УНИВЕРСИТЕТ ИТМО

А.О. Ключев, В.Ю. Пинкевич, А.Е. Платунов, В.А. Ключев

СТЕНД-КОНСТРУКТОР SDK-1.1M. ОРГАНИЗАЦИЯ И ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

МАТЕРИАЛ ДОПОЛНЯЕТСЯ

Санкт-Петербург
Дата редактирования: 21.09.2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.О. Ключев, В.Ю. Пинкевич, А.Е. Платунов, В.А. Ключев

СТЕНД-КОНСТРУКТОР SDK-1.1M. ОРГАНИЗАЦИЯ И ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

УЧЕБНОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлениям подготовки 09.04.01 «Информатика и вычислительная техника», 09.04.04 «Программная инженерия» в качестве учебного пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2022

А.О. Ключев, В.Ю. Пинкевич, А.Е. Платунов, В.А. Ключев. Стенд-конструктор SDK-1.1M. Организация и программирование микроконтроллеров. – СПб: Университет ИТМО., 2022. – 79 с.

Рецензент(ы):
Быковский С.В., кандидат технических наук, доцент (квалификационная категория «ординарный доцент») факультета программной инженерии и компьютерной техники, Университет ИТМО.

Учебное пособие предназначено для демонстрации базовых аппаратных и программных механизмов современных встраиваемых вычислительных систем с помощью стенда-конструктора SDK-1.1M. Рассматриваются вопросы устройства микроконтроллеров и их составных элементов на примере современных устройств семейства STM32. Обсуждается организация типовых интерфейсов ввода-вывода и внешних устройств (датчиков, микросхем памяти, расширителей портов ввода-вывода и др.), приемы разработки драйверов и прикладных программ.

Для закрепления теоретического материала в учебном пособии предусмотрен лабораторный практикум, в ходе выполнения которого студенты на практике используют и реализуют рассмотренные механизмы, знакомятся с разными проблемами и способами решения типовых задач создания встраиваемых систем, осваивают современные инструментальные средства программирования и отладки микроконтроллеров.

Университет ИТМО – национальный исследовательский университет, ведущий вуз России в области информационных, фотонных и биохимических технологий. Альма-матер победителей международных соревнований по программированию – ICPC (единственный в мире семикратный чемпион), Google Code Jam, Facebook Hacker Cup, Яндекс.Алгоритм, Russian Code Cup, Topcoder Open и др. Приоритетные направления: IT, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication. Входит в ТОП-100 по направлению «Автоматизация и управление» Шанхайского предметного рейтинга (ARWU) и занимает 74 место в мире в британском предметном рейтинге QS по компьютерным наукам (Computer Science and Information Systems). С 2013 по 2020 гг. – лидер Проекта 5–100.

Оглавление

ВВЕДЕНИЕ **4**

1. ОСНОВНЫЕ МЕХАНИЗМЫ ВСТРАИВАЕМЫХ СИСТЕМ НА ПРИМЕРЕ СТЕНДА SDK-1.1M **6**

1.1. Сигналы сброса и синхронизации 6
1.2. Интерфейс ввода-вывода общего назначения (GPIO) 6
1.3. Прерывания 10
1.4. Последовательный интерфейс UART 12
1.5. Программируемые таймеры 19
1.6. Интерфейс I^2C 27
1.7. Матричная клавиатура 29
1.8. Дисплейный модуль 33

2. ОТКРЫТОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВСТРАИВАЕМЫХ СИСТЕМ **37**

2.1. Операционная система реального времени FreeRTOS 37
2.2. Сетевой стек LwIP 38
2.3. Программная экосистема микропроцессоров линейки STM32MP15x 41

3. ОРГАНИЗАЦИЯ СТЕНДА SDK-1.1M **47**

3.1. Общий вид стенда SDK-1.1M 47
3.2. Комплектация SDK-1.1M 49
3.3. Вычислители процессорных модулей SDK-1.1M 50
3.4. Интерфейсы и периферийные устройства несущей платы SDK-1.1M 55

4. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА СТЕНДА SDK-1.1M **56**

4.1. Подключение стенда к компьютеру 56
4.2. Установка драйверов встроенного программатора-отладчика 56
4.3. Инструментальные средства разработки для микроконтроллеров STM32 57
4.4. Пример создания и настройки проекта в STM32CubeIDE 58
4.5. Использование отладочного вывода semihosting 62
4.6. Система виртуальных лабораторий ITMO.cLAB 64
4.7. Пример работы с SDK-1.1M через ITMO.cLAB 65
4.8. Электрическая схема и примеры программирования стенда SDK-1.1M 67
4.9. Частые вопросы 67

5. ЛАБОРАТОРНЫЙ ПРАКТИКУМ **68**

Требования к выполнению лабораторных работ 68
5.1. Лабораторная работа 1. Интерфейсы ввода-вывода общего назначения (GPIO) 69
5.2. Лабораторная работа 2. Последовательный интерфейс UART 71
5.3. Лабораторная работа 3. Таймеры 74
5.4. Лабораторная работа 4. Интерфейс I^2C и матричная клавиатура 78
5.5. Лабораторная работа 5. OLED-дисплей 79
Дополнительные задания 82

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА **83**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **84**

Введение

Стенды-конструкторы семейства SDK-1.1M (рис. 1) представляют собой многофункциональные устройства, построенные на базе современных микроконтроллеров, микропроцессоров и других вычислителей. Стенды предназначены для изучения архитектуры и методов проектирования:

- киберфизических систем и интернета вещей;
- систем на базе микропроцессоров и микроконтроллеров;
- встраиваемых контроллеров и систем сбора данных;
- периферийных блоков вычислительных систем;
- подсистем ввода-вывода встраиваемых систем.

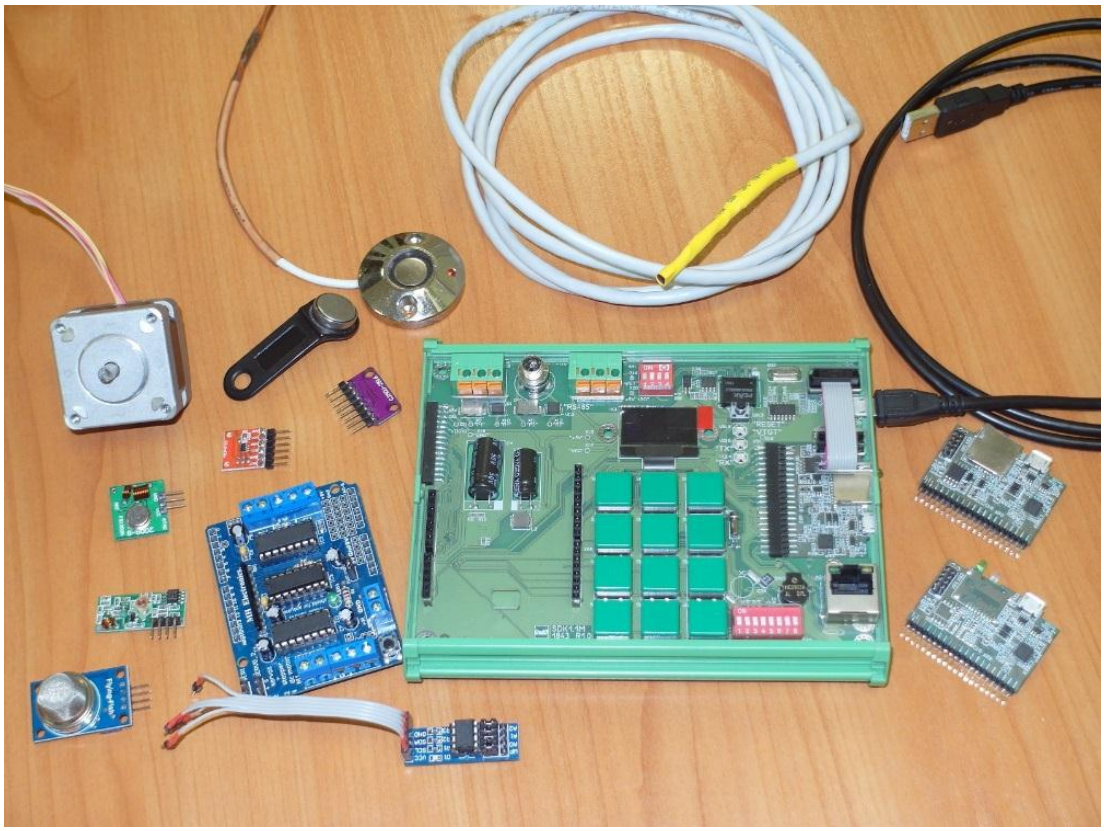


Рисунок 1 – Стенд-конструктор SDK-1.1M

Стенды семейства SDK-1.1M – новое поколение в серии учебных стендов SDK, которые на протяжении более чем 20 лет успешно применяются в учебном процессе более чем в 50 вузах и средних учебных заведениях России и стран ближнего зарубежья.

Стенды серии SDK созданы специалистами ООО «ЛМТ» [1] в сотрудничестве с преподавателями факультета программной инженерии и компьютерной техники Университета ИТМО [2, 3].

ВУЗы, колледжи могут использовать стенд SDK-1.1M для проведения учебных и исследовательских работ. Преподаватели могут как разрабатывать свои задания для лабораторных работ, так и использовать готовые.

Стенд SDK-1.1M является удобной платформой для прототипирования в сфере автоматизации производства, инфраструктурной автоматики и научных исследований. Особый интерес представляет возможность широкого использования SDK-1.1M в распределенных системах сбора данных и управления класса «Интернет вещей», «умный

дом», «умное здание». Благодаря своей модульной архитектуре и поддержке различных коммуникационных технологий и интерфейсов SDK-1.1M обеспечивает подключение GSM-модемов, модулей GPS/ГЛОНАСС, Wi-Fi, ZigBee, элементов программируемой логики и др.

Авторы рассматривают стенд-конструктор SDK-1.1M в качестве платформы для разнообразных студенческих проектов, что подтверждено успешным опытом применения стенда магистрантами и аспирантами при выполнении выпускных квалификационных работ.

1. Основные механизмы встраиваемых систем на примере стенда SDK-1.1M

В качестве примера в данном разделе будут рассматриваться стенды SDK-1.1M на базе микроконтроллеров линейки STM32F407/STM32F427, имеющие процессорное ядро с архитектурой ARM Cortex-M4.

Устройство SDK-1.1M описано в главе 3.

1.1. Сигналы сброса и синхронизации

Общие сведения

Сигнал сброса (reset) возвращает микроконтроллер в начальное состояние. При этом микроконтроллер начинает исполнять программу с начала. В стенде SDK-1.1M сброс можно выполнить, нажав на кнопку «RESET» на лицевой панели (рис. 35).

Для работы любого микроконтроллера необходимы синхросигналы, которые задают тактовые частоты работы процессорного ядра и других аппаратных блоков. В микроконтроллерах обычно предусмотрены несколько источников синхросигналов (внешних и внутренних, имеющих разную частоту работы) и широкие возможности по настройке их частот.

Пример настройки синхросигналов в STM32

Включить использование внешнего скоростного кварцевого резонатора (System Core → RCC→ High Speed Clock (HSE): выбрать «Crystal/Ceramic Resonator»).

Настроить дерево синхронизации (режим конфигурирования микроконтроллера, вкладка Clock Configuration), на примере STM32F427VI (рис. 2):

- ввести частоту внешнего скоростного кварцевого резонатора (HSE Input frequency): 25 МГц (указана на электрической принципиальной схеме стенда);
- выбрать HSE как источник синхросигнала для основного блока подстройки частоты (Main PLL);
- выбрать PLLCLK как источник синхросигнала SYSCLK;
- ввести желаемую частоту HCLK (основная частота микроконтроллера), равную максимальной (указана под данным полем), для STM32F427VI – 180 МГц.

После этого STM32CubeIDE автоматически рассчитает настройки дерева синхронизации для получения нужных частот с учетом всех ограничений.

1.2. Интерфейс ввода-вывода общего назначения (GPIO)

Общие сведения о GPIO

Интерфейс ввода-вывода общего назначения (general-purpose input/output, GPIO) – базовый интерфейс взаимодействия компьютерной системы с внешним миром. С его помощью к контактам (ножкам, «пинам») микроконтроллера чаще всего подключаются такие внешние элементы как светодиоды, кнопки, переключатели, осуществляется управление периферийными устройствами и т.д.

Контакты микроконтроллера внутри подключаются к портам ввода-вывода. Пример организации единичного порта ввода-вывода приведен на рис. 3. Такие порты GPIO обычно объединяются в группы по 8, 16 или 32 порта с общими регистрами управления. Одни и те же порты могут выступать в роли входа или выхода в зависимости от настроек. По умолчанию обычно включен режим входа, чтобы исключить влияние порта на другие части схемы. Для работы в качестве выхода необходимо настроить соответствующий порт на выход при помощи управляющих регистров (описаны в документации микроконтроллера). Каждый контакт настраивается индивидуально – в одной группе портов входы и выходы могут чередоваться в любых комбинациях.

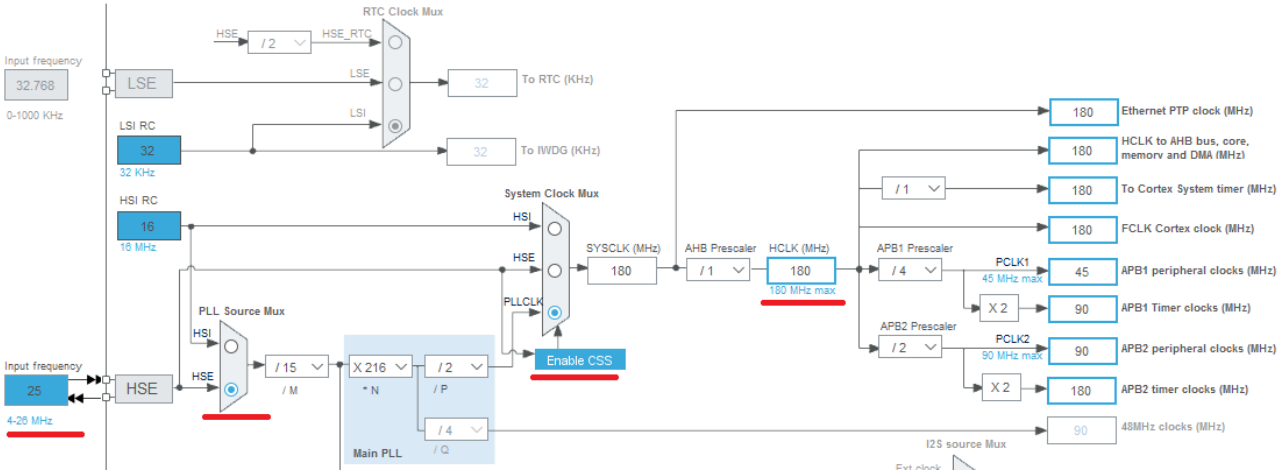


Рисунок 2 – Настройки дерева синхронизации микроконтроллера STM32F427VI

Также порты ввода-вывода позволяют в качестве «альтернативной функции» подключать к контактам микроконтроллера различные аппаратные блоки – контроллеры интерфейсов ввода вывода, генераторы сигналов и т.п.

В STM32 единичные порты GPIO объединяются в групповые порты по 16 штук, которые называются GPIOA, GPIOB и т.д. Соответствующие им контакты микроконтроллера называются PA1, PA2, …, PB1, PB2, … и т.д.

В стенде SDK-1.1M с процессорным модулем на базе STM32 на боковой панели имеется одна кнопка, подключённая к контакту PC15, и два управляемых светодиода: зеленый, подключенный к PD13, и двухцветный красный/желтый, подключенный к контактам PD14 и PD15 (рис. 4, 36).

Пример работы с GPIO в STM32

Произведем настройку контактов микроконтроллера в SDK-1.1M. В STM32CubeIDE в режиме конфигурирования микроконтроллера на вкладке Pinouts & Configuration необходимо перевести контакты PD13 – PD15 в режим GPIO_Output (рис. 5).

После сохранения конфигурации микроконтроллера STM32CubeIDE сгенерирует и добавит в проект код для инициализации микроконтроллера в соответствии с заданными настройками.

ВНИМАНИЕ: при повторной генерации файлов из них удаляется весь код, добавленный пользователем, кроме кода, который написан между парами комментариев вида `/* USER CODE BEGIN ... */, /* USER CODE END ... */`.

Для работы с GPIO, как и с другой периферией, на STM32 рекомендуется использовать библиотеку HAL (Hardware Abstraction Layer), поддержка которой встроена в IDE. При

генерации проекта STM32CubeIDE добавляет в код функцию `MX_GPIO_Init()`, инициализирующую порты GPIO в соответствии с настройками:

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
}
```

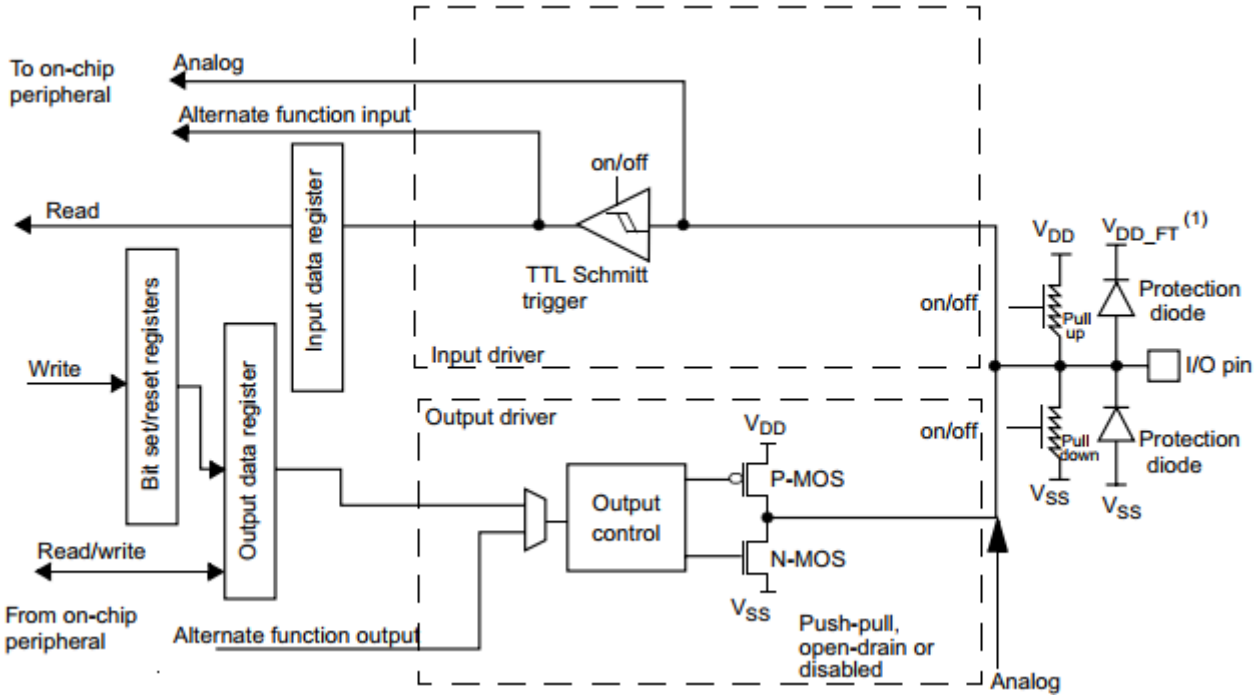


Рисунок 3 – Организация единичного порта ввода-вывода в STM32 [4]

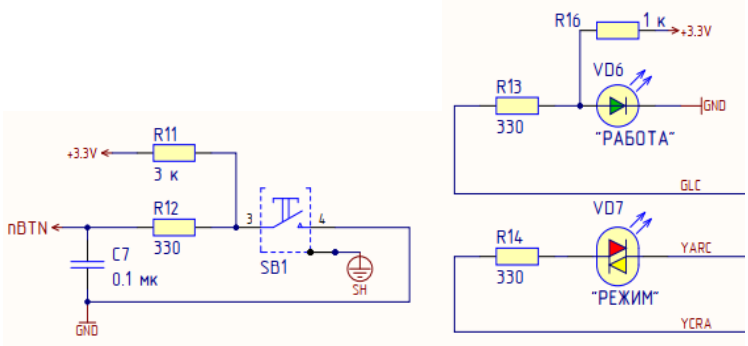


Рисунок 4 – Подключение кнопки и управляемых светодиодов на принципиальной электрической схеме SDK-1.1M

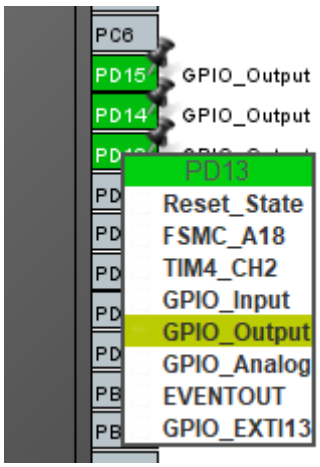


Рисунок 5 – Окно настройки портов GPIO

Для установки значения на выход и чтения входа используются стандартные функции из библиотеки HAL:

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

Параметры:

- `*GPIOx` – выбор порта; указатель на контекст драйвера конкретного порта; буква «x» соответствует имени порта; для управления светодиодами необходимо использовать порт GPIOD;
- `GPIO_Pin` – номер контакта, например, `GPIO_PIN_13;`
- `PinState` – состояние контакта: `GPIO_PIN_SET` («1») или `GPIO_PIN_RESET` («0»).

Для переключения состояния светодиода в противоположное состояние можно использовать следующую функцию:

```
HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

Для выполнения задержки на заданное количество миллисекунд (активное ожидание), получения текущего значения миллисекундного счетчика используются следующие функции:

```
void HAL_Delay(uint32_t Delay);
uint32_t HAL_GetTick(void);
```

Пример кода, который переключает состояние зеленого светодиода каждые 500 мс:

```
while (1)
{
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
        HAL_Delay(500);
}
```

Дребезг механических контактов

Механические контакты подвержены явлению дребезга. Его суть в том, что при замыкании контакта нажатием на кнопку напряжение устанавливается не сразу, а в течение некоторого времени (десятки миллисекунд) «скачет», пока контакт надежно не замкнется. После того, как кнопка будет отпущена, напряжение также «скачет», пока не установится на нужном уровне (рис. 6). Такое многократное замыкание/размыкание контактов вызвано тем, что контакты пружинят, обгорают и т. п.

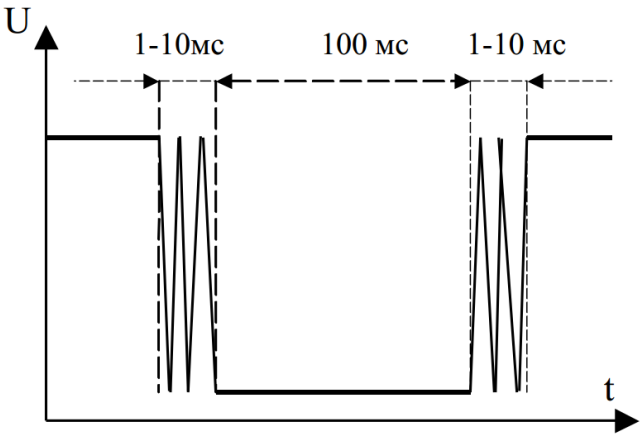


Рисунок 6 – Пример временной диаграммы напряжения цепи при нажатии кнопки с дребезгом контактов

Поскольку процессор реагирует на события очень быстро, то он может воспринять эти скачки напряжения за несколько нажатий. Решить данную проблему можно как аппаратно с помощью RS-триггера или триггера Шмитта, так и программно посредством отслеживания временных интервалов между замыканием и размыканием контактов.

Например, можно использовать небольшую задержку перед следующим опросом кнопки после фиксации замыкания. Задержка подбирается такой, чтобы дребезг успел прекратиться к ее окончанию. Если второй опроса также показал, что контакт замкнут, можно считать, что кнопка нажата.

Использование сигнала от кнопки как внешнего прерывания из-за дребезга требует реализации специальных механизмов ожидания и использования таймеров, поэтому является более сложным. Еще одним вариантом является постоянный периодический опрос состояния сигнала кнопки по прерыванию от таймера (будет рассмотрено в следующих разделах).

В SDK-1.1M аппаратная защита от дребезга не реализована, поэтому необходимо предусматривать один из вариантов программной защиты.

1.3.Прерывания

Общие сведения о прерываниях

Прерывание – процесс переключения процессорного ядра с одной программы на другую по внешнему сигналу с сохранением информации для последующего возобновления прерванной программы.

Основная роль механизма прерываний в процессорном ядре – реализация асинхронного режима работы программ и распараллеливания работы отдельных устройств вычислительного комплекса. Механизм прерываний реализуется аппаратными и программными средствами: контроллером прерываний и обработчиками прерываний.

Каждое событие, требующее прерывания, сопровождается сигналом («запросом») прерывания, оповещающим об этом вычислительную машину.

Обработка прерывания выполняется в три основных этапа:

1. **Прекращение выполнения текущей программы.** Для корректного возврата к выполнению программы после обработки прерывания необходимо предварительно сохранить в специальном стеке контекст программы – содержимое регистров микроконтроллера (счетчик команд и др.).

2. **Переход к выполнению программы обработчика.** Определяется приоритетный источник прерывания и соответствующий вектор прерывания. Адрес вектора прерывания записывается в регистр счетчика команд. После этого осуществляется переход к программе-обработчику прерывания и ее выполнение.

3. **Возврат управления прерванной программе.** Для корректного возврата управления необходимо восстановить контекст программы из стека. Последней командой программы обработки прерывания должна быть команда, которая осуществляет возврат в основную программу и восстановление предварительно сохраненного контекста.

Обработчик прерывания предназначен для быстрого реагирования на события, и в программе может быть определено множество обработчиков. В связи с этим время исполнения каждого из них должно быть максимально коротким, чтобы не задерживать обработку других прерываний. Крайне нежелательно выполнение в обработчике прерывания таких долгих операций, как обмен данными по опросу (с ожиданием конца обмена), задержка на определенное время и пр. Такие операции необходимо перекладывать на основной поток управления (главный цикл), который может быть прерван в любой момент.

Вектор прерывания – вектор начального состояния обработчика прерывания. Содержит всю необходимую информацию для перехода к обработчику, в том числе его начальный адрес. Каждому типу прерываний соответствует свой вектор прерывания, который инициализирует выполнение соответствующего обработчика. Обычно векторы прерывания хранятся в специально выделенных фиксированных ячейках памяти с короткими адресами, представляющих собой таблицу векторов прерываний. Для перехода к соответствующей прерывающей программе процессор должен располагать вектором прерывания и адресом этого вектора. По этому адресу, как правило, находится команда безусловного перехода к подпрограмме обработки прерывания.

Приоритеты прерываний – это механизм, позволяющий установить определенный порядок обработки запросов прерываний. При наличии нескольких запросов прерываний, поступивших одновременно, приоритет прерывания будет определять, какой из поступивших запросов будет обработан в первую очередь.

Вложенные прерывания – механизм, позволяющий осуществлять обработку поступившего запроса прерывания во время обработки другого прерывания. Если во время обработки прерывания поступает запрос на прерывание с более высоким уровнем приоритета, управление передается обработчику прерывания более высокого приоритета, при этом работа обработчика прерывания с более низким уровнем приоритета приостанавливается. Максимальное число обработчиков, которые могут приостанавливать друг друга, называется глубиной прерываний.

Более подробная информация об устройстве подсистемы прерываний и применении прерываний доступна в пособии [5], раздел 1.2.4 и пособии [6], раздел 2.3.

Критические секции

В некоторых случаях нельзя допускать, чтобы какой-либо участок программы был прерван другим процессом (например, вызовом прерывания). В таком случае, на время выполнения этого участка программы прерывания запрещаются (можно запрещать все прерывания или только те, которые в данном случае нежелательны). Такие участки программ называются критическими секциями.

В STM32 запрет и возобновление обработки всех прерываний можно выполнять следующим образом. В начале критической секции необходимо запомнить текущее значение глобального флага разрешения прерываний, и только после этого вызвать функцию запрета прерываний:

```
uint32_t pmask = __get_PRIMASK();
__disable_irq();
```

Запоминать текущее значение требуется, поскольку прерывания могут быть уже запрещены.

В конце критической секции восстанавливаем значение флага:

```
__set_PRIMASK(pmask);
```

Такой подход позволяет, в том числе, корректно создавать вложенные критические секции (например, если в критической секции вызывается функция, которая тоже содержит критическую секцию).

1.4.Последовательный интерфейс UART

Общие сведения об UART

Интерфейс UART (USART) широко применяется в вычислительной технике для связи между цифровыми устройствами и фактически является стандартом «де-факто» для подключения различных периферийных устройств (самый распространённый пример – беспроводные модемы).

Непосредственно с помощью UART соединяются отдельные физически близко расположенные микросхемы (как правило, микросхемы на одной плате) способом точка-точка. Для передачи сигналов между конструктивно самостоятельными устройствами и на большие расстояния сигналы UART необходимо пропустить через приёмопередатчик, преобразующий их в сигналы таких стандартов физического уровня, как RS-232 или RS-485 (см. табл. 1).

Интерфейс UART является дуплексным каналом передачи данных. Вообще обмен данными бывает:

- дуплексный (полнодуплексный) – одновременный прием и передача данных;
- полудуплексный – данные передаются в одном направлении с возможностью смены направления;
- симплексный – данные передаются только в одном направлении.

Таблица 1 – Сравнение интерфейсов RS-232 и RS-485

Характеристика	**RS-232**	**RS-485**
Топология	Точка-точка	Общая шина
Способ связи	Полнодуплексный	Полудуплексный
Уровни напряжений	«0» – от +5 до +15 В «1» – от -5 до -15 В	От -7 до +12 В отн. земли (дифференциальные сигналы)
Максимальное расстояние	До 15 м на скорости 115200 бит/с, больше при снижении скорости	До 1200 м на скорости 62,5 кбит/с, меньше при увеличении скорости
Кабель	Плоский шлейф	Витая пара

Современные микроконтроллеры практически всегда имеют в своём составе один или несколько отдельных функциональных блоков – контроллеров ввода-вывода UART (USART), аппаратно реализующих обмен данными по данному интерфейсу. USART отличается от UART наличием синхронного режима обмена данными. Контроллеры ввода-

вывода самостоятельно переключают соответствующие входы и выходы микроконтроллера, формируют и считывают сигналы. Управление контроллером ввода-вывода со стороны процессорного ядра осуществляется путем записи и чтения управляющих регистров, подключённых к системной шине микроконтроллера.

UART – последовательный интерфейс передачи данных. Это предполагает по одной сигнальной линии (проводу, проводнику на плате) для передачи данных в каждом направлении. Информационные биты передаются последовательно друг за другом.

Стандарт UART является чисто асинхронным интерфейсом, но реализующий его контроллер, как правило, может настраиваться в широких пределах и функционировать как в синхронном (USART), так и асинхронном (UART) режимах.

Синхронный режим предполагает наличие средств синхронизации передатчика и приемника. Как правило, для синхронизации используют специальную линию для передачи тактовых импульсов (синхросигналов). Информация с линии данных считывается приемником только по синхросигналу.

В асинхронном режиме при отсутствии передачи на линии данных установлена логическая «1». Перед очередным байтом информации передается специальный старт-бит, сигнализирующий о начале передачи (логический «0»). Затем следуют биты данных (их обычно восемь), за которыми может следовать дополнительный бит (его наличие зависит от режима передачи, обычно этот бит выполняет функцию контроля четности – «parity bit»). Завершается посылка стоп-битом (логическая «1»), длина которого (длительность единичного состояния линии) может соответствовать длительности передачи 1, 1,5 («полтора стоп-бита») или 2 бит. Стоп-бит гарантирует некоторую выдержку между соседними посылками, при этом пауза между ними может быть сколь угодно долгой (без учета «тайм-аута»). Если на линии данных долгое время находится логический «0», это считается ошибкой («break»). Пример посылки показан на рис. 7.

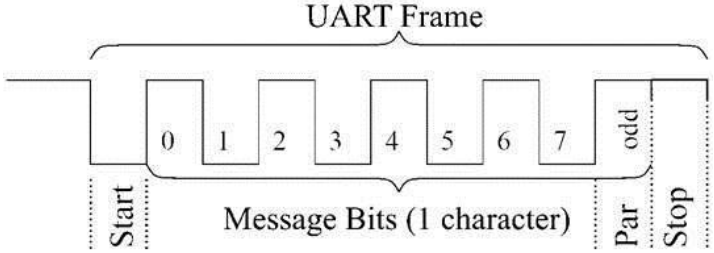


Рисунок 7 – Временная диаграмма передачи сообщения через UART

Для асинхронного режима предусмотрен ряд стандартных скоростей обмена: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 и 115200 бод. В бодах (baud) измеряется скорость изменения передатчиком состояния сигнальной линии за одну секунду. В простейшем случае (как в UART) в линии имеется всего два состояния сигнала – «0» и «1», т.е. одним состоянием кодируется один бит. Если все они несут полезную информацию, скорость изменения состояния в бодах совпадает со скоростью передачи двоичной информации, бит/с (bps, bit per second). Однако при использовании других методов модуляции возможны несколько состояний сигнала, что позволяет одним состоянием кодировать сразу несколько передаваемых бит, и здесь скорость передачи данных бит/с превышает скорость изменения сигнала в бодах. В UART фактическая скорость обмена полезными данными будет ниже бодовой частоты из-за накладных расходов на старт- и стоп-биты, биты четности, паузы.

Особенности реализации отладочного интерфейса UART в SDK-1.1M

Для взаимодействия стенда SDK-1.1M с персональным компьютером выделен один канал UART, который подключен к встроенному программатору-отладчику. Он обеспечивает преобразование сигналов интерфейса UART в виртуальный последовательный канал поверх USB, поэтому обмен данными физически осуществляется по тому же кабелю USB, через который происходит программирование. При подключении стенда к компьютеру в операционной системе должен определяться виртуальный COM-порт, номер которого можно узнать с помощью диспетчера устройств в Windows или аналогичных средств в других операционных системах (имя устройства – «RS232 (Interface 1)» или «SDK 1.1M Debugger (Interface 1)»).

Открыть терминал для отправки и получения данных через COM-порт можно с помощью таких программ как Putty, TeraTerm, Minicom или аналогичных. При этом необходимо корректно установить параметры обмена (скорость, количество бит в сообщении и пр.) так, чтобы их значения совпадали со значениями данных параметров, которые настроены для приемопередатчика в микроконтроллере.

Пример работы с UART в STM32

Произведем настройку UART для стенда SDK-1.1M. UART/USART подключается в режиме конфигурирования микроконтроллера на вкладке Pinouts & Configuration в разделе Connectivity. Для STM32F407VG (или STM32F427VI) выберем USART6, так как к отладочному разъему SDK-1.1M подключены сигналы «USART1/6_TX/VBUS» и «USART1/6_RX». Порты PC6 и PC7 необходимо настроить в режим сигналов USART6, если они не настроились автоматически. (Также можно использовать USART1 с выводом на PA9, PA10, но это не рекомендуется делать, так как основное назначение USART1 – работа на контактах PB6, PB7). Полный список вариантов назначения альтернативных функций портов микроконтроллера приведен в его листе данных (datasheet).

Установить следующие параметры USART6 (рис. 8):

- Mode (режим) – Asynchronous (асинхронный);
- Baud Rate (скорость обмена) – 115200 бит/с;
- Word Length (длина сообщения) – 8 бит;
- Parity (контроль четности) – None;
- Stop Bits (количество стоп-битов) – 1;
- Data direction (направление передачи) – Receive and Transmit (прием и передача).

При генерации проекта STM32CubeIDE создает функцию `MX_UART6_Init()`, инициализирующую USART6 в соответствии с настройками:

```
static void MX_USART6_UART_Init(void)
{
  huart6.Instance = USART6;
  huart6.Init.BaudRate = 115200;
  huart6.Init.WordLength = UART_WORDLENGTH_8B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart6) != HAL_OK)
```

```
  {
    Error_Handler();
  }
}
```

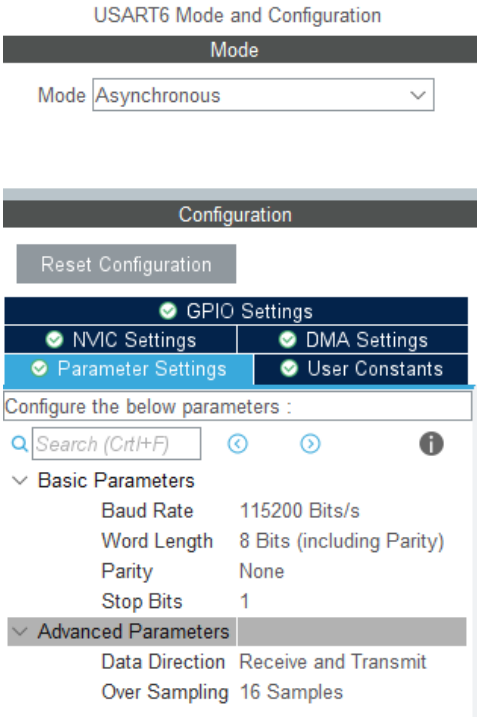


Рисунок 8 – Окно настройки UART

В библиотеке HAL существует две стандартных функции, отвечающих за прием и передачу данных:

```
HAL_StatusTypeDef HAL_UART_Receive(
  UART_HandleTypeDef *huart,
  uint8_t *pData,
  uint16_t Size,
  uint32_t Timeout);
HAL_StatusTypeDef HAL_UART_Transmit(…);
```

Параметры:

- `*huart` – указатель на структуру, которая содержит контекст драйвера конкретного контроллера UART;
- `*pData` – указатель на буфер данных;
- `Size` – количество байт, которые будут переданы или приняты;
- `Timeout` – максимальное время обмена;

Возвращаемое значение – статус приема/передачи.

Пример простейшей программы, которая каждую секунду выводит в консоль строку «Hello world!»:

```
char s[] = "Hello world!\n";
while (1)
{
      HAL_UART_Transmit( &huart6, (uint8_t *) s, sizeof( s ), 10 );
      HAL_Delay( 1000 );
}
```

Для того, чтобы узнать, был ли принят символ, функция приема данных `HAL_UART_Receive()` постоянно опрашивает состояние контроллера UART в течение времени тайм-аута. Поэтому такой режим работы называется работой «по опросу» (polling).

При опросе процессорное время тратится непродуктивно. Чтобы этого избежать, используется режим работы по прерыванию.

Для использования прерываний необходимо в меню настройки блока USART6 перейти на вкладку «NVIC Settings» (настройки контроллера прерываний) и отметить опцию «USART6 global interrupt» (рис. 9).

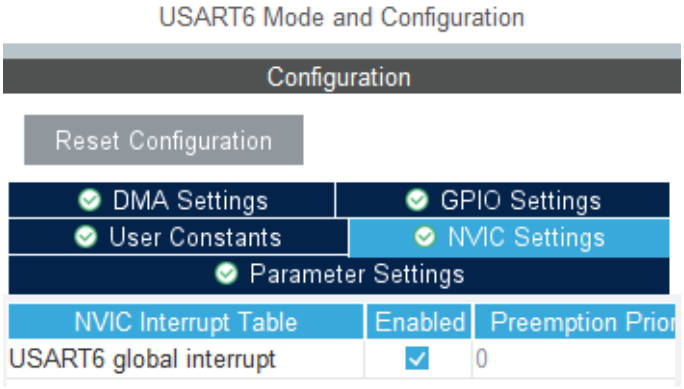


Рисунок 9 – Включение прерываний USART6

В код функции `MX_UART6_Init()` будет добавлено включение прерывания, а в код функции `MX_UART6_DeInit()` – выключение. Также, в файл stm32f4xx_it.c будет добавлен обработчик данного прерывания:

```
void USART6_IRQHandler(void)
{
  HAL_UART_IRQHandler(&huart6);
}
```

Этот обработчик привязан именно к UART6 и вызывает стандартный библиотечный обработчик прерываний UART, который использует контекст драйвера для обработки текущих заданий по обмену данными. Прерывание может инициироваться разными событиями, связанными с UART (прием байта, окончание отправки байта, ошибка приема и т.п.). По флагам прерываний библиотечный обработчик определяет причину вызова и выполняет необходимые действия.

В режиме прерываний можно использовать функции приема/передачи данных по прерыванию: `HAL_UART_Receive_IT()`, `HAL_UART_Transmit_IT()` и др. Эти функции не «ждут» конца обмена, а только инициализируют обмен и обработку прерываний по факту окончания приема/передачи. Сам обмен происходит «в фоне» относительно главного цикла. Когда обмен будет завершен, библиотечный обработчик прерывания вызовет callback-функцию `HAL_UART_RxCpltCallback()` или `HAL_UART_TxCpltCallback()`, которую можно переопределить для выполнения пользовательских действий. Например, поместить принятые данные в очередь или проверить, нет ли новых данных в очереди на передачу.

ВНИМАНИЕ: если вызывать функции, работающие в режиме прерывания (такие как `HAL_UART_Receive_IT()`), когда прерывания выключены, можно получить ситуацию, в которой драйвер окажется занятым навсегда (т.к. таймаута на такие операции нет). Даже если функция была вызвана до запрета прерываний, но обмен не был завершен в момент запрета, это тоже приводит к тому, что завершить его становится невозможно. Например, вызвана функция `HAL_UART_Receive_IT()`, данные так и не были приняты, и прерывания запрещаются. Драйвер занят, т.к. ждет прерывания по окончанию приема, но прерываний уже не будет – они запрещены. На вызовы функции `HAL_UART_Receive()` всегда будет возвращаться результат «драйвер занят». Чтобы избежать такой ситуации, перед запретом

прерываний необходимо дождаться завершения всех операций (и освобождения драйвера) или принудительно завершить их функцией `HAL_UART_Abort_IT()`.

В некоторых случаях функции стандартных драйверов библиотеки HAL являются недостаточно гибкими и не обеспечивают удобного приема и передачи данных. В таком случае необходимо разработать собственный вариант драйвера UART или расширить имеющийся. В частности, можно заменить вызов библиотечного обработчика в файле stm32f4xx_it.c на собственную функцию.

Пример организации буфера данных

Буферизация данных необходима для временного хранения данных, которые ждут своей очереди на передачу, или принятых данных, которые ждут своей очереди на обработку. Источник данных записывает в очередь, а приемник – вычитывает из нее.

Один из простейших вариантов такого буфера – очередь типа FIFO, организованная с использованием статически выделенного массива фиксированного размера и двух переменных.

Для организации буфера объявим массив элементов фиксированного размера N и две переменные: указатель и счетчик (можно также использовать два указателя и флаг заполненности массива; это требует трех переменных, но несколько меньшего количества вычислений). N будет максимальным количеством элементов в очереди и должно выбираться исходя из требований к взаимодействию процессов. Указатель содержит номер элемента массива, в который будет записан новый элемент, добавляемый в очередь. Изначально указатель равен 0 и при каждой записи увеличивается, пока не достигнет N – 1, а затем снова сбрасывается в 0, «бегая по кругу». Счетчик также вначале равен 0 и увеличивается при каждой записи в очередь. Его максимальное значение равно N. При попытке записи в очередь, в которой уже N элементов, новые данные либо должны быть отброшены без модификации указателя, либо должны заменить самые старые сохраненные в очереди данные (при выводе текстовых данных логично отбрасывать данные, которые не удается записать, и возвращать код ошибки). При чтении данных из очереди указатель не меняется, а только уменьшается счетчик, пока не достигнет 0. Номер элемента массива для считывания данных вычисляется из значений указателя и счетчика. В нормальном режиме работы не должно происходить переполнений очереди с потерей данных. Если это все-таки происходит, это означает, что неверно выбран размер очереди или допущены более серьезные ошибки проектирования программы.

Для удобства доступа к такому буферу разрабатываются специальные функции, которые скрывают манипуляции с указателями, а также реализуют защиту от гонок при доступе к ним. Гонки могут возникать, если два процесса, передающие данные через очередь, являются асинхронными (например, один является главным циклом программы, а второй – это обработчик прерывания). В таком случае один процесс в любом месте может прервать другой процесс и нарушить целостность общих данных, работа с которыми выполняется не атомарно. В нашем примере при добавлении элемента в очередь значения указателя и счетчика сначала анализируются, а затем изменяются. Это занимает несколько ассемблерных команд. Если эта последовательность команд будет прервана вызовом другого процесса, который извлекает данные из очереди, то значение счетчика в результате может оказаться некорректным: первый процесс прочитает переменную и начнет ее инкрементировать, а второй процесс в это время запишет уже уменьшенное значение. В

результате инкрементировано будет «старое» значение, а «новое» потеряется, и уменьшение фактически не произойдет. В итоге, в буфере «появятся» «лишние» данные.

В общем случае несогласованное состояние переменных может приводить к потерям данных или появлению «лишних» данных, утечкам памяти и обращению к памяти за пределами массива. Для обеспечения атомарности в случае, когда один или оба процесса работают по прерыванию, необходимо создавать критическую секцию (см. соответствующий раздел про прерывания), запрещая соответствующие прерывания или все прерывания на время выполнения операций с общими данными.

Все переменные и массивы, относящиеся к очереди, удобно объединять в структуру, указатель на которую передавать функциям манипуляции очередью. Тогда функции работы с очередью можно выделить в повторно используемый программный модуль, и использовать в программе одновременно несколько очередей, просто объявляя для каждой из них свой экземпляр структуры.

Пример набора функций, необходимых для работы с очередью:

- очистка очереди;
- помещение элемента в очередь;
- проверка наличия элементов в очереди;
- чтение элемента из очереди.

Пример использования очередей в драйвере UART

В драйвере UART, работающем по прерыванию, необходимо использовать очереди передаваемых и принимаемых данных, чтобы помещать туда данные, не дожидаясь готовности процесса-приемника их обработать.

Для очереди на передачу процесс-приемник реализуется обработчиком прерывания по окончанию передачи данных, а процесс-источник – это главный цикл программы и другие процессы, которые могут записывать в данную очередь. Для очереди принятых данных процессом-приемником является главный цикл программы, а процесс-источник реализуется обработчиком прерывания по окончанию приема данных. С точки зрения главного цикла программы ввод-вывод идет «фоновым» процессом.

У процесса передачи данных имеется внутреннее состояние – идет ли постоянная выборка данных из очереди с их последующей пересылкой в контроллер ввода-вывода. Когда главный цикл записывает данные в очередь, он проверяет, в каком состоянии находится процесс. Если передача идет, то ничего предпринимать не требуется: только что записанные данные будут переданы, когда до них дойдет очередь. Если же передача не идет, ее необходимо начать. Передача останавливается автоматически, когда в очереди заканчиваются данные.

Использование форматного вывода (printf)

С использованием функций последовательного ввода-вывода в UART можно реализовать поддержку привычных функций форматного ввода-вывода. Это можно сделать разными способами.

Можно вначале подготавливать отформатированную строку, а затем выводить ее в UART:

```
char buf[128];
```

```
sprintf( buf, "Number: %d\n", 123 );
HAL_UART_Transmit( &huart6, (uint8_t *) buf, strlen( buf ), 100 );
```

Другой вариант – переопределение функций низкоуровневого ввода-вывода, используемых стандартными функциями printf(), scanf() и т.д. Для использования функции printf() достаточно переопределить функцию _write():

```
int _write(int file, char *ptr, int len )
{
    return HAL_UART_Transmit( &huart6, (uint8_t *) ptr, len, 100 );
}
```

Данное определение должно быть размещено в каком-либо файле исходных текстов программы, например, main.c. Если используется ввод-вывод по прерыванию, функция должна быть соответственно изменена.

1.5.Программируемые таймеры

Таймеры в STM32

В микроконтроллерах семейства STM32 существует несколько типов аппаратных таймеров, которые можно использовать для точного контроля отрезков времени и регулярного выполнения операций, управления контактами микроконтроллера в режиме широтно-импульсной модуляции (ШИМ; pulse-width modulation, PWM), измерения импульсов на входах (input capture) и т.п. Для обеспечения точных временных характеристик работы с сигналами на входах/выходах микроконтроллера таймеры имеют специальные аппаратные расширения – каналы ввода-вывода, подключенные к контактам микроконтроллера.

ШИМ-сигнал изображен на рис. 10. Его основной характеристикой является **скважность** – отношение периода следования (повторения) импульсов одной последовательности к их длительности. Величина, обратная скважности, называется **коэффициентом заполнения (duty cycle)**.

Основным назначением таймера является увеличение или уменьшение значения в счетном регистре по каждому входному импульсу. Границы счета зависят от настроек. Таймеры имеют режим автоперезагрузки (autoreload), что позволяет не настраивать их после каждого окончания счета.

Большинство таймеров может использовать в качестве входных импульсов как внутренние синхросигналы микроконтроллера, так и внешние сигналы, подаваемые на счетный вход через внешний контакт микроконтроллера, с другого таймера и т.д. Когда импульсы следуют не через регулярные промежутки времени, то говорят, что таймер работает в режиме счетчика (импульсов). Таймер с такими возможностями называют таймером-счетчиком.

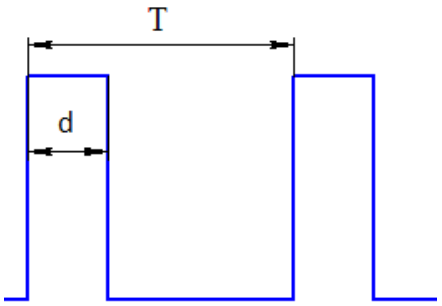


Рисунок 10 – ШИМ-сигнал: T – период импульсов, d – длительность импульса

Краткое описание типов таймеров микроконтроллеров STM32F407 и STM32F427 из RM0090:

1. Простые таймеры (basic timers) TIM6 и TIM7. 16-битные таймеры без каналов ввода-вывода.

2. Таймеры общего назначения (general-purpose timers) TIM2 – TIM5. 16- и 32-битные таймеры с 4 каналами ввода-вывода.

3. Таймеры общего назначения (general-purpose timers) TIM9 – TIM14. 16-битные таймеры с 1–2 каналами ввода-вывода.

4. Таймеры TIM1 и TIM8 с расширенными возможностями (advanced control timers). 16-битные таймеры с 4 каналами ввода-вывода и дополнительными функциями настройки каналов.

На рис. 11 изображена структурная схема аппаратного блока таймера типа TIM2 – TIM5 с 4 каналами ввода-вывода.

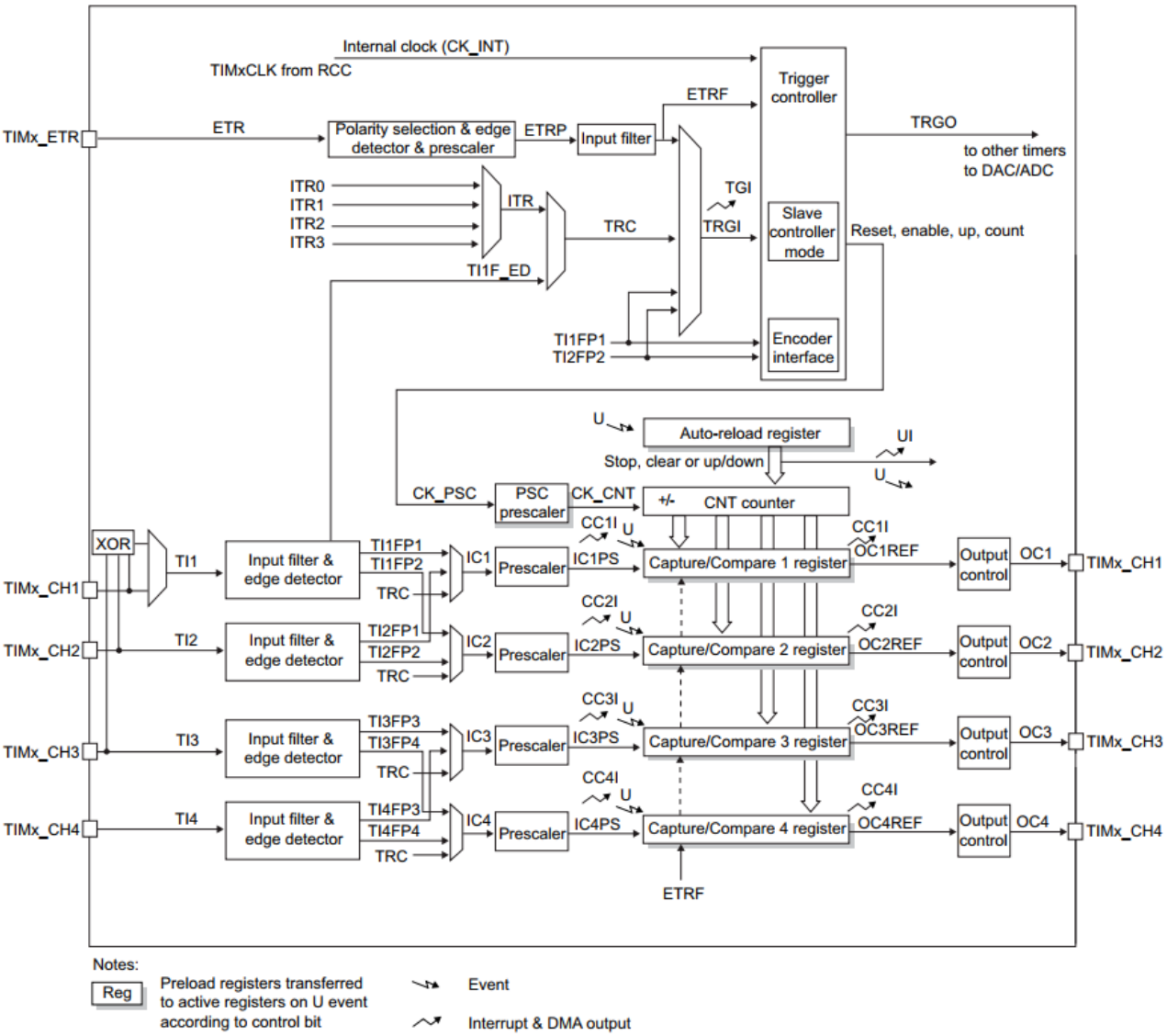


Рисунок 11 – Структурная схема блока таймера (TIM2 – TIM5) [4]

Пример использования таймеров в STM32

Рассмотрим пример использования двух таймеров в программе для стенда SDK-1.1M. Один таймер будет задавать яркость зеленого светодиода путем подачи на него ШИМ-сигнала с необходимыми характеристиками, а второй таймер – отсчитывать периоды времени горения зеленого светодиода с заданной скважностью.

13-й сигнал порта GPIOD, к которому подключен зеленый светодиод, в качестве одной из альтернативных функций имеет 2-й канал таймера TIM4. Следовательно, мы можем использовать аппаратные возможности TIM4 для генерации ШИМ-сигнала. Таблица альтернативных функций приведена в datasheet микроконтроллера (рис. 12).

Чтобы настроить таймер TIM4, вначале определим, какова частота его внутреннего синхросигнала. По схеме в datasheet найдем, к какой шине подключен таймер. Это шина APB1 (рис. 13а). В конфигураторе микроконтроллера проверим установленную частоту синхросигнала для таймеров на этой шине. В нашем примере она составляет 90 МГц (рис. 13б).

Table 12. STM32F427xx and STM32F429xx alter

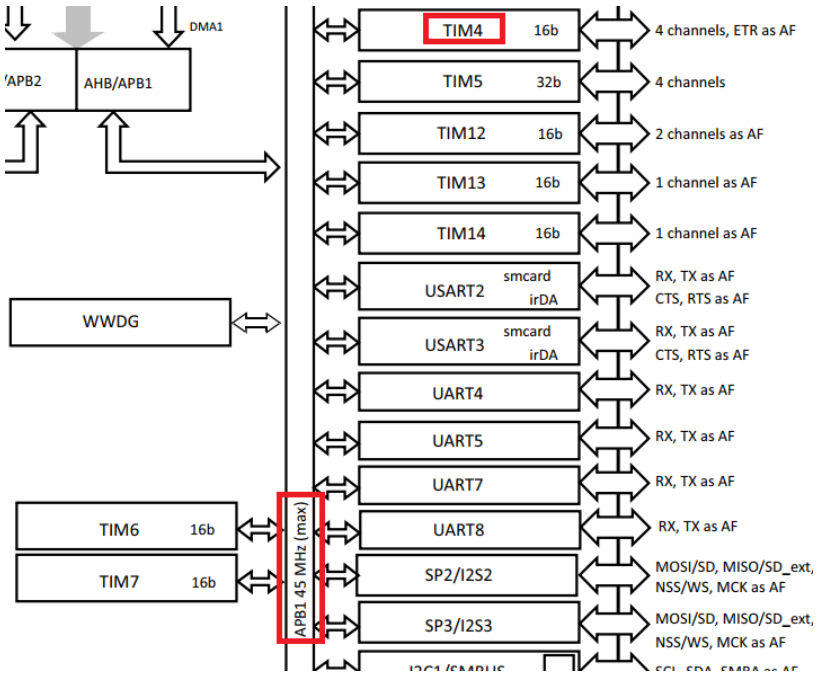
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF
		SYS	TIM1/2	TIM3/4/5	TIM8/9/ 10/11	I2C1/ 2/3	SPI1/2/ 3/4/5/6	SPI2/3/ SAI1	SPI3/ USART1/ 2/3	USAI UART /8
Port D	PD7	-	-	-	-	-	-	-	USART2_ CK	-
	PD8	-	-	-	-	-	-	-	USART3_ IX	-
	PD9	-	-	-	-	-	-	-	USART3_ RX	-
	PD10	-	-	-	-	-	-	-	USART3_ CK	-
	PD11	-	-	-	-	-	-	-	USART3_ CTS	-
	PD12	-	-	TIM4_ CH1	-	-	-	-	USART3_ RTS	-
	PD13	-	-	TIM4_ CH2	-	-	-	-	-	-
	PD14	-	-	TIM4_ CH3	-	-	-	-	-	-
	PD15	-	-	TIM4_ CH4	-	-	-	-	-	-

Рисунок 12 – Фрагмент таблицы альтернативных функций портов ввода-вывода [7]

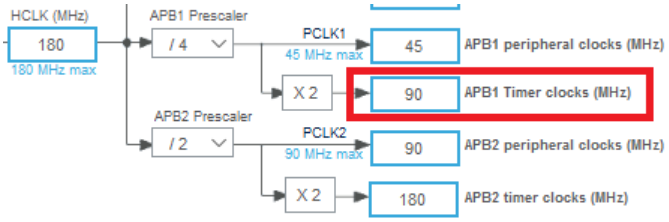
Теперь выберем для PD13 функцию TIM4_CH2 и настроим параметры таймера TIM4, как показано на рис. 14. Включено тактирование таймера от внутреннего источника, включена генерация ШИМ на канале 2, предделитель (prescaler) таймера будет делить входную частоту на 90, то есть счетный регистр будет тактироваться частотой 1 МГц. Таймер считает от нуля «вверх». Период счета (значение регистра автоперезагрузки) установлено в 999, чтобы регистр счета сбрасывался каждые 1000 тактов счетчика, то есть с периодом в 1 мс.

Для ШИМ на канале 2 установлен режим 1 и длительность импульса в 500 тактов счетчика. В результате каждую миллисекунду будет генерироваться импульс в 500 мкс, что соответствует коэффициенту заполнения ШИМ-сигнала в 50 % или скважности 2.

Светодиод, на который подается ШИМ-сигнал с коэффициентом заполнения 50 %, будет гореть лишь половину времени, а из-за очень малого периода включения-выключения визуально будет казаться, что он равномерно горит с половиной максимальной яркости.



а)



б)

Рисунок 13 – Частота синхросигнала таймеров шины APB1 [7]

В результате генерации кода в проект будут добавлены следующие функции:

```
void MX_TIM4_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 89;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 999;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 500;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htim4);
}

void HAL_TIM_MspPostInit(TIM_HandleTypeDef* timHandle)
{
  if(timHandle->Instance==TIM4)
  {
    __HAL_RCC_GPIOD_CLK_ENABLE();
    /**TIM4 GPIO Configuration
    PD13     ------> TIM4_CH2
    */
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF2_TIM4;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
  }
}
```

После генерации кода остается добавить в программу включение таймера и генерации ШИМ-сигнала. Для этого после вызовов функций инициализации перед главным циклом в функции main() добавим строку:

```
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
```

Изменяя длительность импульса от 0 до 1000 тактов в настройках, можно получить разную яркость светодиода от полностью выключенного до максимальной интенсивности.

ВНИМАНИЕ: если генерация сигнала не включена или выключена функцией `HAL_TIM_PWM_Stop()`, соответствующий контакт микроконтроллера находится в режиме «входа» (а не «выхода» с выдачей «0»). Поэтому для работы с желтым и красным светодиодами второй контакт надо сохранять выходом с «0», задавая нулевую длительность импульса, а не выключая генерацию.

Дополним программу, добавив использование второго таймера, с помощью которого реализуем автоматическое изменение яркости с некоторой периодичностью. Для этого не нужны каналы ввода-вывода сигналов, поэтому возьмем самый простой таймер TIM6. Он тоже тактируется от шины APB1 (рис. 13). Настроим его для работы в режиме автоперезагрузки с генерацией прерываний по каждому переполнению. Период переполнения установим в 1 с. Настройки приведены на рис. 15 (заметим, что значения

предделителя и перезагрузки выбраны так, чтобы уложиться в 16-битную разрядность соответствующих регистров). Кроме этого, необходимо установить галочку разрешения прерываний на вкладке «NVIC Settings».

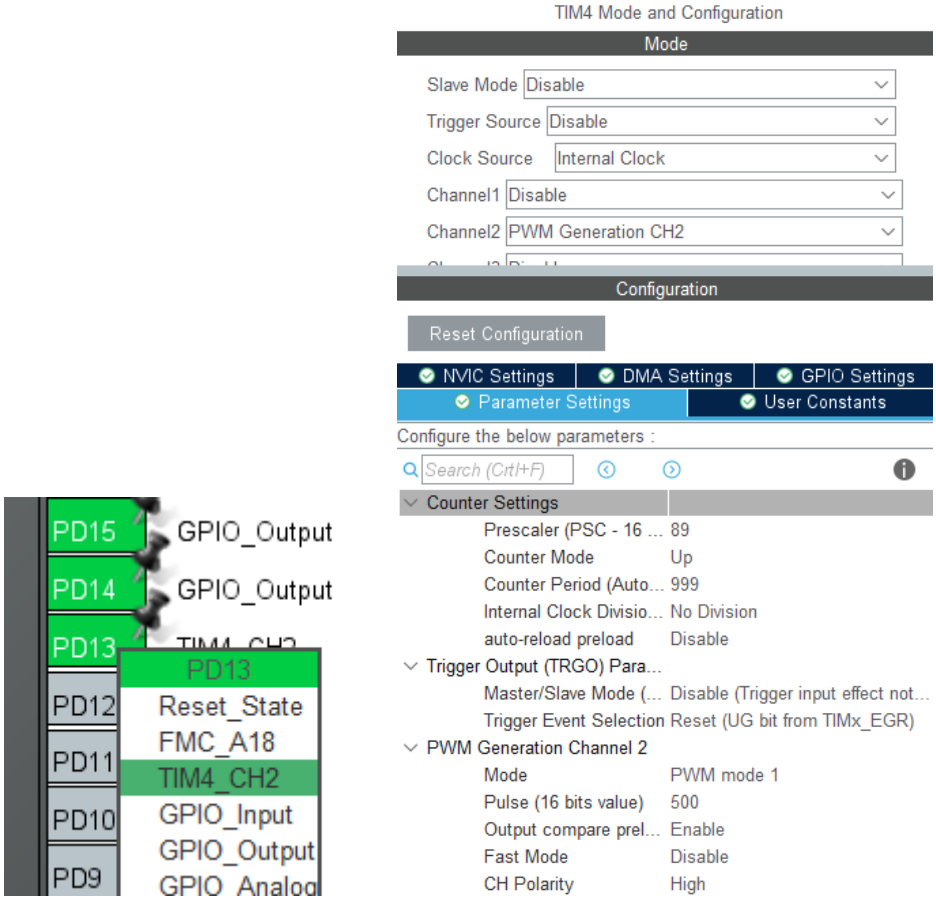


Рисунок 14 – Настройка ШИМ на канале 2 таймера TIM4

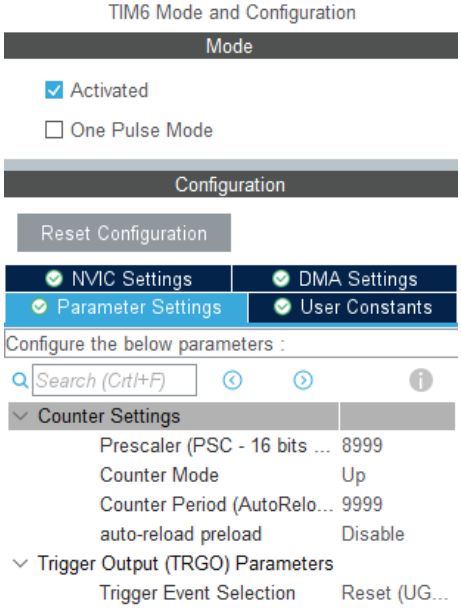


Рисунок 15 – Настройка таймера TIM6

Добавим callback-функцию, которая будет менять длительность импульса в канале 2 таймера TIM4 при каждом прерывании по окончанию счета (перезагрузке) таймера TIM6. Итак, каждую секунду яркость зеленого светодиода будет возрастать на 10 %.

```
#define MAX_PHASE 10
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
      static int phase = 0;
      if( htim->Instance == TIM6 )
      {
            phase++;
            if( phase > MAX_PHASE ) phase = 0;
            htim4.Instance->CCR2 = 100 * phase;
      }
}
```

Заметим, что мы записываем новое значение напрямую в регистр CCR2 (capture-compare resister of channel 2) таймера TIM4. Использование функций HAL для установки этого регистра требовало бы остановки генерации ШИМ-сигнала, смены настроек и нового запуска. Аналогично можно записывать значения в регистр автоперезагрузки.

Теперь добавим в функцию main() включение таймера TIM6. Функция примет вид:

```
...
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM4_Init();
MX_TIM6_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
HAL_TIM_Base_Start_IT(&htim6);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
...
```

Заметим, что, если бы на контакт управления зеленым светодиодом не был выведен аппаратный канал таймера, генерацию ШИМ-сигнала можно было бы реализовать программно по аналогии с использованием таймера TIM6 в примере выше. Для этого нужно установить малый период перезагрузки и в обработчике прерываний переключать выход GPIO. Это решение требует использования процессорного времени и обеспечивает меньшую точность временных характеристик сигнала, но позволяет генерировать ШИМ-сигналы сразу по нескольким выходам.

Использование излучателя звука

Излучатель звука – это электромагнитный динамик, который управляется ШИМ-сигналом. Излучатель подключен к порту PE9, на который выводится канал 1 таймера TIM1, поэтому ШИМ-сигнал можно генерировать любым из двух описанных выше способов. Частота звука соответствует частоте сигнала. Громкость можно грубо регулировать коэффициентом заполнения. При 50 % громкость максимальная. Она начинает заметно уменьшаться, когда коэффициент заполнения становится меньше 5 % или больше 95 % (на короткий импульс электромагнит реагирует слабее).

ВНИМАНИЕ: чтобы излучатель звука воспринимал сигнал от микроконтроллера, необходимо замкнуть 7-й контакт движкового переключателя SA2 на лицевой панели стенда (см. рис. 16, 35).

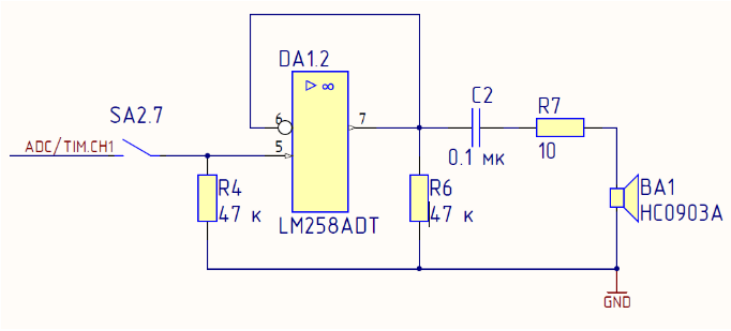


Рисунок 16 – Тракт управления излучателем звука

Использование таймеров для опроса кнопок

Периодическое прерывание от таймера можно использовать для опроса состояний кнопок, одновременно устраняя возможный дребезг контактов. Алгоритм опроса рекомендуется реализовать в виде конечного автомата (Finite State Machine, FSM) – функции, которая в зависимости от своего состояния (значения определенной переменной) и входного воздействия, выполняет разную работу.

Функция опроса кнопки и определения ее нажатия/отпускания вызывается в обработчике прерывания от таймера. Сообщения о событиях (нажатие/отпускание кнопок) помещаются в очередь, из которой их может вычитывать прикладная программа.

С программной точки зрения кнопка может находиться только в двух состояниях (рис. 16, а): нажата (FSM_KEY_PRESSED) и не нажата (FSM_KEY_RELEASED). Сначала в обработчике прерывания от таймера выполняется опрос текущего физического состояния кнопки: контакты замкнуты или разомкнуты. После этого счетчик key_count инкрементируется (если контакты замкнуты) или декрементируется (если разомкнуты). Далее в виде конечного автомата реализуется алгоритм определения нажатия/отпускания клавиши на основе значения счетчика: если значение key_count превышает порог нажатия кнопки (KEY_PRESSED_COUNT), то фиксируется ее нажатие (в буфер клавиатуры записывается сообщение MSG_KEY_PRESSED); если значение key_count падает ниже порога отпускания кнопки (KEY_RELEASED_COUNT), то фиксируется ее отпускание (в буфер клавиатуры записывается сообщение MSG_KEY_RELEASED). Таким образом используется свойство гистерезиса: любой дребезг и другие помехи, которые приводят к отклонению key_count на величину меньше (KEY_PRESSED_COUNT – KEY_RELEASED_COUNT), отсекаются. Состояния нажатой и отпущенной кнопки четко разделяются (рис. 16, б).

Особенностью такой схемы работы является то, что пороговые значения счетчика нажатий кнопки (KEY_PRESSED_COUNT и KEY_RELEASED_COUNT) прямо пропорциональны частоте прерываний от таймера, поэтому могут легко настраиваться при повторном использовании драйвера опроса кнопок в других программах. Кроме того, данная схема абсолютно симметрично обрабатывает дребезг при нажатии и отпускании клавиши, а свойство гистерезиса позволяет регулировать чувствительность клавиатуры. Если ввести новые пороговые значения счетчика нажатий клавиши key_count, то можно организовать работу с клавиатурой с переповторами, т.е. с отслеживанием длительного удержания кнопки и генерацией в таком состоянии периодических событий о нажатии (как на клавиатуре персонального компьютера).

Данный принцип организации работы с одной кнопкой можно перенести на набор кнопок и, в том числе, клавиатуру.

1.6.Интерфейс I²C

Общие сведения об интерфейсе I²C

I²C (I2C, inter-integrated circuit, т.е. межмикросхемный интерфейс) – двухпроводной интерфейс для организации взаимодействия между микросхемами электронных устройств. Во время обмена данными одно устройство играет роль ведущего (master), а второе – ведомого (slave). Допускается одновременное подключение нескольких ведущих устройств к одной шине I²C. При этом для устранения конфликтов доступа к шине предусмотрен специальный механизм арбитража.

На рис. 17 – 19 приведены временные диаграммы сигналов на линиях I2C_SCL и I2C_SDA для старт- и стоп-последовательностей, передачи битов данных, бита подтверждения.

Более подробная информация об интерфейсе I2C доступна в дополнительных источниках: статья [8], пособие [6] (раздел 3.10.3).

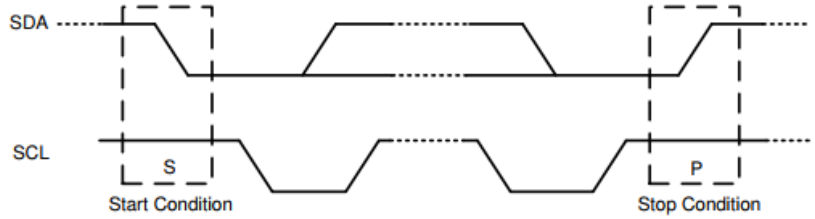


Рисунок 17 – Старт- и стоп-последовательности на шине I²C

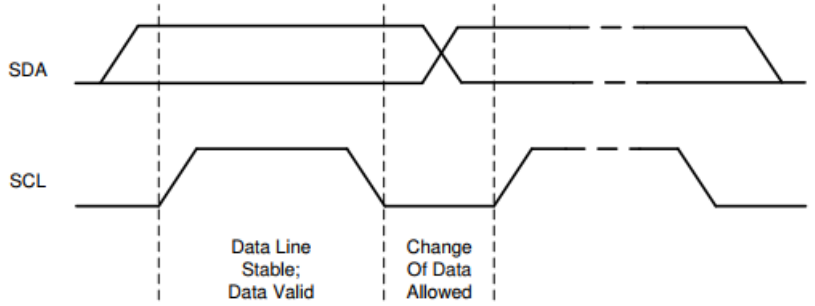


Рисунок 18 – Передача битов данных по шине I²C

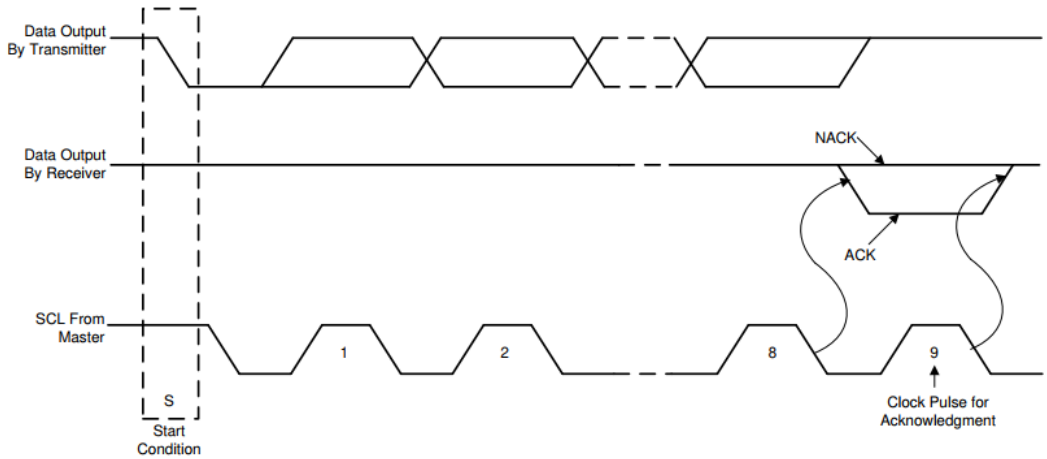


Рисунок 19 – Бит подтверждения приема байта на шине I²C

Пример использования I²C в STM32

В стенде SDK-1.1M имеется одна шина I²C, которая подключена к контактам микроконтроллера PB8 и PB9. На эти контакты выводится первый из трех контроллеров I²C, имеющихся в микроконтроллере. Чтобы включить его, в графическом конфигураторе на вкладке Pinouts & Configuration в разделе Connectivity необходимо выбрать I2C1 и в разделе Mode задать режим «I2C». После этого необходимо проверить, на какие контакты микроконтроллера были автоматически назначены сигналы I²C. Если это не PB8 и PB9, то исправить вручную: настроить нужные контакты, а предыдущие отключатся автоматически.

Настройки контроллера I²C следует задать как на рис. 20 и сгенерировать код для проекта.

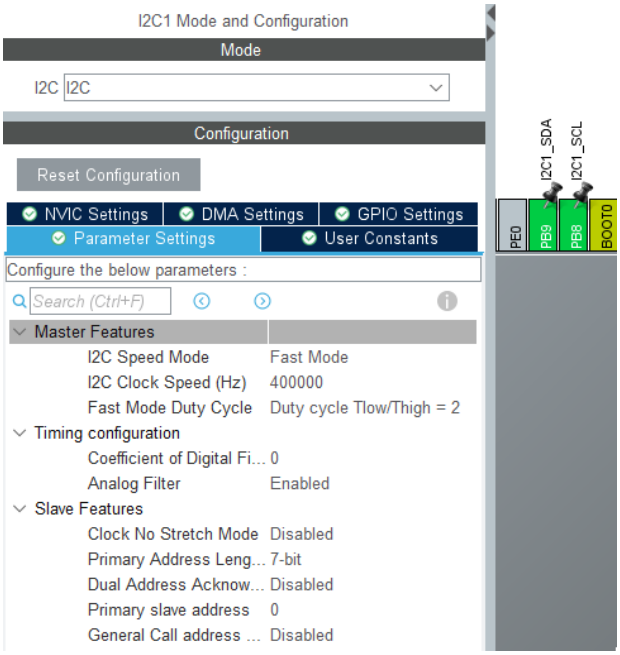


Рисунок 20 – Пример настройки контроллера I²C

STM32CubeIDE добавит в проект следующую функцию:

```
void MX_I2C1_Init(void)
{
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 400000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Стандартный драйвер I²C имеет функции чтения и записи данных по шине I²C:

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(
  I2C_HandleTypeDef *hi2c,
  uint16_t DevAddress,
  uint8_t *pData,
  uint16_t Size,
  uint32_t Timeout);
HAL_StatusTypeDef HAL_I2C_Master_Receive(…);
```

Параметры:

- `*hi2c` – указатель на структуру, которая содержит контекст драйвера конкретного контроллера I²C;
- `DevAddress` – адресный байт ведомого устройства I²C (8 бит), причем младший бит R/W# можно оставлять равным 0, т.к. он будет подставлен драйвером автоматически;
- `*pData` – указатель на буфер данных;
- `Size` – количество байт, которые будут переданы или приняты;
- `Timeout` – максимальное время обмена.

Возвращаемое значение – статус приема/передачи.

1.7.Матричная клавиатура

Общие сведения о матричной клавиатуре

Клавиатура SDK-1.1M организована в виде матрицы 3×4. Строки и столбцы клавиатуры подключены к контактам 8-разрядной микросхемы-расширителя портов ввода-вывода PCA9538 (рис. 21), которая подключается к микроконтроллеру по интерфейсу I²C. К контактам P0 – P3 подключены строки, к P4 – P6 подключены столбцы. Доступ к расширителю организован как чтение/запись байта, в котором каждый бит отвечает за определенный контакт микросхемы PCA9538.

Все сигналы, соответствующие столбцам, схемотехнически подтянуты к «1». Чтобы узнать состояние кнопок клавиатуры, необходимо провести сканирование, в котором последовательно считывается состояние кнопок каждой строки. Для проведения цикла сканирования одной строки на сигнал, соответствующий опрашиваемой строке, выставляется «0», а сигналы остальных строк должны быть в режиме входа. После этого считывается состояние сигналов, соответствующих столбцам. Нажатым кнопкам в опрашиваемой строке будут соответствовать «0» на линии столбца, а не нажатым – «1». Для полного опроса клавиатуры необходимо четыре таких цикла опроса строк, и для опроса каждого ряда необходимо выполнять два обмена по I²C: выдача «нуля» на очередной ряд и чтение состояния столбцов.

В матричной клавиатуре однозначно определяется одновременное нажатие не более чем двух кнопок. Если нажато три кнопки, которые расположены в вершинах

прямоугольника, то возникает эффект «фантомного нажатия»: кнопка, находящаяся в четвертой вершине прямоугольника, тоже будет считываться как нажатая.

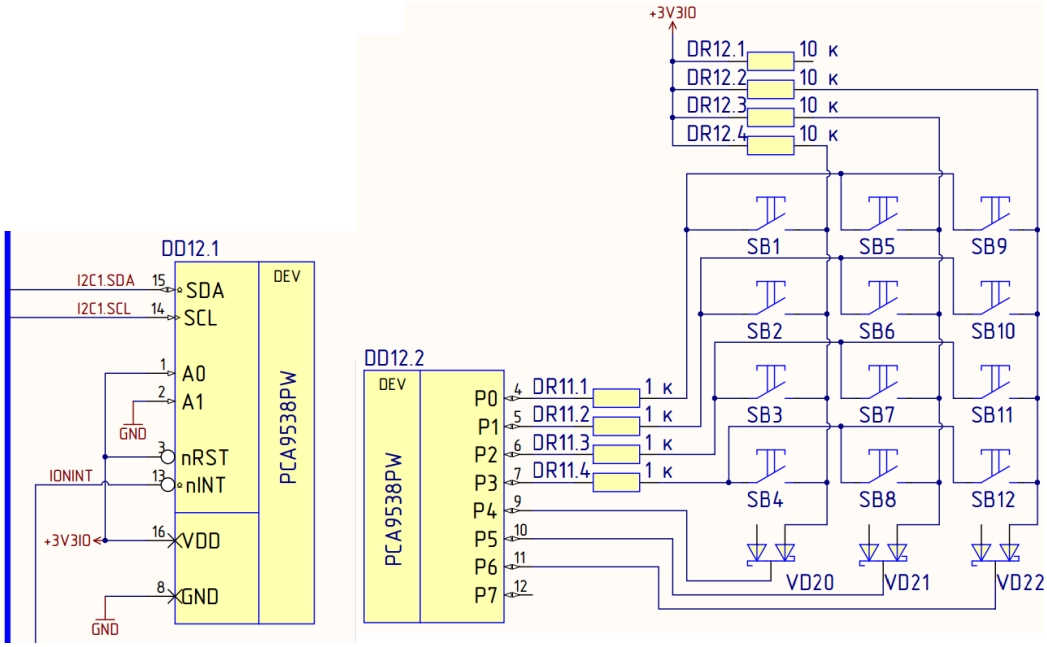


Рисунок 21 – Схема подключения матричной клавиатуры к PCA9538

Расширитель портов ввода-вывода PCA9538

Микросхема PCA9538 [9] является внешним расширителем GPIO-портов ввода-вывода, который подключается к микроконтроллеру по интерфейсу I^2C. Данная микросхема реализует подключение до 8 сигналов, каждый из которых может быть настроен как вход или выход.

При взаимодействии с микросхемой адресный байт (рис. 22) содержит постоянную часть и два бита (A1, A0), значения которых определяются в зависимости от того, как подключены соответствующие контакты микросхемы на печатной плате устройства – к земле или питанию. Соответственно, возможно одновременное подключение на шину I^2C до 4 микросхем PCA9538. В SDK-1.1M имеется две микросхемы – с адресами 0xE0 (входы прерываний устройств и др.) и 0xE2 (клавиатура). Последний бит адресного байта подчиненного устройства определяет операцию (чтение или запись), которая должна быть выполнена.

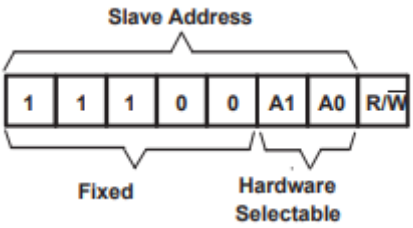


Рисунок 22 – Адресный байт PCA9538

После успешного подтверждения командного байта ведущее устройство шины отправляет командный байт (табл. 2). Значение командного байта записывается в регистр управления (control register, биты B7 – B0), и определяет регистр микросхемы, который будет использован далее для чтения или записи.

Таблица 2 – Командный байт [9]

Биты регистра управления		Командный байт	Регистр	Операции	Значение по умолчанию
B1	B0				
0	0	0x00	Входной порт	Чтение	XXXX XXXX
0	1	0x01	Выходной порт	Чтение/запись	1111 1111
1	0	0x02	Инверсия	Чтение/запись	0000 0000
1	1	0x03	Конфигурация	Чтение/запись	1111 1111

Процедура записи регистра. Данные передаются на PCA9538 путем отправки адреса устройства и установки младшего значащего бита в «0». После адресного байта отправляется командный байт, который определяет, какой регистр получает данные, следующие за командным байтом (рис. 23).

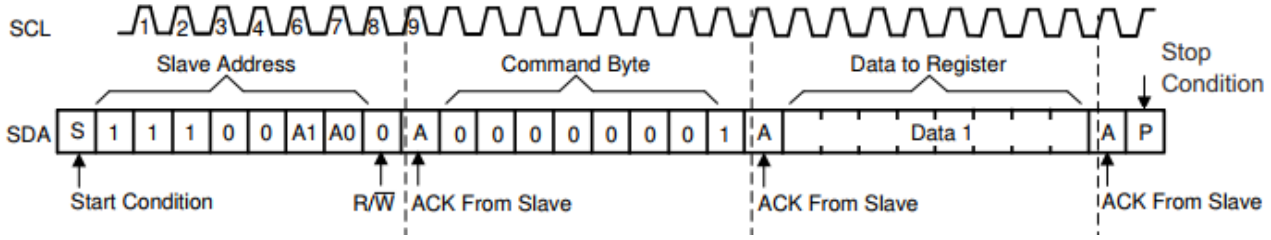


Рисунок 23 – Процедура записи регистра в PCA9538 [9]

Процедура чтения регистров. Сначала ведущее устройство шины должно отправить адресный байт PCA9538 с младшим значащим битом, установленным в «0» (запись). После адреса отправляется командный байт, и определяет, к какому регистру обращаются. После этого на шине I^2C формируется последовательность «повторный старт», снова отправляется адресный байт, но на этот раз младший значащий бит устанавливается в «1» (чтение). После этого данные из регистра, определенного командным байтом, читаются из PCA9538 (рис. 24). На количество байтов данных, полученных в одной передаче чтения, ограничения нет (можно прочитать все 4 регистра подряд). Когда ведущее устройство читает последний нужный ему байт, оно не должно подтверждать данные.

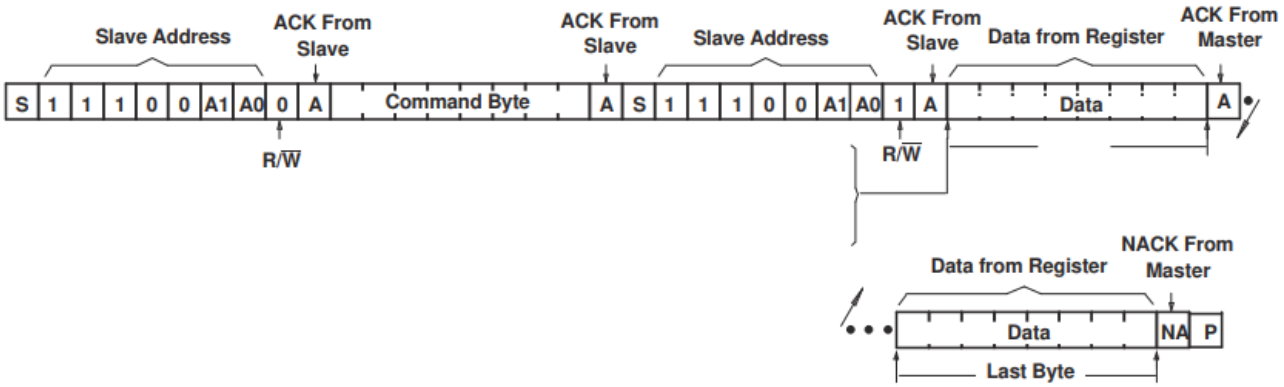


Рисунок 24 – Процедура чтения регистров из PCA9538 [9]

Регистр входного порта (input port register, регистр 0) отражает входящие логические уровни контактов микросхемы независимо от того, определен ли контакт как вход или выход регистром конфигурации. Действует только на операцию чтения. Запись в этот регистр не оказывает никакого эффекта.

Регистр выходного порта (output port register, регистр 1) устанавливает исходящие логические уровни контактов, определенных как выходы регистром конфигурации. Битовые значения в этом регистре не влияют на контакты, определенные как входы. В свою очередь,

чтение из этого регистра возвращает значения, которые находятся в триггерах, управляющих контактами, а не фактические уровни сигнала на контактах.

Регистр инверсии полярности (polarity inversion register, регистр 2) позволяет инвертировать полярность сигналов, определенных как входы регистра конфигурации. Если бит в этом регистре установлен (записана «1»), полярность соответствующего выходного порта инвертирована. Если бит в этом регистре сброшен (записывается «0»), полярность соответствующего выходного порта сохраняется.

Конфигурационный регистр (configuration register, регистр 3) устанавливает направление ввода-вывода. Если бит в этом регистре установлен в «1», соответствующий контакт активируется как вход. Если бит в этом регистре принимает значение «0», соответствующий контакт активируется как выход.

Обмен данными с расширителем портов ввода-вывода

Для чтения и записи по интерфейсу I^2C регистров в микросхемах, которые требуют установки адреса внутреннего регистра (например, микросхема PCA9538), в библиотеке HAL существуют функции, который сразу выполняют все необходимые действия:

```
HAL_I2C_Mem_Write(
  I2C_HandleTypeDef *hi2c,
  uint16_t DevAddress,
  uint16_t MemAddress,
  uint16_t MemAddSize,
  uint8_t *pData,
  uint16_t Size,
  uint32_t Timeout);
HAL_I2C_Mem_Read(…);
```

Параметры аналогичны описанным выше с тем же именами. Новые параметры:

- `MemAddress` – адрес внутри ведомого устройства I^2C, который записывается в устройство после передачи адресного байта (например, номер регистра PCA9538);
- `MemAddSize` – размер адреса внутри ведомого устройства в байтах; может быть равен 1 или 2 (для PCA9538 равен 1).

Также существуют варианты этих функций, работающие в режиме прерывания аналогично драйверу UART. Чтобы их использовать, необходимо разрешить прерывания I^2C в настройках NVIC.

Варианты организации драйвера клавиатуры

Возможны несколько основных способов организации драйвера клавиатуры:

1. Работа с I^2C **по опросу (без прерываний от I^2C)**. Опрос клавиатуры периодически выполняется из главного цикла while() в функции main(). При этом остальной код главного цикла должен отдавать управление достаточно быстро, чтобы не создавать заметных пауз в опросе клавиатуры.

2. Работа с I^2C **по прерываниям в главном цикле**. Аналогично предыдущему, функции опроса клавиатуры вызываются из главного цикла, но используются функции, работающие в режиме прерывания. Для получения результата надо дожидаться окончания выполнения транзакций (их можно отслеживать с помощью callback-функций).

3. Работа с I^2C **по прерываниям по таймеру**. Помимо главного цикла, программа должна иметь еще один поток управления, который отвечает за опрос клавиатуры. Для этого

необходимо настроить один из таймеров микроконтроллера так, чтобы он регулярно генерировал прерывания в режиме автоперезагрузки. В обработчике прерывания от данного таймера необходимо генерировать посылки для **обмена по шине I^2C в режиме прерываний**. Таким образом, полный опрос клавиатуры требует восемь прерываний от таймера. Период между прерываниями должен обеспечивать достаточный запас времени на выполнение транзакций I^2C и работу основного цикла. При этом регистрация нажатия должна происходить достаточно быстро, чтобы пользователь не чувствовал задержки отклика. Оценить время, необходимое на выполнение транзакции, достаточно просто, так как известно время передачи одного бита и количество данных. При частоте шины I^2C в 400 кГц один бит передается за 2,5 мкс, а транзакция в 4 байта занимает не более 0,1 мс. Начальная инициализация устройств I^2C (если требуется) может быть сделана до того, как будет запущен процесс опроса клавиатуры. При таком способе опроса зафиксированные события нажатия кнопок помещаются в программный FIFO-буфер, из которого их может считывать процесс (поток управления), реализуемый главным циклом функции main(); доступ к буферу должен быть защищен от состояния гонки в моменты модификации указателей критическими секциями с запретами прерываний (см. соответствующий раздел).

1.8.Дисплейный модуль

Общие сведения о дисплейном модуле

Дисплейный модуль WEO012865D имеет диагональ 0,96 дюйма и разрешение 128×64 точек. За работу модуля отвечает встроенный в него контроллер дисплея – микросхема SSD1306BZ [10], обменивающаяся данными с микропроцессором через интерфейс I^2C.

Контроллер дисплея имеет встроенную видеопамять GDDRAM, которая реализована как статическое ОЗУ. Размер памяти составляет 128×64 бита (8096 бит или 1024 байта). Каждый бит памяти кодирует состояние одного пиксела монохромного дисплея. Память разделена на восемь страниц, от PAGE0 до PAGE7 (рис. 25). Одна страница содержит информацию для 8 строк дисплея. При доступе к памяти байты располагаются «вертикально», т.е. каждый байт кодирует 8 пикселов одного столбца: в нормальном режиме младший бит (D0) соответствует нулевой строке, а старший (D7) – седьмой строке (рис. 26).

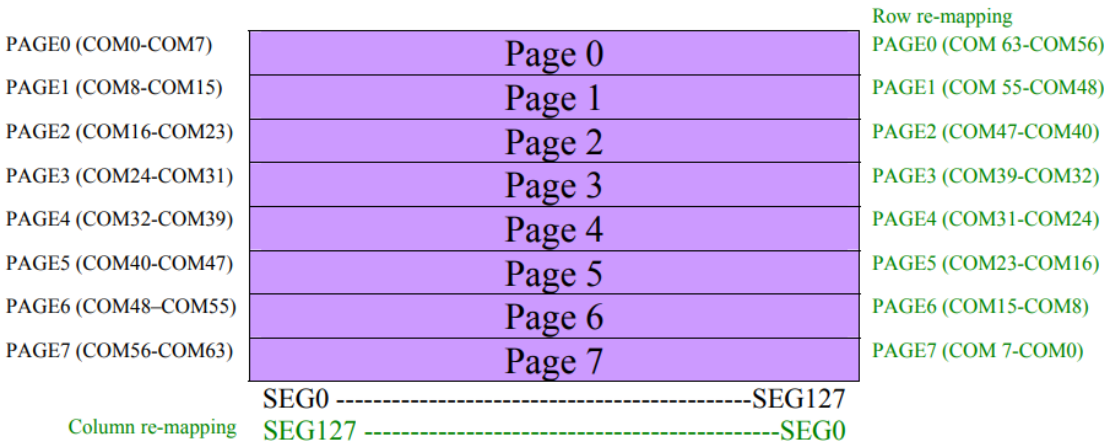


Рисунок 25 – Структура видеопамяти GDDRAM [10]

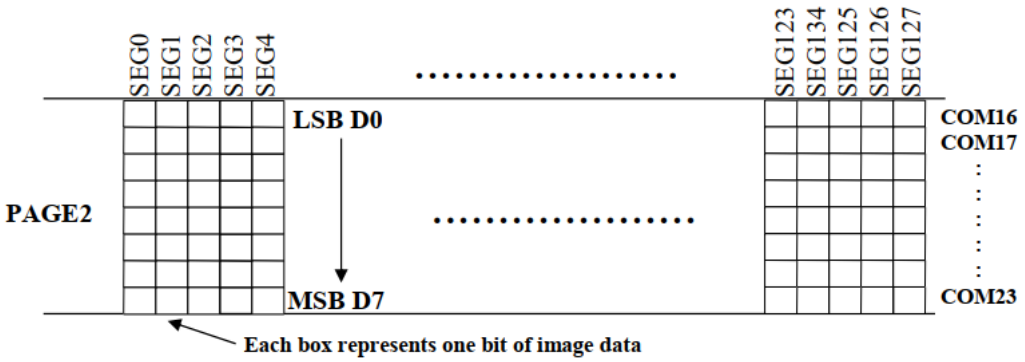


Рисунок 26 – Соответствие точек дисплея данным в видеопамяти [10]

Видеопамять имеет несколько режимов адресации: страничный, горизонтальный, вертикальный. Далее будем рассматривать работу с памятью на примере горизонтального режима. Порядок адресации данных в горизонтальном режиме «слева направо, сверху вниз» (рис. 27).

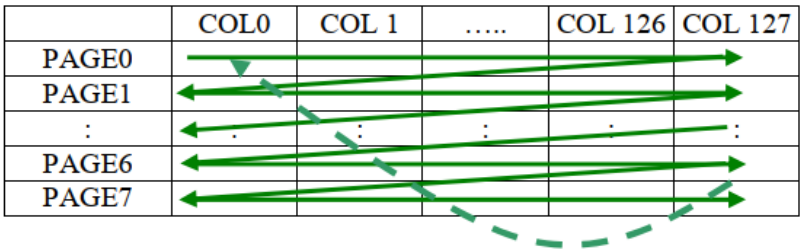


Рисунок 27 – Адресация видеопамяти в горизонтальном режиме

Адрес контроллера дисплея на шине I^2C в стенде SDK-1.1M – 0x78. Формат транзакции передачи данных по I^2C показан на рис. 28.

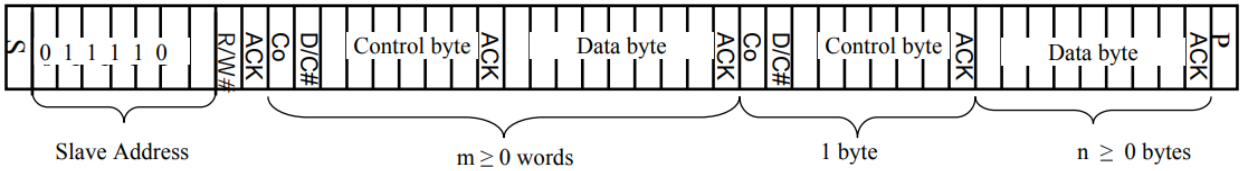


Рисунок 28 – Формат записи данных в дисплейный модуль по I^2C

Обозначения:

- S – стартовая последовательность;
- Co – бит «продолжения» (continuation);
- D/C# – бит выбора данные (1)/команда (0);
- ACK – бит подтверждения;
- SA0 – бит адреса, задаваемый схемотехнически;
- R/W# – бит выбора режима чтение/запись;
- P – стоповая последовательность.

После адреса ведомого устройства (slave address) передается одно или больше «слов» (words), где каждое слово состоит из управляющего байта (control byte) и байтов данных (data byte). Два старших бита управляющего байта – это биты Co и D/C#, а остальные биты должны быть равны нулю. Бит Co показывает, является ли это слово последним в посылке; если Co = 1, то далее будет передано еще как минимум одно слово, иначе это последнее слово. Бит D/C# показывает, как воспринимать байты данных, идущие за управляющим байтом. Если D/C# = 1, то это данные для записи в видеопамять, иначе это команды. Одно слово может содержать несколько идущих подряд команд.

Команда состоит из одного или нескольких байтов данных. Первый байт – это код операции, от него зависит, сколько за ним следует байтов параметров команды. Полное описание команд и их параметров приводится в документации микросхемы SSD1306BZ [10].

Инициализация дисплейного модуля

Перед тем, как начать работу с дисплейным модулем, необходимо выполнить его аппаратный сброс. Это делается с помощью младшего сигнала внешнего расширителя ввода-вывода PCA9538 с адресом 0xE0. Вначале необходимо установить сигнал в «1», а затем сформировать импульс низкого уровня длительностью не менее 1 мкс. После импульса необходимо сделать паузу примерно в 1 мс. После этого можно проводить инициализацию контроллера. В табл. 3 приведен один из вариантов последовательности команд, которые необходимо записать в контроллер дисплея, чтобы провести его начальную инициализацию. Данные команды можно отправлять в единой транзакции I^2C в качестве данных «слова» с управляющим байтом 0x00.

Работа с видеопамятью

После инициализации дисплея в его видеопамять можно записывать данные, которые будут отображаться на пикселах.

В горизонтальном режиме адресации, чтобы сбросить указатели записи видеопамяти в начальное состояние, необходимо отправить две команды, приведенные в табл. 5. После этого можно записывать данные в видеопамять. Можно записать весь буфер как одно «слово» с управляющим байтом 0x40. Указатель записи инкрементируется автоматически, поэтому данные можно разбить на несколько слов или транзакций без необходимости установки адреса. После достижения конца видеопамяти указатель автоматически перейдет в ее начало (рис. 27).

Для более удобной работы с видеопамятью данные для записи заранее подготавливаются в программном буфере, а после этого передаются в контроллер дисплея. В зависимости от способа организации программы может быть необходима двойная или тройная буферизация.

Таблица 3 – Пример последовательности команд инициализации контроллера дисплея

Код операции	Параметры	Описание
0xAE	-	Выключение дисплея.
0x20	0x00	Режим адресации памяти: горизонтальный.
0xC8	-	Инверсия отображения сигналов COM (PAGE0: COM63 – COM56).
0x40	-	Начальная строка: 0.
0x81	0xFF	Контраст: 256.
0xA1	-	Инверсия отображения сигналов SEG (SEG0 = column 127).
0xA6	-	Нормальный цвет (не инвертированный).
0xA8	0x3F	Настройка мультиплексора COM (mux ratio = 64).
0xA4	-	Отображение содержимого видеопамяти.
0xD3	0x00	Вертикальный сдвиг: 0.
0xD5	0xF0	Настройка синхросигналов (DCLK divide ratio = 1, Fosc = 15).
0xD9	0x22	Настройка предзарядки (phase1 = 2, phase2 = 2).
0xDA	0x12	Настройка конфигурации сигналов COM.
0xDB	0x20	Настройка напряжения Vcomh = ~0.77 × Vcc.
0x8D	0x14	Включение встроенного источника напряжения (зарядового насоса).
0xAF	-	Включение дисплея.

Таблица 4

Таблица 5 – Команды сброса указателей записи видеопамяти дисплея

Код операции	Параметры	Описание
0x21	0x00, 0x7F	Номера используемых столбцов: 0 – 127, текущий = 0.
0x22	0x00, 0x07	Номера используемых страниц: 0 – 7, текущая = 0.

Для манипуляции с буфером видеопамяти разрабатываются графические примитивы, позволяющие что-либо рисовать, используя привычные понятия. Например:

- **SetCursor()** – установка курсора по заданным координатам;
- **DrawPixel()** – рисование пикселя белого или черного цвета по заданным координатам;
- **WriteChar()** – рисование символа;
- **WriteString()** – рисование строки символов;
- **Fill()** – заполнение буфера одним цветом (белым или черным).

Для вывода символов и строк необходимо предварительно создать библиотеку шрифтов, где каждый символ представлен в виде последовательности байтов. Пример см. на рис. 29. Символ «S» размера 7×10 может быть закодирован так: 0x38, 0x44, 0x40, 0x30, 0x08, 0x04, 0x44, 0x38, 0x00, 0x00. Существуют программы, составляющие необходимые библиотеки шрифтов автоматически [11].

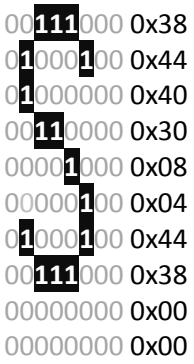


Рисунок 29 – Пример кодирования символа для вывода на дисплей

2. Открытое программное обеспечение встраиваемых систем

2.1. Операционная система реального времени FreeRTOS

Общие сведения о FreeRTOS

FreeRTOS – открытая многозадачная операционная система реального времени (ОС РВ, RTOS) для встраиваемых систем [12]. За переключение задач отвечает диспетчер, работающий по прерыванию от таймера. Задача выглядит как обычная функция на языке С, которая выполняет некоторые действия в бесконечном цикле.

Так как **задачи** выполняются асинхронно, то использование общих переменных для межпроцессного взаимодействия может приводить к конфликтам доступа к данным. Для обеспечения правильного обмена данными между задачами используются **очереди.** Несколько задач могут записывать данные в очередь, а одна задача их вычитывает. Диспетчер обрабатывает ситуации переполнения очереди и попыток чтения из пустой очереди. Если очередь пуста/переполнена, то та задача, которая собирается считать/записать данные, переводится в состояние WAIT, и диспетчер выведет ее из этого состояния только когда очередь будет готова принять/отдать данные.

Семафоры представляют собой механизм синхронизации задач. Семафоры бывают двух типов: бинарные и счетные. Работа **бинарного семафора** очень похожа на работу флага, который устанавливается по некоторому событию, при этом программа отслеживает его состояние в цикле. В FreeRTOS функцию такого цикла выполняет диспетчер. **Счетный семафор** работает по тому же принципу, разница лишь в том, что у него состояние – это не единичный флаг, а числовая переменная. Задача, которая «получает» семафор, эту переменную уменьшает, а которая «освобождает» – увеличивает. Также, FreeRTOS поддерживает **мьютексы**, которые очень похожи на бинарные семафоры, но имеют механизм наследования приоритетов.

Основные функции прикладного программного интерфейса (API) FreeRTOS перечислены в табл. 6.

FreeRTOS и STM32CubeMX

STM32CubeMX имеет встроенную поддержку FreeRTOS с использованием CMSIS-RTOS API. Конфигурация FreeRTOS производится в конфигураторе STM32CubeMX, в разделе Middleware. Чтобы использовать FreeRTOS, необходимо назначить один из таймеров микроконтроллера на генерацию системных тактов («тиков»). Также в этом разделе можно конфигурировать задачи, очереди, семафоры, таймеры и мьютексы.

Пример задачи, переключающей состояние светодиода каждые 500 мс:

```
void Start_ledTask(void const * argument)
{
  for(;;)
  {
        HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
        osDelay(500);
  }
}
```

Таблица 6 – Описание API FreeRTOS

Функции API	Описание
osThreadCreate()	Начать выполнение потока.
osThreadTerminate()	Закончить выполнение потока.
osThreadYield()	Передать выполнение другому потоку.
osThreadGetId()	Получить идентификатор потока для ссылки на него.
osThreadSetPriority()	Изменить приоритет выполнения потока.
osThreadGetPriority()	Получить текущий приоритет выполнения потока.
osDelay()	Подождать в течение указанного времени.
osWait()	Ожидать события типа Signal, Message, Mail.
osTimerCreate()	Определить атрибуты Callback таймера.
osTimerStart()	Запустить таймер с назначением времени.
osSignalSet()	Установить сигнальные флаги потока.
osSignalClear()	Сбросить сигнальные флаги потока.
osMutexCreate()	Инициализация мьютекса.
osMutexWait()	Получить мьютекс или подождать, пока он не станет доступным.
osMutexRelease()	Освободить мьютекс.
osMutexDelete()	Удалить мьютекс.
osSemaphoreCreate()	Инициализация семафора.
osSemaphoreWait()	Получить семафор или подождать, пока он не станет доступным.
osSemaphoreRelease()	Освободить семафор.
osSemaphoreDelete()	Удалить семафор.
osMessageCreate()	Инициализация очереди.

2.2. Сетевой стек LwIP

Общие сведения о LwIP

LwIP (Lightweight IP) – открытая легковесная реализация стека TCP/IP для встраиваемых систем [13]. Архитектура стека LwIP построена на модели, включающей четыре уровня абстракции (рис. 30):

1. Прикладной уровень – содержит все протоколы для передачи данных между процессами.
2. Транспортный уровень – обрабатывает соединения между хостами.
3. Интернет-уровень – соединяет независимые сети, обеспечивая межсетевое взаимодействие.
4. Уровень сетевого интерфейса – содержит технологии коммуникации для одного сегмента локальной сети.

Стек LwIP предлагает три типа API: Raw API, Netconn API, Socket API.

Поддержка стека LwIP встроена в STM32CubeMX, ее можно включить в разделе Middleware.

Raw API

Raw API – это «сырой» интерфейс LwIP (табл. 7, 8). Он позволяет использовать обратные вызовы функций (callback) внутри стека. Для этого перед началом работы со стеком необходимо инициализировать указатели на функции-обработчики событий, которые в процессе работы будут вызываться внутри LwIP.

Рисунок 30 – Уровни стека LwIP

Таблица 7 – Описание RAW API TCP

Функции API		**Описание**
Установка TCP-соединения	tcp_new()	Создает новый TCP PCB (блок управления протоколом).
	tcp_bind()	Привязывает TCP PCB к локальному IP-адресу и порту
	tcp_listen()	Начало процесса прослушивания.
	tcp_accept()	Назначает функцию обратного вызова, которая будет вызваться при получении нового соединения TCP.
	tcp_accepted()	Сообщает стеку LwIP, что входящее соединение TCP принято.
	tcp_connect()	Подключение к удаленному хосту TCP.
Отправка данных через TCP	tcp_write()	Записывает данные в очередь на отправку.
	tcp_sent()	Назначает функцию обратного вызова, которая будет вызваться при подтверждении данных удаленным хостом.
	tcp_output()	Принудительно отправляет данные в очередь.
Получение данных через TCP	tcp_recv()	Назначает функцию обратного вызова, которая будет вызываться при поступлении новых данных.
	tcp_recved()	Вызывается, когда приложение обработало входящий пакет данных.
Опрос приложений	tcp_poll()	Назначает функции обратного вызова, которые будут вызываться периодически. Используется приложением для проверки того, имеются ли оставшиеся данные приложения, которые необходимо отправить, или существуют ли соединения, которые необходимо закрыть.
Закрытие и прерывание соединения	tcp_close()	Закрывает TCP-соединение с удаленным хостом.
	tcp_err()	Назначает функцию обратного вызова для обработки соединений, прерванных LwIP из-за ошибок.
	tcp_abort()	Прерывает TCP-соединение.

Таблица 8 – Описание RAW API UDP

Функции API	Описание
udp_new()	Создает новый UDP PCB.
udp_remove()	Удаляет UDP PCB.
udp_bind()	Привязывает UDP PCB к локальному IP-адресу и порту.
udp_connect()	Устанавливает UDP PCB к удаленному IP-адресу и порту.
udp_disconnect()	Удаляет IP-адрес и порт UDP PCB.
udp_send()	Отправляет данные через UDP.
udp_recv()	Определяет функцию обратного вызова, которая вызывается при получении данных.

Netconn API

Netconn API – высокоуровневый последовательный интерфейс, разработанный поверх RAW API (табл. 9). Этот интерфейс требует наличия операционной системы реального времени и поддерживает многопоточные операции.

Таблица 9 – Описание Netconn API

Функции API	Описание
netconn_new()	Создает новое соединение.
netconn_delete()	Удаляет существующее соединение.
netconn_bind()	Привязывает соединение к локальному IP-адресу и порту.
netconn_connect()	Присоединяется у удаленному IP-адресу и порту.
netconn_send()	Отправляет данные по подключенному в данный момент удаленному IP-адресу или порту.
netconn_recv()	Принимает данные от netconn.
netconn_listen()	Устанавливает TCP-соединение в режим прослушивания.
netconn_accept()	Принимает входящее соединение в режиме прослушивания TCP-соединения.
netconn_write()	Отправляет данные в подключенный TCP netconn.
netconn_close()	Закрывает TCP-соединение, не удаляя его.

Socket API

Socket API – высокоуровневый интерфейс сокетов, разработанный поверх Netconn API (табл. 10). Этот интерфейс обеспечивает высокую переносимость написанных программ, потому что является стандартизированным API.

Таблица 10 – Описание Socket API

Функции API	Описание
soket()	Создает новый сокет.
bind()	Связывает сокет с IP-адресом и портом.
listen()	Прослушивает соединения сокетов.
connect()	Соединяет сокет с удаленным IP-адресом и портом.
accept()	Принимает новое соединение на сокете.
read()	Читает данные из сокета.
write()	Записывает данные в сокет.
close()	Закрывает сокет (с удалением).

2.3.Программная экосистема микропроцессоров линейки STM32MP15x

Общие сведения

Встраиваемые микропроцессоры (microprocessor unit, MPU) линейки STM32MP15x включают два процессорных ядра Cortex-A7 и одно процессорное ядро Cortex-M4 [14].

На ядрах Cortex-A7 обычно запускаются приложения с развитыми пользовательскими сервисами под управлением операционной системы (ОС) на базе ядра Linux. Эти приложения взаимодействуют с окружением через стандартные сетевые интерфейсы (Ethernet, Wi-Fi и т.п.), а также обеспечивают подключение стандартной компьютерной периферии (внешний жесткий диск, дисплей, камеры, клавиатура и т.п.).

Ядро Cortex-M обычно используется для реализации операций реального времени (работа с таймерами, управление контроллерными сетями и т.п.).

Разделение функций позволяет реализовывать подход асимметричной мультипроцессорности (Asymmetric Multi-Processing, AMP). Взаимодействие между ядрами выполняется через общую память или с помощью специальных аппаратных блоков с поддержкой буферизации данных.

Компоненты программного обеспечения

Компания STMicroelectronics предлагает комплект (так называемую «экосистему») стандартного программного обеспечения (ПО) и средств разработки для микропроцессоров SPM32MP15x. Далее описывается экосистема версии 3.0.

Рекомендованным дистрибутивом Linux для процессорных ядер Cortex-A7 является OpenSTLinux [15]. Также доступна установка Android и других дистрибутивов на базе ядра Linux.

Процессорное ядро Cortex-M4 программируется без операционной системы или с использованием операционной системы реального времени.

На рис. 31 показаны компоненты используемого ПО на базе OpenSTLinux. В него входят следующие основные компоненты:

- TF-A (Trusted Firmware A) [16] – загрузчик первого уровня (first stage bootloader, FSBL, также BL2), которому передает управление микропроцессор после выбора устройства для загрузки; поддерживает режим доверенной загрузки;
- OP-TEE (Open Portable Trusted Execution Environment) [17] – доверенная программная среда для выполнения различных приложений в безопасном режиме, реализует технологию ARM TrustZone; если не требуется, вместо нее может быть использована минимальная «заглушка» – TF-A SP-MIN; этот компонент также называют также BL32;
- U-Boot [18] – загрузчик второго уровня (second stage bootloader, SSBL, также BL33), который загружает ядро ОС Linux;
- Linux [19] – ядро ОС, которая используется для запуска основных пользовательских приложений.

Кроме этого, в микропроцессорах имеется встроенный неизменяемый загрузчик нулевого уровня (BL1), который загружает FSBL с одного из подключенных устройств памяти.

В качестве инструментального обеспечения используются:

– SDK – набор средств для сборки ПО ядер Cortex-A7 (в том числе пользовательских приложений);
– Yocto Project [20] – фреймворк для сборки дистрибутивов Linux;
– STM32CubeIDE [21] – среда разработки для STM32, в которой выполняется генерация деревьев устройств для компонентов ПО ядер Cortex-A7 и разработка ПО ядра Cortex-M4.

На рис. 32 показана последовательность загрузки компонентов ПО микропроцессора.

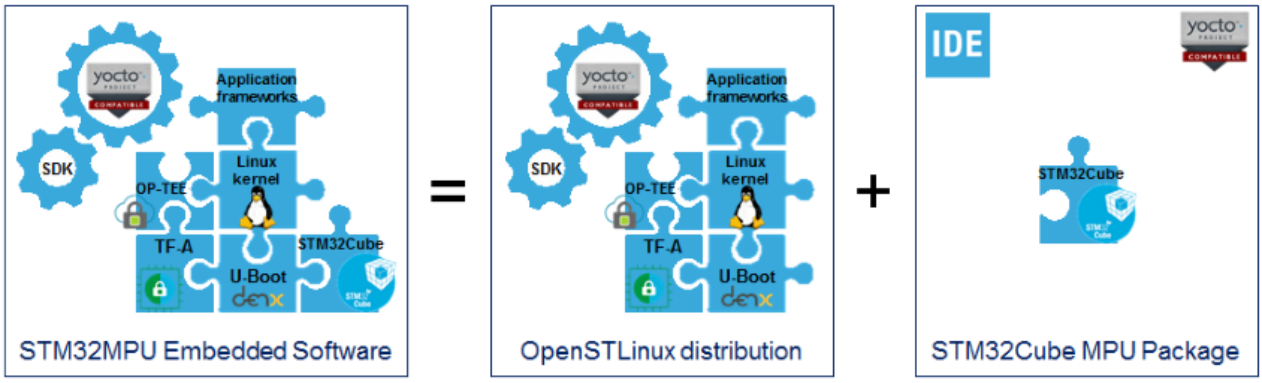


Рисунок 31 – Компоненты ПО микропроцессоров STM32MP15x [22]

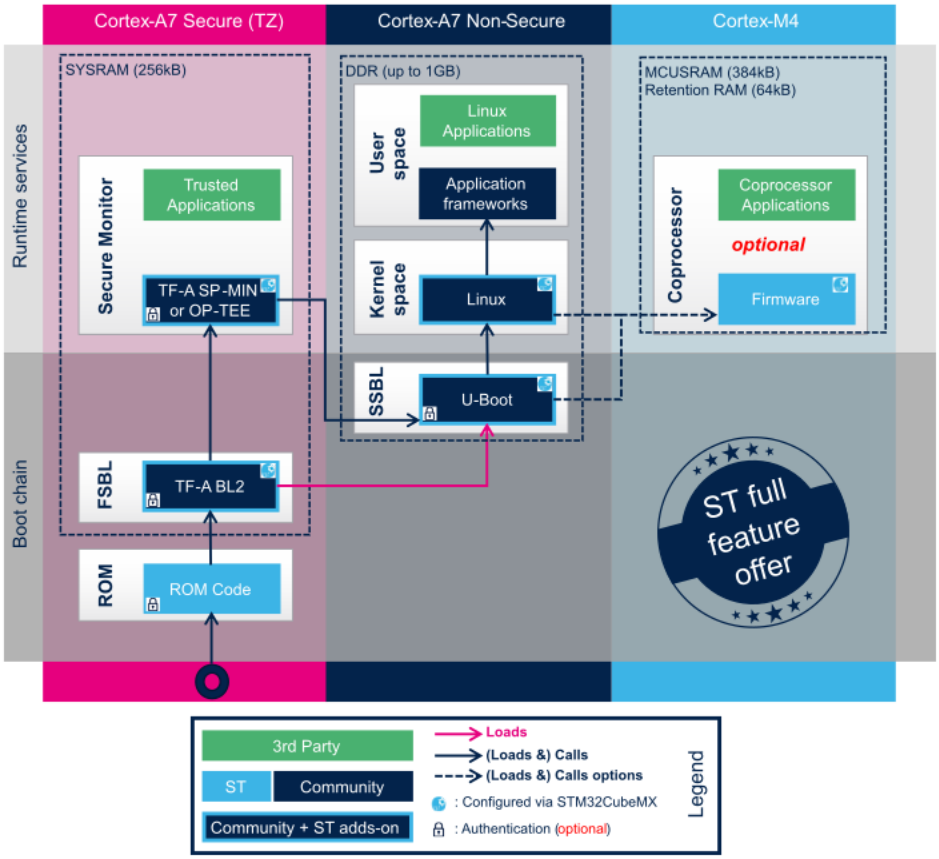
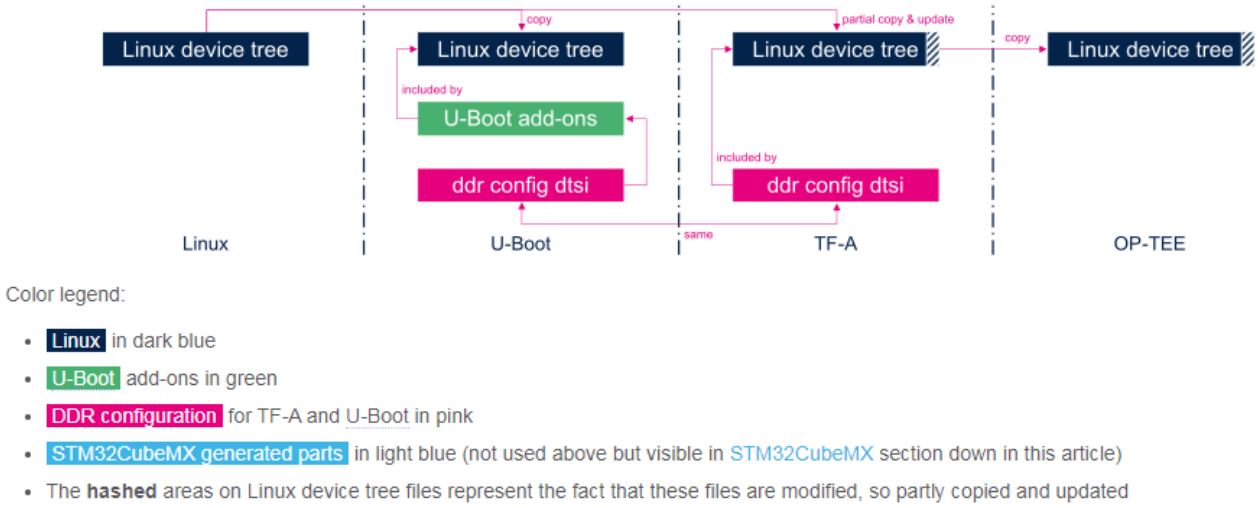


Рисунок 32 – Последовательность загрузки компонентов ПО STM32MP15x [23]

Программные компоненты ядер Cortex-A7 требуют специального конфигурационного файла – дерева устройств (DeviceTree) [24]. Он содержит информацию об аппаратной платформе, которую невозможно узнать путем автоматического опроса (enumeration). Также в дереве устройств указываются и пользовательские настройки аппаратуры, такие как параметры синхросигналов, режимы работы контроллеров ввода-вывода, параметры загрузки ядра ОС и т.п.

Данные файлы генерируются с помощью STM32CubeIDE на основе настроек проекта, и затем могут быть вручную дополнены пользовательскими параметрами. Основным в STM32CubeIDE является дерево для Linux, а остальные генерируются из него путем удаления и добавления фрагментов (см. рис. 33). Пользовательские параметры следует добавлять в дерево устройств ядра между комментариями вида `/* USER CODE BEGIN */` … `/* USER CODE END */`.



Color legend:

- Linux in dark blue
- U-Boot add-ons in green
- DDR configuration for TF-A and U-Boot in pink
- STM32CubeMX generated parts in light blue (not used above but visible in STM32CubeMX section down in this article)
- The **hashed** areas on Linux device tree files represent the fact that these files are modified, so partly copied and updated

Рисунок 33 – Последовательность генерации деревьев устройств [25]

Загрузка программного обеспечения

Микропроцессоры STM32MP15x могут загружать и исполнять программы с нескольких видов внешних устройств памяти и интерфейсов:

- получение образов ПО через USB или UART;
- Serial NOR Flash;
- микросхема e•MMC;
- NAND Flash;
- SD-карта;
- Serial NAND Flash.

Выбор устройства осуществляется с помощью подтяжки конфигурационных входов в момент перезагрузки или с помощью битов однократного программирования. Можно задать основной и резервный источник загрузки. После загрузки FSBL источник данных для дальнейшей загрузки определяется настройками FSBL. FSBL и SSBL могут располагаться в разных устройствах памяти или загружаться с разных интерфейсов. Далее будем рассматривать ситуацию, когда все программные компоненты находятся на одном устройстве памяти. В таком случае устройство внешней памяти содержит все необходимые загрузчики и образы нескольких разделов файловой системы Linux.

Генерация файлов программного обеспечения

Для запуска ОС Linux необходимо сгенерировать три основных компонента:

- FSBL – файл с расширением *.stm32, включает TF-A;
- FIP (Firmware Image Package) – файл с расширением *.bin, включающий U-Boot и OP-TEE или SP-MIN, а также их деревья устройств и другие данные загрузчика;
- образы разделов файловой системы ОС Linux.

В OpenSTLinux используется четыре раздела файловой системы Linux:

- bootfs – ядро Linux, дерево устройств, конфигурация загрузчика;
- rootfs – основные каталоги, файлы и приложения Linux;
- userfs – пользовательские программы;
- vendorfs – проприетарные программы.

Генерировать все перечисленные файлы и образы «с нуля» не обязательно. Часто можно использовать скомпилированные ранее варианты, заменяя настройки или модифицируя их содержимое. В связи с этим в OpenSTLinux предлагается три комплекта файлов и инструментальных средств разработчика, позволяющих с необходимой глубиной модифицировать состав файлов ОС Linux [26]:

1. Starter Package – комплект скомпилированных файлов, готовых к записи в память и к запуску на микропроцессоре. Должен быть генерирован под необходимую модель микропроцессора и плату. С сайта производителя можно скачать Starter Package для стандартных отладочных плат. Для собственных плат его необходимо сгенерировать самостоятельно. В Starter Package невозможно модифицировать загрузчики, а образы файловых систем можно модифицировать только «вручную» путем их распаковки, изменения содержимого и повторной запаковки.

2. Developer Package – комплекты исходных текстов всех загрузчиков и ядра Linux, для которых можно настроить необходимые параметры сборки с помощью стандартных инструментов типа menuconfig и ключей компилятора, а также адаптировать эти компоненты под конкретную плату с помощью деревьев устройств. Все эти компоненты компилируются при помощи SDK. Также Developer Package позволяет компилировать собственные приложения для Linux. Тем не менее, модификация образов файловой системы (в том числе замена дерева устройств ядра, самого ядра и его модулей) по-прежнему выполняется вручную.

3. Distribution Package – максимальный комплект, который позволяет автоматически создавать необходимые образы файловой системы, выбрав требуемые программные пакеты. Для сборки дистрибутива используется фреймворк Yocto Project. Чтобы выполнять настройки дистрибутива необходимо владение технологией Yocto и менеджером сборки BitBake. Полная сборка стандартного дистрибутива OpenSTLinux занимает значительное время – не менее нескольких часов на среднем компьютере или сервере, а также требует не менее 100–150 Гбайт свободного места на жестком диске. Существует два основных варианта стандартного дистрибутива: openstlinux-weston (с поддержкой графического пользовательского интерфейса) и openstlinux-core (без такой поддержки). В большинстве случаев достаточно использовать один из данных дистрибутивов, дополнив их необходимым набором стандартных пакетов и пользовательского ПО.

Связь между перечисленными пакетами разработчиков показана на рис. 34.

Для запуска ОС Linux на новой плате «с нуля» необходимо:

- создать в STM32CubeIDE проект и задать конфигурацию микропроцессора (дерево синхронизации, настройки периферийных блоков);
- сгенерировать деревья устройств;
- установить Starter Package с сайта производителя, чтобы получить стандартные образы разделов файловой системы Linux;
- установить Developer Package и собрать загрузчики, деревья устройств и ядро Linux под новую плату;

– заменить образы ядра и дерева устройств Linux в стандартном образе bootfs на собственные;
– записать FSBL, FIP и разделы файловой системы Linux в память, из которой микропроцессор сможет их загрузить.

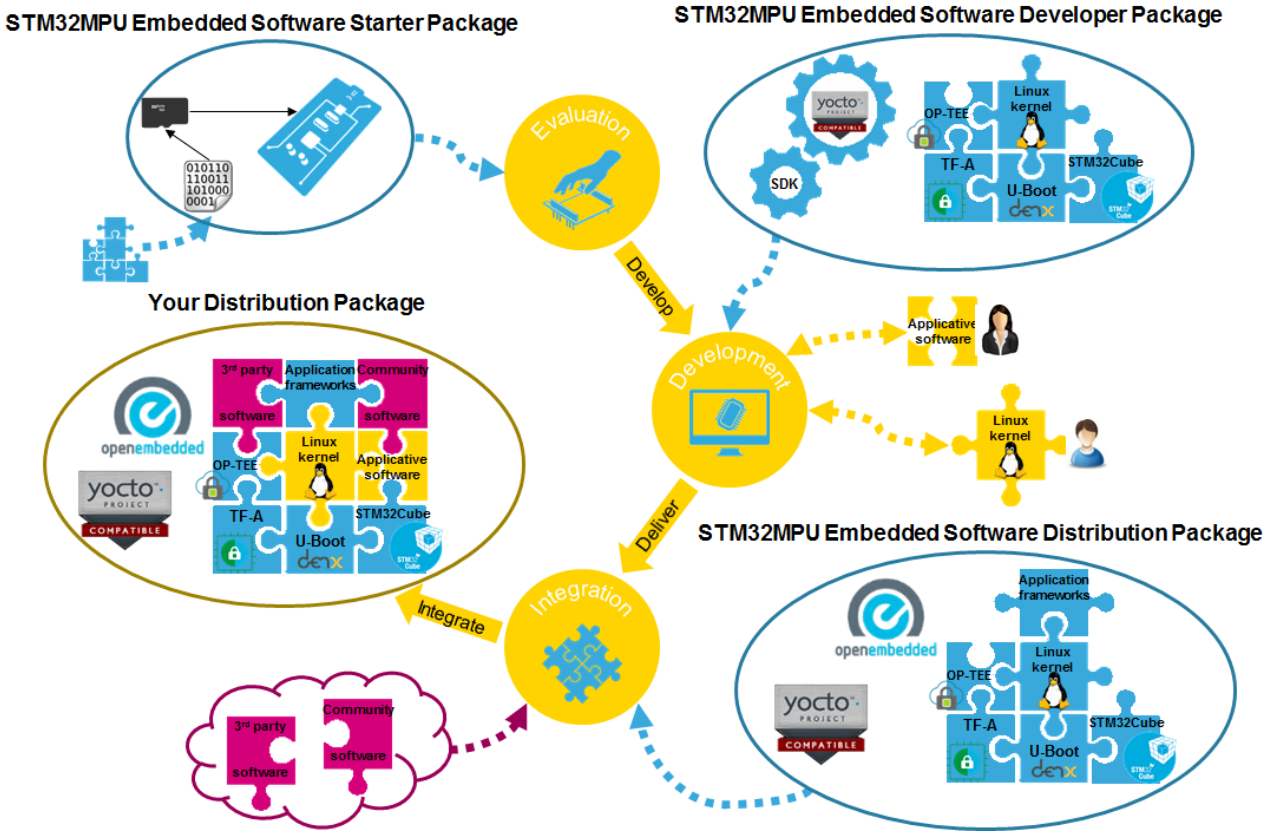


Рисунок 34 – Комплекты разработчика OpenSTLinux [26]

Использование ОС Linux

После загрузки ОС можно подключиться к ее консоли через COM-порт, назначенный в списке параметров загрузки ядра, для первичной настройки: подключения к сети и т.п.

Основным сценарием использования является подключение с консоли Linux через сетевой интерфейс по протоколу SSH [27].

При наличии подключения к сети можно устанавливать пакеты приложений из сетевых репозиториев с помощью менеджера пакетов apt [28]. Вначале необходимо добавить репозиторий в настройки apt:

```
# echo "deb http://packages.openstlinux.st.com/3.0 dunfell main" >>/etc/apt/sources.list.d/my_custom.list
```

Затем необходимо обновить список пакетов:

```
# apt-get update
```

Теперь можно устанавливать пакеты. Например, установить средства разработки и исполнения программ java:

```
# apt-get install openjdk-8
```

Для того, чтобы новые файлы были сохранены на файловой системе, необходимо выполнять безопасное завершение работы Linux:

```
# shutdown now
```

Особенности запуска ОС Linux на процессорном модуле Type 153

Запись программ в память

В процессорном модуле Type 153 для стенда SDK-1.1M доступны для загрузки программ два устройства памяти: SD-карта и NAND Flash. Устройством, с которого начинается поиск программ для исполнения, по умолчанию является SD-карта.

Запись образов на SD-карту можно выполнить с помощью стандартного инструментария ОС инструментального компьютера и компонентов Starter Package.

Запись образов в NAND Flash осуществляется с помощью программы STM32CubeProgrammer, файлов описания памяти и вспомогательных образов загрузчиков из Starter Package. Микропроцессор необходимо предварительно перевести в режим программирования. Для этого требуется замкнуть перемычку процессорного модуля на боковой панели стенда и перезагрузить стенд. Аналогичным способом можно записать образы на SD-карту, установленную в процессорный модуль.

Если в качестве памяти используется SD-карта, замену файлов в разделах файловой системы можно производить прямо на ней, подключив SD-карту к компьютеру.

Примеры модификации дерева устройств

Для того, чтобы сделать зеленый светодиод (подключен к PA9) мигающим индикатором работы системы, необходимо добавить в корневой раздел следующий код:

```
led{
      compatible = "gpio-leds";
      green{
            label = "heartbeat";
            gpios = <&gpioa 9 GPIO_ACTIVE_HIGH>;
            linux,default-trigger = "heartbeat";
            default-state = "off";
      };
};
```

Для настройки микросхемы физического уровня Ethernet в режиме RMII для работы от опорной синхрочастоты 50 МГц необходимо настроить выходной синхросигнал в микропроцессоре и добавить в пользовательскую секцию раздела ethernet0 следующий код [29]:

```
phy-mode = "rmii";
max-speed = <100>;
phy-handle = <&phy0>;
st,eth-ref-clk-sel;
local-mac-address = [00 00 00 00 00 00];
mdio0 {
            #address-cells = <1>;
            #size-cells = <0>;
            compatible = "snps,dwmac-mdio";
            phy0: ethernet-phy@1 {
                        reg = <1>;
                        clocks = <&rcc ETHCK_K>;
                        clock-names = "rmii-ref";
            };
};
```

Здесь [00 00 00 00 00 00] – MAC-адрес, который будет назначен по умолчанию.

3. Организация стенда SDK-1.1M

3.1. Общий вид стенда SDK-1.1M

Стенд-конструктор состоит из двух основных частей:

- процессорного модуля, оснащенного центральным вычислителем (микроконтроллером, микропроцессором и др.);
- несущей платы, на которой установлены внешние устройства и слоты расширения.

Стенд представляет собой печатный узел, установленный в корпус. Основные элементы пользовательского интерфейса и разъемы расположены на лицевой стороне стенда, защищенной прозрачной панелью.

На рис. 35 представлено схематическое изображение лицевой панели стенда SDK-1.1M. На рис. 36 показано расположение элементов боковой панели. В табл. 11 дана расшифровка обозначений элементов на панелях стенда.

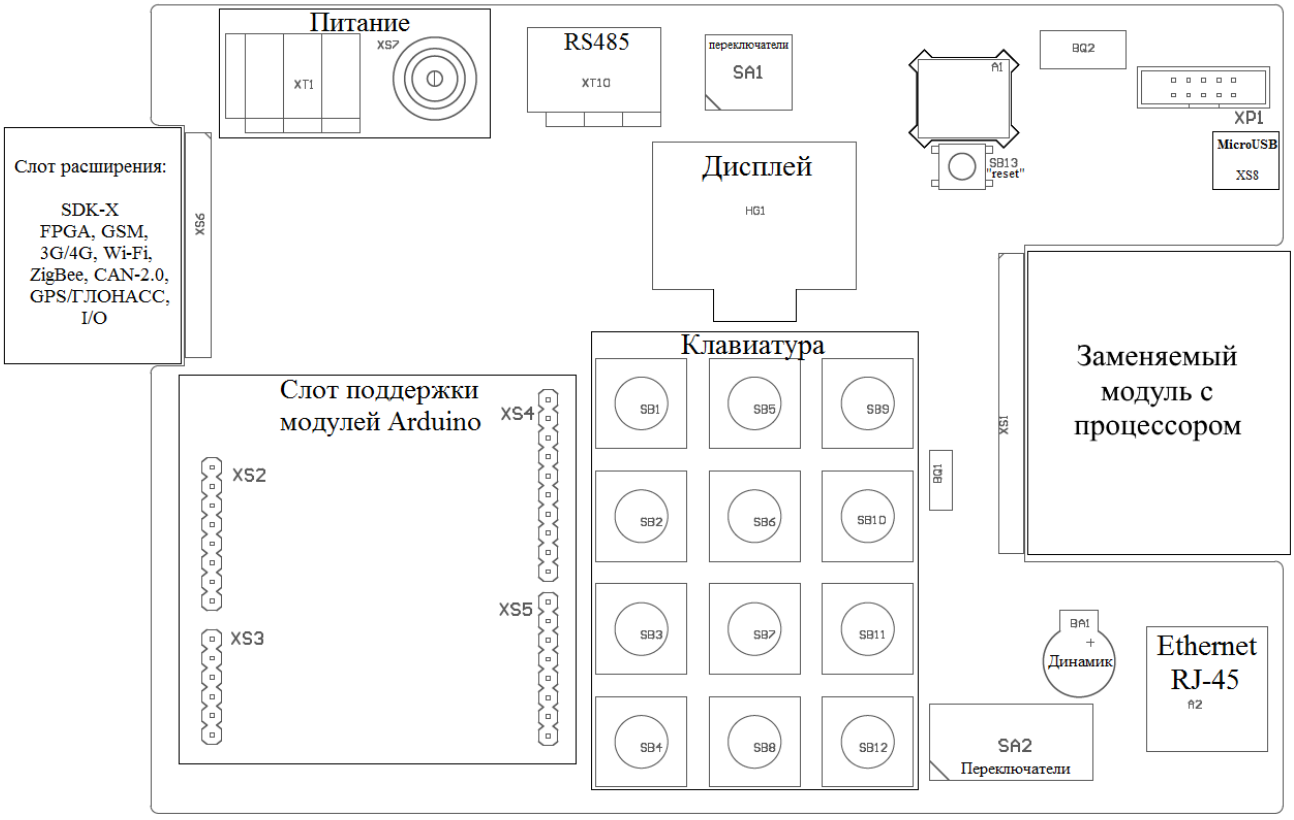


Рисунок 35 – Общий вид лицевой панели SDK-1.1M

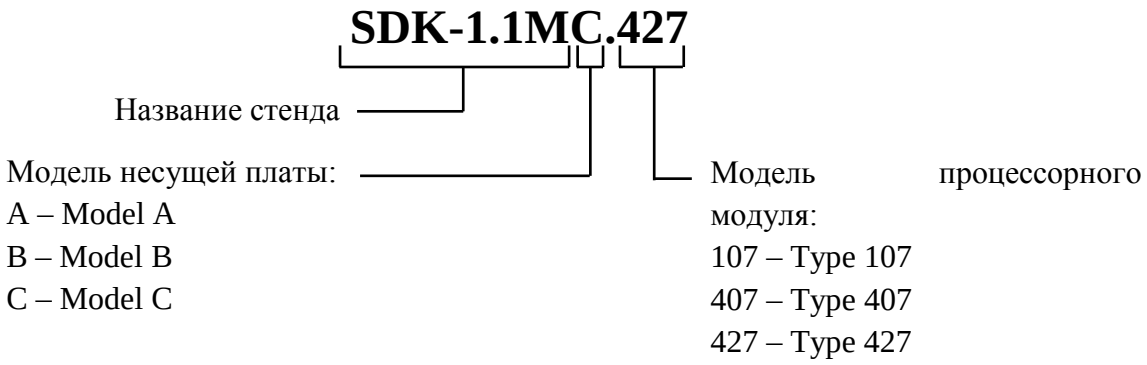
Рисунок 36 – Общий вид боковой панели SDK-1.1M

Таблица 11 – Элементы лицевой и боковой панелей стенда SDK-1.1M

Элементы	Описание
Разъемы питания XT1, XS7	Вход питания 9–36 В. Полярность подключения не имеет значения.
Кнопка «RESET» SB13	Кнопка перезагрузки процессорного модуля.
XS1	Слот подключения сменных процессорных модулей.
XS6	Слот подключения модулей расширения SDK-X.
XS2, …, XS5	Слот подключения модулей расширения Arduino.
Разъем «DBG USB» XS8	Разъем встроенного программатора-отладчика SDK-1.1M. Используется для питания, программирования и отладки стенда. Тип USB Micro-B.
Разъем RJ-45 A2	Разъем RJ-45 для подключения кабеля Ethernet 10/100BASE-TX.
Разъем «RS485» XT10	Разъем для подключения к сети RS-485.
Переключатели SA1, SA2	Конфигурационные переключатели.
Дисплей HQ1	Графический OLED-дисплей WEO012864DL.
Кнопки клавиатуры SB1, …, SB12	Клавиатура.
Динамик BA1	Излучатель звука HC0903A.

3.2.Комплектация SDK-1.1M

Расшифровка обозначения модели стенда SDK-1.1M:



В табл. 12, 13 представлено сравнение характеристик моделей процессорных модулей, несущих плат SDK-1.1M.

Таблица 12 – Характеристики процессорных модулей SDK-1.1M

Характеристика	**Type 107**	**Type 407**	**Type 427**	**Type 153**
Вычислитель				
Модель	STM32F107 VCT6	STM32F407 VGT6	STM32F427 VIT6	STM32MP153 CAB3
Тактовая частота ядра	72 МГц	168 МГц	180 МГц	до 800 МГц
Встроенная RAM	64 КБ	192 КБ	256 КБ	708 КБ
Внешняя RAM	-	-	-	512 МБ
Встроенная FLASH-память	256 КБ	1 МБ	2 МБ	-
Внешняя FLASH-память	-	16 МБ	16 МБ	1 ГБ
Внешние интерфейсы				
Micro SD	-	да	да	да
USB	да	да	да	да

Таблица 13 – Характеристики несущих плат SDK-1.1M

Характеристика	**Model A**	**Model B**	**Model C**
Внешние интерфейсы и периферийные устройства			
Ethernet	-	-	10/100 Мбит/с
RS-485	-	да	да
OLED-дисплей 128×64 точек	да	да	да
Клавиатура 3×4	да	да	да
Подключение модулей SDK-X	-	да	да
Подключение Arduino-совместимых модулей	-	-	да
Питание от отладочного USB-порта («DBG USB»)			
Напряжение	5 В ± 0,5 В		
Ток	до 500 мА		
Питание от внешнего источника постоянного тока			
Напряжение	9÷36 В		
Ток	до 750 мА		

3.3.Вычислители процессорных модулей SDK-1.1M

В данном разделе перечислены основные возможности и характеристики вычислителей, установленных в разных моделях процессорных модулей SDK-1.1M.

Микроконтроллер STM32F107VCT6 (Type 107)

- Процессорное ядро ARM 32-bit Cortex-M3: частота 72 МГц; однотактное умножение и аппаратное деление.
- Память:
– 256 Кбайт FLASH-памяти;
– 64 Кбайт SRAM.
- Управление синхронизацией, сбросом, питанием:
– напряжение питания и ввода-вывода от 2 до 3,6 В;
– сброс при включении (POR), выключении (PDR) питания, программируемый датчик напряжения (PVD);
– внешний кварцевый генератор от 3 до 25 МГц;
– внешний генератор 32 кГц для часов реального времени (RTC) с калибровкой;
– встроенные RC-генераторы 8 МГц и 40 кГц.
- Управление энергосбережением:
– режимы сна, остановки, дежурный режим;
– питание от батареи для RTC и резервной памяти.
- 2 12-битных АЦП (16 каналов):
– диапазон от 0 до 3,6 В;
– датчик температуры;
– до 2 млн. выборок/с в чередующемся режиме.
- 2 12-битных ЦАП.
- Универсальный 12-канальный контроллер прямого доступа к памяти (DMA):
– поддерживаемые устройства: таймеры, АЦП, ЦАП, I^2C, SPI, I^2S и USART.
- Отладочные возможности: интерфейсы JTAG и SWD (serial wire debug); Cortex-M3 Embedded Trace Macrocell.
- 80 быстрых портов ввода-вывода.
- Блок вычисления CRC, уникальный 96-битный идентификатор.
- 10 таймеров с возможностью переназначения контактов.
- 14 коммуникационных интерфейсов с возможностью переназначения контактов:
– 2 интерфейса I^2C (SMBus/PMBus);
– 5 USART;
– 3 SPI (18 Мбит/с);
– 2 интерфейса CAN с 512 байт выделенной SRAM;
– контроллер USB 2.0 Full Speed device/host/OTG с PHY;
– 10/100 Ethernet MAC с выделенной DMA и SRAM (4 Кб), IEEE 1588.

Микроконтроллер STM32F407VGT6 (Type 407)

- Процессорное ядро ARM 32-bit Cortex-M4: модуль операций с плавающей запятой; частота до 168 МГц; блок защиты памяти (MPU).
- Память:
– 1 Мбайт FLASH-памяти для программ и данных;
– 192+4 Кбайт SRAM-памяти, включая 64 Кбайт CCM (core coupled memory);
– контроллер внешней памяти, поддерживающий Compact Flash, SRAM, PSRAM, NOR и NAND.
- Параллельный интерфейс LCD-дисплея с режимами 8080 и 6800.
- Управление синхронизацией, сбросом, питанием:
– напряжение питания и ввода-вывода от 1,8 до 3,6 В;
– сброс при включении (POR), выключении (PDR) питания;
– внешний кварцевый генератор от 4 до 26 МГц;
– внешний генератор 32 кГц для часов реального времени (RTC) с калибровкой;
– встроенные RC-генераторы 16 МГц и 32 кГц;
– режимы работы с пониженным энергопотреблением.
- 3 12-битных АЦП с 2,4 млн выборок/с: до 24 каналов и 7,2 млн выборок/с в режиме тройного чередования.
- 2 12-битных ЦАП.
- Универсальный 16-канальный контроллер прямого доступа к памяти (DMA) с FIFO-буферизацией и поддержкой пакетной передачи.
- Отладочные возможности: интерфейсы JTAG и SWD (serial wire debug); Cortex-M4 Embedded Trace Macrocell.
- До 140 портов ввода-вывода с поддержкой прерываний.
- Стандартные интерфейсы:
– I^2C (SMBus/PMBus);
– USART/UART;
– SPI (до 42 Мбит/с);
– I^2S (полнодуплексный);
– 2 интерфейса CAN с 512 байт выделенной SRAM;
– SDIO.
- Высокопроизводительные интерфейсы:
– контроллер USB 2.0 Full Speed device/host/OTG с PHY;
– контроллер USB 2.0 High Speed/Full Speed device/host/OTG с выделенной DMA, Full Speed PHY и ULPI;
– 10/100 Ethernet MAC с выделенной DMA и SRAM (4 Кбайт), IEEE 1588v2.
- 8–14-битный параллельный интерфейс камеры со скоростью до 54 Мбайт/с.
- Аппаратный генератор случайных чисел, блок вычисления CRC, 96-битный уникальный идентификатор.

Микроконтроллер STM32F427VIT6 (Type 427)

- Процессорное ядро ARM 32-bit Cortex-M4: модуль операций с плавающей запятой; ART-ускоритель, обеспечивающий мгновенное выполнение из FLASH-памяти; частота до 180 МГц; блок защиты памяти (MPU).
- Память:
– 2 Мбайта FLASH-памяти;
– 256+4 Кбайт SRAM, включая 64 Кбайт CCM (core coupled memory);
– контроллер внешней памяти, поддерживающий Compact Flash, SRAM, PSRAM, SDRAM/LPSDR SDRAM, NOR и NAND.
- Параллельный интерфейс LCD-дисплея с режимами 8080 и 6800.
- Ускоритель Chrom-ART, расширяющий функции работы с графикой (DMA2D).
- Управление синхронизацией, сбросом, питанием:
– напряжение питания и ввода-вывода от 1,7 до 3,6 В;
– сброс при включении (POR), выключении (PDR) питания, программируемый датчик напряжения (PVD);
– внешний кварцевый генератор с частотой от 4 до 26 МГц;
– внешний генератор 32 кГц для часов реального времени (RTC) с калибровкой;
– встроенные RC-генераторы 16 МГц и 32 кГц;
– режимы сна, остановки, дежурный режим.
- 3 12-битных АЦП с 2,4 млн выборок/с: до 24 каналов и 7,2 млн выборок/с в режиме тройного чередования.
- 2 12-битных ЦАП.
- Универсальный 16-канальный контроллер прямого доступа к памяти (DMA) с FIFO-буферизацией и поддержкой пакетной передачи.
- Отладочные возможности: интерфейсы JTAG и SWD (serial wire debug); Cortex-M3 Embedded Trace Macrocell.
- До 168 портов ввода-вывода с поддержкой прерываний.
- До 21 интерфейса связи:
– 3 интерфейса I^2C (SMBus/PMBus);
– 4 USART и 4 UART;
– 6 SPI (45 Мбит/с), 2 с полнодуплексным I^2S;
– SAI (serial audio interface);
– 2 интерфейса CAN с 512 байт выделенной SRAM;
– SDIO.
- Высокопроизводительные интерфейсы:
– контроллер USB 2.0 Full Speed device/host/OTG с PHY;
– контроллер USB 2.0 High Speed/Full Speed device/host/OTG с выделенной DMA, Full Speed PHY и ULPI;
– 10/100 Ethernet MAC с выделенной DMA и SRAM (4 Кб), IEEE 1588v2.
- 8–14-битный параллельный интерфейс камеры со скоростью до 54 Мбайт/с.
- Аппаратный генератор случайных чисел, блок вычисления CRC, 96-битный уникальный идентификатор.

Микропроцессор STM32F153CAB3 (Type 153)

- Процессорные ядра:
– 32-битный двухъядерный Arm Cortex-A7 с частотой до 800 МГц:
    - кэш L1 32 Кбайта для программ/32 Кбайта для данных на каждое ядро;
    - общий кэш L2 256 Кбайт;
    - Arm NEON и Arm TrustZone;
– 32-битный Arm Cortex-M4 с FPU/MPU с частотой до 209 МГц.
- Память:
– внешняя DDR-память до 1 Гбайта;
– 708 Кбайт внутренней SRAM (256+384+64+4 Кбайт);
– двухрежимный интерфейс памяти Quad-SPI;
– контроллер внешней памяти с шиной данных до 16 бит: параллельный интерфейс для подключения внешних микросхем и SLC NAND с ECC до 8 бит.
- Безопасность:
– доверенная загрузка, TrustZone;
– изоляция ресурсов ядра Cortex-M4.
- Управление сбросом и питанием:
– питание ввода-вывода от 1,71 до 3,6 В (входы устойчивы к 5 В);
– преобразователи питания на кристалле: RETRAM, BKPSRAM, USB 1,8 В, 1,1 В;
– режимы пониженного энергопотребления: сон, остановка, дежурный;
– сохранение содержимого DDR в дежурном режиме.
- Управление синхронизацией:
– внешние источники: HSE 8-48 МГц, LSE 32,768 кГц;
– внутренние источники: HSI 64 МГц, CSI 4 МГц, LSI 32 кГц;
– 5 блоков подстройки частоты PLL.
- До 176 портов ввода-вывода с поддержкой прерываний.
- 2 шинные матрицы:
– 64-битная Arm AMBA AXI до 266 МГц;
– 32-битная Arm AMBA AHB до 209 МГц.
- 3 контроллера прямого доступа к памяти (DMA):
– 48 физических каналов;
– 1 высокоскоростной контроллер;
– 2 двухпортовых контроллера с FIFO для управления периферией.
- До 37 интерфейсов ввода-вывода:
– 6 I^2C FastMode+ (1 Мбит/с, SMBus/PMBus);
– 4 UART + 4 USART (12,5 Мбит/с, ISO7816, LIN, IrDA, SPI slave);
– 6 SPI (50 Мбит/с, включая 3 полнодуплексных I^2S);
– 4 SAI (стерео аудио: I^2S, PDM, SPDIF Tx);
– SPDIF Rx с 4 входными каналами;
– HDMI-CEC;
– MDIO Slave;
– 3 SDMMC до 8 бит (SD/e•MMC/SDIO);
– 2 CAN с поддержкой CAN FD, из них один поддерживает TTCAN;
– 2 USB 2.0 high-speed Host + 1 USB 2.0 full-speed OTG одновременно:

    - или 1 USB 2.0 high-speed Host + 1 USB 2.0 high-speed OTG одновременно;
- 10/100M или Gigabit Ethernet GMAC:
    - IEEE 1588v2, MII/RMII/GMII/RGMII;
- интерфейс камеры от 8 до 14 бит со скоростью до 140 Мбайт/с.
- 6 аналоговых блоков:
- 2 АЦП с разрешением до 16 бит на скорости 3,6 млн. выборок/с;
- температурный датчик;
- 2 12-битных ЦАП (1 МГц);
- цифровые фильтры.
- Графика:
- контроллер LCD-TFT, до 24 бит (RGB888):
    - разрешение до WXGA (1366 × 768) @60 fps;
    - или до Full HD (1920 × 1080) @30 fps.
- До 29 таймеров и 3 сторожевых таймеров:
- 2 32-битных таймера, имеющих до 4 каналов ввода-вывода;
- 2 16-битных таймера для управления двигателями;
- 10 16-битных таймера общего назначения;
- 5 16-битных таймера с низким энергопотреблением;
- часы реального времени;
- 2 комплекта по 4 системных таймера Cortex-A7 (secure, nonsecure, virtual, hypervisor);
- 1 системный таймер Cortex-M4 (SysTick);
- 3 сторожевых таймера (2 независимых и оконный).
- Аппаратные ускорители:
- AES 128, 192, 256, TDES;
- HASH (MD5, SHA-1, SHA224, SHA256), HMAC;
- 2 аппаратных генератора истинно случайных чисел;
- 2 блока вычисления CRC.
- Отладочные возможности:
- трассировщик и отладчик Arm CoreSight: интерфейсы SWD и JTAG;
- встроенный трассировочный буфер на 8 Кбайт.

3.4.Интерфейсы и периферийные устройства несущей платы SDK-1.1M

Расширители портов ввода-вывода PCA9538PW

PCA9538PW [9] – это 8-битный расширитель портов ввода-вывода (GPIO) с поддержкой прерываний, подключенный к процессорному модулю по интерфейсу I^2C. PCA9538PW включает регистры конфигурации (настройка портов на вход или выход), входного и выходного портов, инверсии полярности. В SDK-1.1M имеется два расширителя, использующихся для подключения различных периферийных устройств и сигналов.

Часы реального времени MCP79411

MCP79411 – часы/календарь, совмещенные с 1 Кбит встроенной энергонезависимой памяти EEPROM. Часы используют внешний источник синхросигнала с частотой 32,768 кГц. Время отслеживается с использованием внутренних счетчиков часов, минут, секунд, дней, месяцев, лет, дней недели. Аппаратные прерывания (alarm) могут генерироваться по показаниям всех счетчиков вплоть до месяцев. Для использования и настройки MCP79411 поддерживает I^2C со скоростью до 400 кГц.

Графический OLED-дисплей WEO012864DL

WEO012864DL – монохромный OLED-дисплей 128×64 точек с диагональю 0,96 дюйма, подключенный по интерфейсу I^2C. Размеры области отображения 21,8×10,9 мм. Дисплей оборудован встроенным контроллером типа SSD1306BZ [10].

Ethernet

На стенде SDK-1.1M имеется разъем RJ-45 и приемопередатчик физического уровня Ethernet KSZ8081 [30]. Поддерживается передача данных по витой паре со скоростью 10/100 Мбит/с.

Излучатель звука HC0903A

HC0903A – электромагнитный излучатель звука (звукоизлучатель), управляемый прямоугольным периодическим сигналом.

Инерциальный измерительный модуль iNEMO LSM9DS

LSM9DS – это измерительный модуль, включающий трехмерные цифровые датчики линейного ускорения, угловой скорости и магнитного поля. Модуль подключается к процессорному модулю по последовательной шине I^2C и имеет отдельные выходы прерываний.

Клавиатура

Клавиатура имеет 12 кнопок и организована в виде матрицы 3×4, подключенной к расширителю портов ввода-вывода PCA9538PW. Три бита порта соответствуют столбцам, четыре бита соответствуют рядам.

4. Инструментальные средства стенда SDK-1.1M

4.1. Подключение стенда к компьютеру

Стенд SDK-1.1M подключается к персональному компьютеру через разъем «DBG USB» (см. рис. 36) кабелем с разъемом USB 2.0 Micro-B (входит в комплект поставки стенда). После подключения на лицевой панели стенда должен загореться светодиодный индикатор «VTGT».

Через разъем «DBG USB» обеспечивается электропитание стенда, а также подключение компьютера к встроенному программатору-отладчику стенда. Программатор-отладчик предназначен для загрузки программ в стенд и их отладки. Он обеспечивает подключение к вычислителю процессорного модуля по интерфейсам JTAG и UART (в режиме виртуального COM-порта). Через JTAG происходит взаимодействие с отладочной инфраструктурой вычислителя, а UART можно использовать для организации пользовательского отладочного ввода-вывода.

Для выключения стенда необходимо отключить его от компьютера.

ВНИМАНИЕ: когда стенд не используется, рекомендуется отключать USB-кабель от стенда во избежание случайного механического повреждения USB-разъема.

4.2. Установка драйверов встроенного программатора-отладчика

Встроенный программатор-отладчик SDK-1.1M построен с применением универсальной микросхемы-конвертера интерфейса USB фирмы FTDI [31]. Подключение с инструментального компьютера к программатору-отладчику выполняется с использованием специального драйвера USB и открытого фреймворка OpenOCD.

При первом использовании SDK-1.1M требуется установить на компьютер необходимые драйверы, как указано ниже.

Процедура установки драйверов для Windows

1. Подключить SDK-1.1M к компьютеру через разъем «DBG USB» (рис. 36) кабелем с разъемом USB 2.0 Micro-B. **ВНИМАНИЕ:** не путать разъем «DBG USB» с разъемом «USB» на процессорном модуле!
2. Проверить, что в операционной системе определилось новое USB-устройство. Если драйвер микросхемы FTDI уже установлен, стенд SDK-1.1M должен определяться как два виртуальных COM-порта. Драйвер FTDI может быть установлен автоматически через интернет при первом подключении стенда. Если драйвер отсутствует или не установился автоматически, требуется вручную установить FTDI Virtual COM port (VCP) driver [32].
3. Скачать и запустить программу Zadig [33] (позволяет заменять драйверы USB-устройств).
4. Убедиться, что SDK-1.1M подключен к компьютеру через разъем «DBG USB».
5. Во вкладке «Options» выбрать «List All Devices» (рис. 37).
6. Выбрать из списка устройство «RS232 (Interface 0)» или «SDK 1.1M Debugger (Interface 0)». **ВНИМАНИЕ:** выбор другого устройства может привести к нарушению работоспособности устройств компьютера!
7. Выбрать из списка драйвер «WinUSB» и нажать кнопку «Replace driver» (рис. 37).

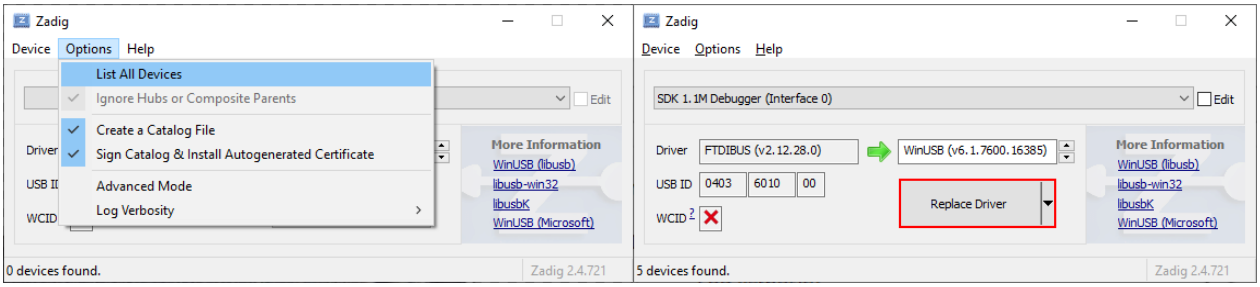


Рисунок 37 – Установка драйвера в программе Zadig

Процедура установки драйверов для Linux (Debian/Ubuntu)

1. Подключить SDK-1.1M к компьютеру.
2. Проверить, что SDK-1.1M определяется как два виртуальных COM-порта. В противном случае установить драйвер FTDI вручную (см. описание для Windows).
3. Открыть терминал.
4. Ввести команду:

```
$ sudo apt-get install libusb-1.0-0
```

5. Ввести команду:

```
$ sudo nano /etc/udev/rules.d/50-myusb.rules
```

или

```
$ sudo vi /etc/udev/rules.d/50-myusb.rules
```

6. Добавить строку:

```
SUBSYSTEMS=="usb",ATTRS{idVendor}=="0403",ATTRS{idProduct}=="6010",TAG+="uaccess"
```

7. Сохранить и закрыть файл.
8. Ввести команду:

```
$ sudo udevadm control –reload
```

9. Отключить и снова подключить SDK-1.1M к компьютеру.

Процедура установки драйверов для macOS

Установка драйверов для **macOS** не требуется.

4.3. Инструментальные средства разработки для микроконтроллеров STM32

Для разработки и отладки программ для микроконтроллеров семейства STM32 рекомендуется использовать среду разработки STM32CubeIDE [21], которая доступна для скачивания с сайта производителя [34].

STM32CubeIDE – это интегрированная среда разработки (IDE), которая включает:

- графический конфигуратор STM32CubeMX (Micro eXplorer; также доступен в виде отдельного приложения), позволяющий выбирать желаемую конфигурацию блоков микроконтроллера STM32 и генерировать шаблоны проектов с необходимым кодом инициализации на языке C или C++ посредством наглядного пошагового процесса (рис. 38);
- поддержку стандартной библиотеки драйверов (HAL) и дополнительных программных модулей (middleware);
- классические средства редактирования кода, сборки и отладки проектов.

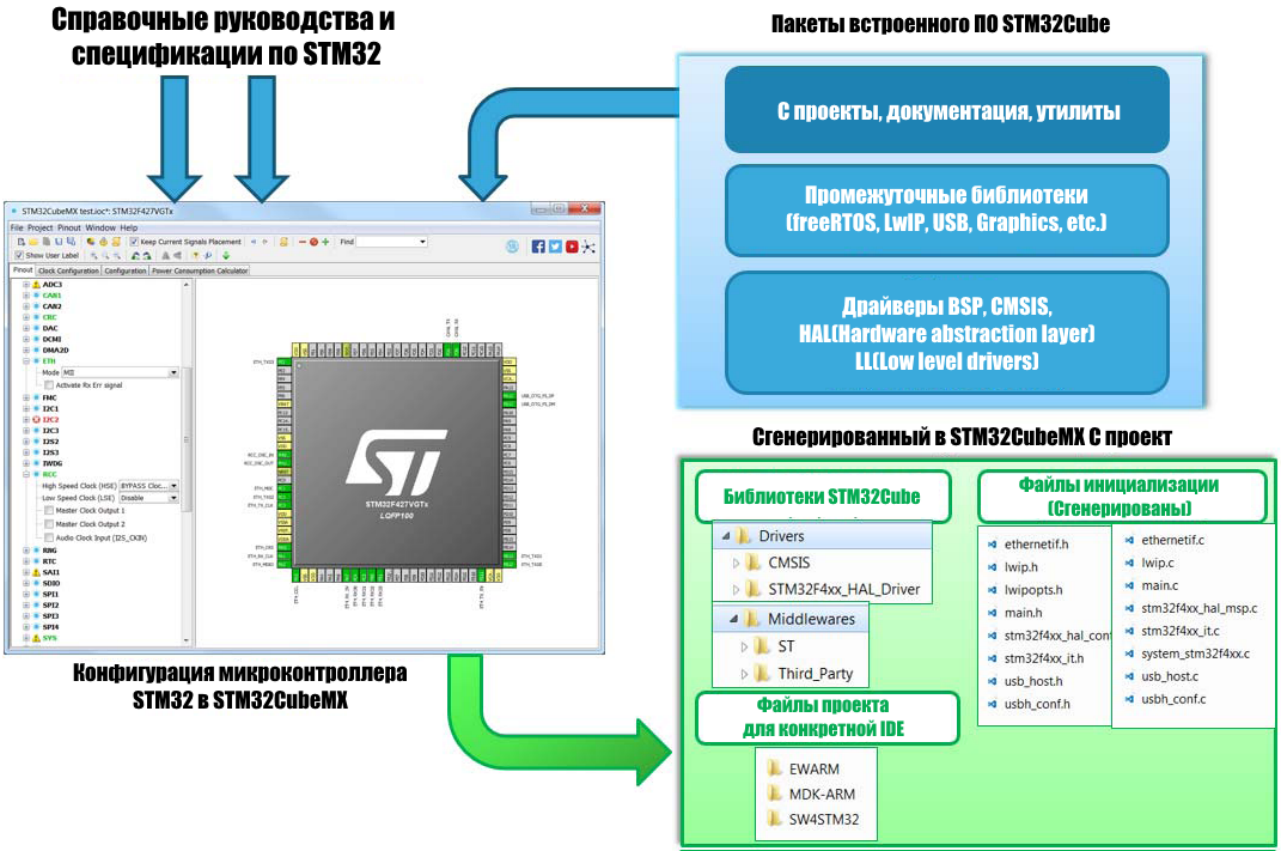


Рисунок 38 – Схема генерации проекта в STM32CubeMX

4.4.Пример создания и настройки проекта в STM32CubeIDE

STM32CubeIDE основана на среде разработки Eclipse, поэтому использует аналогичный данной IDE подход к работе с проектами. Каждый проект относится к какому-либо рабочему пространству (workspace). Простейший вариант рабочего пространства – это каталог, в котором располагаются каталоги проектов. Каждый workspace может содержать несколько проектов.

Рассмотрим пример создания и настройки проекта для SDK-1.1M с процессорным модулем на базе микроконтроллера STM32 в STM32CubeIDE версии 1.7.0. Более подробную информацию и документацию можно найти на сайте поставщика IDE.

На компьютере должен быть установлен драйвер встроенного программатора-отладчика SDK-1.1M (см. раздел выше).

Необходимо выполнить следующие действия:

1. Подготовить пустой каталог, который будет использоваться для workspace.

2. Запустить STM32CubeIDE. При каждом запуске предлагается выбрать каталог workspace, который требуется использовать. Указать подготовленный ранее каталог. (Позже можно переключиться на другой каталог, выбрав в меню File → Switch Workspace.)

3. Запустить мастер создания проекта: File → New → STM32 Project. При этом может быть запущено скачивание последней версии базы микроконтроллеров.

4. Необходимо выбрать целевой микроконтроллер. В строке поиска «Part Number» ввести или выбрать из списка наименование модели микроконтроллера, используемого в SDK-1.1M (можно ввести только начало названия). Например, для версии стенда SDK-1.1MC.427 это «STM32F427VI». После выбора справа появится список доступных вариантов

этой модели микроконтроллеров. Необходимо в списке справа указать необходимую модель и нажать кнопку «Next» (рис. 39).

5. В следующем окне ввести имя проекта и нажать кнопку «Finish».

6. Мастер создаст шаблон проекта. При этом может быть запущено скачивание последней версии библиотеки драйверов для выбранного семейства микроконтроллеров (объем – до нескольких сотен мегабайт).

7. Откроется для редактирования файл конфигурации проекта (*.ioc) в графическом инструменте STM32CubeMX, встроенном в IDE (рис. 40).

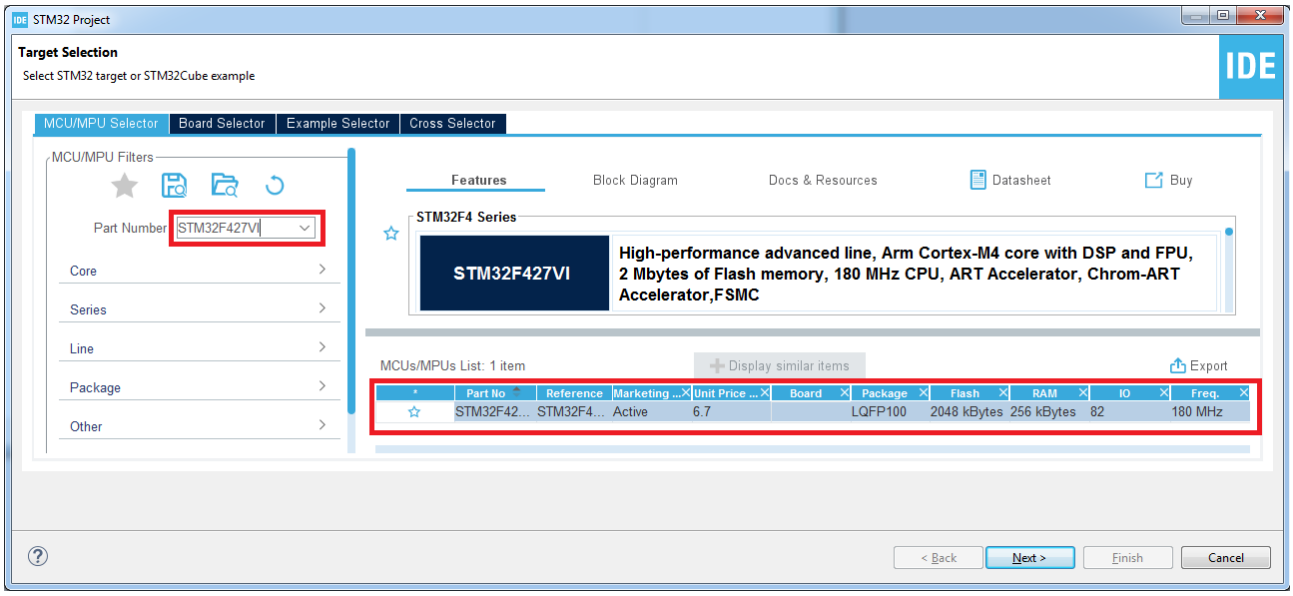


Рисунок 39 – Выбор микроконтроллера в мастере создания проекта

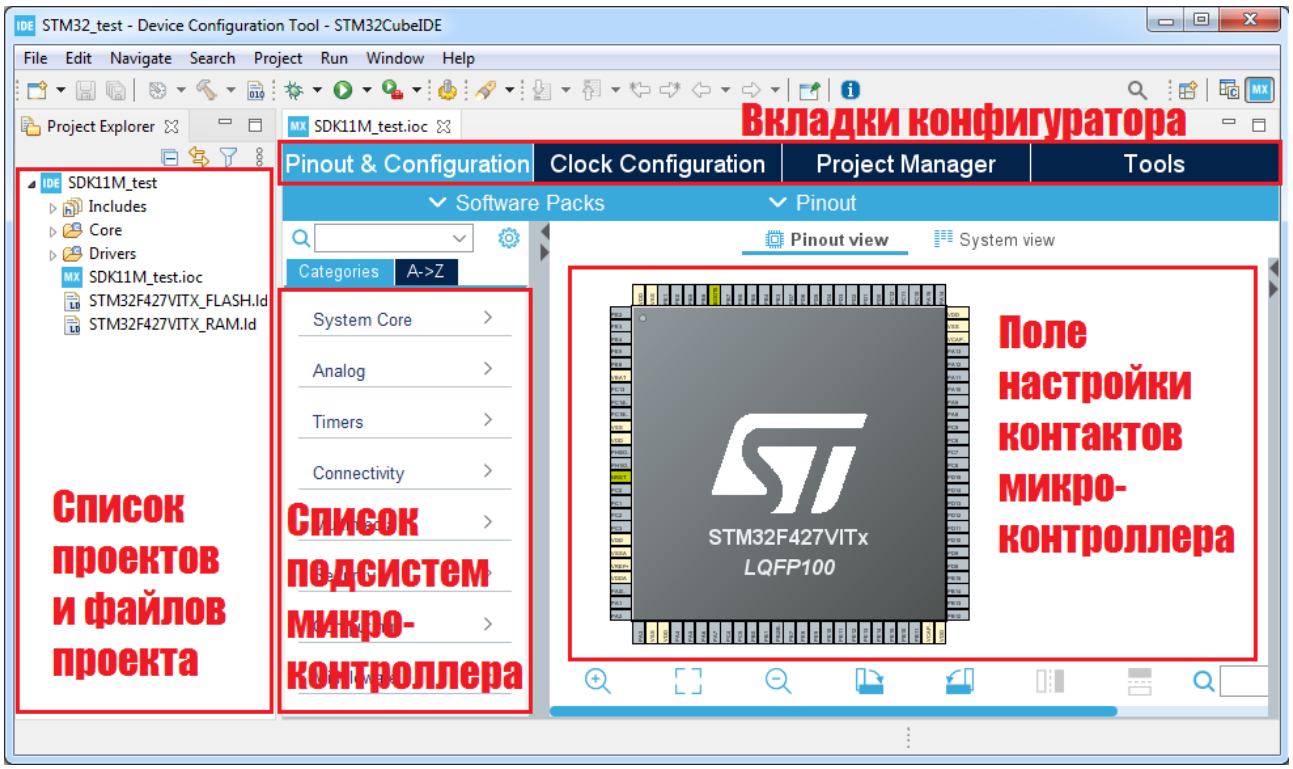


Рисунок 40 – Окно графического конфигуратора

8. Включить отладочный интерфейс JTAG. Вкладка Pinouts & Configuration → System Core → SYS → Debug → JTAG (5 pins).

9. Рекомендуется перейти на вкладку Project Manager и поставить галочку Code Generator → Generated files → Generate peripheral initialization as pair of '.c/.h' files per peripheral. Это необходимо для более удобной организации исходных текстов проекта.

10. Сохранить файл конфигурации проекта (нажать Ctrl+S или выбрать в главном меню File → Save). Будет предложено произвести повторную генерацию проекта с измененной конфигурацией и открыть исходный текст программы. Принять предложения в диалоговых окнах. STM32CubeIDE сгенерирует код для инициализации подсистем микроконтроллера и скопирует в проект необходимые библиотечные файлы. Генерацию также можно запускать вручную, выбрав в главном меню Project → Generate Code.

ВНИМАНИЕ: при повторной генерации файлов из них удаляется весь код, добавленный пользователем, кроме кода, который написан между парами комментариев вида `/* USER CODE BEGIN ... */` , `/* USER CODE END ... */` . Файлы, созданные пользователем, не удаляются и не модифицируются.

11. Создать в корневом каталоге проекта два пустых файла «SDK1_1_M FTDBG.cfg» и «SDK11M_FT.cfg». Они будут содержать конфигурацию отладчика. Скопировать в эти файлы текстовое содержимое, приведенное ниже.

ВНИМАНИЕ: файлы для разных моделей стенда отличаются!

Для SDK-1.1MC.107

SDK1_1_M FTDBG.cfg

```
source [find SDK11M_FT.cfg]
set WORKAREASIZE 0x8000
transport select jtag
set CHIPNAME STM32F107VCTx
set BOARDNAME SDK1_1M
reset_config srst_only
set CONNECT_UNDER_RESET 1
source [find target/stm32f1x.cfg]
```

SDK11M_FT.cfg

```
interface ftdi
ftdi_vid_pid 0x0403 0x6010
ftdi_layout_init 0x0028 0x0ffb
ftdi_layout_signal nSRST -oe 0x0800
```

Для SDK-1.1MC.407

SDK1_1_M FTDBG.cfg

```
source [find SDK11M_FT.cfg]
set WORKAREASIZE 0x8000
transport select jtag
set CHIPNAME STM32F407VGTx
set BOARDNAME SDK1_1_M
reset_config srst_only
set CONNECT_UNDER_RESET 1
source [find target/stm32f4x.cfg]
```

SDK11M_FT.cfg

```
interface ftdi
ftdi_vid_pid 0x0403 0x6010
ftdi_layout_init 0x0408 0x0ffb
ftdi_layout_signal nSRST -oe 0x0800
```

Для SDK-1.1MC.427

SDK1_1_M FTDBG.cfg

```
source [find SDK11M_FT.cfg]
set WORKAREASIZE 0x8000
transport select jtag
set CHIPNAME STM32F427VITx
set BOARDNAME SDK1_1_M
reset_config srst_only
set CONNECT_UNDER_RESET 1
source [find target/stm32f4x.cfg]
```

SDK11M_FT.cfg

```
interface ftdi
ftdi_vid_pid 0x0403 0x6010
ftdi_layout_init 0x0408 0x0ffb
ftdi_layout_signal nSRST -oe 0x0800
```

12. Скомпилировать проект. Для этого в главном меню выбрать: Project → Build All.

13. Ввести настройки программирования и отладки. Для этого в главном меню выбрать Project → Properties → Run/Debug Settings → New… → STM32 Cortex-M C/C++ Application. Перейти на вкладку «Debugger» и ввести следующие настройки (рис. 41):

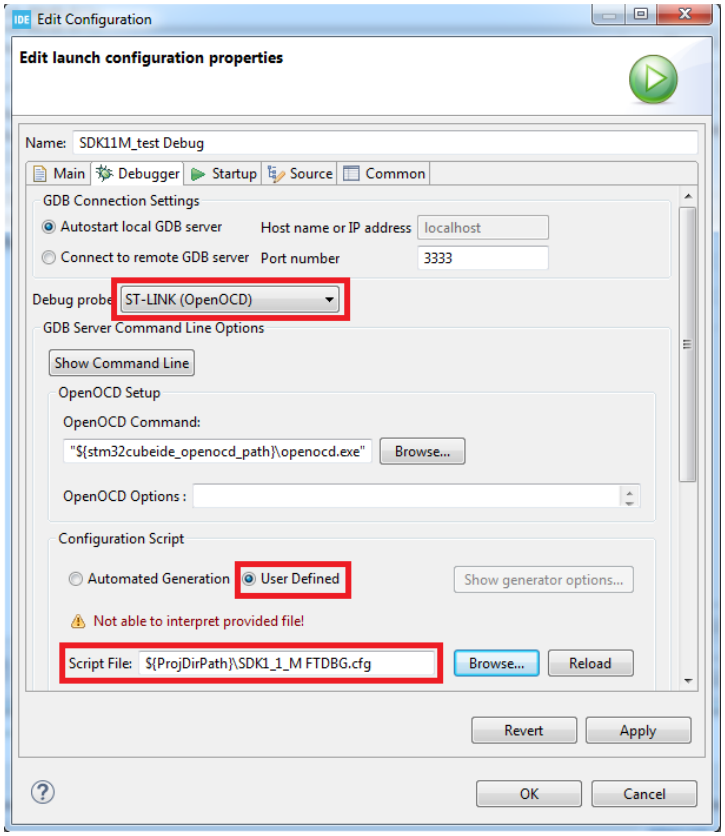


Рисунок 41 – Настройки отладочного интерфейса

a. в выпадающем списке «Debug probe» выбрать «ST-LINK (OpenOCD)»;
b. переключатель «Configuration Script» установить в «User Defined»;
c. в поле «Script File» задать путь к файлу «**SDK1_1_M FTDBG.cfg**» (нажать «Browse…»); игнорировать предупреждение IDE о том, что невозможно интерпретировать содержимое файла.

14. Сохранить введенные настройки и закрыть окно.

15. Выбрать в главном меню Run → Run или Run → Debug, чтобы загрузить программу в микроконтроллер и запустить ее или начать пошаговую отладку соответственно.

4.5. Использование отладочного вывода semihosting

В процессорных ядрах с архитектурой ARM имеются встроенные средства для отладки. В частности, они позволяют выполнять текстовый отладочный ввод-вывод с помощью функций printf() и scanf() поверх стандартного отладочного интерфейса JTAG или SWD без использования контроллера UART и отдельных проводов. Такой подход называется semihosting. При достижении функции printf() целевая программа останавливается для обмена данными с инструментальным компьютером (host-системой).

Semihosting можно использовать только во время сеанса отладки. Программа, скомпилированная с функциями ввода-вывода, работающими в режиме semihosting, запущенная без соединения с отладочным сервером на инструментальном компьютере, работать не будет.

Недостатком использования semihosting является сравнительно большое время, которое тратится на выполнение ввода-вывода и непредсказуемое влияние на производительность программы.

Для использования отладочного вывода semihosting необходимо выполнить в проекте следующие настройки [35]:

1. Открыть настройки подключаемых библиотек: Project → Properties → C/C++ General → Paths and Symbols, вкладка «Source Location».

2. Раскрыть пункт <Имя проекта>/Core, выделить пункт «Filter (empty)» и нажать кнопку «Edit Filter…».

3. Нажав кнопку «Add…» добавить путь «Src/syscalls.c» и нажать кнопку «OK». Так из проекта исключается стандартная библиотека системных вызовов (рис. 42). Сохранить введенные настройки и закрыть окно.

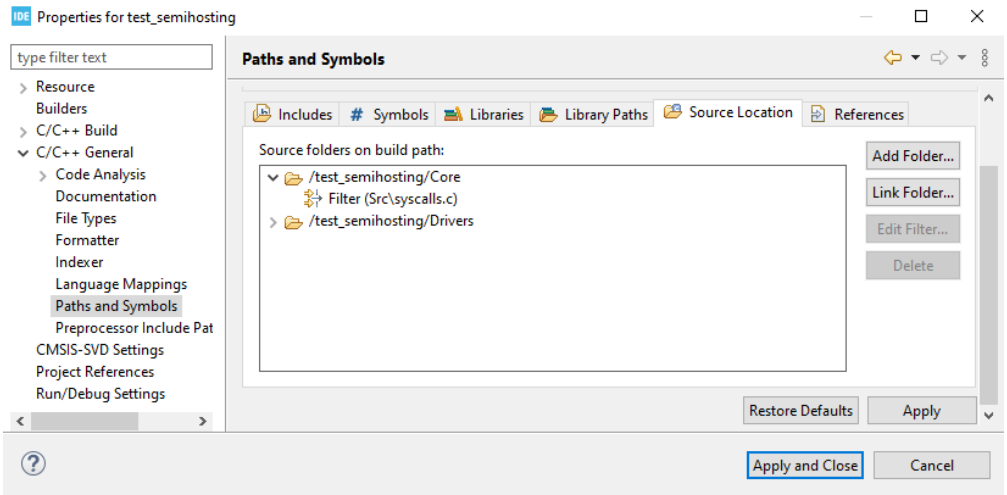


Рисунок 42 – Добавление в проект фильтра библиотеки syscalls.c

4. Теперь необходимо добавить в проект библиотеку с реализацией системных вызовов, использующих semihosting. Для этого требуется открыть настройки компоновщика: Project → Properties → C/C++ Build → Settings, вкладка «Tool Settings».

5. Выбрать пункт MCU GCC Linker → Libraries и на панели «Libraries» нажать кнопку добавления пункта. Добавить строку «rdimon».

6. Выбрать пункт MCU GCC Linker → Miscellanous и на панели «Other flags» аналогично добавить строку «-specs=rdimon.specs» (рис. 43).

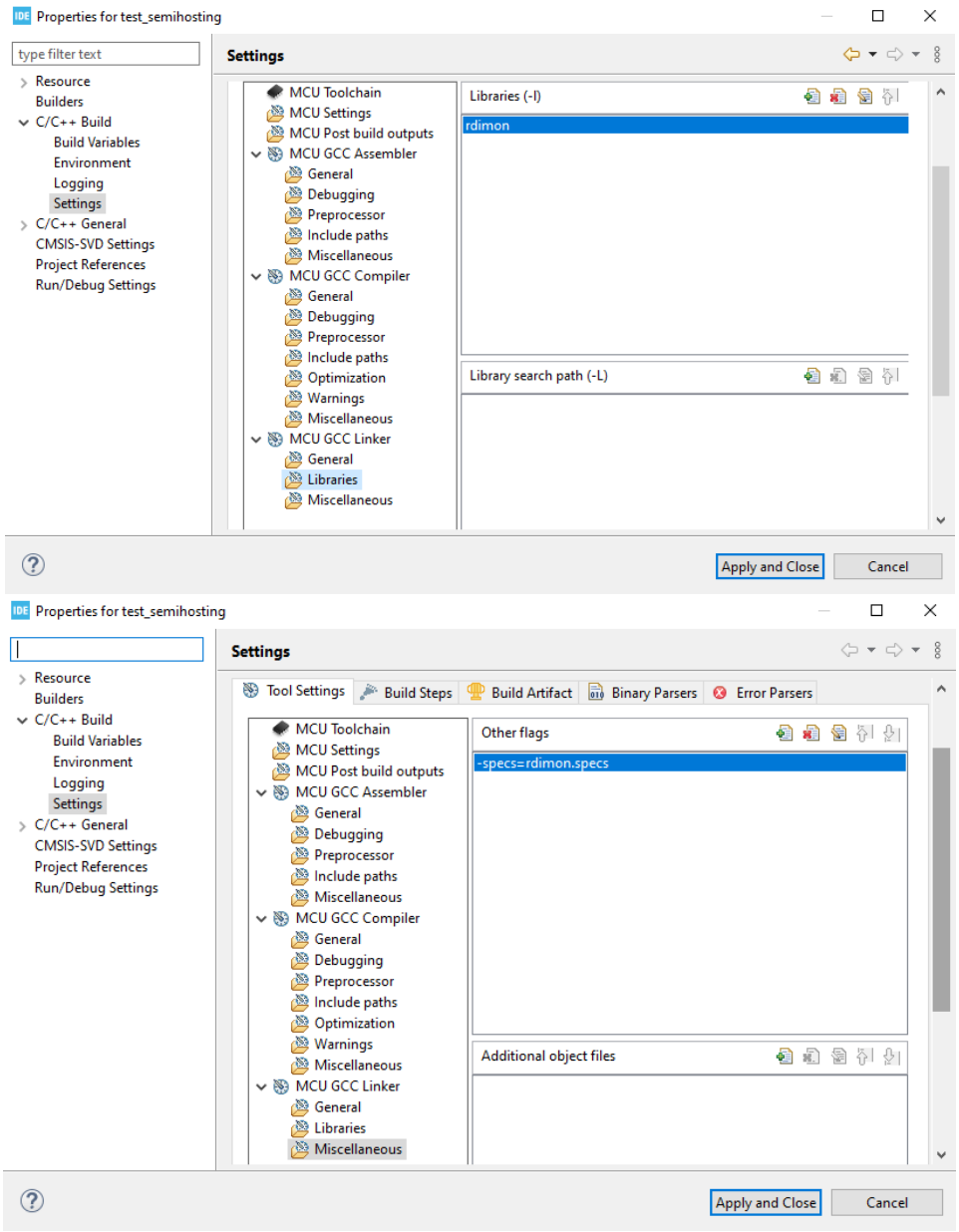


Рисунок 43 – Добавление в проект библиотеки rdimon

7. В файл main.c выше определения функции main() (между комментариями USER CODE 0) добавить строку: `extern void initialise_monitor_handles(void);`

8. Внутри функции main() (между комментариями USER CODE 1) добавить строку: `initialise_monitor_handles();`

9. В каждый файл, где планируется использовать функции отладочного вывода, необходимо добавить заголовочный файл: `#include "stdio.h"` (в сгенерированных файлах – между комментариями USER CODE Includes).

10. Теперь можно вставлять в программу строки вида: `printf("Hello world!\n");`.

11. Настроить отладчик. Для этого перейти к конфигурациям отладчика: Project → Properties → Run/Debug Settings, выбрать созданную ранее конфигурацию и нажать кнопку «Edit…». Перейти на вкладку «Startup» и в поле «Initialization Commands» ввести строку: «monitor arm semihosting enable». Сохранить настройки и выйти.

После выполнения указанных настроек проект необходимо запускать в режиме отладки, выбрав в главном меню: Run → Debug. После запуска отладчика исполнение программы будет остановлено на первой строке. Необходимо запустить программу командой Run → Resume. В консоли должен появиться отладочный вывод функции printf() (рис. 44).

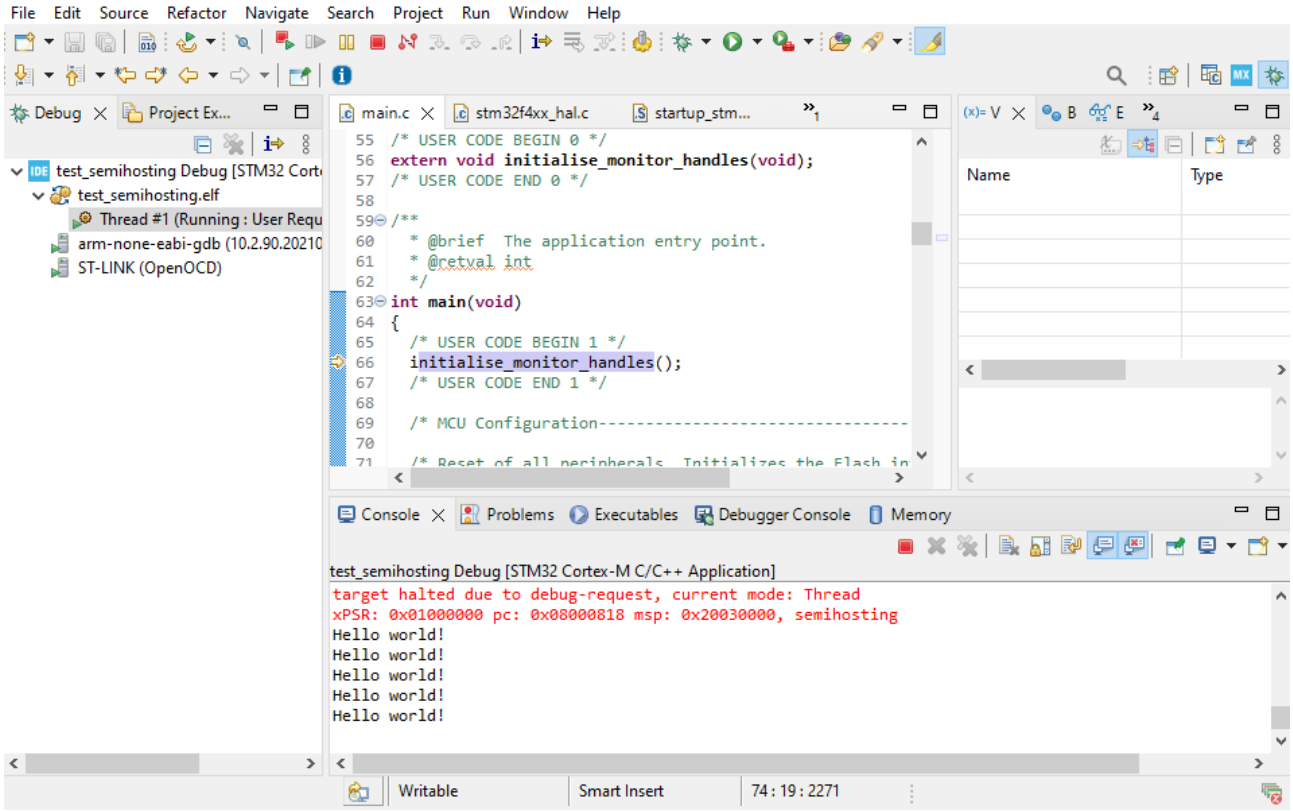


Рисунок 44 – Запуск программы с отладкой в режиме semihosting

4.6. Система виртуальных лабораторий ITMO.cLAB

Облачная лаборатория ITMO.cLAB (ITMO Cloud Laboratory) – онлайн-сервис с веб-интерфейсом для удаленного доступа к различному лабораторному оборудованию и установкам, разработанный на факультете ПИиКТ Университета ИТМО [36, 37]. Общая организация ITMO.cLAB показана на рис. 45.

ITMO.cLAB предназначена для решения широкого спектра образовательных и научно-технических задач:

1. Обеспечение лабораторной базы для проведения дистанционных онлайн-курсов ИТМО с большим количеством студентов (тысячи человек).

2. Обеспечение студентов лабораторной базой для проведения экспериментов в рамках научной и преддипломных практик.

3. Поддержка научно-технического творчества студентов в рамках различных неформальных объединений.

4. Предоставление доступа к лабораторному оборудованию на коммерческой основе для университетов, колледжей, лицеев, кружков, коммерческих фирм и стартапов, а также энтузиастов-любителей (DIY).

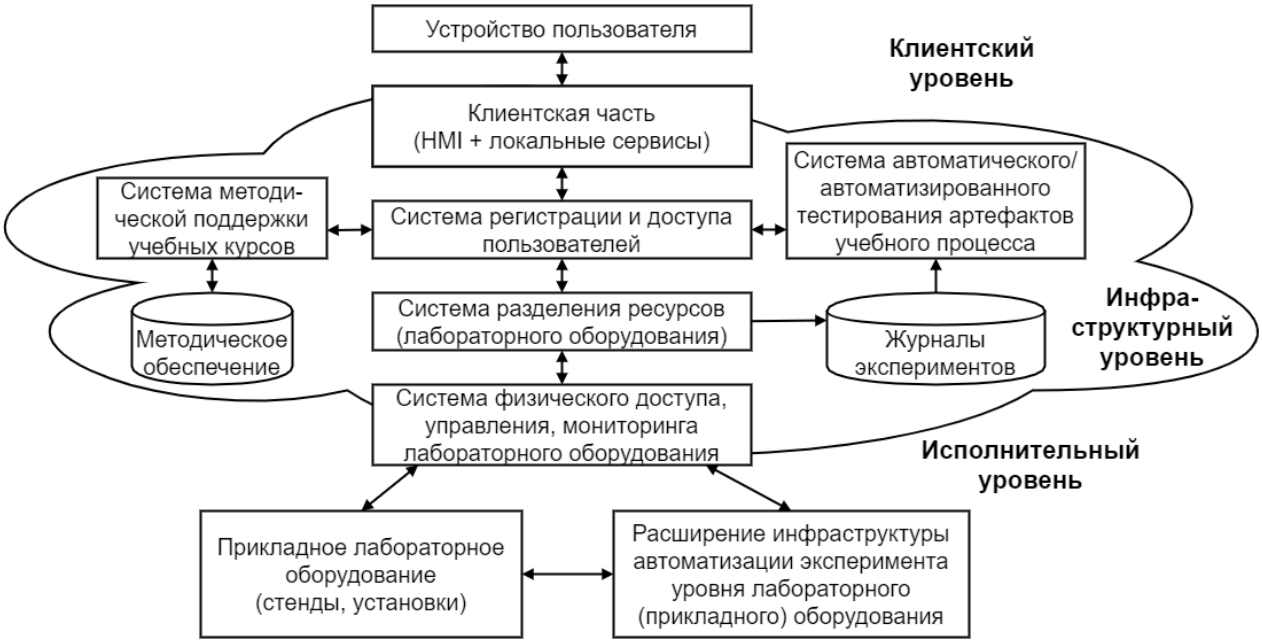


Рисунок 45 – Общая организация ITMO.cLAB

Виртуальные лаборатории ITMO.cLAB могут применяться для поддержки учебных курсов, связанных с проектированием современной вычислительной техники (встроенных и киберфизических систем, систем на базе «Интернета вещей», систем для Индустрии 4.0 и т.д.), курсов в естественнонаучных областях (лабораторных работ по теплофизике, физике, оптике и т.д.) и других областях. ITMO.cLAB позволяет выполнять лабораторные и курсовые работы, реализовать практическую часть для дипломного проекта, ставить эксперименты в ходе научного исследования или проверять на практике различные идеи.

Высокую ценность для образовательного процесса представляет то, что ITMO.cLAB обеспечивает работу не с виртуальным, а с реальным оборудованием, так как ни один симулятор не сможет заменить реальный прибор. Используя ITMO.cLAB, можно получать доступ к оборудованию из любой точки земного шара и в любое время.

В ITMO.cLAB реализована поддержка микропроцессорных стендов SDK-1.1M, ведутся работы по интеграции оборудования для курсов по физике тепловых процессов [38].

По сравнению с большинством аналогичных проектов ITMO.cLAB обеспечивает пакетный режим работы, позволяющий уменьшить затраты на оборудование и увеличить количество пользователей без потери производительности. Например, кластер из восьми стендов SDK-1.1M может за одну минуту провести несколько десятков экспериментов.

4.7.Пример работы с SDK-1.1M через ITMO.cLAB

Программирование SDK-1.1M для запуска в ITMO.cLAB имеет некоторые отличия по сравнению с вариантом, когда стенд подключается непосредственно к инструментальному компьютеру. В базовом сценарии программа исполняется из ОЗУ микроконтроллера, и на ее выполнение отводится около 8 секунд, после чего сторожевой таймер перезагружает микроконтроллер.

Для начала работы необходимо скачать шаблон проекта для STM32CubeIDE [39]. В этот проект добавлены модули, позволяющие отслеживать процесс и получать результаты выполнения программы на удаленном SDK-1.1M. Для этого необходимо пользоваться функциями, определенными в файле trace.c. Описание функций представлено в исходных текстах, а также в README-файле исходного репозитория.

После импорта проекта в среду разработки можно сконфигурировать дополнительную периферию стенда, учитывая особенности выполнения программ в ITMO.cLAB. Перед сборкой проекта рекомендуется убедиться в правильной настройке компоновщика и векторов прерываний микроконтроллера STM32F407:

1. В файле system_stm32f4xx.c должна быть раскомментирована строка:

```
#define VECT_TAB_SRAM
```

2. Убедиться, что в файле system_stm32f4xx.c установлено правильное значение смещения таблицы векторов прерываний:

```
#define VECT_TAB_OFFSET  0xD000
```

3. Карта памяти в файле STM32F407VGTX_RAM.ld должна иметь следующий вид:

```
MEMORY {
CCMRAM    (xrw)    : ORIGIN = 0x10000000,   LENGTH = 64K
RAM    (xrw)    : ORIGIN = 0x2000D000,   LENGTH = 76K
FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 1024K
}
```

4. В настройках компоновщика (Project → Properties → C/C++ Build → Settings → Tool Settings → MCU GCC Linker → General) выбрать правильный Linker Script (STM32F407VGTX_RAM.ld).

5. Включить генерацию двоичного файла образа программы. Для этого открыть раздел настроек сборки проекта (Project → Properties → C/C++ Build → Settings → Tool Settings → MCU Post Build Outputs) и отметить галочкой пункт «Convert to binary file».

После сборки проекта в каталог Debug (или Release) сохраняется двоичный файл с образом программы. Его можно загружать в ITMO.cLAB на соответствующей странице интерфейса.

Также на странице загрузки есть две области для отображения результатов исполнения программы (рис. 46):

– временная диаграмма событий, которые были записаны в память во время исполнения пользовательской программы;
– текстовый вывод строк с временной меткой.

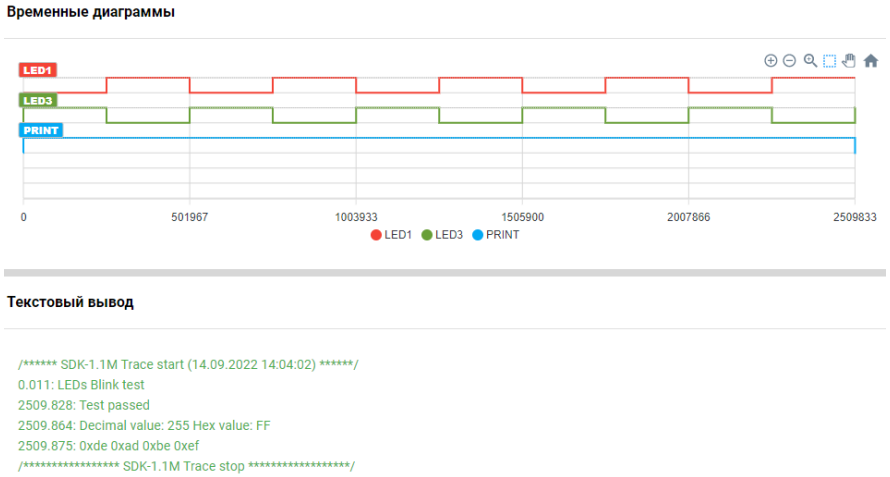


Рисунок 46 – Отображение результатов исполнения программы SDK-1.1M в ITMO.cLAB

4.8. Электрическая схема и примеры программирования стенда SDK-1.1M

Электрическая принципиальная схема стенда SDK-1.1M и примеры программ для стенда с процессорным модулем на базе микроконтроллеров STM32F407 и STM32F427 размещены в общедоступном репозитории [40]. Примеры программ демонстрируют работу с различными компонентами стенда и могут служить основной для новых проектов.

4.9. Частые вопросы

В табл. 14 представлены описания проблем, которые могут возникать при работе со стендом и способы их устранения.

Таблица 14 – Возможные проблемы и способы их устранения

№	Описание	Возможные причины	Способы устранения
1	При загрузке программы в стенд появляются ошибки вида «не найден программатор».	USB-кабель подключен к разъему «USB» стенда SDK-1.1M, а не к разъему «DBG USB».	Подключить кабель к разъему «DBG USB».
		Неправильно установлен драйвер встроенного программатора-отладчика.	Установите драйвер программатора-отладчика, как указано в соответствующем разделе.
		Неверно заданы настройки программирования и отладки.	Задать настройки, как указано в примере создания проекта.
2	При загрузке программы в стенд появляется сообщение об ошибке вида «Program file does not exists».	Настройки программирования и отладки задавались до компиляции проекта, из-за чего IDE автоматически не подставила имя исполняемого файла.	Открыть настройки: Project → Properties → Run/Debug Settings → <Название конфигурации> → Edit… На вкладке Main в поле «C/C++ Application» указать путь к образу программы *.elf (выбрать файл, нажав кнопку «Search Project»).
3	При загрузке программы в стенд появляются сообщения об ошибке вида «JTAG scan chain interrogation failed: all ones», «Invalid ACK (7) in DAP response».	Следствие аппаратной ошибки в микроконтроллерах семейства STM32F4: использование контакта SYS_JTRST приводит к блокированию интерфейса программирования и отладки.	Не инициализировать контакт SYS_JTRST в программном проекте. Чтобы перепрограммировать стенд, использовать программу STM32CubeProgrammer и загрузить программу через разъем «USB». Предварительно перевести микроконтроллер в режим программирования, замкнув контакты программирования («BOOT») на процессорном модуле при перезагрузке.
4	Программа не загружается в стенд с другими сообщениями об ошибке или загружается, но не исполняется	Необходимо перезагрузить стенд	Нажать кнопку перезагрузки на стенде
		Предыдущая загруженная программа имеет неправильную конфигурацию или записаны некорректные значения в конфигурационные биты микроконтроллера (OPB).	Подключиться к микроконтроллеру по USB с помощью программы STM32CubeProgrammer и выполнить полное стирание FLASH-памяти микроконтроллера. Проверить, что конфигурационные биты (OPB) имеют состояние по умолчанию. В случае отклонений привести их к состоянию по умолчанию.

5. Лабораторный практикум

Требования к выполнению лабораторных работ

Требования к оформлению программ

Код программы должен быть разделен на файлы, каждый из которых содержит отдельный драйвер или логически обособленный модуль программы. Желательно продумывать разбиение программы на модули по критериям минимальной взаимозависимости и максимального потенциала повторного использования.

Код должен быть структурированным, хорошо читаемым. Названия переменных и функций должны быть осмысленными и поясняющими их назначение. На «верхнем уровне» программы, т.е. в функции main() и других функциях, реализующих прикладную логику программы, не должно присутствовать деталей работы с аппаратурой. Конкретные номера портов ввода-вывода, интерфейсов, особенности их реализации скрываются от программиста прикладной логики и конкретизируются на уровне драйверов.

В программе не должно быть одинаковых или очень похожих блоков кода. Желательно оформлять такой код в виде отдельных функций, макросов и т.п.

Необходимо избегать циклов с «активным ожиданием» событий, где это не является необходимым, особенно в драйверах.

Требования к содержанию отчёта

1. Титульный лист, на котором указывается следующая информация:
- название университета;
- название факультета, кафедры;
- название дисциплины;
- номер и тема лабораторной работы;
- вариант задания;
- фамилия, инициалы и номер группы каждого исполнителя;
- фамилия и инициалы преподавателя;
- текущий год.

2. Задание к лабораторной работе в соответствии с вариантом.

3. Описание организации программы и структуры драйверов, блок-схема прикладного алгоритма.

4. Дополнительные материалы, если они требуются по заданию конкретной лабораторной работы (руководство по использованию и т.п.).

5. Исходные коды разработанной программы, включая драйверы и библиотеки. При большом объеме исходных кодов можно включать только наиболее значимые в данной работе части.

6. Выводы, сделанные в процессе выполнения лабораторной работы.

Требования к оформлению отчёта

1. Отчет выполняется в виде самодостаточного документа, никакие фрагменты отчета не должны быть представлены в виде ссылок на внешние ресурсы. Материал, изложенный в отчете, должен быть понятен без дополнительных комментариев со стороны исполнителей.

2. Отчет выполняется как текстовый документ в соответствии с ГОСТ 2.105-95.

3. Размер шрифта основного текста 12-14 pt (Times, Calibri или аналогичный), межстрочный интервал 1-1,5, поля с краев листа не менее 2 см.

4. Листы отчета должны быть пронумерованы, кроме титульного листа, который считается первым.

5. Обязательны нумерация и подписи к рисункам и таблицам, а также ссылки на них в тексте отчета.

6. Схемы, рисунки, диаграммы должны быть выполнены темными/цветными линиями на светлом фоне. В распечатанном отчете линии и тексты на рисунках должны быть четко видны.

Защита лабораторной работы

На защите лабораторной работы задаются вопросы по:

- теоретическому материалу, используемому в данной работе;
- инструментальным средствам, которые использовались в данной работе;
- принципиальной электрической схеме стенда SDK-1.1M;
- исходным кодам программы, включая драйвера и библиотеки.

5.1.Лабораторная работа 1. Интерфейсы ввода-вывода общего назначения (GPIO)

Цели работы

1. Получить базовые знания о принципах устройства стенда SDK-1.1M и программировании микроконтроллеров.

2. Изучить устройство интерфейсов ввода-вывода общего назначения (GPIO) в микроконтроллерах и приемы использования данных интерфейсов.

Задание

Разработать и реализовать драйверы управления светодиодными индикаторами и чтения состояния кнопки стенда SDK-1.1M (расположены на боковой панели стенда). Контакты подключения кнопки и светодиодов должны быть настроены в режиме GPIO. Функции и другие компоненты драйверов должны быть универсальными, т.е. пригодными для использования в любом из вариантов задания и не должны содержать прикладной логики программы. Функции драйверов должны быть неблокирующими, то есть не должны содержать ожиданий события (например, нажатия кнопки). Также, в драйверах не должно быть пауз с активным ожиданием функция HAL_Delay() и собственные варианты аналогичной реализации. Обработка нажатия кнопки в программе должна включать программную защиту от дребезга.

Написать программу с использованием разработанных драйверов в соответствии с вариантом задания.

Порядок выполнения работы

1. Изучить:

- устройство и инструкцию по эксплуатации стенда SDK-1.1M;
- основные характеристики микроконтроллера STM32 по листу данных (datasheet); документация выложена на сайте производителя [34] (например, см. документацию микроконтроллера STM32F427VIT6 [41], обозначение документа – DS9405 [7]);
- разделы учебного пособия:
    - 1.1. Сигналы сброса и синхронизации;
    - 1.2. Интерфейс ввода-вывода общего назначения (GPIO);
- электрическую принципиальную схему стенда в части управления светодиодами и кнопкой, используемыми в данной работе;
- раздел «General purpose I/O (GPIO)» из справочного руководства (reference manual) по микроконтроллерам линейки STM32F4 (обозначение документа – RM0090 [4]), знать устройство портов ввода-вывода, режимы их работы и способы настройки.

2. Создать и настроить пустой проект программы для SDK-1.1M.
3. Настроить тактовые частоты.
4. Настроить сигналы GPIO, необходимые для выполнения задания.
5. Изучить:

- состав стандартного драйвера GPIO из библиотеки HAL (файлы stm32f4xx_hal_gpio.c/.h в проекте), знать основные определения и функции, принцип работы драйвера. Справочная информация находится в комментариях в файле драйвера в документации (обозначение документа – UM1725 Description of STM32F4 HAL and low-layer drivers [42]) с сайта производителя;
- содержимое инициализационного кода в созданных генератором файлах gpio.c/.h.

6. Разработать драйверы управления светодиодами и чтения состояния кнопки.
7. Разработать программу согласно варианту задания и провести ее тестирование.

Варианты заданий

Вариант 1

Сымитировать работу светофора пешеходного перехода. Светофор циклически переключает цвета в следующем порядке (порядок условный, соответствие реальному светофору не соблюдается): красный, зелёный, зелёный мигающий, жёлтый, снова красный и т.д. По умолчанию период горения красного в четыре раза больше периода горения зеленого. Если во время горения зеленого мигающего, желтого или красного нажимается кнопка, светофор запоминает необходимость скорейшего переключения на зелёный. После нажатия кнопки общий цикл работы светофора не нарушается, но период горения красного должен быть сокращен до ¼ своего обычного периода. Если кнопка нажата во время горения красного, когда он уже горит более ¼ периода, то сразу происходит переключение на зеленый.

Вариант 2

Реализовать простой имитатор гирлянды с переключением режимов. Должно быть реализовано не менее четырех последовательностей переключения светодиодов, обязательно с разной частотой мигания (должны быть визуально отличимыми). В каждом режиме задействуется не менее двух светодиодов. По нажатию кнопки происходит циклическое переключение режимов. При повторном входе в какой-либо режим анимация на светодиодах должна запускаться с того места, на котором в прошлый раз была прервана переключением

на следующий режим. Удерживание зажатой кнопки не должно приводить к «повисанию» анимации.

Вариант 3

Реализовать «передатчик» азбуки Морзе. Последовательность из нажатий кнопки (короткое – точка, длинное – тире) запоминается и после сигнала окончания ввода (долгая пауза) начинает «отправляться» при помощи зелёного светодиода, последовательностью быстрых (точка) и долгих (тире) мерцаний. Во время ввода последовательности после каждого нажатия двухцветный светодиод должен коротким мерцанием индицировать, какой сигнал был введён (мигание жёлтым – точка, мигание красным – тире).

Вариант 4

Реализовать двоичный двухразрядный счётчик на светодиодах с возможностью вычитания (использовать зеленый светодиод и один из двух цветов двухцветного). Быстрое нажатие кнопки должно прибавлять единицу к отображаемому на светодиодах двоичному числу. По переполнению счётчика должна отображаться простая анимация: мигание обоими светодиодами, затем количество миганий зелёным светодиодом, равное количеству переполнений с момента перезагрузки микроконтроллера. Долгое нажатие кнопки должно вычитать единицу из отображаемого на светодиодах двоичного числа. Если происходит вычитание из нуля, количество переполнений уменьшается на единицу, и отображается анимация, аналогичная анимации при переполнении.

Вариант 5

Реализовать «кодовый замок». После ввода единственно верной последовательности не менее чем из восьми коротких и длинных нажатий должен загореться зелёный светодиод, обозначающий «открытие» замка. Светодиод горит некоторое время, потом гаснет, и система вновь переходит в «режим ввода». Каждый неправильно введённый элемент последовательности должен сопровождаться миганием красного светодиода и сбросом в «начало», каждый правильный – миганием жёлтого. После трёх неправильных вводов начинает мигать красный светодиод, и через некоторое время возвращается в «режим ввода». Если код не введен до конца за некоторое ограниченное время, происходит сброс в «начало».

5.2.Лабораторная работа 2. Последовательный интерфейс UART

Цели работы

1. Изучить протокол передачи данных по интерфейсу UART.
2. Получить базовые знания об организации системы прерываний в микроконтроллерах на примере микроконтроллера STM32.
3. Изучить устройство и принципы работы контроллера интерфейса UART, получить навыки организации обмена данными по UART в режимах опроса и прерываний.

Задание

Разработать и реализовать два варианта драйверов UART для стенда SDK-1.1M: с использованием и без использования прерываний. Драйверы, использующие прерывания, должны обеспечивать работу в «неблокирующем» режиме (возврат из функции происходит

сразу же, без ожидания окончания приема/отправки), а также буферизацию данных для исключения случайной потери данных. В драйвере, не использующем прерывания, функция приема данных также должна быть «неблокирующей», то есть она не должна зависать до приема данных (которые могут никогда не поступить). При использовании режима «без прерываний» прерывания от соответствующего блока UART должны быть запрещены.

Написать с использованием разработанных драйверов программу, которая выполняет определенную вариантом задачу. Для всех вариантов должно быть реализовано два режима работы программы: с использованием и без использования прерываний. Каждый принимаемый стендом символ должен отсылаться обратно, чтобы он был выведен в консоли (так называемое «эхо»). Каждое новое сообщение от стенда должно выводиться с новой строки. Если вариант предусматривает работу с командами, то на каждую команду должен выводиться ответ, определенный в задании или «ОК», если ответ не требуется. Если введена команда, которая не поддерживается, должно быть выведено сообщение об этом.

Порядок выполнения работы

1. Изучить:

- разделы учебного пособия:
  - 1.3. Прерывания;
  - 1.4. Последовательный интерфейс UART;
- электрическую принципиальную схему стенда в части назначения сигналов отладочного интерфейса UART;
- раздел справочного руководства RM0090: «Universal synchronous asynchronous receiver transmitter (USART)», знать устройство контроллера USART, режимы его работы и способы настройки;
- раздел справочного руководства PM0214 (programming manual) [43] о контроллере прерываний: «Nested vectored interrupt controller (NVIC)», знать принципы организации системы прерываний в микроконтроллере STM32F4 и приемы ее использования;
- состав стандартного драйвера USART из библиотеки HAL и содержимое создаваемых генератором файлов usart.c/.h.

2. Подготовить шаблон проекта для STM32CubeIDE, настроить тактовые частоты.

3. Задать настройки контроллера UART, настроить входы и выходы микроконтроллера.

4. Разработать драйверы UART для работы в режимах опроса и прерывания. Драйвер, работающий в режиме прерывания, должен поддерживать буферизацию данных и работу в «неблокирующем режиме».

5. Разработать программу согласно варианту задания и провести ее тестирование.

Варианты заданий

Вариант 1

Доработать программу «светофор», добавив возможность отключения кнопки и задание величины тайм-аута (период, в течение которого горит красный).

Должны обрабатываться следующие команды, посылаемые через UART:

- *?* – в ответ стенд должен прислать состояние, которое отображается в данный момент на светодиодах: *red*, *yellow*, *green*, *blinking green*, режим – *mode 1* или *mode 2* (см.

далее), величину тайм-аута (сколько горит красный) – *timeout …*, и задействованы ли прерывания – символ *I* (interrupt) или *P* (polling);
- *set mode 1* или *set mode 2* – установить режим работы светофора, когда обрабатываются или игнорируются нажатия кнопки;
- *set timeout X* – установить тайм-аут (X – длина периода в секундах);
- *set interrupts on* или *set interrupts off* – включить или выключить прерывания.

Вариант 2

Доработать программу «гирлянда», реализовав возможность добавления четырёх новых последовательностей миганий светодиодов с индивидуальной настройкой частоты переключения состояний для каждой последовательности. Каждая вводимая последовательность должна иметь от двух до восьми состояний. В один момент времени может гореть только один светодиод (или не гореть ни один). Смена отображаемой в данный момент последовательности должна осуществляться нажатием кнопки или командой, посылаемой через UART.

Должны обрабатываться следующие команды, посылаемые через UART:

- *new xx…* – ввести новую последовательность, где «x» – это одна из букв g, r, y, n («g» соответствует включению зелёного светодиода, «r» – красного, «y» – жёлтого, «n» означает, что ни один светодиод не горит); количество вводимых значений «x» может быть от двух до восьми, ввод завершается либо по нажатию Enter, либо после ввода восьми значений; после окончания ввода последовательности мерцаний стенд должен послать сообщение произвольного содержания, приглашающее ввести частоту мерцаний светодиодов (должны предусматриваться минимум три градации); ввод частоты мерцаний заканчивается по нажатию Enter; новой последовательности присваивается очередной свободный номер от 5 до 8; если уже есть 8 последовательностей, то переопределяется последовательность 5 и т.д.; номер новой сохраненной последовательности выводится в UART;
- *set x* – сделать активной последовательность мерцаний *x*, где *x* – порядковый номер;
- *set interrupts on* или *set interrupts off* – включить или выключить прерывания.

Вариант 3

Доработать программу, посылающую сигналы азбуки Морзе для латинского алфавита. Последовательность точек и тире, полученных нажатием кнопки стенда, должна дешифроваться и отправляться через последовательный канал в виде символов (букв латинского алфавита).

Символы, получаемые стендом через последовательный канал, должны шифроваться и «проигрываться» с помощью светодиода. Если в момент «мигания» приходят новые символы, они добавляются в очередь. Должна быть возможность одновременной буферизации до восьми символов: они должны запоминаться стендом и последовательно проигрываться.

Считывание нажатий кнопки и отправка дешифрованных символов должна функционировать одновременно с приёмом через последовательный канал и миганием светодиода.

Включение/выключение прерываний должно осуществляться при получении стендом символа «+».

Вариант 4

Разработать программу-калькулятор. Ввод значений производится с компьютера через UART: *xx…xyxx…x*=, где x – десятичные цифры, y – знак (+, -, *, /). Ввод чисел завершается либо знаком операции (для первого числа), либо знаком «равно» (для второго числа), либо после ввода пяти цифр числа. Обратите внимание, что операнды не могут быть отрицательными, а ответ может.

Размерность результата и обоих операндов должна быть short int (16-битовое знаковое число), и должна быть предусмотрена защита от переполнения. В случае выполнения недопустимых операций (ответ или вводимые числа больше, чем размер переменных в памяти) должен загораться красный светодиод, а в последовательный канал вместо ответа выводиться слово *error*.

Включение/отключение прерываний должно осуществляться нажатием кнопки на стенде и сопровождаться отправкой в последовательный порт сообщения произвольного содержания, сообщающего, какой режим включен (с прерываниями или без прерываний).

Вариант 5

Доработать программу кодового замка. Теперь ввод кода должен происходить не с помощью кнопки стенда, а по UART. После ввода единственно верной последовательности из не более чем восьми латинских букв без учёта регистра и цифр должен загореться зелёный светодиод, обозначающий «открытие» замка. Светодиод горит некоторое время, потом гаснет, и система вновь переходит в «режим ввода». Каждый неправильно введённый элемент последовательности должен сопровождаться миганием красного светодиода и сбросом в «начало», каждый правильный – миганием жёлтого. После трёх неправильных вводов начинает мигать красный светодиод, и через некоторое время система вновь возвращается в «режим ввода». Если код не введен до конца за некоторое ограниченное время, происходит сброс в «начало».

Должно быть предусмотрено изменение отпирающей последовательности, что производится следующей последовательностью действий:

- ввод символа «+»;
- ввод новой последовательности, который завершается либо по нажатию *enter*, либо по достижению восьми значений;
- стенд отправляет сообщение произвольного содержания, спрашивая, сделать ли последовательность активной, и запрашивает подтверждение, которое должно быть сделано вводом символа *y*;
- после ввода *y* введённая последовательность устанавливается как активная.

Включение/отключение прерываний должно осуществляться нажатием кнопки на стенде и сопровождаться отправкой в последовательный порт сообщения произвольного содержания, сообщающего, какой режим включен (с прерываниями или без прерываний).

5.3.Лабораторная работа 3. Таймеры

Цели работы

1. Получить базовые знания об устройстве и режимах работы таймеров в микроконтроллерах.

2. Получить навыки использования таймеров и прерываний от таймеров.

3. Получить навыки использования аппаратных каналов ввода-вывода таймеров.

Задание

Разработать программу, которая использует таймеры для управления яркостью светодиодов и излучателем звука (по прерыванию или с использованием аппаратных каналов). Блокирующее ожидание (функция HAL_Delay()) в программе использоваться не должно.

Стенд должен поддерживать связь с компьютером по UART и выполнять указанные действия в качестве реакции на нажатие кнопок на клавиатуре компьютера. В данной лабораторной работе каждая нажатая кнопка (символ, отправленный с компьютера на стенд) обрабатываются отдельно, ожидание ввода полной строки не требуется.

Для работы с UART на стенде можно использован один из двух вариантов драйвера (по прерыванию и по опросу) на выбор исполнителя. Поддержка двух вариантов не требуется.

Порядок выполнения работы

1. Изучить:

- разделы учебного пособия:
  - 1.5. Программируемые таймеры;
- электрическую принципиальную схему стенда в части излучателя звука;
- разделы справочного руководства RM0090:
  - Basic timers (TIM6 and TIM7);
  - General-purpose timers (TIM2 to TIM5);
  - General-purpose timers (TIM9 to TIM14);
  - Advanced-control timers (TIM1 and TIM8);
- состав стандартных драйверов таймеров из библиотеки HAL и содержимое создаваемых генератором файлов tim.c/.h (появляются в проекте после настройки таймеров в графическом конфигураторе).

2. Подготовить шаблон проекта для STM32CubeIDE, настроить тактовые частоты.

3. Настроить входы и выходы микроконтроллера, таймеры и UART.

4. Разработать необходимые драйверы управляемых светодиодов и/или излучателя звука (зависит от варианта).

5. Разработать прикладную программу согласно варианту задания и протестировать ее.

Варианты заданий

Вариант 1

Реализовать «музыкальную клавиатуру» с помощью излучателя звука.

Существует девять стандартных октав от субконтроктавы (первая по порядку) до пятой октавы (девятая по порядку) (более подробно об октавах см. в специализированных источниках). Частоты нот в соседних октавах отличаются ровно в два раза и растут с номером октавы. Частоты для первой октавы (пятая по порядку):

Нота	Частота, Гц
До	261,63
Ре	293,67
Ми	329,63
Фа	349,23
Соль	392,00
Ля	440,00
Си	493,88

Стенд должен выполнять следующие действия при получении символов от компьютера:

Символ	Действие
«1» – «7»	Воспроизведение одной ноты (от «до» до «си») текущей октавы с текущей длительностью звучания. Начальные значения: первая октава (пятая по порядку), длительность 1 с.
«+»	Увеличение номера текущей октавы (максимальная – пятая).
«-»	Уменьшение номера текущей октавы (минимальная – субконтроктава).
«A»	Увеличение длительности воспроизведения ноты на 0,1 с (максимум – 5 с).
«a»	Уменьшение длительности воспроизведения ноты на 0,1 с (минимум – 0,1 с).
«Enter»	Последовательное воспроизведение всех нот текущей октавы с текущей длительностью без пауз.

По вводу каждого символа в UART должно выводиться сообщение:

- для символов «1» – «7», «Enter»: какая нота какой октавы и с какой длительностью проигрывается;
- для символов настройки: новые значения номера октавы и длительности звучания ноты;
- для символов, не перечисленных в таблице выше: сообщение «неверный символ» и его код.

Вариант 2

Реализовать настраиваемый пульт управления светодиодами боковой панели. Пульт должен иметь два основных режима: рабочий режим и режим настройки.

Действия стенда при получении символов от компьютера в рабочем режиме:

Символ	Действие
«1» – «9»	Зажигание светодиода в соответствии с настройками пульта. Светодиод горит до ввода следующего символа. Предыдущий светодиод выключается. Настройки по умолчанию: «1» – зеленый, 10 % яркости; «2» – зеленый, 40 % яркости; «3» – зеленый, 100 % яркости; «4» – желтый, 10 % яркости; «5» – желтый, 40 % яркости; «6» – желтый, 100 % яркости; «7» – красный, 10 % яркости; «8» – красный, 40 % яркости; «9» – красный, 100 % яркости.
«0»	Погасить все светодиоды.
«Enter»	Вход в режим настройки.

После входа в меню настройки сначала надо ввести символ от «1» до «9», настройки для которого требуется изменить. Далее символом «a», «b» или «c» выбирается светодиод (зеленый, желтый, красный соответственно). После этого несколькими нажатиями кнопок

«+» и «-» задается коэффициент заполнения от 0 до 100 % с шагом 10 %. По нажатию кнопки «Enter» сохраняется новая настройка и происходит переход в рабочий режим.

По вводу каждого символа в UART должно выводиться сообщение о том, какой режим активирован, или текущие вводимые настройки.

Вариант 3

Реализовать имитатор гирлянды с плавными включениями/выключениями светодиодов и переходами между цветами. Должно быть предусмотрено четыре разных предустановленных режима анимации и один пользовательский режим, который можно настраивать. В каждом режиме должно быть задействовано минимум два светодиода, при этом не обязательно, чтобы они могли гореть одновременно.

Действия стенда при получении символов от компьютера:

Символ	**Действие**
«1»	Переход на следующий режим.
«2»	Возврат к предыдущему режиму.
«3»	Ускорение воспроизведения анимации на 10 % от текущей скорости.
«4»	Замедление воспроизведения анимации на 10 % от текущей скорости.
«5»	Вывод в UART параметров текущего сохраненного пользовательского режима.
«Enter»	Вход в меню настройки.
На усмотрение исполнителей	Ввод параметров пользовательского режима: цвет светодиода, яркость, необходимость плавного перехода, длительность, конец последовательности и т.п.

После ввода каждого символа в UART должно выводиться сообщение о том, какой режим активирован, или текущие настройки, вводимые в меню.

Вариант 4

Реализовать «музыкальную шкатулку» с мелодиями, которые состоят из последовательности звуков определенной частоты и длительности, а также пауз. Шкатулка должна иметь четыре стандартные мелодии и одну пользовательскую, которую можно настроить.

Действия стенда при получении символов от компьютера:

Символ	**Действие**
«1» – «4»	Воспроизведение одной из стандартных мелодий.
«5»	Воспроизведение пользовательской мелодии.
«Enter»	Вход в меню настройки.
На усмотрение исполнителей	Ввод параметров пользовательской мелодии: частота (нота, октава), длительность, конец мелодии и т.п.

После ввода каждого символа в UART должно выводиться сообщение о том, какой режим активирован, или текущие настройки, вводимые в меню.

Вариант 5

Реализовать музыкальную ритм-игру. С помощью звукоизлучателя воспроизводится последовательно, состоящая из звуков разной частоты («мелодия»). Каждый звук сопровождается зажиганием светодиода определенного цвета и с определенной яркостью (регулируется коэффициентом заполнения). Должно существовать взаимно однозначное соответствие между частотой звука и цветом/яркостью светодиода. Во время каждого звука/импульса светодиода игрок должен ввести символ, соответствующий текущей частоте

звука или цвету/яркости. Чем больше звуков будет «угадано» правильно и на большей скорости игры, тем больше очков заработает игрок (система начисления очков – на усмотрение исполнителей).

Всего необходимо предусмотреть девять видов импульсов: зеленый, желтый и красный на 20 %, 50 % и 100 % яркости. К ним следует подобрать звуки произвольных частот, легко отличимых одна от другой на слух. Предусмотреть одну стандартную последовательность импульсов длительностью не менее 20-ти элементов (простейший вариант – циклический перебор девяти импульсов). Когда последовательность заканчивается или досрочно останавливается игроком, в UART выводится количество набранных очков и «трассировка» нажатий, где отмечены правильные и неправильные нажатия. Отсутствие нажатия в течение импульса должно считаться неправильным нажатием.

Действия стенда при получении символов от компьютера:

Символ	Действие
«1» – «9»	Символы, которые надо вводить в течение игры во время соответствующих импульсов. Когда игра не запущена, при вводе данных символов соответствующие импульсы воспроизводятся на светодиодах и излучателе звука, а в UART выводится описание импульса, которому соответствует символ (цвет/яркость и частота звука).
«+»	Циклически переключать скорость игры (длительность каждого импульса): малая, средняя, быстрая. Необходимо подобрать длительности импульсов так, чтобы на малой скорости было комфортно набирать очки, а максимальная скорость не была «непроходимой».
«a»	Циклически переключать режим воспроизведения последовательности: светодиоды и звук, только светодиоды, только звук.
«Enter»	Запустить или досрочно остановить игру. При запуске должна выдерживаться пауза в три секунды после ввода символа, чтобы игрок успел подготовиться.

После ввода каждого символа в UART должно выводиться сообщение о том, какой импульс выбран или какой режим игры установлен.

5.4.Лабораторная работа 4. Интерфейс I^2C и матричная клавиатура

Цели работы

1. Получить базовые знания об интерфейсе I^2C и особенностях передачи данных по данному интерфейсу.

2. Получить базовые знания об устройстве и принципах работы контроллера интерфейса I^2C в микроконтроллерах и получить навыки его программирования.

Задание

Разработать программу, которая использует интерфейс I^2C для считывания нажатий кнопок клавиатуры стенда SDK-1.1.

Подсистема опроса клавиатуры должна удовлетворять следующим требованиям:

- реализуется защита от дребезга;
- нажатие кнопки фиксируется сразу после того, как было обнаружено, что кнопка нажата (с учетом защиты от дребезга), а не в момент отпускания кнопки; если необходимо, долгое нажатие может фиксироваться отдельно;
- кнопка, которая удерживается дольше, чем один цикл опроса, не считается повторно нажатой до тех пор, пока не будет отпущена (нет переповторов);

- распознается и корректно обрабатывается множественное нажатие (при нажатии более чем одной кнопки считается, что ни одна кнопка не нажата, если это не противоречит требованиям к программе);
- всем кнопкам назначаются коды от 1 до 12 (порядок на усмотрение исполнителей).

Программа должна иметь два режима работы, переключение между которыми производится по нажатию кнопки на боковой панели стенда:

- режим тестирования клавиатуры;
- прикладной режим.

Уведомление о смене режима выводится в UART.

В режиме тестирования клавиатуры программа выводит в UART коды нажатых кнопок.

В прикладном режиме программа обрабатывает нажатия кнопок и выполняет действия в соответствии с вариантом задания.

Порядок выполнения работы

1. Изучить:

- разделы учебного пособия:
  - 1.6. Интерфейс I^2C;
  - 1.7. Матричная клавиатура;
- электрическую принципиальную схему стенда в части сигналов интерфейса I^2C и подключения к нему внешних микросхем, а также других устройств, если они требуются по варианту задания;
- разделы справочного руководства RM0090:
  - Inter-integrated circuit (I2C) interface;
- состав стандартных драйверов I^2C из библиотеки HAL и содержимое создаваемых генератором файлов i2c.c/.h (появляются в проекте после настройки соответствующих блоков).

2. Подготовить шаблон проекта для STM32CubeIDE, настроить тактовые частоты.
3. Настроить входы и выходы микроконтроллера, таймеры, контроллеры I^2C и UART.
4. Разработать подсистему опроса клавиатуры и протестировать ее.
5. Разработать необходимые драйверы управляемых светодиодов и/или излучателя звука (зависит от варианта).
6. Разработать прикладную программу согласно варианту задания и протестировать ее.

Варианты заданий

Задания аналогичны вариантам лабораторной работы №3, за исключением того, что ввод символов должен выполняться не с клавиатуры через UART, а с помощью клавиатуры стенда. Выбор кнопок клавиатуры стенда, играющих роль кнопок клавиатуры компьютера должен выполняться по усмотрению исполнителей. В отчете необходимо привести описание функций кнопок в реализованной программе.

5.5.Лабораторная работа 5. OLED-дисплей

Цели работы

1. Изучить устройство и принципы работы дисплейного модуля в стенде SDK-1.1M.

2. Получить навыки программирования дисплейных устройств.

Задание

Разработать программу, которая включает драйвер OLED-дисплея. Для организации обмена данными с дисплеем можно использовать те же три способа, что и в предыдущей работе. Поскольку дисплей управляется по той же шине I^2C, что и клавиатура, необходимо избежать конфликтов доступа к шине. При работе с I^2C по опросу они не возникнут, а при работе по прерыванию следует планировать вычисления так, чтобы транзакции не пересекались по времени.

Если транзакции I^2C генерируются по прерыванию от таймера, следует встроить транзакции обновления видеопамяти дисплея в расписание опроса клавиатуры. При этом следует учесть, что отправка буфера данных в контроллер дисплея происходит значительно дольше, чем любая транзакция опроса клавиатуры. Например, после восьми вызовов обработчика прерываний для опроса клавиатуры можно инициировать отправку буфера и несколько вызовов обработчика отправлять никаких транзакций по I^2C, давая время на завершение отправки буфера. Начальную инициализацию дисплея, как и клавиатуры, удобнее делать в режиме опроса, пока не включены прерывания от таймера.

Порядок выполнения работы

1. Изучить:
- разделы учебного пособия:
    - 1.8. Дисплейный модуль;
- подключение дисплейного модуля на схеме стенда SDK-1.1M;
- функциональность контроллера дисплея SSD1306BZ по документации;
- пример программы для работы с дисплеем.

2. Подготовить шаблон проекта для STM32CubeIDE.

3. Разработать подсистему поддержки OLED-дисплея в соответствии с требованиями задания и протестировать ее.

4. Разработать набор функций для реализации графических примитивов, необходимых для выполнения задания.

5. Разработать прикладную программу в соответствии с вариантом задания и протестировать ее.

6. Оформить отчет, в котором привести краткое руководство пользователя для разработанной программы: режимы работы, назначение кнопок, и т.п.

Варианты заданий

Вариант 1

Написать программу, реализующую функции электронного секундомера. Точность измерения времени – десятые доли секунды. Управление производится посредством двух кнопок клавиатуры SDK-1.1M (старт/стоп и сброс).

На дисплее отображается строка символов в формате ММ:СС.Д, где ММ – минуты, СС – секунды, Д – десятые доли секунды.

Вариант 2

Написать программу, реализующую простую игру «пройди ворота». С левой стороны дисплея располагается «корабль» игрока, который может перемещаться верх и вниз (форма и размеры корабля – на усмотрение исполнителей). С правой стороны на него движется последовательность «ворот» – вертикальных стенок с отверстием в два раза больше корабля. Отверстия располагаются на разном уровне, поэтому для прохождения ворот необходимо перемещать корабль.

Игра запускается и останавливается по нажатию кнопки клавиатуры SDK-1.1M. Управление кораблем вверх/вниз выполняется с помощью еще двух кнопок. Последовательность должна содержать не менее 5 ворот. Если корабль врезается в стенку, игра считается не пройденной. После завершения игры на дисплей выводится сообщение с оценкой результата: «Success» или «Fail».

Вариант 3

Написать программу, реализующую простой графический редактор. На дисплее должна отображаться мигающая точка – «курсор». С помощью кнопок клавиатуры курсор можно перемещать по экрану с шагом в одну или более точек. По нажатию одной из кнопок переключается режим перемещения: без рисования, рисование белым, рисование черным (при рисовании курсор оставляет за собой «след» соответствующего цвета).

По нажатию одной из кнопок клавиатуры на экране запускается игра «жизнь», начальным состоянием которой должно являться текущее состояние экрана.

Необходимо предусмотреть кнопку для очистки экрана. Очистка также останавливает игру «жизнь».

Вариант 4

Адаптировать программу-калькулятор для использования с дисплеем и клавиатурой SDK-1.1M. Кнопки клавиатуры использовать для:

- ввода цифр 0 – 9;
- ввода операции: тип операции выбирается нажатием отдельной кнопки (одно нажатие – «+», два – «-», три – «*», четыре – «/», далее снова происходит переход к «+»), текущая выбранная операция запоминается при начале ввода второго операнда;
- окончания ввода и расчета ответа (эквивалент «=»).

На экране отображается три строки:

- первый операнд и знак операции;
- второй операнд и знак «=»;
- результат.

Пример:

xxxxx+

yyyyy=

zzzzz

При начале ввода первого операнда экран очищается от предыдущих данных.

Вариант 5

Адаптировать программу-кодовый замок для использования с дисплеем и клавиатурой SDK-1.1M. Кнопки клавиатуры использовать для:

- ввода пароля (цифры 0 – 9);

– перехода в режим изменения пароля/сохранения нового пароля.

Отображение статуса выполняется на дисплее. В обычном режиме на экране выведено текстовое приглашение для ввода пароля. После ввода производится проверка его правильности, и на экран выводится соответствующее сообщение. При переходе в режим изменения пароля выводится текстовое приглашение вида «введите новый пароль». Длина пароля – от 8 до 12 цифр. Если пользователь пытается сохранить пароль неправильной длины, выводится сообщение об ошибке.

Дополнительные задания

В дополнение к представленным выше лабораторным работам могут быть разработаны задания для изучения:

– других внешних интерфейсов и периферийных устройств SDK-1.1M, а также плат расширения;
– организации программного обеспечения встраиваемых систем, операционных систем, драйверов, библиотек;
– взаимодействия встраиваемых систем с персональными компьютерами и интернет-серверами.

Обобщенные примеры некоторых возможных заданий приведены ниже.

Инерциальный измерительный модуль

Реализовать на SDK-1.1M программу, выводящую показания датчиков измерительного модуля, выводящую на монитор направление действия силы тяжести, направление на северный магнитный полюс Земли.

Интерфейс USB

Реализовать на SDK-1.1M эмуляцию стандартного устройства, подключаемого к персональному компьютеру по USB: адаптер COM-порта, клавиатура (использовать клавиатуру стенда или ввод по UART), мышь (использовать акселерометр или магнитометр), устройство памяти и т.д.

Операционная система реального времени FreeRTOS

Реализовать на SDK-1.1M программу с несколькими взаимодействующими процессами, используя механизмы многозадачности FreeRTOS.

Интернет-стек LwIP

Реализовать на SDK-1.1M программу для обмена данными с компьютером, используя один из интерфейсов LwIP.

Микропроцессоры STM32MP15x

Разработать программы для ОС Linux и ядра Cortex-M4, обменивающиеся данными через общую память или специализированные каналы взаимодействия.

Рекомендуемая литература

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е издание. Издательский дом ПИТЕР, 2022.

2. Паттерсон Д., Хеннесси Дж. Архитектура компьютера и проектирование компьютерных систем. 4-е издание. Издательский дом ПИТЕР, 2012.

3. Харрис Д., Харрис С. Цифровая схемотехника и архитектура компьютера. ДМК Пресс, 2018.

4. Батоврин В.К. Системная и программная инженерия. ДМК Пресс, 2010.

5. Ключев А.О., Быковский С.В., Болдырева Е.А. Онлайн-курс «Встроенные системы». URL: https://openedu.ru/course/ITMOUniversity/EMBSYS/ (дата обращения: 14.01.2022).

6. Ключев А.О., Кустарев П.В., Платунов А.Е. Аппаратные средства информационно-управляющих систем. Учебное пособие – Санкт-Петербург: СПб: Университет ИТМО, 2015. URL: https://books.ifmo.ru/book/1568/apparatnye_sredstva_informacionno-upravlyayuschih_sistem._uchebnoe_posobie.htm (дата обращения: 14.01.2022).

7. Ключев А.О., Ковязина Д.Р., Кустарев П.В., Платунов А.Е. Аппаратные и программные средства встраиваемых систем – Санкт-Петербург: Университет ИТМО, 2010. URL: https://books.ifmo.ru/book/575/apparatnye_i_programmnye_sredstva_vstraivaemyh_sistem.htm (дата обращения: 14.01.2022).

8. Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительных систем – Санкт-Петербург, 2009. URL: https://books.ifmo.ru/book/460/programmnoe_obespechenie_vstroennyh_vychislitelnyh_sistem.htm (дата обращения: 14.01.2022).

9. Ключев А.О., Ковязина Д.Р., Петров Е.В., Платунов А.Е. Интерфейсы периферийных устройств – Санкт-Петербург: СПбГУ ИТМО, 2010. URL: https://books.ifmo.ru/book/612/interfeysy_periferiynyh_ustroystv.htm (дата обращения: 14.01.2022).

10. Ключев А.О., Кустарев П.В., Платунов А.Е. Распределенные информационно-управляющие системы. Учебное пособие – Санкт-Петербург: СПб.: Университет ИТМО, 2015. URL: https://books.ifmo.ru/book/1569/raspredelennye_informacionno-upravlyayuschie_sistemy._uchebnoe_posobie..htm (дата обращения: 14.01.2022).

11. Быковский С.В., Горбачев Я.Г., Ключев А.О., Пенской А.В., Платунов А.Е. Сопряжённое проектирование встраиваемых систем (Hardware/Software Co-Design). Часть 1: Учебное пособие – Санкт-Петербург: Университет ИТМО, 2016. URL: https://books.ifmo.ru/book/1861/sopryazh%D1%91nnoe_proektirovanie_vstraivaemyh_sistem_(Hardware/Software_Co-Design)._chast_1:_uchebnoe_posobie.htm (дата обращения: 14.01.2022).

12. Быковский С.В., Горбачев Я.Г., Ключев А.О., Пенской А.В., Платунов А.Е. Сопряжённое проектирование встраиваемых систем (Hardware/Software Co-Design). Часть 2: Учебное пособие – Санкт-Петербург: Университет ИТМО, 2016. URL: https://books.ifmo.ru/book/1862/sopryazh%D1%91nnoe_proektirovanie_vstraivaemyh_sistem_(Hardware/Software_Co-Design)._chast_2:_uchebnoe_posobie.htm (дата обращения: 14.01.2022).

Список использованных источников

1. Дизайн-центр "ЛМТ". URL: http://lmt.spb.ru/ (дата обращения: 14.01.2022).
2. Факультет программной инженерии и компьютерной техники. Университет ИТМО. URL: https://scs.ifmo.ru/ru/ (дата обращения: 14.01.2022).
3. Ключев А.О., Платунов А.Е., Дергачев А.М. Опыт использования лабораторных стендов SDK в учебном процессе // Научно-технический вестник информационных технологий, механики и оптики - 2019. - Т. 19. - № 2(120). - С. 306-313.
4. RM0090 Reference manual. STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm-based 32-bit MCUs. STMicroelectronics.
5. Ключев А.О., Ковязина Д.Р., Кустарев П.В., Платунов А.Е. Апп. и прогр. средства встраиваемых систем – СПб: Университет ИТМО, 2010. URL: https://books.ifmo.ru/book/575/apparatnye_i_programmnye_sredstva_vstraivaemyh_sistem.htm (дата обращения: 14.01.2022).
6. Ключев А.О., Ковязина Д.Р., Петров Е.В., Платунов А.Е. Интерфейсы периферийных устройств – Санкт-Петербург: СПбГУ ИТМО, 2010. URL: https://books.ifmo.ru/book/612/interfeysy_periferiynyh_ustroystv.htm (дата обращения: 14.01.2022).
7. STM32F427xx, STM32F429xx. Datasheet - production data. STMicroelectronics. URL: https://www.st.com/resource/en/datasheet/stm32f427vi.pdf (дата обращения: 14.01.2022).
8. Шина I2C и как её использовать. URL: http://www.gaw.ru/html.cgi/txt/interface/iic/index.htm (дата обращения: 14.01.2022).
9. PCA9538. Product data sheet. NXP Semiconductors. URL: https://www.nxp.com/docs/en/data-sheet/PCA9538.pdf (дата обращения: 14.01.2022).
10. SSD1306B. 128x64 Dot Matrix OLED/PLED Segment/Common Driver with Controller. May 2014. Solomon Systech Limited.
11. GLCD Font Creator. URL: https://www.mikroe.com/glcd-font-creator (дата обращения: 14.01.2022).
12. FreeRTOS - Real-time operating system for microcontrollers. URL: https://freertos.org/ (дата обращения: 14.01.2022).
13. lwIP - A Lightweight TCP/IP stack. URL: http://savannah.nongnu.org/projects/lwip/ (дата обращения: 14.01.2022).
14. STM32MP153C/F. Datasheet - production data. STMicroelectronics. URL: https://www.st.com/resource/en/datasheet/stm32mp153c.pdf (дата обращения: 14.01.2022).
15. STM32 MPU OpenSTLinux Distribution. STMicroelectronics. URL: https://www.st.com/en/embedded-software/stm32-mpu-openstlinux-distribution.html (дата обращения: 14.01.2022).
16. Trusted Firmware A (TF-A). URL: https://www.trustedfirmware.org/projects/tf-a/ (дата обращения: 14.01.2022).
17. Open Portable Trusted Execution Environment. URL: https://www.op-tee.org/ (дата обращения: 14.01.2022).
18. Das U-Boot - the Universal Boot Loader. URL: http://www.denx.de/wiki/U-Boot/ (дата

обращения: 14.01.2022).

19. The Linux Kernel Organization. URL: https://www.kernel.org/ (дата обращения: 14.01.2022).
20. The Yocto Project. URL: https://www.yoctoproject.org/ (дата обращения: 14.01.2022).
21. STM32CubeIDE. Integrated Development Environment for STM32. URL: https://www.st.com/en/development-tools/stm32cubeide.html (дата обращения: 14.01.2022).
22. STM32 MPU Embedded Software distribution. URL: https://wiki.st.com/stm32mpu/wiki/Category:STM32MPU_Embedded_Software_distribution (дата обращения: 14.01.2022).
23. STM32 MPU Boot chain overview. URL: https://wiki.st.com/stm32mpu/wiki/Boot_chain_overview (дата обращения: 14.01.2022).
24. The DeviceTree Specification. URL: https://www.devicetree.org/ (дата обращения: 14.01.2022).
25. STM32MP15 device tree. URL: https://wiki.st.com/stm32mpu/wiki/STM32MP15_device_tree (дата обращения: 14.01.2022).
26. Which STM32MPU Embedded Software Package better suits your needs. URL: https://wiki.st.com/stm32mpu/wiki/Which_STM32MPU_Embedded_Software_Package_better_suits_your_needs (дата обращения: 14.01.2022).
27. How to configure ethernet interface. URL: https://wiki.st.com/stm32mpu/wiki/How_to_configure_ethernet_interface (дата обращения: 14.01.2022).
28. Package repository for OpenSTLinux distribution. URL: https://wiki.st.com/stm32mpu/wiki/Package_repository_for_OpenSTLinux_distribution (дата обращения: 14.01.2022).
29. Ethernet device tree configuration. URL: https://wiki.st.com/stm32mpu/wiki/Ethernet_device_tree_configuration (дата обращения: 14.01.2022).
30. KSZ8081 10/100 Base-T/TX Physical Layer Transceiver. Microchip Technology Inc. URL: https://www.microchip.com/en-us/product/KSZ8081 (дата обращения: 14.01.2022).
31. Future Technology Devices International Ltd. URL: https://ftdichip.com/ (дата обращения: 14.01.2022).
32. VCP Drivers. FTDI. URL: https://ftdichip.com/drivers/vcp-drivers/ (дата обращения: 14.01.2022).
33. Zadig. URL: https://zadig.akeo.ie/ (дата обращения: 14.01.2022).
34. STMicroelectronics. URL: https://www.st.com/ (дата обращения: 14.01.2022).
35. AN4989 Application note. STM32 microcontroller debug toolbox. STMicroelectronics.
36. Platunov A., Kluchev A., Pinkevich V., Kluchev V., Kolchurin M. Training laboratories with online access on the ITMO.cLAB platform // CEUR Workshop Proceedings - 2020, Vol. 2893.
37. Pinkevich V., Platunov A., Kluchev A. How to Improve IT Specialists Training for Designing Cyber-Physical Systems // 10th Mediterranean Conference on Embedded Computing, MECO 2021 - Proceedings - 2021, pp. 9460181.
38. Baranov I., Platunov A., Kluchev A., Kluchev V. et al. Automated Training Laboratory Bench

for Studies of Thermophysical Properties and Thermal Processes Based on a Programmable Controller SDK-1.1M//CEUR Workshop Proceedings, 2020, Vol. 2590, pp. 1-8.

39. Шаблон проекта STM32CubeIDE для SDK-1.1M для загрузки в ITMO.cLAB. URL: https://github.com/lmtspbru/SDK_cLAB (дата обращения: 14.01.2022).
40. Примеры программирования стенда-конструктора SDK-1.1M на базе микроконтроллера STM32. URL: https://github.com/lmtspbru/SDK-1.1M (дата обращения: 14.01.2022).
41. STM32F427VI microcontrollers. STMicroelectronics. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f427vi.html (дата обращения: 14.01.2022).
42. UM1725 User manual. Description of STM32F4 HAL and low-layer drivers. STMicroelectronics.
43. PM0214 Programming manual. STM32 Cortex-M4 MCUs and MPUs programming manual. STMicroelectronics.

Ключев Аркадий Олегович
Пинкевич Василий Юрьевич
Платунов Алексей Евгеньевич
Ключев Владислав Аркадьевич

Стенд-конструктор SDK-1.1M. Организация и программирование микроконтроллеров

Учебное пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А