// батарейка, зарядка
int8_t Buttery[6][8] = {
                    {0x0E,0x1F,0x11,0x11,0x11,0x11,0x11,0x1F},
                    {0x0E,0x1F,0x11,0x11,0x11,0x11,0x1F,0x1F},
                    {0x0E,0x1F,0x11,0x11,0x11,0x1F,0x1F,0x1F},
                    {0x0E,0x1F,0x11,0x11,0x1F,0x1F,0x1F,0x1F},
                    {0x0E,0x1F,0x11,0x1F,0x1F,0x1F,0x1F,0x1F},
                    {0x0E,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F}
};
// линия прогресса вид 1
int8_t progress_bar1[7][8] = {
          {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1F},
          {0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0x1F},
          {0x00,0x00,0x00,0x00,0x00,0x1F,0x1F,0x1F},
          {0x00,0x00,0x00,0x00,0x1F,0x1F,0x1F,0x1F},
          {0x00,0x00,0x00,0x1F,0x1F,0x1F,0x1F,0x1F},
          {0x00,0x00,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F},
          {0x00,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F}
};

// линия прогресса вид 2
int8_t progress_bar2[4][8] = {
          {0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10},
          {0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18},
          {0x1C,0x1C,0x1C,0x1C,0x1C,0x1C,0x1C,0x1C},
          {0x1E,0x1E,0x1E,0x1E,0x1E,0x1E,0x1E,0x1E}
};
```

```
// беспроводная сеть
int8_t Wireless[3][8] = {
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x10},
        {0x00,0x00,0x00,0x00,0x00,0x18,0x04,0x14},
        {0x00,0x00,0x00,0x1C,0x02,0x19,0x05,0x15},
};

// стрелочка вверх
int8_t Strelochka_Up[8][8] = {
        {0x00,0x00,0x00,0x04,0x0E,0x1F,0x0E,0x0E},
        {0x00,0x00,0x04,0x0E,0x1F,0x0E,0x0E,0x00},
        {0x00,0x04,0x0E,0x1F,0x0E,0x0E,0x00,0x00},
        {0x04,0x0E,0x1F,0x0E,0x0E,0x00,0x00,0x00},
        {0x0E,0x1F,0x0E,0x0E,0x00,0x00,0x00,0x00},
        {0x1F,0x0E,0x0E,0x00,0x00,0x00,0x00,0x00},
        {0x0E,0x0E,0x00,0x00,0x00,0x00,0x00,0x00},
        {0x0E,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
};

// стрелочка вниз
int8_t Strelochka_Down[8][8] = {
        {0x0E,0x0E,0x1F,0x0E,0x04,0x00,0x00,0x00},
        {0x00,0x0E,0x0E,0x1F,0x0E,0x04,0x00,0x00},
        {0x00,0x00,0x0E,0x0E,0x1F,0x0E,0x04,0x00},
        {0x00,0x00,0x00,0x0E,0x0E,0x1F,0x0E,0x04},
        {0x00,0x00,0x00,0x00,0x0E,0x0E,0x1F,0x0E},
        {0x00,0x00,0x00,0x00,0x00,0x0E,0x0E,0x1F},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x0E},
        {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0E},
};

// градусы
int8_t Degrees[8] = {0x06,0x09,0x09,0x06,0x00,0x00,0x00,0x00};

// Вывод произвольного символа, анимации
// img - номер картинки/анимации из набора
void LCD_LoadCustomCharAnimation(uint8_t img)
{
```

```
	uint8_t lineOfChar = 0;

	// записать пользовательские символы, используемые в анимации, в контроллер дисплея
	for (uint8_t j = 0; j < 8; j++)
	{// пробежаться по всем символам

		// Выбрать в CGRAM адрес очередного символа.
		// Обратите внимание, один символ занимает 8 байт, в каждой строке младшие 5бит определяют как будет выглядеть символ.
		LCD_Command(HD44780_SETCGRAMADDR | (j * 8));
		HAL_Delay(100);

		for (uint8_t i = 0; i < 8; i++)
		{// пробежаться по всем строкам символов

			switch(img) {
				case 0: // батарейка, зарядка
					if (j < 6)
						lineOfChar = Buttery[j][i];
					else
						lineOfChar = Buttery[0][i];
					break;

				case 1: // линия прогресса вид 1
					if (j < 7)
						lineOfChar = progress_bar1[j][i];
					else
						lineOfChar = progress_bar1[0][i];
					break;

				case 2: // линия прогресса вид 2
```

```
                    if (j < 4)
                        lineOfChar =
progress_bar2[j][i];
                    else
                        lineOfChar =
progress_bar2[0][i];
                    break;

                case 3:// беспроводная сеть
                    if (j < 3)
                        lineOfChar =
Wireless[j][i];
                    else
                        lineOfChar =
Wireless[0][i];
                    break;

                case 4: // стрелочка вверх
                    if (j < 8)
                        lineOfChar =
Strelochka_Up[j][i];
                    else
                        lineOfChar =
Strelochka_Up[0][i];
                    break;

                case 5: // стрелочка вниз
                    if (j < 6)
                        lineOfChar =
Strelochka_Down[j][i];
                    else
                        lineOfChar =
Strelochka_Down[0][i];
                    break;

                case 6: // градусы
                    lineOfChar = Degrees[i];
                    break;
```

```
                }

                LCD_SendByte(lineOfChar); HAL_Delay(10);
            }
        }

        // переключить адреса на DDRAM и установить указатель на адрес 0000000 - первый символ в первой строке
        LCD_Command(HD44780_SETDDRAMADDR | 0b00000000);
        HAL_Delay(20);
}
/* USER CODE END 0 */
```

Перейдем в конец раздела «/* USER CODE BEGIN 3 */» в основном цикле и добавим к существующему коду смену сообщения, вывод произвольных символов и анимацию.

```
        // Вывести произвольный символ на экран
        // выбрать в CGRAM адрес 0x08 - начало второго символа (обратите внимание, что один символ занимает 8 байт)
        LCD_Command(HD44780_SETCGRAMADDR | 0b001000);
        HAL_Delay(200);
        // нарисовать значок «галочка», нас интересуют младшие 5 бит, старшие три бита не используются:
        LCD_SendByte(0b00000001); HAL_Delay(10);
        LCD_SendByte(0b00000001); HAL_Delay(10);
        LCD_SendByte(0b00000001); HAL_Delay(10);
        LCD_SendByte(0b00000010); HAL_Delay(10);
        LCD_SendByte(0b00000010); HAL_Delay(10);
        LCD_SendByte(0b00010010); HAL_Delay(10);
        LCD_SendByte(0b00001100); HAL_Delay(10);
        LCD_SendByte(0b00001100); HAL_Delay(10);
        // переключить адреса на DDRAM и установить указатель на адрес 0000000 - первый символ в первой строке
        LCD_Command(HD44780_SETDDRAMADDR | 0b00000000);
        HAL_Delay(20);
        // вывести произвольный символ на экран (RS=1), код 01 – номер произвольного символа, который мы нарисовали сами
```

```
        LCD_SendByte(0b00000001);
        HAL_Delay(20);

        // задержка
        HAL_Delay(2000);
        LCD_Clear();

        // выключить курсор
        LCD_Command(HD44780_DISPLAYCONTROL |
HD44780_DISPLAYON);
        HAL_Delay(2);

        for (uint8_t j = 0; j < 6; j++)
        {
                // Вывести анимацию из пользовательских
символов на экран
                // загрузить анимацию, параметр функции может
принимать значения
                //              0. батарейка, зарядка
                //              1. линия прогресса вид 1
                //              2. линия прогресса вид 2
                //              3. беспроводная сеть
                //              4. стрелочка вверх
                //              5. стрелочка вниз
                //              6. градусы
                LCD_LoadCustomCharAnimation(j);

                HAL_Delay(10);

                // анимация
                for (int i = 0; i < 8; i++)
                {
                        LCD_SetPos(j, 0);
                        HAL_Delay(10);
                        LCD_SendByte(i);
                        HAL_Delay(500);
                }
                HAL_Delay(20);
```

```
        }

        // включить дисплей и курсор
        LCD_Command(HD44780_DISPLAYCONTROL |
HD44780_DISPLAYON | HD44780_CURSORON |
HD44780_BLINKON);
        HAL_Delay(2);
        // очистить экран
        LCD_Clear();
  }
  /* USER CODE END 3 */
```

Кнопка «Resume» (F8) запускает проект на исполнение в режиме отладки и, если программа написана правильно и нет ошибок, на LCD дисплее отображается сообщение, а также произвольные символы, реализуется вывод анимированных символов (рисунок П 5.27).

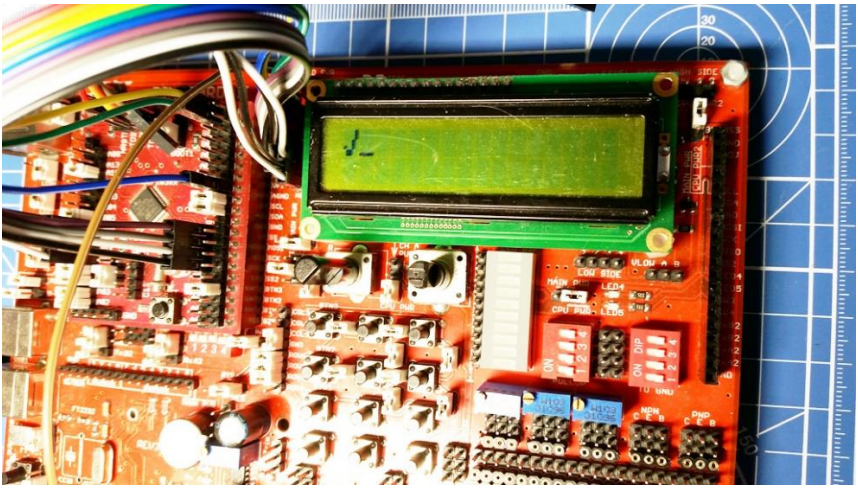


Рис. П 5.27 – Результат вывода произвольного символа на экран

# ПРИЛОЖЕНИЕ 6. Защита флеш-памяти от копирования

Предположим, микроконтроллерное устройство создано и необходимо его выпустить на рынок. Есть опасения того, что кто-то начнёт его копировать. Для защиты от данной угрозы достаточно подключиться программатором к устройству и считать прошивку через утилиту ST-Link Utility **Target ⇒ Connect** (рисунок П 6.1) **[93]**.

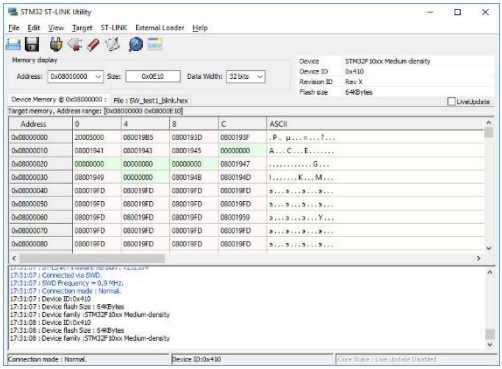



Рис. П 6.1 – Интерфейс ST-Link Utility

Разработчики ST Microelectronics для всех МК реализовали систему защиты (read out protection). Суть её проста – если в специальном регистре (Option bytes) установлено определённое значение, то возможность отладки и считывания прошивки отключается (рисунок П 6.2).

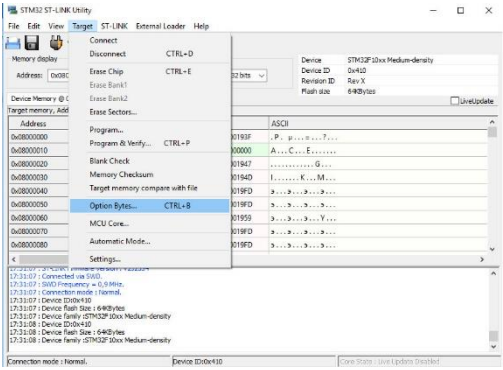



Рис. П 6.2 – Сменить значение в регистре Option bytes

В этом режиме пропадёт возможность изменения прошивки МК. Защиту можно отключить, поменяв значение в регистре Option bytes, однако в таком случае память программы будет стерта, а значит её никто не сможет скопировать.

Поменять значение в регистре можно при помощи ST-Link Utility, **Target ⇒ Option Bytes… ⇒ Read out protection ⇒ ENABLE** (рисунок П 6.2, рисунок П 6.3).

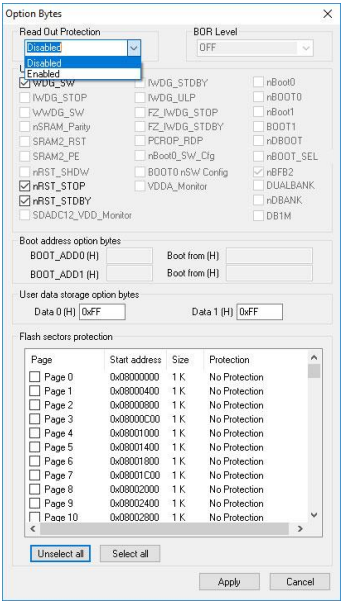



Рис. П 6.3 – Включение/отключение защиты микроконтроллера

Показанный способ включения/отключения защиты микроконтроллера не удобен в случаях, когда устройств много. В этом случае для реализации защиты удобно воспользоваться стандартной библиотекой периферии, а именно «***stm32f10x_flash.c***». Эта библиотека содержит необходимую функцию. Регистр и работа с флеш-памятью описана в документе ***PM0063 Programming manual STM32F100xx value line Flash programming [94]***.

При реализации используется функция защиты от многократного включения заголовочных файлов. Полезны такие директивы препроцессора как:

- **#define «идентификатор» «аргумент»** – директива указывает препроцессору на необходимость заменять последовательность символов «идентификатор» в файле на «аргумент». Однако, даже если идентификатор в тексте не встречается, он остаётся в системе и может быть проверен другими директивами. Используя данную особенность можно написать условие, при котором определённый код будет выполняться лишь в случаях, когда идентификатор объявлен.
- **#ifdef «идентификатор»** – директива, проверяющая наличие объявления идентификатора.
- **#ifndef «идентификатор»** – директива, проверяющая отсутствие определения идентификатора.
- **#endif** – директива, указывающая конец условия.

Создать идентификатор, который будет говорить, что данная сборка финальная, можно с помощью условных директив предкомпилятора. Пока строка закомментирована, код помещённый между #ifdef и #endif выполняться не будет.

```
//#define RELEASE
```

При конечной сборке достаточно просто убрать комментарий со строки и перепрошить устройство.

Поместим условие в функцию Init_MCU():

```
#ifdef RELEASE
	// здесь код пользователя
#endif
```

Подготовительный этап закончился, перейдем к сути. Мы уже разбирались как работать с внутренней флеш-памятью микроконтроллера. Перед записью битов защиты необходимо разблокировать доступ к памяти, затем необходимо произвести нужные операции и снова заблокировать доступ:

```
#ifdef RELEASE
#warning "Protection is ON. Debug is OFF"
	if (FLASH_GetReadOutProtectionStatus() == RESET) {
		FLASH_Unlock();
		FLASH_ReadOutProtection(ENABLE);
		FLASH_Lock();
	}
#endif
```

# ПРИЛОЖЕНИЕ 7. Порядок выполнения работ

1. Получить задание у преподавателя.
2. В соответствии с заданием выбрать необходимые периферийные устройства, изучить их структуру, особенности применения и программную модель.
3. Разработать алгоритмы ввода/вывода и обработки данных
4. Определить ресурсы микроконтроллера, которые планируется использовать.
   - Выбрать режимы работы периферийных модулей и модуля микроконтроллера.
   - Если на плате отладочного модуля периферийные устройства отсутствуют, подключить их через расширитель внутренней шины.
   - Обратить внимание на возможные конфликты, которые могут возникнуть при подключении периферийных устройств к портам МК. В случае необходимости использовать ремапинг.
   - Разработать схему соединений периферийных устройств и БМК.
5. **При выключенном напряжении питания** собрать схему с помощью шнуровой коммутации и/или перемычек.
6. Проверить правильность установки битов управления загрузкой контроллера BOOT0, BOOT1, расположенных на плате МК.
7. Подключить программатор.
8. Получить разрешение преподавателя на включение оценочного модуля.
9. С помощью утилиты STM32 ST-LINK Utility проверить доступность устройства через программатор.
10. Создать проект в код-генераторе STM32CubeMX, произвести настройку проекта и параметров микроконтроллера. Выбрать интегрированную среду разработки SW4STM32. Сгенерировать проект и открыть его в IDE.
11. Разработать программное обеспечение отладочного модуля в соответствии с заданием. В процессе отладки предусмотреть возможность визуального контроля промежуточных и конечных результатов работы программы. При отладке программы следует использовать весь доступный функционал интегрированной среды разработки и вспомогательных утилит.

12. Сдать преподавателю программу, работающую в соответствии с заданием.

13. Особенность отладки в IDE Proteus изложенны в разделе 2.1.4.

14. Оформить отчет.

**Содержание отчета**

1. Задание.

2. Алгоритмы обработки в виде граф-схемы или словесного описания.

3. Используемые ресурсы МК и обоснование выбранных режимов работы периферийных устройств.

4. Схема соединения модулей системы.

5. Основные экранные формы инициализации используемых периферийных устройств (режимы работы, параметры и так далее).

6. Листинг программ с комментариями на русском языке.

## Для заметок

Учебное издание

***Иоффе Владислав Германович,***
***Графкин Алексей Викторович,***
***Графкин Владимир Викторович***

**АРХИТЕКТУРА,
ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ
И ПРОГРАММНЫЕ СРЕДСТВА
МИКРОКОНТРОЛЛЕРОВ STM32**

*Учебное пособие*

Техническое редактирование: А.С. Никитина
Компьютерная верстка: А.С. Никитина

Подписано в печать 26.11.2021. Формат 60×84 1/16.
Бумага офсетная. Печ. л. 30,75.
Тираж 120 экз. (1-й завод 25 экз.). Заказ . Арт. – 6 (РЗУ)/2021.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34.

---

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.