Ключарёв А. А., Кочин К. А.,
Фоменкова А. А.

Программирование микроконтроллеров STM32

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

А. А. Ключарёв, К. А. Кочин, А. А. Фоменкова

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ STM32

Учебное пособие

ГУАП
Санкт-Петербург
2023

УДК 004.438
ББК 32.973.26
К52

Рецензенты:
кандидат технических наук, доцент *В. И. Исаков;*
доктор технических наук, профессор *Е. В. Копкин*

Утверждено
редакционно-издательским советом университета
в качестве учебного пособия

Протокол № 6 от 29 ноября 2022 г.

Ключарёв, А. А.

К52 Программирование микроконтроллеров STM32: учеб. пособие / А. А. Ключарёв, К. А. Кочин, А. А. Фоменкова. – СПб.: ГУАП, 2023. – 196 с.

ISBN 978-5-8088-1829-3

Приводится подробное описание работы с наиболее часто используемыми средствами разработки программного обеспечения для встраиваемых устройств, рассматриваются особенности стандартных средств разработки программного обеспечения под микроконтроллеры семейства STM32.

Учебное пособие предназначено для изучения теоретического и практического материала в рамках дисциплины «Программирование встраиваемых приложений» студентами, обучающимися по направлению подготовки 09.03.04 «Программная инженерия» и может быть использовано при освоении аналогичной дисциплины студентами инженерных направлений.

УДК 004.438
ББК 32.973.26

ISBN 978-5-8088-1829-3

ВВЕДЕНИЕ

Учебное пособие предназначено, в первую очередь, для студентов, обучающихся по направлениям, связанным с использованием информационных технологий и программированием, и содержит специальный материал, касающийся особенностей разработки приложений для встраиваемых устройств.

Логически пособие состоит из трех частей. В разделах 1–2 приведен обзор инструментальных средств разработки программного обеспечения для встраиваемых устройств. В разделах 3–8 рассматриваются различные аспекты создания программ с помощью стандартных средств разработки программного обеспечения под микроконтроллеры семейства STM32. В конце каждого раздела даны задания для практического освоения приведенного материала, сформулированы варианты для индивидуального выполнения.

В приложениях приведены вспомогательные программы и наборы данных для выполнения индивидуальных заданий.

СПИСОК СОКРАЩЕНИЙ

IDE	Integrated Development Environment
ARM	Advanced Risc Machine
API	Application Programming Interface
HAL	Hardware Abstraction Layer
GPIO	General-Purpose Input/Output
NVIC	Nested Vectored Interrupt Controller
EXTI	Extended Interrupts And Events Controller
UART	Universal Asynchronous Receiver-Transmitter
ШИМ	Широтно-импульсная модуляция

1. ИНСТРУМЕНТАРИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПОД STM32

Рассмотрим наиболее часто используемые средства разработки программного обеспечения для встраиваемых устройств. В общем случае их можно разделить на следующий категории:

– компиляторы;

– системы сборки;

– системы управления зависимостями;

– утилиты для загрузки программ и отладки программ на микроконтроллерах;

– библиотеки, фреймворки, драйвера, встраиваемые операционные системы;

– интегрированная среда разработки;

– прочие вспомогательные утилиты.

Некоторые утилиты независимы друг от друга и гибко конфигурируются, другие же могут быть жестко привязаны к конкретной среде разработки, что упрощает первоначальную установку и настройку, но усложняет интеграцию со сторонними элементами.

1.1. Компиляторы

В настоящее время наиболее распространены и популярны следующие компиляторы C/C++ для микроконтроллеров, основанных на ARM архитектуре:

1. Arm GNU Toolchain [1] – GCC компилятор для ARM. Данный компилятор является свободным программным обеспечением, поставляется в комплекте со многими IDE, поддерживает последние стандарты C/C++ и хорошо оптимизирован, однако его стандартная библиотека C имеет более объемный код по сравнению с другими. Этот недостаток преодолевается путем замены ряда функций на более оптимальные [2].

2. IAR [3] – платный компилятор от IAR Systems, производящий наиболее маленький и быстрый код, но в настоящее время разница с рассматриваемым выше компилятором GCC незначительна [4, 5]. В отличие от GCC имеет лучшую поддержку других архитектур (8051, PIC).

3. ARM Compiler Version 5 [6] – платный компилятор от компании Keil. В настоящее время считается устаревшим и заменен 6-й версией.

4. Компиляторы, основанные на Clang [7]. Ввиду высокой сложности языка C++ и его новых стандартов, многие разработчики

компиляторов для встраиваемых систем решили отказаться от своих разработок в пользу создания бэкенда для Clang, что позволяет получить хорошую поддержку новых стандартов C/C++ без значительных затрат:

– Clang имеет встроенную поддержку cortex-m ядер и может напрямую использоваться для компиляции программ [8];

– ARM Compiler Version 6 [9] – платный компилятор от Keil, основанный на Clang. Но в комплекте с некоторыми IDE его можно использовать бесплатно [10].

Все приведенные компиляторы поддерживают работу с микроконтроллерами STM32, при правильных опциях компиляции размер и скорость итоговой прошивки различается незначительно, поэтому выбор компилятора обычно диктуется средой разработки.

1.2. Интегрированные среды разработки

Среды разработки, сборки и управления зависимостями при программировании микроконтроллеров обычно тесно взаимосвязаны. Поэтому далее рассмотрим их совместно.

1. **Arduino** представляет собой открытую платформу и набор инструментов для разработки встраиваемых приложений. Включает следующие основные части:

– Arduino core – сам фреймворк, поддерживающий большое количество микроконтроллеров и предоставляющий базовый простой API для аппаратных интерфейсов (GPIO, UART, Timer, I2C, SPI и др.). Данный API не использует все возможности микроконтроллера, но за счет простоты позволяет реализовать библиотеки и драйвера, которые легко подключать в свои проекты.

– Aduino IDE [11] – простая IDE для разработки под Arduino.

– Arduino-cli [12] – консольная утилита для сборки проекта и установки библиотек, позволяющая настраивать и автоматизировать сборку проекта.

Arduino имеет широкий набор библиотек и драйверов для разнообразных сенсоров, поскольку она предоставляет простое низкоуровневое API. Данная платформа владеет возможностям автоматического импорта библиотек, соответствующих ее стандартному формату [13], поддерживает транзитивные зависимости и простую установку библиотек через IDE. Дополнительно большинство библиотек для сенсоров и других устройств автоматически выполняют минимальную конфигурацию сенсора/устройства, что дает возможность использовать сенсор сразу, без чтения документации к нему.

Из недостатков Arduino можно отметить крайне простую IDE без возможности интерактивной отладки, ограниченное API для работы с аппаратурой и невозможность указания списка зависимостей для конечного проекта.

2. **PlatformIO** – кроссплатформенный фреймворк с отрытым исходным кодом для создания embedded приложений [14]. В отличие от других фреймворков, не имеет своего API, а вместо этого предлагает использовать другие фреймворки (Arduino, MbedOS, STM32CubeMX), занимается их установкой и разрешением зависимостей. В дополнение имеет плагины для VSCode и CLion для полноценной embedded разработки. Для описания проекта PlatformIO использует «platformio.ini» – файл в корневой директории, который содержит описание используемых фреймворков, библиотек и т. д. В отличие от настроек IDE (.vscode, .idea, .settings, .cproject и т. д.), он не содержит специфичных для локальной среды и IDE опций, что позволяет легко переносить проект между разными машинами, IDE и версиями. Ввиду хорошей совместимости с Arduino (библиотеки и проекты) становится популярным, в том числе и для Arduino разработки.

3. **STM32CubeMX** – вспомогательная утилита для создания и настройки STM32 проектов для IDE [15]. Многие интегрированные среды разработки для встраиваемых устройств имеют только свои конфигурационные файлы и не поддерживают сторонние системы сборки (cmake и др.). Из-за этого, в общем случае, необходимо с нуля создавать проект, добавлять в него все файлы вручную, и указывать нужные макросы и флаги компиляции. Это заметно усложняет создание даже простых проектов. Поэтому, для упрощения создания проектов и их конфигурации, STMicroelectronics создало и поддерживает STM32CubeMX. Основными преимуществами STM32CubeMX являются:

– поддержка всех семейств и официальных плат микроконтроллеров STM32;

– простой интерфейс для настройки тактовых частот микроконтроллера;

– автоматическое добавление драйверов для работы с периферией в проект;

– возможность настройки периферийных интерфейсов (SPI, таймеры и др.) с помощью графического интерфейса и автоматическая генерация кода инициализации, что сглаживает недостатки официального API драйверов;

– наличие готовых примеров программ;

– возможность добавления дополнительных библиотек и драйверов сенсоров;

– поддержка популярных IDE для встраиваемых систем;

– другие популярные IDE (CLion, PlatformIO, VSCode) имеют плагины или встроенную интеграцию с STM32CubeMX.

Из недостатков STM32CubeMX отметим:

– отсутствие совместимости между разными версиями STM32 CubeMX и библиотек, возможна миграция только на более новую версию;

– примеры программ имеются, но не всегда работают;

– количество драйверов для внешней периферии заметно меньше, чем для Arduino.

В связке с STM32CubeMX можно использовать следующие IDE:

– STM32CubeIDE – официальная IDE от STMicroelectronics, основанная на Eclipse (будет подробно рассмотрена в следующих главах этого пособия);

– EWARM (IAR Embedded Workbench for Arm) – среда от IAR Systems с советующим компилятором;

– MDK-ARM – платная IDE от компании Keil [16], поставляется с компиляторами ARMCC 5 и ARMCC 6, имеет многообразные и развитые средства для отладки микроконтроллеров, однако в качестве среды разработки имеет функциональность на уровне базового редактора текста: отсутствует автоформатирование кода, какие-либо возможности рефакторинга (переименование переменных и т. д.).

4. **CLion** – современная, платная среда для разработки C/C++ приложений [17]. Имеет интеграцию с STM32CubeMX и поддержку OpenOCD для загрузки и отладки программ на микроконтроллерах. Отличается хорошей поддержкой C/C++ кода (встроенный статический анализатор, возможности рефакторинга и автодополнения).

5. **VSCode** – современная IDE с отрытым исходным кодом [18]. Имеет хорошую поддержку C/C++ кода, а также интеграцию с STM32CubeMX через PlatformIO или другие плагины.

2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МИКРОКОНТРОЛЛЕРОВ STM32

В дальнейшем изложении и приведенных примерах в пособии будет использоваться следующее программное обеспечение:

1. STM32CubeMX, версия 6.6. Является вспомогательной программой для создание базовой заготовки проекта. Включает в себя следующую функциональность:

– настройка структуры и опций проекта, компилятора и линковщика для одной из заданных сред разработки;

– подключение библиотек для работы с периферией микроконтроллера;

– создание базовой заготовки проекта и кода инициализации микроконтроллера в соответствии с выбранными настройками.

2. STM32CubeIDE, версия 1.10. Бесплатная сборка Eclipse IDE для микроконтроллеров STM32. Поставляется с GCC компилятором и встроенными утилитами для загрузки программы на микроконтроллер и отладки. Создание проекта в дальнейшем изложении выполняется в два этапа:

– инициализация и настройка проекта в STM32CubeMX, экспорт проекта в выбранную IDE;

– реализация кода, загрузка программы на микроконтроллер и отладка в выбранной IDE.

В качестве аппаратного обеспечения будет предполагаться использование набора STM32F3Discovery [19].

2.1. Создание базового проекта для STM32F3DISCOVERY

Базовая настройка проекта в STM32CubeMX. Для настройки и инициализации проекта в STM32CubeMX необходимо выполнить следующие действия:

1. Откройте STM32CubeMX и нажмите «ACCESS TO BOARD SELECTOR».

2. На вкладке «Board Selector» укажите в фильтрах тип платы «Discovery Kit» и серию «STM32F3». В отфильтрованном списке плат выделите «STM32F3DISCOVERY» и нажмите «Start Project».

3. Подтвердите инициализацию периферии по умолчанию.

4. После этого вы увидите окно для настройки режимов пинов микроконтроллера (рис. 2.1).

5. Кликните на пин PE2 и в появившемся выпадающим списке выберите «Reset_State». Аналогично сбросьте настройки пинов PE0, PE1, PE3, PE4, PE5, PA5, PA6, PA7, PA11, PA12, PB6, PB7. Они подключены к периферии, которая не нужна для простых примеров. Их отключение упростит проект (рис. 2.2).

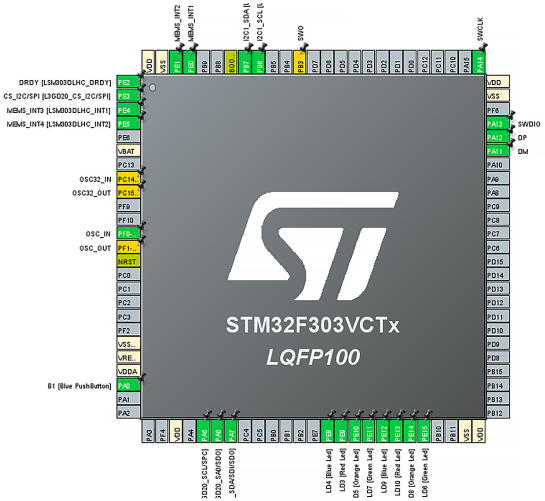


Рис. 2.1. Окно для настройки режимов пинов микроконтроллера

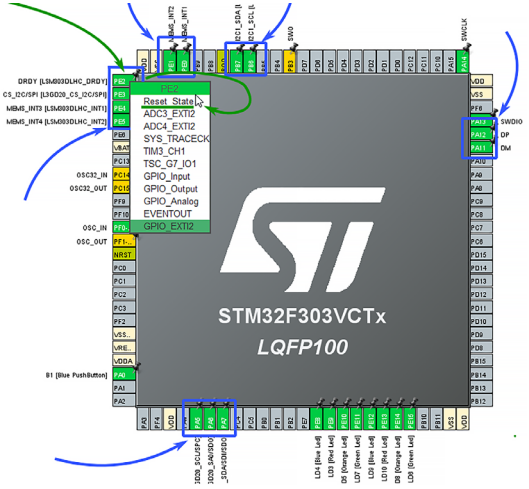


Рис. 2.2. Настройка режимов пинов микроконтроллера

6. В итоге распиновка должна выглядеть следующим образом (рис. 2.3).

7. Перейдите во вкладку «Project Manager» → «Project». Выберите имя и расположение папки проекта. Имя и путь не должны содержать пробелы и буквы, кроме латинских. Выберите «STM32CubeIDE» в качестве целевой IDE.

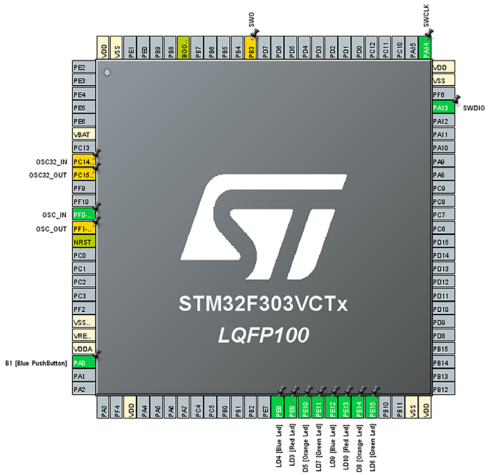


Рис. 2.3. Итоговая распиновка микроконтроллера

8. Откройте вкладку «Code Generation» и выберите опции «Copy only the necessary library files» и «Generate peripheral initialization as pair of '.c/.h' files per peripheral».

9. Сохраните проект и нажмите «GENERATE CODE» для инициации проекта.

По окончании генерации проекта вам будет приложено открыть папку в выбранной IDE. Нажмите «Open Project».

Базовая настройка проекта в STM32CubeIDE:

1. При открытии проекта среда может попросить выбрать расположение папки рабочей области. Она хранит общие настройки группы проектов. Оставьте значение по умолчанию и нажмите «Launch».

2. Закройте стартовое окно, если оно открыто (рис. 2.4).

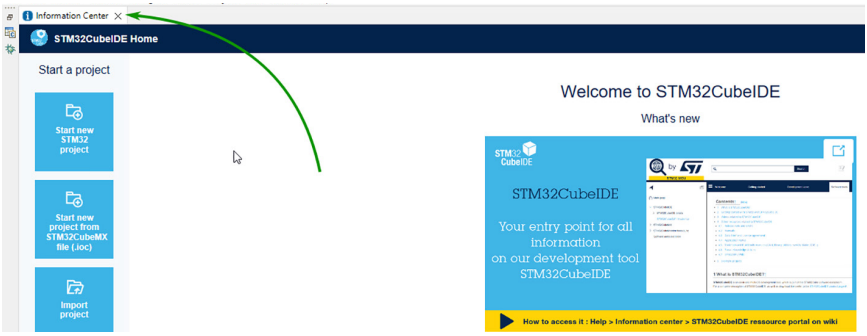


Рис. 2.4. Стартовое окно STM32CubeIDE

3. Перед вами появится окно проекта. Справа в «Project Explorer» вы увидите файлы проекта, а в центре окно редактирования кода. Откройте файл «Core/Src/main.c»

4. Замените код функции `main` следующим:

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration--------------------------------*/
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */
    uint8_t led_state = 0x03;

    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */
        HAL_GPIO_WritePin(GPIOE, 0xFF00, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOE, led_state << 8, GPIO_PIN_
SET);
        HAL_Delay(200);

        led_state = led_state >> 1 | led_state << 7;
        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

5. Скомпилируйте код и убедитесь, что нет никаких ошибок.

6. Нажмите на «Run» для загрузки программы на микроконтроллер.

7. При первом запуске появится окно настройки конфигурации запуска. Просто нажмите «OK».

8. Если микроконтроллер подключен к компьютеру через отладчик STLink, то загрузка должна завершиться без ошибок в окне «Console» (рис. 2.5).

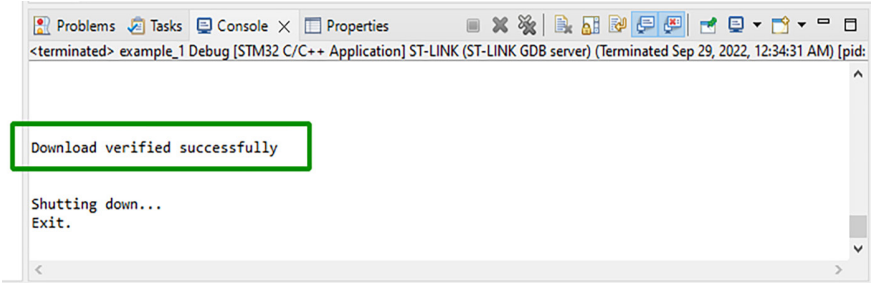


Рис. 2.5. Завершение загрузки микроконтроллера

9. Если все сделано правильно, то на круге из светодиодов будет отражаться анимация вращения двух светодиодов.

2.2. Базовая работа в STM32CubeIDE

Отметим следующие особенности и возможности работы с проектом:

1. STM32CubeIDE поддерживает автоматическое форматирование кода. Для этого необходимо выбрать «Source» → «Format» при открытом файле.

2. В случае, если вы случайно закрыли часть окон интерфейса, настройки можно вернуть назад через «Windows» → «Perspective» → «Reset Perspective ...» (рис. 2.6).

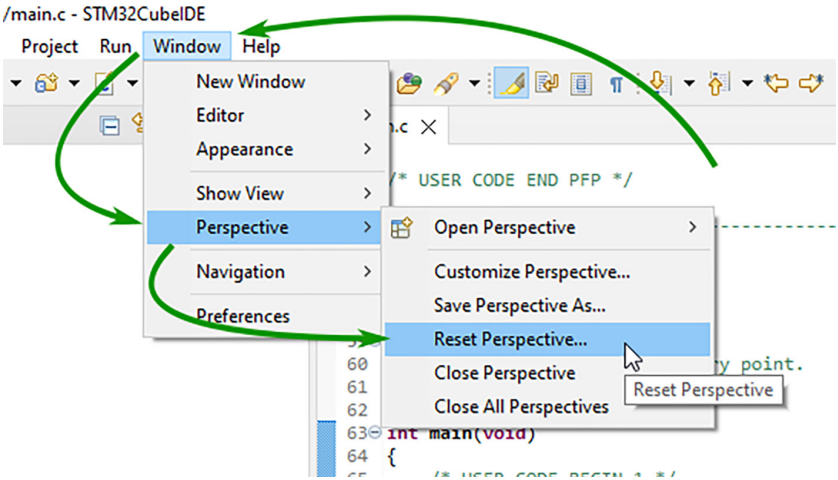


Рис. 2.6. Сброс настроек интерфейса STM32CubeIDE

3. IDE поддерживает отладку на микроконтроллере, как и для обычных программ с точками останова и просмотром локальных переменных. Единственное ограничение – максимальное количество установленных точек останов – 6. Это ограничение обусловлено аппаратной поддержкой точек останова и конкретными микроконтроллерами.

2.3. Структура базового проекта

Особенности STM32CubeMX. После создания проекта в сгенерированных файлах можно увидеть большое число комментариев типа «USER CODE BEGIN» «USER CODE END»:

```
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */
```

Свой код в сгенерированных файлах рекомендуется размещать именно между этими комментариями. Это позволяет перегенерировать проект из CubeMX без потери своего кода (весь код между данными комментариями сохраняется, а вне – затирается). Это позволяет:

– добавить в проект новое оборудование или изменить настройки существующего из CubeMX для текущего проекта;

– использовать код существующих проектов для создания нового.

В файлах, добавленных в проект самостоятельно, подобных ограничений с комментариями нет.

Обработка ошибок. Обычно программы для микроконтроллеров пишутся на языке «C» или «C++» с отключенными исключениями. Поэтому все ошибки необходимо обрабатывать явно. Общепринятым стилем сигнализации об ошибке в функции является возврат кода ошибки как целого числа. Если код нулевой, ошибки нет. Любое другое значение обычно означает ошибку:

```
int some_fun(int x, int y) {

    // some code

}

int main() {

    // some code
```

```
    int err;
    err = some_fun(3, 5);
    if (err) {
         // process error somehow

    }

    // normal workflow

}
```

Если восстановление состояния программы при ошибке не предусмотрено, обычно вызывается функция с вечным циклом, которая может мигать красным светодиодом. Это позволяет легко найти место ошибки при отладке программы. В случае с программой от CubeMX, по соглашению эта функция называется `Error_Handler`. По умолчанию она находится в файле main.c без реализации, поэтому имеет смысл добавить в нее следующий код:

```
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    HAL_GPIO_WritePin(GPIOE, 0xFF00, GPIO_PIN_RESET);
    while (1) {
         HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_9); // switch
red led
        HAL_Delay(100);
    }
    /* USER CODE END Error_Handler_Debug */
}
```

и вызывать данную функцию при обнаружении «фатальных» ошибок в вашей программе (ошибки инициализации и т. д.).

Структура программы в функции main. Обычно программы на микроконтроллере работают, начиная от старта программы и заканчивая физическим отключением питания. Ввиду этого типовая программа без использования операционной системы состоит из 2 основных частей:

– инициализация периферии и вспомогательных структур данных;

– вечный цикл, проверяющий внешние события и реализующий бизнес-логику программы.

Например:

```
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */
    /* MCU Configuration--------------------------*/
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */
    uint8_t led_mask = 0x01;
    /* USER CODE END 2 */

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

        /* USER CODE BEGIN 3 */
        // show current state
        HAL_GPIO_WritePin(GPIOE, 0xFF00, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOE, led_mask << 8, GPIO_PIN_SET);
        HAL_Delay(100);
        // update led mask
         led_mask = (led_mask << 1) | (led_mask & 0x80 ?
0x00 : 0x01);
    }
    /* USER CODE END 3 */
}
```

Отметим следующие особенности CubeMX:

– пользовательский код инициализации лучше разместить между «USER CODE BEGIN 2» и «USER CODE END 2»;

– код обработки событий лучше разместить между «USER CODE BEGIN 3» и концом цикла.

3. ПОРТЫ ВВОДА-ВЫВОДА ОБЩЕГО НАЗНАЧЕНИЯ (GPIO)

3.1. Общие сведения о портах ввода-вывода

Микроконтроллеры серии STM32 обладают большим числом пинов. Например, микроконтроллер платы STM32F3Discovery имеет 100 пинов, из которых 87 можно использовать как цифровые входы/выходы [20] (рис. 3.1).

Каждый пин может быть сконфигурирован в качестве [21]:

– цифрового выхода;

– цифрового входа;

– аналогового входа;

– выхода/входа периферии (таймер, шина SPI, I2S и т. д).

Общая схема пина [22], с учетом перечисленных режимов, показана на рис. 3.2.

Для более эффективного управления пинами они объединены в группы, называемыми портами. У STM32 каждый порт содержит 16 пинов. В программах и документации они обозначаются как GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOE.

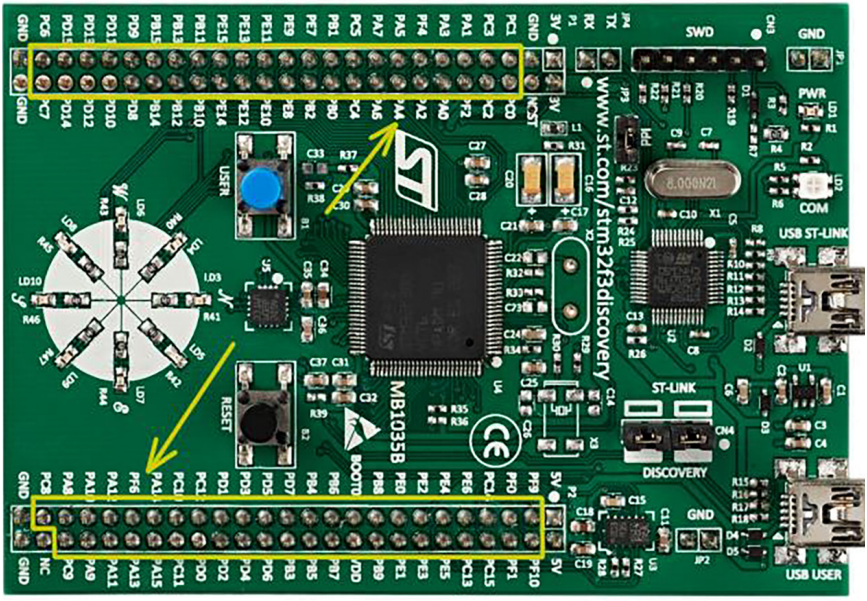


Рис. 3.1. Выводы GPIO на STM32F3Discovery

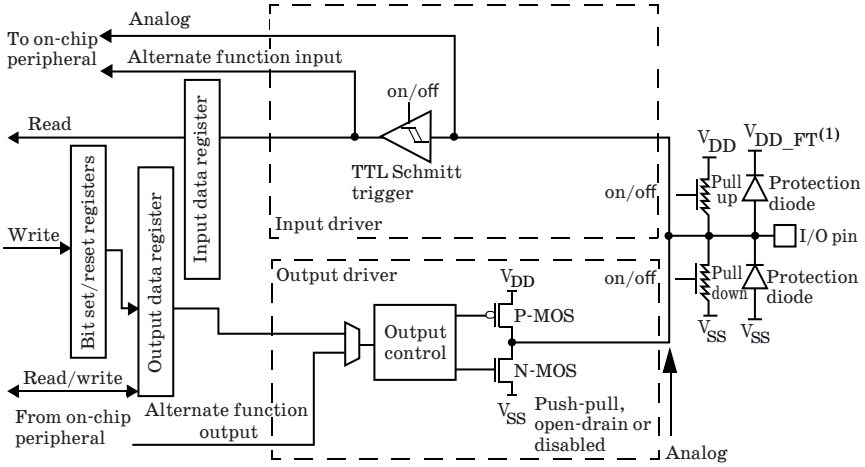


Рис. 3.2. Базовая схема пина ввода-вывода STM32F3

Общая схема работы с портами ввода-вывода следующая [23]:

1) инициализация пинов в нужном GPIO режиме с помощью CubeMX;

2) включение/выключение/чтение состояния пина при помощи функций: `HAL_GPIO_WritePin`, `HAL_GPIO_ReadPin`, `HAL_GPIO_TogglePin`.

3.2. Библиотеки HAL. Базовый API

Для упрощения использования возможностей оборудования и переносимости программ между семействами микроконтроллеров, STmicroelectronics разрабатывает и официально поддерживает библиотеку для работы с периферией HAL (Hardware Abstraction Layer) [24]. Рассмотрим наиболее базовый API, который понадобится для первой программы на STM32.

Функции работы с временем. Для базовой работы с временем нам понадобятся две функции:

1) `void HAL_Delay(uint32_t Delay)` – обеспечивает паузу на заданное количество миллисекунд;

2) `uint32_t HAL_GetTick(void)` – возвращает количество миллисекунд, прошедших с начала работы программы на микроконтроллере.

Функции для работы с портами ввода. Для GPIO пина, сконфигурированного в режиме ввода, имеется следующая функция чтения его состояния:

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx,
uint16_t GPIO_Pin)
```

Аргументы:

– `GPIOx` – порт пина;

– `GPIO_Pin` – пин, который нужно проверить. На вход подается одна из констант `GPIO_PIN_0`, `GPIO_PIN_1`, …, `GPIO_PIN_15`.

Функция `HAL_GPIO_ReadPin` возвращает одно из следующих значений:

– `GPIO_PIN_SET` – пин находится в состоянии логической единицы (+3V);

– `GPIO_PIN_RESET` – пин находится в состоянии логического нуля (0V).

Пользовательская кнопка STM32F3Discovery. На плате имеется синяя пользовательская кнопка, состояние которой можно считывать с помощью `HAL_GPIO_ReadPin`. Она расположена на порте `GPIOA` и пине `GPIO_PIN_0`. При отжатом состоянии `HAL_GPIO_ReadPin` возвращает `GPIO_PIN_RESET`, при нажатом – `GPIO_PIN_SET`.

Работа с портами вывода. Для управления GPIO пинами, сконфигурированными как выходы, имеются следующие функции:

1. Функция включения/выключения заданных пинов:

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

Аргументы:

– `GPIOx` – порт пина;

– `GPIO_Pin` – пин или пины, которые нужно переключить. На вход подается одна из констант `GPIO_PIN_0`, `GPIO_PIN_1`, …, `GPIO_PIN_15` или их комбинация.

– `PinState` – желаемое состояние указанных в `GPIO_Pin`. Либо `GPIO_PIN_SET`, либо `GPIO_PIN_RESET`.

2. Функция переключения указанных пинов на противоположное состояние:

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

Аргументы:

– `GPIOx` – порт пина;

– `GPIO_Pin` – пин или пины, которые нужно переключить. На вход подается одна из констант `GPIO_PIN_0`, `GPIO_PIN_1`, …, `GPIO_PIN_15` или их комбинация.

Аргумент GPIO_Pin. Данный аргумент является битовой маской, где:

– номер пина соответствует номеру бита в маске;

– если состояния пина нужно изменить, то соответствующий бит должен иметь значение 1, иначе – 0.

Как показано на примере в табл. 3.1, для указания нескольких пинов можно через операцию побитового ИЛИ «|» объединить существующие константы GPIO_PIN_X или напрямую записать битовую маску в шестнадцатеричной форме.

Таблица 3.1

Примеры битовых масок

Пример	Шестнадцатеричное число или константа	Двоичная маска
Выделить нулевой пин	GPIO_PIN_0	0000_0000_0000_0001
Восьмой пин	GPIO_PIN_8	0000_0001_0000_0000
Выделить пятнадцатый и десятый пины	GPIO_PIN_15 | GPIO_PIN_10	1000_1000_0000_0000
Выделить пины с восьмого по пятнадцатый	0xFF00	1111_1111_0000_0000

Светодиоды на плате STM32F3Discovery. Отладочная плата имеет восемь светодиодов, подключенных к портам ввода-вывода. Светодиоды расположены на порте GPIOE на пинах с 8-го по 15-й (табл. 3.2).

Таблица 3.2

Подключение светодиодов к порту ввода-вывода

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LED	LED	LED	LED	LED	LED	LED	LED								

3.3. Пример программы для работы с портами ввода-вывода

Ниже приведен фрагмент кода (только функция `main`) программы, которая показывает два вращающийся по кругу светодиода периодом обновления 200 мс. При нажатии на кнопку, направление вращения мгновенно изменяется, а задержка обновления сокращается до 50 мс.

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration--------------------------*/
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */
    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    /* USER CODE BEGIN 2 */
    uint8_t led_mask = 0x03;
    int animation_delay_up = 200;
    int animation_delay_down = 50;
    int update_state_downcount = 0;
    int btn_state = GPIO_PIN_RESET;
    int prev_btn_state = GPIO_PIN_RESET;
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
        // read button state
        btn_state = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
        if (btn_state != prev_btn_state) {
            // force led update
            update_state_downcount = 0;
        }
        prev_btn_state = btn_state;
        // check led update downcount
        update_state_downcount--;
        if (update_state_downcount < 0) {
            // update delay counter
            if (btn_state == GPIO_PIN_SET) {
                update_state_downcount = animation_delay_
down;
```

```
            } else {
             update_state_downcount = animation_delay_up;
            }
            // update leds
            if (btn_state == GPIO_PIN_SET) {
                // left "led" rotation
                led_mask = led_mask << 1 | led_mask >> 7;
            } else {
                // right "led" rotation
                led_mask = led_mask >> 1 | led_mask << 7;
            }
            // disable all leds
              HAL_GPIO_WritePin(GPIOE, 0xFF00, GPIO_PIN_
RESET);
            // enable leds according led_mask
            HAL_GPIO_WritePin(GPIOE, led_mask << 8, GPIO_
PIN_SET);
        }
        // loop delay
        HAL_Delay(1);
    }
    /* USER CODE END 3 */
}
```

Выделим следующие особенности программы:

– Для удобства использования битовых операций, использована 8-битная базовая маска светодиодов. Она преобразуется в 16-битную со светодиодами на 8–15 позициях только перед применением функции `HAL_GPIO_WritePin`.

– В цикле имеется только одна задержка на 1 мс. Для организации более длительных задержек используются вспомогательные счетчики. Это позволяет опрашивать кнопку с частотой 1 кГц и мгновенно менять направление движения светодиодов.

3.4. Практические задания к разделу 3

В соответствии с вариантом (табл. 3.3) нужно реализовать три режима работы светодиодов. Переключение режимов происходит по нажатию на кнопку:

1) одинарный клик – первый режим;

2) двойной клик – второй режим;

3) тройной клик и более – третий режим.

Состояние кнопки следует обрабатывать с частотой 1 кГц, поэтому в основном цикле программы нельзя делать большие задержки (более 1 мс на одну итерацию основного цикла).

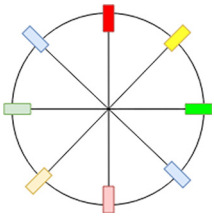
Таблица 3.3

Варианты индивидуальных заданий к разделу 3

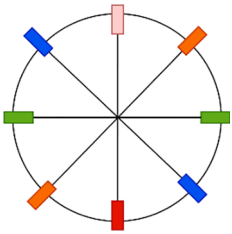
№ варианта	Индивидуальное задание
1	Реализовать режимы работы светодиодов 1, 9, 5
2	Реализовать режимы работы светодиодов 2, 10, 6
3	Реализовать режимы работы светодиодов 3, 11, 7
4	Реализовать режимы работы светодиодов 4, 12, 8
5	Реализовать режимы работы светодиодов 5, 1, 9
6	Реализовать режимы работы светодиодов 6, 2, 10
7	Реализовать режимы работы светодиодов 7, 3, 10
8	Реализовать режимы работы светодиодов 8, 4, 12
9	Реализовать режимы работы светодиодов 1, 5, 12
10	Реализовать режимы работы светодиодов 2, 7, 9

Режимы работы светодиодов

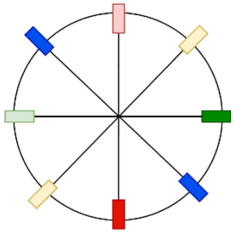
1. Включить 3 светодиода, как показано ниже, и вращать их по часовой стрелке с паузой между состояниями в 100 мс.



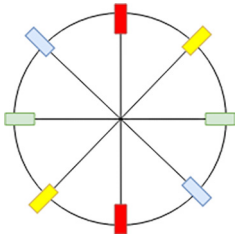
2. Включить 7 светодиодов, как показано ниже, и вращать их против часовой стрелки с паузой между состояниями в 200 мс.



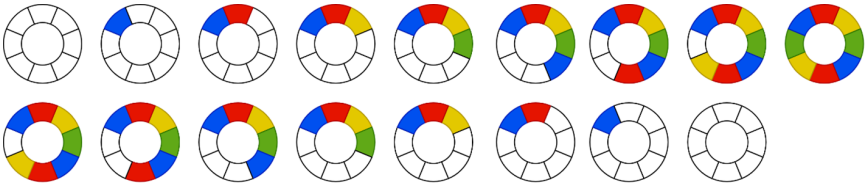
3. Включить 4 светодиода, как показано ниже, и вращать их против часовой стрелки с паузой между состояниями в 100 мс.



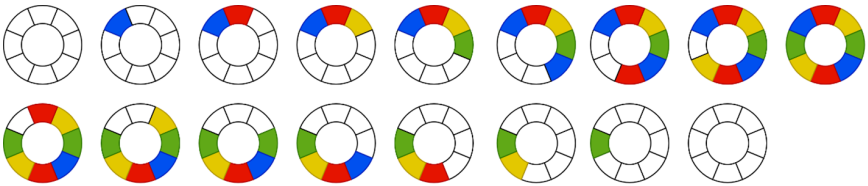
4. Включить 4 светодиода, как показано ниже, и вращать их по часовой стрелке с паузой между состояниями в 50 мс.



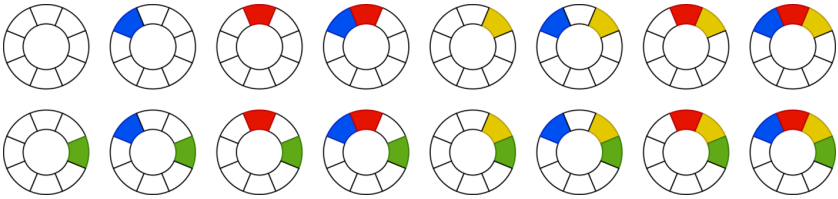
5. Организовать постепенное заполнение и опустошение светодиодного круга, как показано ниже. Задержка между двумя последовательными состояниями – 80 мс.



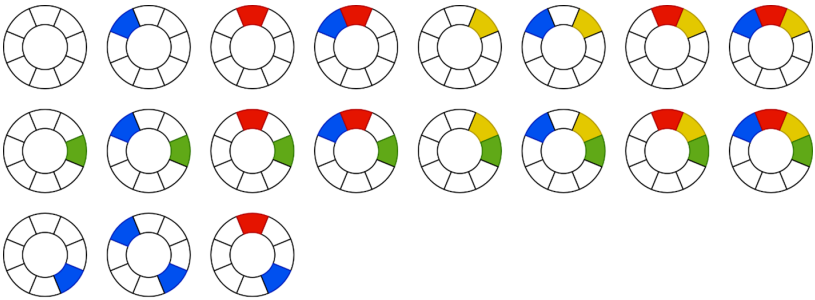
6. Организовать постепенное заполнение и опустошение светодиодного круга, как показано ниже. Задержка между двумя последовательными состояниями – 120 мс.



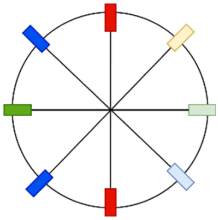
7. Использовать 8 светодиодов как двоичный счетчик для последовательного отображения чисел от 0 до 15. Задержка между двумя последовательными состояниями – 200 мс.



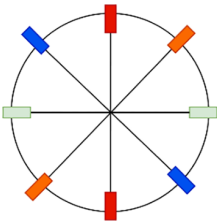
8. Использовать 8 светодиодов как двоичный счетчик для последовательного отображения чисел от 0 до 255. Задержка между двумя последовательными состояниями – 50 мс.



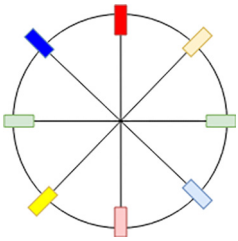
9. Включить 5 светодиодов, как показано ниже, и вращать их по часовой стрелке с паузой между состояниями в 120 мс.



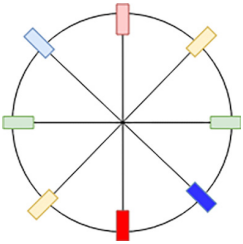
10. Включить 6 светодиодов, как показано ниже, и вращать их по часовой стрелке с паузой между состояниями в 100 мс.



11. Включить 3 светодиода, как показано ниже, и вращать их против часовой стрелки с паузой между состояниями в 120 мс.



12. Включить 2 светодиода, как показано ниже, и вращать их против часовой стрелки, с паузой между состояниями в 140 мс.



4. ОБРАБОТКА ВНЕШНИХ ПРЕРЫВАНИЙ

4.1. Основные сведения о прерываниях в STM32

Прерывание – это сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленной обработки. Реакцией процессора на прерывание является остановка текущей программы и вызов заранее подготовленной функции – обработчика прерываний [25].

Обработкой прерываний во всех ARM микроконтроллерах занимается NVIC (Nested Vectored Interrupt Controller) – контроллер приоритетных векторных прерываний. Данное устройство тесно интегрировано с ядром процессора и выполняет следующие функции [26]:

– вызов обработчиков прерывания по поступившему запросу;

– организация приоритетов прерываний;

– маскирование прерываний.

Обработчики представляют собой обычные С-функции, адреса которых записаны в массив, расположенный в начале flash-памяти. Этот массив называется вектором прерываний.

В STM32F3 доступно 16 уровней приоритетов прерываний, разбитых на две группы и, соответственно, отвечающих за приоритет вытеснения и субприоритет [21]. В зависимости от настроек можно использовать или обе части (например, выделить 2 бита приоритету вытеснения и 2 – субприоритету), или только одну.

Если во время обработки текущего прерывания появляется новое, с более высоким приоритетом вытеснения, то текущее будет приостановлено, и управление будет передано приоритетному (рис. 4.1). Если новое прерывание имеет тот же приоритет вытеснения, то процессор дождется завершения выполнения текущего и начнет выполнять более приоритетное, согласно субприоритету.

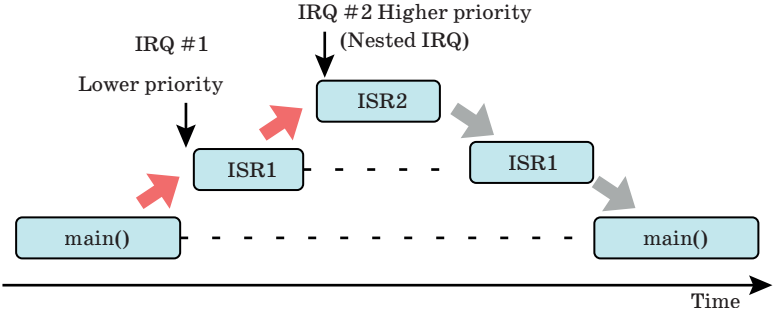


Рис. 4.1. Обработка прерываний с приоритетами

В дополнение к NVIC, в STM имеется дополнительный блок – EXTI (extended interrupts and events controller) [21]. Функционально он стоит между пином и NVIC (рис. 4.2).

Блок EXTI добавляет следующий функционал:

– детектирование переднего и заднего фронтов сигналов (рис. 4.3) (NVIC умеет детектировать только передние фронты);

– подача сигналов не только на NVIC, но и на другие устройства. Например, DMA, таймеры и др.

Общая схема NVIC представлена на рис. 4.4.

Рис. 4.2. Блок EXTI между пином и NVIC

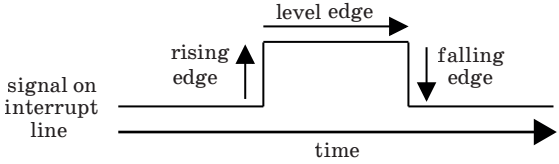


Рис. 4.3. Передний и задний фронты сигнала

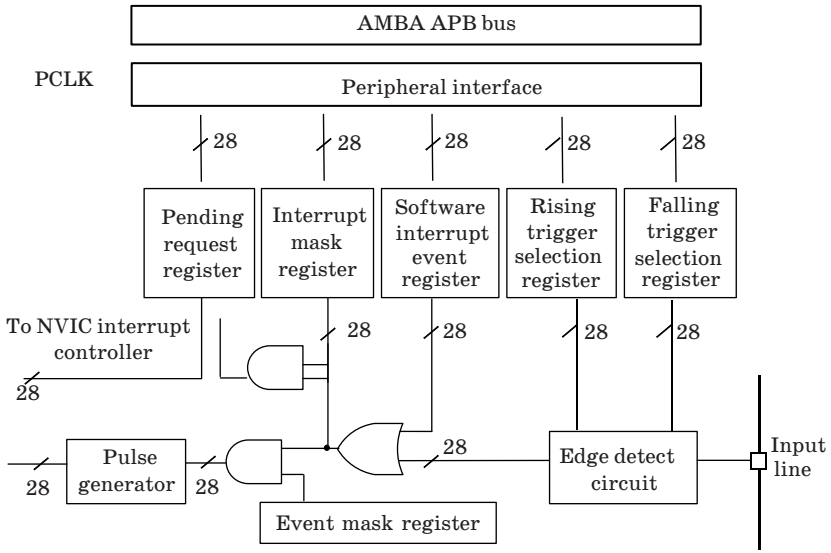


Рис. 4.4. Функциональная схема блока EXTI

EXTI имеет 36 линий, 16 из которых предназначены для внешних прерываний, происходящих при детектировании изменения уровня логического сигнала на пинах. Подключение пинов к линиям с помощью мультиплексоров показано на рис. 4.5.

Как показано на схеме (рис. 4.5), одновременно можно задействовать не более 16 внешний прерываний, при этом номера пинов должны быть разными. Так, например, можно одновременно использовать пины PA0 и PB1 для прерываний, но одновременно задействовать пины PA0 и PB0 не получится.

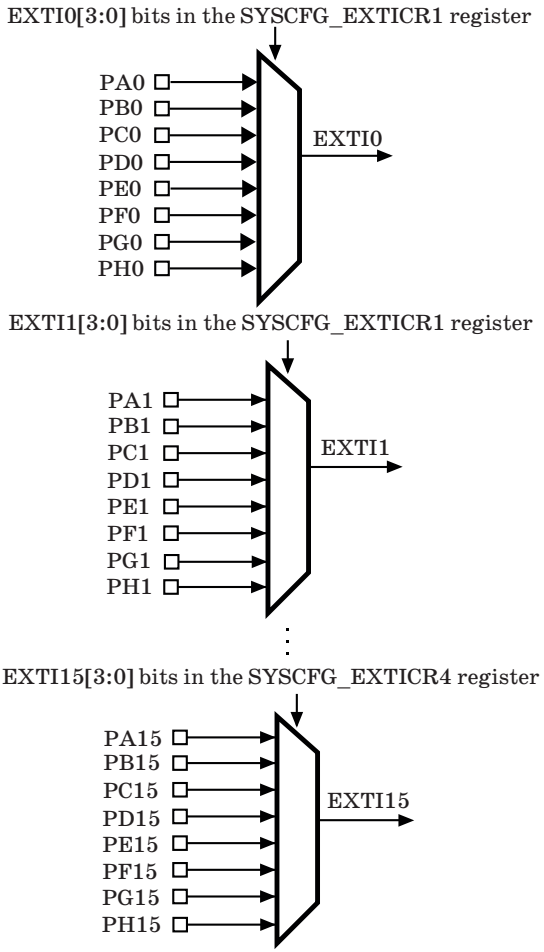


Рис. 4.5. Подключение пинов к линиям EXTI

4.2. Использование прерываний на примере матричной клавиатуры

Матричная клавиатура HC-35. В качестве источника внешних прерываний может быть использован блок матичной клавиатуры HC-35 (рис. 4.6). Ее принципиальная схема представлена на рис. 4.7.

В клавиатуре нас интересует матричная часть. В лабораторном макете секция матрицы клавиатуры подключена к младшему байту порта GPIO_C в соответствии с рис. 4.8.

Для работы с клавиатурой пины PC_0-PC_3 микроконтроллера должны быть инициализированы в режим приема внешних прерываний EXTI0-EXTI3 по переднему и заднему фронтам (Mode with Rising/Falling edge trigger) и установлены в режим «Pull down». Выходы PC_4-PC_7 сконфигурированы как цифровые выходы с высоким уровнем напряжения (GPIO output level == High), что обеспечивает

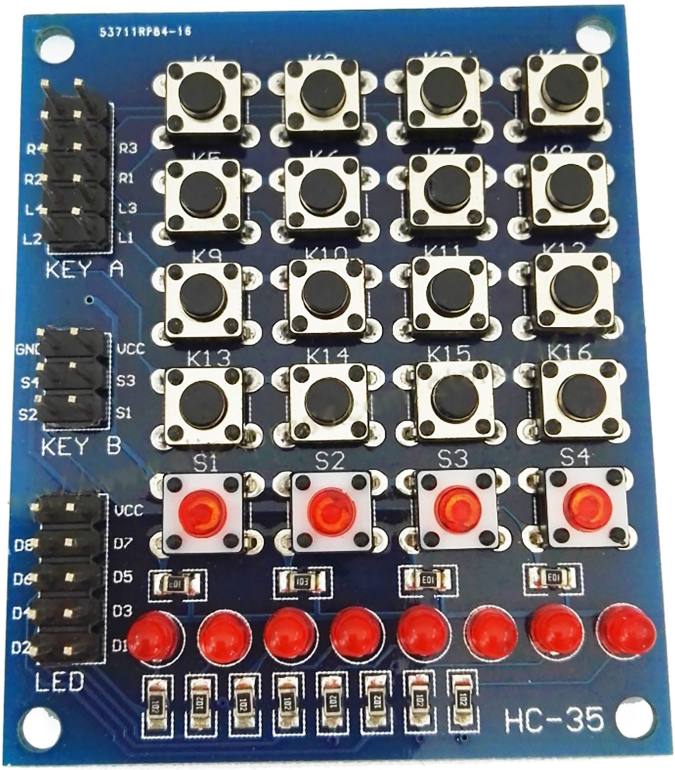
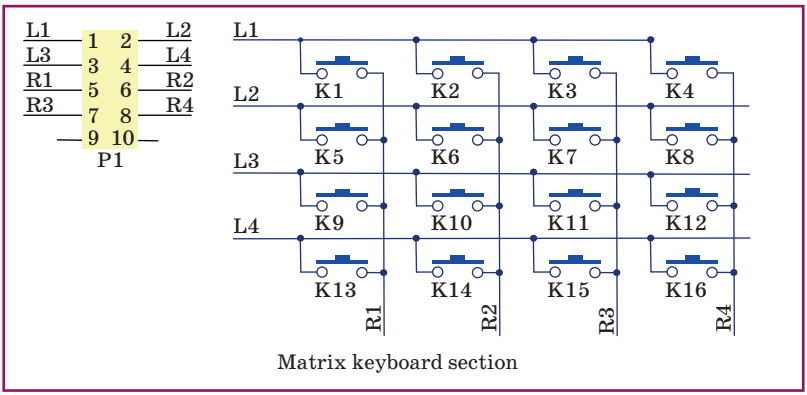
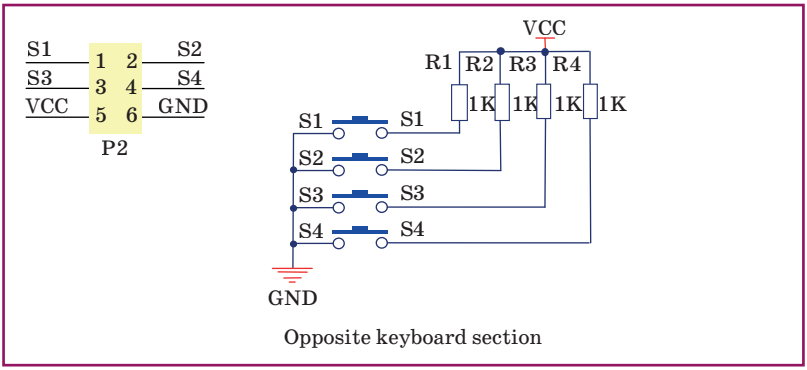


Рис. 4.6. Матричная клавиатура HC-35





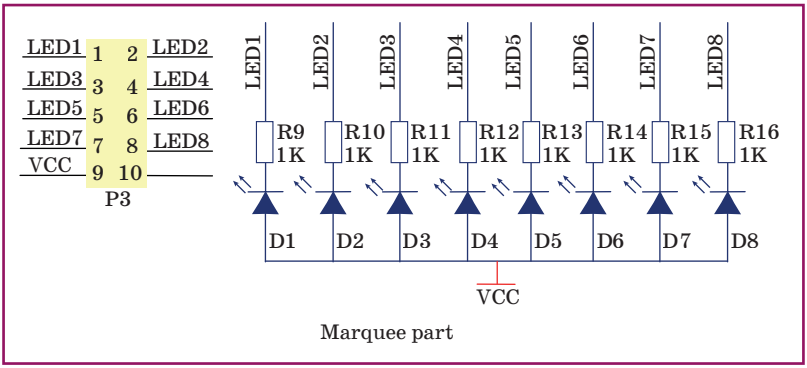


Рис. 4.7. Схема клавиатуры НС-35

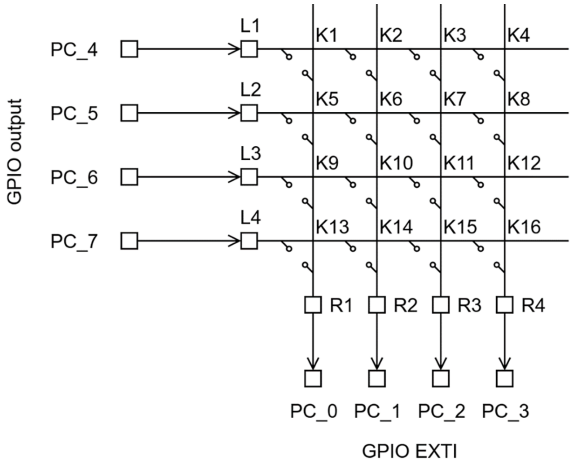


Рис. 4.8. Схема подключения матричной клавиатуры к лабораторному макету

высокий уровень напряжения на линиях L1-L4 клавиатуры. Таким образом, если ни одна кнопка не нажата, на входах PC0-PC3 установлен нулевой потенциал, соответствующий логическому «0». При нажатии кнопки на соответствующей линии R1-R4 устанавливается высокий уровень напряжения, соответствующий логической «1», что должно вызвать обработку прерывания.

Для определения номера нажатой клавиши можно использовать следующий алгоритм обработчика прерывания:

1. Определить пин *r*, с которого пришло прерывания. Этим действием мы найдем столбец, в котором была нажата/отпущена кнопка.

2. Проверить состояние пина *r*. Если на нем нет сигнала, значит кнопка была отпущена. В этом случае нужно сделать какое-либо действие и завершить обработку прерывания. Иначе продолжим определение номера нажатой кнопки.

3. Поочередно отключить и включить пины *L*1–*L*4, проверяя вход пина *r*, чтобы найти строку, в которой была нажата кнопка: если при отключении *i*-го пина сигнал с *r* пропадает, то кнопка была нажата в строке *i*.

4. Очистить бит прерывания *r*-го пина, так как действия предыдущего пункта инициировали данное прерывание снова.

5. На основании номера строки и столбца вычисляем номер кнопки и выставляем глобальную переменную с номером нажатой копки и фактом нажатия, после чего можем завершить обработку прерывания.

Настройка проекта в CubeMX

1. Создайте новый проект в CubeMX и выполните базовую конфигурацию проекта.

2. Перейдите на вкладку «Pinout & Configuration» и выберите для выходов PC0-PC3 режим внешнего прерывания GPIO_EXTI<N> (рис. 4.9).

3. Задайте режим цифрового выхода GPIO_Output для выходов PC4-PC7 (рис. 4.10).

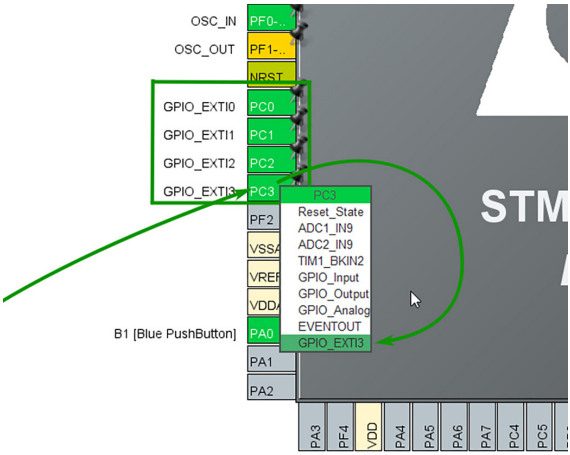


Рис. 4.9. Выбор режима внешнего прерывания

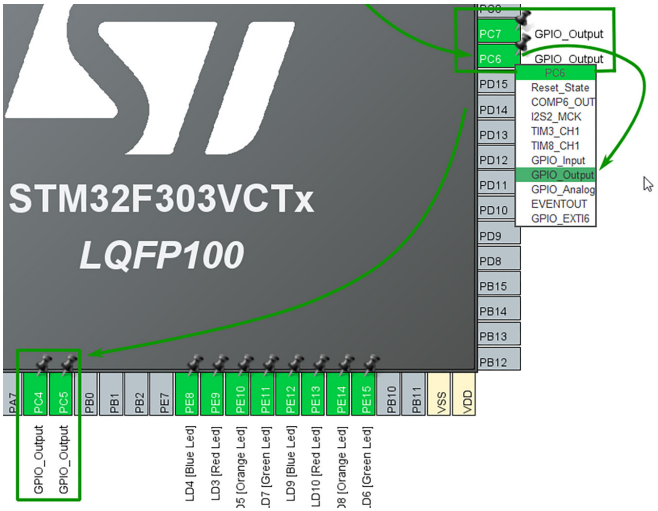


Рис. 4.10. Задание режима цифрового выхода

4. Раскройте список «System Core» на вкладке справа и выберите пункт «GPIO». В появившемся виджете «GPIO Mode and Configuration» откройте вкладку «GPIO» и задайте следующие параметры:

пины PC0-PC3:

GPIO mode: External Interrupt Mode with Rising/Falling edges trigger detection

GPIO Pull-up/Pull-down: Pull down

пины PC4-PC7:

GPIO output level: High

GPIO mode: Output Push Pull

GPIO Pull-up/Pull-down: No pull up pull down

Maximum output speed: Low

5. Вернитесь к списку «System Core» и выберите пункт «NVIC». В виджете «NVIC Mode and Configuration» выберите вкладу NVIC. В ней включите прерывания «EXTI line 0», «EXTI line 1», «EXTI line 2», «EXTI line 3» и назначьте им любые приоритеты ниже основных системных прерываний (любое число от 1 до 15).

6. Дополнительно включите UART. Для этого в списке «Connectivity» выберите «USART2» и переключите его «Mode» на «Asynchronous». Более подробное описание UART приведено в разделе 5, а на текущий момент достаточно настроек по умолчанию.

7. Завершите настройку проекта и сгенерируйте проект для CubeIDE. На этом дополнительная настройка проекта в CubeMX закончена.

Работа с проектом в CubeIDE

Для настройки обработки внешних прерываний выполните следующие действия.

1. Откройте созданный проект в CubeIDE и в папке «Src» найдите файл «stm32f3xx_it.c». Данный файл содержит все обработчики прерываний, которые непосредственно вызываются ядром процессора. Убедитесь, что файл содержит четыре обработчика внешних прерываний:

– EXTI0_IRQHandler;
– EXTI1_IRQHandler;
– EXTI2_IRQHandler;
– EXTI3_IRQHandler.

2. Откройте файл «main.c». Для упрощения работы с прерываниями дефолтные обработчики в «stm32f3xx_it.c» вызывают функцию `HAL_GPIO_EXTI_IRQHanlder`. Она очищает нужные биты в регистрах блока EXTI и передает управление функции `HAL_GPIO_EXTI_`

Callback, которую необходимо реализовать самостоятельно. Для этого добавьте следующий код в файл «main.c»:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
	//
	// Входной аргумент GPIO_Pin представляет из себя
битовую
	// маску с номером пина, откуда пришло прерывание.
	// Его значение равно одной из следующих констант:
	// - GPIO_PIN_0
	// - GPIO_PIN_1
	// - GPIO_PIN_2
	// - GPIO_PIN_3

	// Определим тип события (кнопка нажата или отпущена).
	// Для этого просто считаем ее состояние.
	int btn_state = HAL_GPIO_ReadPin(GPIOC, GPIO_Pin);
	if (btn_state) {
		// включить все светодиоды
		HAL_GPIO_WritePin(GPIOE, 0xFF00, GPIO_PIN_SET);
	} else {
		// выключить все светодиоды
		HAL_GPIO_WritePin(GPIOE,  0xFF00,  GPIO_PIN_
RESET);
	}

	// Перед выходом из прерывания на всякий случай
	// очистим его бит в EXTI. Это поможет избежать "за-
цикливания"
	// прерывания если при работе с пинами мы снова
сформировали
	// импульс, который инициировал данное прерывание
снова.
	__HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
}
```

3. Скомпилируйте полученную программу и загрузите ее на микроконтроллер. При нажатии на любую кнопку матричной клавиатуры обработчик прерываний будет включать светодиоды платы. При отпускании – отключать.

Вывод сообщений через UART

Для реализации вывода сообщений в UART добавьте следующий код в «main.c» и подключите библиотеку stdio.h (`#include «stdio.h»`):

```
/**
 * GNUC specific stdout/stderr hooks.
 *
 * @note: other compilers have other stdout/stderr hooks.
 */
#ifndef __GNUC__
#error Unknown compilator
#else
#include <unistd.h>
int _write(int fd, const void *buf, size_t count) {
      int res = 0;
      if (fd == STDOUT_FILENO || fd == STDERR_FILENO) {
            // write data to UART
            HAL_StatusTypeDef hal_res = HAL_UART_Transmit
(&huart2, (uint8_t*) buf, count, HAL_MAX_DELAY);
            res = hal_res == HAL_OK ? count : -1;
      } else {
            res = -1;
      }
      return res;
}
#endif
```

После этого вы сможете использовать стандартную функцию вывода `printf()` для вывода текстовых сообщений в UART. Например:

```
printf(«Hello world!\n»);
```

Прием сообщений по UART на стороне хоста

Для общения по UART на стороне хоста будет использоваться бесплатная программа Termite, которую можно скачать с официального сайта: https://www.compuphase.com/software_termite.htm. Для работы с этой программой:

1. Найдите и запустите программу Termite для работы с COM-портом.

2. Нажмите «settings».

3. Выберите нужный COM-порт. Если их несколько, придется попробовать каждый, пока не будет найдет нужный. Установите скорость передачи на 38400 (рис. 4.11).

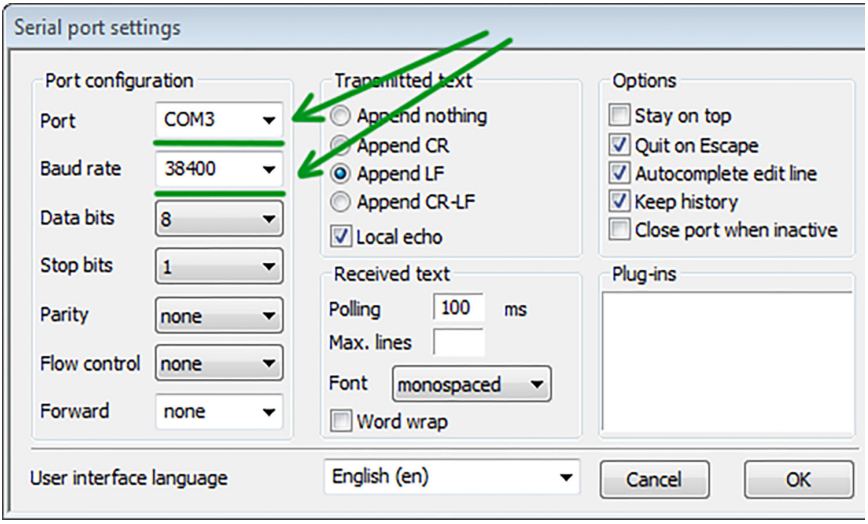


Рис. 4.11. Настройки порта

Рис. 4.12. Сообщения от микроконтроллера

4. Перейдите обратно в основное окно. Если COM-порт выставлен правильно и микроконтроллер передает сообщения, вы увидите их в основном окне (рис. 4.12).

4.3. Практические задания к разделу 4

Реализуйте программу, реагирующую на нажатие/отпускания кнопок K1-K16 клавиатуры НС-35, посредством внешних прерываний. Реакцию на прерывания выбирайте в соответствии с вариантом.

Варианты для индивидуальной работы

Вариант 1. Вывести номер нажатой кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов задается как «номер кнопки» % 8. Если «номер кнопки» <= 8, то анимация должна двигаться по часовой стрелке, иначе – против часовой.

Вариант 2. Вывести номер нажатой кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов – 2. Скорость анимации должна задаваться номером последней нажатой кнопки. При нажатии K1 – скорость должна быть минимальной, при нажатии K16 – максимальной.

Вариант 3. Вывести номер нажатой и отпущенной кнопки в UART. С помощью светодиодов отобразить двоичное представление числа S. Число S формируется следующим образом: начальное значение $S = 0$; при нажатии кнопки i, добавить номер этой кнопки к S; если $S > 255$, то из S вычесть 256.

Вариант 4. Вывести номер нажатой кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов – 3. Скорость анимации должна задаваться длительностью нажатия кнопки. Короткое нажатие должно вызвать медленное движение светодиодов по кругу. Длительное – быстрое.

Вариант 5. Вывести номер нажатой и отпущенной кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов – 2. В обычном состоянии светодиоды не двигаются. Но при нажатии кнопки они плавно продвигаются по часовой стрелке на число позиций равному номеру кнопки.

Вариант 6. Вывести номер отпущенной кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов задается как «номер кнопки» % 8. Если «номер кнопки» <= 8, то анимация должна двигаться против часовой стрелки, иначе – по часовой.

Вариант 7. Вывести номер отпущенной кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов – 1. Скорость анимации должна задаваться номером последней отпущенной кнопки. При нажатии K1 – скорость должна быть максимальной, при нажатии K16 – минимальной.

Вариант 8. Вывести номер нажатой и опушенной кнопки в UART. При нажатии отобразить номер кнопки и включить n смежных светодиодов, где n = «номер столбца кнопки» + «номер строки кнопки».

Вариант 9. Вывести номер отпущенной кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов – 2. Скорость анимации должна задаваться длительностью нажатия кнопки. Короткое нажатие должно вызвать быстрое движение светодиодов по кругу. Длительное – медленное.

Вариант 10. Вывести номер кнопки в UART. Реализовать анимацию бегающих по кругу светодиодов. Количество активных светодиодов – 3. В обычном состоянии светодиоды не двигаются. Но при нажатии кнопки они плавно продвигаются на i шагов. i = «номер столбца нажатой кнопки» – «номер строки нажатой кнопки». Если $i > 0$, то светодиоды двигаются по часовой стрелке, иначе – против часовой.

5. АСИНХРОННЫЙ ПОСЛЕДОВАТЕЛЬНЫЙ АДАПТЕР (UART) И ИСПОЛЬЗОВАНИЕ ПРЕРЫВАНИЯ ДЛЯ ВВОДА-ВЫВОДА

5.1. Основные сведения об UART

UART (*Universal Asynchronous Receiver-Transmitter*, универсальный асинхронный приемопередатчик) – узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами [27]. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству. Метод преобразования хорошо стандартизован и широко применяется в компьютерной технике. Например, его часто используют для управления модемами, модулями Wi-Fi, bluetooth, GPS. Отметим основные особенности протокола устройства:

1. Протокол является асинхронным. Это означает, что он не требует дополнительной линии с тактовым сигналом для синхронизации. Однако из-за этого скорость передачи должна быть заранее согласована между приемником и передатчиком.

2. Передача данных происходит последовательно. В каждый момент времени может передаваться только один бит информации.

3. В минимальной конфигурации с полнодуплексным режимом достаточно трех линий: Ground, RX, TX. Их соединение показано на рис. 5.1.

4. Данные передаются пакетом. Пакет состоит из ряда служебных полей и блока данных (рис. 5.2). Обычно одним пакетом передается 1 байт информации.

5. Данные в линиях кодируются высоким и низким уровнем напряжения. Во время простоя на линии подается высокий уровень напряжения (рис. 5.3).

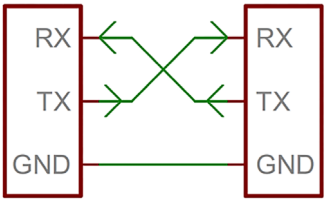


Рис. 5.1. Схема подключения UART

Рис. 5.2. Формат пакета UART

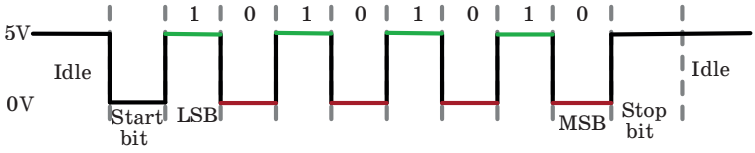


Рис. 5.3. Кодирование битов в UART

5.2. Использование UART в микроконтроллерах STM32

Настройка проекта в CubeMX

1. Создайте проект и выполните базовую конфигурацию проекта аналогично разделу 2.

2. Откройте «Categories» → «Connectivity» → «USART» на вкладке «Pinout & Configuration» и настройте UART:

– *Mode:* «Asynchronous»;

– *Overrun:* «Disable»;

– *Baud rate:* «38400 Bits/s».

Остальные параметры оставьте в значениях по умолчанию.

После включения UART (Mode: «Asynchronous»), CubeMX автоматически настроит пины PA_2 и PA_3 для UART (рис. 5.4).

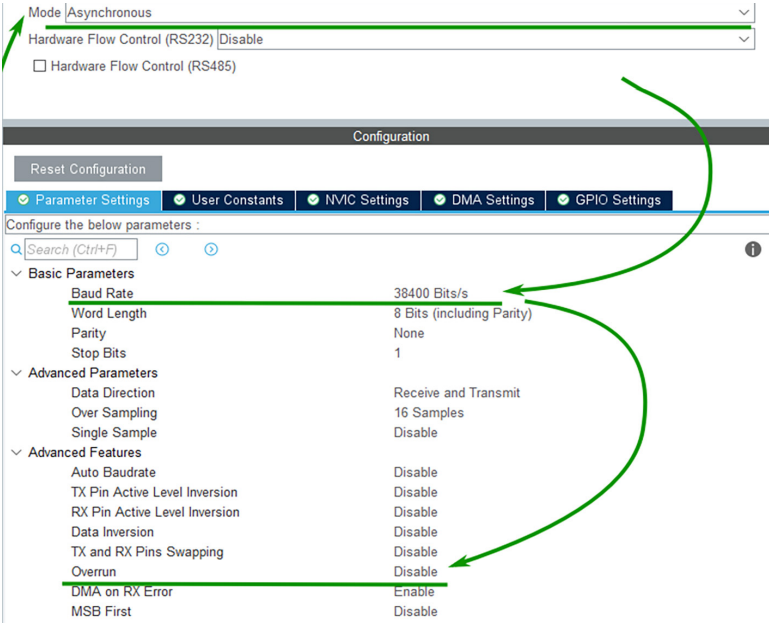


Рис. 5.4. Настройка UART

UART является асинхронным, поэтому запомните скорость «Baud rate». Позже ее нужно будет выставить в программе для работы с UART на стороне хоста.

3. Настройте прерывания. Внутренние регистры UART в STM32 позволяют хранить до 2 принятых байт. Если не считать их из UART, то новые байты, приходящие на UART, будут потеряны или перезапишут существующие данные (зависит от настроек UART). Из-за того, что данные могут приходить в любой момент времени, мы должны либо постоянно проверять UART на новые данные, либо настроить прерывание на прием данных. Для настройки прерывания перейдите на «System Core» → «NVIC» → «NVIC» и включите прерывание «UART2 global interrupt» с любым не нулевым приоритетом.

4. Увеличьте размер выделяемой памяти. По умолчанию CubeMX выделяет мало памяти под «кучу» и стек программы. Поэтому имеет смысл их увеличить, что позволит создавать массивы/буферы на этапе инициализации с помощью malloc. Перейдите на вкладку «Project Manager» и увеличьте значения «Minimum Heap Size» и «Minimum Stack Size» до 0×3000 и 0×1000 соответственно.

5. Завершите создание проекта.

Использование блокирующего HAL UART API. Вывод сообщений

В простейшем случае передачу данных микроконтроллера по UART можно осуществить с помощью функции `HAL_UART_Transmit`. Ее использование приведено в примере 5.1.

Пример 5.1. Базовое использование `HAL_UART_Transmit`. Фрагмент кода main.c:

```
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */
    /* MCU Configuration--------------------------- ---
--------*/
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
  HAL_Init();
    /* USER CODE BEGIN Init */
```

```
    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */
    const size_t num_messages = 4;
    const char *messages[4] = { "-- Start --\n", "<Message
1>\n", "<Message 2>\n", "-- End --\n" };
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        for (size_t i = 0; i < num_messages; i++) {
            // show message
                  HAL_UART_Transmit(&huart2,  (uint8_t*)
messages[i], strlen(messages[i]), HAL_MAX_DELAY);

            HAL_GPIO_TogglePin(GPIOE, 0xFF00);
            HAL_Delay(500);
        }

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

Пример 5.1 содержит только код функции `main`. Остальной код из main.c оставлен без изменений. Из дополнительных библиотек в данном и следующих примерах подключены:

```
/* Private includes ----------------------------------------
--------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include <ctype.h>
/* USER CODE END Includes */
```

Приведенная программа выводит сообщения в UART с задержкой 500 мс между соседними сообщениями. Для вывода сообщений используется функция:

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

со следующим параметрами:

1) `UART_HandleTypeDef *huart` – указатель на структуру данных с описанием UART. Она генерируется CubeMX и по умолчанию называется: `huart1`, `huart2` и т. п.;

2) `uint8_t *pData` – массив с данными, которые нужно передать по UART;

3) `uint16_t Size` – размер массива `pData`;

4) `uint32_t Timeout` – максимальное время ожидания операции. Если нужно, чтобы функция всегда ждала завершения операции, то в качестве `Timeout` передается константа `HAL_MAX_DELAY`.

Функция `HAL_UART_Transmit` возвращает одно из следующих значений:

– `HAL_OK` – указанное количество данных, переданное в пределах указанного таймаута;

– код ошибки в случае ее возникновения, например: `HAL_TIMEOUT` – указанное количество данных, которое не было передано в пределах указанного промежутка времени `Timeout`.

Прием данных на стороне хоста

Для приема сообщения используйте программу Termite, как было показано в разделе 4.

Прием сообщений

Для приема сообщений имеется функция `HAL_UART_Receive`, аналогичная `HAL_UART_Transmit`. Поскольку эта функция требует знать заранее, сколько данных необходимо принять, воспользуемся более низкоуровневыми функциями. Они продемонстрированы в примере 5.2.

Пример 5.2. Базовое чтение данных по UART.

```
int main(void) {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */
     /* MCU Configuration--------------------------------
-----------------------*/
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */
    const char *start_message = "-- Start --\n";
    char echo_message[32];
    int has_data_flag;
    char sym;

   HAL_UART_Transmit(&huart2, (uint8_t*) (start_message),
strlen(start_message), HAL_MAX_DELAY);
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        // check
        has_data_flag = __HAL_UART_GET_FLAG(&huart2, UART_
FLAG_RXNE);
        if (has_data_flag) {
            // read data
            sym = huart2.Instance->RDR;
            // prepare echo message
```

```
            if (isgraph(sym)) {
                // show printable character as is
              sprintf(echo_message, "\necho: %c\n", sym);
                  HAL_UART_Transmit(&huart2, (uint8_t*)
(echo_message), strlen(echo_message), HAL_MAX_DELAY);
            } else {
                  // show hexadecimal code of the non-
printable character
                sprintf(echo_message, "\necho: 0x%02X\n",
sym);
                  HAL_UART_Transmit(&huart2, (uint8_t*)
(echo_message), strlen(echo_message), HAL_MAX_DELAY);
            }
        }

        HAL_GPIO_TogglePin(GPIOE, 0xFF00);
        HAL_Delay(500);

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

Программа считывает символы по UART и отправляет их обратно. Для поверки наличия пришедшего символа используется макрос `__HAL_UART_GET_FLAG`:

```
has_data_flag = __HAL_UART_GET_FLAG(&huart2, UART_
FLAG_RXNE);
```

Считывание пришедшего байта происходит из регистра UART следующим образом:

```
sym = huart2.Instance->RDR;
```

Если считанный байт кодирует символ с графическим представлением, он выводится обратно с помощью функции `HAL_UART_Transmit` без изменений, иначе выводится его код в шестнадцатеричной форме.

Для проверки примера запустите Termite и выключите в настройках автоматическое дописывание в конце строки (рис. 5.5).

Введите несколько фраз в окно ввода (рис. 5.6).

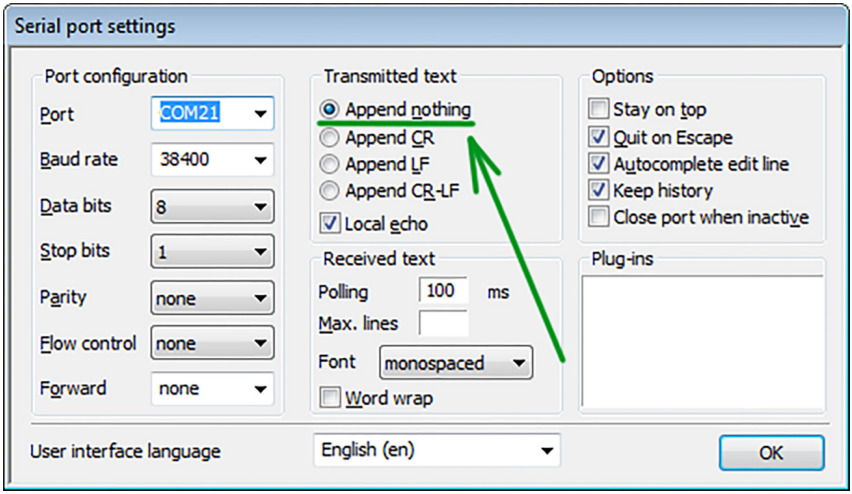


Рис. 5.5. Настройки Termite

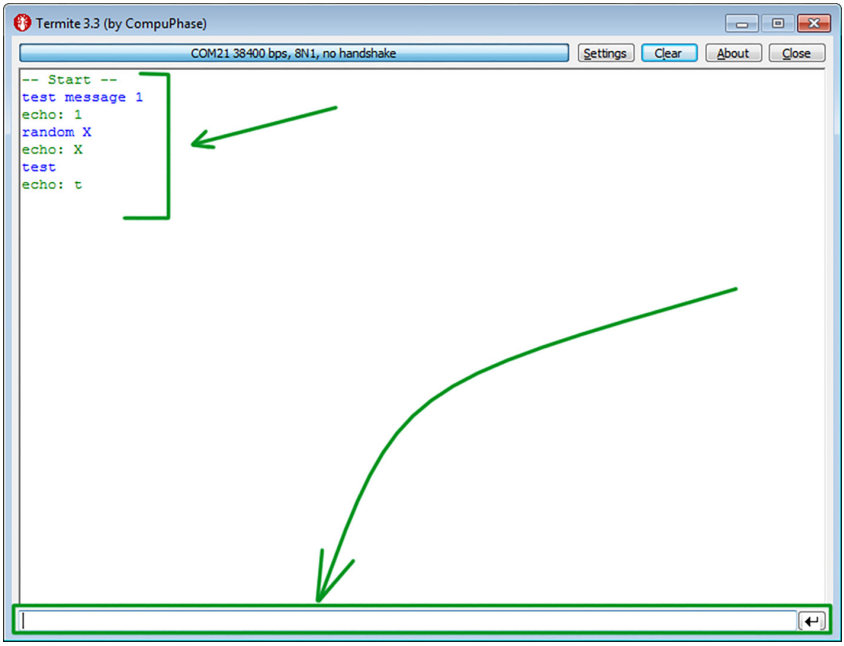


Рис. 5.6. Окно ввода Termite

Из-за задержки `HAL_Delay(500);` программа микроконтроллера проверяет приходящие данные на UART только каждые 0,5 секунд, и считывает только последний пришедший байт. Все остальные байты, пришедшие за этот промежуток, теряются.

Перенаправление стандартного вывода в UART

Непосредственное использование функции `HAL_UART_Transmit` неудобно для формирования и отправки текстовых и отладочных сообщений по UART. Поэтому имеет смысл перенаправить стандартные потоки вывода и ошибок (`stdout` и `stder`) в UART. Это позволит использовать стандартную функцию `printf` для вывода сообщений.

Способы переопределения функций работы с потоками не определены в стандартах C/C++ и зависят от конкретного компилятора. Для ARM-GCC (используется по умолчанию в CubeIDE) необходимо реализовать функцию `_write`, как показано в примере 5.3.

Пример 5.3. Перенаправление потока вывода в UART.

```
* Private user code ---------------------------------------
--------------------*/
/* USER CODE BEGIN 0 */

/**
 * GNUC specific stdout/stderr hooks.
 *
 * @note: other compilers have other stdout/stderr hooks.
 */
#ifndef __GNUC__
#error Unknown compilator
#else
#include <unistd.h>
int _write(int fd, const void *buf, size_t count) {
    int res = 0;
    if (fd == STDOUT_FILENO || fd == STDERR_FILENO) {
        // write data to UART
    HAL_StatusTypeDef hal_res = HAL_UART_Transmit(&huart2,
(uint8_t*) buf, count, HAL_MAX_DELAY);
        res = hal_res == HAL_OK ? count : -1;
    } else {
        res = -1;
    }
    return res;
}
#endif
```

```
/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------
-------------------*/
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */
    printf("--- start ---\n");
    uint8_t led_mask = 0x01;
    int counter = 0;

    /* USER CODE END 2 */

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

        // update leds
        HAL_GPIO_WritePin(GPIOE, 0xFF00, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOE, led_mask << 8, GPIO_PIN_
SET);

        // show demo message
       printf("Iteration %i. led_mask=0x%02X\n", counter,
led_mask);
```

```
        // update led mask
        counter++;
        led_mask = led_mask << 7 | led_mask >> 1;

        HAL_Delay(500);
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

При вызове функции `printf`, она выполняет форматирование сообщения (преобразование аргументов в строки), складывает его во внутренний буфер и периодически вызывает функцию `_write`, куда передает следующие аргументы:

- `int fd` – идентификатор потока вывода (1 для stdout и 2 для stderr);
- `const void *buf` – массив с байтами, которые нужно вывести;
- `size_t count` – длинна массива `buf`.

После передачи байтов, функция `_write` должна вернуть либо количество обработанных байт, либо отрицательное значение в случае ошибки.

Скомпилируйте и запустите указанную программу. Если все настроено правильно, в Termite вы увидите следующий вывод (рис. 5.7).

```
--- start ---
Iteration 0. led_mask=0x01
Iteration 1. led_mask=0x80
Iteration 2. led_mask=0x40
Iteration 3. led_mask=0x20
Iteration 4. led_mask=0x10
Iteration 5. led_mask=0x08
Iteration 6. led_mask=0x04
Iteration 7. led_mask=0x02
Iteration 8. led_mask=0x01
Iteration 9. led_mask=0x80
Iteration 10. led_mask=0x40
Iteration 11. led_mask=0x20
Iteration 12. led_mask=0x10
Iteration 13. led_mask=0x08
Iteration 14. led_mask=0x04
Iteration 15. led_mask=0x02
Iteration 16. led_mask=0x01
Iteration 17. led_mask=0x80
```

Рис. 5.7. Вывод Termite

Функция `printf`

Функция `printf` определена в библиотеке `stdio.h` и имеет следующую сигнатуру:

```
int printf(const char *format, ...);
```

`printf` выводит в стандартный поток вывода строку, отформатированную в соответствии с правилами, указанными в строке, на которую указывает аргумент `format`.

Строка форматирования состоит из элементов двух типов. К элементам первого типа относятся символы, которые выводятся на экран. Элементы второго типа содержат спецификации формата, определяющие способ отображения аргументов. Спецификация формата начинается символом процента, за которым следует спецификатор формата. Количество аргументов должно соответствовать количеству спецификаторов формата, причем соответствие устанавливается в порядке их следования. Например, при вызове следующей функции `printf`:

```
printf(«Hi %c %d %s», 'c', 10, «there!»);
```

на экране будет отображено:

```
«Hi c 10 there!»
```

Между знаком «%» и спецификатором формата могут присутствовать дополнительные управляющие символы, влияющие на ширину поля вывода, выравнивание и прочее, но они опциональны. Некоторые спецификаторы формата приведены в табл. 5.1.

Таблица 5.1

Спецификаторы формата printf

Код	Описание формата
%c	символ
%d	десятичное целое число со знаком
%i	десятичное целое число со знаком (тоже самое что и %d)
%f	десятичное число с плавающей точкой
%s	шестнадцатеричное без знака (строчные буквы)
%X	шестнадцатеричное без знака (прописные буквы)
%%	выводит знак процента

Использование прерываний для неблокирующего ввода/вывода UART. Передача данных

Для организации передачи данных можно использовать прерывание «transmit data register empty». Оно вызывается каждый раз, когда регистр UART готов принять данные для передачи. Для его использования необходимо добавить/обновить код в трех файлах (примеры 5.4, 5.5, 5.6).

Пример 5.4. Использование прерываний для вывода в UART. Фрагмент файла main.h:

```
/* USER CODE BEGIN EFP */

/**
 * Helper structure to hold data for UART.
 */
typedef struct {
    volatile char buf[128];
    volatile size_t start;
    volatile size_t end;
    UART_HandleTypeDef *huart;
} output_buf_t;

// declare global variable
extern output_buf_t project_output_buf;

/**
 * Initialize output_buf_t object.
 */
int   output_buf_init(output_buf_t   *output_buf,   UART_
HandleTypeDef *huart);

/**
 * Send data to UART;
 */
int  output_buf_send_str(output_buf_t  *output_buf,  const
char *str);

/**
 * Process UART interruption.
 */
int output_buf_process_txe_it(output_buf_t *output_buf);

/* USER CODE END EFP */
```

Пример 5.5. Использование прерываний для вывода в UART. Фрагмент файла main.c:

```
/* USER CODE BEGIN 0 */

// define global variable
output_buf_t project_output_buf;

/**
 * Initialize output_buf_t object.
 */
int   output_buf_init(output_buf_t   *output_buf,   UART_
HandleTypeDef *huart) {
    output_buf->start = 0;
    output_buf->end = 0;
    output_buf->huart = huart;
    return 0;
}

/**
 * Send data to UART;
 */
int  output_buf_send_str(output_buf_t  *output_buf,  const
char *str) {
    if (output_buf->start != output_buf->end) {
        // Error: previous transmission isn't completed
        return -1;
    }

    size_t str_len = strlen(str);
    if (str_len > sizeof(output_buf->buf)) {
        // Error: str is too long for internal buffer
        return -2;
    }

    // copy buffer
    memcpy((void*) output_buf->buf, str, str_len);
    output_buf->start = 0;
    output_buf->end = str_len;

    // enable interrupts
      __HAL_UART_ENABLE_IT(output_buf->huart,  UART_IT_
TXE);
```

```
    return 0;
}

/**
 * Process UART interruption.
 */
int  output_buf_process_txe_it(output_buf_t  *output_
buf) {
   // ignore interruption if it isn't related with data
transmission
    if (!__HAL_UART_GET_FLAG(output_buf->huart, UART_
FLAG_TXE)) {
        return 0;
    }

    if (output_buf->start == output_buf->end) {
            // all data has been transmitted. Stop
interruptions
        __HAL_UART_DISABLE_IT(output_buf->huart, UART_
IT_TXE);
        return 0;
    }

    // move data from buffer to register
      output_buf->huart->Instance->TDR = output_buf-
>buf[output_buf->start];
    output_buf->start++;

    return 0;
}

/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */
```

```
    /* USER CODE END 1 */
  /* MCU Configuration--------------------------------
---------------- -----*/
    /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */
    // initialize output buffer
    output_buf_init(&project_output_buf, &huart2);

    char buf[128];
    int current_time;
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        current_time = HAL_GetTick();
        sprintf(buf, "Time: %i\n", current_time);
        output_buf_send_str(&project_output_buf, buf);

        HAL_GPIO_TogglePin(GPIOE, 0xFF00);
        HAL_Delay(500);
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
      /* USER CODE END 3 */
  }
```

Пример 5.6. Использование прерываний для вывода в UART. Фрагмент файла stm32f3xx_it.c:

```
/**
 * @brief This function handles USART2 global interrupt /
USART2 wake-up interrupt through EXTI line 26.
 */
void USART2_IRQHandler(void) {
    /* USER CODE BEGIN USART2_IRQn 0 */
    output_buf_process_txe_it(&project_output_buf);

    /* USER CODE END USART2_IRQn 0 */
    HAL_UART_IRQHandler(&huart2);
    /* USER CODE BEGIN USART2_IRQn 1 */

    /* USER CODE END USART2_IRQn 1 */
}
```

Передача данных по прерываниям в приведенной выше программе сводится к следующими шагам:

1. В основном потоке:

1) подготовить буфер с данными для передачи (функция `output_buf_send_str`):

```
    // copy buffer
    memcpy((void*) output_buf->buf, str, str_len);
    output_buf->start = 0;
    output_buf->end = str_len;
```

2) запустить прерывание:

```
    // enable interrupts
   __HAL_UART_ENABLE_IT(output_buf->huart, UART_IT_TXE);
```

2. В обработчике прерывания (функции `USART2_IRQHandler`, `output_buf_process_txe_it`):

1) проверить, что произошло прерывание «transmit data register empty». Это необходимо сделать, так как обработчик прерываний `USART2_IRQHandler` обслуживает все события, связанные с USART2:

```
    // ignore interruption if it isn't related with data
transmission
      if (!__HAL_UART_GET_FLAG(output_buf->huart, UART_
FLAG_TXE)) {
        return 0;
    }
```

2) если данных для передачи нет, необходимо выключить прерывание, иначе перейти к следующему шагу:

```
    if (output_buf->start == output_buf->end) {
      // all data has been transmitted. Stop interruptions
        __HAL_UART_DISABLE_IT(output_buf->huart, UART_IT_
TXE);
        return 0;
    }
```

3) перенести очередной байт из буфера в регистр UART для передачи данных:

```
    // move data from buffer to register
        output_buf->huart->Instance->TDR  =  output_buf-
>buf[output_buf->start];
    output_buf->start++;
```

Скомпилируйте и загрузите приведенную программу на микроконтроллер. Если все сделано правильно, в Termite вы увидите следующий вывод (рис. 5.8).

Из особенностей программы отметим объявление глобальных переменных, видимых в разных файлах. Как и публичные функции, глобальные переменные нужно объявить в заголовочном файле со спецификатором `extern`:

Рис. 5.8. Вывод Termite

```
/**
 * Helper structure to hold data for UART.
 */
typedef struct {
    volatile char buf[128];
    volatile size_t start;
    volatile size_t end;
    UART_HandleTypeDef *huart;
} output_buf_t;

// declare global variable
extern output_buf_t project_output_buf;
и одном из файлов программ объявить переменную без extern:
// define global variable
output_buf_t project_output_buf;
```

Прием данных

Для организации передачи данных можно использовать прерывание «Receive data register not empty (data ready to be read)». Оно вызывается каждый раз, когда на UART поступают данные. Для его использования необходимо добавить/обновить код в трех файлах (примеры 5.7, 5.8, 5.9).

Пример 5.7. Использование прерываний для вывода в UART. Фрагмент файла main.h:

```
/* USER CODE BEGIN EFP */
/**
 * Helper structure read lines from UART.
 */
#define INPUT_BUF_SIZE 32
typedef struct {
    volatile char buf[INPUT_BUF_SIZE];
    volatile size_t pos;
    UART_HandleTypeDef *huart;
} input_buf_t;

// declare global variable
extern input_buf_t project_input_buf;

/**
 * Initialize input_line_buf_t object.
 */
```

```
int    input_buf_init(input_buf_t    *input_buf,    UART_
HandleTypeDef *huart);

/**
 * Read current content of the buffer.
 */
int  input_buf_read_content(input_buf_t  *input_buf,  char
*output);

/**
 * Process UART interruption.
 */
int input_buf_process_rxne_it(input_buf_t *input_buf);
/* USER CODE END EFP */
```

Пример 5.8. Использование прерываний для вывода в UART. Фрагмент файла main.c:

```
/* USER CODE BEGIN 0 */

// define global variable
input_buf_t project_input_buf;

/**
 * Initialize input_line_buf_t object.
 */
int    input_buf_init(input_buf_t    *input_buf,    UART_
HandleTypeDef *huart) {
    memset((void*) input_buf->buf, 0, INPUT_BUF_SIZE);
    input_buf->pos = 0;
    input_buf->huart = huart;

    // enable interrupt
    __HAL_UART_ENABLE_IT(&huart2, UART_IT_RXNE);

    return 0;
}

  /**
 * Read current content of the buffer.
 */
int  input_buf_read_content(input_buf_t  *input_buf,  char
*output) {
```

```
    int pos = input_buf->pos;
    // copy data to `output` buffer
    for (int i = 0; i < INPUT_BUF_SIZE; i++) {
        output[i] = input_buf->buf[pos];
        pos++;
        if (pos >= INPUT_BUF_SIZE) {
            pos = 0;
        }
    }

    return INPUT_BUF_SIZE;
}

/**
 * Process UART interruption.
 */
int input_buf_process_rxne_it(input_buf_t *input_buf) {
    // ignore interrupt if it isn't related with received
data
    if (!__HAL_UART_GET_FLAG(input_buf->huart, UART_FLAG_
RXNE)) {
        return 0;
    }

    // process received data
    char sym = input_buf->huart->Instance->RDR;

    // save symbol into buffer
    size_t pos = input_buf->pos;
    input_buf->buf[pos] = sym;
    pos++;
    if (pos >= INPUT_BUF_SIZE) {
        pos = 0;
    }
    input_buf->pos = pos;

    return 0;

}
/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
```

```
 */
int main(void) {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

     /* MCU Configuration--------------------------------
-----------------------*/
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */
    input_buf_init(&project_input_buf, &huart2);
    char buf[INPUT_BUF_SIZE + 1];
    char output_buf[INPUT_BUF_SIZE + 32];
    int size;
    int offset;
    char sym;
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        // read input buffer content
        size = input_buf_read_content(&project_input_buf,
buf);
        // prepare content for "echo"
        strcpy(output_buf, "input buf: \"");
        offset = strlen(output_buf);
        for (int i = 0; i < size; i++) {
```

```
            sym = buf[i];
           // replace non printable symbols except spaces
with dash
            if (!(sym == ' ' || isgraph(sym))) {
                sym = '-';
            }
            output_buf[offset + i] = sym;
        }
        offset += size;
        strcpy(output_buf + offset, "\"\n");
        // show echo
       HAL_UART_Transmit(&huart2, (uint8_t*) output_buf,
strlen(output_buf), HAL_MAX_DELAY);

        HAL_GPIO_TogglePin(GPIOE, 0xFF00);
        HAL_Delay(1000);

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

Пример 5.9. Использование прерываний для вывода в UART. Фрагмент файла `stm32f3xx_it.c`:

```
/**
 * @brief This function handles USART2 global interrupt /
USART2 wake-up interrupt through EXTI line 26.
 */
void USART2_IRQHandler(void) {
    /* USER CODE BEGIN USART2_IRQn 0 */
    input_buf_process_rxne_it(&project_input_buf);

    /* USER CODE END USART2_IRQn 0 */
    HAL_UART_IRQHandler(&huart2);
    /* USER CODE BEGIN USART2_IRQn 1 */

    /* USER CODE END USART2_IRQn 1 */
}
```

Прием данных по прерываниям в приведенной программе сводится к следующими шагам:

1. В основной программе на этапе инициализации:

1) подготовка буфера:

```
memset((void*) input_buf->buf, 0, INPUT_BUF_SIZE);
input_buf->pos = 0;
input_buf->huart = huart;
```

2) включение «receive data register not empty» прерывания:

```
// enable interrupt
__HAL_UART_ENABLE_IT(&huart2, UART_IT_RXNE);
```

2. В основной программе (вечный цикл):

чтение буфера принятых байт и его обработка:

```
        // read input buffer content
        size = input_buf_read_content(&project_input_buf,
buf);
        // prepare content for "echo"
        strcpy(output_buf, "input buf: \"");
        offset = strlen(output_buf);
        for (int i = 0; i < size; i++) {
            sym = buf[i];
           // replace non printable symbols except spaces
with dash
            if (!(sym == ' ' || isgraph(sym))) {
                sym = '-';
            }
            output_buf[offset + i] = sym;
        }
        offset += size;
        strcpy(output_buf + offset, "\"\n");
        // show echo
        HAL_UART_Transmit(&huart2, (uint8_t*) output_buf,
strlen(output_buf), HAL_MAX_DELAY);
```

3. В обработчике прерывания (функции `USART2_IRQHandler`, `input_buf_process_rxne_it`):

1) проверка того, что произошло прерывание «receive data register not empty». Это необходимо сделать, так как обработчик прерываний `USART2_IRQHandler` обслуживает все события, связанные с USART2:

```
    // ignore interrupt if it isn't related with received
data
    if (!__HAL_UART_GET_FLAG(input_buf->huart, UART_FLAG_
RXNE)) {
        return 0;
    }
```

2) считывание байта из регистра UART и сохранение его в буфере:

```
    // read received data
    char sym = input_buf->huart->Instance->RDR;

    // save symbol into buffer
    size_t pos = input_buf->pos;
    input_buf->buf[pos] = sym;
    pos++;
    if (pos >= INPUT_BUF_SIZE) {
        pos = 0;
    }
    input_buf->pos = pos;
```

Скомпилируйте, запустите приведенную программу и введите в Termine несколько сообщений. Если все сделано правильно, вы увидите вывод наподобие следующего (рис. 5.9).

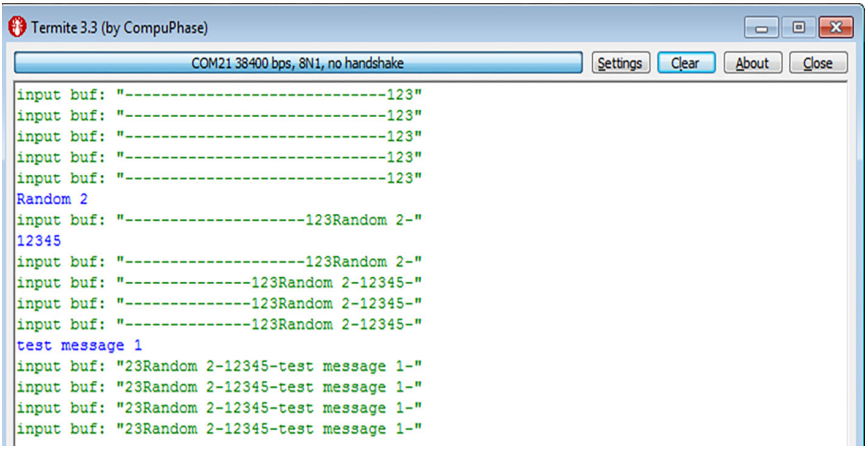


Рис. 5.9. Вывод Termite

5.3. Команды управления данными при сетевом обмене

Для управления устройством по UART необходимо согласовать и реализовать какой-либо протокол. Чтобы полностью не разрабатывать протокол «с нуля», можно воспользоваться существующими протоколами, определяющими формат сообщений. В качестве примера рассмотрим команды управления данными при сетевом обмене (AT-команды) [28]. Данный протокол широко используется при работе с устройствами связи (сотовые модемы, Wi-Fi и bluetooth модули).

В протоколе имеется клиент и исполнительное устройство. Клиент посылает устройству тестовые команды формата:

```
AT+<ИМЯ_КОМАНДЫ>[:<аргумент_1>[,<аргумент 2>]]
```

а устройство может вернуть какие-либо значения:

```
+<ИМЯ_КОМАНДЫ>:<значение_1>[,<заначение_2>[,<значен
ие_3>]]
OK
```

или просто сообщение, что команда выполнена успешно:

```
OK
```

Если имеется команда, задающая состоянии устройства:

```
AT+<ИМЯ_КОМАНДЫ>:<аргумент_1>
```

то обычно имеется следующая команда для чтения состояния:

```
AT+<ИМЯ_КОМАНДЫ>?
```

Приведем несколько примеров (табл. 5.2).

Таблица 5.2

Пример протокола AT

Запросы клиента	Ответы устройства
AT+MODE=1	
	OK
AT+MODE?	
	+MODE: 1
	OK
AT+MODE=234	
	ERRROR
AT+BTNSTATE	
	+BTNSTATE:0
	OK

Отметим, что команды всегда завершаются либо «OK», либо «ERROR» в зависимости от результата выполнения.

Обработка ввода с помощью sscanf

Для обработки поступающих команд их нужно каким-либо образом парсить. Проще всего это сделать с помощью стандартной функции `sscanf` (пример 5.10).

Пример 5.10. Использование `sscanf` для обработки AT команд в обычной C программе:

```
#include <stdio.h>
#include <string.h>

int process_command(const char *cmd) {
    int n = 0;
    int cmd_len = strlen(cmd);

    sscanf(cmd, "AT+MODE=%*d,%*d%n", &n);
    if (n == cmd_len) {
        int i1, i2;
        sscanf(cmd, "AT+MODE=%d,%d", &i1, &i2);
        printf("set mode: %i, %i\n", i1, i2);
        return 0;
    }

    sscanf(cmd, "AT+SAYHELLO%n", &n);
    if (n == cmd_len) {
        printf("Hello world!\n");
        return 0;
    }
    printf("Unknown command: %s\n", cmd);
    return -1;

}

int main ()
{
    process_command("AT+MODE=1,2");
    process_command("AT+SAYHELLO");
    process_command("AT+TEST?");
    return 0;
}
```

Обработка команды в программе выше состоит из двух шагов:

1) проверка, что команда советует заданному шаблону;

2) извлечение из команды аргументов (опционально).

В первом случае в строку-шаблон для `sscanf` на места, где ожидается число, записывается последовательность «`%*d`» и в конце строки всегда добавляется «`%n`». «`*`» означает, что указанный тип не нужно сохранять в выходной аргумент. Если исходная строка советует указанному шаблону, то в последний аргумент будет записана длина исходной строки.

Во втором случае в места, где ожидается число, записывается последовательность «%d». Остальными аргументами sscanf идут переменные, в которые необходимо записать считанные числа.

5.4. Практические задания к разделу 5

Реализовать программу для исполнения AT команд, поступающих по интерфейсу UART. Список команд и реакция микроконтроллера определяются вариантом и приведены ниже.

Получение и прием данных по UART необходимо буферизовать. Для этого следует организовать 2 циклических буфера (на прием и передачу) и обработку прерываний UART. Основная программа должна записывать и считывать данные из буферов, а по прерываниям UART данные должны непосредственно поступать из буферов в регистры UART и наоборот.

Варианты для индивидуальной работы

Вариант 1

Команда	Пример	Описание
AT+BTNSTATE	запрос: AT+BTNSTATE ответ: +BTNSTATE:0 OK	Получить текущее состояние кнопки микроконтроллера. 0 – кнопка не нажата, 1 – кнопка нажата.
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE:0 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение двух соседних светодиодов по часовой стрелке; 1 – вращение двух светодиодов против часовой стрелки
	запрос (установка режима): AT+MODE=1 ответ: OK	
AT+BLINK	запрос на 3 мигания: AT+BLINK=3 ответ: OK	Команда мигания всеми светодиодами. После поступления команды, микроконтроллер прекращает текущую анимацию (см. AT+MODE), делает указанное количество миганий, выводит «OK» и возвращается в режим, указанной командой AT+MODE.
	запрос на 10 миганий: AT+BLINK=10 ответ: OK	

Вариант 2

Команда	Пример	Описание
AT+TIME	запрос: AT+TIME ответ: +TIME:1231 OK	Получить количество миллисекунд, со старта микроконтроллера (см. HAL_GetTick)
	запрос: AT+TIME ответ: +TIME:812221 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE: 1 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение одного светодиода по часовой стрелке; 1 – вращение одного светодиода против часовой стрелки; 2 – управление отдельными светодиодами (см. команду AT+LED)
	запрос (установка режима): AT+MODE=2 ответ: OK	
AT+LED	запрос (получение состояния 3-го светодиода): AT+LED=3 ответ: +LED: 0 OK	Команда включение/выключение и считывания состояния светодиодов. Непосредственно управляется светодиодами только в 3-м режиме работы. В остальных она просто запоминает заданное состояние светодиодов. Первым аргументом команды является номер светодиода. Вторым – требуемое состояние (0 – выключен, 1 – включен). Если второй аргумент отсутствует, необходимо вывести текущее состояние светодиода
	запрос (включение 5-го светодиода): AT+LED=5,1 ответ: OK	

Вариант 3

Команда	Пример	Описание
AT+BTNCLK	запрос: AT+BTNCLK ответ: +BTNCLK: 0 OK	Получить количество кликов по кнопке (PA_0) с начала работы микроконтроллера
	запрос: AT+BTNCLK ответ: +BTNCLK: 42 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE:0 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение двух соседних светодиодов по часовой стрелке; 1 – вращение двух соседних светодиодов против часовой стрелки
	запрос (установка режима): AT+MODE=1 ответ: OK	
AT+BLINK	запрос на 3 мигания: AT+BLINK=3 ответ: OK	Команда мигания всеми светодиодами. После поступления команды, микроконтроллер прекращает текущую анимацию (см. AT+MODE), делает указанное количество миганий, выводит «OK» и возвращается в режим, указанной командой AT+MODE.
	запрос на 10 миганий: AT+BLINK=10 ответ: OK	

Вариант 4

Команда	Пример	Описание
AT+BTNSTATE	запрос: AT+BTNSTATE ответ: +BTNSTATE:0 OK	Получить текущее состояние кнопки микроконтроллера. 0 – кнопка не нажата, 1 – кнопка нажата
	запрос: AT+BTNSTATE ответ: +BTNSTATE:1 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE:0 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение трех соседних светодиодов по часовой стрелке; 1 – вращение трех соседних светодиодов против часовой стрелки
	запрос (установка режима): AT+MODE=1 ответ: OK	
AT+BLINK	запрос на 3 мигания 1-го светодиода: AT+BLINK=1,3 ответ: OK	Команда мигания заданным светодиодом. После поступления команды, микроконтроллер прекращает текущую анимацию (см. AT+MODE), делает указанное количество миганий заданным светодиодом, выводит «OK» и возвращается в режим, указанный командой AT+MODE
	запрос на 10 миганий 2-го светодиода: AT+BLINK=2,10 ответ: OK	

Вариант 5

Команда	Пример	Описание
AT+BTNCLK	запрос: AT+BTNCLK ответ: +BTNCLK: 0 OK	Получить количество кликов по кнопке (PA_0) с начала старта микроконтроллера
	запрос: AT+BTNCLK ответ: +BTNCLK: 42 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE: 1 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение двух соседних светодиодов по часовой стрелке; 1 – вращение двух соседних светодиодов против часовой стрелки; 2 – управление отдельными светодиодами (см. команду AT+LED)
	запрос (установка режима): AT+MODE=2 ответ: OK	
AT+LED	запрос (получение состояния 3-го светодиода): AT+LED=3 ответ: +LED: 0 OK	Команда включение/выключение и считывания состояния светодиодов. Непосредственно управляется светодиодами только в 3-м режиме работы. В остальных она просто запоминает заданное состояние светодиодов. Первым аргументом команды является номер светодиода. Вторым – требуемое состояние (0 – выключен, 1 – включен). Если второй аргумент отсутствует, необходимо вывести текущее состояние светодиода
	запрос (включение 5-го светодиода): AT+LED=5,1 ответ: OK	

Вариант 6

Команда	Пример	Описание
AT+BTNSTATE	запрос: AT+BTNSTATE ответ: +BTNSTATE:0 OK	Получить текущее состояние кнопки микроконтроллера. 0 – кнопка не нажата, 1 – кнопка нажата
	запрос: AT+BTNSTATE ответ: +BTNSTATE:1 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE: 1 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение одного светодиода по часовой стрелке; 1 – вращение одного светодиода против часовой стрелки; 2 – управление отдельными светодиодами (см. команду AT+LED)
	запрос (установка режима): AT+MODE=2 ответ: OK	
AT+LED	запрос (получение состояния 3-го светодиода): AT+LED=3 ответ: +LED: 0 OK	Команда включение/выключение и считывания состояния светодиодов. Непосредственно управляется светодиодами только в 3-м режиме работы. В остальных она просто запоминает заданное состояние светодиодов. Первым аргументом команды является номер светодиода. Вторым – требуемое состояние (0 – выключен, 1 – включен). Если второй аргумент отсутствует, необходимо вывести текущее состояние светодиода
	запрос (включение 5-го светодиода): AT+LED=5,1 ответ: OK	

Вариант 7

Команда	Пример	Описание
AT+RELTIME	запрос: AT+RELTIME ответ: +RELTIME:14749 OK	Получить время с последнего вызова команды AT+RELTIME в миллисекундах. Самый первый вызов AT+RELTIME должен возвращать время со старта микроконтроллера
	запрос: AT+RELTIME ответ: +RELTIME:5210 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE:0 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение двух соседних светодиодов по часовой стрелке; 1 – вращение двух светодиодов против часовой стрелки
	запрос (установка режима): AT+MODE=1 ответ: OK	
AT+BLINK	запрос на 3 мигания: AT+BLINK=3 ответ: OK	Команда мигания всеми светодиодами. После поступления команды, микроконтроллер прекращает текущую анимацию (см. AT+MODE), делает указанное количество миганий, выводит «OK» и возвращается в режим, указанной командой AT+MODE
	запрос на 10 миганий: AT+BLINK=10 ответ: OK	

Вариант 8

Команда	Пример	Описание
AT+BTNSTAT	запрос: AT+BTNSTAT ответ: +BTNSTAT:5201,0 OK	Вывести общее время кнопки в отпущенном и нажатом состояниях со старта микроконтроллера. Первое число в ответе — это общее время кнопки в отпущенном состоянии в миллисекундах, второе – в нажатом состоянии
	запрос: AT+BTNSTAT ответ: +BTNSTAT:8201,1233 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE: 1 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение трех светодиодов по часовой стрелке; 1 – вращение трех светодиодов против часовой стрелки; 2 – управление отдельными светодиодами (см. команду AT+LED)
	запрос (установка режима): AT+MODE=2 ответ: OK	
AT+LED	запрос (получение состояния 3-го светодиода): AT+LED=3 ответ: +LED: 0 OK	Команда включение/выключение и считывания состояния светодиодов. Непосредственно управляется светодиодами только в 3-м режиме работы. В остальных она просто запоминает заданное состояние светодиодов. Первым аргументом команды является номер светодиода. Вторым – требуемое состояние (0 – выключен, 1 – включен). Если второй аргумент отсутствует, необходимо вывести текущее состояние светодиода
	запрос (включение 5-го светодиода): AT+LED=5,1 ответ: OK	

Вариант 9

Команда	Пример	Описание
AT+TIME	запрос: AT+TIME ответ: +TIME:1231 OK	Получить количество миллисекунд, со старта микроконтроллера (см. HAL_GetTick)
	запрос: AT+TIME ответ: +TIME:81221 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE:0 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение двух противоположных светодиодов по часовой стрелке; 1 – вращение двух противоположных светодиодов против часовой стрелки
	запрос (установка режима): AT+MODE=1 ответ: OK	
AT+BLINK	запрос на 4 мигания 7-го светодиода: AT+BLINK=7,4 ответ: OK	Команда мигания заданным светодиодом. После поступления команды, микроконтроллер прекращает текущую анимацию (см. AT+MODE), делает указанное количество миганий заданным светодиодом, выводит «OK» и возвращается в режим, указанной командой AT+MODE
	запрос на 6 миганий 2-го светодиода: AT+BLINK=2,6 ответ: OK	

Вариант 10

Команда	Пример	Описание
AT+BTNCOUNT	запрос: AT+BTNCOUNT ответ: +BTNCOUNT:0 OK	Получить количество кликов по кнопке (PA_0) с начала работы микроконтроллера
	запрос: AT+BTNCOUNT ответ: +BTNCOUNT:5 OK	
AT+MODE	запрос (получение текущего режима работы): AT+MODE? ответ: +MODE: 1 OK	Выбор режима работы светодиодов. Должны быть доступны следующие режимы: 0 – вращение одного светодиода по часовой стрелке; 1 – вращение одного светодиода против часовой стрелки; 2 – управление отдельными светодиодами (см. команду AT+LED)
	запрос (установка режима): AT+MODE=2 ответ: OK	
AT+LED	запрос (получение состояния 3-го светодиода): AT+LED=3 ответ: +LED: 0 OK	Команда включение/выключение и считывания состояния светодиодов. Непосредственно управляется светодиодами только в 3-м режиме работы. В остальных она просто запоминает заданное состояние светодиодов. Первым аргументом команды является номер светодиода. Вторым – требуемое состояние (0 – выключен, 1 – включен). Если второй аргумент отсутствует, необходимо вывести текущее состояние светодиода
	запрос (включение 5-го светодиода): AT+LED=5,1 ответ: OK	

6. ШИРОТНО-ИМПУЛЬСНАЯ МОДУЛЯЦИЯ

6.1. Основные сведения о таймерах

Таймеры являются важным блоком микроконтроллеров и центральных процессоров. Они применяются для [29]:

1) измерения временных интервалов;

2) формирования прерываний с заданной периодичностью;

3) формирования широтно-импульсной модуляции (ШИМ) и других сигналов для управления силовой и другой электроникой;

4) измерения длительности импульсов.

Микроконтроллеры STM32F3 имеют следующие таймеры [30]:

1) system timer (SysTick);

2) basic timers (TIM6, TIM7);

3) general-purpose timers (TIM2, TIM3, TIM4, TIM15, TIM16, TIM17);

4) advanced-control timers (TIM1, TIM8).

Системный таймер встроен в ядро всех ARM микроконтроллеров [31]. Он способен генерировать прерывания с заданным периодом и главным образом используется для нужд операционной системы или библиотеки HAL.

Остальные таймеры (TIMx) реализованы STMicroelectronics и разделены на 3 типа: basic, general-purpose, advanced-control. Более продвинутый тип таймеров имеет тот же функционал, что и предыдущий, но обладает дополнительными возможностями. Общая функциональная схема таймера представлена на рис. 6.1.

Далее из всей функциональности расширенного таймера STM32F3 будет использована только часть, представленная на рис. 6.2.

Главным элементом в схеме таймера является счетчик CNT Counter. По тактовым сигналам CK_CNT он постепенно увеличивается до значения Autoreload register включительно, потом сбрасывается в 0, и снова продолжает увеличиваться (рис. 6.3).

В простейшем случае счетчик напрямую подключен к системному генератору и инкрементируется со скоростью тактовой частоты процессора, составляющей десятки или сотни МГц. Однако большинство таймеров STM32 являются 16-битными и быстро переполняются на такой скорости. Поэтому на пути сигнала от SYSCLK к CNT counter, стоят два устройства, уменьшающие частоту тактирующего сигнала [21]:

1) Internal Clock Division (CKD) – может уменьшать частоту сигнала в 1, 2 или 4 раза;

2) PSK prescaler – может уменьшать частоту сигала от 1 до 2^{16} раз.

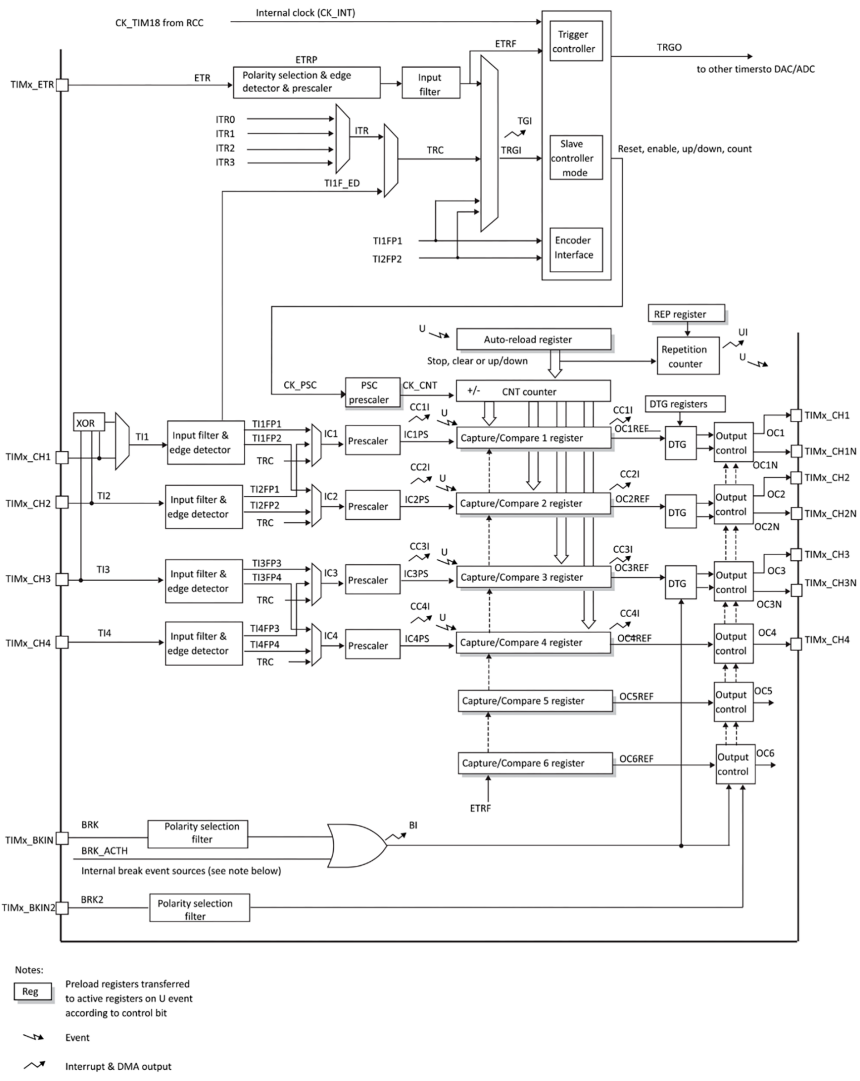


Рис. 6.1. Функциональная схема расширенного таймера STM32F3

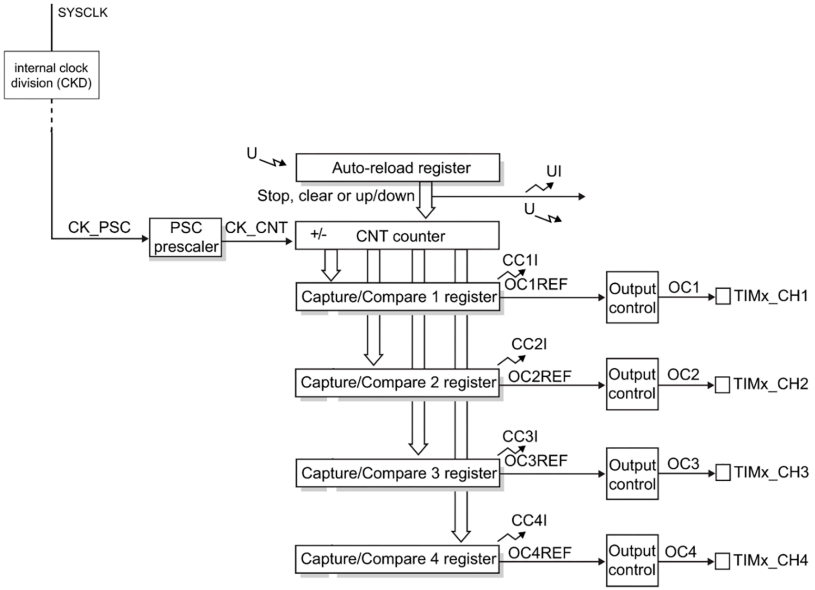


Рис. 6.2. Часть функциональной схемы таймера, отвечающая за ШИМ сигнал

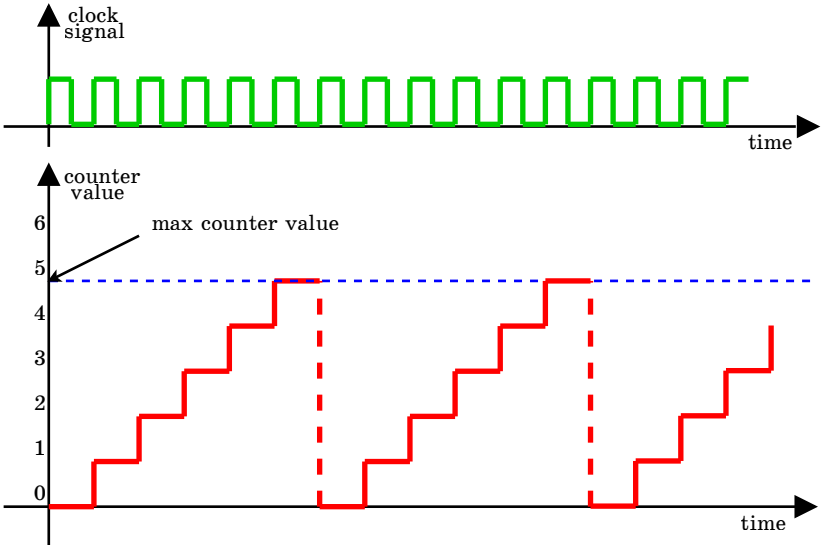


Рис. 6.3. Схема работы счетчика таймера

В итоге частота таймера (частота сброса счетчика CNT counter) рассчитывается следующим образом:

$$f = \frac{f_{SYSCLK}}{CKD(PSK+1)(ARR+1)},$$

где f_{SYSCLK} – частота ядра микроконтроллера; CKD – значение Internal Clock Division (1, 2 или 4); PSK – значение регистра предделителя (от 0 до 2^{16}-1); ARR – значение ARR регистра.

Обычно f_{SYSCLK}, CKD и PSK задаются на этапе инициализации микроконтроллера, а в процессе работы программы изменяется только значение ARR регистра.

«+1» в формулах возникает из-за того, что счетчики устройств считают такты начиная от 0 и до указанного значения включительно.

6.2. Формирование сигнала широтно-импульсной модуляции

Широтно-импульсная модуляция сигналов (ШИМ, англ. *pulse-width modulation* (PWM)) – процесс представления сигнала в виде череды импульсов с постоянной частотой и управления уровнем этого сигнала путем изменения скважности данных импульсов [32].

Широтно-импульсная модуляция может применяться для [33]:

– регулирования мощности, подаваемой на нагрузку. Например, в схемах управления электродвигателями постоянного тока, в импульсных преобразователях, для регулирования яркости светодиодных светильников, экранов ЖК-мониторов, дисплеев в смартфонах и планшетах и т. п.;

– формирования управляющих сигналов. Например, управление сервоприводами.

В микроконтроллерах STM32 для формирования ШИМ сигнала, таймеры снабжаются рядом каналов (рис. 6.1). Центральным элементом канала является Capture/Compare регистр. В режиме ШИМ его значение постоянно сравнивается со значением счетчика. Если его значение больше счетчика (CNT Counter), то на выход подключенного пина подается +3.3 V, иначе – 0 V (рис. 6.4).

Перечислим основные характеристики PWM сигнала:

– период T;

– ширина импульса τ;

– частота $f = \frac{1}{T}$;

– коэффициент заполнения $D = \frac{\tau}{T}$;

– скважность $S = \frac{T}{\tau}$.

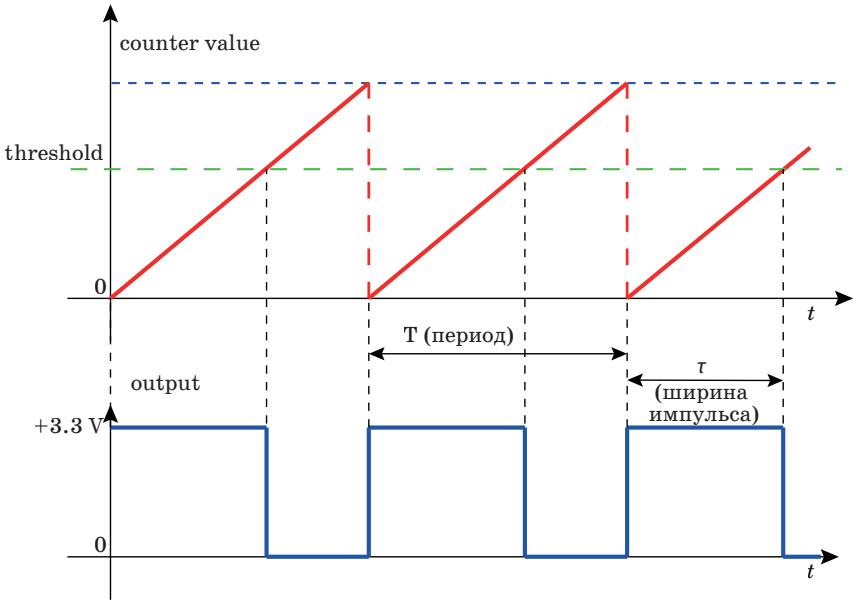


Рис. 6.4. Формирование ШИМ сигнала

Меандр

Особым случаем PWM сигнала является меандр [34], у которого ширина импульса равна половине периода (рис. 6.5). В простейшем случае ее можно использовать как замену синусоиды для генерации простых мелодий, что будет показано далее в примерах.

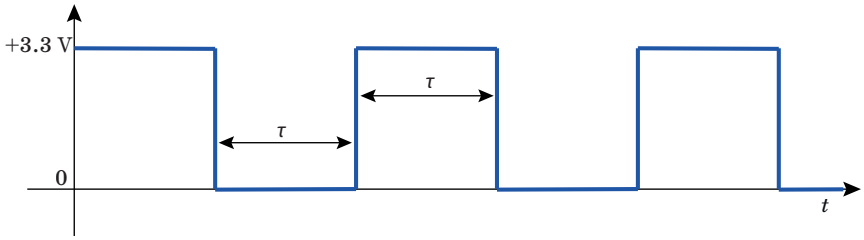


Рис. 6.5. Меандр

6.3. Использование таймеров в STM32

Настройка проекта в CubeMX:

1. Создайте новый проект или выполните базовые настройки.

2. Переключите пин PE9 (первый красный светодиод) в режим TIM1_CH1 (рис. 6.6).

3. Переключите пин PE13 (второй красный светодиод) в режим TIM1_CH3.

4. Переключите пин PA4 (динамик на макетной плате) в режим TIM3_CH2.

5. В итоге должно быть выбрано 3 пина для выходов каналов первого и третьего таймеров (рис. 6.7).

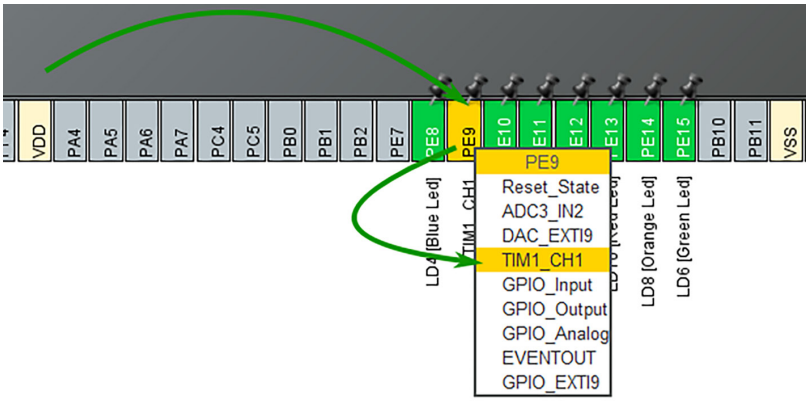


Рис. 6.6. Активация ШИМ пина первого таймера

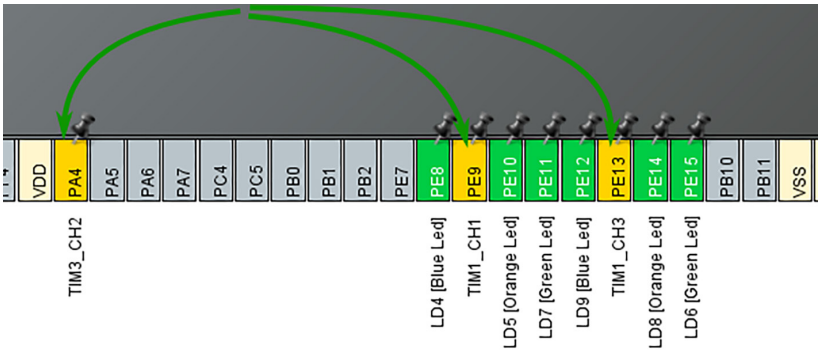


Рис. 6.7. Активация ШИМ пинов первого и второго таймеров

6. Откройте «Categories» → «Timers» → «TIM1». В настройках первого таймера выберите режим для первого канала – «PWM Generation CH1», и задайте следующие опции:

– Counter Settings:

Prescaler: 5;

Counter Period: 999;

Internal Clock Division: No Division;

auto-reload preload: Enable.

– PWM Generation Channel 1:

Pulse: 200.

7. Для третьего канала укажите «PWM Generation CH3» и следующие опции:

– PWM Generation Channel 3:

Pulse: 800.

8. Аналогично откройте опции третьего таймера («Categories»→ «Timers» → «TIM3»), выберите для второго канала режим «PWM Generation CH2» и задайте следующие опции:

– Counter Settings:

Prescaler: 47;

Counter Period: 999;

Internal Clock Division: No Division;

auto-reload preload: Enable.

– PWM Generation Channel 2:

Pulse: 500.

9. На данном этапе настройка таймеров завершена. Закончите настройку и создание проекта в CubeMX, как было описано в разделе 2, и запустите проект в CubeIDE.

Функции библиотеки HAL для работы с таймерами

Для демонстрации работы широтно-импульсной модуляции замените функцию `main` в main.c фрагментом из примера 6.1.

Пример 6.1. Фрагмент кода формирования ШИМ сигнала.

```
**
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */
    /* MCU Configuration--------------------------------
-----------------------*/
```

```
     /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */

    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    MX_TIM1_Init();
    MX_TIM3_Init();
    /* USER CODE BEGIN 2 */
    // local variable with pointer to timer struct
    TIM_HandleTypeDef *tim_led = &htim1;
    // timer channel
    uint32_t tim_led_channel = TIM_CHANNEL_1;
    // timer tick frequency CNT
    uint32_t tim_led_cnt_tick_freq;
    // helper variables for timer parameters calculations
    uint32_t arr_value;
    uint32_t cc_value;

    // reset CCR register
    __HAL_TIM_SET_COMPARE(tim_led, tim_led_channel, 0);
    // run timer
    HAL_TIM_PWM_Start(tim_led, tim_led_channel);
   // calculate timer tick frequency: f_tick = f_sysclcock
/ (CDK * (PSK + 1))
    tim_led_cnt_tick_freq = SystemCoreClock;
    switch (__HAL_TIM_GET_CLOCKDIVISION(tim_led)) {
    case TIM_CLOCKDIVISION_DIV1:
        tim_led_cnt_tick_freq /= 1;
        break;
    case TIM_CLOCKDIVISION_DIV2:
        tim_led_cnt_tick_freq /= 2;
        break;
    case TIM_CLOCKDIVISION_DIV4:
        tim_led_cnt_tick_freq /= 4;
        break;
    }
```

```
   tim_led_cnt_tick_freq /= (tim_led->Instance->PSC + 1);
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        // PWN signal with 0.0 duty cycle (i.e. output is
always 0)
        arr_value = 999;
        cc_value = 0;
        __HAL_TIM_SET_AUTORELOAD(tim_led, arr_value);
          __HAL_TIM_SET_COMPARE(tim_led, tim_led_channel,
cc_value);
        HAL_Delay(2000);

        // PWM signal
        // frequency:  1000 Hz
        // duty cycle: 0.5
        arr_value = tim_led_cnt_tick_freq / 1000 - 1;
        cc_value = (arr_value + 1) * 0.5f;
        __HAL_TIM_SET_AUTORELOAD(tim_led, arr_value);
          __HAL_TIM_SET_COMPARE(tim_led, tim_led_channel,
cc_value);
        HAL_Delay(2000);

        // PWM signal
        // frequency:  2000 Hz
        // duty cycle: 0.2
        arr_value = tim_led_cnt_tick_freq / 2000 - 1;
        cc_value = (arr_value + 1) * 0.2f;
        __HAL_TIM_SET_AUTORELOAD(tim_led, arr_value);
          __HAL_TIM_SET_COMPARE(tim_led, tim_led_channel,
cc_value);
        HAL_Delay(2000);

        // PWM signal
        // frequency:  2000 Hz
        // duty cycle: 0.8
        arr_value = tim_led_cnt_tick_freq / 2000 - 1;
        cc_value = (arr_value + 1) * 0.8f;
        __HAL_TIM_SET_AUTORELOAD(tim_led, arr_value);
          __HAL_TIM_SET_COMPARE(tim_led, tim_led_channel,
cc_value);
        HAL_Delay(2000);
```

```
        // PWN signal with 1.0 duty cycle (i.e. output is
always 1)
        arr_value = 999;
        cc_value = 1000;
        __HAL_TIM_SET_AUTORELOAD(tim_led, arr_value);
          __HAL_TIM_SET_COMPARE(tim_led, tim_led_channel,
cc_value);
        HAL_Delay(2000);

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
```

После компиляции и запуска программ, красный светодиод на STM32Discovery будет периодически менять свою яркость.

Получите осциллограф и подключите его к пину PE9. На нем должны отобразиться следующие, последовательно сменяющие друг друга, ШИМ сигналы:

1) $D = 0$ (на выходе постоянный 0) (рис. 6.8);
2) $f = 1000$ Гц, $D = 0{,}5$ (меандр) (рис. 6.9);
3) $f = 2000$ Гц, $D = 0{,}2$ (рис. 6.10);

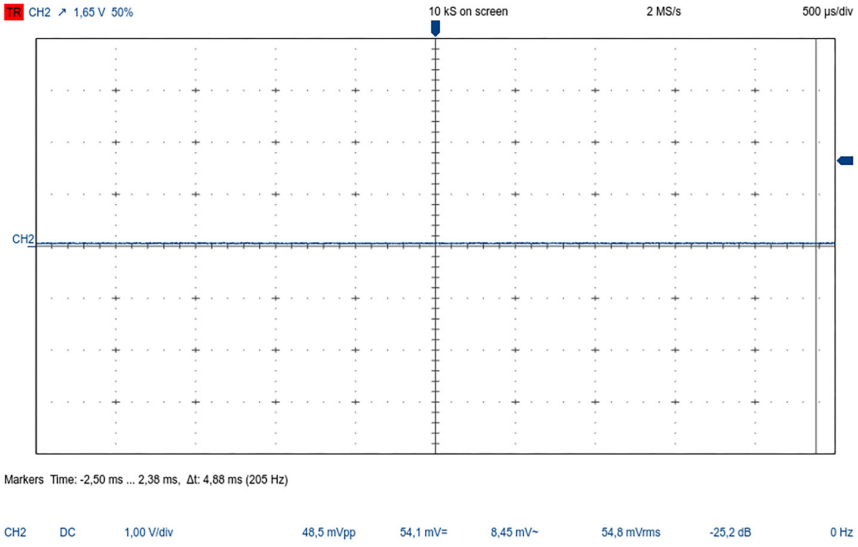


Рис. 6.8. ШИМ сигнал на осциллографе

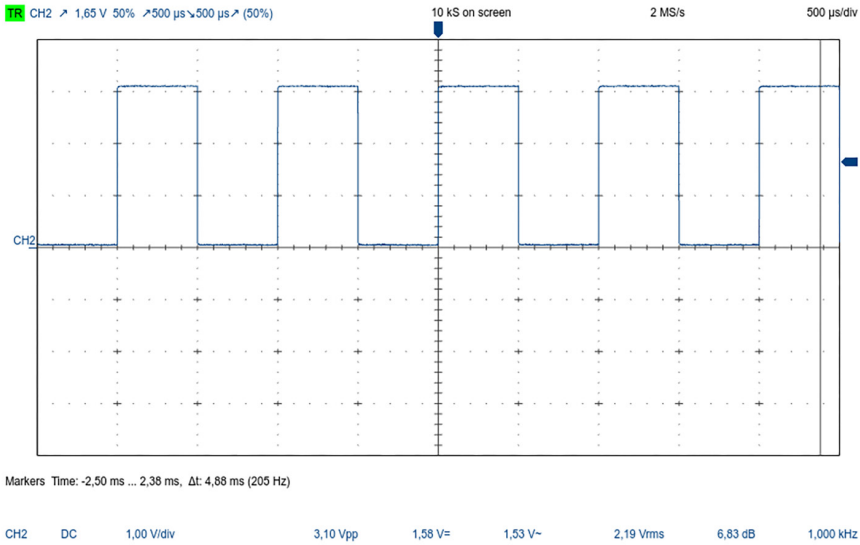


Рис. 6.9. ШИМ сигнал (меандр) на осциллографе

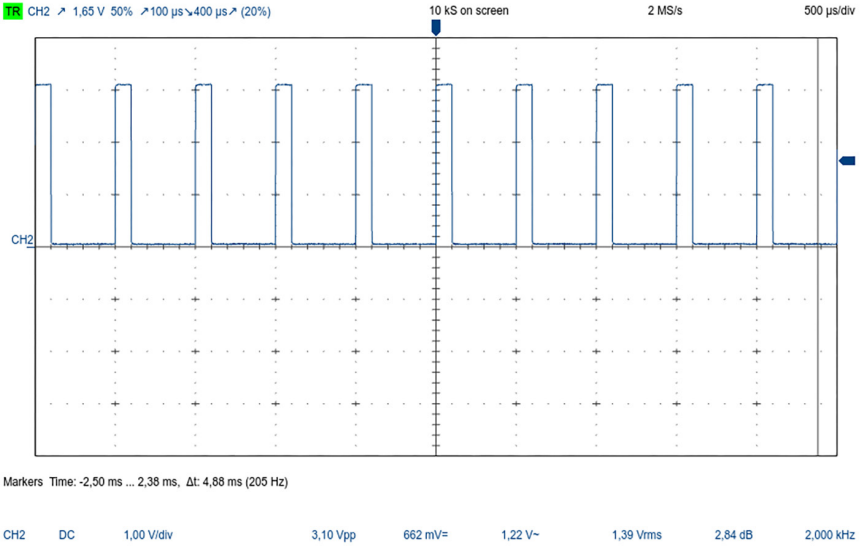


Рис. 6.10. ШИМ сигнал на осциллографе

4) $f = 2000$ Гц, $D = 0{,}8$ (рис. 6.11);
5) $D = 1$ (на выходе постоянные +3.3 В) (рис. 6.12).

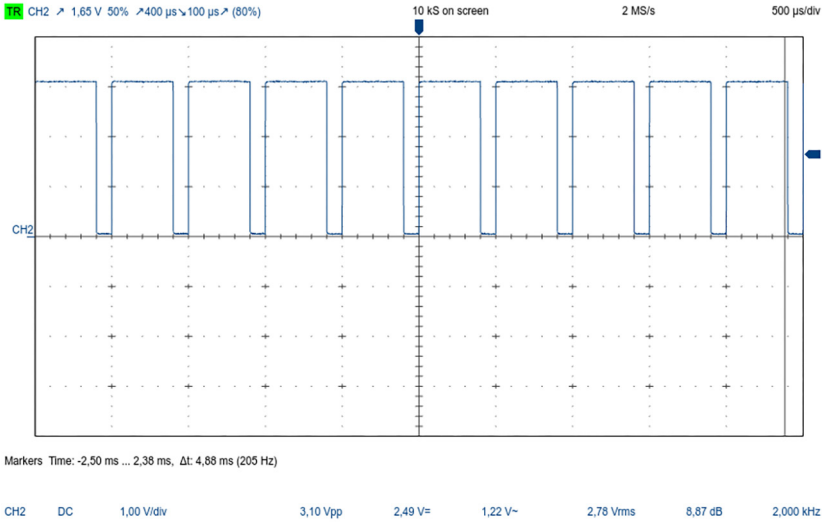


Рис. 6.11. ШИМ сигнал на осциллографе

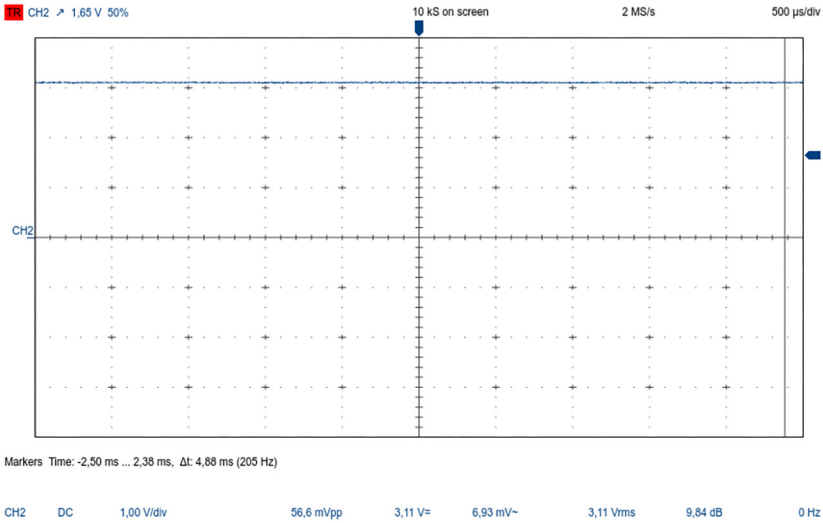


Рис. 6.12. ШИМ сигнал на осциллографе

Рассмотрим основные части примера 6.1.

1. Структуры данных и константы с описанием таймера:

```
// local variable with pointer to timer struct
TIM_HandleTypeDef *tim_led = &htim1;
// timer channel
uint32_t tim_led_channel = TIM_CHANNEL_1;
// timer tick frequency CNT
uint32_t tim_led_cnt_tick_freq;
```

Для каждого таймера CubeMX создает вспомогательную структуру htim<n>. Она содержит указатель на блок регистров таймера, и используется в качестве входного аргумента для большинства функции HAL по работе с таймерами.

Для работы с каналами отдельных структур данных нет. Вместо них используются константы TIM_CHANNEL_<n>, которые при необходимости передаются в функции HAL.

2. Расчет частоты инкремента счетчика таймера:

```
 // calculate timer tick frequency: f_tick = f_sysclcock
/ (CDK * (PSK + 1))
  tim_led_cnt_tick_freq = SystemCoreClock;
  switch (__HAL_TIM_GET_CLOCKDIVISION(tim_led)) {
  case TIM_CLOCKDIVISION_DIV1:
      tim_led_cnt_tick_freq /= 1;
      break;
  case TIM_CLOCKDIVISION_DIV2:
      tim_led_cnt_tick_freq /= 2;
      break;
  case TIM_CLOCKDIVISION_DIV4:
      tim_led_cnt_tick_freq /= 4;
      break;
  }
 tim_led_cnt_tick_freq /= (tim_led->Instance->PSC + 1);
```

Для задания необходимой частоты ШИМ сигнала необходимо знать частоту инкремента счетчика таймера. Она рассчитывается следующим образом:

$$f_{CNT} = \frac{f_{SYSCLK}}{CKD(1+PSK)},$$

где:

f_{SYSCLK} – частота ядра. В библиотеке CMSIS ее значение хранится в глобальной переменной SystemCoreClock.

CKD – значение первого предделителя. Его можно получить с помощью макроса `__HAL_TIM_GET_CLOCKDIVISION`. Макрос возвращает одну из следующих констант: `TIM_CLOCKDIVISION_DIV1`, `TIM_CLOCKDIVISION_DIV2`, `TIM_CLOCKDIVISION_DIV4`, которые нужно преобразовать в 1, 2 или 4.

PSK – значение второго предделителя. Оно храниться в поле Instance→PSC структуры таймера `TIM_HandleTypeDef`.

3. Запуск таймера:

```
// reset CCR register
__HAL_TIM_SET_COMPARE(tim_led, tim_led_channel, 0);
// run timer
HAL_TIM_PWM_Start(tim_led, tim_led_channel);
```

При использовании таймеров с библиотекой HAL, их необходимо стартовать явно. Для запуска таймера в режиме ШИМ, используется функция `HAL_TIM_PWM_Start`. Первым аргументом она принимает структуру с описанием таймера. Вторым – константу с номером канала.

Чтобы таймер сразу не начал генерировать сигнал, имеет смысл установить ширину импульса ШИМ сигнала в 0 с помощью макроса `__HAL_TIM_SET_COMPARE`.

4. Задания параметров ШИМ сигнала:

```
// PWM signal
// frequency:  2000 Hz
// duty cycle: 0.2
arr_value = tim_led_cnt_tick_freq / 2000 - 1;
cc_value = (arr_value + 1) * 0.2f;
__HAL_TIM_SET_AUTORELOAD(tim_led, arr_value);
  __HAL_TIM_SET_COMPARE(tim_led, tim_led_channel,
cc_value);
```

Частота ШИМ сигнала $f = \frac{f_{cnt}}{(1+ARR)}$, где f_{CNT} – частота инкремента счетчика таймера; ARR – значение «auto reload register», который содержит максимально значение счетчика CNT, после которого он сбрасывается в 0.

Зная f_{CNT} и необходимое значение f, мы можем рассчитать значение ARR регистра и установить его с помощью макроса `__HAL_TIM_SET_AUTORELOAD`.

Для задания нужного коэффициента заполнения *D* необходимо рассчитать значение Capture/Compare регистра (рис. 6.2). Оно рассчитывается следующим образом: $CC=(1+ARR)\cdot D$ и устанавливается с помощью макроса `__HAL_TIM_SET_COMPARE`.

Особенности расчета значений регистров *ARR*, *CC*:

– если необходимо изменить частоту ШИМ сигнала без изменения коэффициента заполнения, то после обновления *ARR* необходимо пересчитать и обновить советующий *CC* регистр;

– большая часть таймеров имеет 16-битные регистры *ARR* и *CC*, и они могут содержать только числа в диапазоне от 0 до 65535;

– При установке *ARR* в 0, таймер блокируется. Для упрощения кода следует избегать данного случая.

6.4. Практические задания к разделу 6

Реализовать управление яркостью двух светодиодов (пины PE_9, PE_13), проигрыванием заданных нот и мелодий. Управление осуществлять матричной клавиатурой НС-35. Действия на кнопки назначить в соответствии с табл. 6.1.

Таблица 6.1

Распределение действий на кнопки

Кнопка	Действие
K1	увеличить яркость первого светодиода
K2	уменьшить яркость первого светодиода
K3	увеличить яркость второго светодиода
K4	уменьшить яркость второго светодиода
K5	играть ноту 1, пока нажата кнопка
K6	играть ноту 2, пока нажата кнопка
K7	играть ноту 3, пока нажата кнопка
K8	играть ноту 4, пока нажата кнопка
K9	играть ноту 5, пока нажата кнопка
K10	играть ноту 6, пока нажата кнопка
K11	играть ноту 7, пока нажата кнопка
K12	играть ноту 8, пока нажата кнопка
K13	запустить проигрывание мелодии 1
K14	запустить проигрывание мелодии 2
K15	запустить проигрывание мелодии 3
K16	запустить проигрывание мелодии 4

Увеличение и уменьшение яркости можно реализовать по однократному нажатию (1 клик изменяет яркость на 10%) или по длительному (пока кнопка нажата, яркость постоянно уменьшается или увеличивается).

При нажатии кнопки запуска мелодии, должно начаться ее проигрывание. После окончания мелодии, проигрывание останавливается. Если во время проигрывания мелодии нажата любая из кнопок K5 – K16, проигрывание должно остановиться, и далее необходимо перейти к обработке действия соответствующей кнопки.

Для создания ноты заданной частоты следует сформировать меандр с помощью таймера.

Мелодии находятся в файлах `melody_<n>.h` и `melody_<n>.c` (приложение 1) и представляют собой простые тоновые мелодии, у которых в каждый момент времени может исполняться только одна нота. Каждый файл мелодий содержит три глобальные переменные:

- `MELODY_<N>_LEN` – количество нот в мелодии;
- `MELODY_<N>_FREQUENCIES` – частоты нот в герцах;
- `MELODY_<N>_DURATIONS` – длительность каждой ноты.

Ноты проигрываются последовательно в порядке их следования в массиве `MELODY_<N>_FREQUENCIES`. Значение частоты ноты 0 означает, что нужно сделать паузу вместо проигрывания звука.

Частоты нот и мелодии задаются в соответствии с вариантом.

Варианты для индивидуальной работы

Вариант 1

Нота	Частота	Мелодия	Файлы мелодий
1	131 Гц	1	melody_1.h, melody_1.c
2	147 Гц	2	melody_3.h, melody_3.c
3	165 Гц	3	melody_5.h, melody_5.c
4	175 Гц	4	melody_7.h, melody_7.c
5	196 Гц		
6	220 Гц		
7	247 Гц		
8	262 Гц		

Вариант 2

Нота	Частота	Мелодия	Файлы мелодий
1	262 Гц	1	melody_2.h, melody_2.c
2	294 Гц	2	melody_4.h, melody_4.c
3	330 Гц	3	melody_6.h, melody_6.c
4	350 Гц	4	melody_8.h, melody_8.c
5	392 Гц		
6	440 Гц		
7	494 Гц		
8	524 Гц		

Вариант 3

Нота	Частота	Мелодия	Файлы мелодий
1	524 Гц	1	melody_3.h, melody_3.c
2	588 Гц	2	melody_5.h, melody_5.c
3	660 Гц	3	melody_7.h, melody_7.c
4	700 Гц	4	melody_9.h, melody_9.c
5	784 Гц		
6	880 Гц		
7	988 Гц		
8	1048 Гц		

Вариант 4

Нота	Частота	Мелодия	Файлы мелодий
1	1048 Гц	1	melody_1.h, melody_1.c
2	1176 Гц	2	melody_2.h, melody_2.c
3	1320 Гц	3	melody_3.h, melody_3.c
4	1400 Гц	4	melody_4.h, melody_4.c
5	1568 Гц		
6	1760 Гц		
7	1976 Гц		
8	2096 Гц		

Вариант 5

Нота	Частота	Мелодия	Файлы мелодий
1	2096 Гц	1	melody_5.h, melody_5.c
2	2352 Гц	2	melody_6.h, melody_6.c
3	2640 Гц	3	melody_7.h, melody_7.c
4	2800 Гц	4	melody_8.h, melody_8.c
5	3136 Гц		
6	3520 Гц		
7	3952 Гц		
8	4192 Гц		

Вариант 6

Нота	Частота	Мелодия	Файлы мелодий
1	4192 Гц	1	melody_1.h, melody_1.c
2	4704 Гц	2	melody_2.h, melody_2.c
3	5280 Гц	3	melody_8.h, melody_8.c
4	5600 Гц	4	melody_9.h, melody_9.c
5	6272 Гц		
6	7040 Гц		
7	7904 Гц		
8	8384 ГЦ		

Вариант 7

Нота	Частота	Мелодия	Файлы мелодий
1	131 Гц	1	melody_3.h, melody_3.c
2	147 Гц	2	melody_4.h, melody_4.c
3	165 Гц	3	melody_6.h, melody_6.c
4	175 Гц	4	melody_7.h, melody_7.c
5	196 Гц		
6	220 Гц		
7	247 Гц		
8	262 Гц		

Вариант 8

Нота	Частота	Мелодия	Файлы мелодий
1	262 Гц	1	melody_1.h, melody_1.c
2	294 Гц	2	melody_7.h, melody_7.c
3	330 Гц	3	melody_8.h, melody_8.c
4	350 Гц	4	melody_9.h, melody_9.c
5	392 Гц		
6	440 Гц		
7	494 Гц		
8	524 Гц		

Вариант 9

Нота	Частота	Мелодия	Файлы мелодий
1	524 Гц	1	melody_1.h, melody_1.c
2	588 Гц	2	melody_2.h, melody_2.c
3	660 Гц	3	melody_3.h, melody_3.c
4	700 Гц	4	melody_9.h, melody_9.c
5	784 Гц		
6	880 Гц		
7	988 Гц		
8	1048 Гц		

Вариант 10

Нота	Частота	Мелодия	Файлы мелодий
1	1048 Гц	1	melody_3.h, melody_3.c
2	1176 Гц	2	melody_4.h, melody_4.c
3	1320 Гц	3	melody_5.h, melody_5.c
4	1400 Гц	4	melody_6.h, melody_6.c
5	1568 Гц		
6	1760 Гц		
7	1976 Гц		
8	2096 Гц		

7. ВЗАИМОДЕЙСТВИЕ С ВНЕШНИМИ УСТРОЙСТВАМИ. LCD-ДИСПЛЕЙ

7.1. Основные сведения об LCD-дисплее на базе контроллера ILI9325

LCD-дисплеи оснащены сеткой проводников, где на пересечении строки и столбца стоит транзистор, отвечающий за регулировку прозрачности пикселя [35] (рис. 7.1).

Для формирования всего изображения контроллер дисплея должен пройтись по каждому пикселю и установить необходимый заряд на транзисторе, отвечающий за прозрачность пикселя, и, соответственно, пропускаемую яркость подсветки.

Однако заряды на транзисторе необходимо периодически обновлять в зависимости от спецификаций производителя дисплея, иначе дисплей будет сильно мерцать. Для этого контроллеру дисплея нужно постоянно получать текущую картинку извне или иметь свою графическую память (GRAM) для хранения текущего изображения.

Макет, схема которого представлена на рис. 7.2, построен на базе контроллера ILI9325 и использует дисплей с разрешением 320 на 240 пикселей. Он оснащен графической памятью размером 172820 байт с 18 битами на пиксель, блоком управляющих регистров и рядом интерфейсов для взаимодействия с микроконтроллером [36]. Из всех интерфейсов на макете реализован только 16-битный интерфейс i80. Схематично взаимодействие с контроллером представлено на рис. 7.2.

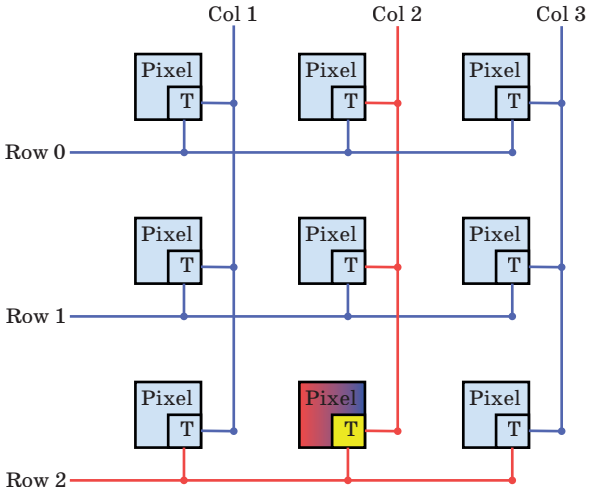


Рис. 7.1. Подключение пикселя на LCD-дисплее

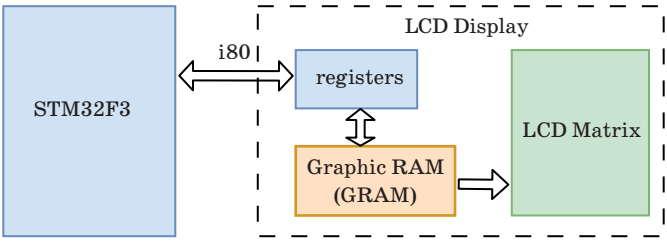


Рис. 7.2. Взаимодействие STM32 с LCD-дисплеем

Посредством интерфейса i80 происходит взаимодействие микроконтроллера с блоком регистров ILI9325. Регистры позволяют настраивать режимы работы дисплея и передавать данные в GRAM.

Все регистры контроллера являются 16-битными, имеют адреса от 0 до 0×A5 и делятся на следующие категории:

1) чтение статуса;
2) управление дисплеем;
3) управление питанием;
4) обработка графических данных;
5) установка внутреннего адреса GRAM (AC);
6) перемещение данных во внутреннюю память GRAM и из нее (R22);
7) внутренняя гамма-коррекция градаций серого (R30 ... R39).

Подробное описание регистров можно найти в документации к контроллеру. Поэтому мы рассмотрим только регистры, которые непосредственно понадобятся для формирования изображения.

Для записи данных (цветов пикселей) в GRAM необходимо установить прямоугольную область для записи данных и начальную позицию (рис. 7.3).

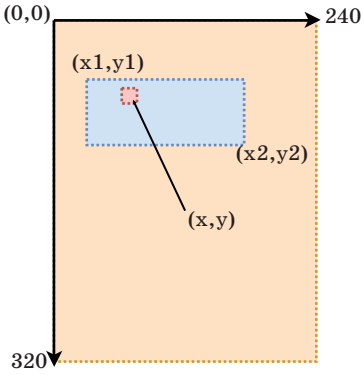


Рис. 7.3. Выбор области для передачи данных в GRAM

Координаты $x1$, $y1$ записываются в регистры 0×50, 0×52, $x2$, $y2$ – в 0×51, 0×53, x, y – в 0×20, 0×21. Цвета пикселей передаются в регистр 0×22. После каждой передачи пикселя, координаты x и y инкрементируются автоматически, чтобы построчно пройтись по области ($x1$, $y1$, $x2$, $y2$).

Отметим, что регистр для передачи данных в GRAM является 16-битным, в то время как сама GRAM выделяет под каждый пиксель 18 бит. Поэтому для передачи одного пикселя нужно совершить либо 2 операции записи в регистр (14 бит одной передачи будут отброшены), либо одну с преобразованием 16-битного цвета в 18-битный. Будем использовать последний подход с преобразованием цвета, показанный на рис. 7.4. Исходный цвет пикселя хранится в формате RGB565.

Чтение и запись значений регистров контроллера дисплея осуществляется через 16-битный интерфейс i80 [36]. Интерфейс использует 20 пин/линий. Из них 4 линии являются управляющими, а 16 линий – для передачи и приема данных, и передачи текущего адреса регистра (рис. 7.5).

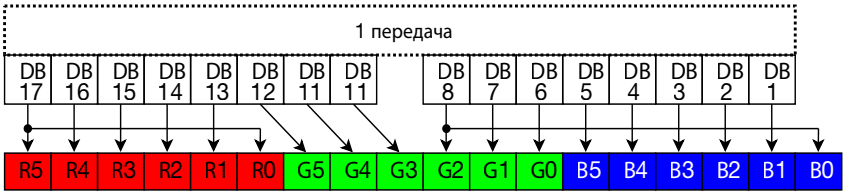


Рис. 7.4. Преобразование 16-битного цвета в 18-битный

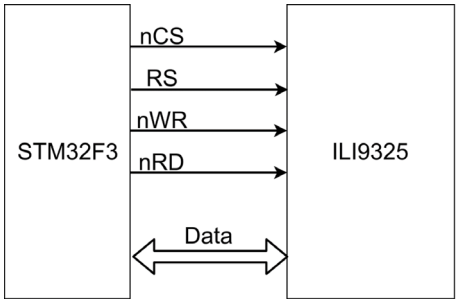


Рис. 7.5. Схема интерфейса i80

Управляющие линии имеют следующие назначения:

– nCS – линия выбора чипа. Состояние по умолчанию – логическая 1. Каждый раз, когда происходит взаимодействие с устройством, ее нужно переводить в логический 0.

– RS – выбор режима передачи данных/адреса регистра. Это необходимо, так как интерфейс i80 использует одну шину для передачи адреса регистра и значений самих регистров. Если RS установлена в логическую 1 – шина используется для передачи данных, 0 – для передачи адреса регистра.

– nWR – линия сигнала записи данных. Состояние по умолчанию – 1. После того как MCU выставил данные на шину, необходимо подать короткий нулевой импульс, чтобы ILI9325 считал их.

– nRD – линия сигнала чтения данных. Состояния по умолчанию – 1. Работает по аналогии с nWR, но используется, когда нужно получить данные от ILI9325.

Для работы с i80 у ряда микроконтроллеров STM32 имеется блок FSMC (Flexible static memory controller) [21]. Но на контроллере макетной платы он отсутствует, поэтому используется подход программной эмуляции интерфейсов ввода-вывода.

Использование bit-banging для реализации интерфейса i80

Bit-banging – технология организации последовательного соединения с использованием программной эмуляции вместо специализированного аппаратного устройства [37, 38]. При данном подходе программа микроконтроллера сама устанавливает нужные состояния выводов GPIO и обеспечивает нужные тайминги. Это дополнительно расходует вычислительные ресурсы микроконтроллера, и зачастую эмулируемые интерфейсы являются более медленными, чем их аппаратные аналоги, но решает проблему использования нужного интерфейса, если отсутствует его аппаратная реализация или нужные выводы уже заняты. Рассмотрим bit-banging для реализации интерфейса i80 для ILI9325.

Запись в регистр ILI9325:

1. Установить nCS в 0.

2. Передать адрес регистра:

1) установить RS в 0;

2) выставить на линии DATA адрес регистра;

3) установить nWR в 0;

4) установить nWR в 1;

5) установить RS в 1.

3. Передать данные:

1) выставить на линии DATA данные, которые нужно передать;

2) установить nWR в 0;

3) установить nWR в 1;

4. Установить nCS в 1.

Для записи нескольких блоков данных по 16 бит в один регистр необходимо несколько раз подряд повторить пункт 3 (передача данных).

Диаграмма состояния линий при записи в регистр представлена на рис. 7.6 и 7.7.

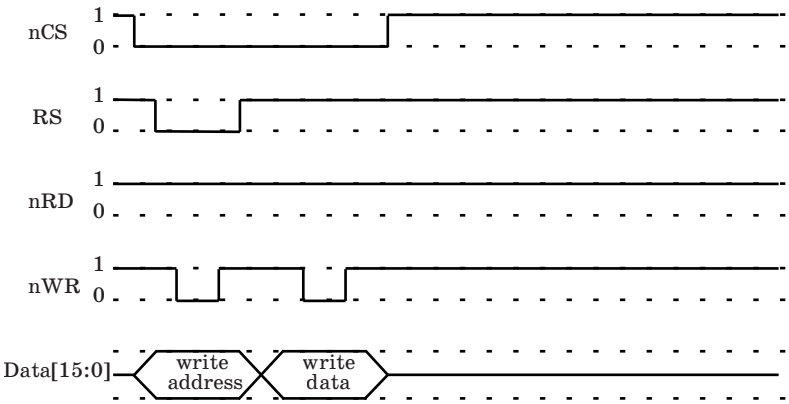


Рис. 7.6. Запись данных в регистр

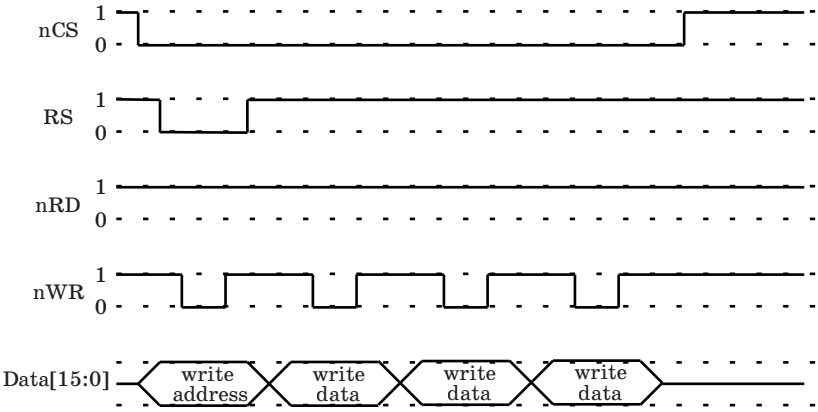


Рис. 7.7. Запись нескольких слов подряд в один и тот же регистр

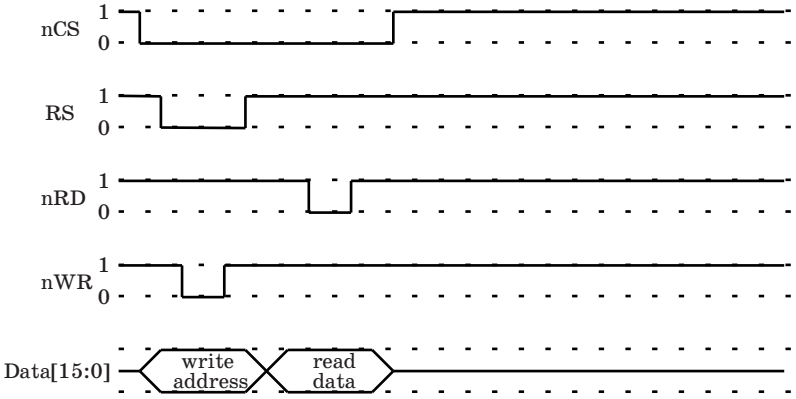


Рис. 7.8. Запись данных в регистр

Чтение регистра ILI9325:
1. Установить nCS в 0.
2. Передать адрес регистра:
1) установить RS в 0;
2) выставить на линии DATA адрес регистра;
3) установить nWR в 0;
4) установить nWR в 1;
5) установить RS в 1.
3. Чтение данных:
1) переключить GPIO режим линии DATA на выход;
2) установить nRD в 0;
3) установить nRD в 1;
4) считать данные с линии DATA;
5) переключить GPIO режим линии DATA обратно на выход.
4. Установить nCS в 1.

Диаграмма состояния линий при чтении регистра представлена на рис. 7.8.

7.2. Программирование микроконтроллера STM32 с дисплеем ILI9325

Подготовка проекта в STM32CubeMX

1. Создайте проект для STM32F3Discovery и выполните базовую настройку.

2. Переведите пины PD0-PD15, PB8-PB11, PC10-PC11 в режим GPIO_Output.

3. Откройте настройки пинов и выполните следующее:

1) PD0-PD15, PB8-PB11 – установить высокий уровень сигнала по умолчанию, режимы «Output Push Pull» и «No pull up pull down», и высокую скорость переключения пинов;

2) PC10-PC11 – установить высокий уровень сигнала по умолчанию, режимы «Output Open Drain» и «Pull up», и высокую скорость переключения пинов.

4. Завершите оставшиеся шаги настройки проекта и запустите генерацию проекта под STM32CubeIDE.

Работа с проектов в STM32CubeIDE

1. Для упрощения работы с дисплеем, в приложении 2 имеются следующие файлы, которые необходимо добавить в проект:

1) open32f3_lcd_utils.h и open32f3_lcd_utils.h – вспомогательный макросы и функции для работы с GPIO:

- `OPEN32F3_LCD_CLEAR_CS()` – очистка CS пина;
- `OPEN32F3_LCD_SET_CS()` – установка CS пина;
- `OPEN32F3_LCD_CLEAR_RS()` – очистка RS пина;
- `OPEN32F3_LCD_SET_RS()` – установка RS пина;
- `OPEN32F3_LCD_CLEAR_WR()` – очистка WR пина;
- `OPEN32F3_LCD_SET_WR()` – установка WR пина;
- `OPEN32F3_LCD_CLEAR_RD()` – очистка RD пина;
- `OPEN32F3_LCD_SET_RD()` – установка RD пина;
- `OPEN32F3_LCD_DATA_READ()` – чтение данных с шины;
- `OPEN32F3_LCD_DATA_WRITE(value)` – запись данных в шину;
- `OPEN32F3_LCD_DATA_SET_INPUT_MODE()` – переключение пинов шины данных в режим чтения;
- `OPEN32F3_LCD_DATA_SET_OUTPUT_MODE()` – переключение пинов шины данных в режим вывода;

2) lcd_ili93xx_driver.h и lcd_ili93xx_driver.c – вспомогательный код инициализации и работы с дисплеем. Код драйвера принимает колбеки для приема и передачи данных по i80 и предоставляет базовый интерфейс для задания цветов пикселей дисплея.

2. Добавьте в main.c следующий код (части, сгенерированные CubeMX, в нем опущены):

```
#include "lcd_ili93xx_driver.h"
#include "open32f3_lcd_utils.h"

int app_lcd_write_register(void *user_data, uint16_t address,
uint16_t value) {
      // реализация записи значения "value" в регистр
"address"
    // ...

    return 0;
}

int app_lcd_write_words(void *user_data, uint16_t address,
uint16_t *data, size_t size) {
     // реализация записи массива "data" размером "size"
элементов в регистр "address"
    // ...

    return 0;
}

int app_lcd_read_reagiter(void *user_data, uint16_t address,
uint16_t *value) {
     // реализация чтения значения из регистра "address"
в "value"
    // ...

    return 0;
}

int main() {

    //
    // базовый код main ...
    //

    int err;
    // инициализация драйвера
    lcd_ili93xx_driver_t lcd_driver;

    lcd_ili93xx_init_clear(&lcd_driver);
    lcd_driver.user_data = NULL;
    lcd_driver.read_reg = app_lcd_read_reagiter;
```

```
    lcd_driver.write_reg = app_lcd_write_register;
    lcd_driver.write_words = app_lcd_write_words;
    lcd_driver.reset = open32f3_lcd_reset;
    lcd_driver.delay = open32f3_lcd_delay;
    err = lcd_ili93xx_init(&lcd_driver);
    if (err) {
         // вывести сообщения об ошибки и прекратить вы-
полнение программы
        // ...
        //
    }

    // нарисовать красно-зеленые клеточки
    for (int x = 0; x < 240; x += 20) {
        for (int y = 0; y < 320; y += 20) {
            uint16_t color;
            if ((x + y) % 40 == 0) {
                color = LCD_ILI93XX_COLOR_GREEN;
            } else {
                color = LCD_ILI93XX_COLOR_MAROON;
            }
            lcd_ili93xx_fill_area_color(&lcd_driver, x, y,
x + 19, y + 19, color);
        }
    }

    // остальная логика main
    // ...

}
```

3. В нем необходимо реализовать колбеки app_lcd_read_register, app_lcd_write_register и app_lcd_write_words, используя временные i80 (см. рис. 7.6, 7.7 и 7.8) и макросы из «open32f3_lcd_utils.h».

Если все реализовано правильно, то на экране дисплея будут нарисованы красные и зеленые квадраты.

Драйвер из «lcd_ili93xx_driver.h» предоставляет следующие базовые функции для рисования на дисплее:

- `lcd_ili93xx_fill_area_color` – заливка указанной прямоугольной области заданным цветом;
- `lcd_ili93xx_fill_area` – заполнение указанной прямоугольной области заданным изображением. Изображение представляет собой двухмерный массив 16-битных пикселей.

7.3. Основные сведения о сенсорной панели на базе контроллера XPT2046

Рассмотрим подробнее резистивный экран с контроллером XPT2046, реализованный в дисплее макетной платы.

Резистивные сенсорные экраны

Резистивные панели имеют многослойную структуру, состоящую из двух проводящих поверхностей, разделенных специальным изолирующим составом, распределенным по всей площади активной области экрана (рис. 7.9) [39]. При касании наружного слоя, выполненного из тонкого прозрачного пластика, его внутренняя проводящая поверхность совмещается с проводящим слоем основной пластины, благодаря чему происходит изменение сопротивления всей системы. Это изменение фиксируется контроллером панели.

В рассматриваемом макете используется четырехпроводной сенсор XPT2046. Его эквивалентная схема сопротивлений при нажатии на экран показана на рис. 7.10. Величины сопротивлений $R1$, $R2$, $R3$ и $R4$ пропорциональны длине участка проводящей пластины от ее начала до места касания. Относительные координаты касания связаны с ними следующим образом:

$$X_{rel} = \frac{R1}{R1 + R2}, \quad Y_{rel} = \frac{R3}{R3 + R4}.$$

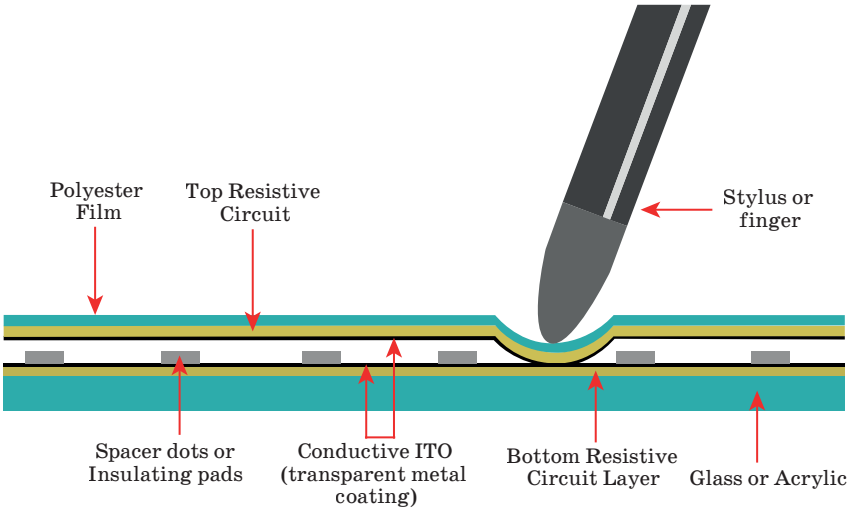


Рис. 7.9. Устройство резистивной сенсорной панели

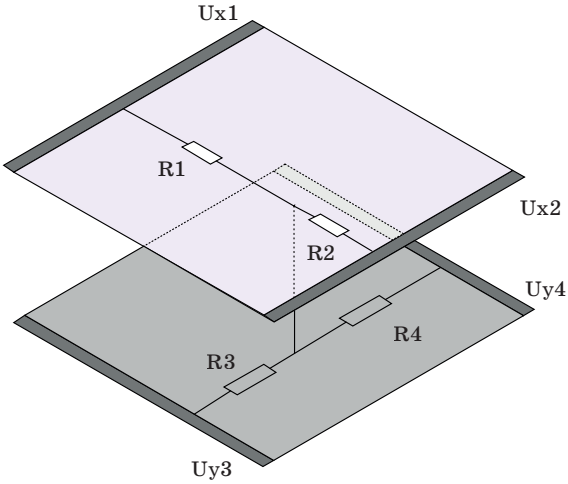


Рис. 7.10. Схема эквивалентных сопротивлений при нажатии на резистивный экран

Для вычисления координат X, верхняя пластина подключается к земле и питанию, а нижняя – к входу АЦП (рис. 7.11). Для координаты Y – нижняя пластина подключается к земле и питанию, а верхняя – к входу АЦП (рис. 7.12). В результате относительные координаты можно вычислить как

$$X_{rel} = \frac{V_x}{V_{\max}}, \quad Y_{rel} = \frac{V_y}{V_{\max}},$$

где $V_{\max}$= 4095 – максимальное значение показаний 12-битного АЦП, V_x – показание АЦП при считывании напряжения с нижней пластины, V_y – с верхней.

Определение факта касания происходит следующим образом. Контроллер подключает вывод X_N к земле, Y_P – к питанию и делает замеры Z_1 и Z_2 в точках X_P и Y_N (рис. 7.13). Если абсолютная разница между замерами V_{Z1} и V_{Z2} велика, то касания нет, иначе – есть. Для лабораторных макетов можно выбрать следующий порог:

$$\begin{cases} |V_{Z1} - V_{Z1}| \geq 3800 - \text{касания нет;} \\ |V_{Z1} - V_{Z1}| < 3800 - \text{касание обнаружено.} \end{cases}$$

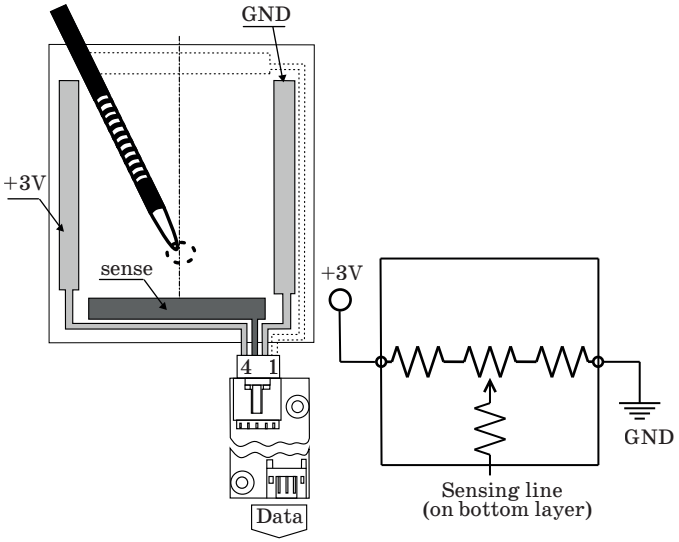


Рис. 7.11. Измерение координаты X

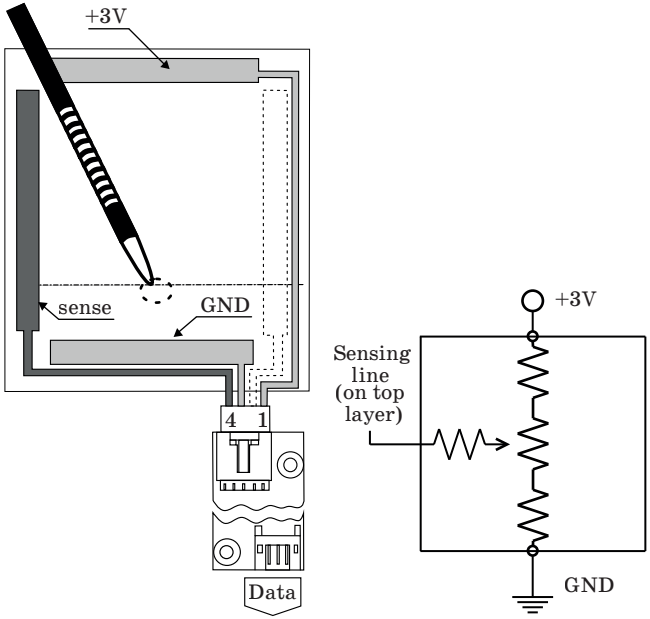


Рис. 7.12. Измерение координаты Y

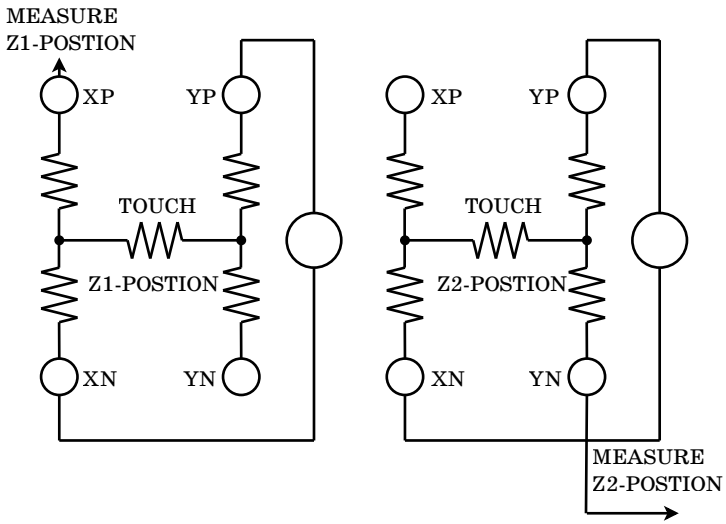


Рис. 7.13. Измерение Z1 и Z2 для детектирования касания

SPI интерфейс

Сенсор дисплея управляется чипом XPT2046. Для взаимодействия с ним используется SPI интерфейс [40]. Для подключения он требует 4 провода:

– MOSI – выход ведущего, вход ведомого (*Master Out Slave In*). Служит для передачи данных от ведущего устройства ведомому.

– MISO – вход ведущего, выход ведомого (*Master In Slave Out*). Служит для передачи данных от ведомого устройства ведущему.

– SCLK или SCK – последовательный тактовый сигнал (*Serial Clock*). Служит для передачи тактового сигнала для ведомых устройств.

– CS или SS – выбор микросхемы, выбор ведомого (*Chip Select, Slave Select*).

С помощью SPI можно подключить одно или несколько устройств, выделив для каждого отдельный CS пин (рис. 7.14).

Типичная схема передачи данных по SPI представлена на рис. 7.15. Из особенностей SPI отметим следующие [21]:

– он позволяет одновременно передавать и получать данные;

– размер пакета обычно составляет 1 байт (8 бит);

– интерфейс широко распространен и реализован во множестве микроконтроллеров, в том числе в STM32, с учетом требований к таймингам тактового сигнала и сигналам с данными.

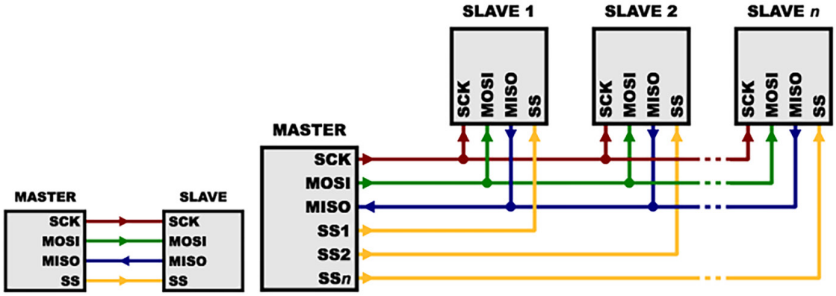


Рис. 7.14. Подключение одного или нескольких SPI устройств

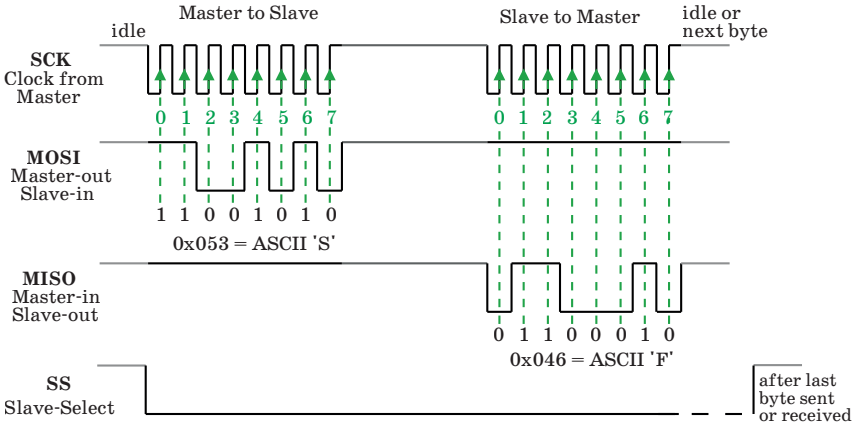


Рис. 7.15. Передача данных по SPI

Для практической работы с SPI необходимо знать только два параметра, которые находятся в документации к используемым устройствам:

– частота интерфейса;

– режим работы.

Режим работы определяется полярностью тактового сигнала (CPOL) и сдвигом его фазы относительно сигналов с данными. Возможны следующие комбинации (рис. 7.16):

– режим 0 (CPOL = 0, CPHA = 0);

– режим 1 (CPOL = 0, CPHA = 1);

– режим 2 (CPOL = 1, CPHA = 0);

– режим 3 (CPOL = 1, CPHA = 1).

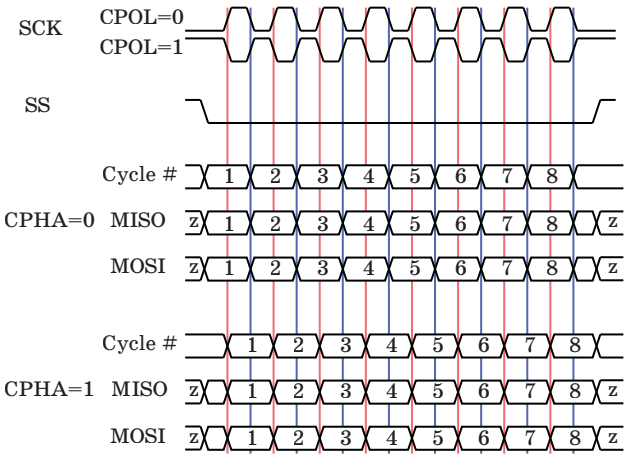


Рис. 7.16. Временные диаграммы работы SPI в разных режимах

7.4. Программирование микроконтроллера STM32 с дисплеем XPT2046

Подготовка проекта в STM32CubeMX

1. Откройте текущий проект. На вкладке «Pinout & Configuration» выберите «Categories» → «Connectivity» → «SPI». Включите mode «Full-Duplex master» и установите следующие опции:

– Data Size: 8 Bits;

– CPOL: low;

– CHAP: 1 Edge;

– Prescaler: 32 (частота SPI должна быть не выше 2 МГц для XPT2046).

2. Переведите пин PC13 в режим работы «GPIO_Output». Данный пин управляет линией CS. Настройки пина следующие:

– GPIO output level: High;

– GPIO mode: Output Open Drain;

– GPIO Pull-up/Pull-down: Pull up.

3. Обновите проект, нажав на «Generate Code».

Работа с проектом в STM32CubeIDE

1. Добавьте в проект файлы из приложения 3:

– open32f3_lcd_touch_utils.h;

– open32f3_lcd_touch_utils.c.

В добавленных файлах содержится реализация протокола для считывания значений сенсорной панели в точках X, Y, $Z1$, $Z2$.

2. Добавьте следующий код в проект для использования драйвера сенсора (код сгенерированный CubeMX опущен для упрощения):

```
#include "open32f3_lcd_touch_utils.h"

// сервисный код CubeMX и т.п.
// ...

/**
 * Колбэк данных по SPI для XPT2046
 */
static int app_lcd_touch_callback(void *user_data, uint8_t
*out_buf, uint8_t *in_buf, size_t size) {
    // установить CS линию в 0 для начала приема/передачи
   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    // передать/принять данные
 HAL_StatusTypeDef status = HAL_SPI_TransmitReceive(&hspi1,
out_buf, in_buf, size, HAL_MAX_DELAY);
    // установить CS линию в 1 для конца приема/передачи
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
    return status == HAL_OK ? 0 : -1;
}

int main {

    // сервисный код CubeMX и т.п.
    // ...

    // объявление и инициализация драйвера xpt2046
    lcd_xpt2046_driver_t touch_driver;
    lcd_xpt2046_init_clear(&touch_driver);
    touch_driver.user_data = NULL;
  touch_driver.communication_cb = app_lcd_touch_callback;
    int err = lcd_xpt2046_init(&touch_driver);
    if (err) {
        // код обработки ошибки
        // ...
    }

    // считывать и выводить измеренные значения в точках
X, Y, Z1, Z2
    while (1) {
        int x, y, z1, z2;
```

```
          lcd_xpt2046_measure(&touch_driver, XPT2046_CMD_
MEASURE_X, &x);
          lcd_xpt2046_measure(&touch_driver, XPT2046_CMD_
MEASURE_Y, &y);
          lcd_xpt2046_measure(&touch_driver, XPT2046_CMD_
MEASURE_Z1, &z1);
          lcd_xpt2046_measure(&touch_driver, XPT2046_CMD_
MEASURE_Z2, &z2);
        printf("| x=%04i | y=%04i | z1=%04i | z2=%04i |\n",
x, y, z1, z2);
         HAL_Delay(100);
    }
}
```

Если `printf` настроен правильно, то через UART вы сможете наблюдать измеренные значения в точках X, Y, $Z1$ и $Z2$.

Обработка значений X, Y, Z1, Z2

После запуска приведенной выше программы и касания дисплея, вы увидите следующие особенности:

– значения X, Y, $Z1$, $Z2$ зашумлены и постоянно колеблются;

– значения X, Y лежат в диапазоне от 0 до 4095, что не позволяет напрямую использовать их для работы с дисплеем размером 320×240.

Для решения первой проблемы необходимо сделать несколько измерений и взять среднее.

Для решения второй задачи следует применить следующие формулы перевода координат от XPT2046 (X_{raw}, Y_{raw}) в координаты дисплея (X_{disp}, Y_{disp}):

$$X_{disp} = k_x X_{raw} + b_x;$$

$$Y_{disp} = k_y Y_{raw} + b_y.$$

В качестве начальных значений коэффициентов можно взять следующие значения:

$$k_x = \frac{240}{4096} = 0{,}059;$$

$$k_y = \frac{320}{4096} = 0{,}078;$$

$$b_x = 0;$$

$$b_y = 0.$$

В процессе тестирования дисплея их нужно будет уточнить для определения места касания.

7.5. Практические задания к разделу 7

Реализовать демонстрационное приложение использования LCD-дисплея. Приложение нужно выполнять в соответствии с вариантом:

1) провести обработку данных от резистивного сенсора для уменьшения шумов: при измерении значений в точках X, Y, $Z1$, $Z2$ сделать по 3 измерения в каждой точке (X_1, X_2, X_3, Y_1, …) и усреднить их по заданной формуле;

2) отображать заданный курсор в месте касания. Бинарные маски курсоров размером 16 на 16 необходимо взять из файлов lab_5_icons.h, lab_5_icons.c (файлы находятся в приложении 4). Пиксель маски со значением 0 соответствует фону, 1 – курсору;

3) изменять параметры сцены (цвет фона, цвет курсора, форма курсора) при нажатии на кнопки матричной клавиатуры.

Варианты для индивидуальной работы

Вариант 1

– формула обработки данных от резистивного сенсора: среднеарифметическое значение $X = \dfrac{X_1 + X_2 + X_3}{3}$;

– форма курсора при касании дисплея: cursor_1_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): курсор отсутствует;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки K1-K4: изменить текущую форму курсора (к начальному курсору выбрать еще любые 3 формы отображения).

Вариант 2

– формула обработки данных от резистивного сенсора: медиана

$$X = \begin{cases} X_1, \ X_3 \le X_1 < X_2 \text{ или } X_2 \le X_1 < X_3 \\ X_2, \ X_1 \le X_2 < X_3 \text{ или } X_3 \le X_2 < X_1 \\ X_3, \ X_1 \le X_3 < X_2 \text{ или } X_2 \le X_3 < X_1 \end{cases};$$

– форма курсора при касании дисплея: cursor_2_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): cursor_3_image;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки K1-K4: изменить цвет курсора (выбрать любые 4 цвета).

Вариант 3

– формула обработки данных от резистивного сенсора: среднее значение ближайших точек

$$X = \begin{cases} \dfrac{X_1 + X_2}{2}, |X_1 - X_2| \le |X_2 - X_3| \text{ и } |X_1 - X_2| \le |X_1 - X_3| \\ \dfrac{X_1 + X_3}{2}, |X_1 - X_3| \le |X_2 - X_3| \text{ и } |X_1 - X_3| \le |X_1 - X_2| \\ \dfrac{X_2 + X_3}{2}, |X_2 - X_3| \le |X_1 - X_2| \text{ и } |X_2 - X_3| \le |X_1 - X_3| \end{cases};$$

– форма курсора при касании дисплея: cursor_4_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): курсор отсутствует;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки K1-K4: изменить цвет фона (выбрать любые 4 цвета).

Вариант 4

– формула обработки данных от резистивного сенсора: среднеарифметическое значение $X = \dfrac{X_1 + X_2 + X_3}{3}$;

– форма курсора при касании дисплея: cursor_5_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): cursor_6_image;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки K1-K4: изменить цвет фона (выбрать любые 4 цвета).

Вариант 5

– формула обработки данных от резистивного сенсора: медиана

$$X = \begin{cases} X_1, \ X_3 \le X_1 < X_2 \text{ или } X_2 \le X_1 < X_3 \\ X_2, \ X_1 \le X_2 < X_3 \text{ или } X_3 \le X_2 < X_1 \\ X_3, \ X_1 \le X_3 < X_2 \text{ или } X_2 \le X_3 < X_1 \end{cases};$$

– форма курсора при касании дисплея: cursor_7_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): курсор отсутствует;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки К1-К4: изменить текущую форму курсора (к начальному курсору выбрать еще любые 3 формы отображения).

Вариант 6

– формула обработки данных от резистивного сенсора: среднее значение ближайших точек

$$X = \begin{cases} \dfrac{X_1 + X_2}{2}, |X_1 - X_2| \le |X_2 - X_3| \text{ и } |X_1 - X_2| \le |X_1 - X_3| \\ \dfrac{X_1 + X_3}{2}, |X_1 - X_3| \le |X_2 - X_3| \text{ и } |X_1 - X_3| \le |X_1 - X_2| \\ \dfrac{X_2 + X_3}{2}, |X_2 - X_3| \le |X_1 - X_2| \text{ и } |X_2 - X_3| \le |X_1 - X_3| \end{cases};$$

– форма курсора при касании дисплея: cursor_8_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): cursor_9_image;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки К1-К4: изменить цвет курсора (выбрать любые 4 цвета).

Вариант 7

– формула обработки данных от резистивного сенсора: среднеарифметическое значение $X = \dfrac{X_1 + X_2 + X_3}{3}$;

– форма курсора при касании дисплея: cursor_10_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): курсор отсутствует;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки К1-К4: изменить текущую форму курсора (к начальному курсору выбрать еще любые 3 формы отображения).

Вариант 8

– формула обработки данных от резистивного сенсора: медиана

$$X = \begin{cases} X_1, \ X_3 \le X_1 < X_2 \text{ или } X_2 \le X_1 < X_3 \\ X_2, \ X_1 \le X_2 < X_3 \text{ или } X_3 \le X_2 < X_1 \\ X_3, \ X_1 \le X_3 < X_2 \text{ или } X_2 \le X_3 < X_1 \end{cases};$$

– форма курсора при касании дисплея: cursor_1_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): cursor_5_image;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки К1-К4: изменить цвет курсора (выбрать любые 4 цвета).

Вариант 9

формула обработки данных от резистивного сенсора: среднее значение ближайших точек

$$X = \begin{cases} \dfrac{X_1 + X_2}{2}, |X_1 - X_2| \le |X_2 - X_3| \text{ и } |X_1 - X_2| \le |X_1 - X_3| \\ \dfrac{X_1 + X_3}{2}, |X_1 - X_3| \le |X_2 - X_3| \text{ и } |X_1 - X_3| \le |X_1 - X_2| \\ \dfrac{X_2 + X_3}{2}, |X_2 - X_3| \le |X_1 - X_2| \text{ и } |X_2 - X_3| \le |X_1 - X_3| \end{cases};$$

– форма курсора при касании дисплея: cursor_4_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): cursor_6_image;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки К1-К4: изменить цвет фона (выбрать любые 4 цвета).

Вариант 10

– формула обработки данных от резистивного сенсора: среднеарифметическое значение $X = \frac{X_1 + X_2 + X_3}{3}$;

– форма курсора при касании дисплея: cursor_7_image;

– форма курсора при отсутствии касаний (отображать в месте последнего касания): курсор отсутствует;

– цвет фона: любой;

– цвет курсора: любой;

– реакция при нажатии на кнопки K1-K4: изменить текущую форму курсора (к начальному курсору выбрать еще любые 3 формы отображения).

8. ГРАФИЧЕСКИЕ ИНТЕРФЕЙСЫ ВО ВСТРАИВАЕМЫХ ПРИЛОЖЕНИЯХ. БИБЛИОТЕКА LVGL

8.1. Библиотеки для построения графических интерфейсов

Для создания графических интерфейсов во встраиваемых устройствах существует ряд библиотек. Перечислим некоторые из них:

– LVGL (https://lvgl.io/);

– uGFX (https://ugfx.io/);

– TouchGFX (https://support.touchgfx.com/);

– GUIslice (https://github.com/ImpulseAdventure/GUIslice).

В отличие от обычных библиотек для построения графического интерфейса, они имеют следующие особенности:

– малое потребление памяти (десятки килобайт флеш-памяти и RAM);

– возможность отрисовки кадров сцены по частям для случаев, когда микроконтроллер не имеет достаточного количества памяти для всего кадра. Например, для хранения изображения разрешением 320 на 240 с 16 битами на пиксель требуется 150 KB памяти, в то время как микроконтроллер имеет только 40 KB;

– отрисовка всех элементов средствами самой библиотеки;

– использование средств аппаратного ускорения отрисовки при их наличии;

– поддержка цветов различной глубины (1 – monochrome, 8 – RGB332, 16 – RGB565, 32 – ARGB8888);

– поддержка разнообразных устройств ввода (сенсорный ввод, кнопки и т. д.).

– платформенно-независимая реализация.

Библиотека LVGL

LVGL предназначена для создания графических интерфейсов встраиваемых систем, имеет открытые исходные коды под лицензией MIT и хорошо документирована. Она спроектирована с учетом поддержки дисплеев разного типа (монохромные, цветные), разнообразных устройств ввода (клавиатура, мышь, сенсорный экран) и поддержки сразу нескольких дисплеев.

Подробную информацию можно найти на сайте библиотеки и github:

– https://lvgl.io/;

– https://github.com/lvgl/lvgl.

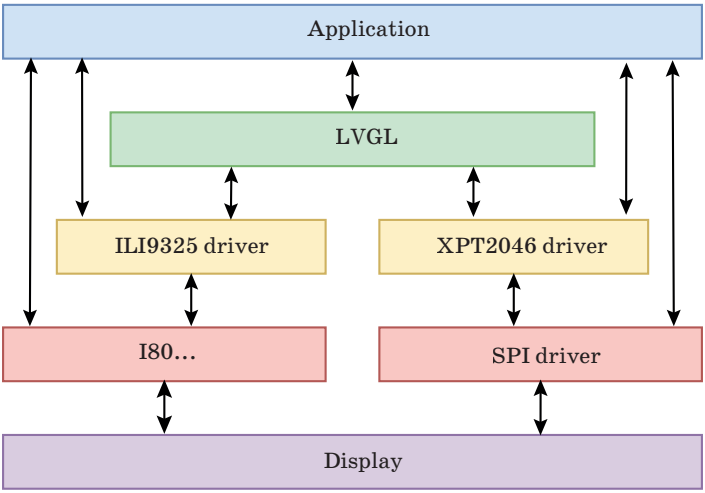


Рис. 8.1. Схема взаимодействия приложения и LVGL с дисплеем

Ввиду отсутствия программных стандартов интерфейсов дисплеев и устройств ввода, обычно для подключения библиотеки необходимо самостоятельно реализовать:

– функцию передачи изображения от библиотеки на экран;

– функцию считывания текущего положения курсора.

С использованием драйверов дисплея из предыдущей главы схема работы LVGL и пользовательского приложения показана на рис. 8.1.

Подключение драйверов дисплея и сенсора были рассмотрены в пп. 7.2 и 7.4, поэтому рассмотрим только часть подключения LVGL версии 8.2 к реализованным ранее драйверам.

8.2. Программирование микроконтроллера STM32 с использованием библиотеки LVGL

Настройка проекта в CubeMX

1. Повторите настройку дисплея и сенсорного экрана, как в пп. 7.2 и 7.4, или используйте ее в качестве основы.

2. Ввиду высоких вычислительных затрат на графику, перейдите на вкладку «Clock Configuration» и увеличьте частоту процессора с 48 до 72 MHz (рис. 8.2).

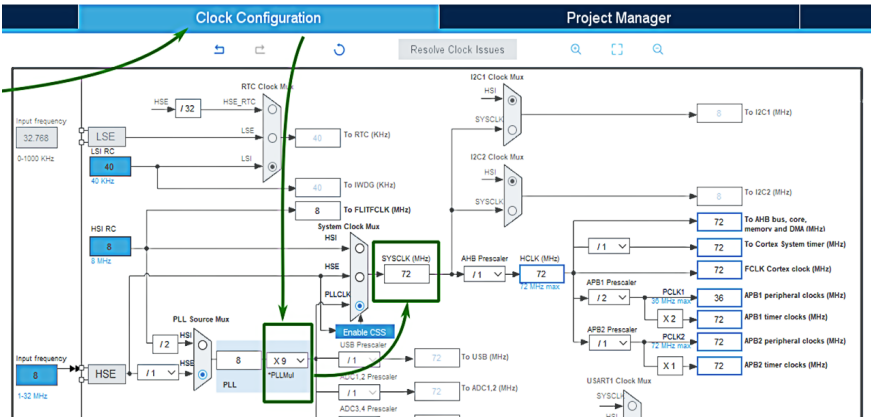


Рис. 8.2. Настройка проекта в CubeMX

3. Из-за увеличения частоты процессора автоматически увеличилась частота всех периферийных устройств. Поэтому необходимо проверить, что частота SPI сенсора касаний XPT2046 не превышает 2,5 MHz и подкорректировать ее в случае необходимости. Если вы используете другие устройства (таймеры и т. д.), то их частоты необходимо пересчитать и обновить.

4. Закончите создание проекта и откройте CubeIDE.

Добавление LVGL в проект

1. Скачайте (https://github.com/lvgl/lvgl/releases) и распакуйте библиотеку в корень проекта или в любую подпапку, например «Libs».

2. Откройте окно свойств проекта, перейдите «C/C++ General» → «Path and Symbols» → «Includes» и добавьте в «Includes directories» путь к распакованной папке (рис. 8.3).

3. Перейдите в «Source Location» и добавьте туда папку «src» из LVGL (рис. 8.4).

4. Перейдите в «Symbols» и добавьте символ «LV_CONF_INCLUDE_SIMPLE» (рис. 8.5).

5. Добавьте в папку проекта «Inc» подготовленный файл конфигурации «lvgl_conf.h» из приложения 5.

Подключения LVGL к дисплею

Рассмотрим пример кода подключения LVGL к дисплею макета. Код предполагает, что драйвер дисплея (lcd_ili93xx_driver_t) и сенсора (lcd_xpt2046_driver_t) уже настроены и подключены. Если нет, вернитесь к предыдущей главе работы.

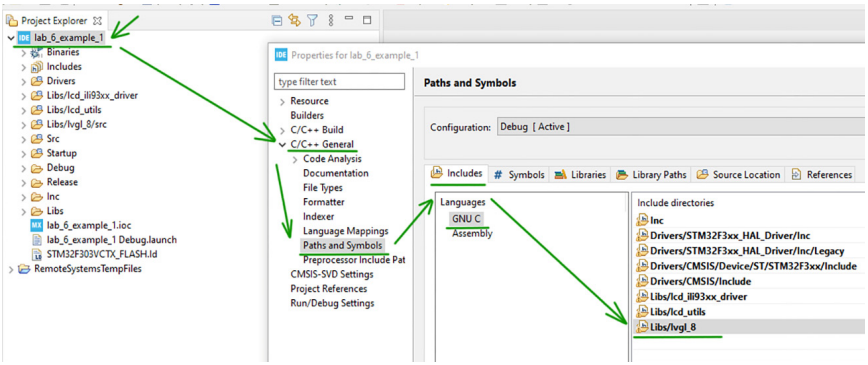


Рис. 8.3. Окно свойств проекта

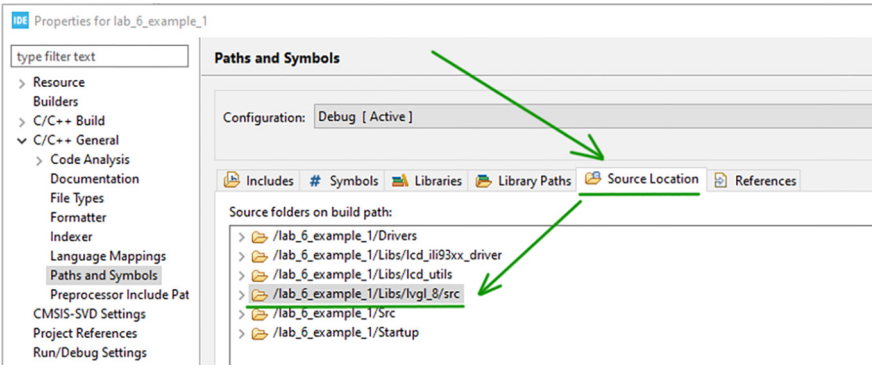


Рис. 8.4. Окно Source Location

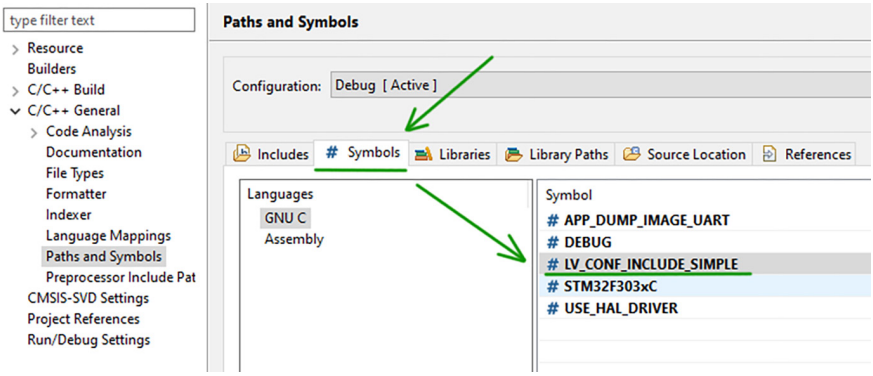


Рис. 8.5. Окно Symbols

```
//
// Подключение библиотек и прочее
//
#include "main.h"
#include "spi.h"
#include "gpio.h"

#include "stdio.h"
#include "stdlib.h"
#include "stdbool.h"

#include "lvgl.h"
#include "lcd_ili93xx_driver.h"
#include "open32f3_lcd_touch_utils.h"


/**
 * Вспомогательная структура с данными драйверов LVGL.
 */
typedef struct {
     // Драйверы дисплея и touch сенсора из предыдущего
раздела
    lcd_ili93xx_driver_t *lcd_driver;
    lcd_xpt2046_driver_t *touch_driver;

    // Данные для привязки LVGL к дисплею
    size_t buf_size;
    lv_color_t *buf;
    lv_disp_draw_buf_t disp_draw_buf;
    lv_disp_drv_t disp_driver;
    // Объект дисплея LVGL
    lv_disp_t *disp;

    // Данные для привязки LVGL к touch сенсору
    lv_indev_drv_t indev_drv;
    lv_indev_t *indev;
    int16_t last_x_position;
    int16_t last_y_position;
    // Любые пользовательские данные для пересчета данных
touch сенсора в координаты дисплея.
     // Если у вас нет подобных структур, можно удалить
данное поле.
    void *app_touch_data;
} app_open32f3_lvgl_driver_t;
```

```
/**
 * Колбек для передачи отрисованного интерфейса на дисплей.
 *
 * LVGL сам вызовет его, когда ему понадобиться передать данные на дисплей.
 */
static void app_open32f3_lvgl_display_flush_cb(lv_disp_drv_t *disp_drv, const lv_area_t *area, lv_color_t *color_p) {
    // получить app_open32f3_lvgl_driver_t объект
      app_open32f3_lvgl_driver_t *lvgl_data = disp_drv->user_data;
    // передать изображение на экран
    lcd_ili93xx_fill_area(lvgl_data->lcd_driver, area->x1, area->y1, area->x2, area->y2, (uint16_t*) color_p);
    // оповестить LVGL, что передача изображения на экран закончена
    lv_disp_flush_ready(disp_drv);
}

/**
 * Колбек для получения текущих координат касания от сенсорного дисплей.
 *
 * LVGL сам вызовет его, когда ему понадобиться считать данные ввода.
 */
static void app_open32f3_lvgl_display_touch_sensor_read_cb(lv_indev_drv_t *disp_drv, lv_indev_data_t *data) {
    // получить app_open32f3_lvgl_driver_t объект
      app_open32f3_lvgl_driver_t *lvgl_data = disp_drv->user_data;

    int touch_x;
    int touch_y;
    int touch_flag;

    // !!! ============================================================
    // Определить, имеется ли касание и записать результат во флаг touch_flag
    // Если касание произошло, то записать его координаты в touch_x и touch_y
```

```
    // ...
    // Код реализации необходимо взять из предыдущего раз-
дела
    // ...
    // !!! ============================================
======================

     // обработка логики устройства ввода в соответствии
с LVGL
    if (touch_flag) {
        lvgl_data->last_x_position = touch_x;
        lvgl_data->last_y_position = touch_y;
    }
    data->point.x = lvgl_data->last_x_position;
    data->point.y = lvgl_data->last_y_position;
    if (touch_flag) {
        // касание дисплея обнаружено
        data->state = LV_INDEV_STATE_PR;
    } else {
        // касание дисплея не обнаружено
        data->state = LV_INDEV_STATE_REL;
    }
}
/**
 * Функция инициализации дисплея для LVGL.
 *
 * @param lcd_driver инициализированный драйвер дисплея из
предыдущего раздела
  * @param touch_driver инициализированный драйвер touch
сенсора из предыдущего раздела
  *  @param  lvgl_data  неинициализированные  объект  app_
open32f3_lvgl_driver_t
 * @return 0 в случае успеха, иначе не нулевое значение
 */
int     app_open32f3_lvgl_init(app_open32f3_lvgl_driver_t
*lvgl_data,   lcd_ili93xx_driver_t   *lcd_driver,   lcd_
xpt2046_driver_t *touch_driver) {
 int err;

    //
    // Шаг 1. Базовая инициализация LVGL
    //
    lv_init();

    // базовая инициализация lvgl_data
```

```
    lvgl_data->lcd_driver = lcd_driver;
    lvgl_data->touch_driver = touch_driver;

    //
    // Шаг 2. Подключение дисплея
    //

    // получить ширину и высоту дисплея
    int16_t width;
    int16_t height;
    err = lcd_ili93xx_get_width(lcd_driver, &width);
    if (err) {
        return err;
    }
    err = lcd_ili93xx_get_height(lcd_driver, &height);
    if (err) {
        return err;
    }

    // Шаг 2.1. Инициализация буфера отрисовки кадра
    lvgl_data->buf_size = width * 20;
     lvgl_data->buf = malloc(sizeof(lv_color_t) * lvgl_
data->buf_size);
        lv_disp_draw_buf_init(&lvgl_data->disp_draw_buf,
lvgl_data->buf, NULL, lvgl_data->buf_size);
    // Шаг 2.2. Регистрация дисплея
    lv_disp_drv_init(&lvgl_data->disp_driver);
    lvgl_data->disp_driver.hor_res = width;
    lvgl_data->disp_driver.ver_res = height;
     lvgl_data->disp_driver.draw_buf = &lvgl_data->disp_
draw_buf;
    lvgl_data->disp_driver.user_data = lvgl_data;
     lvgl_data->disp_driver.flush_cb = app_open32f3_lvgl_
display_flush_cb;
      lvgl_data->disp = lv_disp_drv_register(&lvgl_data-
>disp_driver);
    if (lvgl_data->disp == NULL) {
        return -1;
    }

    //
```

```
    // Шаг 3. Инициализация устройства ввода
    //
    lv_indev_drv_init(&lvgl_data->indev_drv);
    // Будем использовать устройство тип "POINTER"
    lvgl_data->indev_drv.type = LV_INDEV_TYPE_POINTER;
    lvgl_data->last_x_position = 0;
    lvgl_data->last_y_position = 0;
    lvgl_data->indev_drv.user_data = lvgl_data;
    lvgl_data->indev_drv.disp = lvgl_data->disp;
    // !!! ============================================
    // !!! Если вам нужны вспомогательные данные для об-
работки сенсора дисплея,
    // !!! запишите их в поле app_touch_data. Если нет,
пропустите данную строку
    // !!! ============================================
    lvgl_data->app_touch_data = <ваши данные>;
    // Регистрация устройства ввода
      lvgl_data->indev_drv.read_cb = app_open32f3_lvgl_
display_touch_sensor_read_cb;
    lvgl_data->indev = lv_indev_drv_register(&lvgl_data-
>indev_drv);
    if (lvgl_data->indev == NULL) {
        return -1;
    }

    return 0;
}

/**
 * Функция main.
 */
int main(void) {
    ///
      // Код инициализация библиотеки HAL (HAL_Init,
SystemClock_Config) , и периферии (MX_GPIO_Init, MX_USART2_
UART_Init, ...)
    //
    ...

    //
    // Инициализация драйвера дисплея
    //
    lcd_ili93xx_driver_t lcd_driver;
    ... // initialize lcd_driver
```

```
    //
    // Инициализация touch сенсора
    //
    lcd_xpt2046_driver_t touch_driver;
    ... // initialize touch_driver

    //
    // Инициализация LVGL
    //
    app_open32f3_lvgl_driver_t lvgl_driver;
     int err = app_open32f3_lvgl_init(&lvgl_driver, &lcd_
driver, &touch_driver));
    if (err) {
         // вывести сообщение об ошибке и остановить про-
грамму
         ...
    }

    //
    // Создание и инициализация сцены
    //
    ...

    while (1) {

        // Обработки задач LVGL:
        // - опрос устройств ввода
        // - обновление состояния сцены
        // - отрисовка виджетов
        lv_timer_handler();

        // задержка цикла
        HAL_Delay(10);
    }
}
```

Из данного примера скопируйте и добавьте в свой «main.c» следующее:

– **определение структуры** `app_open32f3_lvgl_driver_t`;

– **функцию** `app_open32f3_lvgl_display_flush_cb`;

– **функцию** `app_open32f3_lvgl_display_touch_sensor_read_cb`;

– **функцию** `app_open32f3_lvgl_init`.

Из примера функции `main` добавьте в свой код инициализацию LVGL. Для простоты в примере опущены настройки периферии, поэтому копировать весь код функции `main` целиком нельзя.

Рассмотрим пример более подробно и отметим места, требующие кастомизации:

1. Функция `main`. До основного цикла в ней происходит:

1) настройка периферии (генерируется CubeMX);

2) создание и инициализация драйверов lcd_ili93xx_driver_t и lcd_xpt2046_driver_t (необходимо добавить из предыдущего раздела);

3) вызов функции `app_open32f3_lvgl_init` для инициализации LVGL;

4) создание графического интерфейса (будет показано позже).

В самом вечном цикле вызывается функция LVGL `lv_timer_handler`. В ней происходит:

– опрос устройств ввода;

– обновление состояния сцены;

– отрисовка интерфейса.

`lv_timer_handler` следует вызывать не реже, чем раз в 50 мс. Иначе интерфейс будет неотзывчивым.

2. Функция `app_open32f3_lvgl_init`. В ней происходит:

1) инициализация LVGL (lv_init);

2) регистрация дисплея;

3) регистрация сенсора экрана (Pointer device).

Обратите внимание на строку:

```
lvgl_data->app_touch_data = <ваши данные>;
```

Если вам необходимы вспомогательные переменные для перевода значений от XPT2046 в координаты экрана, добавьте их в структуру `app_open32f3_lvgl_driver_t` и инициализируйте здесь.

3. Функция `app_open32f3_lvgl_display_flush_cb`. Это колбек для передачи части отрисованного интерфейса на дисплей. Оставьте его без изменений.

4. Функция `app_open32f3_lvgl_display_touch_sensor_read_cb`. Это колбек опроса устройства ввода. Изучите его и добавьте свой код из п. 7.4 для получения координат касания дисплея.

На этом настройка LVGL завершена. Подробную информацию о подключении LVGL к дисплеям можно найти в документации: https://docs.lvgl.io/8.2/porting.

Базовый пример интерфейса

Рассмотрим пример с рядом виджетов для реализации таймера, графика и переключателей светодиодов.

```
//
// Подключение библиотек и прочее
//
#include "main.h"
#include "spi.h"
#include "gpio.h"

#include "stdio.h"
#include "stdlib.h"
#include "stdbool.h"

#include "lvgl.h"
#include "lcd_ili93xx_driver.h"
#include "open32f3_lcd_touch_utils.h"

/**
 * Структура с элементами интерфейса.
 *
 * Она содержит указатели на созданные элементы,
 * чтобы к ним можно было легко обращаться,
 * и вспомогательные переменные.
 */
typedef struct {
    // пример стилей
    lv_style_t container_style;

    // блок интерфейса с таймером
    lv_obj_t *timer_container;
    lv_obj_t *timer_reset_button;
    lv_obj_t *timer_reset_button_label;
    lv_obj_t *timer_label;
    uint32_t timer_last_click_time;
    char time_label_text[16];

    // блок интерфейса с графиком
    lv_obj_t *chat_container;
    lv_obj_t *chat_obj;
    lv_chart_series_t *chat_series;
    uint32_t chat_next_update_time;
    uint32_t chat_update_period;
    int16_t chat_last_y;
```

```
    // блок интерфейса управления светодиодами
    lv_obj_t *led_container;
    lv_obj_t *led_switches[8];
} app_scene_t;

static  void  app_scene_timer_button_callback(lv_event_t
*event) {
   app_scene_t *app_scene = lv_event_get_user_data(event);

    // запомнить время клика по кнопке
    app_scene->timer_last_click_time = HAL_GetTick();
}

/**
 * Вспомогательные данные для колбека переключения све-
тодиодов.
 */
typedef struct {
    app_scene_t *app_scene;
    int led_no;
} app_scene_toggle_led_data_t;

static   void   app_scene_toggle_led_common_callback(lv_
event_t *event) {
   app_scene_toggle_led_data_t *data = lv_event_get_user_
data(event);
    lv_obj_t *obj = lv_event_get_target(event);

     //  формирования маски для переключения пина свето-
диода
    int led_no = data->led_no;
    uint32_t led_pin = 0x01 << (led_no + 8);

    // включить/выключить светодиод
    if (lv_obj_has_state(obj, LV_STATE_CHECKED)) {
        HAL_GPIO_WritePin(GPIOE, led_pin, GPIO_PIN_SET);
    } else {
       HAL_GPIO_WritePin(GPIOE, led_pin, GPIO_PIN_RESET);
    }
}

/**
 * Создание и настройка виджетов интерфейса
 *
```

```
 * @param app_scene неинициализированный объект app_
scene_t
 * @param scr объект экрана
 * @return 0 в случае успеха, иначе не нулевое значение
 */
int   app_scene_init(app_scene_t   *app_scene,   lv_obj_t
*screen) {
    // установить цвет экрана с использованием локальных
стилей
   lv_obj_set_style_bg_color(screen, lv_color_make(0xFF,
0x98, 0x00), LV_PART_MAIN | LV_STATE_DEFAULT);

   // создать 3 контейнера, чтобы разделить экран на 3 части
    int16_t width = lv_obj_get_width(screen);
    int16_t height = lv_obj_get_height(screen);
    int16_t container_width = width - 10;
    int16_t container_height = height / 3 - 8;
    // подготовить общий стиль для контейнеров
    lv_style_init(&app_scene->container_style);
      lv_style_set_bg_color(&app_scene->container_style,
lv_color_make(0xB3, 0xB3, 0xFF));
    lv_style_set_radius(&app_scene->container_style, 5);
    lv_style_set_pad_all(&app_scene->container_style, 2);

    // контейнер таймера
    app_scene->timer_container = lv_obj_create(screen);
     lv_obj_add_style(app_scene->timer_container,  &app_
scene->container_style, LV_PART_MAIN | LV_STATE_DEFAULT);
   lv_obj_set_size(app_scene->timer_container, container_
width, container_height);
    lv_obj_set_pos(app_scene->timer_container, 5, 5); //
указать позицию контейнера явно
    // контейнер графика
    app_scene->chat_container = lv_obj_create(screen);
      lv_obj_add_style(app_scene->chat_container,  &app_
scene->container_style, LV_PART_MAIN | LV_STATE_DEFAULT);
    lv_obj_set_size(app_scene->chat_container, container_
width, container_height);
      lv_obj_align(app_scene->chat_container,  LV_ALIGN_
CENTER, 0, 0); // указать позицию через выравнивание
    // контейнер для переключателей светодиодов
    app_scene->led_container = lv_obj_create(screen);
   lv_obj_add_style(app_scene->led_container, &app_scene-
>container_style, LV_PART_MAIN | LV_STATE_DEFAULT);
```

```
    lv_obj_set_size(app_scene->led_container, container_
width, container_height);
       lv_obj_align(app_scene->led_container, LV_ALIGN_
BOTTOM_MID, 0, -5); // указать позицию через выравнивание

    // настройка таймера
    // создать кнопку сброса таймера с надписью "Reset"
      app_scene->timer_reset_button = lv_btn_create(app_
scene->timer_container);
       app_scene->timer_reset_button_label = lv_label_
create(app_scene->timer_reset_button);
        lv_label_set_text(app_scene->timer_reset_button_
label, "Reset");
      lv_obj_set_size(app_scene->timer_reset_button, LV_
SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_align(app_scene->timer_reset_button, LV_ALIGN_
LEFT_MID, 20, 0);
     app_scene->timer_label = lv_label_create(app_scene-
>timer_container);
    lv_obj_align(app_scene->timer_label, LV_ALIGN_RIGHT_
MID, -20, 0);
    app_scene->timer_last_click_time = HAL_GetTick();
    // повесить на кнопку колбек (вызывается при нажатии
на кнопку)
       lv_obj_add_event_cb(app_scene->timer_reset_button,
app_scene_timer_button_callback, LV_EVENT_CLICKED, app_
scene);

    // build chat
   app_scene->chat_obj = lv_chart_create(app_scene->chat_
container);
    lv_obj_set_size(app_scene->chat_obj, container_width
- 10, container_height - 10);
    lv_obj_align(app_scene->chat_obj, LV_ALIGN_CENTER, 0,
0);
       lv_chart_set_range(app_scene->chat_obj, LV_CHART_
AXIS_PRIMARY_Y, -1, 5); // диапазон y
    const int num_points = 20;
      lv_chart_set_point_count(app_scene->chat_obj, num_
points); // количество точке по X
    // создать линию/серию на графике
      app_scene->chat_series = lv_chart_add_series(app_
scene->chat_obj, lv_color_make(0x43, 0xA0, 0x47), LV_
CHART_AXIS_PRIMARY_Y);
    // заполнить ее нулями
```

```
    app_scene->chat_last_y = 0;
    for (int i = 0; i < num_points; i++) {
        lv_chart_set_next_value(app_scene->chat_obj, app_
scene->chat_series, app_scene->chat_last_y);
    }
    app_scene->chat_update_period = 500;
     app_scene->chat_next_update_time = HAL_GetTick() +
app_scene->chat_update_period;

    // создать 8 переключателей светодиодов
    for (int i = 0; i < 8; i++) {
       app_scene->led_switches[i] = lv_switch_create(app_
scene->led_container);
        // задаем размер переключателей явно
         lv_obj_set_size(app_scene->led_switches[i], 40,
23);
        int16_t y_offset = -20 + 40 * (i / 4);
        int16_t x_offset = -75 + 50 * (i % 4);
       lv_obj_align(app_scene->led_switches[i], LV_ALIGN_
CENTER, x_offset, y_offset);
    app_scene_toggle_led_data_t *data = malloc(sizeof(app_
scene_toggle_led_data_t));
        data->app_scene = app_scene;
        data->led_no = i;
         // повесить на переключатель колбек (вызывается
при смене состояния переключателя)
          lv_obj_add_event_cb(app_scene->led_switches[i],
app_scene_toggle_led_common_callback,    LV_EVENT_VALUE_
CHANGED, data);
    }

    return 0;
}

/**
 * Функция обновления сцены (обновление таймера, графиков
и т.д.)
 *
 * @param app_scene объект app_scene_t
 * @param scr объект экрана
 * @return 0 в случае успеха, иначе не нулевое значение
 */
int app_scene_update(app_scene_t *app_scene) {
    // Обновить текстовый блок с таймером
```

```
    int timer_value = (HAL_GetTick() - app_scene->timer_
last_click_time) / 1000;
      lv_label_set_text_fmt(app_scene->timer_label, "%8i
s", timer_value);

    // обновить график
    if (HAL_GetTick() > app_scene->chat_next_update_time)
{
          app_scene->chat_next_update_time += app_scene-
>chat_update_period;
        app_scene->chat_last_y += 1;
        if (app_scene->chat_last_y > 5) {
            app_scene->chat_last_y = 0;
        }
         // добавить новую точку на график (самая старая
точка будет удалена)
       lv_chart_set_next_value(app_scene->chat_obj, app_
scene->chat_series, app_scene->chat_last_y);
    }

    return 0;
}


/**
 * Функция main.
 */
int main(void) {
    ///
       // Код инициализация библиотеки HAL (HAL_Init,
SystemClock_Config) , и периферии (MX_GPIO_Init, MX_USART2_
UART_Init, ...)
    //
    ...

    // вспомогательные переменные
    int err;

    //
    // Инициализация драйвера дисплея
    //
    lcd_ili93xx_driver_t lcd_driver;
    ... // initialize lcd_driver

    //
```

```
    // Инициализация touch сенсора
    //
    lcd_xpt2046_driver_t touch_driver;
    ... // initialize touch_driver

    //
    // Инициализация LVGL
    //
    app_open32f3_lvgl_driver_t lvgl_driver;
       err = app_open32f3_lvgl_init(&lvgl_driver, &lcd_
driver, &touch_driver);
    if (err) {
        // вывести сообщение об ошибке и остановить про-
грамму
        ...
    }

    //
    // Создание и инициализация сцены
    //
    app_scene_t app_scene;
     lv_obj_t *screen = lv_disp_get_scr_act(lvgl_driver.
disp);
    err = app_scene_init(&app_scene, screen);
    if (err) {
        // вывести сообщение об ошибке и остановить про-
грамму
        ...
    }

    while (1) {
        // обновление элементов сцены (таймер и график)
        app_scene_update(&app_scene);

        // Обработки задач LVGL:
        // - опрос устройств ввода
        // - обновление состояния сцены
        // - отрисовка виджетов
        lv_timer_handler();

        // задержка цикла
        HAL_Delay(10);
    }
}
```

Рис. 8.6. Пример интерфейса приложения на LVGL

Из данного пример скопируйте и добавьте в свой «main.c» следующее:
– определение структуры `app_scene_t`;
– функцию `app_scene_timer_button_callback`;
– функцию `app_scene_toggle_led_common_callback`;
– функцию `app_scene_init`;
– функцию `app_scene_update`.

Из примера функции `main` добавьте в свой код инициализацию переменной app_scene и обновления элементов сцены интерфейса.

Если все добавлено и настроено правильно, то после компиляции и загрузки программы на дисплее макета вы увидите интерфейс, показанный на рис. 8.6.

В верхней части расположен таймер, сбрасывающийся при нажатии кнопки, в середине – пример графика, внизу – переключатели для управления светодиодами.

Оптимизация памяти

Библиотеки для создания графических интерфейсов занимают относительно много места. Без включенных оптимизаций компилятора пример проекта п. 8.2 занимает почти всю память микроконтроллера (рис. 8.7).

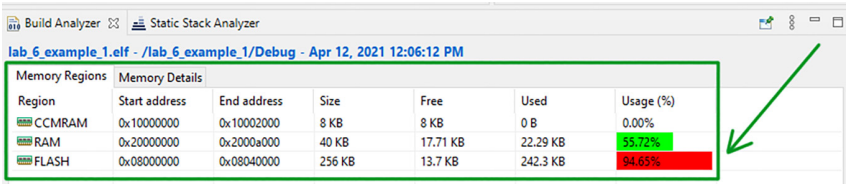


Рис. 8.7. Расход памяти проекта без оптимизаций

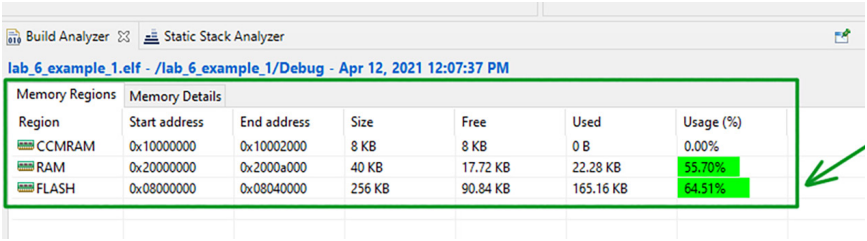


Рис. 8.8. Итоговый размер прошивки

Из-за этого, при добавлении новых виджетов в проект, вы можете превысить лимит памяти и получить ошибку компиляции. В этом случае следует повысить уровень оптимизации компилятора с «-O0» (все оптимизации выключены), который используется по умолчанию, на «-Og», включающий базовые оптимизации, но сохраняющей возможности отладки. Для этого необходимо:

1. Отрыть свойства проекта.

2. Перейти на вкладку «C/C++» → «Settings» → «Tool Settings» и выбрать пункт «MCU GCC Compiler» → «Optimization». По умолчанию все оптимизации компилятора отключены («-O0»).

3. Выберите «Optimize for Debug (-Og)» и примените настройки.

4. После повторной компиляции проекта, итоговый размер прошивки значительно увеличится (рис. 8.8).

API библиотеки LVGL

Виджеты. В LVGL интерфейс состоит из виджетов (кнопки, текстовые поля, переключатели), имеющих следующие общие параметры:

– позиция на экране;

– размер;

– родительский элемент;

– дочерние элементы.

Для создания виджетов имеются функции:

```
lv_obj_t * lv_ <тип_виджета>_create(lv_obj_t * parent);
```

где *parent* – родительский виджет. В качестве него можно использовать виджет экрана, получаемого с помощью функции `lv_scr_act`.

Возвращаемое значение всегда указатель типа `lv_obj_t *`. Например:

```
// создать кнопку на экране
lv_obj_t * btn1 = lv_btn_create(lv_scr_act());
// создать надпись на кнопке
lv_obj_t * label = lv_label_create(btn1);
```

Удаляются виджеты при помощи единой функции:

```
void lv_obj_del(lv_obj_t * obj);
```

Она удаляет объект и все его дочерние элементы. Например:

```
lv_obj_del(btn1);
```

Размер виджетов задается функцией lv_obj_set_size, а координаты с помощью lv_obj_set_pos. Например:

```
// создать объект на экране
lv_obj_t * par = lv_obj_create(lv_scr_act());
lv_obj_set_size(par, 100, 80);
lv_obj_set_pos(par, 50, 50);

// создать на предыдущем объекте новый
lv_obj_t * obj1 = lv_obj_create(par);
lv_obj_set_pos(obj1, 10, 10);
```

Отметим особенности позиционирования:

– начало координат (точка 0,0) находится в верхнем левом углу;

– координаты объекта всегда указываются относительно родителя;

– дочерний объект по умолчанию виден только в пределах родительского. Все, что выходит за родителя, будет обрезано.

Все специфичные свойства виджетов задаются с помощью функций:

```
void lv_<тип_виджета>_set_<название_свойвтва>(lv_obj_t *obj, <тип_свойства> <значение>);
```

Например, установка текста на label:

```
lv_label_set_text(label1, "Reset");
```

Подробную информацию о виджетах и их пример можно найти в документации: https://docs.lvgl.io/8.2/widgets.

Стили. LVGL позволят задать внешний вид объектов (цвет фона, границы, положение, отступы и т. д.) с помощью CSS подобных стилей. В простейшем случае можно использовать локальные стили, принадлежащие конкретному объекту. Для этого существует ряд функций:

```
void lv_obj_set_style_<property_name>(obj, <value>, <selector>);
```

где:

- `property_name` – имя свойства;
- `obj` – виджет;
- `value` – значение свойства;
- `selector` – CSS-подобный селектор. В большинстве случаев достаточно использовать значение `LV_PART_MAIN | LV_STATE_DEFAULT`.

Например, задание фона виджета экрана выглядит следующим образом:

```
lv_obj_set_style_bg_color(screen, lv_color_make(0xFF, 0x98, 0x00), LV_PART_MAIN | LV_STATE_DEFAULT);
```

Подробную информацию о стилях можно найти в документации: https://docs.lvgl.io/8.2/overview/style.html.

События. С каждым виджетом связан ряд событий, например, клик пользователя по элементу. Для их обработки необходимо реализовать функцию, принимающую объект события, и назначить ее обработчиком событий виджета с помощью функции `lv_obj_add_event_cb`. Например:

```
static void my_button_event_cb(lv_event_t *event)
{
      // Получить пользовательские данные, привязанные к кнопке.
    // Они могут понадобится, если нужно передать какие-либо
     // данные в обработчик
    scene_controller_t *app = lv_event_get_user_data(obj);

     // переключить нужный светодиод при нажатии на кнопку
     HAL_GPIO_TogglePin(GPIOE, app->led_state << 8);
}

...
// создать и инициализировать «класс» контроллер приложения
scene_controller_t app_controller;
...

// создать кнопку на экране
lv_obj_t * btn = lv_btn_create(lv_scr_act());
// задать обработчик событий кнопки
lv_obj_add_event_cb(app_scene->timer_reset_button,    app_scene_timer_button_callback, LV_EVENT_CLICKED, app_scene);
```

Функция добавления события имеет следующий прототип:

```
struct lv_event_dsc_t* lv_obj_add_event_cb(struct lv_
obj_t * obj, lv_event_cb_t event_cb, lv_event_code_t
filter, void * user_data);
```

где:

- `obj` – виджет к которому нужно добавить обработчик;
- `event_cb` – функция-обработчик;
- `filter` – код события (например, `LV_EVENT_CLICKED`);
- `user_data` – кастомные данные, которые можно получить внутри обработчика с помощью функции `lv_event_get_user_data`.

Подробную информацию о колбеках виджетов (все коды событий и т. д.) можно найти в документации: https://docs.lvgl.io/8.2/overview/event.html.

8.3. Практические задания к разделу 8

Реализовать встраиваемое приложение с графическим интерфейсом пользователя в соответствии с вариантом.

Варианты для индивидуальной работы

Вариант 1

Реализовать простой «плеер» с 4 предустановленными мелодиями. Плеер должен иметь копку для запуска и остановки проигрывания музыки. Выбор мелодии необходимо осуществлять с помощью 4 элементов типа «lv_switch» (switch), из которых только один всегда активен. Мелодии взять из Приложения 1.

Вариант 2

Реализовать приложение отображения графика состояния кнопки (PA0) в реальном времени. Приложение должно отображать 20 последних состояний кнопок с интервалом 500 мс. График должен иметь следующие опции, реализованные с помощью «lv_switch» (switch):

– переключение направления оси *X* (перевернуть точки графика сверху-вниз);

– переключения направления оси *Y* (вместо добавления новых точек справа и движения графика справа-налево, добавлять новые точки слева и двигать график слева-направо).

Вариант 3

Реализовать таймер, показывающий время со старта микроконтроллера в миллисекундах, и кнопку, фиксирующую время клика. Время последних 5 кликов с их номерами отображать в отдельных текстовых полях (lv_label). Расположение элементов интерфейса продумать самостоятельно.

Вариант 4

Реализовать простой «плеер» с 4 предустановленными мелодиями. Плеер должен иметь кнопку для запуска и остановки проигрывания музыки. Выбор мелодии необходимо осуществлять с помощью 4 элементов типа «lv_cb» (checkbox), из которых только один всегда активен. Мелодии взять из из Приложения 1.

Вариант 5

Реализовать приложение отображения графика состояния кнопки в реальном времени. Приложение должно отображать 20 последних состояний кнопок с интервалом 500 мс. График должен иметь следующие опции, реализованные с помощью «lv_cb» (checkbox):

– переключение направления оси *X* (перевернуть точки графика сверху-вниз);

– переключения направления оси *Y* (вместо добавления новых точек справа и движения графика справа-налево, добавлять новые точки слева и двигать график слева-направо).

Вариант 6

Реализовать приложение управления анимацией светодиодов. С помощью «lv_switch» (switch) должна быть возможность указывать один из 4 режимов анимации. Первые 3 режима взять из практического задания раздела 3.4; 4-м режимом сделать индивидуальное управление светодиодами с помощью 8 отдельных «lv_cb» (checkbox) элементов.

Вариант 7

Реализовать простой «плеер» с 4 предустановленными мелодиями. Плеер должен иметь кнопку для запуска и остановки проигрывания музыки. Выбор мелодии необходимо осуществлять с помощью элемента «lv_roller» (roller). Мелодии взять из материалов к главе с таймерами.

Вариант 8

Реализовать таймер, показывающий время со старта микроконтроллера в миллисекундах, и кнопку сброса таймера. При сбросе таймера необходимо запомнить его время и отобразить в отдельном текстовом поле (lv_label). Необходимо иметь 5 подобных полей для отображения последних 5 времен таймера перед его сбросом. Расположение элементов интерфейса продумать самостоятельно.

Вариант 9

Реализовать приложение управления анимацией светодиодов. С помощью «lv_cb» (checkbox) должна быть возможность указывать один из 4 режимов анимации. Первые 3 режима взять из практического задания раздела 3.4; 4-м режимом сделать индивидуальное управление светодиодами с помощью 8 отдельных «lv_switch» (switch) элементов.

Вариант 10

Реализовать простой «плеер» с 4 предустановленными мелодиями. Плеер должен иметь кнопку для запуска и остановки проигрывания музыки. Выбор мелодии необходимо осуществлять с помощью элемента «lv_dropdown» (Drop-down list). Мелодии взять из материалов к главе с таймерами.

ЗАКЛЮЧЕНИЕ

Технологии программирования микроконтроллеров имеют ряд особенностей, связанных, прежде всего, с их сложной регистровой структурой и необходимостью учета особенностей аппаратных средств в процессе проектирования системы. Это обуславливает необходимость создания комплексных программных систем, обеспечивающих как разработку программ, так и их отладку в реальном времени. Рассмотренная в настоящем пособии экосистема фимы STM является интегрированной программной средой, ориентированной на поддержку всех основных этапов проектирования микроконтроллерной системы. Рассмотрены типовые задачи организации ввода-вывода и отображения данных. Приведенные в приложениях примеры программирования направлены на сокращение времени освоения технологий и формирование навыков разработки простейших программ, которые могут быть в дальнейшем использованы студентом при создании более сложных программных проектов.

СПИСОК ИСТОЧНИКОВ

1. Arm GNU Toolchain. URL: https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain (дата обращения: 15.08.2022).

2. Mbed OS. Minimal printf and snprintf. URL: https://github.com/ARMmbed/mbed-os/tree/master/platform/source/minimal-printf (дата обращения: 15.08.2022).

3. IAR Embedded Workbench for Arm. URL: https://www.iar.com/products/architectures/arm/iar-embedded-workbench-for-arm/ (дата обращения: 15.08.2022).

4. Speed & code size STM32CubeIDE vs IAR // ST Community. 2021. URL: https://community.st.com/s/question/0D53W00000Vj3tZSAR/speed-code-size-stm32cubeide-vs-iar (дата обращения: 15.08.2022).

5. Сравнение компиляторов ARMCC, IAR и GCC // Хабр. 2020. URL: https://habr.com/ru/post/527820/ (дата обращения: 15.08.2022).

6. ARM Compiler Version 5. URL: https://www2.keil.com/mdk5/compiler/5/ (дата обращения: 15.08.2022).

7. LLVM // The Architecture of Open Source Applications. Elegance, Evolution, and a Few Fearless Hacks. URL: https://www.aosabook.org/en/llvm.html (дата обращения: 15.08.2022).

8. Better Firmware with LLVM/Clang // The Interrupt community. 2020. URL: https://interrupt.memfault.com/blog/arm-cortexm-with-llvm-clang (дата обращения: 15.08.2022).

9. Arm Compiler Version 6. URL: https://www2.keil.com/mdk5/compiler/6 (дата обращения: 15.08.2022).

10. Mbed Studio. URL: https://os.mbed.com/studio/ (дата обращения: 15.08.2022).

11. Arduino IDE. URL: https://www.arduino.cc/en/software (дата обращения: 15.08.2022).

12. Arduino CLI. URL: https://github.com/arduino/arduino-cli (дата обращения: 15.08.2022).

13. Arduino library specification. URL: https://arduino.github.io/arduino-cli/0.26/library-specification/ (дата обращения: 15.08.2022).

14. PlatformIO documentation. URL: https://docs.platformio.org/en/latest/what-is-platformio.html (дата обращения: 15.08.2022).

15. STM32Cube initialization code generator. URL: https://www.st.com/en/development-tools/stm32cubemx.html (дата обращения: 15.08.2022).

16. MDK Microcontroller Development Kit. URL: https://www2.keil.com/mdk5/ (дата обращения: 15.08.2022).

17. CLion. URL: https://www.jetbrains.com/clion/ (дата обращения: 15.08.2022).

18. Visual Studio Code. URL: https://code.visualstudio.com/ (дата обращения: 15.08.2022).

19. STM32F3Discovery. Discovery kit with STM32F303VC MCU. URL: https://www.st.com/en/evaluation-tools/stm32f3discovery.html (дата обращения: 10.09.2022).

20. STM32F303xB STM32F303xC. Datasheet – production data // STMicroelectronics. 2018. 149 p.

21. RM0316 Reference manual // STMicroelectronics. 2017. 1141 p.

22. AN4899 Application note. STM32 microcontroller GPIO hardware settings and low-power consumption // STMicroelectronics. 2022. 31 p.

23. STM32. Getting started with GPIO. URL: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_GPIO (дата обращения: 10.09.2022).

24. UM1786. User manual. Description of STM32F3 HAL and low-layer drivers // STMicroelectronics. 2020. 1354 p.

25. Toulson R. Fast and Effective Embedded. Applying the ARM mbed. Second Edition / Rob Toulson, Tim Wilmshurst. Elsevier. 2017. 512 p.

26. Arm Cortex-M4 Processor. Technical Reference Manual // ARM. 2020. 107 p.

27. Pena .E, Grace M. UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter // AnalogDialogue. 2020. Vol. 54

28. AT Commands Examples. Examples for u-blox cellular modules. Application Note // u-blox. 2019. 153 p.

29. AN4776. Application note. General-purpose timer cookbook for STM32 microcontrollers // STMicroelectronics. 2019. 72 p.

30. AN4013. Application note. STM32 cross-series timer overview // STMicroelectronics. 2021. 45 p.

31. Cortex-M4 Devices. Generic User Guide // ARM. 2011. 277 p.

32. What is Pulse Width Modulation (PWM)? Applications and Accessories // Seeed Studio Blog. 2020. URL: https://www.seeedstudio.com/blog/2020/06/16/basic-electronics-pulse-width-modulationpwm-and-arduino-applications/ (дата обращения: 30.09.2022).

33. Ibrahim D. / Designing Embedded Systems with 32-Bit PIC Microcontrollers and MikroC / Dogan Ibrahim. Elsevier. 2014. 471 p.

34. Sinclair I. / Electronics Simplified. Third edition / Ian Sinclair. Elsevier. 2011. 362 p.

35. Introduction to graphics and LCD Introduction to graphics and LCD // NXP – 2009. 33 p.

36. ILI9325. a-Si TFT LCD Single Chip Driver 240RGBx320 Resolution and 262K color. Datasheet // ILITEK. 2008. 107 p.

37. Introduction to Bit Banging. Build an Interrupt-driven Software Serial for STM32F3 MCU // Medium. 2020. URL: https://medium.com/@kslooi/introduction-to-bit-banging-46e114db3466 (дата обращения: 30.09.2022).

38. AN4655. Application note. Virtually increasing the number of serial communication peripherals in STM32 applications // STMicroelectronics. 2015. 21 p.

39. Chen J. / Visual Display Technology. Second Edition / Janglin Chen, Wayne Cranton, Mark Fihn. Springer. 2016. 3454 p.

40. XPT2046 Data Sheet // XPTEK. 2007. 30 p.

ПРИЛОЖЕНИЯ

Приложение 1

Файлы тоновых мелодий

Файл melody_1.h

```
#ifndef MELODY_1_H
#define MELODY_1_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_1_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_1_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_1_DURATIONS[];
#endif
```

Файл melody_1.c

```
#include "melody_1.h"
// number of notes
const size_t MELODY_1_LEN = 78;
// note frequencies (Hz)
const uint16_t MELODY_1_FREQUENCIES[] = {2637,0,2637,0,0,
0,2637,0,0,0,2093,0,2637,0,0,0,3136,0,0,0,0,0,0,0,1568,0
,0,0,0,0,0,0,2093,0,0,0,0,0,1568,0,0,0,0,0,1319,0,0,0,0,
0,1760,0,0,0,1976,0,0,0,1865,0,1760,0,0,0,1568,0,2637,0,
3136,0,3520,0,0,0,2794,0,3136,0,0,0,2637,0,0,0,2093,0,23
49,0,1976,0,0,0,0,0,2093,0,0,0,0,0,1568,0,0,0,0,0,1319,0
,0,0,0,0,1760,0,0,0,1976,0,0,0,1865,0,1760,0,0,0,1568,0,
2637,0,3136,0,3520,0,0,0,2794,0,3136,0,0,0,2637,0,0,0,20
93,0,2349,0,1976,0,0,0,0,0};
// note durations (ms)
const uint16_t MELODY_1_DURATIONS[] = {83,24,83,24,83,
24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83
,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,8
3,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,8
3,24,83,24,83,24,111,33,111,33,111,33,83,24,83,24,83,2
4,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,2
4,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,24,83,2
4,83,24,83,24,83,24,83,24,83,24,83,24,83,24,111,33,111
,33,111,33,83,24,83,24,83,24,83,24,83,24,83,24,83,24,-
83,24,83,24,83,24,83,24,83,24};
```

Файл melody_2.h

```
#ifndef MELODY_2_H
#define MELODY_2_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_2_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_2_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_2_DURATIONS[];
#endif
```

Файл melody_2.c

```
#include "melody_2.h"
// number of notes
const size_t MELODY_2_LEN = 56;
// note frequencies (Hz)
const uint16_t MELODY_2_FREQUENCIES[] = {262,0,523,0,220
,0,440,0,233,0,466,0,0,0,0,0,262,0,523,0,220,0,440,0,233
,0,466,0,0,0,0,0,175,0,349,0,147,0,294,0,156,0,311,0,0,0
,0,0,175,0,349,0,147,0,294,0,156,0,311,0,0,0,0,0,311,0,2
77,0,294,0,277,0,311,0,311,0,208,0,196,0,277,0,262,0,370
,0,349,0,165,0,466,0,440,0,415,0,311,0,247,0,233,0,220,-
0,208,0,0,0,0,0,0,0};
// note durations (ms)
const uint16_t MELODY_2_DURATIONS[] = {83,24,83,24,83,24,
83,24,83,24,83,24,166,49,333,99,83,24,83,24,83,24,83,24,8
3,24,83,24,166,49,333,99,83,24,83,24,83,24,83,24,83,24,83
,24,166,49,333,99,83,24,83,24,83,24,83,24,83,24,83,24,166
,49,166,49,55,16,55,16,55,16,166,49,166,49,166,49,166,49,
166,49,166,49,55,16,55,16,55,16,55,16,55,16,55,16,100,30,
100,30,100,30,100,30,100,30,333,99,333,99,333,99};
```

Файл melody_3.h

```
#ifndef MELODY_3_H
#define MELODY_3_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_3_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_3_FREQUENCIES[];
```

```
// note durations (ms)
extern const uint16_t MELODY_3_DURATIONS[];
#endif
```

Файл melody_3.c

```
#include "melody_3.h"
// number of notes
const size_t MELODY_3_LEN = 32;
// note frequencies (Hz)
const uint16_t MELODY_3_FREQUENCIES[] = {65,0,175,0,131
,0,110,0,131,0,175,0,131,0,131,0,175,0,131,0,175,0,233,0
,196,0,175,0,165,0,147,0,139,0,131,0,175,0,131,0,110,0,1
31,0,175,0,131,0,233,0,0,0,196,0,175,0,165,0,147,0,139,0
,131,0};
// note durations (ms)
const uint16_t MELODY_3_DURATIONS[] = {250,75,250,75,250,
75,250,75,250,75,250,75,500,150,250,75,250,75,250,75,250
,75,333,99,125,37,125,37,125,37,125,37,125,37,250,75,250
,75,250,75,250,75,250,75,500,150,250,75,125,37,25
0,75,250,75,250,75,250,75,250,75,250,75};
```

Файл melody_4.h

```
#ifndef MELODY_4_H
#define MELODY_4_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_4_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_4_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_4_DURATIONS[];
#endif
```

Файл melody_4.c

```
#include "melody_4.h"
// number of notes
const size_t MELODY_4_LEN = 384;
// note frequencies (Hz)
const uint16_t MELODY_4_FREQUENCIES[] = {659,0,165,0,494,
0,523,0,587,0,659,0,587,0,523,0,494,0,440,0,220,0,440,0,
523,0,659,0,220,0,587,0,523,0,494,0,330,0,392,0,523,0,58
7,0,165,0,659,0,165,0,523,0,220,0,440,0,220,0,440,0,220,
0,123,0,131,0,147,0,587,0,698,0,880,0,523,0,523,0,784,0,
```

```
698,0,659,0,131,0,0,0,523,0,659,0,440,0,392,0,587,0,523,
0,494,0,330,0,494,0,523,0,587,0,392,0,659,0,392,0,523,0,
330,0,440,0,165,0,440,0,0,0,659,0,165,0,494,0,523,0,587,
0,659,0,587,0,523,0,494,0,440,0,220,0,440,0,523,0,659,0,
220,0,587,0,523,0,494,0,330,0,392,0,523,0,587,0,165,0,65
9,0,165,0,523,0,220,0,440,0,220,0,440,0,220,0,123,0,131,
0,147,0,587,0,698,0,880,0,523,0,523,0,784,0,698,0,659,0,
131,0,0,0,523,0,659,0,440,0,392,0,587,0,523,0,494,0,330
,0,494,0,523,0,587,0,392,0,659,0,392,0,523,0,330,0,440,
0,165,0,440,0,0,0,330,0,165,0,110,0,165,0,262,0,165,0,11
0,0,165,0,294,0,165,0,104,0,165,0,247,0,165,0,104,0,165,
0,262,0,165,0,110,0,165,0,220,0,165,0,110,0,165,0,208,0,
165,0,104,0,165,0,247,0,165,0,104,0,165,0,330,0,165,0,11
0,0,165,0,262,0,165,0,110,0,165,0,294,0,165,0,104,0,165,
0,247,0,165,0,104,0,165,0,262,0,165,0,330,0,165,0,440,0,
165,0,110,0,165,0,415,0,165,0,104,0,165,0,104,0,165,0,10
4,0,165,0,659,0,165,0,494,0,523,0,587,0,659,0,587,0,523,
0,494,0,440,0,220,0,440,0,523,0,659,0,220,0,587,0,523,0,
494,0,330,0,392,0,523,0,587,0,165,0,659,0,165,0,523,0,22
0,0,440,0,220,0,440,0,220,0,123,0,131,0,147,0,587,0,698,
0,880,0,523,0,523,0,784,0,698,0,659,0,131,0,0,0,523,0,65
9,0,440,0,392,0,587,0,523,0,494,0,330,0,494,0,523,0,587,
0,392,0,659,0,392,0,523,0,330,0,440,0,165,0,440,0,0,0,65
9,0,165,0,494,0,523,0,587,0,659,0,587,0,523,0,494,0,440,
0,220,0,440,0,523,0,659,0,220,0,587,0,523,0,494,0,330,0,
392,0,523,0,587,0,165,0,659,0,165,0,523,0,220,0,440,0,22
0,0,440,0,220,0,123,0,131,0,147,0,587,0,698,0,880,0,523,
0,523,0,784,0,698,0,659,0,131,0,0,0,523,0,659,0,440,0,39
2,0,587,0,523,0,494,0,330,0,494,0,523,0,587,0,392,0,659,
0,392,0,523,0,330,0,440,0,165,0,440,0,0,0,330,0,165,0,11
0,0,165,0,262,0,165,0,110,0,165,0,294,0,165,0,104,0,165,
0,247,0,165,0,104,0,165,0,262,0,165,0,110,0,165,0,220,0,
165,0,110,0,165,0,208,0,165,0,104,0,165,0,247,0,165,0,10
4,0,165,0,330,0,165,0,110,0,165,0,262,0,165,0,110,0,165,
0,294,0,165,0,104,0,165,0,247,0,165,0,104,0,165,0,262,0,
165,0,330,0,165,0,440,0,165,0,110,0,165,0,415,0,165,0,10
4,0,165,0,104,0,165,0,104,0,165,0};
// note durations (ms)
const uint16_t MELODY_4_DURATIONS[] = {125,37,125,37,125,
37,125,37,125,37,62,18,62,18,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,250,75
```

,125,37,125,37,62,18,62,18,125,37,125,37,125,37,125,37,1
25,37,125,37,125,37,62,18,62,18,125,37,125,37,125,37,125
,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125
,37,125,37,125,37,250,75,250,75,125,37,125,37,125,37,125
,37,125,37,62,18,62,18,125,37,125,37,125,37,125,37,125,3
7,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,3
7,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,3
7,125,37,125,37,125,37,125,37,125,37,125,37,250,75,125,
37,125,37,62,18,62,18,125,37,125,37,125,37,125,37,125,3
7,125,37,125,37,62,18,62,18,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,3
7,125,37,125,37,250,75,250,75,125,37,125,37,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125
,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,12
5,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,1
25,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,1
25,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,1
25,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,1
25,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,1
25,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,1
25,37,125,37,62,18,62,18,125,37,125,37,125,37,125,37,125
,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125
,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125
,37,125,37,125,37,125,37,125,37,125,37,125,37,250,75,125
,37,125,37,62,18,62,18,125,37,125,37,125,37,125,37,125,3
7,125,37,125,37,62,18,62,18,125,37,125,37,125,37,125,37,
125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,
125,37,125,37,125,37,250,75,250,75,125,37,125,37,125,37,
125,37,62,18,62,18,125,37,125,37,125,37,125,37,125,37,12
5,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,12
5,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,12
5,37,125,37,125,37,125,37,125,37,125,37,250,75,125,37,12
5,37,62,18,62,18,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,62,18,62,18,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37
,125,37,250,75,250,75,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,37
,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,3

```
7,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,125,37,125,37};
```

Файл melody_5.h

```
#ifndef MELODY_5_H
#define MELODY_5_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_5_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_5_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_5_DURATIONS[];
#endif
```

Файл melody_5.c

```
#include "melody_5.h"
// number of notes
const size_t MELODY_5_LEN = 83;
// note frequencies (Hz)
const uint16_t MELODY_5_FREQUENCIES[] = {622,0,659,0,740,
0,0,0,988,0,659,0,622,0,659,0,740,0,988,0,1245,0,1319,0,
1245,0,932,0,988,0,0,0,740,0,0,0,622,0,659,0,740,0,0,0,9
88,0,1109,0,932,0,988,0,1109,0,1319,0,1245,0,1319,0,1109
,0,370,0,415,0,294,0,311,0,93,0,277,0,294,0,277,0,247,0,
247,0,277,0,294,0,294,0,277,0,247,0,277,0,311,0,370,0,41
5,0,311,0,370,0,277,0,311,0,247,0,277,0,247,0,311,0,370,
0,415,0,311,0,370,0,277,0,311,0,247,0,294,0,311,0,294,0,
277,0,247,0,277,0,294,0,247,0,277,0,311,0,370,0,277,0,29
4,0,277,0,247,0,277,0,247,0,277,0};
// note durations (ms)
const uint16_t MELODY_5_DURATIONS[] = {62,18,62,18,62,18,
62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62
,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,1
8,125,37,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18
,125,37,125,37,62,18,62,18,62,18,62,18,62,18,62,18,125,3
7,125,37,125,37,125,37,62,18,62,18,62,18,62,18,62,18,62,
18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,125,37,125,3
7,125,37,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18
,62,18,62,18,62,18,62,18,125,37,62,18,62,18,62,18,62,18,
62,18,62,18,62,18,62,18,125,37,125,37,125,37};
```

Файл melody_6.h

```
#include "melody_6.h"
// number of notes
const size_t MELODY_6_LEN = 36;
// note frequencies (Hz)
const uint16_t MELODY_6_FREQUENCIES[] = {440,0,0,0,440,0,
0,0,440,0,0,0,349,0,0,0,523,0,0,0,440,0,0,0,349,0,0,0,52
3,0,0,0,440,0,0,0,659,0,0,0,659,0,0,0,659,0,0,0,698,0,0,
0,523,0,0,0,784,0,0,0,698,0,0,0,523,0,0,0,440,0,0,0};
// note durations (ms)
const uint16_t MELODY_6_DURATIONS[] = {500,150,200,60,500
,150,200,60,500,150,200,60,500,150,50,15,200,60,50,15,10
00,300,100,30,500,150,50,15,200,60,50,15,1000,300,1000,3
00,500,150,200,60,500,150,200,60,500,150,200,60,500,150,
50,15,200,60,50,15,1000,300,100,30,500,150,50,15,200,60,
50,15,1000,300,500,150};
```

Файл melody_6.c

```
#ifndef MELODY_6_H
#define MELODY_6_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_6_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_6_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_6_DURATIONS[];
#endif
```

Файл melody_7.h

```
#ifndef MELODY_7_H
#define MELODY_7_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_7_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_7_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_7_DURATIONS[];
#endif
```

Файл melody_7.c

```
#include "melody_7.h"
// number of notes
const size_t MELODY_7_LEN = 85;
// note frequencies (Hz)
const uint16_t MELODY_7_FREQUENCIES[] = {349,0,262,0,220,
0,262,0,220,0,349,0,349,0,262,0,220,0,262,0,220,0,349,0,
330,0,262,0,196,0,262,0,196,0,330,0,330,0,262,0,196,0,26
2,0,196,0,330,0,330,0,277,0,220,0,277,0,220,0,330,0,330,
0,277,0,220,0,277,0,220,0,330,0,294,0,330,0,349,0,440,0,
392,0,440,0,262,0,294,0,330,0,349,0,330,0,392,0,440,0,39
2,0,349,0,349,0,349,0,349,0,440,0,440,0,392,0,349,0,440,
0,440,0,440,0,392,0,440,0,392,0,349,0,349,0,349,0,349,0,
440,0,440,0,392,0,349,0,440,0,440,0,440,0,554,0,554,0,55
4,0,349,0,349,0,349,0,440,0,440,0,392,0,349,0};
// note durations (ms)
const uint16_t MELODY_7_DURATIONS[] = {125,37,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,500,150,333,99,333,99,250,75,250,75,500,150,50
0,150,500,150,333,99,333,99,250,75,250,75,500,150,500,15
0,500,150,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,125,37,333,99,333,
99,333,99,333,99,333,99,500,150,333,99,333,99,333,99,333
,99,333,99,333,99,333,99};
```

Файл melody_8.h

```
#ifndef MELODY_8_H
#define MELODY_8_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_8_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_8_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_8_DURATIONS[];
#endif
```

Файл melody_8.c

```
#include "melody_8.h"
// number of notes
const size_t MELODY_8_LEN = 76;
// note frequencies (Hz)
const uint16_t MELODY_8_FREQUENCIES[] = {330,0,392,0,440,
0,440,0,0,0,440,0,494,0,523,0,523,0,0,0,523,0,587,0,494,
0,494,0,0,0,440,0,392,0,440,0,0,0,330,0,392,0,440,0,440,
0,0,0,440,0,494,0,523,0,523,0,0,0,523,0,587,0,494,0,494,
0,0,0,440,0,392,0,440,0,0,0,330,0,392,0,440,0,440,0,0,0,
440,0,523,0,587,0,587,0,0,0,587,0,659,0,698,0,698,0,0,0,
659,0,587,0,659,0,440,0,0,0,440,0,494,0,523,0,523,0,0,0,
587,0,659,0,440,0,0,0,440,0,523,0,494,0,494,0,0,0,523,0,
440,0,494,0,0,0};
// note durations (ms)
const uint16_t MELODY_8_DURATIONS[] = {125,37,125,37,250
,75,125,37,125,37,125,37,125,37,250,75,125,37,125,37,125
,37,125,37,250,75,125,37,125,37,125,37,125,37,500,150,12
5,37,125,37,125,37,250,75,125,37,125,37,125,37,125,37,25
0,75,125,37,125,37,125,37,125,37,250,75,125,37,125,37,12
5,37,125,37,500,150,125,37,125,37,125,37,250,75,125,37,1
25,37,125,37,125,37,250,75,125,37,125,37,125,37,125,37,2
50,75,125,37,125,37,125,37,125,37,125,37,250,75,125,37,1
25,37,125,37,250,75,125,37,125,37,250,75,125,37,250,75,1
25,37,125,37,125,37,250,75,125,37,125,37,125,37,125,37,5
00,150,500,150};
```

Файл melody_9.h

```
#ifndef MELODY_9_H
#define MELODY_9_H
#include "stddef.h"
#include "stdint.h"
// number of notes
extern const size_t MELODY_9_LEN;
// note frequencies (Hz)
extern const uint16_t MELODY_9_FREQUENCIES[];
// note durations (ms)
extern const uint16_t MELODY_9_DURATIONS[];
#endif
```

Файл melody_9.c

```
#include "melody_9.h"
// number of notes
const size_t MELODY_9_LEN = 344;
// note frequencies (Hz)
const uint16_t MELODY_9_FREQUENCIES[] = {147,0,147,0,294,
0,220,0,0,0,208,0,196,0,175,0,147,0,175,0,196,0,131,0,13
1,0,294,0,220,0,0,0,208,0,196,0,175,0,147,0,175,0,196,0,
123,0,123,0,294,0,220,0,0,0,208,0,196,0,175,0,147,0,175,
0,196,0,117,0,117,0,294,0,220,0,0,0,208,0,196,0,175,0,14
7,0,175,0,196,0,147,0,147,0,294,0,220,0,0,0,208,0,196,0,
175,0,147,0,175,0,196,0,131,0,131,0,294,0,220,0,0,0,208,
0,196,0,175,0,147,0,175,0,196,0,123,0,123,0,294,0,220,0,
0,0,208,0,196,0,175,0,147,0,175,0,196,0,117,0,117,0,294,
0,220,0,0,0,208,0,196,0,175,0,147,0,175,0,196,0,294,0,29
4,0,587,0,440,0,0,0,415,0,392,0,349,0,294,0,349,0,392,0,
262,0,262,0,587,0,440,0,0,0,415,0,392,0,349,0,294,0,349,
0,392,0,247,0,247,0,587,0,440,0,0,0,415,0,392,0,349,0,2
94,0,349,0,392,0,233,0,233,0,587,0,440,0,0,0,415,0,392,
0,349,0,294,0,349,0,392,0,294,0,294,0,587,0,440,0,0,0,41
5,0,392,0,349,0,294,0,349,0,392,0,262,0,262,0,587,0,440,
0,0,0,415,0,392,0,349,0,294,0,349,0,392,0,247,0,247,0,58
7,0,440,0,0,0,415,0,392,0,349,0,294,0,349,0,392,0,233,0,
233,0,587,0,440,0,0,0,415,0,392,0,349,0,294,0,349,0,392,
0,349,0,349,0,349,0,349,0,349,0,294,0,294,0,294,0,349,0,
349,0,349,0,392,0,415,0,392,0,349,0,294,0,349,0,392,0,0,
0,349,0,349,0,349,0,392,0,415,0,440,0,523,0,440,0,587,0,
587,0,587,0,440,0,587,0,523,0,349,0,349,0,349,0,349,0,34
9,0,294,0,294,0,294,0,349,0,349,0,349,0,349,0,294,0,349,
0,330,0,294,0,262,0,0,0,392,0,330,0,294,0,294,0,294,0,29
4,0,175,0,196,0,233,0,262,0,294,0,349,0,523,0,0,0,349,0,
294,0,349,0,392,0,415,0,392,0,349,0,294,0,415,0,392,0,34
9,0,294,0,349,0,349,0,349,0,415,0,440,0,523,0,440,0,415,
0,392,0,349,0,294,0,330,0,349,0,392,0,440,0,523,0,554,0,
415,0,415,0,392,0,349,0,392,0,175,0,196,0,220,0,349,0,33
0,0,294,0,330,0,349,0,392,0,330,0,440,0,440,0,392,0,349,
0,311,0,277,0,311,0,0,0,349,0,294,0,349,0,392,0,415,0,39
2,0,349,0,294,0,415,0,392,0,349,0,294,0,349,0,349,0,349,
0,415,0,440,0,523,0,440,0,415,0,392,0,349,0,294,0,330,0,
349,0,392,0,440,0,523,0,554,0,415,0,415,0,392,0,349,0,39
2,0,175,0,196,0,220,0,349,0,330,0,294,0,330,0,349,0,392,
0,330,0,440,0,440,0,392,0,349,0,311,0,277,0,311,0};
```

```
// note durations (ms)
const uint16_t MELODY_9_DURATIONS[] = {62,18,62,18,125,37
,166,49,31,9,125,37,125,37,125,37,62,18,62,18,62,18,62,1
8,62,18,125,37,166,49,31,9,125,37,125,37,125,37,62,18,62
,18,62,18,62,18,62,18,125,37,166,49,31,9,125,37,125,37,1
25,37,62,18,62,18,62,18,62,18,62,18,125,37,166,49,31,9,1
25,37,125,37,125,37,62,18,62,18,62,18,62,18,62,18,125,37
,166,49,31,9,125,37,125,37,125,37,62,18,62,18,62,18,62,1
8,62,18,125,37,166,49,31,9,125,37,125,37,125,37,62,18,62
,18,62,18,62,18,62,18,125,37,166,49,31,9,125,37,125,37,1
25,37,62,18,62,18,62,18,62,18,62,18,125,37,166,49,31,9,1
25,37,125,37,125,37,62,18,62,18,62,18,62,18,62,18,125,37
,166,49,31,9,125,37,125,37,125,37,62,18,62,18,62,18,62,1
8,62,18,125,37,166,49,31,9,125,37,125,37,125,37,62,18,62
,18,62,18,62,18,62,18,125,37,166,49,31,9,125,37,125,37,1
25,37,62,18,62,18,62,18,62,18,62,18,125,37,166,49,31,9,1
25,37,125,37,125,37,62,18,62,18,62,18,62,18,62,18,125,37
,166,49,31,9,125,37,125,37,125,37,62,18,62,18,62,18,62,1
8,62,18,125,37,166,49,31,9,125,37,125,37,125,37,62,18,62
,18,62,18,62,18,62,18,125,37,166,49,31,9,125,37,125,37,1
25,37,62,18,62,18,62,18,62,18,62,18,125,37,166,49,31,9,1
25,37,125,37,125,37,62,18,62,18,62,18,125,37,62,18,125,
37,125,37,125,37,125,37,250,75,62,18,125,37,62,18,125,3
7,125,37,125,37,62,18,62,18,62,18,62,18,62,18,125,37,125
,37,62,18,125,37,125,37,125,37,125,37,125,37,125,37,125,
37,125,37,62,18,62,18,62,18,500,150,125,37,62,18,125,37,
125,37,125,37,125,37,250,75,62,18,125,37,62,18,125,37,12
5,37,125,37,125,37,125,37,62,18,125,37,62,18,125,37,125,
37,125,37,125,37,125,37,125,37,125,37,62,18,125,37,66,20
,125,37,125,37,500,150,333,99,62,18,62,18,62,18,62,18,62
,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,125,37,500
,150,62,18,125,37,62,18,125,37,62,18,62,18,62,18,62,18,6
2,18,62,18,125,37,125,37,125,37,125,37,125,37,125,37,62,
18,62,18,62,18,500,150,125,37,125,37,125,37,125,37,250,7
5,250,75,250,75,250,75,250,75,250,75,500,150,125,37,125,
37,125,37,125,37,500,150,500,150,333,99,62,18,62,18,62,1
8,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,62,18,
125,37,500,150,62,18,125,37,62,18,125,37,62,18,62,18,62,
18,62,18,62,18,62,18,125,37,125,37,125,37,125,37,125,37,
125,37,62,18,62,18,62,18,500,150,125,37,125,37,125,37,12
5,37,250,75,250,75,250,75,250,75,250,75,250,75,500,150,1
25,37,125,37,125,37,125,37,500,150,500,150};
```

Драйвера контроллера дисплея ILI9341

Файл open32f3_lcd_utils.h

```
#ifndef OPEN32F3_LCD_UTILS_H
#define OPEN32F3_LCD_UTILS_H

#ifdef __cplusplus
extern "C" {
#endif

#include "stm32f3xx_hal.h"

// helper macros with ports and pins
#define OPEN32F3_LCD_CONTROL_PORT GPIOB
#define OPEN32F3_LCD_DATA_PORT GPIOD
#define OPEN32F3_LCD_CS_PIN GPIO_PIN_8
#define OPEN32F3_LCD_RS_PIN GPIO_PIN_9
#define OPEN32F3_LCD_WR_PIN GPIO_PIN_10
#define OPEN32F3_LCD_RD_PIN GPIO_PIN_11
#define OPEN32F3_LCD_RESET_PIN GPIO_PIN_11
#define OPEN32F3_LCD_RESET_PORT GPIOC
#define OPEN32F3_LCD_BACKLIGHT_PIN GPIO_PIN_10
#define OPEN32F3_LCD_BACKLIGHT_PORT GPIOC
#define OPEN32F3_LCD_PORT_MODER_INPUT 0x00000000
#define OPEN32F3_LCD_PORT_MODER_OUTPUT 0x55555555

// enable lcd bus
#define OPEN32F3_LCD_CLEAR_CS() OPEN32F3_LCD_CONTROL_PORT-
>BRR |= OPEN32F3_LCD_CS_PIN
// disable lcd bus
#define  OPEN32F3_LCD_SET_CS()  OPEN32F3_LCD_CONTROL_PORT-
>BSRR |= OPEN32F3_LCD_CS_PIN
// switch to index mode
#define OPEN32F3_LCD_CLEAR_RS() OPEN32F3_LCD_CONTROL_PORT-
>BRR |= OPEN32F3_LCD_RS_PIN
// switch to data mode
#define  OPEN32F3_LCD_SET_RS()  OPEN32F3_LCD_CONTROL_PORT-
>BSRR |= OPEN32F3_LCD_RS_PIN
// start write signal
#define OPEN32F3_LCD_CLEAR_WR() OPEN32F3_LCD_CONTROL_PORT-
>BRR |= OPEN32F3_LCD_WR_PIN
// stop write signal
```

```
#define  OPEN32F3_LCD_SET_WR()  OPEN32F3_LCD_CONTROL_PORT-
>BSRR |= OPEN32F3_LCD_WR_PIN
// start read signal
#define OPEN32F3_LCD_CLEAR_RD() OPEN32F3_LCD_CONTROL_PORT-
>BRR |= OPEN32F3_LCD_RD_PIN
// stop read signal
#define  OPEN32F3_LCD_SET_RD()  OPEN32F3_LCD_CONTROL_PORT-
>BSRR |= OPEN32F3_LCD_RD_PIN
// read data from data bus
#define  OPEN32F3_LCD_DATA_READ()  OPEN32F3_LCD_DATA_PORT-
>IDR
// write data from data bus
#define  OPEN32F3_LCD_DATA_WRITE(value)  OPEN32F3_LCD_DATA_
PORT->ODR = value
// switch data bus pins to input mode
#define  OPEN32F3_LCD_DATA_SET_INPUT_MODE()  OPEN32F3_LCD_
DATA_PORT->MODER = OPEN32F3_LCD_PORT_MODER_INPUT
// switch data bus pins to output mode
#define  OPEN32F3_LCD_DATA_SET_OUTPUT_MODE()  OPEN32F3_LCD_
DATA_PORT->MODER = OPEN32F3_LCD_PORT_MODER_OUTPUT

/**
 * LCD driver delay callback.
 */
int open32f3_lcd_delay(void *lcd_data, int ms);

/**
 * LCD driver reset callback.
 */
int open32f3_lcd_reset(void *lcd_data);

#ifdef __cplusplus
}
#endif

#endif // OPEN32F3_LCD_UTILS_H
```

Файл open32f3_lcd_utils.c

```
#include "open32f3_lcd_utils.h"

int open32f3_lcd_delay(void *lcd_data, int ms) {
    HAL_Delay(ms);
    return 0;
}
```

```
int open32f3_lcd_reset(void *lcd_data) {
    // reset LCD
     HAL_GPIO_WritePin(OPEN32F3_LCD_RESET_PORT, OPEN32F3_
LCD_RESET_PIN, GPIO_PIN_RESET);
    HAL_Delay(100);
     HAL_GPIO_WritePin(OPEN32F3_LCD_RESET_PORT, OPEN32F3_
LCD_RESET_PIN, GPIO_PIN_SET);
    return 0;
}
```

Файл lcd_ili93xx_driver.h

```
#ifndef LCD_ILI93XX_DRIVER
#define LCD_ILI93XX_DRIVER

#include "stddef.h"
#include "stdint.h"

#ifdef __cplusplus
extern "C" {
#endif

/**
 * Common ILI9325 LCD driver.
 *
 * Driver usage:
 *
 * 1. Create and clear driver:
 *
 * @code{.cpp}
 * lcd_ili93xx_driver_t lcd_driver;
 * lcd_ili93xx_init_clear(&lcd_driver);
 * @endcode
 *
 * 2. Attach callbacks to handler:
 *
 * @code{.cpp}
 * // set custom data. If don't need any data, assign NULL.
 * lcd_driver.user_data = my_data_ptr;
 * // Set delay callback (it's needed for LCD initialization)
 * lcd_driver.delay = my_delay;
 * // Set callback that resets LCD.
 * lcd_driver.reset = my_lcd_reset_callback;
  * // Set callback that can write a single value to
register.
 * lcd_driver.write_reg = my_lcd_write_reg;
```

```
 * // Set callback that can read a single value from
register.
 * lcd_driver.read_reg = my_lcd_read_reg;
 * // Set callback that can write multiple word to register.
 * lcd_driver.write_words = my_lcd_write_words;
 * @endcode
 *
 * 3. Initialize driver:
 *
 * @code{.cpp}
 * int res = lcd_ili93xx_init(&lcd_driver);
 * if (!res) {
 *     // Driver initialization error.
 *     // Show error somehow and stop application
 *     // ...
 * }
 * @endcode
 *
 * 4. Use driver for some operations.
 *
 * @code{.cpp}
 * lcd_ili93xx_fill_area_color(&lcd_driver, 0, 0, 240 - 1,
320 - 1, LCD_ILI93XX_COLOR_GREEN);
 * ...
 * @endcode
 *
 */
typedef struct {
    /**
     * custom data.
     */
    void *user_data;

    /**
     * Delay callback.
     *
     * @param lcd_data custom data
     * @param ms expected delay in milliseconds
     * @return zero on success, otherwise non-zero value
     */
    int (*delay)(void *user_data, int ms);

    /**
     * Hardware LCD reset callback.
     *
```

```
     * @param lcd_data custom data
     * @return zero on success, otherwise non-zero value
     */
    int (*reset)(void *user_data);

    /**
     * Write LCD register callback.
     *
     * @param lcd_data custom data
     * @param address register address
     * @param value register value
     * @return zero on success, otherwise non-zero value
     */
     int (*write_reg)(void *user_data, uint16_t address,
uint16_t value);

    /**
     * Write LCD register callback.
     *
     * @param lcd_data custom data
     * @param address register address
     * @param value register value address
     * @return zero on success, otherwise non-zero value
     */
     int (*read_reg)(void *user_data, uint16_t address,
uint16_t *value);

    /**
     * Write words to LCD register callback.
     *
     * @param lcd_data custom data
     * @param address register address
     * @param data array with words
     * @param size data array size
     * @return zero on success, otherwise non-zero value
     */
    int (*write_words)(void *user_data, uint16_t address,
uint16_t *data, size_t size);

    // private variables
    int _is_initialized;
    int _model;
    int16_t _width;
    int16_t _height;
} lcd_ili93xx_driver_t;
```

```
/**
 * Clear lcd_ili93xx_driver_t driver struct.
 *
 * @param driver
 * @return zero on success, otherwise non-zero value
 */
int lcd_ili93xx_init_clear(lcd_ili93xx_driver_t *driver);

/**
 * Initialize LCD.
 *
 * @param driver
 * @return zero on success, otherwise non-zero value
 */
int lcd_ili93xx_init(lcd_ili93xx_driver_t *driver);

/**
 * Get LCD width.
 *
 * @param driver
 * @param width
 * @return zero on success, otherwise non-zero value
 */
int  lcd_ili93xx_get_width(lcd_ili93xx_driver_t  *driver,
int16_t *width);

/**
 * Get LCD height.
 *
 * @param driver
 * @param height
 * @return zero on success, otherwise non-zero value
 */
int  lcd_ili93xx_get_height(lcd_ili93xx_driver_t  *driver,
int16_t *height);

/**
 * Fill specified area with a color.
 *
 * @param driver
 * @param x1 left rectangle coordinate (inclusive)
 * @param y1 upper rectangle coordinate (inclusive)
 * @param x2 right rectangle coordinate (inclusive)
 * @param y2 bottom rectangle coordinate (inclusive)
```

```
 * @param color rectangle color
 * @return zero on success, otherwise non-zero value
 */
int       lcd_ili93xx_fill_area_color(lcd_ili93xx_driver_t
*driver, int16_t x1, int16_t y1, int16_t x2, int16_t y2,
uint16_t color);

/**
 * Fill specified area.
 *
 * @param driver
 * @param x1 left rectangle coordinate (inclusive)
 * @param y1 upper rectangle coordinate (inclusive)
 * @param x2 right rectangle coordinate (inclusive)
 * @param y2 bottom rectangle coordinate (inclusive)
 * @param color 2d array with rectangle pixels
 * @return zero on success, otherwise non-zero value
 */
int   lcd_ili93xx_fill_area(lcd_ili93xx_driver_t   *driver,
int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t
*colors_data);

/**
 * Helper function to convert RGB color to RGB565 one.
 *
 * @param red red component from 0 to 255
 * @param green green component from 0 to 255
 * @param blue blue component from 0 to 255
 * @return RGB565 color
 */
inline  uint16_t  lcd_ili93xx_rgb565_convert(uint8_t  red,
uint8_t green, uint8_t blue) {
     return (((red >> 3) << 11) | ((green >> 2) << 5) |
(blue >> 3));
}

/**
 * Helper macros to convert RGB color to RGB565 one.
 *
 * @param red red component from 0 to 255
 * @param green green component from 0 to 255
 * @param blue blue component from 0 to 255
 * @return RGB565 color
 */
```

```
#define  LCD_ILI93XX_RGB565_CONVERT_M(red,  green,  blue)
(((red >> 3) << 11) | ((green >> 2) << 5) | (blue >> 3))

/**
 * Color constants.
 */
enum lcd_ili93xx_colors {
         LCD_ILI93XX_COLOR_BLACK  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x00, 0x00, 0x00),
        LCD_ILI93XX_COLOR_MAROON  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x80, 0x00, 0x00),
         LCD_ILI93XX_COLOR_GREEN  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x00, 0x80, 0x00),
         LCD_ILI93XX_COLOR_OLIVE  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x80, 0x80, 0x00),
          LCD_ILI93XX_COLOR_NAVY  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x00, 0x00, 0x80),
        LCD_ILI93XX_COLOR_PURPLE  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x80, 0x00, 0x80),
         LCD_ILI93XX_COLOR_TEAL  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x00, 0x80, 0x80),
        LCD_ILI93XX_COLOR_SILVER  =  LCD_ILI93XX_RGB565_
CONVERT_M(0xC0, 0xC0, 0xC0),
         LCD_ILI93XX_COLOR_GRAY  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x80, 0x80, 0x80),
          LCD_ILI93XX_COLOR_RED  =  LCD_ILI93XX_RGB565_
CONVERT_M(0xFF, 0x00, 0x00),
         LCD_ILI93XX_COLOR_LIME  =  LCD_ILI93XX_RGB565_
CONVERT_M(0xFF, 0xFF, 0x00),
         LCD_ILI93XX_COLOR_BLUE  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x00, 0x00, 0xFF),
       LCD_ILI93XX_COLOR_FUCHSIA  =  LCD_ILI93XX_RGB565_
CONVERT_M(0xFF, 0x00, 0xFF),
         LCD_ILI93XX_COLOR_AQUA  =  LCD_ILI93XX_RGB565_
CONVERT_M(0x00, 0xFF, 0xFF),
        LCD_ILI93XX_COLOR_WHITE  =  LCD_ILI93XX_RGB565_
CONVERT_M(0xFF, 0xFF, 0xFF)
};

#ifdef __cplusplus
}
#endif

#endif
```

Файл lcd_ili93xx_driver.c

```
#include "lcd_ili93xx_driver.h"
#include "stdbool.h"
#include "stdlib.h"

#define ILI93XX_LCD_WIDTH 240
#define ILI93XX_LCD_HEIGTH 320

/* LCD models */
#define ILI93XX_LCD_MODEL_ILI9320 0 /* 0x9320 */
#define ILI93XX_LCD_MODEL_ILI9325 1 /* 0x9325 */
#define ILI93XX_LCD_MODEL_ILI9328 2 /* 0x9328 */
#define ILI93XX_LCD_MODEL_ILI9331 3 /* 0x9331 */

// helper register addresses
#define ILI93XX_LCD_REGISTER_GRAM_X_ADDR 0x0020
#define ILI93XX_LCD_REGISTER_GRAM_Y_ADDR 0x0021
#define ILI93XX_LCD_REGISTER_GRAM_WRITE_DATA 0x0022

#define ILI93XX_LCD_REGISTER_GRAM_X_ADDR_START 0x0050
#define ILI93XX_LCD_REGISTER_GRAM_X_ADDR_END 0x0051
#define ILI93XX_LCD_REGISTER_GRAM_Y_ADDR_START 0x0052
#define ILI93XX_LCD_REGISTER_GRAM_Y_ADDR_END 0x0053

int lcd_ili93xx_init_clear(lcd_ili93xx_driver_t *driver)
{
    driver->user_data = NULL;
    driver->reset = NULL;
    driver->read_reg = NULL;
    driver->write_reg = NULL;
    driver->write_words = NULL;
    driver->_height = ILI93XX_LCD_HEIGTH;
    driver->_width = ILI93XX_LCD_WIDTH;
    driver->_is_initialized = 0;
    driver->_model = -1;
    return 0;
}

static    int    ili93xx_lcd_test_write_reg(lcd_ili93xx_
driver_t *driver, uint16_t test_reg_addr, uint16_t test_
vals[2]) {
    uint16_t act_val = 0;
    for (int i = 0; i < 2; i++) {
```

```
            driver->write_reg(driver->user_data, test_reg_
addr, test_vals[i]);
        driver->read_reg(driver->user_data, test_reg_addr,
&act_val);
            if (act_val != test_vals[i]) {
                return -4;
            }
        }
        return 0;
}

int lcd_ili93xx_init(lcd_ili93xx_driver_t *driver) {
        int err;

        if (driver->_is_initialized) {
            return -2;
        }

        if (driver->reset == NULL) {
            return -1;
        }
        if (driver->read_reg == NULL) {
            return -1;
        }
        if (driver->write_reg == NULL) {
            return -1;
        }
        if (driver->write_words == NULL) {
            return -1;
        }

        // reset LCD
        driver->reset(driver->user_data);

        // read device code
        uint16_t device_code = 0;
        driver->read_reg(driver->user_data, 0x0000, &device_
code);

        if (device_code == 0x9325 || device_code == 0x9328) {
            driver->_model = ILI93XX_LCD_MODEL_ILI9325;
            // test write operation
            uint16_t test_vals[2] = { 0x0077, 0x0000 };
               if ((err = ili93xx_lcd_test_write_reg(driver,
0x0020, test_vals))) {
                return err;
            }
```

```
      driver->write_reg(driver->user_data, 0xE5, 0x78F0);
/* set SRAM internal timing */
      driver->write_reg(driver->user_data, 0x01, 0x0100);
/* set Driver Output Control */
      driver->write_reg(driver->user_data, 0x02, 0x0700);
/* set 1 line inversion */
      driver->write_reg(driver->user_data, 0x03, 0x1030);
/* set GRAM write direction and BGR=1 */
      driver->write_reg(driver->user_data, 0x04, 0x0000);
/* Resize register */
      driver->write_reg(driver->user_data, 0x08, 0x0202);
/* set the back porch and front porch. Note: with an
original value 0x0207 an image trembles */
      driver->write_reg(driver->user_data, 0x09, 0x0000);
/* set non-display area refresh cycle ISC[3:0] */
      driver->write_reg(driver->user_data, 0x0A, 0x0000);
/* FMARK function */
      driver->write_reg(driver->user_data, 0x0C, 0x0000);
/* RGB interface setting */
      driver->write_reg(driver->user_data, 0x0D, 0x0000);
/* Frame marker Position */
      driver->write_reg(driver->user_data, 0x0F, 0x0000);
/* RGB interface polarity */
        /*************Power On sequence ***************/
      driver->write_reg(driver->user_data, 0x10, 0x0000);
/* SAP, BT[3:0], AP, DSTB, SLP, STB */
      driver->write_reg(driver->user_data, 0x11, 0x0007);
/* DC1[2:0], DC0[2:0], VC[2:0] */
      driver->write_reg(driver->user_data, 0x12, 0x0000);
/* VREG1OUT voltage */
      driver->write_reg(driver->user_data, 0x13, 0x0000);
/* VDV[4:0] for VCOM amplitude */
      driver->write_reg(driver->user_data, 0x07, 0x0001);
        driver->delay(driver->user_data, 200);
        /* Dis-charge capacitor power voltage */
      driver->write_reg(driver->user_data, 0x10, 0x1090);
/* SAP, BT[3:0], AP, DSTB, SLP, STB */
      driver->write_reg(driver->user_data, 0x11, 0x0227);
/* Set DC1[2:0], DC0[2:0], VC[2:0] */
          driver->delay(driver->user_data, 50); /* Delay
50ms */
      driver->write_reg(driver->user_data, 0x12, 0x001F);
          driver->delay(driver->user_data, 50); /* Delay
50ms */
```

```
      driver->write_reg(driver->user_data, 0x13, 0x1500);
/* VDV[4:0] for VCOM amplitude */
      driver->write_reg(driver->user_data, 0x29, 0x0027);
/* 04 VCM[5:0] for VCOMH */
      driver->write_reg(driver->user_data, 0x2B, 0x000D);
/* Set Frame Rate */
          driver->delay(driver->user_data, 50); /* Delay
50ms */
      driver->write_reg(driver->user_data, 0x20, 0x0000);
/* GRAM horizontal Address */
      driver->write_reg(driver->user_data, 0x21, 0x0000);
/* GRAM Vertical Address */
       /* ----------- Adjust the Gamma Curve ---------- */
      driver->write_reg(driver->user_data, 0x30, 0x0000);
      driver->write_reg(driver->user_data, 0x31, 0x0707);
      driver->write_reg(driver->user_data, 0x32, 0x0307);
      driver->write_reg(driver->user_data, 0x35, 0x0200);
      driver->write_reg(driver->user_data, 0x36, 0x0008);
      driver->write_reg(driver->user_data, 0x37, 0x0004);
      driver->write_reg(driver->user_data, 0x38, 0x0000);
      driver->write_reg(driver->user_data, 0x39, 0x0707);
      driver->write_reg(driver->user_data, 0x3C, 0x0002);
      driver->write_reg(driver->user_data, 0x3D, 0x1D04);
        /* ------------------ Set GRAM area ------------
--- */
      driver->write_reg(driver->user_data, 0x50, 0x0000);
/* Horizontal GRAM Start Address */
      driver->write_reg(driver->user_data, 0x51, 0x00EF);
/* Horizontal GRAM End Address */
      driver->write_reg(driver->user_data, 0x52, 0x0000);
/* Vertical GRAM Start Address */
      driver->write_reg(driver->user_data, 0x53, 0x013F);
/* Vertical GRAM Start Address */
      driver->write_reg(driver->user_data, 0x60, 0xA700);
/* Gate Scan Line */
      driver->write_reg(driver->user_data, 0x61, 0x0001);
/* NDL,VLE, REV */
      driver->write_reg(driver->user_data, 0x6A, 0x0000);
/* set scrolling line */
        /* -------------- Partial Display Control ------
--- */
      driver->write_reg(driver->user_data, 0x80, 0x0000);
      driver->write_reg(driver->user_data, 0x81, 0x0000);
      driver->write_reg(driver->user_data, 0x82, 0x0000);
      driver->write_reg(driver->user_data, 0x83, 0x0000);
```

```
      driver->write_reg(driver->user_data, 0x84, 0x0000);
      driver->write_reg(driver->user_data, 0x85, 0x0000);
        /* -------------- Panel Control ----------------
--- */
      driver->write_reg(driver->user_data, 0x90, 0x0010);
      driver->write_reg(driver->user_data, 0x92, 0x0600);
      driver->write_reg(driver->user_data, 0x07, 0x0133);
/* 262K color and display ON */
     } else if (device_code == 0x9320 || device_code ==
0x9300) {
        driver->_model = ILI93XX_LCD_MODEL_ILI9320;
        // test write operation
        uint16_t test_vals[2] = { 0x0077, 0x0000 };
           if ((err = ili93xx_lcd_test_write_reg(driver,
0x0020, test_vals))) {
               return err;
        }

      driver->write_reg(driver->user_data, 0x00, 0x0000);
      driver->write_reg(driver->user_data, 0x01, 0x0100);
/* Driver Output Contral */
      driver->write_reg(driver->user_data, 0x02, 0x0700);
/* LCD Driver Waveform Contral */
      driver->write_reg(driver->user_data, 0x03, 0x1018);
/* Entry Mode Set */

      driver->write_reg(driver->user_data, 0x04, 0x0000);
/* Scalling Contral */
      driver->write_reg(driver->user_data, 0x08, 0x0202);
/* Display Contral */
      driver->write_reg(driver->user_data, 0x09, 0x0000);
/* Display Contral 3.(0x0000) */
      driver->write_reg(driver->user_data, 0x0a, 0x0000);
/* Frame Cycle Contal.(0x0000) */
        driver->write_reg(driver->user_data, 0x0c, (1 <<
0)); /* Extern Display Interface Contral */
      driver->write_reg(driver->user_data, 0x0d, 0x0000);
/* Frame Maker Position */
      driver->write_reg(driver->user_data, 0x0f, 0x0000);
/* Extern Display Interface Contral 2. */

         driver->delay(driver->user_data, 100); /* delay
100 ms */
      driver->write_reg(driver->user_data, 0x07, 0x0101);
/* Display Contral */
```

```
         driver->delay(driver->user_data, 100); /* delay
100 ms */

        driver->write_reg(driver->user_data, 0x10, (1 <<
12) | (0 << 8) | (1 << 7) | (1 << 6) | (0 << 4)); /* Power
Control 1.(0x16b0) */
      driver->write_reg(driver->user_data, 0x11, 0x0007);
/* Power Control 2 */
        driver->write_reg(driver->user_data, 0x12, (1 <<
8) | (1 << 4) | (0 << 0)); /* Power Control 3.(0x0138)
*/
      driver->write_reg(driver->user_data, 0x13, 0x0b00);
/* Power Control 4 */
      driver->write_reg(driver->user_data, 0x29, 0x0000);
/* Power Control 7 */

        driver->write_reg(driver->user_data, 0x2b, (1 <<
14) | (1 << 4));

       driver->write_reg(driver->user_data, 0x50, 0); /*
Set X Start */
        driver->write_reg(driver->user_data, 0x51, 239);
/* Set X End */
       driver->write_reg(driver->user_data, 0x52, 0); /*
Set Y Start */
        driver->write_reg(driver->user_data, 0x53, 319);
/* Set Y End */

      driver->write_reg(driver->user_data, 0x60, 0x2700);
/* Driver Output Control */
      driver->write_reg(driver->user_data, 0x61, 0x0001);
/* Driver Output Control */
      driver->write_reg(driver->user_data, 0x6a, 0x0000);
/* Vertical Srcoll Control */

      driver->write_reg(driver->user_data, 0x80, 0x0000);
/* Display Position? Partial Display 1 */
      driver->write_reg(driver->user_data, 0x81, 0x0000);
/* RAM Address Start? Partial Display 1 */
      driver->write_reg(driver->user_data, 0x82, 0x0000);
/* RAM Address End-Partial Display 1 */
      driver->write_reg(driver->user_data, 0x83, 0x0000);
/* Displsy Position? Partial Display 2 */
      driver->write_reg(driver->user_data, 0x84, 0x0000);
/* RAM Address Start? Partial Display 2 */
```

```
      driver->write_reg(driver->user_data, 0x85, 0x0000);
/* RAM Address End? Partial Display 2 */

        driver->write_reg(driver->user_data, 0x90, (0 <<
7) | (16 << 0)); /* Frame Cycle Contral.(0x0013)   */
      driver->write_reg(driver->user_data, 0x92, 0x0000);
/* Panel Interface Contral 2.(0x0000) */
      driver->write_reg(driver->user_data, 0x93, 0x0001);
/* Panel Interface Contral 3. */
      driver->write_reg(driver->user_data, 0x95, 0x0110);
/* Frame Cycle Contral.(0x0110) */
        driver->write_reg(driver->user_data, 0x97, (0 <<
8));
      driver->write_reg(driver->user_data, 0x98, 0x0000);
/* Frame Cycle Contral */

      driver->write_reg(driver->user_data, 0x07, 0x0173);
    } else if (device_code == 0x9331) {
        driver->_model = ILI93XX_LCD_MODEL_ILI9331;
            driver->write_reg(driver->user_data,  0x00E7,
0x1014);
            driver->write_reg(driver->user_data,  0x0001,
0x0100); /* set SS and SM bit */
            driver->write_reg(driver->user_data,  0x0002,
0x0200); /* set 1 line inversion */
            driver->write_reg(driver->user_data,  0x0003,
0x1030); /* set GRAM write direction and BGR=1 */
            driver->write_reg(driver->user_data,  0x0008,
0x0202); /* set the back porch and front porch */
            driver->write_reg(driver->user_data,  0x0009,
0x0000); /* set non-display area refresh cycle ISC[3:0] */
            driver->write_reg(driver->user_data,  0x000A,
0x0000); /* FMARK function */
            driver->write_reg(driver->user_data,  0x000C,
0x0000); /* RGB interface setting */
            driver->write_reg(driver->user_data,  0x000D,
0x0000); /* Frame marker Position */
            driver->write_reg(driver->user_data,  0x000F,
0x0000); /* RGB interface polarity */
        /* Power On sequence */
            driver->write_reg(driver->user_data,  0x0010,
0x0000); /* SAP, BT[3:0], AP, DSTB, SLP, STB*/
            driver->write_reg(driver->user_data,  0x0011,
0x0007); /* DC1[2:0], DC0[2:0], VC[2:0] */
```

```
            driver->write_reg(driver->user_data,  0x0012,
0x0000); /* VREG1OUT voltage    */
            driver->write_reg(driver->user_data,  0x0013,
0x0000); /* VDV[4:0] for VCOM amplitude */
          driver->delay(driver->user_data, 200); /* delay
200 ms */
            driver->write_reg(driver->user_data,  0x0010,
0x1690); /* SAP, BT[3:0], AP, DSTB, SLP, STB*/
            driver->write_reg(driver->user_data,  0x0011,
0x0227); /* DC1[2:0], DC0[2:0], VC[2:0] */
        driver->delay(driver->user_data, 50); /* delay 50
ms */
            driver->write_reg(driver->user_data,  0x0012,
0x000C); /* Internal reference voltage= Vci */
        driver->delay(driver->user_data, 50); /* delay 50
ms */
            driver->write_reg(driver->user_data,  0x0013,
0x0800); /* Set VDV[4:0] for VCOM amplitude */
            driver->write_reg(driver->user_data,  0x0029,
0x0011); /* Set VCM[5:0] for VCOMH */
            driver->write_reg(driver->user_data,  0x002B,
0x000B); /* Set Frame Rate */
        driver->delay(driver->user_data, 50); /* delay 50
ms */
            driver->write_reg(driver->user_data,  0x0020,
0x0000); /* GRAM horizontal Address */
            driver->write_reg(driver->user_data,  0x0021,
0x0000); /* GRAM Vertical Address */
         /* Adjust the Gamma Curve */
            driver->write_reg(driver->user_data,  0x0030,
0x0000);
            driver->write_reg(driver->user_data,  0x0031,
0x0106);
            driver->write_reg(driver->user_data,  0x0032,
0x0000);
            driver->write_reg(driver->user_data,  0x0035,
0x0204);
            driver->write_reg(driver->user_data,  0x0036,
0x160A);
            driver->write_reg(driver->user_data,  0x0037,
0x0707);
            driver->write_reg(driver->user_data,  0x0038,
0x0106);
            driver->write_reg(driver->user_data,  0x0039,
0x0707);
```

```
            driver->write_reg(driver->user_data,  0x003C,
0x0402);
            driver->write_reg(driver->user_data,  0x003D,
0x0C0F);
        /* Set GRAM area */
            driver->write_reg(driver->user_data,  0x0050,
0x0000); /* Horizontal GRAM Start Address */
            driver->write_reg(driver->user_data,  0x0051,
0x00EF); /* Horizontal GRAM End Address */
            driver->write_reg(driver->user_data,  0x0052,
0x0000); /* Vertical GRAM Start Address */
            driver->write_reg(driver->user_data,  0x0053,
0x013F); /* Vertical GRAM Start Address */
            driver->write_reg(driver->user_data,  0x0060,
0x2700); /* Gate Scan Line */
            driver->write_reg(driver->user_data,  0x0061,
0x0001); /*  NDL,VLE, REV */
            driver->write_reg(driver->user_data,  0x006A,
0x0000); /* set scrolling line */
        /* Partial Display Control */
            driver->write_reg(driver->user_data,  0x0080,
0x0000);
            driver->write_reg(driver->user_data,  0x0081,
0x0000);
            driver->write_reg(driver->user_data,  0x0082,
0x0000);
            driver->write_reg(driver->user_data,  0x0083,
0x0000);
            driver->write_reg(driver->user_data,  0x0084,
0x0000);
            driver->write_reg(driver->user_data,  0x0085,
0x0000);
        /* Panel Control */
            driver->write_reg(driver->user_data,  0x0090,
0x0010);
            driver->write_reg(driver->user_data,  0x0092,
0x0600);
            driver->write_reg(driver->user_data,  0x0007,
0x0021);
       driver->delay(driver->user_data, 50); /* delay 50
ms */
            driver->write_reg(driver->user_data,  0x0007,
0x0061);
       driver->delay(driver->user_data, 50); /* delay 50
ms */
```

```
            driver->write_reg(driver->user_data,  0x0007,
0x0133); /* 262K color and display ON */
    } else {
        return -3;
    }

    driver->_is_initialized = true;

    return 0;
}

int  lcd_ili93xx_get_height(lcd_ili93xx_driver_t  *driver,
int16_t *height) {
    *height = driver->_height;
    return 0;
}

int  lcd_ili93xx_get_width(lcd_ili93xx_driver_t  *driver,
int16_t *width) {
    *width = driver->_width;
    return 0;
}

static  int  lcd_ili93xx_prepare_area(lcd_ili93xx_driver_t
*driver, int16_t x1, int16_t y1, int16_t x2, int16_t y2) {
    // set window
    if (x1 > x2) {
        int16_t x_tmp = x1;
        x1 = x2;
        x2 = x_tmp;
    }
    if (y1 > y2) {
        int16_t y_tmp = y1;
        y1 = y2;
        y2 = y_tmp;
    }
    // set gram coordinates
       driver->write_reg(driver->user_data,  ILI93XX_LCD_
REGISTER_GRAM_X_ADDR, x1);
       driver->write_reg(driver->user_data,  ILI93XX_LCD_
REGISTER_GRAM_Y_ADDR, y1);
       driver->write_reg(driver->user_data,  ILI93XX_LCD_
REGISTER_GRAM_X_ADDR_START, x1);
       driver->write_reg(driver->user_data,  ILI93XX_LCD_
REGISTER_GRAM_X_ADDR_END, x2);
```

```
       driver->write_reg(driver->user_data,  ILI93XX_LCD_
REGISTER_GRAM_Y_ADDR_START, y1);
       driver->write_reg(driver->user_data,  ILI93XX_LCD_
REGISTER_GRAM_Y_ADDR_END, y2);
    return (x2 - x1 + 1) * (y2 - y1 + 1);
}

int       lcd_ili93xx_fill_area_color(lcd_ili93xx_driver_t
*driver, int16_t x1, int16_t y1, int16_t x2, int16_t y2,
uint16_t color) {
    // prepare area to draw
   int total_pixes = lcd_ili93xx_prepare_area(driver, x1,
y1, x2, y2);
    const size_t max_buff_size = 128;
    size_t buff_size = total_pixes < max_buff_size ? total_
pixes : max_buff_size;
    uint16_t buff[buff_size];
    for (size_t i = 0; i < buff_size; i++) {
        buff[i] = color;
    }
    size_t i = total_pixes;
    while (i > 0) {
       size_t pixels_to_update = i >= buff_size ? buff_size
: i;
        // write pixels
         driver->write_words(driver->user_data, ILI93XX_
LCD_REGISTER_GRAM_WRITE_DATA, buff, pixels_to_update);
        i -= pixels_to_update;
    }
    return 0;
}

int  lcd_ili93xx_fill_area(lcd_ili93xx_driver_t  *driver,
int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t
*colors_data) {
    // prepare area to draw
   int total_pixes = lcd_ili93xx_prepare_area(driver, x1,
y1, x2, y2);
    // flush pixels
     driver->write_words(driver->user_data,  ILI93XX_LCD_
REGISTER_GRAM_WRITE_DATA, colors_data, total_pixes);
    return 0;
}
```

Драйвера контроллера дисплея XPT2046

Файл open32f3_lcd_touch_utils.h

```
#ifndef LCD_UTILS_OPEN32F3_LCD_TOUCH_UTILS_H
#define LCD_UTILS_OPEN32F3_LCD_TOUCH_UTILS_H

#include "stddef.h"
#include "stdint.h"

#ifdef __cplusplus
extern "C" {
#endif

#define XPT2046_CMD_MEASURE_Z1 0xB0
#define XPT2046_CMD_MEASURE_Z2 0xC0
#define XPT2046_CMD_MEASURE_Y 0x90
#define XPT2046_CMD_MEASURE_X 0xD0

/**
 * Simple XPT2046 LCD touchpad driver to measure resistive
touchpad X,Y,Z1 and Z2 position.
 *
 * Driver usage:
 *
 * 1. Create and clear driver:
 *
 * @code{.cpp}
 * lcd_xpt2046_driver_t touch_driver;
 * lcd_xpt2046_init_clear(&touch_driver);
 * @endcode
 *
 * 2. Attach SPI callback.
 *
 * @code{.cpp}
 * // Set custom data. If you don't need any data, assign
NULL.
 * touch_driver.user_data = my_data_ptr;
 * // Set communication callback
 * touch_driver.communication_cb = my_communication_cb;
 * @endcode
 *
```

```
 * 3. Initialize driver:
 *
 * @code{.cpp}
 * int res = lcd_xpt2046_init(&touch_driver);
 * if (!res) {
 *     // Driver initialization error.
 *     // Show error somehow and stop application
 *     // ...
 * }
 * @endcode
 *
 * 4. Use driver to measure touch resistance.
 *
 * @code{.cpp}
 * int x_value;
 * int y_value;
 * int z1_value;
 * int z2_value;
 *
   *   lcd_xpt2046_measure(&touch_driver,   XPT2046_CMD_X_
MEASURE, &x_value);
   *   lcd_xpt2046_measure(&touch_driver,   XPT2046_CMD_y_
MEASURE, &y_value);
   *   lcd_xpt2046_measure(&touch_driver,   XPT2046_CMD_Z1_
MEASURE, &z1_value);
   *   lcd_xpt2046_measure(&touch_driver,   XPT2046_CMD_Z2_
MEASURE, &z2_value);
 *
 * ...
 * @endcode
 *
 */
typedef struct {
    /**
     * custom data.
     */
    void *user_data;
    /**
     * Touch sensor communication callback.
     *
     * @param lcd_data custom data
     * @param out_buf buffer with data to transfer
     * @param in_buf buffer with data to receive
```

```
     * @param size buffers length
     * @return zero on success, otherwise non-zero value
     */
    int (*communication_cb)(void *user_data, uint8_t *out_
buf, uint8_t *in_buf, size_t size);

    // private variables
    int _is_initialized;
} lcd_xpt2046_driver_t;

/**
 * Clear lcd_xpt2046_driver_t driver struct.
 *
 * @param driver
 * @return zero on success, otherwise non-zero value
 */
int lcd_xpt2046_init_clear(lcd_xpt2046_driver_t *driver);

/**
 * Initialize LCD touch controller.
 *
 * @param driver
 * @return zero on success, otherwise non-zero value
 */
int lcd_xpt2046_init(lcd_xpt2046_driver_t *driver);

/**
 * Measure resistance.
 *
 * @param driver
  *  @param measure_command - measure command XPT2046_
CMD_<TYPE>_MEASURE
 * @param result - measured value
 * @return zero on success, otherwise non-zero value
 */
int   lcd_xpt2046_measure(lcd_xpt2046_driver_t   *driver,
uint8_t measure_command, int *result);

#ifdef __cplusplus
}
#endif

#endif /* LCD_UTILS_OPEN32F3_LCD_TOUCH_UTILS_H */
```

Файл open32f3_lcd_touch_utils.c

```
#include "open32f3_lcd_touch_utils.h"

#define XPT2046_CONV_VAL(byte_0, byte_1) (byte_0 << 5 |
byte_1 >> 3)
#define XPT2046_CMD_DEFAULT_POWER_MODE 0x00

int lcd_xpt2046_init_clear(lcd_xpt2046_driver_t *driver)
{
    driver->user_data = NULL;
    driver->communication_cb = NULL;
    driver->_is_initialized = 0;
    return 0;
}

int lcd_xpt2046_init(lcd_xpt2046_driver_t *driver) {
    int err = 0;

    if (driver->_is_initialized) {
        return -2;
    }

    if (driver->communication_cb == NULL) {
        return -1;
    }

    // try to measure x, z1 and z2 to check interface
    uint8_t in_buf[7] = {
      XPT2046_CMD_MEASURE_X | XPT2046_CMD_DEFAULT_POWER_
MODE, 0x00,
     XPT2046_CMD_MEASURE_Z1 | XPT2046_CMD_DEFAULT_POWER_
MODE, 0x00,
     XPT2046_CMD_MEASURE_Z2 | XPT2046_CMD_DEFAULT_POWER_
MODE, 0x00, 0x00 };
    uint8_t out_buf[7] = { 0x00 };
    err = driver->communication_cb(driver->user_data, in_
buf, out_buf, 7);
    if (err) {
        return err;
    }
   uint16_t x = XPT2046_CONV_VAL(out_buf[1], out_buf[2]);
   uint16_t z1 = XPT2046_CONV_VAL(out_buf[3], out_buf[4]);
   uint16_t z2 = XPT2046_CONV_VAL(out_buf[5], out_buf[6]);
    if (x == z1 && z1 == z2) {
```

```
        // something goes wrong
        return -1;
    }

    driver->_is_initialized = 1;
    return 0;
}

int   lcd_xpt2046_measure(lcd_xpt2046_driver_t   *driver,
uint8_t measure_command, int *result) {
    int err;

    if (!driver->_is_initialized) {
        return -1;
    }

    // measure resistance
    uint8_t in_buf[3] = { measure_command | XPT2046_CMD_
DEFAULT_POWER_MODE, 0x00, 0x00 };
    uint8_t out_buf[3] = { 0x00 };
    err = driver->communication_cb(driver->user_data, in_
buf, out_buf, 3);
    if (err) {
        return err;
    }

    *result = XPT2046_CONV_VAL(out_buf[1], out_buf[2]);
    return 0;
}
```

Битовые маски курсоров

Файл lab_5_icons.h

```
//-----------------------------------------------------
----------//
// Binary icon data                                    //
//                                                     //
// Data represents 2d array where:                     //
// 0 means empty pixel, 1 - icon pixel                 //
//-----------------------------------------------------
----------//

#include "stdint.h"


// icon "cursor_1"
#define CURSOR_1_IMAGE_W 16
#define CURSOR_1_IMAGE_H 16
extern  const  uint8_t  cursor_1_image[CURSOR_1_IMAGE_H]
[CURSOR_1_IMAGE_W];

// icon "cursor_10"
#define CURSOR_10_IMAGE_W 16
#define CURSOR_10_IMAGE_H 16
extern  const  uint8_t  cursor_10_image[CURSOR_10_IMAGE_H]
[CURSOR_10_IMAGE_W];

// icon "cursor_2"
#define CURSOR_2_IMAGE_W 16
#define CURSOR_2_IMAGE_H 16
extern  const  uint8_t  cursor_2_image[CURSOR_2_IMAGE_H]
[CURSOR_2_IMAGE_W];

// icon "cursor_3"
#define CURSOR_3_IMAGE_W 16
#define CURSOR_3_IMAGE_H 16
extern  const  uint8_t  cursor_3_image[CURSOR_3_IMAGE_H]
[CURSOR_3_IMAGE_W];

// icon "cursor_4"
#define CURSOR_4_IMAGE_W 16
#define CURSOR_4_IMAGE_H 16
```

```
extern  const  uint8_t  cursor_4_image[CURSOR_4_IMAGE_H]
[CURSOR_4_IMAGE_W];

// icon "cursor_5"
#define CURSOR_5_IMAGE_W 16
#define CURSOR_5_IMAGE_H 16
extern  const  uint8_t  cursor_5_image[CURSOR_5_IMAGE_H]
[CURSOR_5_IMAGE_W];

// icon "cursor_6"
#define CURSOR_6_IMAGE_W 16
#define CURSOR_6_IMAGE_H 16
extern  const  uint8_t  cursor_6_image[CURSOR_6_IMAGE_H]
[CURSOR_6_IMAGE_W];

// icon "cursor_7"
#define CURSOR_7_IMAGE_W 16
#define CURSOR_7_IMAGE_H 16
extern  const  uint8_t  cursor_7_image[CURSOR_7_IMAGE_H]
[CURSOR_7_IMAGE_W];

// icon "cursor_8"
#define CURSOR_8_IMAGE_W 16
#define CURSOR_8_IMAGE_H 16
extern  const  uint8_t  cursor_8_image[CURSOR_8_IMAGE_H]
[CURSOR_8_IMAGE_W];

// icon "cursor_9"
#define CURSOR_9_IMAGE_W 16
#define CURSOR_9_IMAGE_H 16
extern  const  uint8_t  cursor_9_image[CURSOR_9_IMAGE_H]
[CURSOR_9_IMAGE_W];
```

Файл lab_5_icons.c

```
//-----------------------------------------------------
----------//
// Binary icon data  content                          //
//-----------------------------------------------------
----------//

#include "lab_5_icons.h"

// icon "cursor_1"
const uint8_t cursor_1_image[CURSOR_1_IMAGE_H][CURSOR_1_
IMAGE_W] = {
```

```
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0},
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
    {1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1},
    {1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1},
    {1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1},
    {0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0},
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
};

// icon "cursor_10"
const     uint8_t     cursor_10_image[CURSOR_10_IMAGE_H]
[CURSOR_10_IMAGE_W] = {
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0},
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0},
    {1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1},
    {1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1},
    {1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1},
    {0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0},
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0},
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
};

// icon "cursor_2"
const uint8_t cursor_2_image[CURSOR_2_IMAGE_H][CURSOR_2_
IMAGE_W] = {
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0},
```

```
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0},
    {1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1},
    {0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0},
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0},
    {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
};

// icon "cursor_3"
const uint8_t cursor_3_image[CURSOR_3_IMAGE_H][CURSOR_3_
IMAGE_W] = {
    {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
    {1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0},
    {0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0},
    {0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0},
    {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0},
    {1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
    {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0}
};

// icon "cursor_4"
const uint8_t cursor_4_image[CURSOR_4_IMAGE_H][CURSOR_4_
IMAGE_W] = {
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0}
};

// icon "cursor_5"
const uint8_t cursor_5_image[CURSOR_5_IMAGE_H][CURSOR_5_
IMAGE_W] = {
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1},
    {1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1},
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
    {1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1},
    {1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1}
};

// icon "cursor_6"
const uint8_t cursor_6_image[CURSOR_6_IMAGE_H][CURSOR_6_
IMAGE_W] = {
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1}
};

// icon "cursor_7"
const uint8_t cursor_7_image[CURSOR_7_IMAGE_H][CURSOR_7_
IMAGE_W] = {
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0}
};

// icon "cursor_8"
const uint8_t cursor_8_image[CURSOR_8_IMAGE_H][CURSOR_8_
IMAGE_W] = {
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
    {1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
```

```
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
};

// icon "cursor_9"
const uint8_t cursor_9_image[CURSOR_9_IMAGE_H][CURSOR_9_
IMAGE_W] = {
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
};
```

Пример конфигурации LVGL

Файл lv_conf.h

```
/**
 * Minimal LVGL 8.2 configuration.
 *
 * Note: if some sections/options are omitted, then default
values from "lv_conf_internal.h" are used.
 */

#ifndef LV_CONF_H
#define LV_CONF_H

#include <stdint.h>

/*====================
   COLOR SETTINGS
 *====================*/

/*Color depth: 1 (1 byte per pixel), 8 (RGB332), 16
(RGB565), 32 (ARGB8888)*/
#define LV_COLOR_DEPTH 16

/*=========================
   MEMORY SETTINGS
 *=========================*/

/*1: use custom or standard malloc/free, 0: use the built-
in `lv_mem_alloc()` and `lv_mem_free()`*/
#define LV_MEM_CUSTOM 1

/*====================
   HAL SETTINGS
 *====================*/

/*Default display refresh period. LVG will redraw changed
areas with this period time*/
#define LV_DISP_DEF_REFR_PERIOD 30      /*[ms]*/

/*Input device read period in milliseconds*/
#define LV_INDEV_DEF_READ_PERIOD 30     /*[ms]*/

/* Use STM32 HAL tick tick source. */
#define LV_TICK_CUSTOM 1
#if LV_TICK_CUSTOM == 1
        #define  LV_TICK_CUSTOM_INCLUDE  "stm32f3xx_hal.h"
/*Header for the system time function*/
```

```
     #define LV_TICK_CUSTOM_SYS_TIME_EXPR (HAL_GetTick())
/*Expression evaluating to current system time in ms*/
#endif   /*LV_TICK_CUSTOM*/

/*=======================
 * FEATURE CONFIGURATION
 *=======================*/

/*-------------
 * Drawing
 *-----------*/

/*Enable complex draw engine.
 *Required to draw shadow, gradient, rounded corners, circles,
arc, skew lines, image transformations or any masks*/
#define LV_DRAW_COMPLEX 1

/*-------------
 * Logging
 *-----------*/

/*Enable the log module*/
#define LV_USE_LOG 1
/*1: Print the log with 'printf';
*0: User need to register a callback with `lv_log_register_
print_cb()`*/
#define LV_LOG_PRINTF 1

/*-------------
 * Asserts
 *-----------*/

/*Enable asserts if an operation is failed or an invalid
data is found.
 *If LV_USE_LOG is enabled an error message will be printed
on failure*/
#define LV_USE_ASSERT_NULL          1   /*Check if the
parameter is NULL. (Very fast, recommended)*/
#define LV_USE_ASSERT_MALLOC        1    /*Checks is
the memory is successfully allocated or no. (Very fast,
recommended)*/
#define LV_USE_ASSERT_STYLE       1  /*Check if the styles
are properly initialized. (Very fast, recommended)*/
#define LV_USE_ASSERT_MEM_INTEGRITY 0      /*Check the
integrity of `lv_mem` after critical operations. (Slow)*/
#define LV_USE_ASSERT_OBJ         0  /*Check the object's
type and existence (e.g. not deleted). (Slow)*/
```

```
/*-------------
 * Others
 *-----------*/

/*Change the built in (v)snprintf functions*/
#define LV_SPRINTF_CUSTOM 1

#define LV_SPRINTF_INCLUDE <stdio.h>
#define lv_snprintf  snprintf
#define lv_vsnprintf vsnprintf

/* Custom data of user objects */
#define LV_USE_USER_DATA 1

/*==================
 *   FONT USAGE
 *==================*/

/*Montserrat fonts with ASCII range and some symbols using
bpp = 4
 *https://fonts.google.com/specimen/Montserrat*/
#define LV_FONT_MONTSERRAT_8  0
#define LV_FONT_MONTSERRAT_10 0
#define LV_FONT_MONTSERRAT_12 0
#define LV_FONT_MONTSERRAT_14 1
#define LV_FONT_MONTSERRAT_16 0
#define LV_FONT_MONTSERRAT_18 0
#define LV_FONT_MONTSERRAT_20 0
#define LV_FONT_MONTSERRAT_22 0
#define LV_FONT_MONTSERRAT_24 0
#define LV_FONT_MONTSERRAT_26 0
#define LV_FONT_MONTSERRAT_28 0
#define LV_FONT_MONTSERRAT_30 0
#define LV_FONT_MONTSERRAT_32 0
#define LV_FONT_MONTSERRAT_34 0
#define LV_FONT_MONTSERRAT_36 0
#define LV_FONT_MONTSERRAT_38 0
#define LV_FONT_MONTSERRAT_40 0
#define LV_FONT_MONTSERRAT_42 0
#define LV_FONT_MONTSERRAT_44 0
#define LV_FONT_MONTSERRAT_46 0
#define LV_FONT_MONTSERRAT_48 0

/*Always set a default font*/
#define LV_FONT_DEFAULT &lv_font_montserrat_14
```

```
/*==================
 *  TEXT SETTINGS
 *==================*/

/*==================
 *  WIDGET USAGE
 *================*/

/*Documentation of the widgets: https://docs.lvgl.io/
latest/en/html/widgets/index.html*/

#define LV_USE_ARC        1

#define LV_USE_ANIMIMG    1

#define LV_USE_BAR        1

#define LV_USE_BTN        1

#define LV_USE_BTNMATRIX  1

#define LV_USE_CANVAS     1

#define LV_USE_CHECKBOX   1

#define LV_USE_DROPDOWN   1   /*Requires: lv_label*/

#define LV_USE_IMG        1   /*Requires: lv_label*/

#define LV_USE_LABEL      1

#define LV_USE_LINE       1

#define LV_USE_ROLLER     1   /*Requires: lv_label*/

#define LV_USE_SLIDER     1   /*Requires: lv_bar*/

#define LV_USE_SWITCH     1

#define LV_USE_TEXTAREA   1   /*Requires: lv_label*/

#define LV_USE_TABLE      1

/*==================
 * EXTRA COMPONENTS
 *==================*/

/*-----------
 * Widgets
 *----------*/
#define LV_USE_CALENDAR   1

#define LV_USE_CHART      1
```

```
#define LV_USE_COLORWHEEL 1
#define LV_USE_IMGBTN     1
#define LV_USE_KEYBOARD   1
#define LV_USE_LED        1
#define LV_USE_LIST       1
#define LV_USE_MENU       1
#define LV_USE_METER      1
#define LV_USE_MSGBOX     1
#define LV_USE_SPINBOX    1
#define LV_USE_SPINNER    1
#define LV_USE_TABVIEW    1
#define LV_USE_TILEVIEW   1
#define LV_USE_WIN        1
#define LV_USE_SPAN       1

/*-----------
 * Themes
 *----------*/

/*A simple, impressive and very complete theme*/
#define LV_USE_THEME_DEFAULT 1
#define LV_THEME_DEFAULT_DARK 0
#define LV_THEME_DEFAULT_GROW 1

/*A very simple theme that is a good starting point for a
custom theme*/
#define LV_USE_THEME_BASIC 0

/*A theme designed for monochrome displays*/
#define LV_USE_THEME_MONO 0

/*-----------
 * Layouts
 *----------*/

/*A layout similar to Flexbox in CSS.*/
#define LV_USE_FLEX 1

/*A layout similar to Grid in CSS.*/
#define LV_USE_GRID 1

/*--END OF LV_CONF_H--*/

#endif /*LV_CONF_H*/
```

СОДЕРЖАНИЕ

Учебное издание

Ключарёв Александр Анатольевич,

Кочин Константин Александрович,

Фоменкова Анастасия Алексеевна

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ STM32

Учебное пособие

ISBN: 978-5-8088-1829-3

9 785808 818293

Редактор *Е. В. Лазарева*

Компьютерная верстка *И. А. Мосиной*

Подписано к печати 26.04.2023. Формат 60 × 84 1/16.

Усл. печ. л. 11,4. Уч.-изд. л. 11,7.

Тираж 50 экз. Заказ № 120.

Редакционно-издательский центр ГУАП

190000, г. Санкт-Петербург, ул. Большая Морская, д. 67, лит. А