

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»**

Кафедра №43

А.А. Попов

Методические рекомендации к проведению лабораторных
и практических занятий

**«Создание приложений в интегрированной среде разработки Keil-MDK-ARM и их
отладка на симуляторе ядра Cortex-M3»**

для бакалавров
(заочной формы обучения)

Санкт-Петербургский
государственный
университет
аэрокосмического
приборостроения

2023

Содержание

Содержание.....	1
Введение	3
1 Осциллографирование электрических сигналов	4
1.1 Инструкция по технике безопасности	4
1.2 Назначение осциллографа.....	5
1.3 Органы управления осциллографом	8
1.4 Порядок проведения измерений.....	12
1.5 Контрольные вопросы	14
2 Инструменты интегрированной среды разработки MDK Keil µVision.....	15
2.1 Подготовка среды разработки	16
2.2 Документация и библиотеки.....	17
2.3 Порядок создания проекта в ИСР Keil	18
2.4 Свойства проекта в ИСР Keil.....	19
2.5 Файл карты компоновки	21
2.6 Разбор структуры файла startup_stm32f10x_md.s обеспечивающих запуск микроконтроллера.....	23
2.7 Инструменты отладки в ИСР Keil.....	26
2.8 Практические задания к разделу 2	29
2.9 Контрольные вопросы	30
3 Общие принципы программного управления микроконтроллером	32
3.1 Понятие порта ввода/вывода	32
3.2 Программная настройка линии ПВВ в режим логического выхода.....	34
3.3 Использование библиотеки CMSIS для доступа к регистрам	37
3.4 Практические задания к разделу 3	40
3.5 Контрольные вопросы	42
4 Настройка подсистемы тактирования микроконтроллера	44
4.1 Основные регистры управления подсистемой сброса и управления тактированием (RCC)	44
4.2 Пример программной настройки системной частоты.....	45
4.3 Практические задания к разделу 4	49
4.4 Контрольные вопросы	51
5 Настройка системы прерываний микроконтроллера	53
5.1 Симулятор нажатия кнопок в проекте	53
5.2 Пример работы с прерываниями в симуляторе STM32F103x8	56
5.3 Подключение к проекту стандартной библиотеки периферии (SPL).....	60

5.4	Практические задания к разделу 5	62
5.5	Контрольные вопросы	63
6	Таймеры микроконтроллера	65
6.1	Порядок создания проекта с подключением библиотеки HAL LL	65
6.2	Генерация меандра с помощью таймера TIM2	67
6.3	Пример решения задачи по расчёту значений управляющих регистров таймера и подсистемы тактирования	72
6.4	Практические задания к разделу 6	74
6.5	Контрольные вопросы	75
	Литература	77

Санкт-Петербургский
государственный
университет
аэрокосмического
приборостроения

Введение

Основной целью курса является – обучение принципам организации современных встраиваемых систем и первоначальным навыкам программирования встраиваемых приложений.

Встраиваемая система (embedded system) – специализированная цифровая система управления, контроля и мониторинга, концепция разработки которой заключается в том, что такая система будет работать, будучи встроенной непосредственно в устройство, которым она управляет. Основой построения простых встроенных систем часто служат одноплатные (однокристальные) цифровые вычислители (микроконтроллеры), специализированные или универсальные микропроцессоры, ПЛИС. Среди многочисленных современных многофункциональных устройств широко используются микропроцессоры архитектуры ARM.

Во многих отношениях программирование встраиваемых систем не слишком отличается от написания кода для персонального компьютера, но есть некоторые ключевые различия:

- во встраиваемых системах ресурсы (память и вычислительная мощность процессора) ограничены;
- встраиваемые системы, как правило, работают в режиме реального времени, непрерывно, в течение всего срока эксплуатации;
- для встраиваемых систем существует множество вариантов операционных систем, но обычная практика — это вариант работы вообще без операционной системы;
- аппаратная часть каждой встраиваемой системы уникальна и зависит от устройства, которым она управляет.

Результатом изучения курса должны стать:

- представления о принципах работы RISC архитектуры ARM Cortex-M3 процессоров на примере 32х-битных микроконтроллеров STM32F103x;
- знания принципов организации архитектуры встраиваемых систем;
- навыки написания программ на языке *c/c++* и контроля подключенных к микроконтроллеру внешних устройств (датчики, средства индикации, интерфейсы);
- навыки использования измерительных инструментов для отладки встраиваемых приложений.

1 Осциллографирование электрических сигналов

1.1 Инструкция по технике безопасности

Требования безопасности во время занятий:

- 1) Проверить исправность стола и стула (кресла).
- 2) Прежде чем включить электроприбор (осциллограф, компьютер, зарядку и т.п.) в электросеть проверить исправность шнуровой пары: изоляционные втулки штепселей не должны иметь трещин, а шнуры - оголенных от изоляции мест. Розетка должна быть плотно укреплена в стене.
- 3) При обнаружении каких-либо неисправностей в работе оборудования или возникновении каких-либо сомнений по его исправности, доложить преподавателю и до их устранения к учебе не приступать.
- 4) При мигании света, перегорании ламп, неисправности электророзеток, выключателей или их крышек, нарушении изоляции электропроводки сообщить преподавателю или секретарю кафедры.
- 5) При включении и выключении электроприборов браться только за корпус вилки или разъема.
- 6) Запутанный питающий провод любого электроприбора распутывать только при вынутой вилке из штепсельной розетки.

В целях соблюдения электробезопасности студентам запрещается:

- брать в руки оборванные, висящие или лежащие на полу электропровода и наступать на них – они могут находиться под напряжением;
- подходить к электрощитам, открывать двери электрощитов и электрошкафов;
- прикасаться к токоведущим частям электроприборов, клеймам, неизолированным или поврежденным электропроводкам, к арматуре освещения;
- включать в сеть переносные электроприборы без штепсельных розеток;
- пытаться устранить самостоятельно неполадки электрооборудования (освещение, розетки и т. п.).

В целях соблюдения пожаробезопасности студентам запрещается:

- в учебных и служебных помещениях курение и применение открытого огня;
- вешать одежду и другие предметы на выключатели, штепселя или рубильники.
- допускать скопление мусора, одежды, посторонних предметов.

Студент должен знать:

- места расположения медицинской аптечки и средств пожаротушения;

- номера телефонов медицинской службы и пожарной охраны;
- пути эвакуации, а также главные и запасные выходы в случае аварии и пожара.

Студент должен соблюдать требования стандартов, норм и правил по охране труда, а также приказов, распоряжений, постановлений, инструкций по охране труда и пожарной безопасности, регламентированные для охраны учебного процесса.

Меры безопасности при работе с безкорпусной отладочной платой:

- 1) Приступая к работе с открытой печатной платой, где микросхемы и проводники не защищены корпусом, нужно понимать опасность выхода из строя микросхем при касании от статического электричества, которое почти всегда накапливается на теле человека.
- 2) Будьте внимательны и аккуратны! При небрежном отношении к оборудованию возможны механические повреждения.

1.2 Назначение осциллографа

Сигнал (от лат. *signum* - знак) – знак, физический процесс (или явление), несущий сообщение (информацию) о каком-либо событии, состоянии объекта наблюдения либо передающий команды управления, указания, оповещения и т.д.

Signal – форма представления данных, при которой данные рассматриваются в виде последовательности значений скалярной величины – амплитуды, записанной (т.е. измеренной, напечатанной или нарисованной) во времени. Чаще всего, однако не всегда, амплитудой является электрический потенциал.

Электрический сигнал – материальный носитель информации об электромагнитных процессах, происходящих в электрической цепи, в качестве которого используется обычно либо ток, либо напряжение. Наиболее распространено временное и спектральное (частотное) представление и описание электрических сигналов.

Осциллограф – измерительный прибор, предназначенный для визуального наблюдения и исследования электрических сигналов.

Исследуемый сигнал отображается на экране в виде светящихся линий или фигур, называемых осциллограммами. Осциллограмма представляет собой функциональную зависимость двух или трех величин $y=F(x)$ или $y=F(x, z)$, каждая из которых является, в свою очередь, функцией времени: $y(t)$, $x(t)$, $z(t)$. Вертикальная ось (Y) представляет значения

напряжения, горизонтальная ось (X) – время. Интенсивность или яркость выведенной на экран прибора картинки называется осью Z (см. рис. 1.1).

Осциллограф измеряет форму сигнала [1]. Форма колебания напряжения выражается в виде диаграммы зависимости величины напряжения (по вертикали) от времени (по горизонтали) которые показаны на рисунке 1.1. Современные цифровые осциллографы обладают функциями, значительно облегчающими измерения формы сигналов. В качестве органов управления эти приборы используют клавиши лицевой панели и/или экранные меню, через которые можно выбрать режимы полностью автоматических измерений, включающих в себя измерение: амплитуды, периода, частоты, времени нарастания/спада импульса, длительности импульса и др. Результаты автоматических измерений отображаются на экране в текстовом формате. Обычно такие показания более точны, чем интерпретация графического изображения.

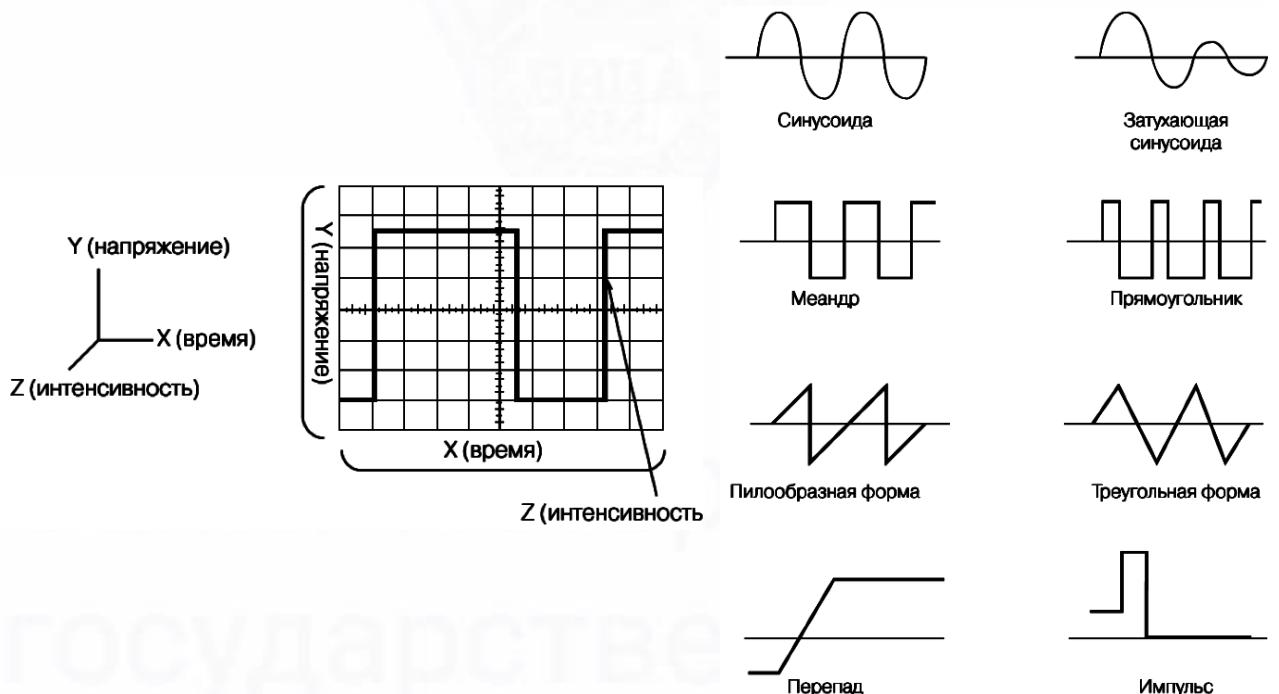


Рис. 1.1. Компоненты X, Y и Z отображаемого сигнала, основные формы сигналов

Обычный цифровой осциллограф (ЦО) позиционируется как цифровой запоминающий осциллограф (DSO - digital storage oscilloscope). Дисплей такого прибора относится к экрану растрового типа. ЦО позволяют захватывать и просматривать события, случающиеся однократно, например переходные процессы. Поскольку информация о сигнале существует в цифровом формате в виде последовательности сохранённых бинарных значений, эти значения можно легко анализировать, архивировать, распечатывать, либо обрабатывать каким-либо иным способом, как в самом осциллографе, так и во вне. В отличие от аналоговых моделей, цифровые запоминающие осциллографы обеспечивают постоянное сохранение в памяти

захваченной информации, разностороннюю обработку параметров и их анализ (см. рис. 1.2). Однако такие приборы не отображают градации яркости развёртки сигнала в реальном времени, поэтому DSO неспособны наглядно представлять изменяющиеся «живые» сигналы.



Рис. 1.2. Маршрут последовательной обработки входных сигналов ЦО

Как и в аналоговых осциллографах, первым (входным) функциональным узлом ЦО является усилитель вертикального отклонения. Органы управления вертикальным отклонением позволяют регулировать амплитуду и положение развёртки сигнала. Далее, аналого-цифровой преобразователь (АЦП) в системе горизонтального отклонения осуществляет выборку сигнала в дискретных точках определенного временного интервала и преобразует напряжение исследуемого сигнала в этих точках в цифровые значения, называемые элементами выборки. Весь этот процесс называется оцифровкой сигнала.

Схема синхронизации системы горизонтального отклонения устанавливает частоту, с которой АЦП делает выборки. Эта величина называется частота выборки и измеряется в выборках в секунду (выб/с). Выборки, полученные от АЦП, сохраняются в оперативной памяти прибора в качестве элементов описания формы сигналов. Некоторое количество выборок могут составить одну точку развёртки сигнала. Взятые вместе точки развёртки сигнала составляют одну развёртку сигнала. Используемое количество точек, необходимое для создания развёртки сигнала называется длина записи. Система запуска осциллографа определяет момент пуска и останова процесса записи.

Сигнальный тракт цифровых осциллографов включает в себя микропроцессор, через который проходит измеряемый сигнал. Микропроцессор обрабатывает сигнал, управляет выводом данных на дисплей, органами управления передней панели прибора и т.п. Затем сигнал поступает в память дисплея, а из нее – выводится на экран.

1.3 Органы управления осциллографом

Обычный осциллограф состоит из 4-х различных систем: системы вертикального отклонения (VERTICAL); горизонтального отклонения (HORIZONTAL); системы запуска (TRIGGER); системы отображения информации (дисплей) (см. рис. 1.3). Правильное понимание того, как работает каждая из этих систем, позволяет эффективно использовать прибор для решения различных измерительных задач. Необходимо помнить, что каждая система вносит свой вклад в способность осциллографа точно воспроизводить форму сигнала.

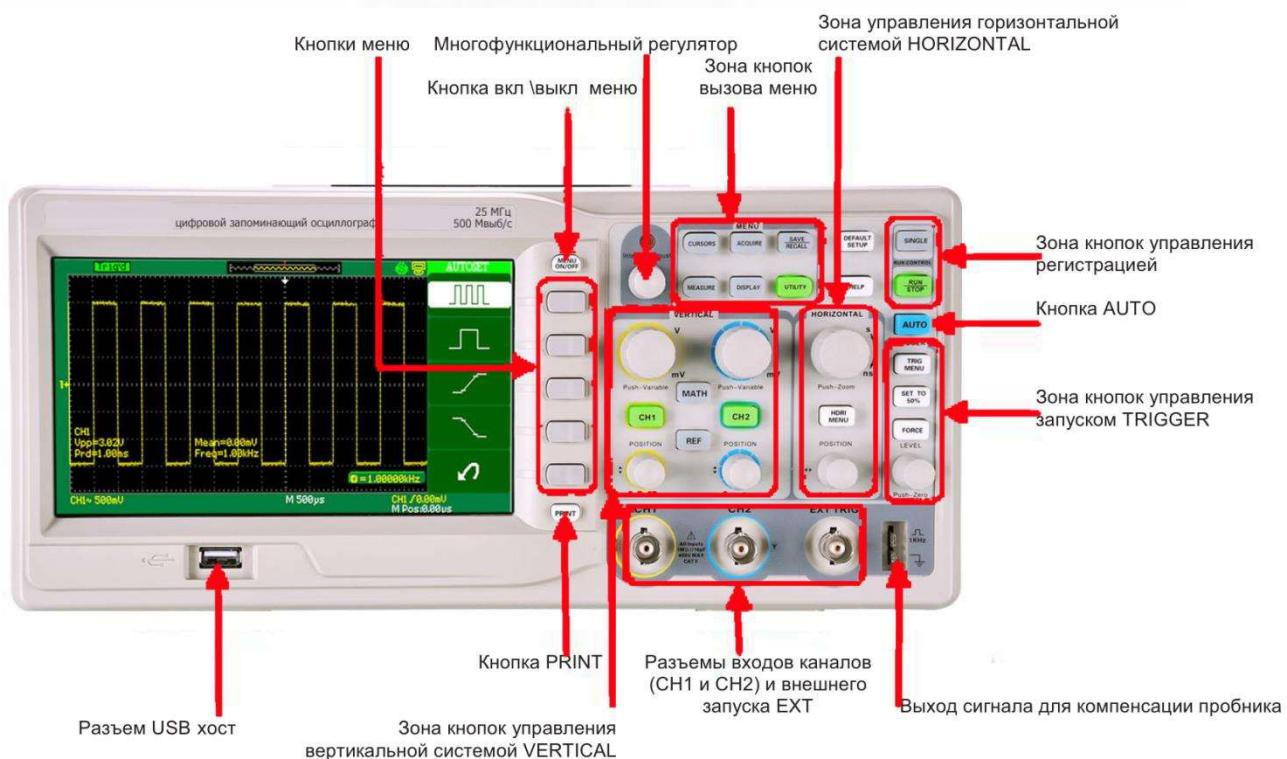


Рис. 1.3. Органы управления осциллографом ПрофКиП С8-2021, расположенные на передней панели

Для того чтобы приступить к измерениям входного сигнала, на осциллографе необходимо выполнить три основные настройки:

- 1) Вертикальное отклонение: установить уровень ослабления или усиления сигнала. Установите ручкой управления чувствительностью (см. рис. 1.4) необходимое число вольт на деление (В/дел) для регулировки вертикального размера отображаемой развёртки сигнала. Если установлено значение 5 В/дел, это значит, что на каждое деление вертикальной шкалы приходится по 5 вольт. Если шкала экрана осциллографа имеет восемь вертикальных делений, то при такой настройке на экране уместится развёртка сигнала амплитудой 40 В. При настройке 0,5 В/дел, весь экран займет развёртка сигнала амплитудой 4 В и т.д. Максимальное

напряжение, которое можно отобразить на экране, равно значению настройки В/дел, умноженному на количество делений вертикальной шкалы. Необходимо учитывать, что используемые пробники с коэффициентами 1x и 10x также влияют на коэффициент масштабирования. В этом случае необходимо умножать значение В/дел на коэффициент ослабления пробника.

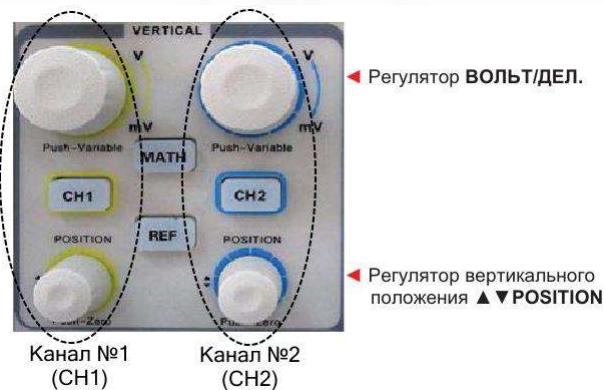


Рис. 1.4. Органы управления вертикальной системы развертки

2) Горизонтальное отклонение: установить скорость развертки. Установите необходимое число секунд на деление (сек/дел) для регулировки масштаба развертки сигнала по горизонтальной оси (см. рис. 1.5). С помощью регуляторов можно изменять горизонтальный масштаб и положение осциллограмм. Стрелка в верхней части масштабной сетки указывает положение момента запуска, а соответствующее числовое значение положения момента запуска в единицах времени отображается в нижней части экрана.

3) Запуск: установить параметры запуска осциллографа. Установите уровень запуска для стабилизации развертки периодического сигнала или выберите режим однократного запуска (см. рис. 1.6). Система запуска определяет момент запуска – момент начала отсчёта времени для регистрируемых данных и отображаемой осциллограммы сигнала. Правильная настройка системы запуска позволяет вместо нестабильного изображения или просто пустого экрана получить осциллограмму, верно воспроизводящую форму сигнала.



Рис. 1.5. Органы управления горизонтальной системы развёртки

Рис. 1.6. Органы управления системы запуска

Функция запуска осциллографа синхронизирует начало горизонтальной развёртки с соответствующей точкой сигнала, что является чрезвычайно важным для точного определения характеристик этого сигнала. Органы управления запуском позволяют стабилизировать изображение периодических сигналов и захватывать однократные и непериодические импульсы. Запуск превращает периодический сигнал в статическое изображение на экране осциллографа посредством непрерывного отображения одного и того же участка входного сигнала. Если каждая развёртка начинается с различных участков исследуемого импульса, то изображение будет нестабильным, с наложениями сигнала, и будет "плыть" влево или вправо, как это проиллюстрировано на рисунке 1.7.

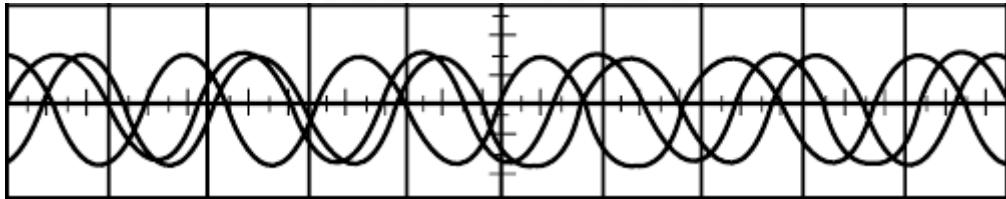


Рис. 1.7. Несинхронизированная развертка синусоидального сигнала

Помимо запуска по уровню (регулятор Level рис. 1.6), применяемому в аналоговых и цифровых осциллографах, большинство цифровых моделей имеют массу специализированных режимов запуска, отсутствующих в аналоговых моделях. В этих режимах запуск осуществляется по определенным условиям или свойствам входного сигнала, что значительно облегчает их выявление, например, обнаружение импульсов, имеющих длительность меньше установленной.

Управление горизонтальным положением точки запуска присутствует только в цифровых осциллографах. Эта функция указывает положение точки запуска по горизонтали при регистрации сигнала как показано на рисунке 1.8. Изменяя это положение, оператор получает возможность увидеть форму сигнала перед моментом запуска. Эта функция известна как «упреждающий запуск». Таким образом, можно определить длину окна обзора исследуемого сигнала как после точки запуска, так и до нее. Способность цифровых осциллографов отображать развертку сигнала до запуска обусловлена тем, что они постоянно обрабатывают входной сигнал, вне зависимости от того, произошла ли команда запуска или нет. Непрерывный поток данных постоянно поступает на прибор, а запуск просто указывает осциллографу на необходимость сохранения части этих данных в памяти.



Рис. 1.8. Установка события запуска

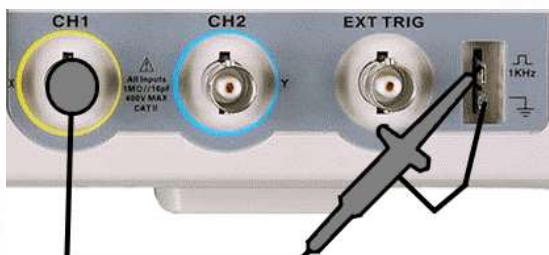
Режимы запуска определяют, каким образом осциллограф отобразит развертку сигнала на основе характеристик этого сигнала. Основными режимами запуска являются «ожиданием» (Normal) и «автоматический» (Auto). При «ожидании» режиме осциллограф запускает развертку, только когда параметры входного сигнала отвечают заданным условиям запуска. В противном случае экран остаётся тёмным (аналоговые осциллографы) или застывшим на последней захваченной осциллограмме (цифровые осциллографы). В «Автоматическом» режиме осциллограф запускает развертку без сигнала запуска. Если в какой-либо момент входного сигнала нет, таймер осциллографа всё равно запускает развертку. При этом на экране прибора всегда присутствует изображение, даже если сигнал не удовлетворяет условиям запуска. Практический интерес представляют оба режима: «ожиданий» режим позволяет просматривать только нужные сигналы, даже на низкой скорости развертки, а «автоматический» режим почти не требует настроек.

Более подробно по настройкам режимов и функциях нужно прочитать в руководстве по эксплуатации осциллографа DS1000 [2] (аналог ПрофКП С8-2021).

При работе с интегральными схемами (ИС) необходимо принять меры по защите исследуемой схемы от статического электричества. Интегральные схемы имеют в своем составе полевые транзисторы, которые могут быть повреждены статическим электричеством, накапливаемым на человеческом теле. Например, вы можете полностью вывести из строя дорогостоящую ИС, прикоснувшись к ее выводам после того, как сняли свитер. Для решения этой проблемы необходимо носить антистатический браслет, который безопасно снимает статический разряд с человеческого тела. Или стараться не касаться к выводам во время лабораторных работ.

1.4 Порядок проведения измерений

Для начала измерений включите питание осциллографа. Подключите пробник к первому каналу (CH1) осциллографа. Для этого, совместите пазы разъёма пробника с выступами входного разъёма BNC канала CH1 и зафиксируйте его поворотом по часовой стрелке. Переключатель ослабления на пробнике установите в положение 1x. При измерениях сигналов требуется наличие двух соединений: соединение наконечника пробника с исследуемой цепью и соединение контакта «земли» пробника с «землей» исследуемого устройства, обычно обозначаемые как GND (ground) или $\frac{1}{\square}$. Для соединения с «землей» пробники, как правило, оснащаются зажимом типа «крокодил». На практике, необходимо прикрепить зажим «крокодил» к известной точке заземления исследуемого устройства, а затем прикасаться наконечником пробника к контрольным точкам тестируемой схемы. Для проверки работоспособности и предварительной настройки осциллографа подключите наконечник пробника и его зажим заземления к выходу осциллографа для компенсации пробников:



Осциллограф ПрофКиП С8-2021 имеет кнопку AUTO (автоматическая настройка) и кнопку DEFAULT SETUP (установка по умолчанию), позволяющие настраивать органы управления в один приём (подраздел 1.2 руководства [2]). Нажмите кнопку DEFAULT SETUP, для восстановления настроек изготовителя, далее нажмите кнопку AUTO, и через несколько секунд на экране появиться осциллограмма меандра с частотой 1 кГц и амплитудой 3 В (см. рис. 1.9). Если меандр не появился или его параметры не соответствуют указанным, попробуйте поменять пробник на другой, возможно неисправность в нём.

Наиболее общий метод измерения напряжения – это подсчёт числа делений вертикальной шкалы, перекрытых разверткой сигнала (рис. 1.9). Выберите такую чувствительность вертикального отклонения (В/дел), чтобы развертка сигнала занимала большую часть площади экрана по вертикали. Чем большую площадь экрана вы займете, тем точнее будут результаты.

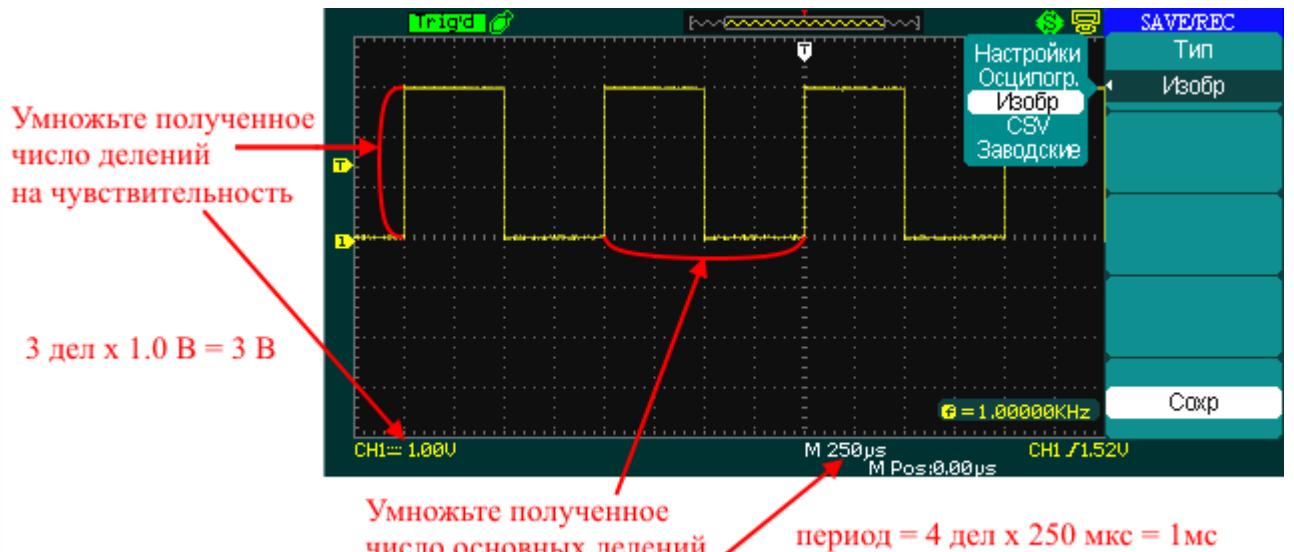


Рис. 1.9. Меандр амплитудой 3В и частотой 1кГц

Для измерения времени используем горизонтальную шкалу осциллографа. Измерения времени включают в себя измерения периода и длительности импульсов. Частота – величина обратная периоду, таким образом, зная период, можно рассчитать значение частоты (рис. 1.9). Точно также как и при измерениях напряжения, измерения времени более точны при позиционировании развертки измеряемого сигнала так, чтобы один период покрывал большую площадь экрана.

Осциллограф обладает экранными маркерами, позволяющими измерять параметры сигналов автоматически, без подсчета делений шкалы вручную. Маркер – обычная линия, которую можно перемещать по экрану осциллографа. Две горизонтальные маркерные линии можно перемещать вверх и вниз для того, чтобы измерить амплитуду сигнала между ними, две вертикальные линии можно перемещать вправо и влево, чтобы измерить время. Управлением экранными маркерами можно изучить самостоятельно по руководству [2] подраздел 2.11.2.

В некоторых случаях очень важно знать точные параметры формы импульса. В стандартные измерения параметров импульсов входят измерения длительности импульса и времени нарастания фронта импульса. Время нарастания – это время, необходимое перепаду импульса для перехода от низкого уровня до высокого уровня. Для удобства время нарастания фронта импульса измеряется от 10% до 90% от значения полного размаха импульса. Такой подход устраняет любые погрешности, связанные с переходными углами. Длительность импульса – это время, требуемое импульсу для перехода от низкого уровня к высокому и обратно к низкому. Для удобства длительность импульса измеряется по уровню 50% от значения полного размаха импульса. На рисунке 1.10 проиллюстрированы эти измерения.

Ручные измерения параметров импульсов требуют точной настройки системы запуска осциллографа и уверенное владение техникой измерения импульсов. Поэтому на начальном этапе лучше использовать режим автоматических измерений, который необходимо самостоятельно изучить (руководство [2], подраздел 2.11.3).

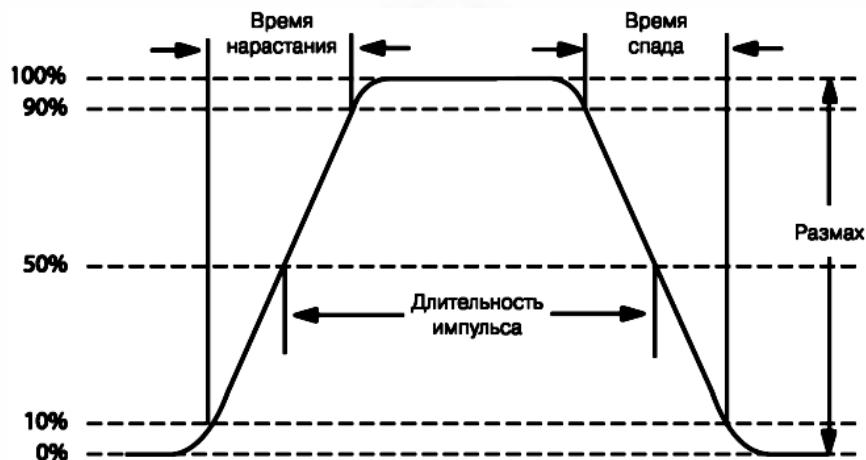


Рис. 1.10. Измерение времени нарастания фронта и длительности импульса

1.5 Контрольные вопросы

- 1) Требования безопасности во время занятий. Что запрещается студенту?
- 2) Назначение осциллографа, порядок проверки работоспособности. Основные системы настройки осциллографа, назначение.
- 3) Системы вертикального и горизонтального отклонения осциллографа, органы управления и их настройка. Система запуска осциллографа, режим работы, органы управления системой и их настройка.
- 4) Порядок измерения амплитуды и частоты сигнала в ручном и автоматическом режимах. Измерение времени нарастания фронта и спада среза импульса.

2 Инструменты интегрированной среды разработки MDK Keil µVision

Установка, настройка и порядок работы с интегрированной средой разработки (ИСР) MDK Keil µVision (ИСР Keil), средства отладки.

В качестве аппаратного обеспечения предполагается использование микроконтроллера (МК) STM32F103C8T6 который получил широкое распространение, благодаря недорогой отладочной плате «Blue Pill» [3] представленной на рисунке 2.1. Принципиальная схема платы представлена здесь [4].

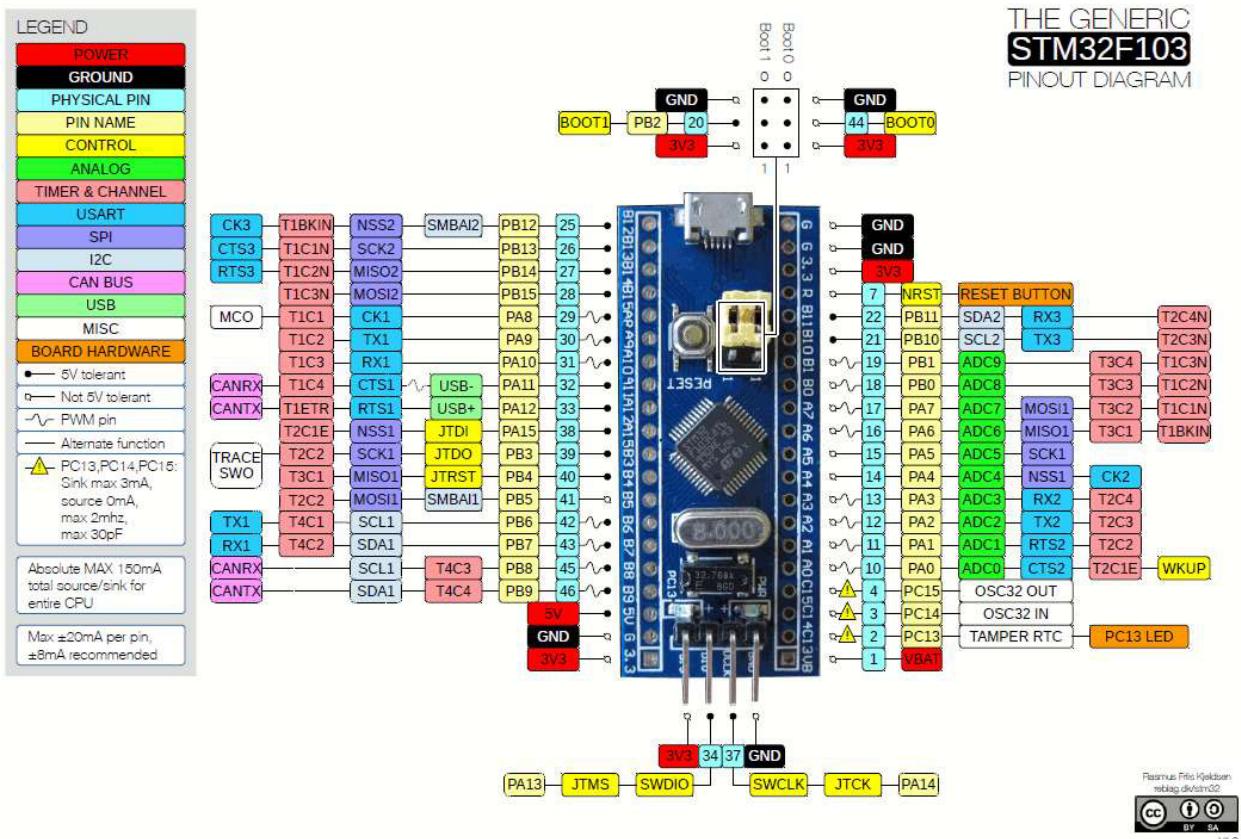


Рис. 2.1. Отладочная плата «Blue Pill» на базе МК STM32F103C8T6

По желанию обучающиеся могут купить отладочную плату и внутрисхемный отладчик-программатор [5] и проверить полученные навыки при изучении дисциплины на практике. Однако для понимания принципов программного управления микроконтроллером достаточно симулятора предоставляемого ИСР Keil. Список симуляторов представлен в разделе µVISION DEBUGGER: Simulation of Cortex-M Devices на сайте www.keil.com.

2.1 Подготовка среды разработки

Для микроконтроллеров на ядре ARM существует достаточное количество сред разработки (см. рис. 2.2). Однако симулятор и качественные средства отладки имеются только в ИСР (Integrated Development Environment - IDE) немецкой компании Keil для платформы ARM – Keil MDK ARM (Microcontroller Development Kit) [6]. Компания Keil является официальным партнером ARM, а сама Keil-MDK является совместной разработкой Keil и ARM. Так ядро IDE (компилятор, линковщик, ассемблер и ряд утилит) собственная разработка ARM, а от Keil – оболочка (μ Vision IDE) и отладчик. Это оценочная версия ПО, которая предоставляется разработчиком бесплатно, но имеет ограничение на размер компонуемого кода в 32 килобайта, которое легко обходится специальным антисанкционным пакетом.

Provider	Product and labels	Cores	Framework	Purpose		Compiler	Debugger	Win	Linux	OS X
				General	Specific					
STMicroelectronics	STM32CubeIDE	All	Eclipse	✓		gcc	gdb / OpenOCD	✓	✓	✓
ac6	System Workbench (SW4STM32)	All	Eclipse	✓		gcc	OpenOCD	✓	✓	✓
Atollic	Atollic TrueSTUDIO	All	Eclipse	✓		gcc	gdb	✓	✓	
iSystem	iSYS-WinIDEAOpen	All	Proprietary	✓		gcc	gdb	✓		
Arm KEIL	MDK5-Cortex-M	M0, M0+	Proprietary	✓		llvm/ARM	uVision	✓		
Arduino	Arduino IDE	All	Proprietary		✓	gcc		✓	✓	✓

Provider	Product and labels	Cores	Framework	Compiler	Debugger	Win	Linux	OS X	Safety edition	Free edition or use, limitation
IAR Systems	EWARM	All	Proprietary	IAR	IAR	✓			✓	Code limit: 32KB
Arm® Keil®	MDK5-Cortex-M	All	Proprietary	llvm / Arm	uVision	✓			✓	Code limit: 32KB
	Keil Studio	All	Theia	llvm	uVision	✓	✓	✓		Online
Emprog	ThunderBench	All	Eclipse	gcc	OpenOCD	✓				Time limit: 30-day
iSystem	iSYS-WinIDEA	All	Proprietary	gcc, others	gdb	✓				WinIDEAOpen
Rowley	CrossWorks	All	Proprietary	gcc	Rowley	✓	✓	✓		Time limit: 30-day
Segger	Embedded Studio	All	Proprietary	gcc / llvm	Segger	✓	✓	✓		Build/Run warning
SysProgs	VisualGDB Embedded	All	Proprietary	gcc / llvm	OpenOCD	✓	✓			Time limit: 30-day
Tasking	TaskingVX	All	Eclipse	Tasking	Tasking	✓	✓	✓		On request
Cosmic	IDEA	All	Proprietary	Cosmic	Cosmic	✓				Code limit: 32KB
Green hills	MultiIDE	All	Proprietary	Green hills	Green hills	✓	✓			

Рис. 2.2. Интегрированные среды разработки под ядро ARM Cortex-M

После окончания установки Keil-MDK-ARM необходимо поставить пакет программного обеспечения с расширением *.pack для поддержки микроконтроллера STM32F103C8T6 [7] в среде.

Для установки скачанного пакета запускаем ИСР Keil-MDK и нажимаем значок менеджера пакетов  (через меню: Project >> Manage >> Pack Installer...).

В появившемся окне менеджера пакетов (Pack installer) импортируем необходимый для работы файл (File >> import >> Keil.STM32F1xx_DFP.2.4.0.pack). После завершения установки проверяем, что файлы были подключены. Для этого, в этом же окне в левой панели во вкладке Devices выбираем STMicroelectronics >> STM32F1 series. В правой панели во вкладке Packs в раскрывающемся списке Device Specific должен быть пакет Keil::STM32F1xx_DFP отмеченный как up to date.

2.2 Документация и библиотеки

Освоение курса подразумевает умение работать с программными библиотеками:

- 1) Библиотека CMSIS [8] (Cortex Microcontroller Software Interface Standard) - стандарт программного интерфейса микроконтроллеров с ядром Cortex, разработанный компанией ARM. Входит в состав ИСР Keil-MDK.
- 2) Библиотека SPL[9] (Standard Peripherals Firmware Library) – стандартная библиотека периферии, набор низкоуровневых драйверов. Каждый драйвер предоставляет пользователю набор функций для работы с периферийным блоком. Входит в состав ИСР Keil-MDK.
- 3) Библиотека HAL LL [10, 11] (Hardware Abstraction Layer and Low-Layer drivers) – слой абстракции встроенного программного обеспечения, гарантирующий максимальную переносимость кода между семействами STM32.

Основной перечень документации на микроконтроллер STM32F103C8 доступен на сайте компании ST Microelectronics по ссылке: <https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html#documentation>

Перечень разделён на разделы. Выделим необходимые документы из раздела Documentation (техническая документация):

- Product Specifications (спецификация изделия). Здесь один файл спецификации DS5319 [12] в котором подробно описаны аппаратные и программные особенности микроконтроллера STM32F103C8. Дополняет справочное руководство RM0008 (ниже).

- Reference Manuals (справочники). Здесь одно справочное руководство RM0008 [13] по семейству STM32F103xx. Здесь описана архитектура, регистры, функции и периферия микроконтроллера.
- Programming Manuals (руководства программиста). Здесь необходимо руководство RM0056 [14] для программирования систем с ядром Cortex-M3, в котором описывается: архитектура системы команд; программная модель; модель памяти; модель прерываний.

Итак, основной перечень необходимой справочной документации для дальнейшей работы состоит всего из трёх документов: RM0008 [13], DS5319 [12], RM0056 [14].

2.3 Порядок создания проекта в ИСР Keil

Запустите среду Keil, выберите в меню Project >> New µVision Project, в открывшемся диалоговом окне укажите папку на диске, где в дальнейшем будут находиться файлы вашего проекта, введите имя проекта. Например, asmstrt, далее Сохранить.

В следующем окне будет предложено выбрать тип МК, под который создаётся проект. Выбираем STMicroelectronics >> STM32F1 Series >> STM32F103 >> STM32F103C8. Если вкладки STM32F1 Series нет, значит, не установлен пакет программного обеспечения [7] (см. п. 2.1).

В следующем окне настройка работы с библиотеками ARM MDK-Professional выбираем пункты CMSIS >> CORE и Device >> Startup, нажимаем OK.

Автоматически создан проект Project: asmstrt включающий цель проекта Target 1, а в цели группа файлов «Source Group 1». Project – это репозиторий для файлов и ресурсов, необходимых для сборки программного продукта. Target — точно определяет, какой продукт будет собран, и содержит инструкции для сборки проекта из набора файлов.

Переименуем названия созданных папок проекта. Для этого наводим указатель на папку проекта (Target 1) и нажимаем правую клавишу мышки, далее в контекстном меню выбираем опцию Manage Project Items... и в диалоговом окне Manage Project Items меняем названия на свои (например «Target 1» на «asmstrt», а «Source Group 1» на «src»).

Также в этом окне можно создавать группы, добавлять и удалять файлы из проекта по этим группам. С группами файлов (особенно когда их много) намного удобнее работать в этом окне, чем непосредственно через контекстное меню в основном окне. В результате окно проекта должно выглядеть как показано на рисунке 2.3

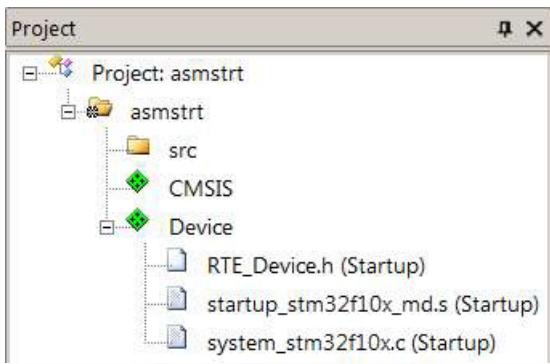


Рис. 2.3. Окно проекта asmstrt

В дальнейшем, для исключения путаницы в файлах при работе со средой Keil лучше согласовывать папки проекта с реальным расположением папок на диске. Поэтому на диске в папке проекта создадим директорию «src».

Для добавления нового файла main.c в папку проекта «src» наведите на неё курсор и нажмите правую клавишу мышки, в контекстном меню выбираем опцию Add New Item to Group 'src'..., в открывшемся окне выбираем C File (.c) и внизу в строке «Name:» вводим имя создаваемого файла main. Далее, ниже в строке «Location:» указываем папку на диске «src» и нажатием «Add» создаём файл.

Таким образом, мы создали простой проект, с подключением дополнительных библиотек, который состоит из четырёх файлов: main.c, RTE_Device.h, system_stm32f10x.c, startup_stm32f10x_md.s.

2.4 Свойства проекта в ИСР Keil

В меню выберите Project >> Options for Target 'src'... В открывшемся диалоговом окне на вкладке Target указан начальный адрес flash-памяти микроконтроллера (IROM1) который начинается с 0x08000000 и имеет размер 0x10000 (64 Кб). Адрес оперативной памяти: IRAM1 соответствует 0x20000000 размер которой 0x5000 (20Кб). Эти данные автоматически выставляются при выборе типа микроконтроллера при создании проекта.

На вкладке Device можно поменять серию MK.

На вкладке Output отметки должны стоять напротив Debug Information (автоматическая вставка на этапе компиляции в код проекта отладочных модулей) и Browse Information: позволяет по имени функции или переменной переходить к месту её

объявления. При усложнении проектов, установка опции Browse Information влияет в сторону увеличения времени компиляции проекта.

На вкладке Listing настраиваются параметры создаваемых файлов листинга, здесь по умолчанию для файла карты компоновки (map-файла) ‘Linker Listing’ должны быть установлены все секции.

На вкладке User можно настроить запуск различных пользовательских программ и сценариев в процессе построения проекта.

На вкладке C/C++ (AC6) выбираются опции компиляции, пути к внешним файлам и прочее. Поскольку работа в основном будет вестись в режиме отладки, то необходимо отключить оптимизацию, выставив опцию Optimization: в положение -O0. Также включаем совместимость библиотек и компилятора 6 версии на уровне предупреждений. Опцию Warnings: в положение AC5-like Warnings, отображать предупреждения только компилятора ARM Compiler 5.

Вкладка Asm предназначена для настройки процедур ассемблирования.

На вкладке компоновщика Linker, устанавливаем опцию Use Memory Layout from TargetDialog, для автоматического согласования раскладки областей памяти с указанным во вкладке Device типом МК. При компиляции проекта будет создан файл с расширением *.sct, где будут указаны эти области памяти. В строке Linker control string появится запись о подключении данного файла: --strict --scatter ".\Objects\asmstrt.sct".

На вкладке Debug в левой половине выбираем Use Simulator. В поле Dialog DLL и Parameter вводим: DARMSTM.DLL и -pSTM32F103C8, соответственно, как показано на рисунке 2.4. Для справки, если вы хотите работать с реальным МК через программатор, то настраиваете правую половину вкладки: опцию Use устанавливаете в положение ST-Link Debugger; в поле Dialog DLL и Parameter вводим: TARMSTM.DLL и -pSTM32F103C8, соответственно.

Последняя вкладка Utilities предназначена для настройки программирования МК из ИСР Keil. По умолчанию включена опция Use Debug Driver которая подключает настройки из вкладки Debug.

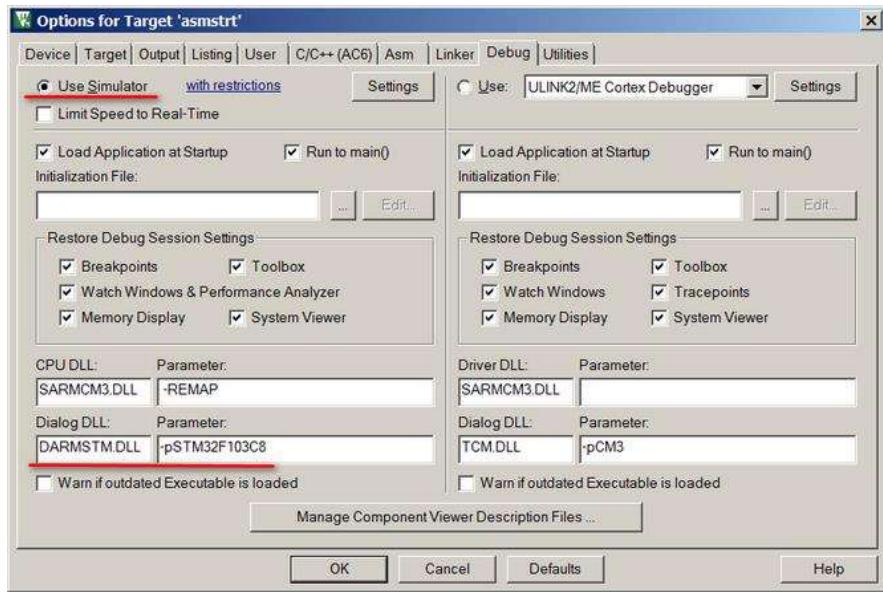


Рис. 2.4. Настройка проекта при работе с симулятором

2.5 Файл карты компоновки

Добавим следующий код в файл main.c:

```
int main ( ) {
    for(;;){}
    return 0;
}
```

Так как программа работы микроконтроллера никогда не должна заканчиваться, вставлен бесконечный цикл for. В противном случае можем получить непредсказуемое поведение микроконтроллера.

Компилируем программу Project >> Build Target. После сборки, в окне Build Output получаем:

```
Build started: Project: asmstrt
*** Using Compiler 'V6.19', folder: '... ARM\ARMCLANG\Bin'
Build target 'asmstrt'
compiling main.c...
assembling startup_stm32f10x_md.s...
compiling system_stm32f10x.c...
linking...
Program Size: Code=732 RO-data=252 RW-data=0 ZI-data=1632
".\Objects\asmstrt.axf" - 0 Error(s), 0 Warning(s).
```

Где Code – размер кода (байт), RO-data – константы в коде (*Read Only Data*), RW-data – переменные (*Read Write Data*), ZI-data – переменные, инициализируемые нулями (*Zero-Initialized Data*). Исходя из этих данных можно подсчитать, сколько оперативной (RAM) и

постоянной (ROM) памяти нужно нашему проекту: $RAM = RW + ZI = 0 + 1632 = 1632$ байт и ROM (FLASH память) = $Code + RO + RW = 732 + 252 + 0 = 984$ байт.

Согласно настроенным свойствам проекта файл карты компоновки ([имя проекта].map) создается автоматически во время компиляции и содержит справочную информацию о результатах сборки проекта. Компоновщик (Linker) сохраняет в карту компоновки отчет о вычислении переменных, размещении входных секций, файлов и библиотечных объектов в выходных секциях исполняемой программы. Данные по размеру памяти необходимой исполняемому коду можно также посмотреть и в этом файле. На диске он расположен в папке проекта Listings.

Для открытия его в среде Keil наведите курсор на папку asmstrt дерева проекта и нажмите правую кнопку мыши, в контекстном меню выберете вкладку Open Map File.

В открывшемся map-файле, промотав в конец, последние две строки как раз и показывают необходимые RAM и ROM:

Total RW Size (RW Data + ZI Data)	1632 (1.59kB)
Total ROM Size (Code + RO Data + RW Data)	984 (0.96kB)

Также в разделе Memory Map of the image показаны адреса, по которым расположены стек и область памяти с неупорядоченным хранением данных ('heap'):

адрес расположения	размер	
0x20000060	-	0x00000200 Zero RW 11 HEAP startup_stm32f10x_md.o
0x20000260	-	0x00000400 Zero RW 10 STACK startup_stm32f10x_md.o

А также адреса, по которым происходит загрузка исполняемого кода в ROM:

Exec Addr	Load Addr	Size	Type	Attr	Idx	E	Section Name	Object
0x08000000	0x08000000	0x000000ec	Data	RO	12	RESET		startup_stm32f10x_md.o
0x080000ec	0x080000ec	0x00000008	Code	RO	40	*	!!!main	c.w.l(main.o)
0x080003bc	0x080003bc	0x0000000a	Code	RO	2	.	.text.main	main.o
		...						

В разделе таблица символов (Image Symbol Table), показаны глобальные и локальные метки и их адреса, используемые в программе:

Local Symbols						
Symbol	Name	Value	Ov	Type	Size	Object(Section)
Heap_Mem		0x20000060		Data	512	startup_stm32f10x_md.o(HEAP)
HEAP		0x20000060		Section	512	startup_stm32f10x_md.o(HEAP)
Stack_Mem		0x20000260		Data	1024	startup_stm32f10x_md.o(STACK)
STACK		0x20000260		Section	1024	startup_stm32f10x_md.o(STACK)
__initial_sp		0x20000660		Data	0	startup_stm32f10x_md.o(STACK)
...						
Global Symbols						
Symbol	Name	Value	Ov	Type	Size	Object(Section)
__Vectors_Size		0x000000ec		Number	0	startup_stm32f10x_md.o ABSOLUTE
__Vectors		0x08000000		Data	4	startup_stm32f10x_md.o(RESET)
__Vectors_End		0x080000ec		Data	0	startup_stm32f10x_md.o(RESET)
Reset_Handler		0x0800016d		Thumb Code	8	startup_stm32f10x_md.o(.text)
main		0x080003bd		Thumb Code	10	main.o(.text.main)
...						

Так как мы подключили файл на языке си (функцию main) то компилятор ARM автоматически подключает код монитора отладки Angel, библиотеки которого усложняют восприятие тар-файла. Поэтому пользуйтесь поиском при анализе файла карты компоновки.

2.6 Разбор структуры файла startup_stm32f10x_md.s обеспечивающих запуск микроконтроллера

Программа на ассемблере файла startup_stm32f10x_md.s имеет следующую структуру:

- 1) Объявление области стека (stack).
- 2) Объявление области памяти с неупорядоченным хранением данных (heap).
- 3) Таблица векторов прерываний.
- 4) Процедура, исполняемая после сброса (reset handler).

1) Объявление области стека (stack) состоит из следующих директив:

```
Stack_Size      EQU      0x00000400
                  AREA     STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE    Stack_Size
__initial_sp
```

В первой строке объявлена метка Stack_Size со значение 0x400 (1024 байта) – размер стека. Здесь директива EQU указывает препроцессору ассемблера сопоставить метке указанное значение. Во второй строке директивой AREA создается отдельная секция в памяти с названием STACK. За названием следуют атрибуты: NOINIT – данные в секции остаются как есть (не инициализируются); READWRITE – секция доступна для чтения и записи; ALIGN=3 – выравнивание секции по границе кратной восьми байт ($2^3=8$).

В третьей строке выделяется пространство в памяти заданного размера (`Stack_Size`) для области стека. Директива `SPACE` резервирует место в памяти, метке `Stack_Mem` сопоставляется начальный адрес резервируемой области памяти (дно стека), а в метке `__initial_sp` следующий адрес после этой области (вершина стека).

2) Объявление области памяти с неупорядоченным хранением (heap) по структуре идентична объявлению стека:

```
Heap_Size      EQU      0x00000200
                AREA     HEAP, NOINIT, READWRITE, ALIGN=3
__heap_base
Heap_Mem       SPACE    Heap_Size
__heap_limit
```

Объявлена метка `Heap_Size` равная 0x200 (512 байт) – выделяемый размер области памяти с неупорядоченным хранением данных. В объявление секции `HEAP` атрибуты имеют те же значения что и при объявлении стека: `__heap_base` и `Heap_Mem` – адреса начала области `heap`, `__heap_limit` – адрес, следующий после области `heap`.

Названия меток для стека и области `heap` такие как: `Stack_Size`, `Stack_Mem`, `__initial_sp`, `Heap_Size`, `__heap_base`, `__heap_limit`, `Heap_Mem`, и др. стандартизированы под восприятие компилятора языка си. В проектах ассемблерный файл кода запуска МК формируется автоматически при подключении библиотек и менять в нём рекомендуется только размеры стека и области `heap`.

Следующие после объявления стека и `heap` директивы: `PRESERVE8` – указывает компоновщику сохранять 8-байтное выравнивание стека (требование стандарта вызова процедур для архитектуры ARM – AAPCS); `THUMB` – указывает ассемблеру интерпретировать последующие инструкции как `THUMB`-инструкции (ARM архитектура позволяет использовать упакованный код – инструкции сокращённые до 16 бит).

3) Таблица векторов прерываний (адреса обработчиков прерываний):

```
AREA      RESET, DATA, READONLY
EXPORT   __Vectors
EXPORT   __Vectors_End
EXPORT   __Vectors_Size
__Vectors DCD     __initial_sp           ; Top of Stack
          DCD     Reset_Handler        ; Reset Handler
...
__Vectors_End
__Vectors_Size EQU   __Vectors_End - __Vectors
```

Сначала создаётся секция `RESET` с атрибутами: `DATA` – секция содержит данные, а не инструкции; `READONLY` – секция только для чтения. Далее директивой `EXPORT` указываем компилятору, что метки `__Vectors`, `__Vectors_End`, `__Vectors_Size` являются глобальными и видны за пределами нашего файла. Директивой `DCD` распределяем в ПЗУ

четыре байта памяти под метку `__initial_sp` (адрес вершины стека), а метке `__Vectors` сопоставляется адрес начала таблицы векторов прерываний `0x08000000` который указан в *map*-файле в разделе *Image Symbol Table* (п. 2.5):

Следующие четыре байта в таблице векторов это адрес первой команды подпрограммы, исполняемой после сброса (Reset) – указатель на процедуру `Reset_Handler`. Код этой процедуры будет выполняться всегда после сброса МК.

Далее в таблице векторов следуют адреса процедур обработки прерываний, порядок которых определен в таблице *Table 62. Vector table for XL-density devices* руководства RM0008 [13].

Заканчивается таблица меткой `__Vectors_End` – адрес конца таблицы векторов прерываний. Размер таблицы векторов прерываний содержит метка `__Vectors_Size`.

Таблица векторов прерываний является неотъемлемой составляющей запуска ядра МК. Это связано с тем, что процессор при запуске начинает выполнение кода с адреса `0x00000000`. А именно, читает первые четыре байта таблицы векторов (значение вершины стека) и записывает их в регистр R13(SP) (Stack Pointer - указатель стека). Затем считывает следующие четыре байта с адреса `0x00000004` и записывает их в регистр R15(PC - Program Counter - счётчик команд), т.е. начинает исполняться первая команда процедуры `Reset_Handler`.

Совмещение адресного пространства регионов памяти, где хранится прошивка по адресу `0x08000000` и запуска с адреса `0x00000000` происходит настройкой линий BOOT0 и BOOT1 микросхемы (более подробно см. п. 3.4 Boot configuration RM0008 [13]).

4) Код процедуры, исполняемой после сброса (`Reset_Handler`), это место, с которого начинается выполнение программы пользователя.

```
AREA      |.text|, CODE, READONLY
; Reset handler
Reset_Handler    PROC
                  EXPORT Reset_Handler          [WEAK]
IMPORT    __main
IMPORT    SystemInit
          LDR     R0, =SystemInit
          BLX     R0
          LDR     R0, =__main
          BX      R0
ENDP
```

Сначала объявлена секция `|.text|` с атрибутами: `CODE` – секция содержит инструкции (исполняемый код); `READONLY` – секция только для чтения. Далее метка `Reset_Handler` сопоставлена с процедурой директивой `PROC`, за которой следует исполняемый код до директивы `ENDP`. Весь код состоит из четырёх команд: команды загрузки

в регистр R0 указателя на функцию `SystemInit` и команды перехода по этому указателю; команды загрузки в регистр R0 указателя на функцию `__main` и команды перехода по этому указателю. То есть сначала будет выполнена функция `SystemInit`, код которой расположен в файле `system_stm32f10x.c`, отвечает за инициализацию подсистемы тактирования МК. А затем начнёт выполнение функция `__main` которая является частью отладчика и из неё будет вызвана функция `main` пользователя.

2.7 Инструменты отладки в ИСР Keil

Ядро микроконтроллера ARM Cortex-M3 оснащено технологией отладки и трассировки ARM CoreSight, которая позволяет осуществлять доступ к памяти во время исполнения рабочей программы, без использования каких-либо дополнительных программных модулей. Во время выполнения программы разработчик непрерывно получает информацию об актуальном состоянии памяти и регистров контроллера.

Переход в режим отладки осуществляется через главное меню `Debug >> Start/Stop Debug Session`. После того как Keil предупредит нас о неполноценности версии с ограничением на размер исполняемого кода в 32 Кб, нажимаем *OK* и продолжаем работу. В режиме отладки симулятора МК окно программы примет вид, показанный на рисунке 2.5.

Окно состояния регистров - `Registers`. Как видно из рисунка 2.5, вершина стека загружена в `R13(SP)`, в регистр `R15(PC)` загружен адрес следующей исполняемой команды функции `main`. Регистр `xPSR` содержит флаги результатов выполнения арифметических и логических операций, состояние выполнения программы и номер обрабатываемого в данный момент прерывания. В документации об этом регистре пишут во множественном числе, так как к его трём частям можно обращаться как к отдельным регистрам. Эти «подрегистры» называются: `APSR` — регистр состояния приложения (тут хранятся флаги), `IPSR` — регистр состояния прерывания (содержит номер обрабатываемого прерывания) и `EPSR` — регистр состояния выполнения (более подробно стр. 16 fig. 3 PM0056[14]).

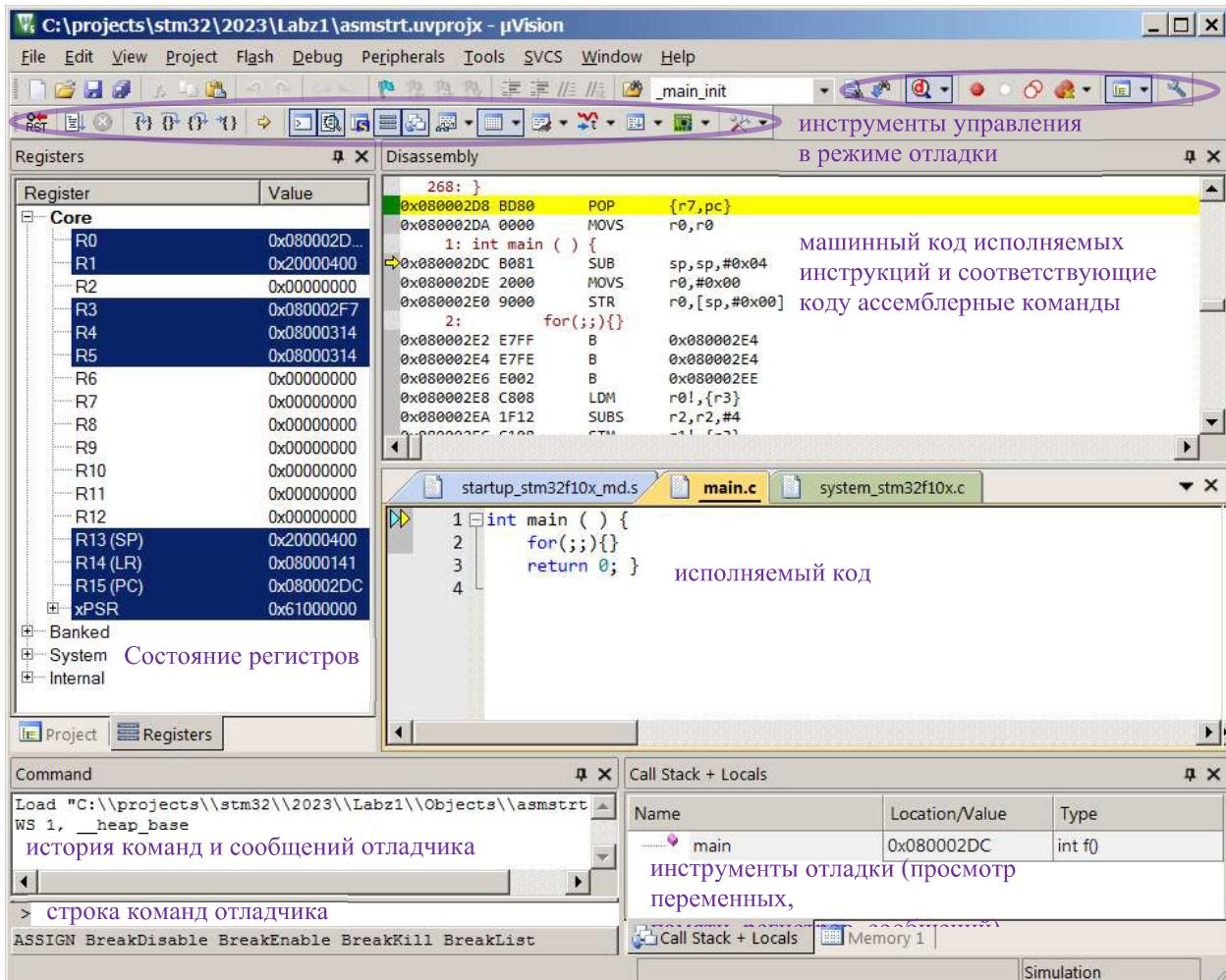


Рис. 2.5. Режим отладки ИСР Keil

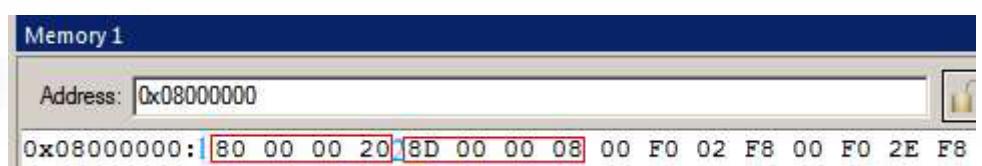
В окне '*Disassembly*' (дизассемблирования) наблюдаем несколько столбцов. Первый столбец – это адреса, по которым хранятся исполняемые инструкции, второй столбец – коды инструкций, третий столбец мнемоническое обозначение инструкций – команды ассемблера:

0x0800013A E7FE	в	0x0800013A
адрес команды	код команды	мнемоническое обозначение операнд

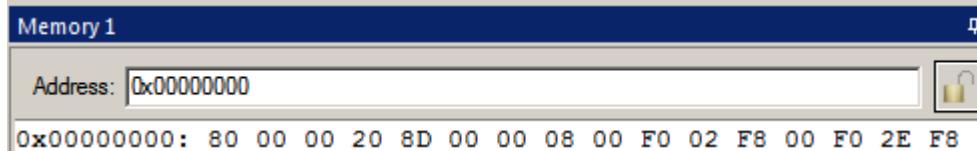
жёлтая стрелка указывает на текущую команду, подлежащую исполнению.

Для просмотра областей памяти нажмите вкладку *Memory 1* в окне инструменты отладки. Если вкладка отсутствует, то вызовите через меню *View >> Memory Windows >> Memory 1 (2, 3, 4)*.

В окне *Memory 1* в поле *Address*: введите адрес начала flash памяти *0x08000000* и нажмите ввод. Получим текущий отпечаток памяти:



из которого видно, что первые четыре байта это указатель вершины стека - адрес 0x20000080 (архитектура little-endian, порядок байт *от младшего к старшему*), вторые четыре байта это адрес функции Reset_Handler 0x0800008D. Далее сравним эту область с памятью по нулевому адресу. Изменим адрес на 0x00000000:



Как видно значения абсолютно одинаковые, так как эти адресные регионы совмещены.

Выйдем из режима отладки (повторное нажатие *Ctrl + F5*) и допишем следующий код в функцию main:

```

1 int main(void)
2 {
3     volatile unsigned char a1=0x12;
4     volatile unsigned short a2=0x1234;
5     volatile unsigned int a4=0x12345678;
6     volatile unsigned long long a8=0x1234567887654321;
7     volatile char name[]="Vladimir Solov'ev";
8     for(;;){}
9     return 0;
10 }
```

Откомпилируем (*F7*) и запустим отладку (*Ctrl + F5*). Переключимся на окно Disassembly. В инструментах отладки нажимаем *F11*, либо значок в панели инструментов, либо через меню Debug >> Step. Проследите за исполнением ассемблерных команд инициализации переменных a1, a2, a4, a8, name.

Рассмотрим расположение указанных переменных в памяти. В режиме отладки, щёлкните на переменную правой клавишей мыши, в появившемся контекстном меню выберите Add 'название переменной' to ... и в выпавшем списке выберете Watch 1(список наблюдаемых переменных).

Другой способ добавить переменную, открыть окно Watch 1 (меню View >> Watch Windows >> Watch 1 (2)) и вбить имя нужной переменной самостоятельно вручную в поле Name. Добавим переменные a1, a2, a4, a8. И по мере нажатия *F11*, с каждым шагом выполнения, наблюдаем инициализацию значениями наших переменных:

Name	Type	Value
a1	uchar	0x12
a2	ushort	0x1234
a4	uint	0x12345678
a8	_uint64	0x1234567887654321
name	uchar[18]	0x200003CE "Vladimir Solov'ev"

Когда значения установились, для получения адресов переменных воспользуемся операцией «взятия адреса» – вводим амперсанд (&) перед переменными a1, a2, a4, a8. И далее по полученным адресам сопоставим со значениями в памяти как показано на рисунке 2.6.

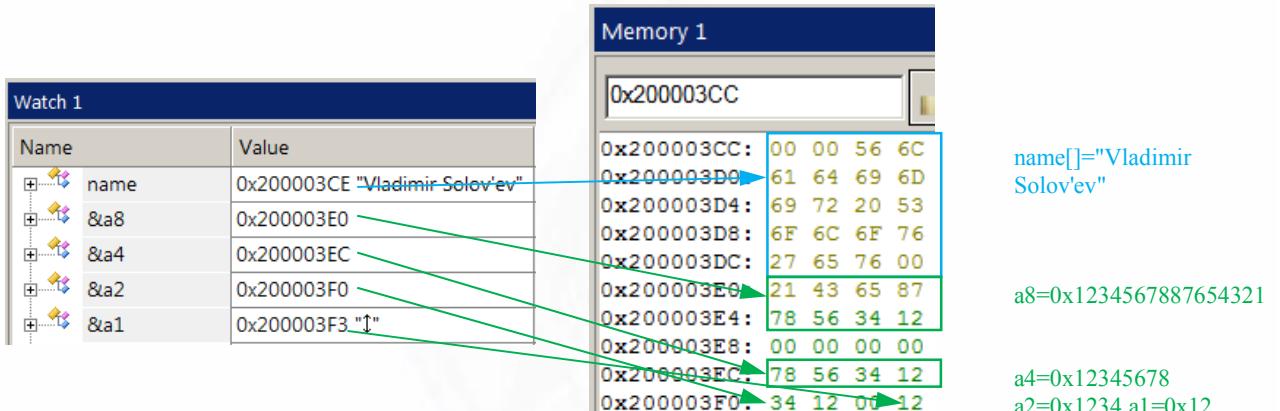


Рис. 2.6. Представление переменных в памяти МК

Сохраним отпечаток этой области памяти в файл, в окне Command вводим в командную строку log > dat.log; d &name[0],&a1; log off:

```
Command
Load "C:\\projects\\stm32\\2023\\Labz1\\Objects\\asmstrt.axf"
WS 1, `name
WS 1, &a8
WS 1, &a4
WS 1, &a2
WS 1, &a1
log > dat.log; d &name[0],&a1; log off
0x200003CE 56 6C 61 64 69 6D 69 72 - 20 53 6F 6C 6F 76 27 65 Vladimir Solov'e
0x200003DE 76 00 21 43 65 87 78 56 - 34 12 00 00 00 00 78 56 v.!Ce.xV4.....xV
0x200003EE 34 12 34 12 00 12
4.4...
```

Команда log > dat.log создаст файл регистрации dat.log. Далее d &name[0],&a1, где: d (команда Display) сохранит значения памяти начиная с адреса &name[0] и заканчивая адресом &a1. Далее команда log off закроет файл регистрации dat.log.

Содержимое файла dat.log:

```
0x200003CE 56 6C 61 64 69 6D 69 72 - 20 53 6F 6C 6F 76 27 65 Vladimir Solov'e
0x200003DE 76 00 21 43 65 87 78 56 - 34 12 00 00 00 00 78 56 v.!Ce.xV4.....xV
0x200003EE 34 12 34 12 00 12
4.4...
```

2.8 Практические задания к разделу 2

- 1) Создайте проект. Задайте размеры стека и heap согласно формуле: 0x40+0x80×[номер варианта]₁₆. Переменным: a1, b1, c1 типа unsigned char; a2, b2, c2 типа unsigned short; a4, b4, c4 типа unsigned int;

`a8, b8, c8` типа `unsigned long long` присвоить повторяющиеся значение $0x11+0x9\times[\text{номер варианта}]_{16}$ (пример: номер варианта $26_{10}=1A_{16}$, $0x11+0x9\times0x1A=0xFB$, $a1=0xFB$, ..., $c4=0xFBFBFBFB$, ...), переменной `name1` присвоить своё имя, переменной `name2` фамилию в латинской транскрипции, `name3` номер группы.

2) Найти в файле карты компоновки (map-файле): затраты оперативной (RAM) и постоянной (ROM) памяти МК для вашего проекта; адрес расположения и размер стека, `heap`; адрес расположения и размер таблицы векторов; адрес расположения и размер функции `main()`.

3) Проанализировать переменные $a1 \div c8$, $name1 \div 3$ инструментами отладки ИСР Keil. Определить адреса переменных. По найденным адресам определить расположение в памяти. Сохранить отпечаток всей области памяти этих переменных в файл `logdat.txt`.

4) Оформить отчёт.

Отчёт должен содержать:

1) Текст задания согласно варианту, с указанием размера стека, значений переменных.

2) Программу проекта.

3) Выписку из файла карты компоновки: затрат оперативной и постоянной памяти проекта; адрес расположения и размер стека, `heap`; адрес расположения и размер таблицы векторов; адрес расположения и размер функции `main`.

4) Адреса расположения в памяти переменных $a1, b1, \dots, b8, c8$, $name1/2/3$ (подобно рис. 2.6).

5) Содержимое файла `logdat.txt` с отпечатком всей области памяти содержащей значения переменных $a1, b1, \dots, b8, c8$, $name1/2/3$.

2.9 Контрольные вопросы

1) Названия основных используемых программных библиотек. Перечень необходимой документации для программирования МК STM32F1xx.

2) Понятие проекта. Свойства проекта в ИСР Keil.

3) Общая структура программы на ассемблере запуска МК.

4) Режим отладки, назначение окон и панелей инструментов. Инструменты слежения за состоянием переменных в режиме отладки.

5) Работа с памятью в режиме отладки, порядок сохранения отпечатка памяти на диск.

Типы данных, количество байт, выделяемое каждому типу.

6) Найти в файле отпечатка памяти значение указанной переменной и выделить.

Санкт-Петербургский
государственный
университет
аэрокосмического
приборостроения

3 Общие принципы программного управления микроконтроллером

Изучение основных принципов программирования микроконтроллера серии STM32F103xx на уровне регистров управления. Изучение принципов работы с портами ввода/вывода (ПВВ).

3.1 Понятие порта ввода/вывода

Количество выводов корпуса микроконтроллера зависит от технологии его исполнения, что отражается в последних знаках маркировки нашего МК – STM32F103C8T6. Используя DS5319 [12], раздел 7 на стр.104, определяем: корпус имеет 48 выводов (pins); flash-память 64 Кб; тип корпуса LQFP (Low Profile Quad Flat Package – корпус для поверхностного монтажа, имеющий планарные выводы, расположенные по всем четырём сторонам). Назначение выводов показано в DS5319, на стр. 26, fig. 8, для удобства он продублирован здесь как рисунок 3.1.

Figure 8. STM32F103xx performance line LQFP48 pinout

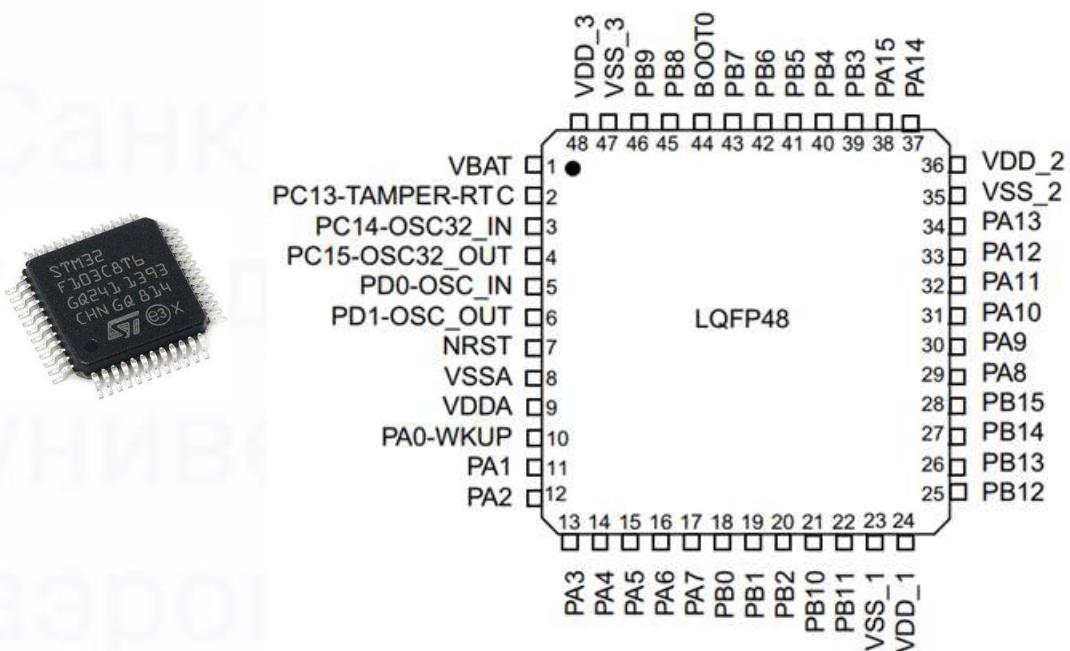


Рис. 3.1. Маркировка выводов корпуса микроконтроллера STM32F103C8T6

Рассмотрим выводы, обозначение которых начинается на латинскую букву P(Port). Такие выводы могут работать в разных режимах, совмещая функции, как ввода, так и вывода

сигналов. Под сигналом будем понимать изменение напряжения по определенному закону. Определим понятие логический вход/выход устройства, как вход/выход, на который подается/с которого поступает сигнал логической единицы или логического нуля. Сигнал логического нуля или единицы – это сигнал, принимающий два возможных уровня напряжения: уровень в диапазоне нуля вольт – «низкое» состояние; три вольта - «высокое» состояние. Сигналы логического нуля и единицы в зарубежных источниках соответственно обозначают как low level и high level.

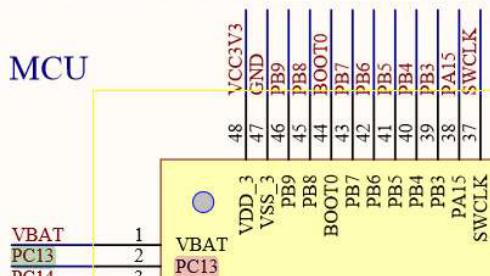
В микроконтроллере линии ввода/вывода логически объединяются в порты ввода/вывода (ПВВ, General-Purpose I/O port - GPIO) и включают 8 или 16 выводов (pins), стандартное обозначение которых выглядит как Pnx, где n – обозначение порта (прописными буквами латинского алфавита – A, B, C, D и т.д.); x – номер бита (линии) в порту (цифры от 0 до 15). ПВВ выступает в роли точки подключения к микроконтроллеру внешних устройств.

Разберём, как установить сигнал логической единицы на выводе №2 нашего микроконтроллера.

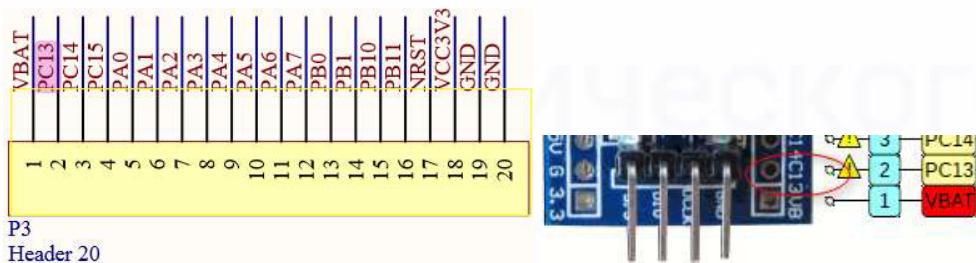
На рисунке 3.1, находим вывод под номером 2 и связанную с ним маркировку – PC13. PC13 означает что линия в/в логически относится к порту в/в C (GPIOC) под номером 13.

Используя принципиальную схему отладочной платы [4] проследим путь линии в/в PC13 от 2-й ножки микросхемы:

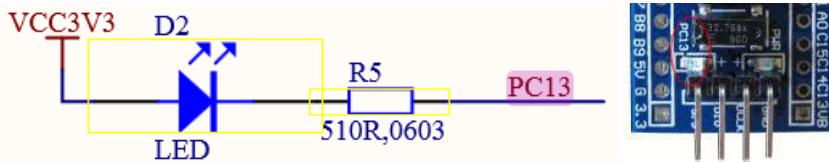
1) На схеме [4] показан MCU (microcontroller unit) где выводу №2 присвоено обозначение PC13:



2) Линия PC13 выведена на однорядный штыревой разъём P3 (квадрат А4). Для удобства проведения измерений с помощью осциллографа:



3) Линия PC13 через резистор R5 присоединена к светодиоду D2 (квадрат A3), для управления свечением (низкий уровень на линии PC13 будет зажигать светодиод):



3.2 Программная настройка линии ПВВ в режим логического выхода

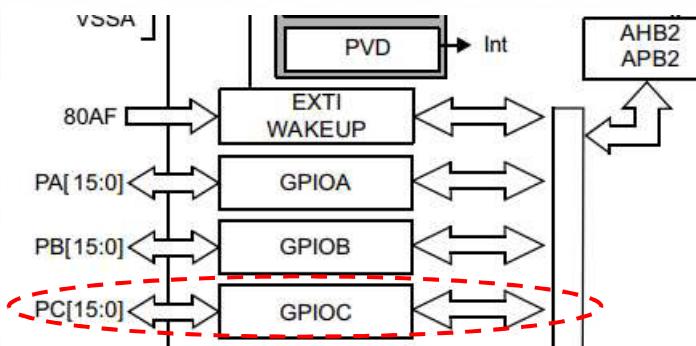
Создадим проект (см. 2.3). В main.c двумя строчками кода подключим библиотеку CMSIS:

```
#include "RTE_Components.h"
#include CMSIS_device_header
```

Файл RTE_Components.h (Run-Time-Environment Components) – файл программных компонентов, создается автоматически, при настройке используемых в среде Keil библиотек.

После этого объявим функцию int main(). И далее, по мере изучения МК, наполним тело функции операторами.

Перед использованием ПВВ С (GPIOC) необходимо включить его в работу и вывести из состояния сброса. Для этого в подсистеме RCC (RM0008 [13] стр. 116) – сброса и управления тактированием, имеется несколько регистров управления. В спецификации DS5319 [12], на странице 11 fig. 1, показано, что все ПВВ подключены к шине периферии APB2, которая через мост подключена к высокоскоростной шине АHB2:



Для управления тактированием устройств шины APB2 используется регистр RCC_APB2ENR (RM0008 стр.146 подраздел 8.3.7) в котором имеются следующие управляющие биты:

	7	6	5	4	3	2	1	0
ved	IOPC EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN	
	rw	rw	rw	rw	rw		rw	

В пояснении указано:

Bit 4 IOPCEN: I/O port C clock enable (разрешает тактирование ПВВ С).

Set and cleared by software (Устанавливается и очищается ПО).

0: I/O port C clock disabled (Тактирование порта ввода/вывода С запрещено).

1: I/O port C clock enabled(Тактирование порта ввода/вывода С разрешено).

Адрес регистра RCC_APB2ENR определим как сумму начала области памяти подсистемы Reset and clock control RCC – 0x4002 1000 (RM0008, Table 3 на странице 50) и смещения 0x18 (поле Address: 0x18 в описании RCC_APB2ENR): 0x4002 1000+0x0000 0018 = 0x4002 1018. Закодируем изменение 4-го бита на языке си строчкой: `* (uint32_t*) (0x40021018) |= 0x00000010;`. Здесь в выражении используется оператор ‘|=’, а не простое присваивание ‘=’, поскольку в регистре могут быть установленные в единицу некоторые биты и их обнуление может привести к системному сбою. uint32_t соответствует типу unsigned int. В общем виде получится программа:

```
main.c
1 #include "RTE_Components.h"
2 #include CMSIS_device_header
3 int main () {
4     *(uint32_t*)(0x40021018) |= 0x00000010;
5     for(;;){}
6     return 0; }
```

Скомпилируем и запустим в режиме отладки. Проконтролируем, что программа действительно установит единицу в 4-й бит регистра RCC_APB2ENR.

Откроем регистры подсистемы RCC через главное меню View >> System Viewer >> RCC. В появившемся окне развернём регистр APB2ENR.

Напротив самого регистра мы видим его текущее значение 0. В самом низу показано, что в регистре 32 бита [Bits 31..0], доступен для чтения и записи (RW), его физический адрес в памяти 0x40021018, и в скобках, мнемоническое обозначение регистра RCC_APB2ENR. Также показано текущее состояние управляющих бит в регистре которое можно поменять левой кнопкой мыши. При пошаговом исполнении инструкций , проход строки 4 изменит состояние бита IOPCEN. А значение напротив регистра APB2ENR измениться на 0x00000010. Таким образом, записав единицу в 4й бит, мы включили в работу порт ввода/вывода С (GPIOC).

Далее, установим высокий уровень на линии в/в PC13. Выделим из множества режимов работы ПВВ показанных в Table 20. Port bit configuration table на странице 161 RM0008 [13], нужный нам режим:

Table 20. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General purpose output	Push-pull	0	0	01		0 or 1
	Open-drain		1	10		0 or 1
Alternate Function output	Push-pull	1	0	11 see Table 21		Don't care
	Open-drain		1			Don't care
Input	Analog	0	0	00		Don't care
	Input floating		1			Don't care
	Input pull-down	1	0			0
	Input pull-up					1

Выбираем режим работы линии в/в – General purpose output Push-Pull (режим двухтактного выхода). Для настройки в этот режим линии в/в PC13 необходимо в регистр GPIOC_CRH в поле CNF13 [1:0] записать 00 (RM0008 стр. 172 подраздел 9.2.2), а в поле MODE13 [1:0] записать 11(максимальная скорость с которой может работать линия).

Определим адрес регистра GPIOC_CRH как сумму адреса начала области памяти ПВВ GPIOC – 0x4001 1000 (RM0008, Table 3 на странице 51) и смещения 0x04 (поле Address offset: 0x04 в описании GPIOC_CRH): 0x4001 1000 + 0x000 0004=0x4001 1004. И в код программы добавим строчки:

```
* (uint32_t*) (0x40011004) &= 0xFF0FFFFF;
* (uint32_t*) (0x40011004) |= 0x00300000;
```

Первая строка обнуляет поля CNF13 и MODE13 в регистре GPIOC_CRH, вторая устанавливает в поле MODE13 две единицы.

Настроив режим работы линии в/в PC13 на вывод, перейдём к непосредственному управлению состоянием сигнала на линии. Регистр GPIOC_ODR (RM0008 стр. 173 подраздел 9.2.4) управляет состоянием линий, настроенных в режим выхода. Установка 13-го бита (ODR13) в единицу – установит на линии PC13 высокий уровень, аброс в ноль – низкий уровень. Так как смещение этого регистра: 0x0C, получаем адрес GPIOC_ODR: 0x4001 1000 + 0x000 000C=0x4001 100C. Теперь в код программы добавим строчку:
`* (uint32_t*) (0x4001100C) |= 0x00002000;`. В целом должна получиться программа показанная на рисунке 3.2.

```
main.c
1 #include "RTE_Components.h"
2 #include CMSIS_device_header
3 int main ( ) {
4     *(uint32_t*)(0x40021018) |= 0x00000010;
5     *(uint32_t*)(0x40011004) &= 0xFF0FFFFF;
6     *(uint32_t*)(0x40011004) |= 0x00300000;
7     *(uint32_t*)(0x4001100C) |= 0x00002000;
8     while(1){}
9 }
```

Рис. 3.2. Программа установки на линии в/в PC13 (выводе МК №2) высокого состояния

Откомпилируйте и в режиме отладки разберите проводимые программой изменения. Дополнительно к регистрам подсистемы RCC откройте окно регистров ПВВ С: View >> System Viewer >> GPIO >> GPIOC.

3.3 Использование библиотеки CMSIS для доступа к регистрам

Работая с программой предыдущего раздела, приходилось быть очень внимательным, чтобы не допустить ошибку при наборе адреса или в константе, изменяющей нужный бит, особенно если отсутствуют комментарии. Таковы человеческие особенности восприятия, символьные названия воспринимаются гораздо лучше, чем цифры адресов и т.п. В микроконтроллерах на ядре компании ARM число регистров, а значит, и используемых ячеек, превышает тысячу. Чтобы упростить работу, необходимо произвести определение символьных указателей. Это определение выполнено компанией ARM в библиотеке CMSIS (Cortex Microcontroller Software Interface Standard) – стандарт программного интерфейса микроконтроллеров с ядром Cortex.

Выход на уровень аппаратной абстракции для регистров процессоров Cortex-M реализован унификацией обозначения регистров подсистем микроконтроллера с их обозначением в документации. Рассмотрим такой подход на примере подсистемы сброса и тактирования (RCC):

- 1) на странице 123 руководства RM0008 [13] начинается раздел №8 под названием ‘Connectivity line devices: reset and clock control (RCC)’. RCC – в библиотеке CMSIS mnemonicное название указателя на структуру, хранящую все регистры подсистемы тактирования;

- 2) на странице 146 руководства RM0008 начинается подраздел 8.3.7 под названием ‘PB2 peripheral clock enable register (RCC_APB2ENR)’. Используя мнемоническое название APB2ENR, доступ к регистру RCC_APB2ENR будет выглядеть как RCC->APB2ENR, а обозначение Bit 4 IOPCEN вот так RCC_APB2ENR_IOPCEN;
- 3) соответственно четвёртую строчку кода (см. рис. 3.2), при использовании библиотеки CMSIS, заменяем на строчку RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;.

В программе, при вводе мнемонических обозначений подсистем, ИСР Keil автоматически будет давать подсказку ввода в выпадающем контекстном меню. Также это меню можно вызвать самому, нажав комбинацию клавиш *Ctrl + Space*. Место объявления или определения, того или иного мнемонического обозначения, можно посмотреть наведя курсор на это слово, нажав правую кнопку мыши, выбрать в появившемся контекстном меню *Go To Definition of ‘слово’*.

В приведенном ниже коде программы, функции настройки регистров продублированы: сначала изменение регистра осуществляется через его адрес, далее следует повторное изменение, но уже с использованием акронимов библиотеки CMSIS.

```
#include "RTE_Components.h"
#include CMSIS_device_header
void delay(){
    volatile uint32_t count=55370;
    while(count--)
        NOP();
}
int main() //управляем PC13
{
    //Включаем тактирование порта GPIOC
    *(uint32_t*)(0x40021018)|=0x00000010;
    RCC->APB2ENR|=RCC_APB2ENR_IOPCEN;
    // Настраиваем PC13 как выход
    // Сбрасываем в ноль биты управления 13 выводом
    *(uint32_t*)(0x40011004)&=0xFF0FFFFF;
    GPIOC->CRH &= ~(GPIO_CRH_MODE13 | GPIO_CRH_CNF13);
    //MODE: выход с частотой 2 МГц
    //CNF: режим push-pull
    *(uint32_t*)(0x40011004)|=0x00200000;
    SET_BIT(GPIOC->CRH,GPIO_CRH_MODE13_1);
    for(;;){
        *(uint32_t*)(0x40011010)=0x00002000;
        GPIOC->BSRR=GPIO_BSRR_BS13;
        delay();
        *(uint32_t*)(0x40011014)=0x00002000;
        GPIOC->BRR=GPIO_BRR_BR13;
        delay();
    }
}
```

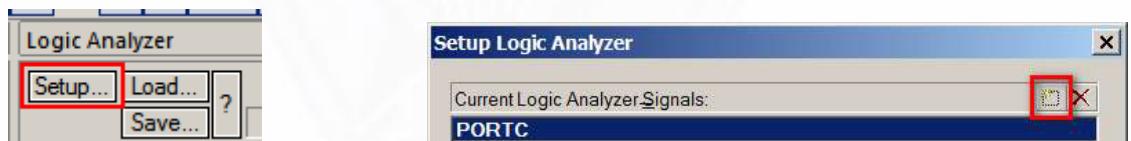
Откомпилируем и в отладочном режиме с помощью симуляции проверим частоту переключения выхода PC13. Для этого, с указанными ранее настройками вкладки Debug,

запускаем режим отладки *Ctrl+F5* (*Start/Stop Debug Session*). Предварительно настроим частоту внешнего осциллятора на частоту 8 МГц: в виртуальный регистр OSC, инструментом Watch, устанавливаем десятичное число 8000000

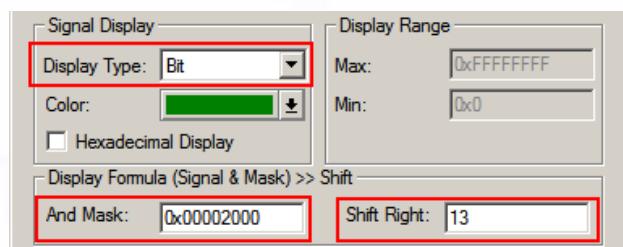
Name	Value
OSC	8000000

. Либо в окне Command вводим в командную строку `OSC=8000000`. Если OSC настроить на другую частоту, то дальнейшие измерения будут считаться неверными!

Выбираем окно логического анализатора: *View >> Analysis Windows >> Logic Analyzer*. В появившемся окне нажимаем *Setup...*, в окне конфигурации нажимаем *New (Insert key)*:



В строчку вводим виртуальный регистр (VTREG): PORTC (симуляция выводов ПВВ С), и нажимаем ввод. Далее настраиваем вывод 13го бита (линия PC13): Display Type выбираем Bit; And Mask вводим маску 0x00002000; Shift Right сдвиг вправо вводим 13. Нажимаем Close.



Теперь, нажатием F5(*Run*), запускаем работу симулятора и в окне логического анализатора наблюдаем переключение состояния линии PC13, как показано на рисунке 3.3. Остановим работу отладчика нажатием *Stop* (), включим инструмент *Cursor* и измерим время между переключениями. Как видно из рисунка 3.3 это время составляет 10ms (мсек) и соответственно период сигнала $T = 20$ мс, а частота $F = 1 / T = 1 / 0.02 = 50$ Гц.

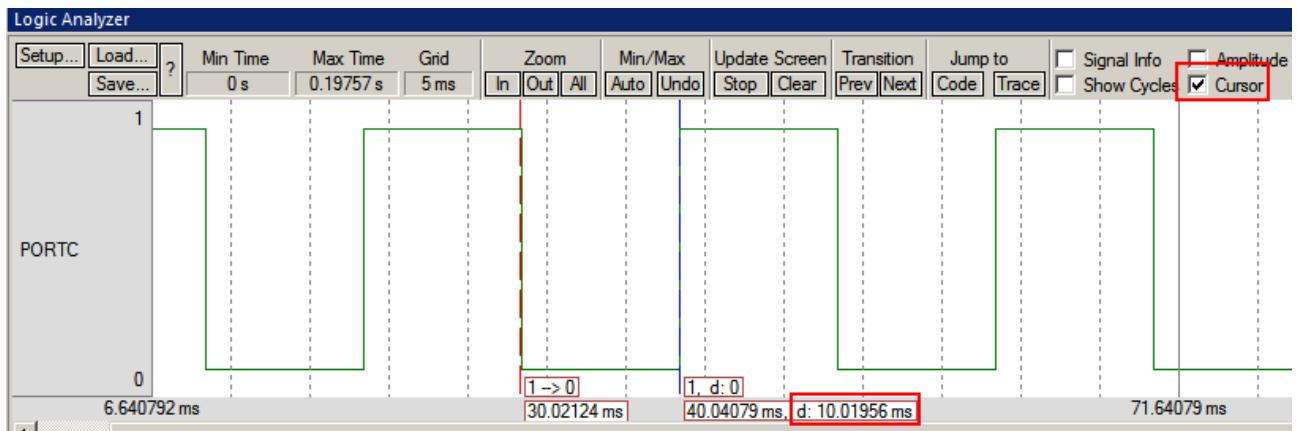


Рис. 3.3. Эпюра сигнала на выводе PC13

3.4 Практические задания к разделу 3

В соответствии с вариантом (табл. 3.1) нужно написать на языке «си» программу и отладить её работу по переключению уровня сигнала на двух линиях в/в микроконтроллера STM32F103C8T6. Значение частоты измерять инструментом отладчика *Logic Analyzer*. В программе один из выводов настраивать через регистры без использования библиотеки CMSIS. Значение счётчика задержки, под заданную вариантом частоту переключения линии в/в, подбирать вручную или рассчитать (тактовая частота 72МГц). Сохранить эпюры сигналов в отчёт.

Таблица 3.1

Варианты индивидуальных заданий к разделу 3

Номер варианта	Номера выводов (pin)	Частота переключений, Гц
1	43	2600; 5200
2	16	2040; 4080
3	12	3160; 6320
4	15	640; 1280
5	11	3300; 6600
6	10	1480; 2960
7	41	2460; 4920
8	43	920; 1840
9	11	2880; 5760
10	19	3020; 6040
11	42	1900; 3800
12	14	4000; 8000
13	15	1060; 2120
14	39	2180; 4360

Номер варианта	Номера выводов (pin)		Частота переключений, Гц
15	39	45	2320; 4640
16	18	25	1760; 3520
17	17	31	1200; 2400
18	40	45	3860; 7720
19	17	30	500; 1000
20	40	22	2740; 5480
21	19	21	780; 1560
22	13	29	3440; 6880
23	14	32	1620; 3240
24	13	33	3720; 7440
25	20	27	1340; 2680
26	41	27	3580; 7160

Отчёт должен содержать:

- 1) Номер варианта с заданием. Пример:

Вариант	16
Номера выводов	11, 19
Частота переключения	1760 Гц; 3520 Гц

Исходный код программы.

- 2) Таблицу трассировки двух заданных выводов STM32F103x8. Трассировку можно выполнить, используя схему [4]. Пример оформления для вывода №2:

Номера выводов	Обозначение согласно DS5319	Номера разъёмов и выводов на отладочной плате
2	PC13	1) штырь №2 на однорядном штыревом разъёме P3. 2) светодиод LED PC13.
x	P...	

- 3) Таблицу используемых регистров STM32F103x8 с расчётом адресов (с указанием на документацию) и управляемые биты. Пример оформления для линии PC13:

Подсистема/ Регистр	Расчёт адреса и ссылки на документацию	Биты и их назначение
RCC_APB2ENR	Адрес регистра RCC_APB2ENR: 0x40021000+0x18 = 0x40021018	Бит 4 IOPC EN: включает ПВВ С (GPIOC).
GPIOC_CRH	Адрес регистра GPIOC_CRH: 0x40011000+0x04 = 0x40011004	Конфигурация линии №13 ПВВ С. Поля бит: 23:22 CNF13[1:0] 21:20 MODE13[1:0] CNF13[1:0]=0x00 – режим двухтактного выхода (push-pull) MODE13[1:0]=0x10 – режим выхода с частотой переключения до 2 МГц.

GPIOC_BSRR	Адрес регистра GPIOC_BSRR: 0x40011000+0x10 = 0x40011010	Бит 13 BS13: устанавливает единицу на линии №13 ПВВ С (13й бит регистра GPIOC_ODR).
------------	--	---

- 4) Две эпюры сигналов на линиях в/в, по образцу рисунка 3.3. Снимок окна Watch со значением OSC. И таблица с характеристиками сигналов (частоты и периода):

Характеристика	Линия PC13	Линия Rx
Период, мс	20	...
Частота, Гц	50	...
Watch 1		
Name	Value	Type
OSC	8000000	ulong

Погрешность частоты сигналов допускается в пределах $\pm 1\%$ от заданной частоты.

3.5 Контрольные вопросы

- 1) Быть готовым пояснить любую букву и цифру в отчёте и программе.
- 2) Понятие стека, назначение, принцип работы, объявление в коде проекта.
- 3) Понятие сегмента памяти с неупорядоченным хранением данных (heap), назначение, объявление в коде проекта. Таблица векторов, назначение, объявление в коде проекта.
- 4) Карта памяти, найти в документации и пояснить назначение разделов.
- 5) Понятие сигналов логического нуля и единицы, понятие логической линии ввода/вывода. Связь состояния линии с состоянием регистра управления линией.
- 6) Понятие логического цифрового порта ввода/вывода. Порядок настройки линии порта: в режим двухтактного выхода; в режим входа.
- 7) Порядок создания проекта с подключением библиотеки CMSIS. Режим отладки, основные инструменты отладки при работе с ПВВ.
- 8) Понятие регистров управления подсистемами микроконтроллера, назначение, принцип работы. Названия основных регистров управления логическими портами ввода/вывода и их назначение.
- 9) Каков результат выполнения следующего кода:

```
1 int main(){
2     typedef unsigned int uint32_t;
3     *(uint32_t*)(0x40021018) |= 0x0000000C;
4     *(uint32_t*)(0x40010800) &= 0xFFFFFFFF0;
5     *(uint32_t*)(0x40010800) |= 0x00000002;
6     *(uint32_t*)(0x40010C04) &= 0xFFFFF0F;
7     *(uint32_t*)(0x40010C04) |= 0x00000030;
8     *(uint32_t*)(0x40010810) |= 0x00000001;
9     *(uint32_t*)(0x40010C0C) =*(volatile uint32_t*)(0x4001080C)<<9;
0 }
```

10) Каким образом возможно управление яркостью светодиода?

Санкт-Петербургский
государственный
университет
аэрокосмического
приборостроения

4 Настройка подсистемы тактирования микроконтроллера

4.1 Основные регистры управления подсистемой сброса и управления тактированием (RCC)

В DS5319 [12] на fig. 2 страницы 12 представлена структурная схема подсистемы тактирования RCC (продублирована на рис. 4.1), подобная схема представлена в разделе №8 RM0008 [13] на fig. 11. Схемы имеют различия, поскольку в RM0008 общее описание подобных МК, а в DS5319 конкретное описание на изделие STM32F103x8, поэтому все разнотечения толкуем в пользу схемы DS5319.

Анализ регистров управления подсистемой RCC подраздела 8.3 RM0008 показывает, что имеется три способа настройки системной частоты SYSCLK:

- 1) Тактирование напрямую от внутреннего генератора тактовой частоты HSI (High Speed Internal oscillator) с предопределенной частотой 8 МГц. На рисунке 4.1 блок ‘8MHz HSI RC’, выходной сигнал имеет обозначение HSI.
- 2) Тактирование от контура PLL (Phase-Locked Loop). Контур PLL – контур фазовой автоподстройки частоты (ФАПЧ). На рисунке 4.1 блок ‘x2, x3, x4 PLL’, выходной сигнал имеет обозначение PLLCLK. Контур ФАПЧ позволяет умножить входной сигнал в 2,3,...16 раз, при этом погрешность источника тактирования также умножается. Источником сигнала тактирования контура PLL может выступать сигнал ‘HSI/2’ или ‘HSE, HSE/2’.
- 3) Тактирование напрямую от внешнего генератора тактовой частоты HSE (High Speed External oscillator) с предопределенной частотой 8 МГц. На рисунке 4.1 блок ‘4-16 MHz HSE OSC’, выходной сигнал имеет обозначение HSE.

Настройка источника системной частоты SYSCLK осуществляется в регистре управления RCC_CFG (стр. 134 RM0008) полем SW (*System clock switch*) значениями бит:

- 00 : HSI выбран как источник системной частоты
- 01 : HSE выбран как источник системной частоты
- 10 : PLL выбран как источник системной частоты

При первом и третьем способе настройки значение системной частоты будет одинаковым и равно 8 МГц, просто источники тактового сигнала будут разные. Вся гибкость настройки системной частоты определяется вторым способом – контуром ФАПЧ (PLL).

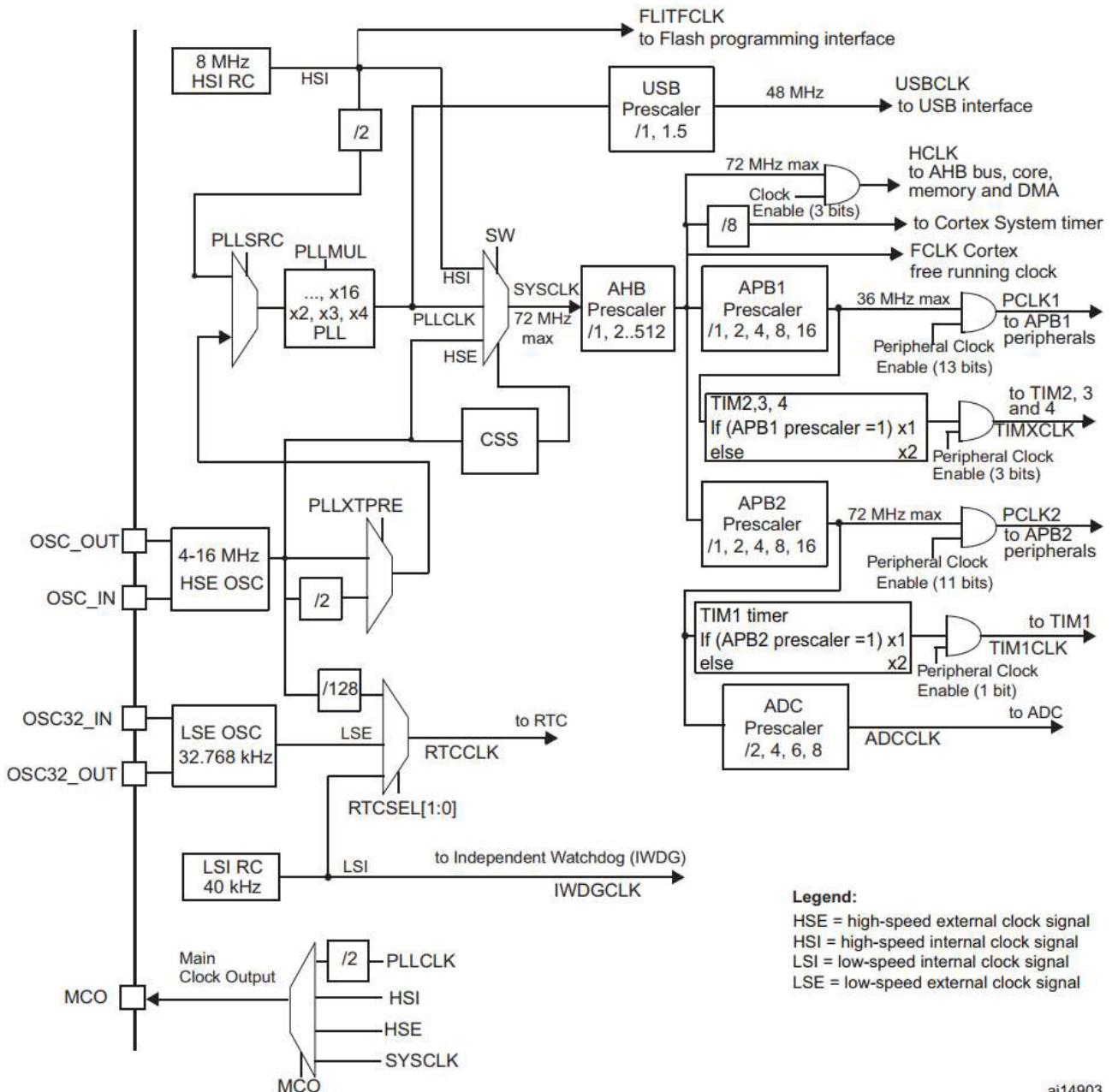


Рис. 4.1. Структурная схема подсистемы RCC

4.2 Пример программной настройки системной частоты

Рассмотрим управляющие поля регистров для настройки частоты тактирования PLLCLK (настраиваются в выключенном состоянии PLL):

- Поле PLLMUL (PLL multiplication factor - стр. 135 RM0008 [13]) регистра RCC_CFG (биты 21:18). Определяет коэффициент умножения входного источника тактовой частоты PLL согласно таблице 4.1.

Таблица 4.1

Управляющие биты настройки коэффициента умножения PLL

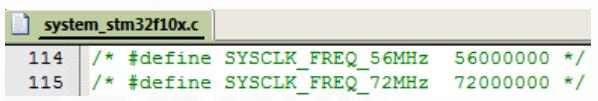
Биты RCC_CFGR	Коэффициент умножения	Биты RCC_CFGR	Коэффициент умножения
21 20 19 18		21 20 19 18	
0 0 0 0	x 2	1 0 0 0	x 10
0 0 0 1	x 3	1 0 0 1	x 11
0 0 1 0	x 4	1 0 1 0	x 12
0 0 1 1	x 5	1 0 1 1	x 13
0 1 0 0	x 6	1 1 0 0	x 14
0 1 0 1	x 7	1 1 0 1	x 15
0 1 1 0	x 8	1 1 1 0	x 16
0 1 1 1	x 9	1 1 1 1	x 16

2) Поле PLLSRC регистра RCC_CFGR (бит 16). Значение бита определяет источник тактирования PLL: **0** – источник HSI/2; **1** - источник HSE или HSE/2. Таким образом, PLL, либо тактируется от HSI частотой 8/2=4 МГц, либо от HSE частотой 8 МГц или 4 МГц.

3) Поле PLLXTPRE регистра RCC_CFGR (бит 17) делит HSE/2 на входе PLL.

Продемонстрируем на примере варианта 37 таблицы 4.2. Создаём проект для МК STM32F103C8. На шаге настройки работы с библиотеками ARM MDK-Professional, выбираем пункты: CMSIS >> CORE; Device >> Startup; Compiler >> I/O >> STDOUT в выпадающем меню выбрать ITM.

На вкладке Debug выбираем Use Simulator. В поле Dialog DLL и Parameter вводим DARMSTM.DLL и -pSTM32F103C8 , соответственно. В созданном проекте откроем файл *system_stm32f10x.c* и закомментируем строчку:



```
#define SYSCLK_FREQ_72MHz 72000000
/* #define SYSCLK_FREQ_56MHz 56000000 */
/* #define SYSCLK_FREQ_48MHz 48000000 */
```

Создадим файл main.c и вставим в него следующий код:

```
#include "RTE_Components.h"
#include CMSIS_device_header
#include <stdio.h>
int main()
{
    volatile uint32_t StartUpCounter = 0, HSEStatus = 0;
    //проверяем установленную частоту тактирования по умолчанию
    SystemCoreClockUpdate(); //устанавливается в глобальной переменной SystemCoreClock
    ITM_SendChar('\n');
    printf("Start clk=%d Hz\n", SystemCoreClock);
    // Bit 16 HSEON: -> RCC_CR_HSEON
    // 0: HSE oscillator OFF; 1: HSE oscillator ON;
    SET_BIT(RCC -> CR, RCC_CR_HSEON); // включаем HSE
    do { // ждем вхождения в работу HSE
        HSEStatus = RCC->CR & RCC_CR_HSERDY;
        StartUpCounter++;
    } while((HSEStatus == 0) && (StartUpCounter != 0x5000));
    //если за 0x5000 итераций, HSE не запустился, то проблемы в аппаратуре
    if ((RCC->CR & RCC_CR_HSERDY) != RESET) //HSE работает
```

```

{
    // настраиваем FLASH, время предварительной выборки в буфер команд
    //000: Zero wait state, if 0 < HCLK ≤ 24 MHz -> FLASH_ACR_LATENCY_0
    //001: One wait state, if 24 MHz < HCLK ≤ 48 MHz-> FLASH_ACR_LATENCY_1
    //010: Two wait states, if 48 < HCLK ≤ 72 MHz      -> FLASH_ACR_LATENCY_2
    // 0: Prefetch is disabled
    // 1: Prefetch is enabled   -> FLASH_ACR_PRFTBE
    FLASH->ACR = FLASH_ACR_PRFTBE|FLASH_ACR_LATENCY_2;
    // HCLK = SYSCLK / ...
    // 1011: SYSCLK divided by 16   -> RCC_CFGR_HPRE_DIV16
    RCC->CFGR |= RCC_CFGR_HPRE_DIV16;//AHB Pre = 16 согласно варианта
    // настройка PLL на 72 МГц = 8 МГц(HSE) x 9
    // сначала выключаем чтобы изменить биты PLL, после настройки включим
    CLEAR_BIT(RCC -> CR,RCC_CR_PLLON);
    // Bit 16 PLLSRC: -> RCC_CFGR_PLLSRC
    // 0: HSI/2 selected as PLL input clock // 1: HSE selected as PLL input clock
    // Bits 21:18 PLLMUL: -> RCC_CFGR_PLLMUL
    // 0111: PLL input clock x 9   -> RCC_CFGR_PLLMUL9
    // Bit 17 PLLXTPRE: -> RCC_CFGR_PLLXTPRE
    // 0: HSE clock not divided   // 1: HSE clock divided by 2
    /* PLL configuration: PLLCLK = HSE * 9 = 72 MHz */
    RCC->CFGR &= ~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLMUL);
    RCC->CFGR |= (RCC_CFGR_PLLSRC_HSE | RCC_CFGR_PLLMUL9);
    // включаем PLL Bit 24 PLLON: -> RCC_CR_PLLON
    // 0: PLL OFF;   1: PLL ON
    SET_BIT(RCC -> CR,RCC_CR_PLLON);
    // ждём запуск и стабилизацию PLL
    while((RCC->CR & RCC_CR_PLLRDY) == 0){}
    // выбираем выход PLL источником тактирования МК
    RCC->CFGR &= ~(RCC_CFGR_SW);
    RCC->CFGR |= RCC_CFGR_SW_PLL;
    //Ожидаем установки PLL источником тактирования МК
    while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL){}
}
else
{ while(1){} } // HSE не запустился
SystemCoreClockUpdate(); //контролируем частоту 72/16=4.5 МГц
printf("After configuration clk=%d Hz\n",SystemCoreClock);
// Настройка MCO на PLLCLK/2
// Bits 26:24 MCO:
// 100: System clock (SYSCLK) selected   RCC_CFGR_MCO_SYSCLK
// 111: PLL/2 clock selected           RCC_CFGR_MCO_PLL
SET_BIT(RCC -> CFGR, RCC_CFGR_MCO_PLL);
SET_BIT(RCC -> APB2ENR,RCC_APB2ENR_IOPAEN); //разрешаем тактирование GPIOA
// Сбрасываем в ноль биты управления выводом PA8
GPIOA->CRH &= ~(GPIO_CRH_MODE8 | GPIO_CRH_CNF8);
//MODE: выход с макс. частотой 50 МГц
//CNF: режим Alternate function output Push-pull
SET_BIT(GPIOA->CRH,GPIO_CRH_MODE8|GPIO_CRH_CNF8_1);
//разрешаем тактирование GPIOC PC13
SET_BIT(RCC->APB2ENR, RCC_APB2ENR_IOPCEN);
//устанавливаем работу линий PC13 и PC14 на вывод
GPIOC->CRH &= ~(GPIO_CRH_MODE13 | GPIO_CRH_CNF13|GPIO_CRH_MODE14 | GPIO_CRH_CNF14);
SET_BIT(GPIOC->CRH,GPIO_CRH_MODE13); // High speed
SET_BIT(GPIOC->CRH,GPIO_CRH_MODE14_1); // Low speed
while(1){ //устанавливаем 1 на выходе линий PC13,14
    GPIOC->BSRR= GPIO_BSRR_BS13|GPIO_BSRR_BS14;
    //сбрасываем в 0 выходы линий PC13,14
    GPIOC->BRR = GPIO_BRR_BR13|GPIO_BRR_BR14;}
}

```

Далее откомпилируем и включим режим отладки. Установим частоту симуляции внешнего источника тактирования напрямую, через виртуальный регистр OSC: в окне Command введите команду OSC=8000000 или добавьте регистр OSC в инструмент Watch и установите значение 8000000.

Проверим с помощью симуляции настройку частоты тактирования и частоту переключения выходов PC13 и PC14. Для начала откроем окно Debug (printf) Viewer: заходим в меню View >> Serial Windows >> Debug (printf) Viewer. Далее в окно Watch (список наблюдаемых переменных) добавим виртуальные регистры (VTREG) отображающие состояние модели симулятора:

Регистр **OSC** – частота HSE.

Регистр **HSI_RC** – частота HSI.

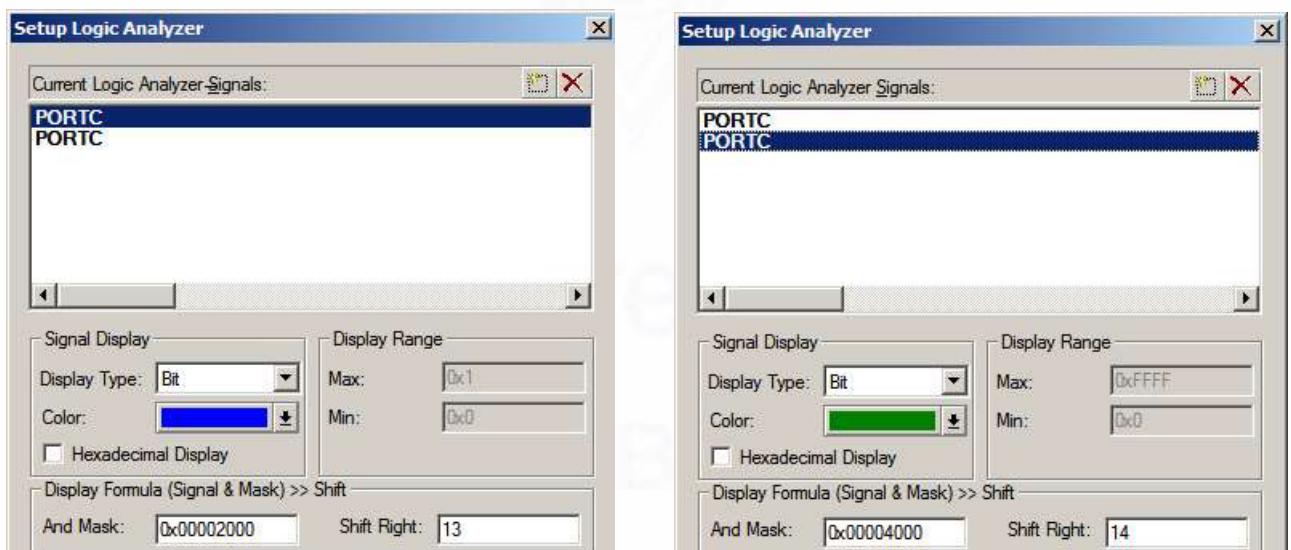
Регистр **SYSCLK** – системная частота.

Регистр **HCLK** – частота тактирования ядра.

Переменную **SystemCoreClock**, в которой после вызова функции **SystemCoreClockUpdate()** сохраняется частота тактирования ядра МК.

Name	Value
OSC	8000000
HSI_RC	8000000
SYSCLK	72000000
HCLK	4500000
PORTC	0x6000
SystemCoreClock	4500000
GPIOC->ODR	0x00006000

Далее через Setup... Logic Analyzer настраиваем отображение сигнала на линиях PC13 и PC14:



Запускаем симуляцию F5(**Run**) и наблюдаем в окне Debug (printf) Viewer:

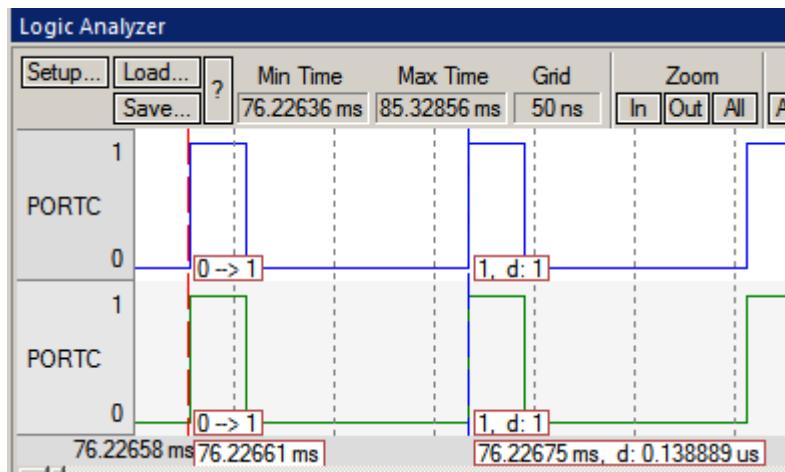
В начале программы частота тактирования ядра была 8МГц, после программной настройки стала $72\text{MГц} / 16 = 4,5\text{MГц}$.

```
Debug (printf) Viewer
Start clk=8000000 Hz
After configuration clk=4500000 Hz
```

Частоту переключения выводов PC13 и PC14 измеряем в окне Logic Analyzer:

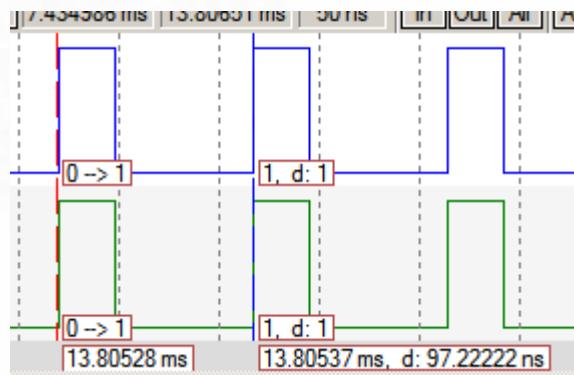
- период составляет 138.9 нс;
- частота $1/138.9 \text{ нс} = 7.2 \text{ МГц}$

Настройки скорости переключения выводов – High/Low speed, симулятором игнорируются.



С включением оптимизации компилятора Level 3 (-O3) получаем следующие измерения:

- период составляет 97.22222 нс;
- частота $1 / 97.22 \text{ нс} \approx 10,286 \text{ МГц}$



В Keil, при симуляции STM32F103C8, модели подсистем не согласованы, поэтому настройка HCLK = 4,5МГц не влияет на скорость работы порта в/в. Линии переключаются с частотой 7.2 МГц, что выше HCLK, т.е. симулятор переключает линию исходя из значения частоты SYSCLK = 72МГц и игнорирует настройку остальных частот HCLK, PCLK1 и др.

4.3 Практические задания к разделу 4

В соответствии с вариантом (табл. 4.2) нужно написать на языке «си» программу и отладить её работу, в которую входит:

- настройка частоты тактирования МК согласно варианту;
- переключение в цикле заданного вывода;

в) настройка МСО на вывод сигнала согласно варианту. Вывод сигнала МСО на линию PA8 в симуляторе отсутствует, поэтому правильность настройки не проверить.

Используя инструмент Logic Analyzer измерить эпюру сигнала заданного вывода. Включить оптимизацию компиляции выставив опцию Optimization: в положение -O3. Пересобрать проект, снова измерить эпюру сигнала, сравнить с полученной ранее.

Таблица 4.2

Варианты индивидуальных заданий к разделу 4

Номер варианта	Частоты для настройки SYSCLK (источник (I)-HSI или (E)-HSE)			выводов (ножек-pin)
	МГц	AHB Pre	MCO	
1	48 (I)	8	HSE	PA9
2	56 (E)	8	HSI	PA4
3	64 (I)	8	HSE	PA6
4	12 (I)	2	HSE	PB5
5	20 (I)	2	HSE	PB14
6	16 (E)	2	SYSCLK	PB7
7	52 (I)	8	HSE	PA7
8	24 (E)	2	HSI	PB0
9	28 (E)	4	SYSCLK	PA5
10	28 (I)	4	HSE	PA2
11	20 (E)	2	HSI	PA0
12	64 (E)	8	SYSCLK	PB9
13	60 (E)	8	SYSCLK	PA12
14	12 (E)	2	HSI	PB11
15	36 (I)	4	HSE	PA3
16	32 (E)	4	HSI	PB15
17	52 (E)	8	HSI	PA8
18	16 (I)	2	SYSCLK	PB1
19	44 (E)	4	HSI	PA1
20	8 (I)	2	HSE	PB13
21	44 (I)	8	SYSCLK	PB10
22	32 (I)	4	HSE	PB6
23	8 (E)	2	HSI	PB4
24	40 (I)	4	SYSCLK	PB12
25	24 (I)	4	HSE	PA11
26	40 (E)	4	HSI	PA10
27	36 (E)	4	SYSCLK	PA15
28	48 (E)	8	SYSCLK	PB8
Вариант примера				
37	72(E)	16	PLLCLK/2	PC13, PC14

Отчёт должен содержать:

- 1) Номер варианта с заданием. Листинг программы.

- 2) Отпечаток окна Watch со значениями OSC, HSI_RC, SYSCLK, HCLK, SystemCoreClock. Отпечаток окна Debug (printf) Viewer.
- 3) Эпюры сигнала без оптимизации компилятора (-O0) и с уровнем оптимизации компилятора (-O3), с расчётом частоты переключения.
- 4) Маршрут настройки частоты согласно варианту, на основе технического описания DS5319 [12], пример показан на рисунке 4.2.

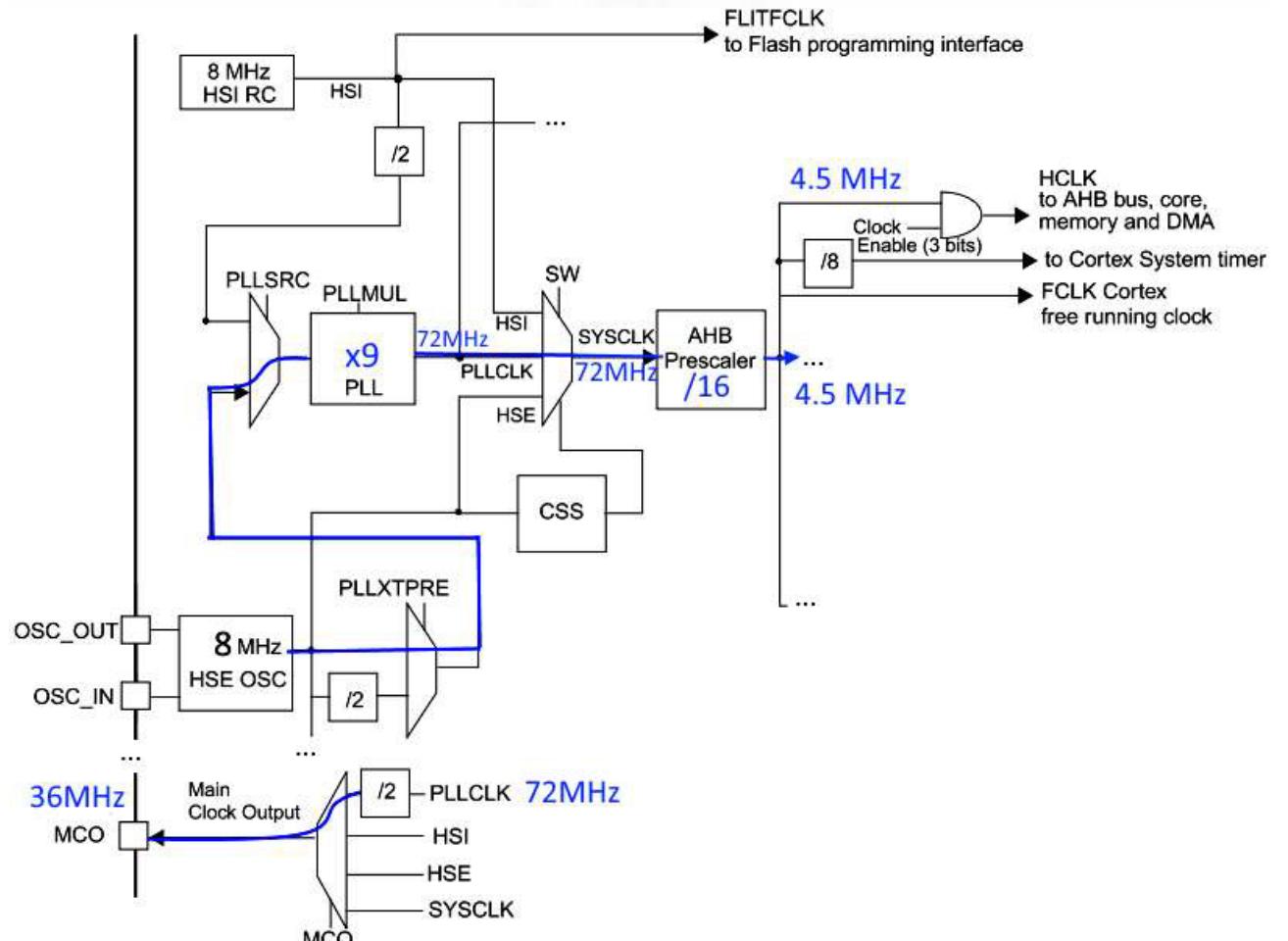


Рис. 4.2. Маршрут настройки частоты $SYSCLK=72$ МГц

4.4 Контрольные вопросы

- 1) Назначение подсистемы сброса и тактирования RCC. Структурная схема сброса.
- 2) Назначение HSI, HSE, LSI, LSE. Используемые регистры для управления.

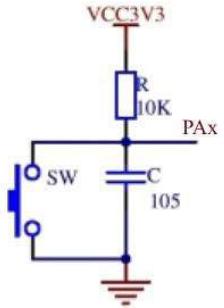
- 3) Назначение PLL, MCO, FLASH_ACR. Алгоритм настройки PLL, используемые регистры.
- 4) Частоты PLLCLK, SYSCLK, HCLK, PCLK1, PCLK2. Принцип формирования.
- 5) Схемы настройки тактирования от HSI. Список возможных частот для настройки.
Используемые регистры.
- 6) Схемы настройки тактирования от HSE. Список возможных частот для настройки.
Используемые регистры.
- 7) Рассчитать коэффициенты настройки подсистемы тактирования для настройки PCLK1 или PCLK2 на частоту, указанную преподавателем.

Санкт-Петербургский
государственный
университет
аэрокосмического
приборостроения

5 Настройка системы прерываний микроконтроллера

5.1 Симулятор нажатия кнопок в проекте

Будем считать, что на входах PA2, PA3, PA4, PA6 (вариант 37 табл. 5.1) подключены кнопки по схеме справа. При этом линии PA2, PA3, PA4, PA6 настраиваем как входы с подключенным pull up резистором. Поскольку, при нажатии реальной кнопки уровень сигнала на PAx изменится с VCC (лог.1) на GND (лог.0), то в симуляции будем изменять сигнал на входе с 1 на 0 на короткое время.



Продемонстрируем, как в Keil происходит подобная эмуляция. При создании проекта выбираем тип MK STM32F103C8. На шаге настройки работы с библиотеками ARM MDK-Professional, выбираем пункты: CMSIS >> CORE; Device >> Startup; Compiler >> I/O >> STDOUT в выпадающем меню ITM и нажимаем OK.

На вкладке Debug выбираем Use Simulator. В поле Dialog DLL и Parameter вводим DARMSTM.DLL и -pSTM32F103C8, соответственно.

Наполним файл main.c кодом:

```
#include "RTE_Components.h" // Component selection
#include CMSIS_device_header // Device header
#include <stdio.h>
int main(){
    uint32_t cnt=0;
    SystemCoreClockUpdate();
    printf("clk=%d Hz\n",SystemCoreClock);
    RCC->APB2ENR|=RCC_APB2ENR_IOPAEN;
    //PA2,3,4,6 Input, Pull up
    GPIOA->CRL = 0;
    SET_BIT(GPIOA->CRL,
    GPIO_CRL_CNF2_1|GPIO_CRL_CNF3_1|GPIO_CRL_CNF4_1|GPIO_CRL_CNF6_1);
    SET_BIT(GPIOA->ODR, GPIO_ODR_ODR2|GPIO_ODR_ODR3|GPIO_ODR_ODR4|GPIO_ODR_ODR6);
    //pull up
    while(1){
        if((GPIOA->IDR&0x04)==0)
            printf("%9i press PA2\n",cnt++);
        if((GPIOA->IDR&0x08)==0)
            printf("%9i press PA3\n",cnt++);
        if((GPIOA->IDR&0x10)==0)
            printf("%9i press PA4\n",cnt++);
        if((GPIOA->IDR&0x40)==0)
            printf("%9i press PA6\n",cnt++);
    }
}
```

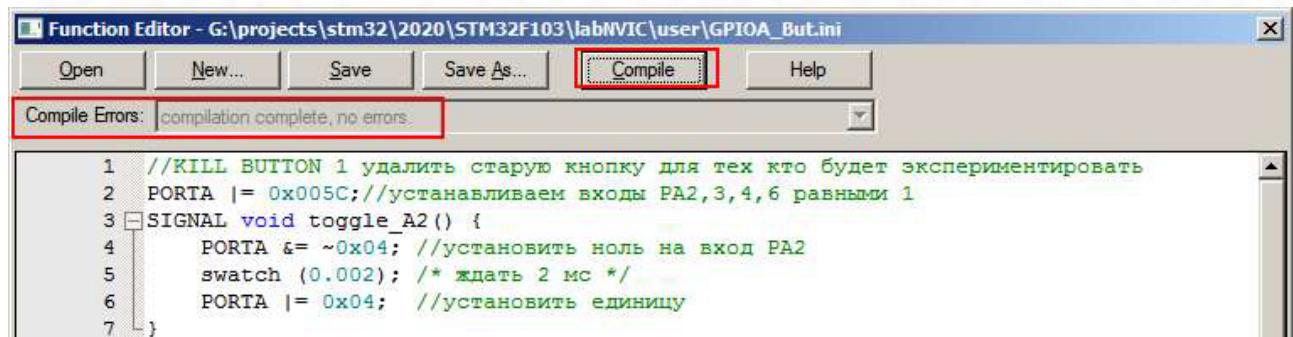
Откомпилируем и запустим отладку. Находясь в режиме отладки из меню Debug >> Function Editor (Open Ini File) . . . вызываем редактор функций. Он сразу откроет диалог открытия INI файла. Закроем диалог, нажмем кнопку New, далее Save as сохраним

под нужным именем (например, GPIOA_But.ini) в каталоге программы, и затем заново откроем его в редакторе функций. Вставляем следующий скрипт:

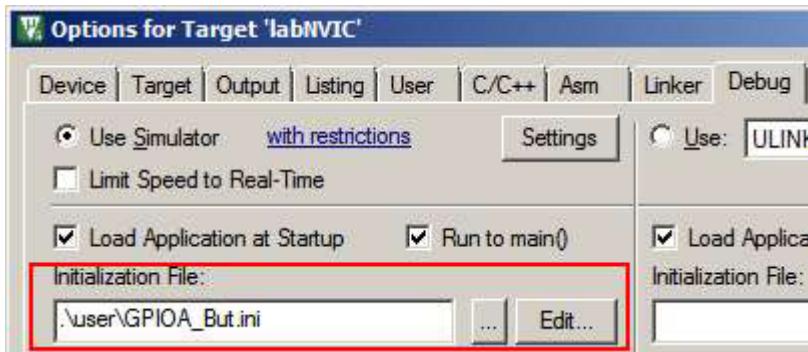
```
//KILL BUTTON 1 //удалить старую кнопку для тех, кто будет экспериментировать
OSC = 8000000; //установили частоту внешнего генератора Xtal (Hz)
PORTA |= 0x005C; //устанавливаем входы PA2,3,4,6 равными 1
SIGNAL void toggle_A2() {
    PORTA &= ~0x04; //установить ноль на вход PA2
    swatch (0.02); /* ждать 20 мс */
    PORTA |= 0x04; //установить единицу
}
SIGNAL void toggle_A3() {
    PORTA &= ~0x08; //установить ноль на вход PA3
    swatch (0.02);
    PORTA |= 0x08; //установить единицу
}
SIGNAL void toggle_A4() {
    PORTA &= ~0x10; //установить ноль на вход PA4
    swatch (0.02);
    PORTA |= 0x10; //установить единицу
}
SIGNAL void toggle_A6() {
    PORTA &= ~0x40; //установить ноль на вход PA6
    swatch (0.02);
    PORTA |= 0x40; //установить единицу
}
//создать кнопки Key PAx и привязать к ней функцию toggle_Ax()
DEFINE BUTTON "Key PA2", "toggle_A2()"
DEFINE BUTTON "Key PA3", "toggle_A3()"
DEFINE BUTTON "Key PA4", "toggle_A4()"
DEFINE BUTTON "Key PA6", "toggle_A6()"
```

Здесь мы определяем сигнальные функции, в которых переключаем нужный нам вход в ноль на 20 мс. Далее командой DEFINE создаём кнопку и сопоставляем её с функцией.

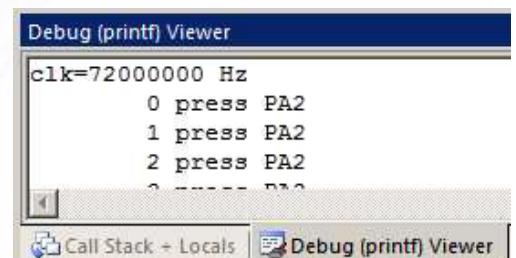
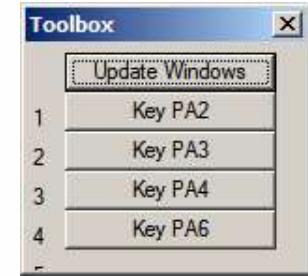
Сохраним файл Save и скомпилируем его Compile, убедимся, что ошибок нет, в строке Compile Errors: — compilation complete, no errors. В командном окне отладчика (окно Command) также не должно быть сообщений об ошибке:



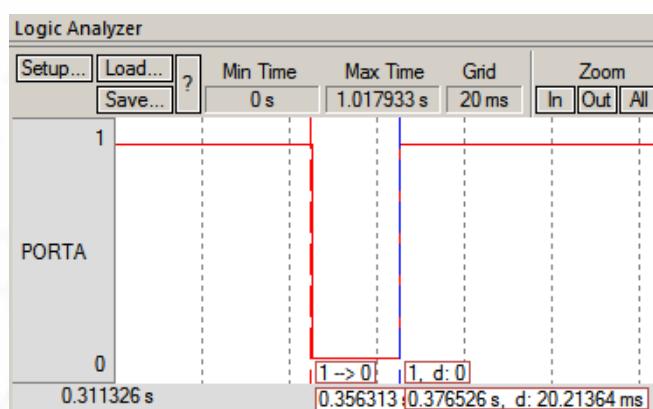
Далее выйдем из режима отладки, откроем настройки проекта и на вкладке Debug укажем название файла GPIOA_But.ini в строке Initialization File:



Теперь запустив отладку, выберем меню View >> ToolBox Window и на поле отладчика появиться окно Toolbox с четырьмя кнопками. Откроем окно вывода отладчика View >> Serial Windows >> Debug (printf) Viewer. И запустим работу Run(F5). При нажатии кнопки на Toolbox будет изменяться сигнал на соответствующем входе, и соответственно будет изменяться значение регистра GPIOA->IDR. На что будут реагировать условия в теле цикла while(1) и в окне вывода Debug (printf) Viewer отобразиться счётчик и название кнопки.



Также можно наблюдать изменение сигналов инструментом Logic Analyzer:



Если сигнал в окне Logic Analyzer отображается только после остановки выполнения, включите опцию View >> Periodic Window Update.

5.2 Пример работы с прерываниями в симуляторе STM32F103x8

Разобравшись как настраивать кнопки для изменения состояния нужных нам линий, перейдём непосредственно к программе варианта №37 таблицы 5.1. Настроим внешние прерывания EXTI на четыре входа линий PA2, 3, 4, 6. Не меняя группировку приоритетов, установим следующие приоритеты 111,110,108,109 для соответствующих входов.

Программа (файл main.c):

```
#include "RTE_Components.h" // Component selection
#include CMSIS_device_header // Device header
#include <stdio.h>
void delay(void){
    volatile uint32_t i=600000;//задержка
    while(i > 0)
        i--;
}
int main(void){
    uint32_t priGroup = 0, PreemptPriority=0, SubPriority=0;
    SystemCoreClockUpdate();
    printf("clk=%d Hz\n",SystemCoreClock);
    RCC->APB2ENR|=RCC_APB2ENR_IOPAEN|RCC_APB2ENR_IOPCEN;//разрешаем работу GPIO A,C
    GPIOC->CRH &= ~(GPIO_CRH_MODE13 | GPIO_CRH_CNF13);
    SET_BIT(GPIOC->CRH,GPIO_CRH_MODE13); //PC13 в режим двухтактного выхода
    GPIOA->CRL = 0;//PA2,3,4,6 Input, Pull up
    SET_BIT(GPIOA->CRL, GPIO_CRL_CNF2_1|GPIO_CRL_CNF3_1|GPIO_CRL_CNF4_1|GPIO_CRL_CNF6_1);
    SET_BIT(GPIOA->ODR, GPIO_ODR_ODR2|GPIO_ODR_ODR3|GPIO_ODR_ODR4|GPIO_ODR_ODR6); //pull up
    SET_BIT(RCC->APB2ENR, RCC_APB2ENR_AFIOEN); //включаем альтернативный режим для ПВВ
    //выбираем в качестве внешних входов EXTI линии:
    AFIO->EXTICR[0] = AFIO_EXTICR1_EXTI2_PA|AFIO_EXTICR1_EXTI3_PA; //линии PA2>>EXTI2, PA3>>EXTI3
    AFIO->EXTICR[1] = AFIO_EXTICR2_EXTI4_PA|AFIO_EXTICR2_EXTI6_PA; //линии PA4>>EXTI4, PA6>>EXTI6
    //NVIC_SetPriorityGrouping(7);
    priGroup = NVIC_GetPriorityGrouping();
    printf("Priority Group=%d\r\n",priGroup);
    NVIC_SetPriority(EXTI2_IRQn,111);
    NVIC_DecodePriority(NVIC_GetPriority(EXTI2_IRQn),priGroup,&PreemptPriority,&SubPriority);
    printf("EXTI2 Preempt Priority=%d \tSubPriority=%d\r\n",PreemptPriority,SubPriority);
    NVIC_SetPriority(EXTI3_IRQn,110);
    NVIC_DecodePriority(NVIC_GetPriority(EXTI3_IRQn),priGroup,&PreemptPriority,&SubPriority);
    printf("EXTI3 Preempt Priority=%d \tSubPriority=%d\r\n",PreemptPriority,SubPriority);

    NVIC_SetPriority(EXTI4_IRQn,108);
    NVIC_DecodePriority(NVIC_GetPriority(EXTI4_IRQn),priGroup,&PreemptPriority,&SubPriority);
    printf("EXTI4 Preempt Priority=%d \tSubPriority=%d\r\n",PreemptPriority,SubPriority);
    NVIC_SetPriority(EXTI9_5_IRQn,109);
    NVIC_DecodePriority(NVIC_GetPriority(EXTI9_5_IRQn),priGroup,&PreemptPriority,&SubPriority);
    printf("EXTI9_5 Preempt Priority=%d \tSubPriority=%d\r\n",PreemptPriority,SubPriority);
    //прерывание по срезу сигнала
    SET_BIT(EXTI->FTSR,EXTI_FTSR_TR2|EXTI_FTSR_TR3|EXTI_FTSR_TR4|EXTI_FTSR_TR6);
    //разрешаем прерывания внешних линий 2,3,4,6
    SET_BIT(EXTI->IMR,EXTI_IMR_MR2|EXTI_IMR_MR3|EXTI_IMR_MR4|EXTI_IMR_MR6);
    NVIC_EnableIRQ(EXTI2_IRQn);
    NVIC_EnableIRQ(EXTI3_IRQn);
    NVIC_EnableIRQ(EXTI4_IRQn);
    NVIC_EnableIRQ(EXTI9_5_IRQn);
    SysTick_Config(0x6DDDD00); //прерывание таймера каждые 100мс
    while(1){}
}
void SysTick_Handler(void){//обработчик прерывание системного таймера
    GPIOC->ODR=~1<<13; //переключаем линию PC13 каждые 100мс
}
void EXTI2_IRQHandler(void){
    EXTI->PR = EXTI_PR_PR2;
    ITM_SendChar('2');
    delay();
    ITM_SendChar('a');
    ITM_SendChar('\n');
}
void EXTI3_IRQHandler(void){
    EXTI->PR = EXTI_PR_PR3;
    ITM_SendChar('3');
    delay();
    ITM_SendChar('b');
    ITM_SendChar('\n');
}
void EXTI4_IRQHandler(void){
```

```

EXTI->PR = EXTI_PR_PR4;
ITM_SendChar('4');
delay();
ITM_SendChar('c');
ITM_SendChar('\n'); }

void EXTI9_5_IRQHandler(void){
EXTI->PR = EXTI_PR_PR6;
ITM_SendChar('6');
delay();
ITM_SendChar('d');
ITM_SendChar('\n'); }

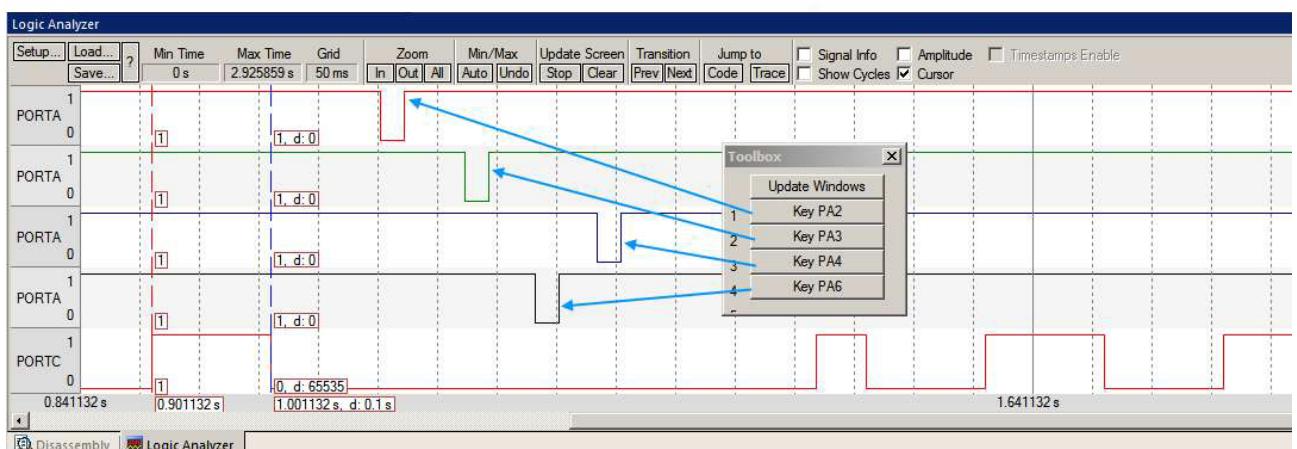
```

Откомпилируем и запустим отладку. Файл GPIOA_But.ini оставляем в настройках вкладки Debug. Откроем необходимы для исследования работы прерываний инструменты: View >> ToolBox Window – окно с кнопками для вызова прерываний; View >> Serial Windows >> Debug (printf) Viewer – окно для вывода сообщений; View >> Analysis Windows >> Logic Analyzer – инструмент графического отображения состояния входов/выходов.

В окне Logic Analyzer настроим нужные линии:

Current Logic Analyzer Signals	Display Type	And Mask	Shift Right
PORTC	Bit	0x000002000	13
PORTA	Bit	0x00000004	2
PORTA	Bit	0x00000008	3
PORTA	Bit	0x00000010	4
PORTA	Bit	0x00000040	6

Продолжим выполнение кода в режиме отладки (F5 Run). При нажатии кнопок в окне ToolBox, наблюдаем в окне Logic Analyzer изменения сигналов соответствующих ВХОДОВ:



где стрелками указаны моменты поступления прерываний по линиям EXTI2, EXTI3, EXTI4, EXTI6 (моменты нажатия кнопок Key PA2, 3, 4, 6). И соответствующие сообщения в окне Debug (printf) Viewer, которые означают:

```

Debug (printf) Viewer
clk=72000000 Hz
Priority Group=0
EXTI2 Preempt Priority=15 SubPriority=0
EXTI3 Preempt Priority=14 SubPriority=0
EXTI4 Preempt Priority=12 SubPriority=0
EXTI6 Preempt Priority=13 SubPriority=0
2364c
d
b
a

```

Расшифровка:

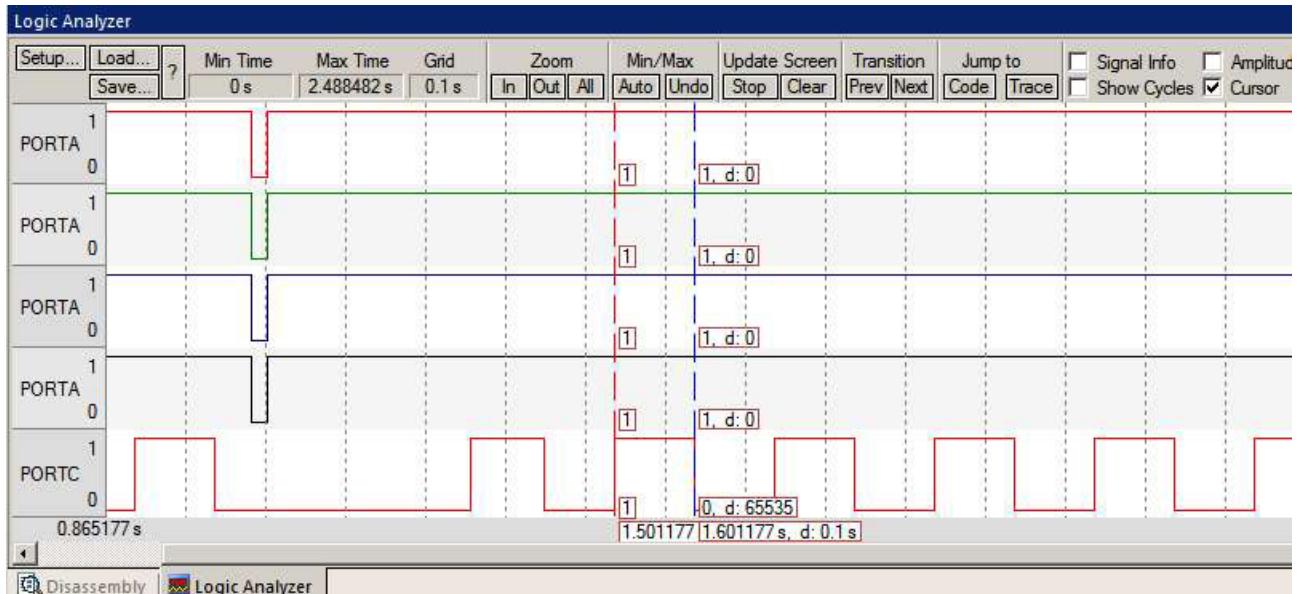
- 2 – вход в обработчик прерывания *EXTI2*
- 3 – вход в обработчик прерывания *EXTI3*
- 6 – вход в обработчик прерывания *EXTI6*
- 4 – вход в обработчик прерывания *EXTI4*
- c – завершение обработчика *EXTI4*
- d – завершение обработчика *EXTI6*
- b – завершение обработчика *EXTI3*
- a – завершение обработчика *EXTI2*

Обработчик прерывания *EXTI2* был прерван прерыванием *EXTI3* в момент его поступления. Далее *EXTI3* был прерван *EXTI6*, в свою очередь *EXTI6* был прерван *EXTI4*. Далее по завершению работы обработчика прерывания *EXTI4* стал выполняться обработчик прерывания *EXTI6*, после него *EXTI3*, и в завершении *EXTI2*.

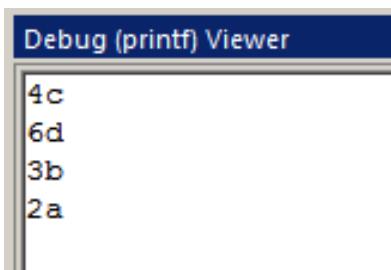
Как видно из эпюры сигнала на линии PC13 (нижняя строка в окне Logic Analyzer), на время работы обработчиков прерываний *EXTI2, 3, 4, 6* прерывания системного таймера не обслуживались из-за низкого приоритета равного 15:

Idx	Source	Name	E	P	A	Priority
15	System Tick Timer	SYSTICK	1	0	0	15
16	Window Watchdog	WWDG	0	0	0	0
17	PVD through EXTI	PVD	0	0	0	0
18	TAMPER Interrupt	TAMPER	0	0	0	0
19	RTC Global Interrupt	RTC	0	0	0	0
20	Flash Global Interrupt	FLASH	0	0	0	0
21	RCC Global Interrupt	RCC	0	0	0	0
22	EXTI Line0 Interrupt	EXTI0	0	0	0	0
23	EXTI Line1 Interrupt	EXTI1	0	0	0	0
24	EXTI Line2 Interrupt	EXTI2	1	0	0	15
25	EXTI Line3 Interrupt	EXTI3	1	0	0	14
26	EXTI Line4 Interrupt	EXTI4	1	0	0	12

Для моделирования события возникновения одновременно четырёх внешних прерываний, остановим выполнение кода нажатием Stop () и нажмём подряд все четыре кнопки в окне ToolBox. Далее возобновим выполнение кода в режиме отладки (F5 Run). В окне Logic Analyzer изменения сигналов теперь выглядят как одновременное возникновение четырёх сигналов на соответствующих входах:



А в окне Debug (printf) Viewer наблюдаем:



Расшифровка:

- 4 – вход в обработчик прерывания *EXTI4*
- с – завершение обработчика *EXTI4*
- 6 – вход в обработчик прерывания *EXTI6*
- d – завершение обработчика *EXTI6*
- 3 – вход в обработчик прерывания *EXTI3*
- b – завершение обработчика *EXTI3*
- 2 – вход в обработчик прерывания *EXTI2*
- а – завершение обработчика *EXTI2*

Для более детального рассмотрения регистров настройки системы прерывания (NVIC) используйте инструменты: View >> System Viewer >> Core Peripherals >> Nested Vectored Interrupt Controller (NVIC); View >> System Viewer >> NVIC ; View >> System Viewer >> EXTI. Анализируйте порядок настройки прерываний, во избежание ошибок.

Описание регистров:

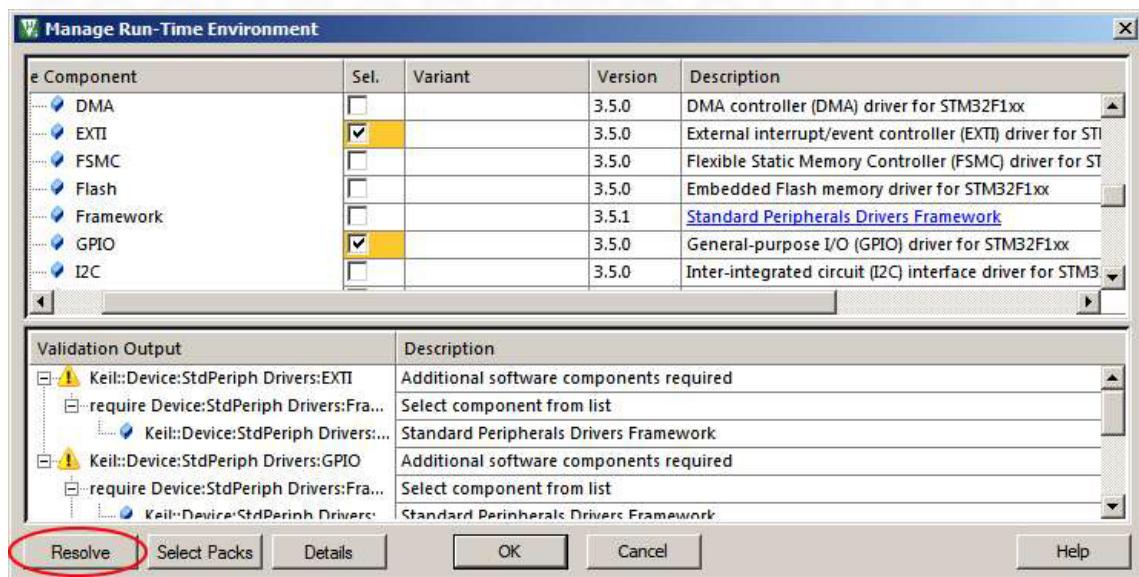
- регистр управления и состояния прерывания ICSR – 4.4.3 *Interrupt control and state register (SCB_ICSR)* PM0056 [14];
 - регистры NVIC: NVIC_ISERx, NVIC_ICERx, NVIC_ISPRx, NVIC_ICPRx, NVIC_IABRx, NVIC_IPRx – 4.3 *Nested vectored interrupt controller (NVIC)* PM0056 [14];
 - регистры EXTI, таблица векторов – раздел №10 RM0008 [13];
 - регистры системного таймера – 4.5 *SysTick timer (STK)* PM0056 [14].

5.3 Подключение к проекту стандартной библиотеки периферии (SPL)

Стандартная библиотека периферии (Standard Peripheral Library) [9] – это набор низкоуровневых драйверов. Каждый драйвер предоставляет пользователю набор функций для работы с соответствующим периферийным устройством. Таким образом, пользователь использует функции, а не обращается напрямую к регистрам. При этом уровень CMSIS оказывается скрытым от программиста. Библиотека [9] скачивается кнопкой *Get Software*, после регистрации.

Описание библиотеки внутри архива: `stm32f10x_stdperiph_lib_um.chm`. Что бы разобраться с библиотекой SPL используйте папку проектов с примерами: `STM32F10x_StdPeriph_Lib_V3.5.0 \ Project \ STM32F10x_StdPeriph_Examples`. Библиотеку можно подключать, используя шаблонный проект в папке `STM32F10x_StdPeriph_Lib_V3.5.0 \ Project \ STM32F10x_StdPeriph_Template \ MDK-ARM`, что на начальном этапе достаточно сложно. Поэтому, для примера, создадим в *Keil* проект с уже подключённой библиотекой SPL:

1) Создаём проект как обычно. При создании проекта выбираем тип МК `STM32F103C8`. На шаге настройки работы с библиотеками ARM MDK-Professional, выбираем пункты: `CMSIS >> CORE; Device >> Startup; Compiler >> I/O >> STDOUT` в выпадающем меню `ITM`. Развернув вкладку драйверов SPL (`Device >> StdPeriph Drivers`) выбираем `EXTI` (для работы с прерываниями) и `GPIO` (для работы с портами в/в), внизу в окне `Validation Output` появиться предупреждение, что эти модули взаимосвязаны с другими которые не подключены, для разрешения этого конфликта нажимаем `Resolve`:



и автоматически будут подключены дополнительные модули RCC и Framework, а предупреждения исчезнут. На этом настройки по подключению библиотеки SPL завершены, нажимаем OK.

2) На вкладке Debug выбираем Use Simulator. В поле Dialog DLL и Parameter вводим DARMSTM.DLL и -pSTM32F103C8 Скопируем ранее созданный файл (п. 5.1) GPIOA_But.ini в проект и укажем его на вкладке Debug в строке Initialization File.

3) Создадим и наполним файл main.c кодом:

```
#include "RTE_Components.h"          // Component selection
#include CMSIS_device_header          // Device header
#include <stdio.h>
int main()
{
    uint32_t cnt=0;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_ClocksTypeDef RCC_ClockFreq;
    RCC_GetClocksFreq(&RCC_ClockFreq);
    printf("SYSCLK=%dHz, HCLK=%dHz, PCLK1=%dHz, PCLK2=%dHz\n", RCC_ClockFreq.HCLK_Frequency,
           RCC_ClockFreq.SYSCLK_Frequency, RCC_ClockFreq.PCLK1_Frequency,
           RCC_ClockFreq.PCLK2_Frequency);
    switch(RCC_GetSYSCLKSource()) {
        case 0x08:
            printf("PLL used as system clock\n");
            break;
        case 0x04:
            printf("HSE used as system clock\n");
            break;
        case 0x00:
            printf("HSI used as system clock\n");
            break;
    }
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    //PA2,3,4,6 Input, Pull up
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    while(1){
        if(GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_2)==0)
            printf("%9i press PA2\n",cnt++);
        if(GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_3)==0)
            printf("%9i press PA3\n",cnt++);
        if(GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_4)==0)
            printf("%9i press PA4\n",cnt++);
        if(GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_6)==0)
            printf("%9i press PA6\n",cnt++);
    }
}
```

Откомпилируем и запустим отладку. Всё работает аналогично программе в п. 5.1.

Для дальнейшей реализации вашего варианта задания с использованием функций стандартной библиотеки периферии используйте примеры:

- STM32F10x_StdPeriph_Examples \ EXTI \ EXTI_Config;
- STM32F10x_StdPeriph_Examples \ SysTick \ TimeBase;
- STM32F10x_StdPeriph_Examples \ NVIC \ IRQ_Priority;
- STM32F10x_StdPeriph_Examples \ GPIO \ IOToggle.

5.4 Практические задания к разделу 5

Используя библиотеку CMSIS, написать и отладить работу программы на языке «си», в которую входит настройка внешних прерываний (EXTI) на четыре линии ПВВ A/B, согласно номера варианта таблицы 5.1.

Этап №1. Не меняя приоритеты и группировку приоритетов (оставляем по умолчанию после сброса) проверить в режиме отладки порядок обработки:

- одновременно установленных четырёх прерываний;
- поступления прерываний последовательно.

Отразить в выводах выявленный порядок обработки прерываний, прерывают ли прерывания друг друга и если да, то в каком порядке.

Этап №2. Не меняя группировку приоритетов, установить приоритеты согласно варианту таблицы 5.1, исследовать изменение порядка обработки прерываний.

Этап №3. Настроить приоритеты прерываний, используя группы и подгруппы, согласно варианту таблицы 5.1, исследовать изменение порядка обработки прерываний.

Выяснить влияние прерываний на сигнал линии PC13 управляемой системным таймером. Пояснить в выводах особенности обработки прерываний при различных настройках.

Используя библиотеку SPL, написать вторую аналогичную программу этапа 3 и отладить её работу.

Таблица 5.1

Варианты индивидуальных заданий к разделу 5

Номер варианта	Номера линий	Количество групп и подгрупп приоритетов		Номера приоритетов до настройки группировки приоритетов и после, соответственно порядку линий	
		Групп	Подгрупп	до	после №группы(№подгруппы)
1	PA0,PB1,PA2,PB3	8	2	99,114,128,113	2(0), 1(0), 1(1), 0(1)
2	PB0,PA1,PB2,PA4	4	4	103,118,132,117	2(2), 1(1), 1(0), 0(0)
3	PA0,PB1,PA2,PB10	2	8	107,122,136,121	0(1), 1(1), 1(0), 0(0)
4	PB0,PA1,PA3,PB4	8	2	111,126,140,125	4(0), 3(1), 3(0), 2(1)
5	PA0,PB1,PB3,PA5	4	4	3,18,32,17	3(3), 2(3), 2(1), 1(1)
6	PB0,PA1,PA4,PB5	2	8	7,22,36,21	1(3), 0(3), 0(2), 1(2)
7	PA0,PB2,PA3,PA5	8	2	11,26,40,25	6(0), 5(0), 5(1), 4(1)
8	PB0,PA2,PB3,PA10	4	4	15,30,44,29	0(3), 3(2), 3(1), 2(0)
9	PA0,PB2,PB4,PB5	2	8	19,34,48,33	0(5), 1(5), 1(4), 0(4)
10	PB0,PA2,PA4,PB10	8	2	23,38,52,37	0(0), 7(1), 7(0), 6(1)
11	PA0,PB2,PA5,PA10	4	4	27,42,56,41	2(3), 0(0), 0(3), 1(3)
12	PB0,PA3,PB4,PB5	2	8	31,46,60,45	1(7), 0(7), 0(6), 1(6)
13	PA0,PB3,PA4,PB10	8	2	35,50,64,49	4(1), 2(0), 2(1), 1(0)

Номер варианта	Номера линий	Количество групп и подгрупп приоритетов		Номера приоритетов до настройки группировки приоритетов и после, соответственно порядку линий	
		Групп	Подгрупп	до	после №группы(№подгруппы)
14	PB0,PA3,PA5,PA10	4	4	39,54,68,53	1(2), 2(1), 0(3), 1(3)
15	PA0,PB4,PB5,PB10	2	8	43,58,72,57	1(1), 0(1), 1(0), 0(0)
16	PB1,PA2,PB3,PA4	8	2	47,62,76,61	5(1), 0(0), 0(1), 5(0)
17	PA1,PB2,PA3,PA5	4	4	51,66,80,65	2(0), 3(2), 1(3), 2(2)
18	PB1,PA2,PB3,PA10	2	8	55,70,84,69	0(3), 1(3), 0(2), 1(2)
19	PA1,PB2,PB4,PB10	8	2	59,74,88,73	4(0), 2(0), 5(1), 4(1)
20	PB1,PA2,PB5,PA10	4	4	63,78,92,77	3(3), 0(2), 2(1), 3(0)
21	PA1,PA3,PA4,PA5	2	8	67,82,96,81	0(3), 1(3), 0(2), 1(2)
22	PB1,PB3,PB4,PB10	8	2	71,86,100,85	6(1), 7(0), 4(1), 6(0)
23	PA1,PA3,PB5,PA10	4	4	75,90,104,89	0(1), 1(0), 2(2), 0(2)
24	PB1,PA4,PA5,PB10	2	8	79,94,108,93	1(5), 0(5), 1(4), 0(4)
25	PB2,PB3,PB5,PA10	8	2	83,98,112,97	7(1), 0(0), 0(1), 7(0)
26	PA2,PB4,PA5,PB10	4	4	87,102,116,101	1(3), 2(0), 1(2), 0(1)
27	PA3,PA4,PB5,PA10	2	8	91,106,120,105	0(7), 1(7), 0(6), 1(6)
37	PA2,PA3,PA4,PA6	0	16	111,110,108,109	0(15), 0(10), 0(7), 0(0)

Отчёт должен содержать:

- 1) Номер варианта с заданием. Листинг программы этапа №3. Листинг файла инициализация симуляции кнопок (*.ini).
- 2) Анализ поведения системы прерываний для всех трёх этапов.
- 3) Листинг варианта программы, написанной с использованием функций стандартной библиотеки периферии STM32F10x standard peripheral library (SPL) [9].

5.5 Контрольные вопросы

- 1) Быть готовым на практике замаскировать или сбросить указанное прерывание и продемонстрировать результат в режиме отладки.
- 2) Назначение и функции контроллера приоритетных векторных прерываний (NVIC). Структурная схема, назначение и функции контроллера внешних событий/прерываний (EXTI).
- 3) Перечислить функции библиотеки CMSIS для работы с прерываниями, назначение функций, параметры.
- 4) Таблица векторов прерываний. Место определения в программе. Описание в документации. Инструменты работы с таблицей во время отладки.

- 5) Регистры: EXTI_IMR1, EXTI_EMR1. Назначение, примеры применения в программе. Просмотр состояний этих регистров в режиме отладки.
- 6) Регистры: EXTI_RTSR1, EXTI_FTSR1, EXTI_SWIER1, EXTI_PR1. Назначение, примеры применения в программе. Просмотр состояний этих регистров в режиме отладки.
- 7) Регистры: NVIC_ISERx, NVIC_ICERx. Назначение, примеры применения в программе. Просмотр состояний этих регистров в режиме отладки.
- 8) Регистры: NVIC_ISPRx, NVIC_ICPRx. Назначение, примеры применения в программе. Просмотр состояний этих регистров в режиме отладки.
- 9) Регистры: NVIC_IABR_x, NVIC_IPR_x. Назначение, примеры применения в программе. Просмотр состояний этих регистров в режиме отладки.
- 10) Исключите влияние внешних прерываний на работу системного таймера.
- 11) Назначение и структура библиотеки SPL.

6 Таймеры микроконтроллера

6.1 Порядок создания проекта с подключением библиотеки HAL LL

Скачайте библиотеку HAL STM32CubeF1 [10], файл en.stm32cubef1.zip. Создайте проект labTimerLL. При создании проекта выбираем тип МК STM32F103C8. На шаге настройки работы с библиотеками ARM MDK-Professional, выбираем пункты: CMSIS >> CORE и Compiler >> I/O >> STDOUT в выпадающем меню ITM, нажимаем OK. Пункт Device >> Startup пропускаем!

На вкладке Debug выбираем Use Simulator. В поле Dialog DLL и Parameter вводим DARMSTM.DLL и -pSTM32F103C8 , соответственно

Из архива в папку проекта labTimerLL копируем папки:

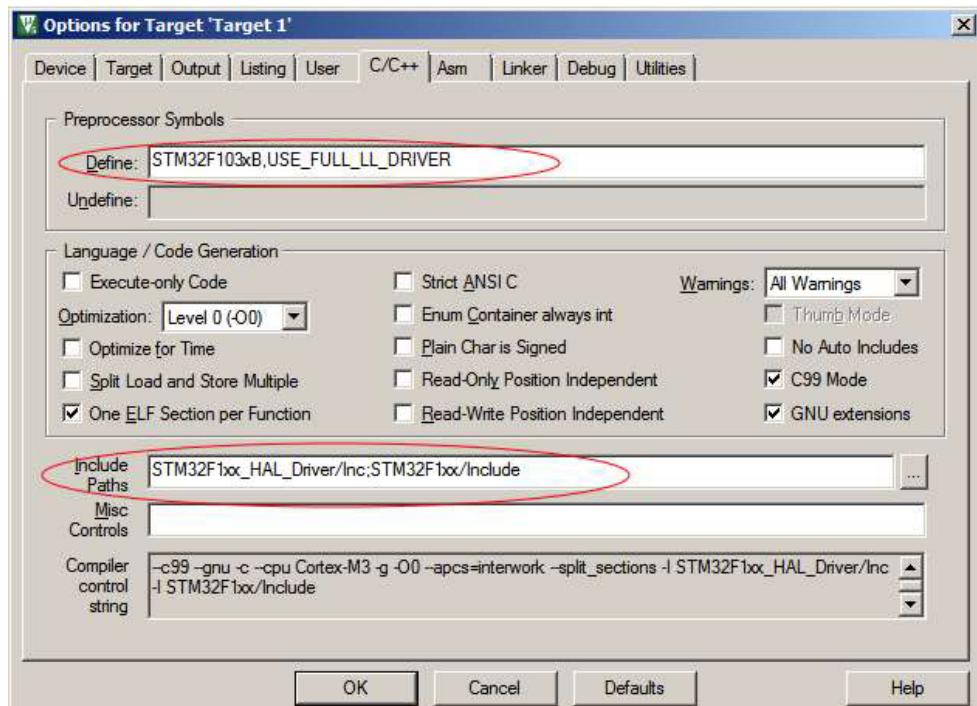
en.stm32cubef1.zip\STM32Cube_FW_F1_V1.8.0\Drivers\STM32F1xx_HAL_Driver;
en.stm32cubef1.zip\STM32Cube_FW_F1_V1.8.0\Drivers\CMSIS\Device\ST\STM32F1xx .

Подключим заголовочные файлы скопированных библиотек. В свойствах проекта на вкладке C/C++ в поле *Include Path* нажимаем кнопку с тремя точками:

 В появившемся диалоговом окне в поле Setup Compiler Include Paths: нажимаем кнопку New(Insert ) и прописываем пути до библиотек:

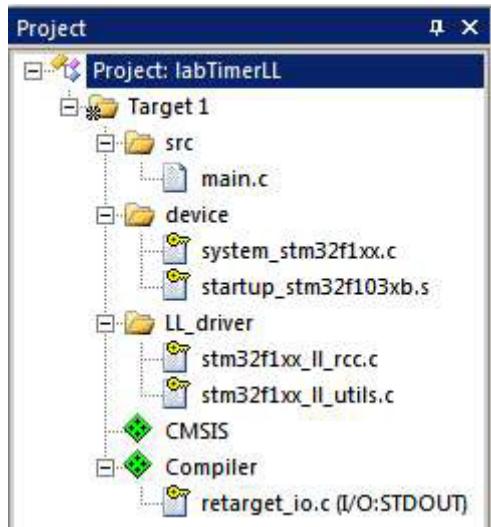
- для первой библиотеки STM32F1xx_HAL_Driver вставляем:
STM32F1xx_HAL_Driver/Inc;
- для второй библиотеки CMSIS STM32F1xx, содержащей код инициализации при старте МК, вставляем: STM32F1xx/Include.

Теперь в регионе Preprocessor Symbols поле Define укажем директивы: STM32F103xB, USE_FULL_LL_DRIVER. Закрываем вкладку.

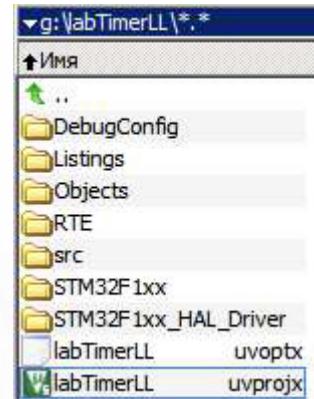


Добавим в проект требуемые файлы с кодом инициализации при старте МК. Нажимаем в окне Project правой кнопкой мыши на папку Target и в выпавшем меню выбираем Add Group... Появилась папка New Group, на неё также нажимаем правой кнопкой и в меню выбираем Add Existing Files to Group 'New Group'... (Add Existing Files to Group 'New Group'...). В открывшемся диалоговом окне выбираем файл: ../STM32F1xx/Source/Templates/**system_stm32f1xx.c** и добавляем кнопкой Add. Далее, не закрывая окна, переключаем отображение на тип файлов Asm Source file (*.s*; *.src; *.a*) и добавляем файл: ../STM32F1xx/ Source/ Templates/arm/ **startup_stm32f103xb.s**. Переименуйте название папки ('New Group') на ('device'), используя инструмент Manage Projects Items... Подобным образом создайте папку 'LL_driver', и добавьте туда файлы: ../STM32F1xx_HAL_Driver/Src/**stm32f1xx_ll_rcc.c** и ../STM32F1xx_HAL_Driver/Src/**stm32f1xx_ll_utils.c**. Также создайте папку например 'src' и в ней файл main.c.

Древо проекта в Keil:



Вид папок проекта на диске:

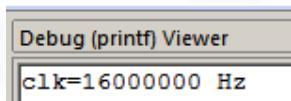


Добавьте следующий код в файл main.c:

```
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
#include "stm32f1xx_ll_exti.h"
#include "stm32f1xx_ll_tim.h"
#include "stm32f1xx_ll_cortex.h"
#include <stdio.h>
int main(void){
    printf("clk=%d Hz\n", SystemCoreClock);
    while (1){}
}
```

Проведите компиляцию и запустите отладку. Если ошибок нет, то в окне *Debug (printf) Viewer*

Viewer получим:



6.2 Генерация меандра с помощью таймера TIM2

Данный пример иллюстрирует настройку с помощью функций HAL LL API:

- системы тактирования ядра на частоту 72 МГц;
- настройку линий портов ввода/вывода PC13 и PB13 на вывод;
- настройку внешнего прерывания на вход с линии PB12;
- настройку таймера №2 (TIM2) в режим счёта вверх и прерывания по обновлению таймера;
- настройку системного таймера.

Добавьте следующий код в файл main.c:

```
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_system.h"
```

```

#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
#include "stm32f1xx_ll_exti.h"
#include "stm32f1xx_ll_tim.h"
#include "stm32f1xx_ll_cortex.h"
#include <stdio.h>
uint16_t TIM2cnt=0; //переменная текущего состояния счётчика TIM2 для лог. анализатора
static uint32_t InitialAutoreload = 0; // начальное состояние TIM2_ARR для частоты 20Гц
static uint8_t AutoreloadMult = 1; //козф. деления частоты
int main(void)
{
    // Настройка системной частоты = 72 MHz
    printf("clk=%d Hz\n", SystemCoreClock);
    LL_FLASH_SetLatency(LL_FLASH_LATENCY_2);
    LL_RCC_HSE_Enable();
    while(LL_RCC_HSE_IsReady() != 1){};
    LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSE_DIV_1, LL_RCC_PLL_MUL_9);
    LL_RCC_PLL_Enable();
    while(LL_RCC_PLL_IsReady() != 1){};
    LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
    LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
    while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL){};
    LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_2);
    LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_1);
    SystemCoreClockUpdate();
    printf("clk=%d Hz\n", SystemCoreClock);
    // Настройка линии PC13 и PB13 на вывод
    LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_GPIOC | LL_APB2_GRP1_PERIPH_GPIOB);
    LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_13, LL_GPIO_MODE_OUTPUT);
    LL_GPIO_SetPinMode(GPIOC, LL_GPIO_PIN_13, LL_GPIO_MODE_OUTPUT);
    // Настройка прерывания на линию PB12 при изменении сигнала на входе с 1 в 0
    LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_12, LL_GPIO_MODE_INPUT);
    LL_GPIO_SetPinPull(GPIOB, LL_GPIO_PIN_12, LL_GPIO_PULL_DOWN);
    LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_AFIO);
    LL_GPIO_AF_SetEXTISource(LL_GPIO_AF_EXTI_PORTB, LL_GPIO_AF_EXTI_LINE12);
    LL EXTI_EnableIT_0_31(LL_EXTI_LINE_12);
    LL EXTI_EnableFallingTrig_0_31(LL_EXTI_LINE_12);
    NVIC_EnableIRQ(EXTI15_10_IRQn);
    NVIC_SetPriority(EXTI15_10_IRQn, 0x03);
    // Настройка таймера №2 (TIM2)
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_TIM2); //включили в работу
    LL_TIM_SetCounterMode(TIM2, LL_TIM_COUNTERMODE_UP); //счёт вверх
    // Значение предделителя (TIM2_PSC) вычисляем на эквивалент 10КГц для тактира. счётчика
    // TIM2CLK = PCLK1x2 = 72 МГц
    // TIM2_PSC (Prescaler) = (TIM2CLK /10 KHz) - 1 = 7 199 = 0x1C1F
    LL_TIM_SetPrescaler(TIM2, __LL_TIM_CALC_PSC(SystemCoreClock, 10000));
    printf("TIM2_PSC=%d\n", __LL_TIM_CALC_PSC(SystemCoreClock, 10000));
    // Значение регистра авто-перезагрузки TIM2_ARR вычисляем на частоту 20 Гц
    // TIM2_ARR = 72000000 / (7199+1) / 20 - 1 = 499 = 0x1F3
    InitialAutoreload = __LL_TIM_CALC_ARR(SystemCoreClock, LL_TIM_GetPrescaler(TIM2), 20);
    LL_TIM_SetAutoReload(TIM2, InitialAutoreload);
    printf("TIM2_ARR=%d\n", InitialAutoreload);
    LL_TIM_EnableIT_UPDATE(TIM2); //разрешаем таймеру №2 прерывание по событию обновления
    NVIC_SetPriority(TIM2_IRQn, 0);
    NVIC_EnableIRQ(TIM2_IRQn); //разрешаем прерывание таймеру №2 в NVIC
    LL_TIM_EnableCounter(TIM2); //включаем в работу таймер №2
    LL_TIM_GenerateEvent_UPDATE(TIM2); //программно вызываем событие обновления таймера №2
    //Настройка системного таймера
    LL_Init1msTick(SystemCoreClock); //настраиваем на прерывание каждую миллисекунду
    LL_SYSTICK_EnableIT(); //разрешаем прерывание системному таймеру
    while (1){ TIM2cnt=TIM2->CNT; } //присваиваем текущее значение счётчика TIM2->CNT
}
void SysTick_Handler(void){//обработчик прерывания системного таймера
    static uint32_t cnt1ms=0;
    cnt1ms++;
    if(cnt1ms%500 == 0)//переключаем каждые полсекунды
        LL_GPIO_TogglePin(GPIOB, LL_GPIO_PIN_13); //частота меандра 1Гц
}
void TIM2_IRQHandler(void){//обработчик прерывания таймера №2
    if(LL_TIM_IsActiveFlag_UPDATE(TIM2) == 1){//контроль, прерывание по обновлению счётчика?
        LL_TIM_ClearFlag_UPDATE(TIM2); //очищаем флаг прерывания
        LL_GPIO_TogglePin(GPIOC, LL_GPIO_PIN_13);
    }
}
void EXTI15_10_IRQHandler(void){
    if(LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_12) != RESET)
    {//контроль, прерывания на линии №12?
        LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_12); //очищаем флаг прерывания
        // Меняем значение регистра авто-перезагрузки TIM2_ARR по закону:
    }
}

```

```

    // Update_event = TIM2CLK /((PSC + 1)*(ARR + 1)*(AutoreloadMult + 1))
    AutoreloadMult = AutoreloadMult % 5; // получаемый ряд частот Гц: 20; 10; 6.(6); 5; 4
    uint32_t coefARR=(InitialAutoreload + 1) * (AutoreloadMult +1);
    LL_TIM_SetAutoReload(TIM2, coefARR-1);
    printf("TIM2_ARR=%d\n",coefARR-1);
    printf("freq rect
PC13:%f\n",SystemCoreClock/(float)(2*(LL_TIM_GetPrescaler(TIM2)+1)*coefARR));
    LL_TIM_GenerateEvent_UPDATE(TIM2); //программно вызываем событие обновления таймера №2
    AutoreloadMult++;
}
}

```

Создадим файл(скрипт) создания кнопки `GPIO_But.ini` и наполним следующим кодом:

```

//KILL BUTTON 1 удалить старую кнопку для тех, кто будет экспериментировать
OSC = 8000000; //Установили частоту внешнего генератор Xtal (Hz)
PORTB &= ~(1<<12);
SIGNAL void toggle_PB12() {
    PORTB |= (1<<12); //установить единицу
    swatch (0.010); /* ждать 10 мс */
    PORTB &= ~(1<<12); //установить ноль
}
DEFINE BUTTON "Key_PB12", "toggle_PB12()"

```

Подключим его на вкладке `Debug`: вставим название файла (`GPIO_But.ini`) в строке `Initialization File` или через кнопку многоточие найдём на диске.

Скомпилируем и запустим в режиме отладки программу. Откроем окно вывода отладчика `View >> Serial Windows >> Debug (printf) Viewer`. Окно `Toolbox` с одной кнопкой `Key_PB12`: `View >> ToolBox Window`. Окно `Logic Analyzer`: `View >> Analysis Windows >> Logic Analyzer`.

В окне `Logic Analyzer` настроим отображение нужных нам входов/выходов:

Current Logic Analyzer Signals	Display Type	And Mask	Shift Right	Max:	Описание
PORTC	Bit	0x00002000	13	-	Выход меандра регулируемой частоты входом PB12
TIM2cnt	Analog	-	-	0xFFFF	Текущее значение счётчика таймера TIM2
PORTB	Bit	0x00001000	12	-	Вход сигнала от кнопки <code>Key_PB12</code>
PORTB	Bit	0x00002000	13	-	Выход меандра, регулируемого системным таймером

Продолжим выполнение кода в режиме отладки (`F5 Run`). `PC13` переключается с частотой 20Гц, поэтому частота меандра 10 Гц. При каждом нажатии кнопки `Key_PB12`, изменяется частота переключения `PC13` по следующему закону: 20 Гц; 10Гц; 6.6(6)Гц; 5Гц; 4Гц (20/1; /2; /3; /4; /5) и далее снова повторяется. Частота переключения выхода `PB13` определяется системным таймером и равна 2 Гц. Общий вид окон `Logic Analyzer` и `Debug (printf) Viewer` показаны на рисунке 6.1.

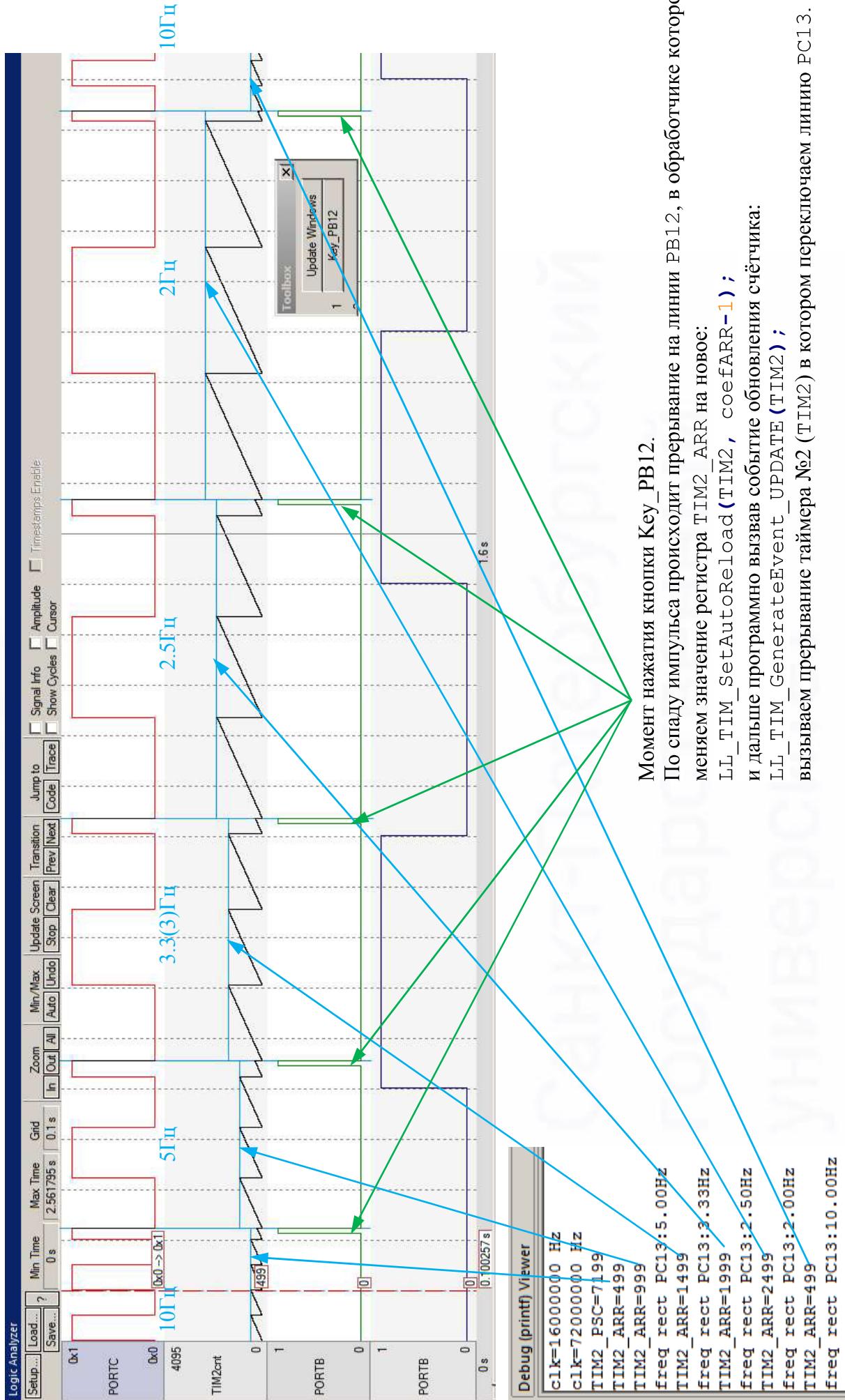


Рис. 6.1. – Эпюры сигналов на линиях РС13, РВ13, РВ12 и изменения счётика таймера ТИМ2

В Logic Analyzer можно отображать переменные только с глобальной областью видимости при отключенной оптимизации компилятора. Поэтому переменная TIM2cnt выступает как вспомогательная для отображения текущего состояния счётчика. При достижении счётчиком значения регистра авто-перезагрузки TIM2_ARR происходит сброс счётчика в ноль, генерация события обновления и прерывания по таймеру №2.

Настройка частоты тактирования таймера осуществлена по схеме показанной на рисунку 6.2, где SYSCLK = 72МГц, AHB Pre = 1, APB1 Pre = 2, PCLK1 = 36МГц, TIM2CLK = PCLK1 x 2 = 72МГц.

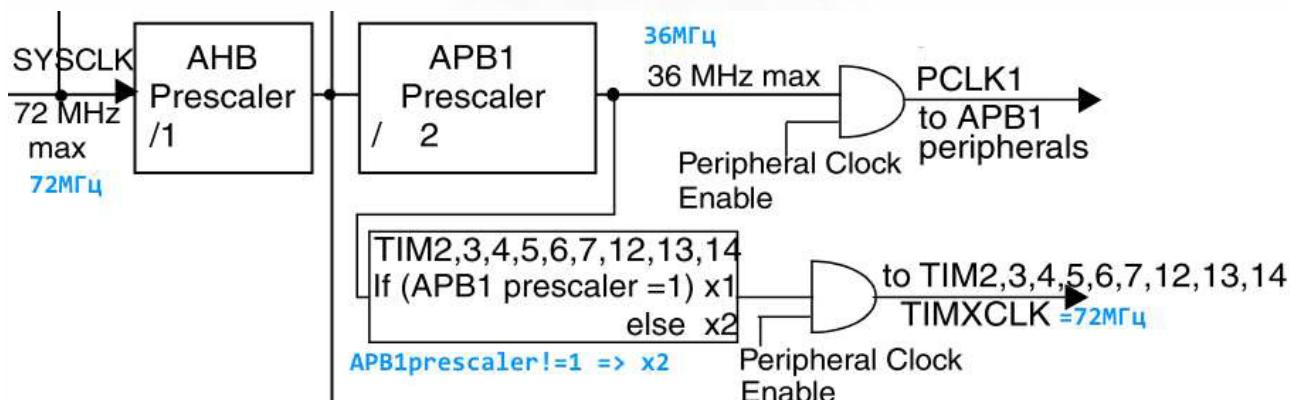


Рис. 6.2. – Схема настройки частоты TIM2CLK

Настройка частот и регистров в таймере №2 показана на рисунке 6.3 (RM0008 [13], Figure 100. General-purpose timer block diagram, стр.367), где CK_PSC = TIM2CLK = 72МГц, PSC prescaler (TIM2_PSC) = 7199, CK_CNT = CK_PSC / (7199 + 1) = 10000Гц (частота тактирования счётчика CNT), Auto-reload register (TIM2_ARR) = 499, отсюда частота прерываний счётчика по обновлению равна CK_CNT / (499 + 1) = 20Гц.

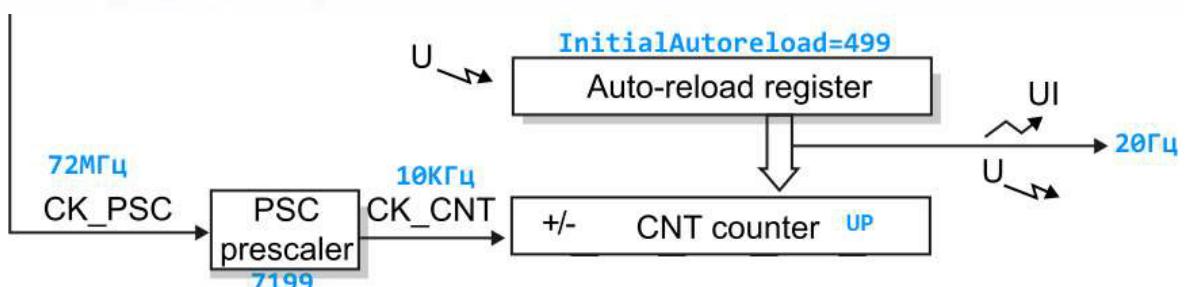


Рис. 6.3. – Схема настройки частот и регистров в таймере

Контроль значений регистров TIM2_PSC, TIM2_ARR, TIM2_CNT осуществляется в окне View >> System Viewer >> TIM >> TIM2.

6.3 Пример решения задачи по расчёту значений управляющих регистров таймера и подсистемы тактирования

Задача:

Рассчитать коэффициенты настройки подсистемы тактирования МК STM32F103C8 и таймера TIMx для настройки прерывания по обновлению таймера один раз в 1.5 минуты.

Решение:

Воспользуемся формулой из AN4013 [15] раздел 2.2 Time base generator на стр. 11:

$$\text{Update_event} = \text{TIM_CLK} / ((\text{PSC} + 1) * (\text{ARR} + 1) * (\text{RCR} + 1)),$$

здесь Update_event (UE) – частота события обновления таймера (f_{UE}); TIM_CLK – частота тактирования время задающего блока таймера ($\text{CK_PSC}=\text{TIM_CLK}$); PSC – значение предделителя частоты тактирования счётчика таймера (TIMx_PSC); ARR – значение регистра авто-перезагрузки (TIMx_ARR); RCR – значение регистра авто-перезагрузки счётчика циклов обновления таймера (TIMx_RCR). Поскольку TIMx_RCR опционально присутствует только в таймере №1, а в остальных отсутствует, то будем считать его значение равным нулю. И таким образом упростим формулу до вида:

$$f_{UE} = \frac{\text{TIM_CLK}}{(\text{PSC} + 1) * (\text{ARR} + 1)} \quad (6.1)$$

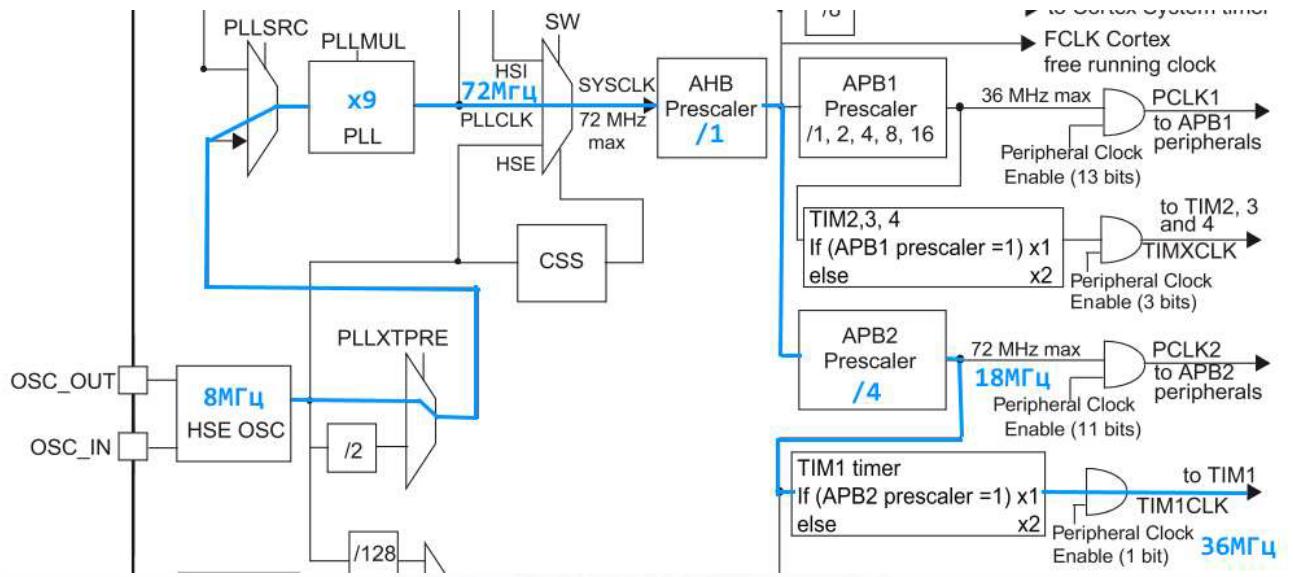
Исходя из обозначений схемы на рисунке 6.3 получаем формулы ($\text{CK_PSC}=\text{TIM_CLK}$):

$$\text{CK_CNT} = \frac{\text{TIM_CLK}}{\text{PSC} + 1}, \quad f_{UE} = \frac{\text{CK_CNT}}{\text{ARR} + 1} \quad (6.2)$$

Поскольку разрядность регистров TIMx_PSC и TIMx_ARR составляет 16 бит, то максимальное число представлено значением $2^{16}-1=65535$. Исходя из этого ограничения, посчитаем значение TIMx_CLK которое нельзя превышать, иначе возможностей делителей не хватит, поскольку $f_{UE} = 1 / (1,5*60) = \frac{1}{90}$ Гц, из формулы (6.1):

$$\text{TIMx_CLK} = \frac{1}{90} \cdot (65535 + 1) \cdot (65535 + 1) = 47721858,8(4) \text{ Гц.}$$

Т.е. нам необходимо настроить частоту TIMx_CLK так чтобы она не превышала 47 721 858,8(4) Гц. Для наглядности воспользуемся схемой DS5319 [12] Figure 2. Clock tree на стр. 12. Поскольку настройка параметров на шинах APB1 и APB2 одинакова, будем обозначать их как APBx и частоту на них PCLKx. Пример настройки частоты TIM1CLK:



Самое простое, что приходит в голову, разделить максимально возможную частоту тактирования на два с помощью предделителя APBx_Pre = 4 и соответственно получаем:

- 1) Настройка системной частоты. Источник SYSCLK = PLLCLK = 8МГц(HSE) × 9(PLLMUL) = 72МГц. HCLK = SYSCLK / 1(APB1_Pre=1).
- 2) Настройка рабочей частоты TIMxCLK. PCLKx = HCLK/4 = 18МГц, TIMxCLK = PCLKx × 2 = 36МГц < 47721858,8(4) Гц.
- 3) Далее, по схеме рисунка 6.3, частота тактирования предделителя CK_PSC = TIMxCLK = 36МГц. Рассчитаем максимально возможное значение частоты тактирования счётчика CK_CNT исходя из зависимости (6.2): $f_{UE} = \frac{CK_CNT}{(ARR + 1)}$. Откуда

$$CK_CNT = \frac{1}{90} \cdot (65535 + 1) = 728,1(7) \text{ Гц}, \text{ возьмём меньшую частоту } CK_CNT = 600 \text{ Гц т.к. она}$$

без остатка делит 36МГц. Теперь посчитаем коэффициенты

$$ARR = \frac{CK_CNT}{f_{UE}} - 1 = \frac{600}{\frac{1}{90}} - 1 = 53999, \text{ т.е. } TIMx_ARR = 53999. \text{ Далее посчитаем значение}$$

предделителя PSC, выразив из формулы (6.2): $CK_CNT = \frac{TIMx_CLK}{PSC + 1}$,

$$PSC = \frac{TIMx_CLK}{CK_CNT} - 1 = \frac{36000000}{600} - 1 = 59999, \text{ т.е. } TIMx_PSC = 59999.$$

- 4) Проверим расчёты:

$$f_{UE} = \frac{TIMx_CLK}{(PSC + 1) \cdot (ARR + 1)} = \frac{36000000}{(59999 + 1) \cdot (53999 + 1)} = 0.01(1) = \frac{1}{90} \text{ Гц},$$

всё верно, получили частоту прерывания таймера по обновлению один раз в 90 секунд.

Ответ: Тактирование от PLL, источник HSE/1; PLLMUL = 9; SYSCLK = PLLCLK = 72МГц; AHB_Pre = 1; APBx_Pre = 4; TIMx_PSC = 59999 ; TIMx_ARR = 53999.

Примечание.

Вариаций решения данной задачи множество, в примере приведена только одна из многих. Можно было для деления частоты использовать AHB_Pre =2 или сразу настраивать частоту SYSCLK = PLLCLK равной 44МГц или ниже. Студент вправе выбрать любой подход при решении задачи, главное требование, чтобы решение было практически реализуемо, т.к. числовой ряд делителей и умножителей ограничен, и эти ограничения надо знать, что подразумевает умение пользоваться документацией.

6.4 Практические задания к разделу 6

В соответствии с вариантом (табл. 6.1) нужно написать на языке «си» программу генерации прямоугольных импульсов на двух линиях в/в одновременно с заданной частотой.

Один из таймеров настроить на счёт вверх (LL_TIM_COUNTERMODE_UP). Второй таймер на счёт вниз (LL_TIM_COUNTERMODE_DOWN). Сравнить эпюры полученных TIMx->CNT сигналов.

Таблица 6.1

Варианты индивидуальных заданий к разделу 6

Номер варианта	Частота SYSCLK, МГц	Генератор №1			Генератор №2		
		Таймер	Линия	Частота, Гц	Таймер	Линия	Частота, Гц
1	32	TIM2	PA9	3952	TIM3	PB8	208
2	36	TIM2	PA4	4180	TIM4	PA15	220
3	40	TIM3	PA6	4427	TIM2	PA10	233
4	44	TIM1	PB5	4693	TIM4	PA11	247
5	48	TIM3	PB14	4978	TIM4	PB12	262
6	52	TIM2	PB7	5263	TIM1	PB4	277
7	56	TIM3	PA7	5586	TIM2	PB6	294
8	60	TIM4	PB0	5909	TIM2	PB10	311
9	64	TIM1	PA5	6270	TIM4	PB13	330
10	32	TIM3	PA2	6631	TIM4	PA1	349
11	36	TIM1	PA0	7030	TIM3	PB1	370
12	40	TIM2	PB9	7448	TIM1	PA8	392
13	44	TIM2	PA12	7885	TIM4	PB15	415
14	48	TIM1	PB11	8360	TIM4	PA3	440
15	52	TIM4	PA3	8854	TIM1	PB11	466
16	56	TIM4	PB15	9386	TIM3	PA12	494

Номер варианта	Частота SYSCLK, МГц	Генератор №1			Генератор №2		
		Таймер	Линия	Частота, Гц	Таймер	Линия	Частота, Гц
17	60	TIM1	PA8	9937	TIM2	PB9	523
18	64	TIM3	PB1	10526	TIM1	PA0	554
19	32	TIM2	PA1	11153	TIM4	PA2	587
20	36	TIM4	PB13	11818	TIM3	PA5	622
21	40	TIM4	PB10	12521	TIM3	PB0	659
22	44	TIM2	PB6	13262	TIM3	PA7	698
23	48	TIM1	PB4	14060	TIM3	PB7	740
24	52	TIM3	PB12	14896	TIM4	PB14	784
25	56	TIM4	PA11	15789	TIM1	PB5	831
26	60	TIM1	PA10	16720	TIM2	PA6	880
27	64	TIM4	PA15	17708	TIM2	PA4	932
28	32	TIM3	PB8	18772	TIM1	PA9	988

Отчёт должен содержать:

- 1) Номер варианта с заданием. Листинг программы.
- 2) Снимки эпюор логического анализатора для заданных линиях ввода/вывода с измерениями частоты и изменения счётчиков таймеров, подобно рисунку 6.1.
- 3) Решить учебную задачу:

Рассчитать коэффициенты настройки подсистемы тактирования МК STM32F103C8 и таймера TIMx для настройки прерывания по обновлению таймера один раз в 3+2×(номер варианта) минут(ы).

Т.е. рассчитать значения частот SYSCLK, PCLKx (указываем источник тактирования PLL, HSE/1 (/2), HSI/1 (/2), и значения коэффи. PLLMUL, AHB_Pre, APBx_Pre), частоту TIMxCLK, рассчитать значения регистров TIMx_PSC и TIMx_ARR.

6.5 Контрольные вопросы.

- 1) Понятие времени в цифровых системах. Виды таймеров, назначение. Обзор таймеров серии STM32. Характеристики таймеров. Акронимы таймеров, описание в документации.
- 2) Устройство базовых таймеров/счётчиков. Блок-схема, функциональное назначение блоков, сигналов. Основные управляющие регистры.
- 3) Регистры базовых таймеров. Назначение управляющих полей и бит.

- 4) Устройство таймеров/счётчиков общего назначения. Блок-схема, функциональное назначение блоков, сигналов.
- 5) Устройство таймеров/счётчиков с расширенным функционалом. Блок-схема, функциональное назначение блоков, сигналов.

Санкт-Петербургский
государственный
университет
аэрокосмического
приборостроения

Литература

- 1 Осциллографы. Основные принципы измерений. Учебное пособие. URL:
<https://download.tek.com/document/03U-8605-5%20Scopes%20Manual.pdf>
- 2 Осциллографы цифровые UDS1000. Руководство по эксплуатации. URL:
http://micromir-nn.ru/Oscil/uniontest/UnionTest_UDS1000.pdf
- 3 STM32-base. Blue Pill STM32F103C8T6. URL: <https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill.html>
- 4 STM32-base. Blue Pill STM32F103C8T6 schematic. URL: https://stm32-base.org/assets/pdf/boards/original-schematic-STM32F103C8T6-Blue_Pill.pdf
- 5 STM32-base. ST-LINK V2 Debugger. URL: <https://stm32-base.org/boards/Debugger-STM32F101C8T6-STLINKV2.html>
- 6 MDK-ARM. URL: <https://lprogs.com/keil-mdk/>
- 7 STMicroelectronics STM32F103C8 Device Family Pack. URL:
https://keilpack.azureedge.net/pack/Keil.STM32F1xx_DFP.2.4.0.pack
- 8 Common Microcontroller Software Interface Standard (CMSIS). URL: https://arm-software.github.io/CMSIS_5/Build/html/index.html
- 9 STM32F10x standard peripheral library URL: <https://www.st.com/en/embedded-software/stsw-stm32054.html>
- 10 STM32Cube MCU Package for STM32F1 series. URL:
<https://www.st.com/en/embedded-software/stm32cubef1.html>
- 11 UM1850 Description of STM32F1 HAL and low-layer drivers. URL:
https://www.st.com/resource/en/user_manual/um1850-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf
- 12 DS5319. Datasheet STM32F103x8, STM32F103xB. URL:
<https://www.st.com/resource/en/datasheet/stm32f103c8.pdf>
- 13 RM0008. Reference manual STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM®-based 32-bit MCUs. URL:
https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
- 14 PM0056. STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 programming manual.
URL: https://www.st.com/resource/en/programming_manual/pm0056-stm32f10xxx20xxx21xxx1xxxx-cortexm3-programming-manual-stmicroelectronics.pdf

15 AN4013. STM32 cross-series timer overview. URL:

https://www.st.com/resource/en/application_note/an4013-stm32-crossseries-timer-overview-stmicroelectronics.pdf

Санкт-Петербургский
государственный
университет
аэрокосмического
приборостроения