Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Микропроцессорные системы»
на тему:

Генератор QR-кода

Студент ИУ6-72Б (Группа) _____ (Подпись, дата) А.И. Толстиков (И.О. Фамилия)

Руководитель _____ (Подпись, дата) И.Б. Трамов (И.О. Фамилия)

2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ6

А.В. Пролетарский

« 2 » сентября 2023 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Микропроцессорные системы

Студент группы ИУ6-72Б

Толстиков Артём Иванович

Тема курсовой работы: Генератор QR-кода

Направленность курсовой работы: учебная

Источник тематики (кафедра, предприятие, НИР): кафедра

График выполнения работы: 25% – 4 нед., 50% – 8 нед., 75% – 12 нед., 100% – 16 нед.

Техническое задание:

Разработать МК-систему для генерации QR-кода. Обеспечить ввод данных с ПЭВМ и телефона. Предусмотреть вывод QR-кода на ПЭВМ и ЖК-дисплей.

Предусмотреть возможность хранения последних 10 QR-кодов с возможностью переключения между ними с помощью управляющих кнопок и/или команд с ПЭВМ.

Информацию об ошибке выводить на ЖК-дисплей.

Выбрать наиболее оптимальный вариант МК. Выбор обосновать.

Разработать схему, алгоритмы и программу. Отладить проект в симуляторе или на макете. Оценить потребляемую мощность. Описать принципы и технологию программирования используемого микроконтроллера.

Оформление курсовой работы:

1. Расчетно-пояснительная записка на 30-35 листах формата А4.
2. Перечень графического материала:
   а) схема электрическая функциональная;
   б) схема электрическая принципиальная.

Дата выдачи задания: «4» сентября 2023 г. Дата защиты: «20» декабря 2023 г.

Руководитель курсовой работы 04.09.2023 (Подпись, дата) И.Б. Трамов (И.О.Фамилия)

Студент 04.09.2023 (Подпись, дата) А.И. Толстиков (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах; один выдается студенту, второй хранится на кафедре

РЕФЕРАТ

РПЗ 71 страницы, 5 таблицы, 22 рисунка, 12 источников, 2 приложения МИКРОКОНТРОЛЛЕР STM32, USART, SPI, ЖИДКОКРИСТАЛЛИЧЕСКИЙ ДИСПЛЕЙ, QR-КОД, ГЕНЕРАЦИЯ

Объектом исследования является устройство, позволяющее генерировать QR-код из подаваемой на вход символьной строки.

Цель работы – разработка устройства ограниченной сложности, создание модели этого устройства и необходимой технической документации к нему.

Результатом работы является комплект конструкторской документации для изготовления устройства, исходные коды программ для программирования памяти микроконтроллера и модель устройства в среде разработки Proteus 8.

Устройство должно выполнять следующие функции:

− Выбор ячейки памяти для хранения символьной строки;

− Хранение введенной символьной строки в выбранной ячейке памяти;

− Генерация QR-кода;

− Автоматическое масштабирование QR-кода под размер дисплея;

− Вывод сгенерированного QR-кода на дисплей;

− Вывод расшифровки QR-кода на дисплее на ПЭВМ.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящем отчете о НИР применяют следующие сокращения и обозначения:

МПС – микропроцессорные системы

ЖК-дисплей – жидкокристаллический дисплей

МК – микроконтроллер

ТЗ – техническое задание

TFT – тонко-пленочный транзистор(thin-film transistor)

USART – последовательный универсальный асинхронный приемопередатчик (universal asynchronous receiver/transmitter)

SPI – интерфейс для взаимодействия МК с внешними устройствами (serial peripheral interface)

ВВЕДЕНИЕ

В данной работе разрабатывается генератор QR-кода с жидкокристаллическим дисплеем для вывода сгенерированных кодов, а также хранящий последние 10 сгенерированных кодов.

В процессе выполнения работы был произведен анализ технического задания и предметной области, создана концепция устройства, создана концепция устройства, разработаны электрические схемы, построен алгоритм и управляющая программа для МК, выполнено интерактивное моделирование устройства.

Актуальность разрабатываемого устройства генерации QR-кода заключается в том, что может быть использовано в точках розничной торговли, складских помещениях или выставках произведений искусства, в качестве динамически подстраивающихся описаний. Например, с помощью серии таких устройств, как разработанное в рамках данной работы, можно оснастить условное складское помещение дисплеями, содержащими описание товара, хранящегося в соответствующей ячейке.

1 Конструкторская часть

1.1 Анализ технического задания

Согласно техническому заданию, необходимо разработать МК-систему для генерации QR-кода. Система должна обеспечить ввод данных с ПЭВМ и телефона, а также предусматривать вывод QR-кода на ПЭВМ и ЖК-дисплей. Исходя из данного абзаца ТЗ, система должна содержать в себе сам микроконтроллер, ЖК-дисплей, а также порт или дополнительное устройство для взаимодействия с ПЭВМ и телефоном. Чтобы минимизировать данную схему, порт или устройство должны поддерживать интеграцию и телефона, и ПЭВМ.

Также, техническим заданием требуется предусмотреть возможность хранения последних 10 QR-кодов с возможностью переключения между ними с помощью управляющих кнопок. То есть, МК-система должна содержать в себе несколько кнопок, которые управляют выбором отображаемого в данный момент QR-кода.

Также, в соответствии с ТЗ, в случае возникновения ошибки, сообщение о сбое должно быть отображено на дисплее.

1.2 Анализ предметной области

Под понятием “QR-код” скрывается большое множество различных кодировок, которые человек может использовать для представления различной информации. ТЗ к данной работе не предъявляет никаких требований к размеру QR-кода, содержащейся служебной информации, вида хранимой информации. Поэтому необходимо разработать эти требования самостоятельно.

QR-код может содержать 3 основных вида информации: буквенный текст, буквенно-цифровые данные и битовые данные. Последнее сразу можно не рассматривать – QR-коды используют для удобной передачи информации на устройство, зачастую содержащей текст. Однако текст, состоящий исключительно из букв, достаточно сложно читать. Буквенно-цифровое кодирование данных содержит не только буквы или цифры, но и специальные

символы. Это решает сразу 2 проблемы – вопрос использования QR-кодов для передачи человеко-читаемого текста и для передачи ссылок на web-ресурсы. Это два основных направления использования QR-кодов в настоящее время.

Для QR-кодов существует 4 уровня коррекции: L (допустимо максимум 7% повреждений), M (15%), Q (25%) и H (30%). Чаще всего используется уровень M. Мы также будем использовать его – он представляет собой золотую середину между объемом информации и ее защищенностью.

Существует 40 версий qr кода, которые отличаются только размером кода. Использование какой-либо версии обосновано размером кодируемой информации. Лимит к кодируемой информации будет зависеть от размера используемого ЖК-дисплея.

1.3 Анализ структуры системы

Для того, чтобы анализировать составные модули разрабатываемой микропроцессорной системы, необходимо понять, какую структуру будут собой создавать компоненты. Для этого разработана структурная схема.

Управление системой будет осуществляться с блока управления, состоящего из нескольких кнопок. Сигналы с блока будут передаваться на микроконтроллер, вызывая переключение хранимых в памяти QR-кодов. Также по кнопке будет начата отправка QR-кода на подключенную ПЭВМ.

ПЭВМ или телефон будут передавать на микроконтроллер сообщение, которое последний будет обрабатывать, выдавая в результате QR-код, который будет запоминаться в память микроконтроллера, заменяя собой самый ранний из 10 хранимых QR-кодов.

Микроконтроллер обрабатывает вводимые данные и сигналы и отправляет на ЖК-дисплей или ПЭВМ сгенерированный QR-код.

ЖК-дисплей должен отображать номер выбранного QR-кода и сам символ.

Разработанная структурная схема генератора QR-кодов представлена на рисунке 1.

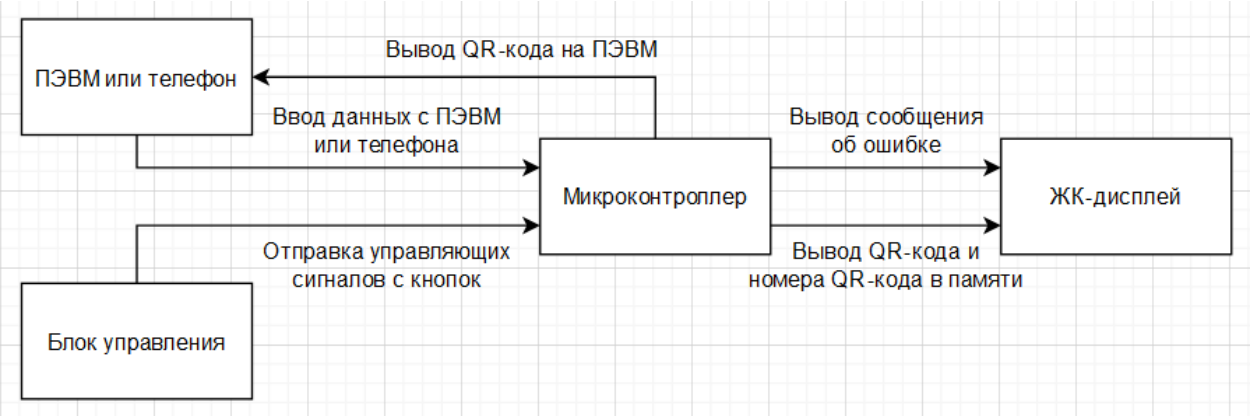

Рисунок 1 – Структурная схема

Далее будет рассмотрен подробнее функционал каждого блока

1.3.1 Функционал блока управления

Блок управления состоит из нескольких кнопок:

1) кнопка выбора следующего QR-кода
2) кнопка выбора предыдущего QR-кода
3) кнопка отправки QR-кода на ПЭВМ
4) кнопка перезагрузки системы

Кнопка выбора следующего и предыдущего QR-кода выбирает, какой из сохраненных в памяти QR-кодов будет сейчас отображаться на ЖК-дисплее. Если кнопка, обозначенная в пункте 1 перечисления будет нажата при отображении самого последнего сохраненного в памяти QR-кода, то переход будет осуществлен на самый первый участок памяти. Аналогично произойдет переход на самый последний участок памяти при нажатии кнопки, обозначенной в пункте 2 перечисления. Если в выбранной ячейке памяти не хранится QR-кода, то будет выведено сообщение с номером ячейки и с информации о том, что данная ячейка памяти не содержит QR-кода в данный момент.

При нажатии на кнопку отправки, на ПЭВМ будет отправлена строка с исходным текстом, которая будет являться расшифровкой к отображаемому на ЖК-дисплее QR-коду. Если в данной ячейке памяти не хранится QR-кода, то при нажатии на эту кнопку ничего не произойдет.

При нажатии на кнопку перезагрузки будет перезагружен микроконтроллер, после чего при его первичной инициализации будет перезагружен и ЖК-дисплей.

1.3.2 Параметры используемого дисплея

Вывод на ЖК-дисплей подразумевает не только вывод текстовой информации. Большую часть экрана будет занимать QR-код, который задается набором пикселей. Для передачи растрового изображения не подойдет символьно-числовой дисплей.

В связи с этим решено использовать TFT-дисплей. Это экран, созданный на основе жидких кристаллов и под управлением тонкопленочных транзисторов. Подобные дисплеи задействуются в самых разных устройствах для отображения текста и изображений. Их можно встретить в компьютерах, ноутбуках, смартфонах, электронных книгах и других девайсах [1] Конструкция TFT ЖК-дисплея представлена следующим образом. Слои накладываются друг на друга, поэтому его часто сравнивают с сэндвичем. Его основная часть состоит из: тонкопленочного транзистора, цветного фильтра и слоя жидких кристаллов. Дисплей обеспечивает более высокую яркость и четкость изображения по сравнению с более старыми типами LCD.

TFT ЖК-дисплей, будучи достаточно бюджетным вариантом четкого и специфического вывода, является идеальным вариантом для поставленной задачи.

1.3.3 Выбор микроконтроллера

Семейство микроконтроллеров STM32 отлично взаимодействует с TFT ЖК-дисплеями благодаря своей мощности, богатым периферийным возможностям и широким набором интерфейсов и библиотек взаимодействия с различными представителями данного вида дисплеем. Микроконтроллеры STM32 поддерживают высокие частоты тактирования, что важно для быстрого обновления графического содержимого на дисплее. Богатство периферийных блоков, таких как таймеры и SPI/I2C, обеспечивает удобные средства для управления TFT-дисплеями, а наличие встроенной графической

библиотеки или поддержки сторонних библиотек упрощает программирование и визуализацию данных.

В то же время в связи с низкими функциональными требованиями устройства были рассмотрены в первую очередь бюджетные варианты микроконтроллеров.

Серии микроконтроллеров семейства STM32 делятся на 4 типа:

1. Высокопроизводительные – F2, F4, F7, H7
2. Широкого использования – F0, G0, F1, F3, G4
3. Сверхнизкого потребления – L0, L1, L4+, L5
4. Беспроводные – WB, WL

Из всех типов наиболее подходящий – тип широкого использования, так как в задаче используется несколько внешних устройств, использующих различные интерфейсы. МК широкого использования обладают достаточной производительностью и являются достаточно экономным вариантом, что делает их лучшим вариантом, а использование высокопроизводительных или сверхнизкого потребления скорее является неоправданным излишком. Беспроводные МК в данной задаче рассматривать не целесообразно.

Среди серий МК широкого использования выбрана золотая середина – серия F1. МК данной серии обладают достаточным характеристиками, необходимыми для решения поставленной задачи, и в то же время не являются столь дорогими, как серии F3, G4, цены на которые выше примерно в два раза.

Среди МК серии STM32F1 были выделены четыре наиболее

популярные модели: STM32F103C8T6, STM32F103RB, STM32F103VET6, STM32F103ZET6. Их основные характеристики приведены в таблице 1.

Таблица 1 – Сравнительная таблица микроконтроллеров серии STM32F1

Микроконтроллер	Flash (Кб)	RAM (Кб)	Частота (МГц)	Число пинов ввода-вывода
STM32F103C8T6	64	20	72	37
STM32F103RB	128	20	72	50
STM32F103VET6	512	64	72	80
STM32F103ZET6	512	64	72	112

Указанные в таблице микроконтроллеры отсортированы сверху вниз в порядке роста памятей и количества пинов и, соответственно, цены. Слишком большое количество памяти или количества пинов МК для данного устройства не нужны, 64 Кбайт Flash памяти и 20 Кбайт RAM памяти будет предельно достаточно. По этой причине выбирать что-то кроме STM32F103C8T6 не выгодно, потому что это будет более затратно.

Еще одной важной причиной для выбора МК STM32F103C8T6 является факт того, что данный МК был изучен во время курса МПС в 6 семестре. Более того мною была проведена физическая лабораторная работа с отладочной платой с этим МК.

Таким образом, выбранный для устройства МК – STM32F103C8T6. С данным микроконтроллером имеется реальный опыт работ, он обладает более чем достаточными характеристиками для решения поставленной задачи и в то же время является бюджетным вариантом.

1.4 Проектирование функциональной схемы

1.4.1 Микроконтроллер STM32F103C8T6

В настоящей работе выбран микроконтроллер STM32F103C8T6.

STM32 – семейство 32-битных микроконтроллеров, производимых фирмой STMicroelectronics. Контроллеры этого семейства основаны на различных ядрах ARM, в зависимости от серии. Микроконтроллеры серии STM32F1 имеют ядро ARM Cortex-M3, сопряженное системой внутренних шин с периферийными устройствами [2].

В составе МК имеются:

− процессорное ядро ARM Cortex-M3 с максимальной тактовой частотой 72 МГц и контроллером прерываний NVIC (Nested vectored interrupt controller);

− Flash-память объемом 64 Кбайт;

− SRAM объемом 20 Кбайт;

− 16-разрядные порты ввода-вывода (GPIO – General-purpose inputs/outputs);

− 16-разрядные таймеры (TIM);

− аналого-цифровые преобразователи (ADC);

− контроллер прямого доступа к памяти (DMA);

− контроллер внешних прерываний (EXTI – External interrupt/event controller);

− интерфейсы для связи с другими устройствами: UART, SPI, I2C, CAN, USB.

Применительно к поставленной задаче семейство микроконтроллеров STM32 предоставляет высокую производительность и широкие возможности в области взаимодействия с различными видами дисплеев.

Структурная схема микроконтроллера представлена на рисунке 2, а его УГО представлено на рисунке 3.

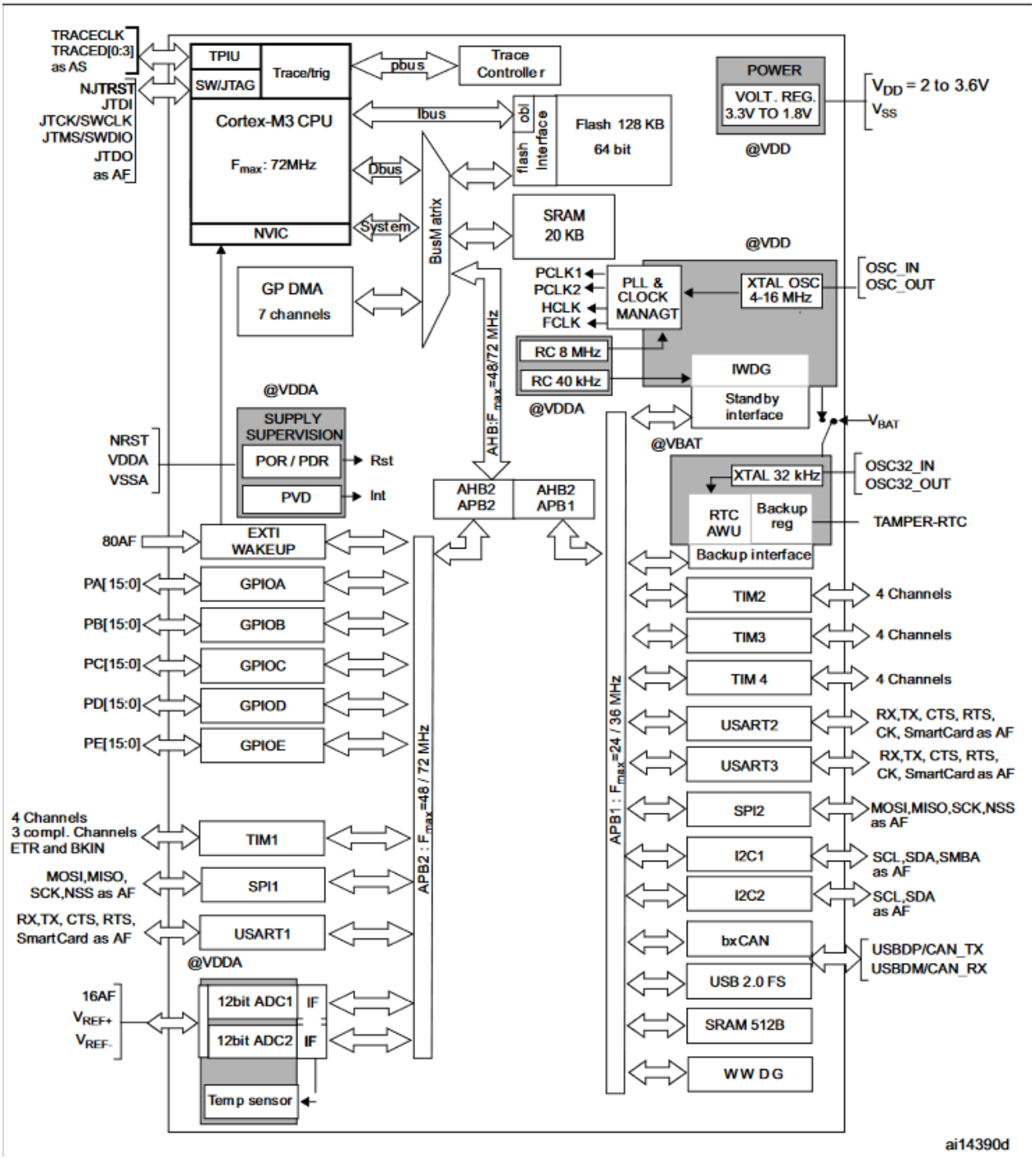

Рисунок 2 – Структурная схема микроконтроллера STM32F103C8T6

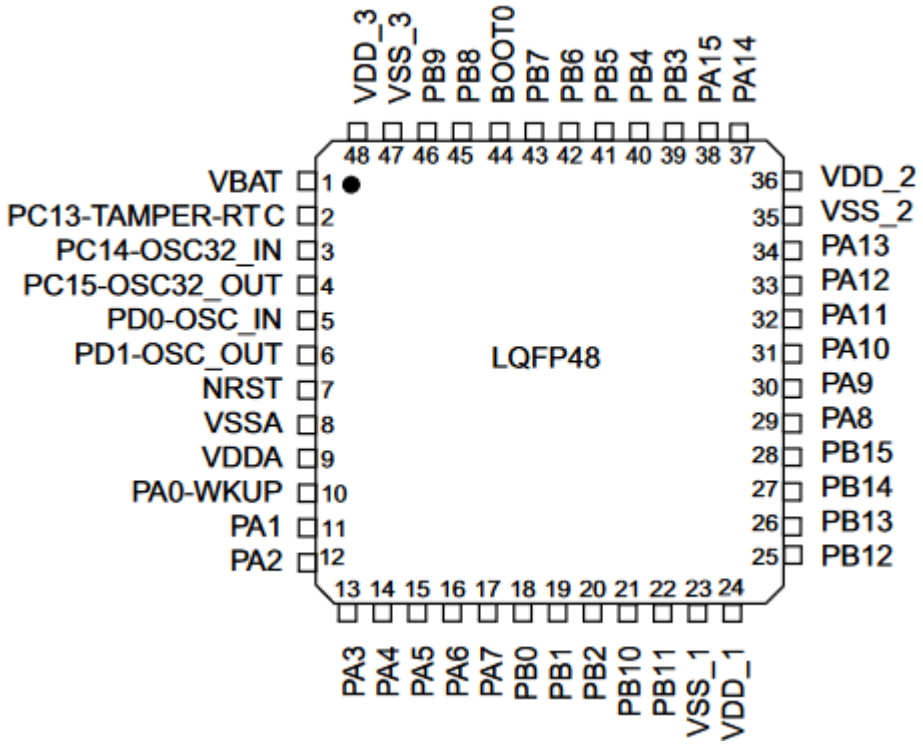

Рисунок 3 – УГО микроконтроллера STM32F103C8T6

1.4.2 Используемые элементы и интерфейсы

Для работы устройства, генерирующего QR-код, использованы не все элементы архитектуры МК STM32F103C8T6. Среди использованных элементов и интерфейсов:

− Порты A, B – использованные пины и их назначение описано в пункте 1.4.3.

− Указатель стека – играет важную роль в организации стека, используемого для управления вызовами подпрограмм. Указатель стека используется для сохранения адреса возврата и регистров при вызове функций. Это обеспечивает корректный возврат из функций и поддерживает структуру вызовов функций.

− Регистры общего назначения – предназначены для хранения операндов арифметико-логических операций, а также адресов или отдельных компонентов адресов ячеек памяти.

− АЛУ – выполняет арифметические и логические операции, обеспечивает выполнение базовых математических операций и манипуляций с битами.

− Память SRAM – статическая память МК, хранящая объявленные переменные.

− Память Flash – память МК, хранящая загруженную в него программу.

− Программный счетчик – указывает на следующую по испольнению команду.

− Регистры команд – содержит исполняемую в настоящий момент команду(или следующую), то есть команду, адресуемую счетчиком команд.

− Декодер – выделяет код операции и операнды команды и далее вызывает микропрограмму, исполняющую данную команду.

− Сигналы управления – нужны для синхронизации обработки данных.

− Логика программирования – устанавливает логику того, как будет вшита программа в МК.

− Генератор – генератор тактовых импульсов. Необходим для синхронизации работы МК.

− Управление синхронизацией и сбросом – обрабатывает тактовые сигналы и отвечает за сброс состояния МК.

− Прерывания – обрабатывает внешние прерывания и прерывания периферийных устройств МК (таймеров, портов и т.д.). В устройстве используются прерывания с портов для обработки нажатия кнопок и прерывания UART.

Прерывания имеют настраиваемый приоритет. Внешние прерывания могут быть настроены от 20 источников, 16 из которых представляют объединенные по номерам 2 линии GPIO. Например, линии 0 всех портов (PA0, PB0, PC0…) объединены в один источник внешнего прерывания.

− UART (Universal Asynchronous Receiver Transmitter) – интерфейс, при помощи которого происходит передача данных в МК из ПЭВМ.

− SPI (Serial Peripheral Interface) – интерфейс для связи МК с другими внешними устройствами. В устройстве используется для прошивки МК и вывода данных на жидкокристаллический дисплей.

1.4.3 Используемые порты

МК STM32F103C8T6 содержит в себе пять портов – A, B, C, D, E. В устройстве используются порты A и B.

Порт A используется для:

− PA3 – отправка данных или команд на дисплей;

− PA5 – тактовый сигнал SCK для интерфейса SPI для LCD-дисплея;

− PA7 – MOSI-пин для интерфейса SPI для LCD-дисплея;

− PA9 – отправка данных по UART на ПЭВМ;

− PA10 – прием данных по UART от ПЭВМ;

− PA13 – SWDIO-пин для программатора ST-LINK V2;

− PA14 – SWCLK-пин для программатора ST-LINK V2.

Порт B используется для:

− PB6 – пин с внешним прерыванием по нажатии кнопки отправки строки на ПЭВМ;

− PB7 – пин с внешним прерыванием по нажатии кнопки перехода к следующей ячейке памяти;

− PB8 – пин с внешним прерыванием по нажатии кнопки перехода к предыдущей ячейке памяти.

1.4.4 Организация памяти МК

На рисунке 4 представлена карта памяти МК STM32F103C8T6 [3].

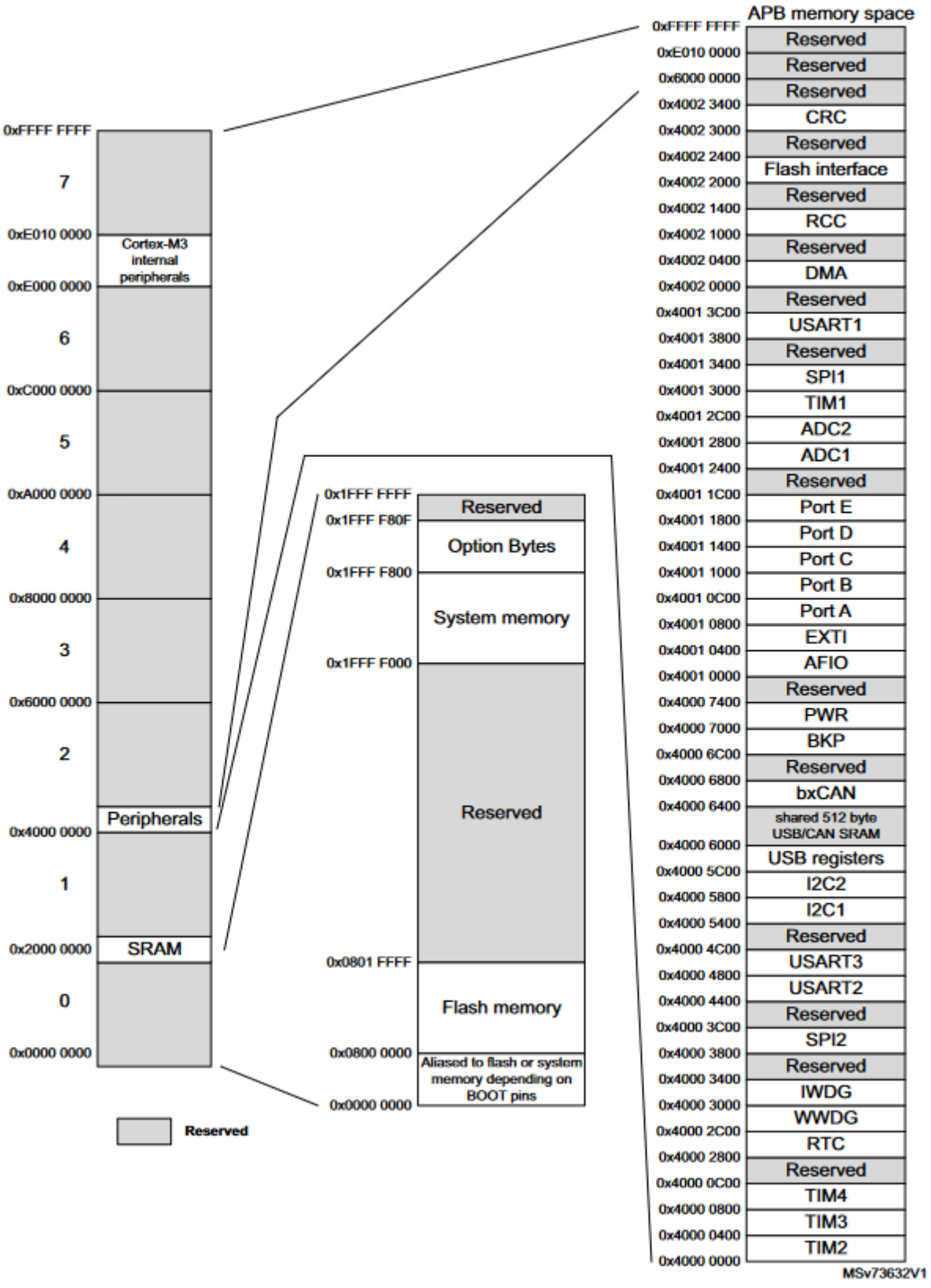

Рисунок 4 – Организация памяти МК STM32F103C8T6

1.4.5 Расчет экрана и выбор дисплея

Для выбора дисплея в первую очередь необходимо рассчитать достаточный размер экрана. Так как был выбран TFT ЖК-дисплей то его размер (или разрешение) измеряется в пикселях. Так как текст должен быть достаточно читаем, ибо это важно при использовании системы пользователем, размер шрифта 5x3 пикселей является не подходящим вариантом – цифры не всегда разборчивы и тяжело ищутся. Размер шрифта 10x7 пикселей на символ является подходящим по читаемости и занимает минимально возможное место, оставаясь читаемым.

В связи с необходимостью моделирования работы системы в Proteus 8, необходимо рассмотреть наибольшие из существующих и адекватно функционирующих в данной системе ЖК-дисплеев. Так как на экран будет выводиться QR-код, то необходимо оценивать меньшую сторону экрана, так как QR-код имеет квадратную форму. Также должно быть место для текстовой информации, чтобы выводить другую информацию о выводимом QR-коде.

Итого имеются несколько подходящих вариантов: ILI9341, ST7735. Однако ILI9341 имеет слишком большое разрешение и слишком маленький размер пикселя. Основные характеристики ST7735, указанные в таблице 2.

Таблица 2 – Сравнительный анализ контроллеров дисплеев

Критерий	**ST7735**
Цветовая глубина	16 бит
Интерфейсы	SPI, I2C
Размер	1,8 Дюймовый

ST7735 – это однокристальный контроллер/драйвер для графического TFT ЖК-дисплея. Он может выполнять операции чтения/записи данных в

оперативной памяти дисплея без внешнего тактового сигнала для минимизации энергопотребления [4]. Схема дисплея показана на рисунке 5.

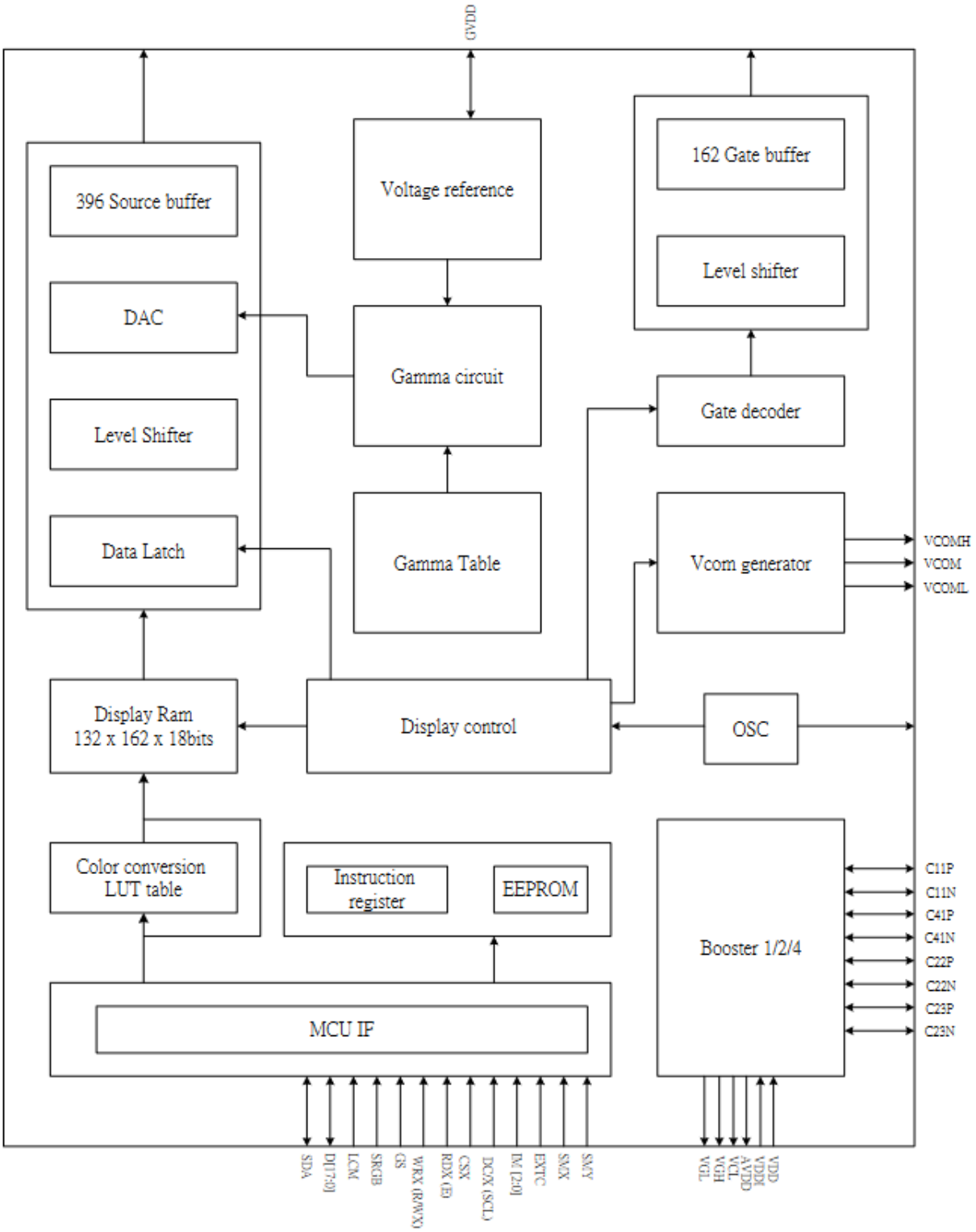

Рисунок 5 – Схема дисплея ST7735

Основные пины взаимодействия дисплея:

− IM2 – выбор шины параллельного и последовательного интерфейса – при установке в 1 параллельный, при 0 – последовательный.

− IM1, IM0 – выбор типа параллельного интерфейса. В таблице 3 представлены возможные значения.

Таблица 3 – Типы параллельного интерфейса

IM1	**IM0**	**Параллельный интерфейс**
0	0	8 бит
0	1	16 бит
1	0	9 бит
1	1	18 бит

− SPI4W – 0 при трех линиях SPI, 1 при четырех линиях.

− RESX – сигнал перезапустит устройство и нужно его использовать для правильной инициализации устройства.

− CSX – пин выбора микроконтроллера устройства, работает по низкому сигналу.

− D/CX – пин выбора данных или команды на интерфейсе микроконтроллера дисплея. При 1 – данные или параметры, при 0 – команды. При SPI используется как SCL.

− RDX – дает возможность считать при включенном параллельном интерфейсе в микроконтроллере.

− WRX(D/CX) – дает возможность писать при включенном параллельном интерфейсе в микроконтроллере. При 4 линейном SPI используется как D/CX.

− D[17:0] – используются как шины отправки данных параллельного интерфейса микроконтроллера. D0 это сигнал входа/выхода при последовательном интерфейсе. При последовательном интерфейсе сигналы D[17:1] не используются.

− TE – пин вывода для синхронизации микроконтроллера с частотой устройства, активируемый программно командой перезапуска.

− OSC – контролирующий пин вывода внутреннего тактового генератора, активируемый программно командой перезапуска.

Пины выбора режима дисплея:

− EXTC – использование режима расширенных команд. При 0 используются обычные команды, при 1 расширенный набор команд NVM.

− GM1, GM0 – пины выбора разрешения. При обоих пинах в состоянии 1 – разрешение 132x162, при обоих 0 – 128x160.

− SRGB – пин настройки порядка фильтров цветов RGB. В устройстве не важен.

− SMX/SMY – пины, отвечающие за направление вывода на дисплей. По умолчанию началом экрана считается левый верхний угол.

− LCM – пин выбора типа кристалла, белый при 0 и черный при 1.

− GS – пин изменения гаммы. Оставлен по умолчанию.

− TESEL – пин используется для изменения вывода TE сигнала. Работает только при GM[1:0] = 00 и при 0 выводит номер строки из 162, при 1 номер строки из 160.

Соединение МК с дисплеем происходит посредством 14-ти контактного шлейфа для коммутации. Его распиновка и соединение с МК представлены на рисунке 6.

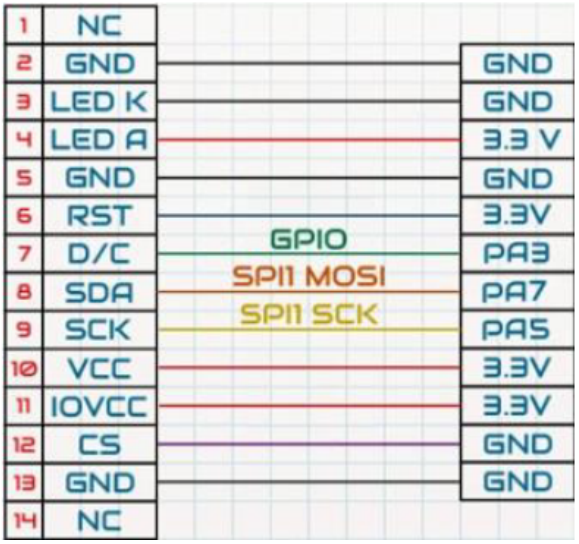

Рисунок 6 – Соединение шлейфа для коммутации и МК

Пояснение распиновки шлейфа:

1. NC – не подключен (Not Connected);
2. GND – земля;
3. LED K – катод светодиодов подсветки;
4. LED A – анод светодиодов подсветки;
5. GND – земля;
6. RST – сигнал сброса;
7. D/C – сигнал выбора передачи данных или команд;
8. SDA – SPI MOSI;
9. SCK – SPI SCK;
10. VCC – напряжение питания;
11. IOVCC – источник питания внутренних интерфейсов;
12. CS – выбор чипа (Chip Select);
13. GND – земля;
14. NC – не подключен (Not Connected).

Для дисплея ток потребления – 40мА, напряжение питания – 3.3В

1.4.6 Использование интерфейса SPI для ЖК-дисплея

Интерфейс SPI (Serial Peripheral Interface – последовательный периферийный интерфейс) является высокоскоростным синхронным последовательным интерфейсом. Он обеспечивает обмен данными между микроконтроллером и различными периферийными устройствами, такими как АЦП, ЦАП, цифровые потенциометры, карты памяти, другие микросхемы и микроконтроллеры [5].

МК STM32F103C8T6 содержит два интерфейса SPI, которые обеспечивают передачу данных на частотах до 18 МГц. Один интерфейс SPI расположен на низкоскоростной шине APB1, работающей на тактовой частоте до 36 МГц, а другой – на высокоскоростной шине периферийных устройств APB2, которая работает на тактовой частоте до 72 МГц. Для увеличения эффективности передачи данных в микроконтроллере выделено

два канала DMA. По интерфейсу SPI можно связать ведущий микроконтроллер с одним или несколькими ведомыми устройствами.

Одно из устройств должно быть определено ведущим (мастер), а остальные – ведомыми (подчинённые). Связь между устройствами осуществляется с помощью следующих линий связи:

− MOSI – выход данных для ведущего или вход данных для ведомого устройства;

− MISO – вход данных для ведущего или выход данных для ведомого устройства;

− SCK – сигнал общей синхронизации интерфейса.

Существует четыре режима передачи данных по SPI, которые определяются полярностью и фазой тактового сигнала. Отличие режимов заключается в том, что активным уровнем сигнала синхронизации может быть единичный или нулевой потенциал, а запись данных может производиться по фронту или спаду импульса данного синхросигнала. Эти режимы интерфейса обозначаются цифрами 0, 1, 2 и 3. На рисунке 7 представлена диаграмма всех перечисленных режимов работы интерфейса SPI.

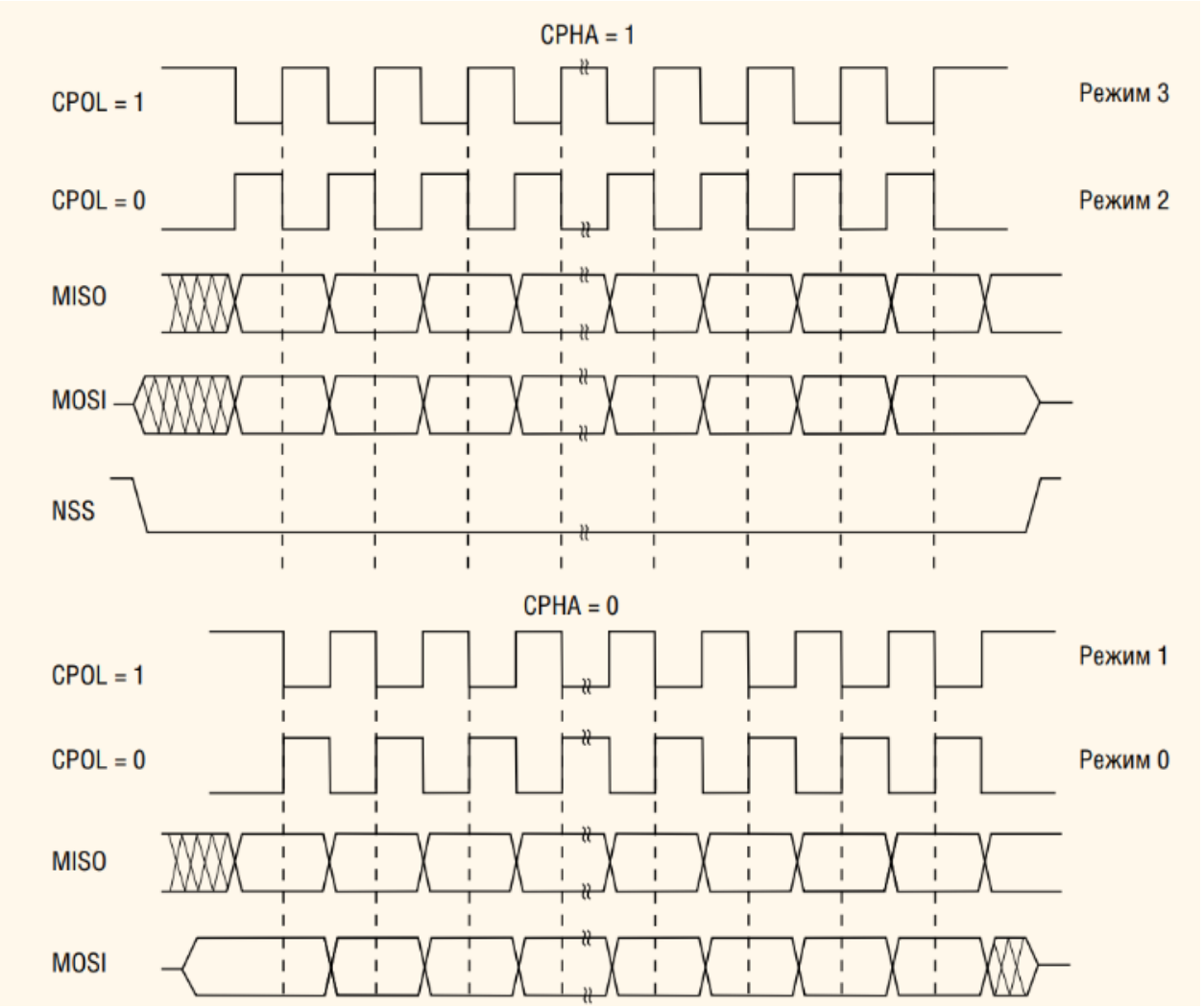

Рисунок 7 – Диаграмма режимов работы интерфейса SPI

Микроконтроллер позволяет для каждого интерфейса SPI задать полярность и фазу тактового сигнала, определяя тем самым режим его работы. Кроме того, для микроконтроллера можно установить формат передачи данных 8-разрядными или 16-разрядными словами и определить порядок передачи данных – старшим или младшим битом вперёд. Это позволяет микроконтроллеру с помощью обоих интерфейсов SPI обмениваться информацией с любыми другими SPI-устройствами.

Рассмотрим внутреннюю архитектуру SPI микроконтроллера STM32, которая представлена на рисунке 8:

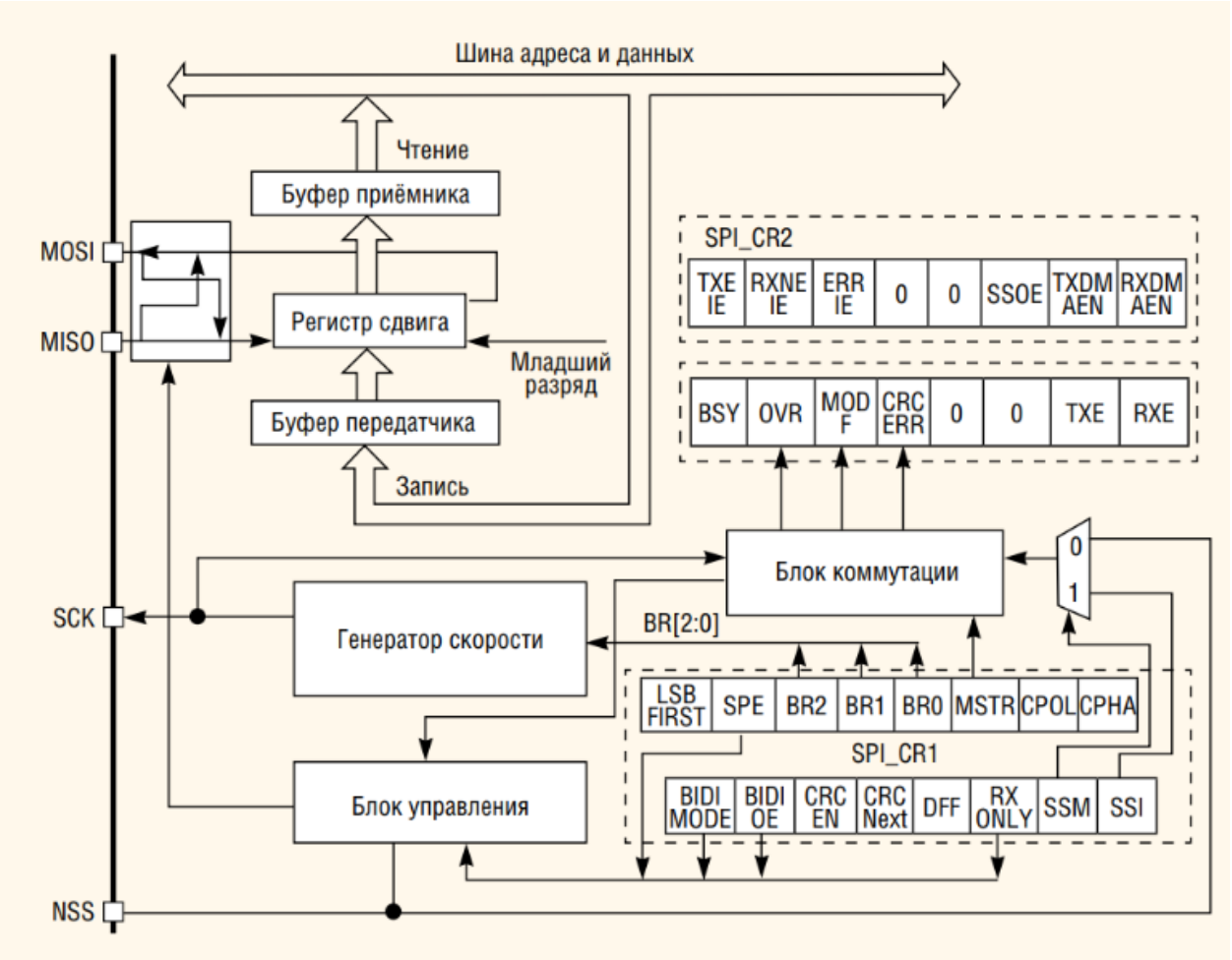

Рисунок 8 – Архитектура SPI МК семейства STM32

Регистр сдвига представляет собой основной регистр, через который передаются и принимаются данные. Если интерфейс SPI работает в режиме ведущего устройства, то вход этого сдвигового регистра соединён с выводом MISO, а выход – с выводом MOSI.

В режиме ведомого устройства происходит обратное переключение, которое регулирует блок управления. Для передачи данных их необходимо записать в регистр передатчика. Принятые данные читаются из регистра приёмника.

Для программы существует один регистр с именем SPI_DR. При чтении этого регистра происходит обращение к регистру приёмника, а при записи – к регистру передатчика. Скорость обмена по SPI определяет блок генератора скорости, который задаёт частоту следования тактовых импульсов. Для этого предназначены разряды BR0, BR1 и BR2 регистра SPI_CR1. Три разряда

предполагают наличие восьми значений скорости. Таким образом, скорость обмена данными по интерфейсу SPI для микроконтроллера STM32 с тактовой частотой N МГц может меняться в диапазоне [N / 2; N /256] Мбит/с.

Для работы с интерфейсом SPI в микроконтроллере STM32 имеются специальные регистры. Формат этих регистров с названием входящих в них разрядов представлен на рисунке 9.

Сдвиг	Регистр	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 × 00	SPI_CR1	Резерв																BIDMODE	BIDIOE	CRCEN	CRCNEXT	DFF	RXONLY	SSM	SSI	LSBFIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 × 04	SPI_CR2	Резерв																								TXEE	RXNEIE	ERRIE	Резерв		SSDE	TXDMAEN	RXDMAEN
	Исх. значение																									0	0	0			0	0	0
0 × 08	SPI_SR	Резерв																								BSY	OVR	MODE	CRCERR	Резерв		TXE	RXNE
	Исх. значение																									0	0	0	0			1	0
0 × 0C	SPI_DR	Резерв																DR[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 × 10	SPI_CRCPR	Резерв																CRCPOLY[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0 × 14	SPI_RXCRCR	Резерв																RXCRC[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 × 18	SPI_TXCRRCR	Резерв																TXCRC[15:0]															
	Исх. значение																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 9 – Формат регистров SPI

Регистры:

− SPI_CR1 – первый управляющий регистр;

− SPI_CR2 – второй управляющий регистр;

− SPI_SR – регистр статуса;

− SPI_DR – регистр данных;

− SPI_CRCPR – регистр, содержащий полином для вычисления CRC;

− SPI_RXCRCR – регистр, содержащий CRC принятых данных;

26

− SPI_TXCRCR – регистр, содержащий CRC передаваемых данных.

Некоторые из этих регистров используются для работы в режиме I2S.

Регистр SPI_CR1 является первым управляющим регистром интерфейса SPI. Он имеет следующие управляющие разряды:

0. CPHA задаёт фазу тактового сигнала;

1. CPOL устанавливает полярность тактового сигнала;

2. MSTR назначает режим работы интерфейса (0 – ведомый, 1 – ведущий);

5…3. 0BR [2:0] задают скорость обмена (000 – $f_{PCLK}/2$, 001 – $f_{PCLK}/4$, 010 – $f_{PCLK}/8$, 011 – $f_{PCLK}/16$, 100 – $f_{PCLK}/32$, 101 – $f_{PCLK}/64$, 110 – $f_{PCLK}/128$, 111 – $f_{PCLK}/256$);

6. SPE управляет интерфейсом (0 – отключает, 1 – включает);

7. LSBFIRST задаёт направление передачи (0 – младшим разрядом вперёд, 1 –старшим разрядом вперёд);

8. SSI определяет значение NSS при SSM=1;

9. SSM выбирает источник сигнала NSS (0 – с внешнего вывода, 1 – программно от разряда SSI);

10. RX ONLY совместно с битом BIDIMODE определяет направление передачи в однонаправленном режиме;

11. DFF определяет формат данных (0–8 бит, 1–16 бит);

12. CRCNEXT управляет передачей кода CRC (0 – данные, 1 – CRC);

13. CRCEN регулирует аппаратное вычисление CRC (0 – запрещено, 1 – разрешено). Для корректной операции этот бит должен записываться только при отключённом интерфейсе SPI, когда SPE = 0;

14. BIDIOE совместно с битом BIDIMODE управляет двунаправленным режимом работы интерфейса (0 – приём, 1 – передача);

15. BIDIMODE управляет двунаправленным режимом работы интерфейса (0 – двухпроводный однонаправленный режим, 1 – однопроводной двунаправленный режим).

SPI_CR2 является вторым управляющим регистром интерфейса SPI и имеет следующие разряды, которые управляют:

0. RXDMAEN – запросом DMA для приёмника (0 – запрещает, 1 – разрешает);

1. TXDMAEN – запросом DMA для передатчика (0 – запрещает, 1 – разрешает);

2. SSOE – сигналом NSS в режиме мастера (0 – запрещает, 1 – разрешает);

4…3. – зарезервированы;

5. ERRIE – прерыванием в случае ошибки (0–запрещает 1–разрешает);

6. RXNEIE – прерыванием приёма данных (0–запрещает 1–разрешает);

7. TXEIE – управляет прерыванием передачи данных (0–запрещает 1–разрешает);

15…8. – зарезервированы.

Регистр статуса SPI_SR включает в себя следующие разряды:

0. RXNE устанавливается, если в буфере приёмника есть принятые данные;

1. TXE – устанавливается, если буфер передатчика пуст и готов принять новые данные;

2, 3. – зарезервированы;

4. CRCERR устанавливается при ошибке CRC при приёме данных;

5. MODF устанавливается, когда в режиме мастера к сигналу NSS прикладывается низкий потенциал;

6. OVR – флаг переполнения, устанавливается при приёме новых данных, если предыдущие не были прочитаны;

7. BSY – флаг занятости, устанавливается, если интерфейс занят обменом данных или буфер данных передатчика не пустой.

Регистр данных SPI_DR состоит из 16 разрядов данных. В этот регистр данные записываются для передачи и читаются из него при приёме.

Для SPI заданы следующие настройки: формат посылки данных – 8 бит, делитель задан равным 8, поэтому скорость передачи данных 4 Мбит/с для частоты МК = 32 МГц. Интерфейс в МК работает в режиме Master.

Настройка SPI1 представлена в листинге 1.

Листинг 1 – Инициализация SPI1 в программе

```
hspi1.Instance = SPI1;
hspi1.Init.Mode = SPI_MODE_MASTER;
hspi1.Init.Direction = SPI_DIRECTION_2LINES;
hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
hspi1.Init.NSS = SPI_NSS_SOFT;
hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
hspi1.Init.CRCPolynomial = 10;
if (HAL_SPI_Init(&hspi1) != HAL_OK)
{
Error_Handler();
}
```

1.4.7 Отправка данных на ПЭВМ

Приём данных от ПЭВМ происходит через драйвер MAX232. MAX232 – интегральная схема, преобразующая сигналы последовательного порта RS-232 в цифровые сигналы.

RS-232 – стандарт физического уровня для синхронного и асинхронного интерфейса (USART и UART). Обеспечивает передачу данных и некоторых специальных сигналов между терминалом и устройством приема. Сигнал, поступающий от интерфейса RS-232, через преобразователь передается в микроконтроллер на вход RxD.

К внешнему устройству MAX232 подключен через разъем DB-9. На схеме условное обозначение – XP2.

УГО MAX232 показано на рисунке 10. Назначение пинов описано в таблице 4.

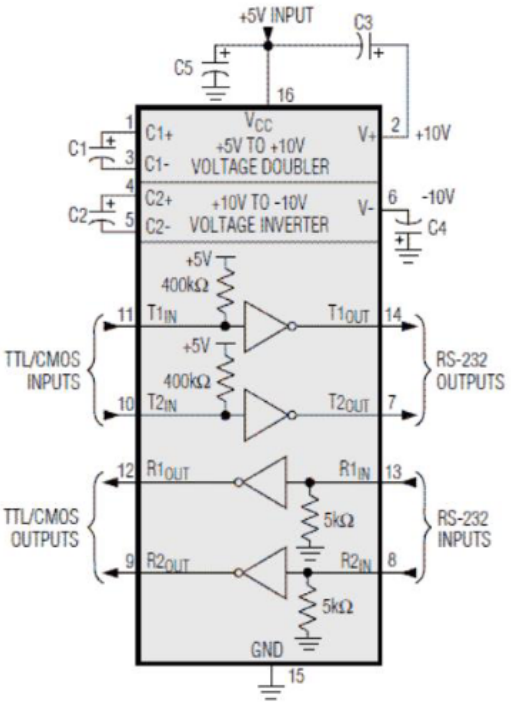

Рисунок 10 – УГО MAX232

Таблица 4 – Назначение пинов MAX232

Номер	**Имя**	**Тип**	**Описание**
1	C1+	–	Положительный вывод C1 для подключения конденсатора
2	VS+	O	Выход положительного заряда для накопительного конденсатора
3	C1-	–	Отрицательный вывод C1 для подключения конденсатора
4	C2+	–	Положительный вывод C2 для подключения конденсатора
5	C2-	–	Отрицательный вывод C2 для подключения конденсатора
6	VS-	O	Выход отрицательного заряда для накопительного конденсатора
7, 14	T2OUT, T1OUT	O	Вывод данных по линии RS232
8, 13	R2IN, R1IN	I	Ввод данных по линии RS232

Продолжение таблицы 4

9, 12	R2OUT, R1OUT	O	Вывод логических данных
10, 11	T2IN, T1IN	I	Ввод логических данных
15	GND	–	Земля
16	Vcc	–	Напряжение питания, подключение к внешнему источнику питания 5 В

Когда микросхема MAX232 получает на вход логический "0" от внешнего устройства, она преобразует его в напряжение от +5 до +15В, а когда получает логическую "1" - преобразует её в напряжение от -5 до -15В, и по тому же принципу выполняет обратные преобразования от RS-232 к внешнему устройству.

1.4.8 Использование интерфейса USART для работы с ПЭВМ

Интерфейс USART (Universal Synchronous Asynchronous Receiver Transmitter) в микроконтроллерах STM32 представляет собой универсальный последовательный интерфейс, который может работать в режиме синхронной или асинхронной передачи данных. Он обеспечивает возможность обмена данными между микроконтроллером и другими устройствами, такими как датчики, модули связи и периферийные устройства.

USART в STM32 поддерживает передачу данных через одну линию для приема (RX) и одну для передачи (TX). Он также может работать в полудуплексном режиме, когда одна линия используется для передачи и приема данных.

USART может настраиваться на разные скорости передачи данных(бодрейты), количество бит данных, контроль четности, стоповые биты и другие параметры через специальные регистры микроконтроллера. Это обеспечивает гибкость в настройке передачи данных в соответствии с требованиями конкретного приложения.

В разрабатываемой системе USART используется в асинхронном режиме для ввода и вывода текста на виртуальный, который выступает в роли ПЭВМ. Рассмотрим настройку USART для устройства.

В устройстве USART используется в асинхронном режиме, контроль сигнала CTS/RTS отключен, baud rate – 115200 бит/с, длина каждой посылки – 8 бит, включая бит четности, контроль четности отключен, используется один стоп-бит, оверсемплинг в режиме 16-семплирования. Кроме того, включены прерывания для USART.

Оверсемплинг в USART относится к технике, используемой для приема данных в асинхронном режиме. Эта техника помогает улучшить точность синхронизации битов данных, особенно при работе с высокими скоростями передачи данных.

Оверсемплинг подразумевает выбор частоты сэмплирования (число раз, которое система измеряет состояние входного сигнала за определенный промежуток времени) значительно выше, чем минимально необходимая частота для корректного считывания данных.

В USART для асинхронной передачи, оверсемплинг обычно используется для более точного определения момента прихода каждого бита данных. К примеру, в режиме 16-семплирования (16x oversampling), каждый бит данных будет сэмплироваться 16 раз за период передачи, что улучшает точность считывания данных и помогает бороться с потерей или искажением сигнала в условиях шумов или неполадок в канале связи.

Эта техника позволяет повысить устойчивость и надежность приема данных по USART, особенно при работе на высоких скоростях передачи данных или в условиях, где возможны помехи или искажения сигнала.

Всего существует 7 регистров, связанных с настройкой и работой USART:

– USART_SR (Status register),

– USART_DR (Data register),

– USART_BRR (Baud rate register),

– USART_CR1 (Control register 1),

– USART_CR2 (Control register 2),

– USART_CR3 (Control register 3),

– USART_GTPR (Guard time and prescaler register).

Ниже будут описаны все регистры кроме неиспользованных регистров для настройки [6]. Формат регистров представлен на рисунке 11.

Table 182. USART register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**USART_SR**	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	**USART_DR**	Reserved																							DR[8:0]								
	Reset value																								0	0	0	0	0	0	0	0	0
0x08	**USART_BRR**	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**USART_CR1**	Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**USART_CR2**	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**USART_CR3**	Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																						0	0	0	0	0	0	0	0	0	0	0
0x18	**USART_GTPR**	Reserved																GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 11 – Формат регистров USART

Начнем с настройки USART. Для этого используются control-регистры и регистр управления скоростью передачи.

Регистр USART_CR1:

− UE: USART enable - включить USART (включается установкой бита в состояние 1).

− M: Word length - длина слова, задаёт количество бит данных в одном фрейме. Бит не должен модифицироваться в процессе обмена данными (это касается как передачи, так и приёма). 0 - 1 старт-бит, 8 бит данных, n стоп-битов; 1 - 1 старт-бит, 9 бит данных, n стоп-битов. Примечание. Бит чётности считается битом данных.

− WAKE: Wakeup method - метод пробуждения USART. 0 - "линия свободна" (Idle line); 1- адресная метка.

− PCE: Parity control enable - включить аппаратный контроль чётности (генерация бита чётности при передаче данных и проверка в принимаемых данных).

− PS: Parity selection - выбор метода контроля чётности. Выбор происходит после завершения передачи/приёма текущего байта. 0 - контроль на чётность; 1 - контроль на нечётность.

− PEIE: PE interrupt enable - разрешение прерывания от PE. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART_SR.PE==1.

− TXEIE: TXE interrupt enable - разрешение прерывания от TXE. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART_SR.TXE==1.

− TCIE: Transmission complete interrupt enable - разрешение прерывания после завершения передачи. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART_SR.TC==1.

− RXNEIE: RXNE interrupt enable - разрешение прерывания от RXNE. 0: прерывание запрещено; 1: генерируется прерывание от USART, когда USART_SR.ORE==1 или USART_SR.RXNE==1.

− IDLEIE: IDLE interrupt enable - разрешение прерывания при обнаружении, что "линия свободна" (Idle line). 0: прерывание запрещено; 1: генерируется прерывание от USART, когда USART_SR.IDLE==1.

− TE: Transmitter enable - включить передатчик USART (включается установкой бита в 1).

− RE: Receiver enable - включить приёмник USART (включается установкой бита в 1). После установки бита, приёмник начинает поиск старт-бита во входном сигнале.

− RWU: Receiver wakeup - переводит USART в тихий режим. Этот бит устанавливается и сбрасывается программно, а также может сбрасываться аппаратно при обнаружении пробуждающей последовательности.

− SBK: Send break - отправить Break посылку. Бит может быть установлен и сброшен программно. Его необходимо программно установить в 1 для формирования Break посылки, он будет сброшен аппаратно во время формирования stop-бита в Break фрейме. 0: Break-символ не передаётся; 1: Break-символ будет передан.

Теперь опишем регистр BRR, с помощью которого контролируется скорость передачи данных через USART.

− DIV_Mantissa[11:0]: mantissa of USARTDIV - целая часть коэффициента деления делителя частоты.

− DIV_Fraction[3:0]: fraction of USARTDIV - дробная часть коэффициента деления. В режиме с OVER8==1 в битовом поле DIV_Fraction[3:0] старший бит [3] не используется и должен быть сброшен.

С помощью регистра USART_BRR задаётся скорость передачи - одновременно как для приёмника USART, так и для передатчика. На рисунке 12 представлена схема, показывающая, как именно высчитывается скорость передачи.

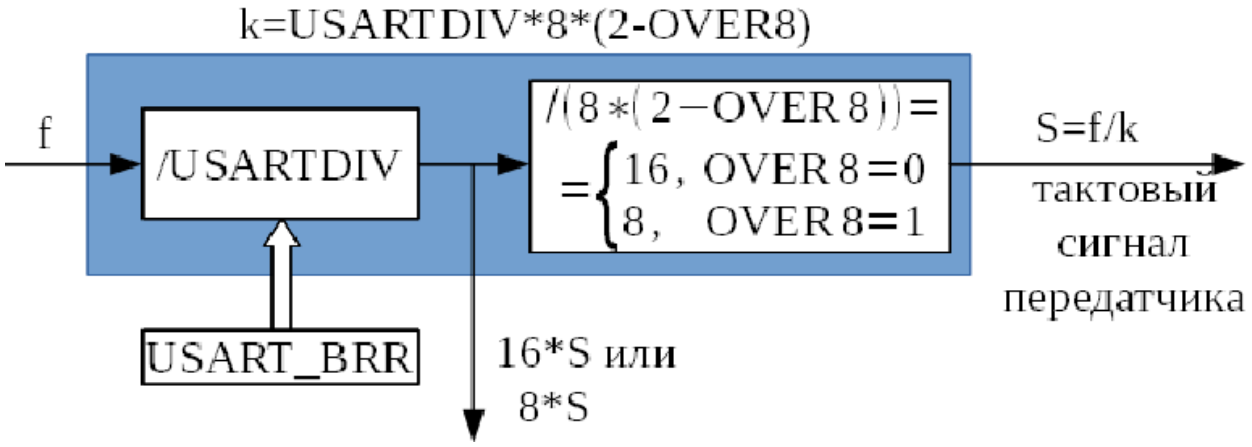

Рисунок 12 – Вычисление скорости приема и передачи

В данной системе было принято решение использовать baud rate = 115200, поэтому был выставлен USART_BRR = 69. Проверим: 8000000 / 69 = 115942 ~ 115200.

Далее рассмотрим USART_DR – регистр, через который передаются непосредственно данные.

DR[8:0]: Data value - регистр данных. Содержит полученный или передаваемый символ, в зависимости от того, производится чтение из него или запись в регистр. Регистр выполняет двойную функцию за счёт того, что он является составным, он объединяет в себе два регистра: один для передачи (TDR) и один для приёма (RDR). TDR обеспечивает загрузку данных в выходной сдвигающий регистр, сдвигающий регистр преобразует загруженное в него слово в последовательную форму. Получаемые в последовательной форме данные накапливаются в приёмном сдвигающем регистре, когда фрейм получен полностью, данные из сдвигающего регистра передаются в регистр RDR, который реализует параллельный интерфейс между внутренней шиной микроконтроллера и входным сдвигающим регистром.

Когда осуществляется передача данных с включённым контролем чётности (USART_CR1.PCE==1), старший бит, записываемый в регистр USART_DR (бит [6] или [7], в зависимости от выбранной длины слова, см. USART_CR1.M), не учитывается. Он замещается вычисленным битом чётности.

При получении данных с включённым контролем чётности, при чтении из USART_DR будем получать значение, содержащее полученный бит чётности.

Последний рассматриваемый регистр в USART – USART_SR (status register).

− CTS: CTS flag - флаг изменения состояния nCTS. Устанавливается аппаратно, когда происходит переключение сигнала на входе nCTS. Если

установлен бит CTSIE (USART_CR3.CTSIE==1), то при установке флага генерируется прерывание. Флаг сбрасывается программно записью 0.

− LBD: LIN break detection flag - флаг приёма посылки Break. Устанавливается аппаратно при обнаружении посылки Break на входе; если установлен бит LBDIE (USART_CR3.LBDIE==1), то генерируется прерывание. Флаг срсывается программно записью 0.

− TXE: Transmit data register empty - флаг устанавливается аппаратно, когда содержимое регистра передаваемых данных TDR пересылается в сдвигающий регистр (доступ к TDR осуществляется путём записи в регистр USART_DR). Если установлен бит TXEIE (USART_CR1.TXEIE==1), генерируется прерывание. Флаг сбрасывается путём записи в регистр USART_DR.

− TC: Transmission complete - флаг завершения передачи, устанавливается аппаратно, если передача фрейма завершена, и флаг TXE установлен (т.е. регистр передаваемых данных пуст, больше нет данных для передачи). Если USART_CR1.TCIE==1, то при установке флага генерируется прерывание. Флаг сбрасывается программно последовательностью действий: чтение регистра USART_SR, затем запись в USART_DR. Также бит может быть сброшен записью в него 0. Примечание. После сброса этот бит установлен.

− RXNE: Read data register not empty - регистр данных для чтения не пуст. Флаг устанавливается аппаратно, когда содержимое принимающегосдвигающего регистра передаётся в регистр принимаемых данных RDR. Если USART_CR1.RXNEIE==1, при этом генерируется прерывание. Флаг сбрасывается чтением из регистра USART_DR. Также бит может быть сброшен записью в него 0.

− IDLE: IDLE line detected - линия свободна. Флаг устанавливается аппаратно, если обнаружено что линия свободна. Это происходит, если получен целый фрейм единиц. При этом генерируется прерывание, если USART_CR1.IDLEIE==1. Флаг сбрасывается программно

последовательностью действий: чтение регистра USART_SR с последующим чтением из регистра USART_DR.

− ORE: Overrun error - ошибка переполнения. Флаг устанавливается аппаратно, когда слово, полученное в сдвигающей регистр готово к перемещению в регистр принимаемых данных RDR, но RXNE==1 (регистр RDR не пуст, содержит ещё не прочитанные из него принятые USART данные). Если USART_CR1.RXNEIE==1, то при установке флага генерируется исключение. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR с последующим чтением из USART_DR.

− NF: Noise detected flag - флаг устанавливается аппаратно при обнаружении шума в полученном фрейме. Сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение из регистра USART_DR.

− FE: Framing error - ошибка фрейма. Флаг устанавливается аппаратно в случае нарушения синхронизации, чрезмерного шума в линии, при обнаружении символа Break. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение из регистра USART_DR. Примечание. В отношении генерации прерывания этот флаг полностью аналогичен флагу NF.

− PE: Parity error - ошибка чётности. Флаг устанавливается аппаратно, когда в принятом фрейме обнаружена ошибка чётности (если контроль чётности включён). Если USART_CR1.PEIE==1, то генерируется прерывание. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART_SR, затем чтение либо запись регистра USART_DR. Перед сбросом флага, программа должна дождаться установки флага RXNE (регистр данных для чтения не пуст).

Настройка USART1 представлена в листинге 2

Листинг 2 – Инициализация USART1 в программе

```
huart1.Instance = USART1;
huart1.Init.BaudRate = 115200;
huart1.Init.WordLength = UART_WORDLENGTH_8B;
huart1.Init.StopBits = UART_STOPBITS_1;
huart1.Init.Parity = UART_PARITY_NONE;
huart1.Init.Mode = UART_MODE_TX_RX;
huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart1.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart1) != HAL_OK)
{
Error_Handler();
}
```

1.4.9 Программатор ST-LINK V2

После написания и тестирования кода в программе идет этап загрузки файла (с расширением elf – бинарный файл) в микроконтроллер. Это может выполнятся следующими способами [8]:

− через JTAG;

− через SWD.

Выбрана прошивка через SWD так как это простой и популярный метод, с которым уже было знакомство на практике.

Для программирования МК используется специальный программатор ST-LINK V2. ST-LINK V2 – внутрисхемный программатор-отладчик для МК STM32, поддерживает отладку по двухпроводному интерфейсу SWD (Serial Wire Debug). Подключение программатора осуществляется при помощи портов PA13 и PA14, которые выполняют роль SWDIO и SWCLK соответственно.

Он имеет следующие разъемы для подключения к МК:

− SWCLK – тактовый сигнал;

− SWDIO – для передачи данных;

− RST – сигналом на RST программатор вводит контроллер в режим программирования.

В МК передается бинарный файл с расширением “.elf” с скомпилированной программой. Происходит это следующим образом: подается команда RESET через пин NRST. Это используется для сброса микроконтроллера в состояние, готовое к прошивке. Затем через SWCLK идет тактовый сигнал, по которому идет запись программы в микроконтроллер через SWDIO. Этот процесс осуществляется с использованием специальных последовательностей битов (протокол SWD) для передачи команд и данных между ST-LINK V2 и микроконтроллером,

обеспечивая правильную последовательность и синхронизацию для успешной прошивки или отладки микроконтроллера STM32.

1.4.10 Построение функциональной схемы

На основе пунктов 1.2.1 – 1.2.9 была построена функциональная схема устройства измерения скорости чтения. Она представлена на рисунке 13.

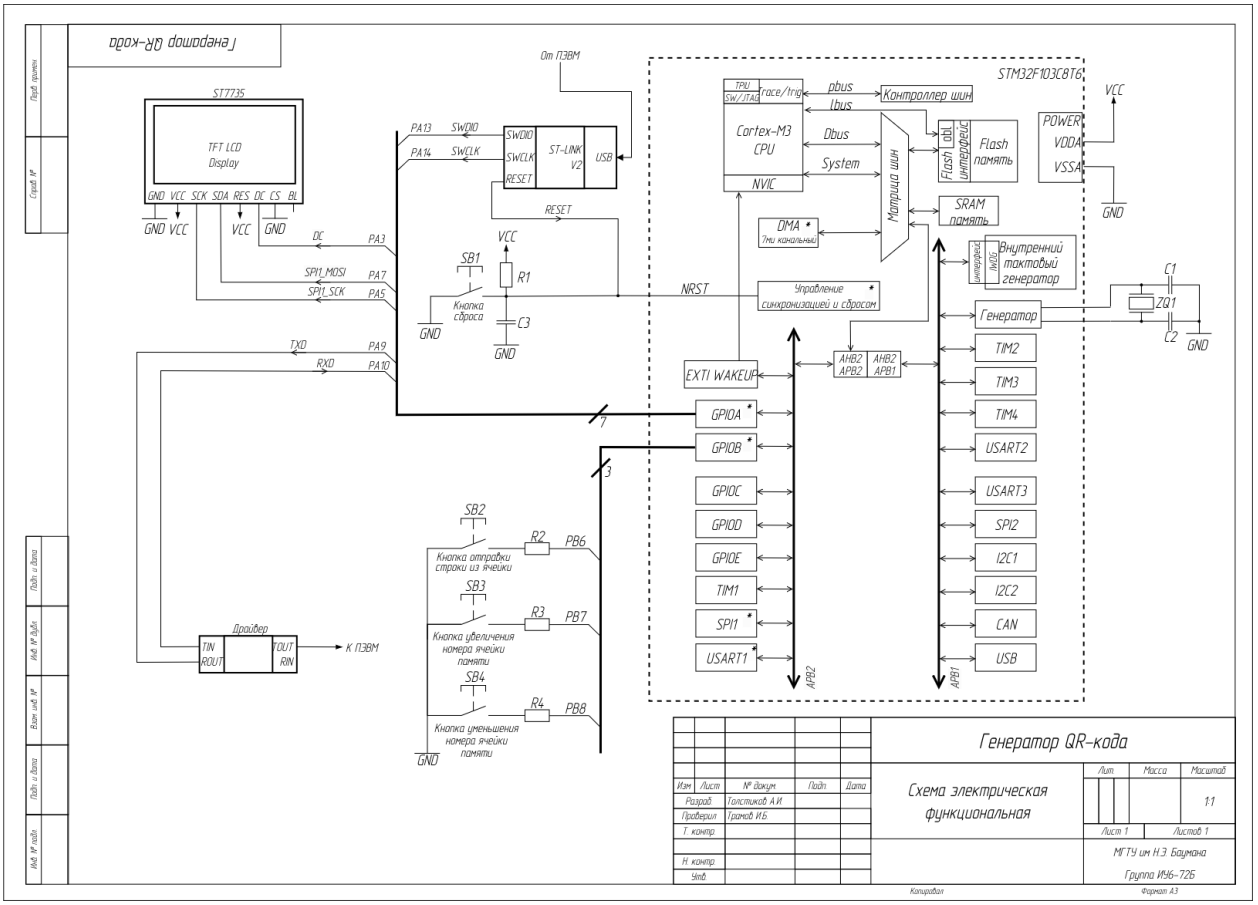

Рисунок 13 – Функциональная схема устройства

1.5 Проектирование принципиальной схемы

1.5.1 Расчет потребляемой мощности

Потребляемая мощность – это мощность, потребляемая интегральной схемой, которая работает в заданном режиме соответствующего источника питания.

Чтобы рассчитать суммарную мощность, рассчитаем мощность каждого элемента. На все микросхемы подается напряжение +3.3В. Мощность, потребляемая один устройством, в статическом режиме, рассчитывается формулой: P = U * I, где U – напряжение питания (В), I – ток потребления микросхемы (мА).

Также в схеме присутствуют резисторы CF-100. Мощность для резисторов рассчитывается по формуле: P = U2 / R, где R – сопротивление резистора, U – напряжение, проходящее через резистор.

Расчет потребляемого напряжения для каждой микросхемы представлен в таблице 5.

Таблица 5 – Расчет потребляемого напряжения

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Кол-во	Суммарная потребляемая мощность, мВт
STM32F103C8T6	100	330	1	330
MAX232	10	33	1	33
ST7735	40	132	1	132

$P_{суммарная} = P_{STM32F103C8T6} + P_{MAX232} + P_{ST7735} = 330 + 33 + 132 + 6 = 495$ мВт

Суммарная потребляемая мощность системы равна 495мВт = 0,5 Вт.

1.5.2 Построение принципиальной схемы

На основе построенной функциональной схемы и выбранных микросхем построена принципиальная схема устройства измерения скорости чтения, представленная на рисунке 14.

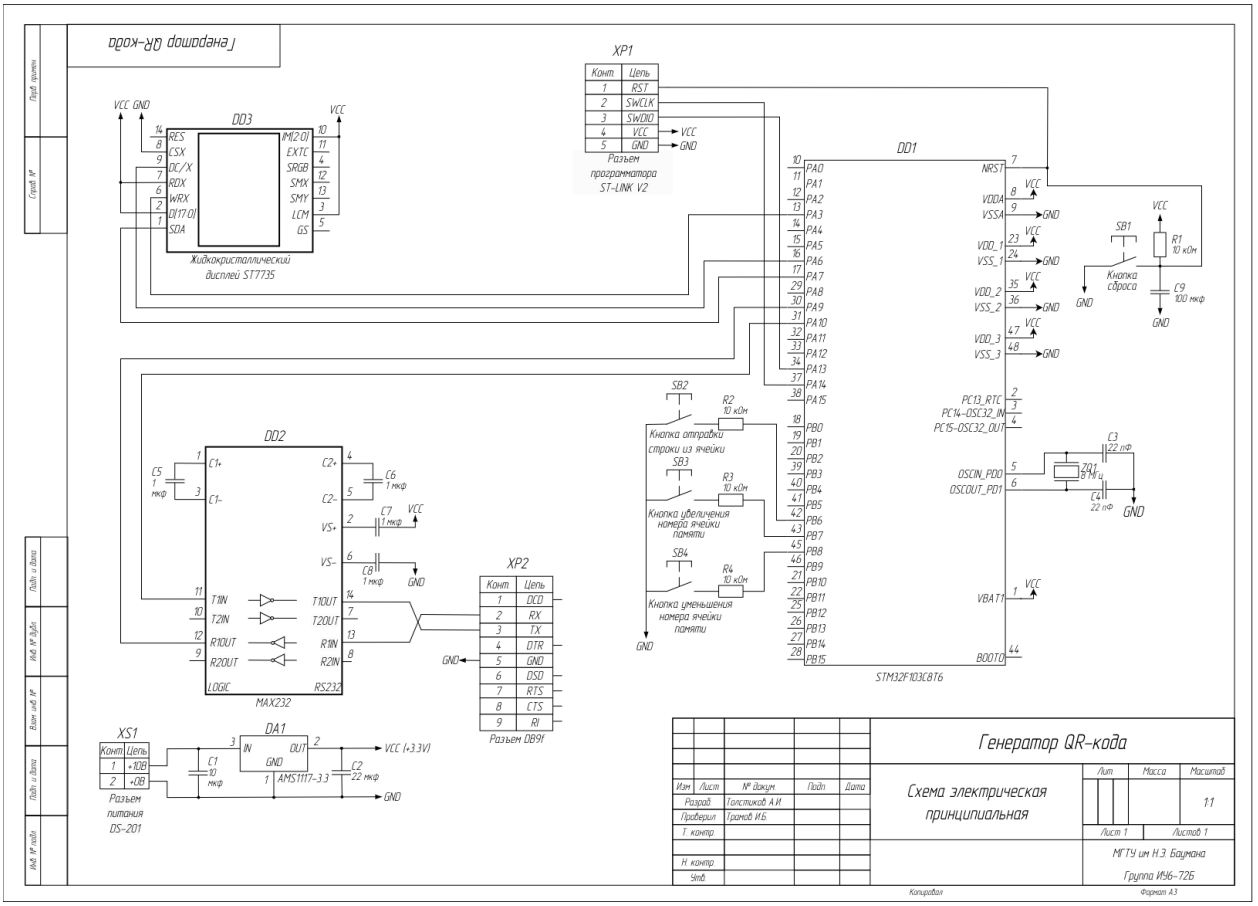

Рисунок 14 – Принципиальная схема устройства

1.6 Алгоритмы работы устройства

1.6.1 Общее описание взаимодействия с устройством

Взаимодействие с устройством происходит при помощи 4 кнопок – «Отправка строки из ячейки», «Увеличение номера ячейки», «Уменьшение номера ячейки». Также присутствует кнопка сброса состояния МК для случая зависания работы программы.

Основной источник вывода информации – дисплей. Если на него не выводится сгенерированный QR-код, то на нем выведено сообщение о том, что данная ячейка памяти является пустой. При нажатии кнопок увеличения и уменьшения номера ячейки изменяется номер ячейки и изображение на дисплее.

При отправке строки с ПЭВМ или телефона на МК-систему генерируется QR-код и выводится на дисплей, а вводимая строка сохраняется в памяти. Алгоритм работы устройства представлен на рисунке 15.

Рисунок 15 – Схема алгоритма работы устройства

1.6.2 Описание алгоритма прерывания кнопок

Кнопки работают на специально включенных для них прерываниях. При вызове функции прерываний проверяется какой из пинов подал сигнал, чтобы понять какая из кнопок была нажата и вызвать соответствующий алгоритм. Описание алгоритма функции прерываний представлено на рисунке 16.

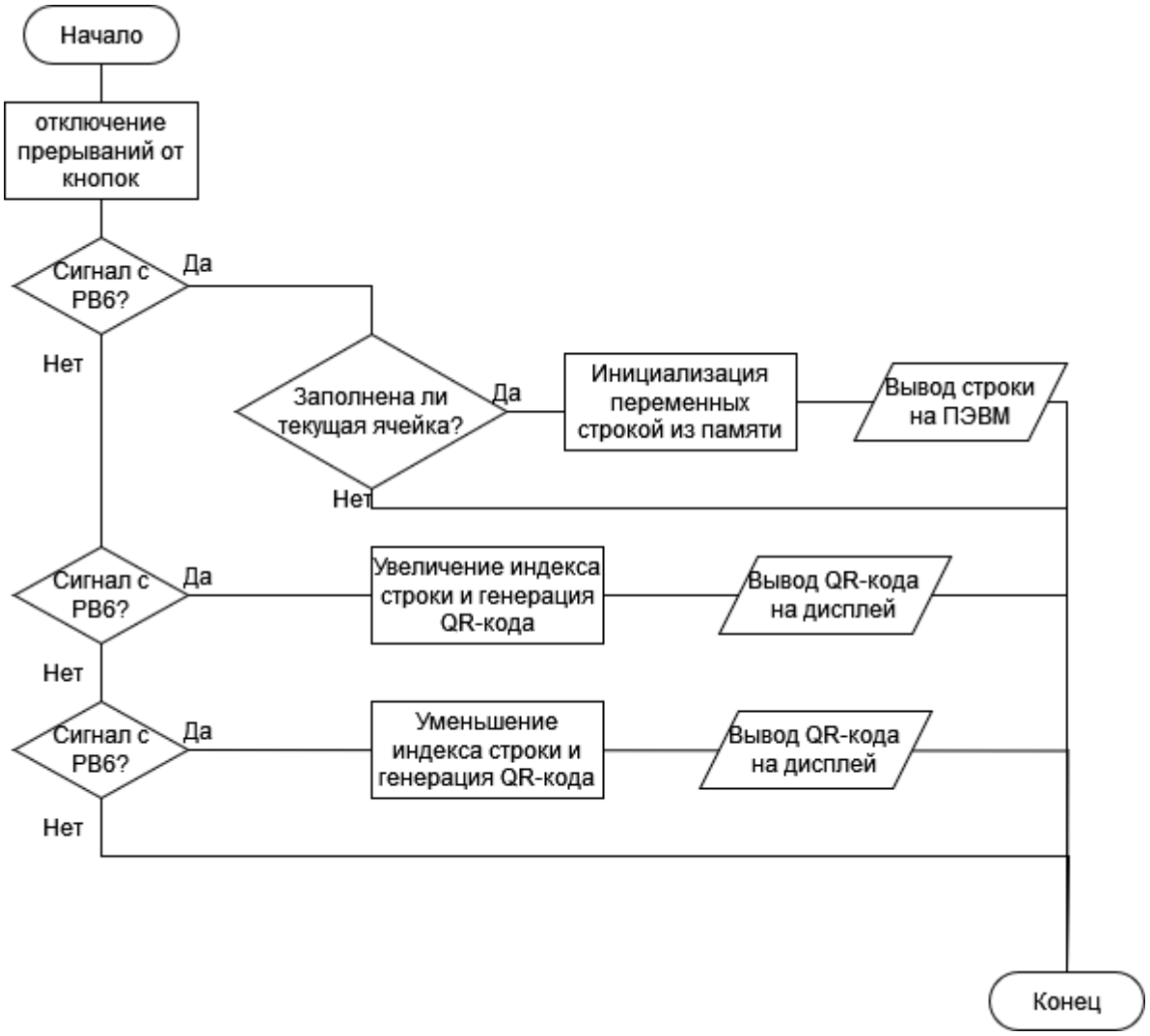

Рисунок 16 – Схема алгоритма прерываний кнопок

1.6.3 Описание и расчет алгоритма генерации QR-кода

Генерация QR-кода происходит регулярно, так как хранение QR-кодов в памяти излишне загружает память. Даже в рассмотренной нами системе наибольший размер отображаемого QR-кода равен 132x132, даже если хранить отображаемые пиксели побитово, он будет занимать более 2 Кбайт. Так как в системе надо хранить 10 QR-кодов, хранение готовых QR-кодов

требует большое количество памяти. Намного более экономно генерировать QR-код по запросу.

Алгоритм генерации QR-кода представлен на рисунке 17.

Рисунок 17 – Схема алгоритма генерации QR-кода

2 Технологическая часть

Для реализации работы устройства для измерения скорости чтения была написана программа на языке Си [8] и создан макет устройства в программе Proteus 8.

2.1 Отладка и тестирование программы

Отладка программы происходила в среде разработки Proteus 8. Среда предназначена для выполнения моделирования аналоговых и цифровых устройств. Созданный в ней макет позволяет взаимодействовать с виртуальным терминалом для имитации работы с ПЭВМ, работать с устройством при помощи кнопок и вывода на дисплей.

2.2 Симуляция работы устройства

Созданный макет в среде разработки Proteus 8 представлен на рисунке 18 в незапущенном состоянии.

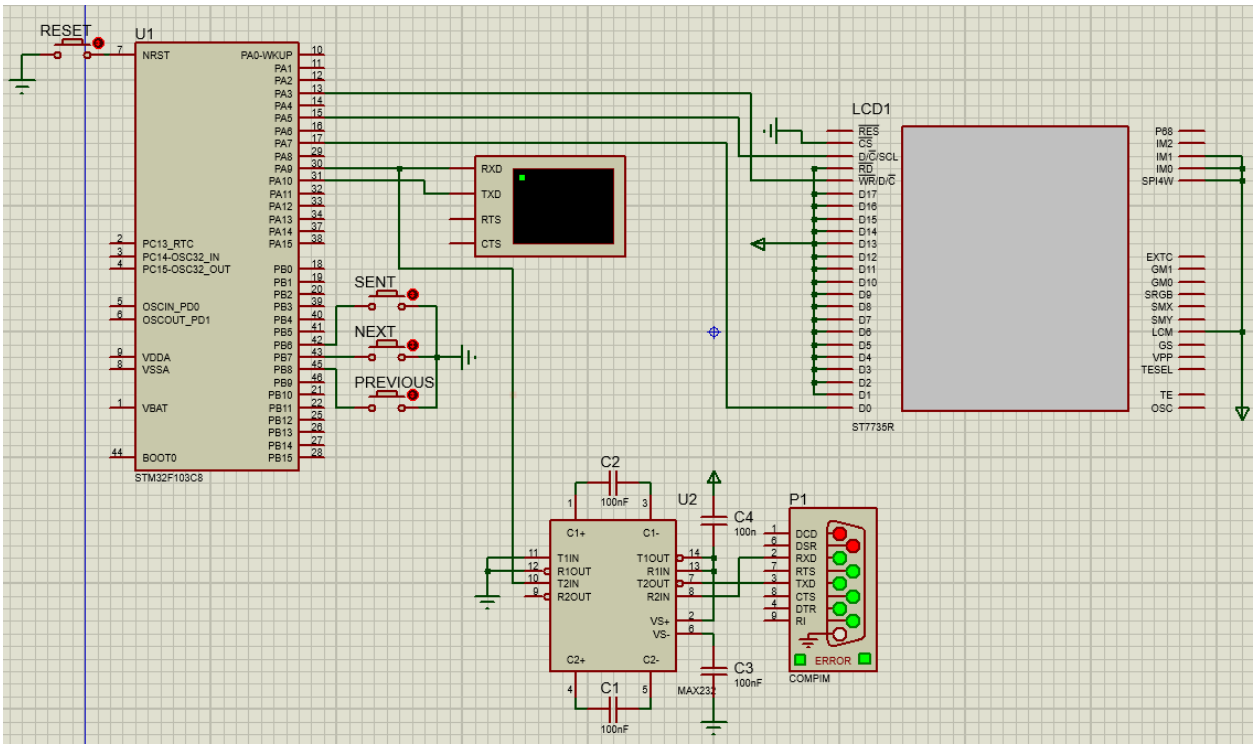

Рисунок 18 – Макет в незапущенном состоянии

Макет в Proteus отличается от принципиальной схемы отсутствием резисторов, так как в симуляции подается стабильные 3.3 вольта и нет необходимости в распределении поступающего от портов тока.

После запуска отображается вступительное сообщение с инструкцией для пользователя. Состояние устройства на этот момент представлено на рисунке 19.

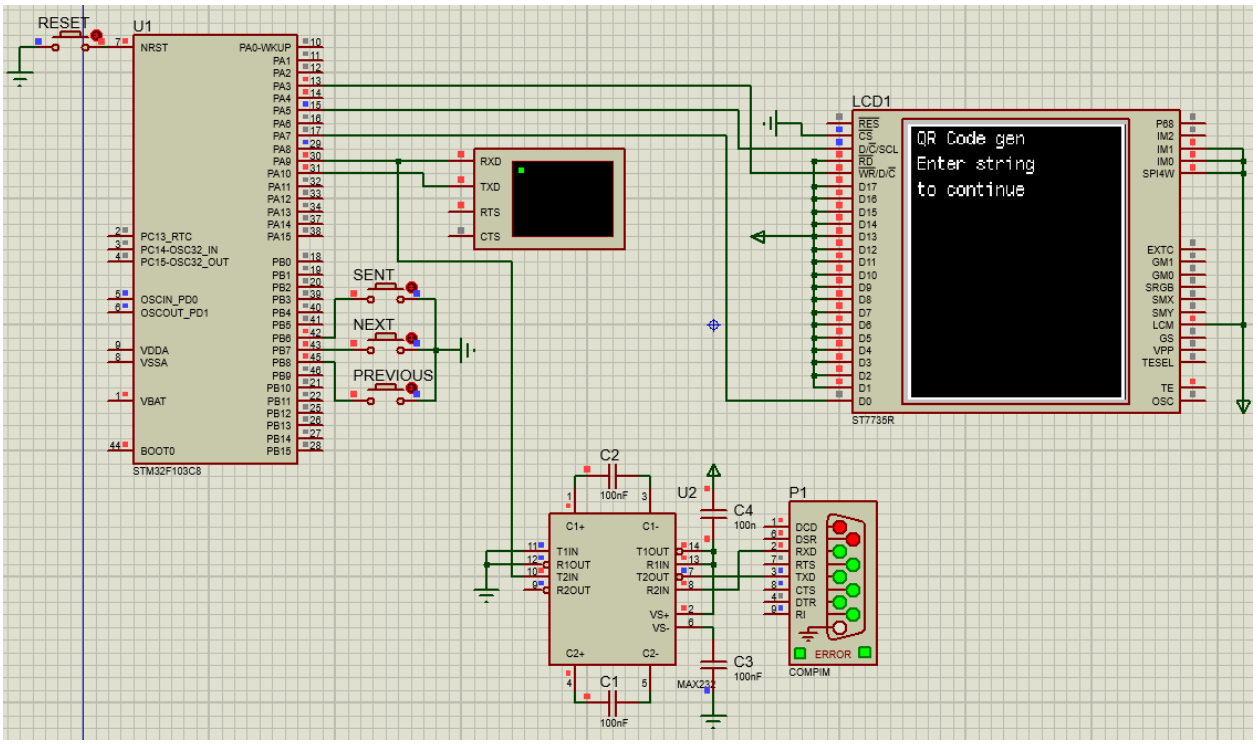

Рисунок 19 – Запущенная программа на этапе выбора размера

При нажатии кнопок перехода между ячейками памяти будет выведено на дисплей сообщение с текущим индексом и сообщением о том, что в этой ячейке не расположен никакой QR-код, или, если он уже находится в памяти, изображение будет выведено на экран. Пример вывода сообщения о пустой ячейке показан на рисунке 20.

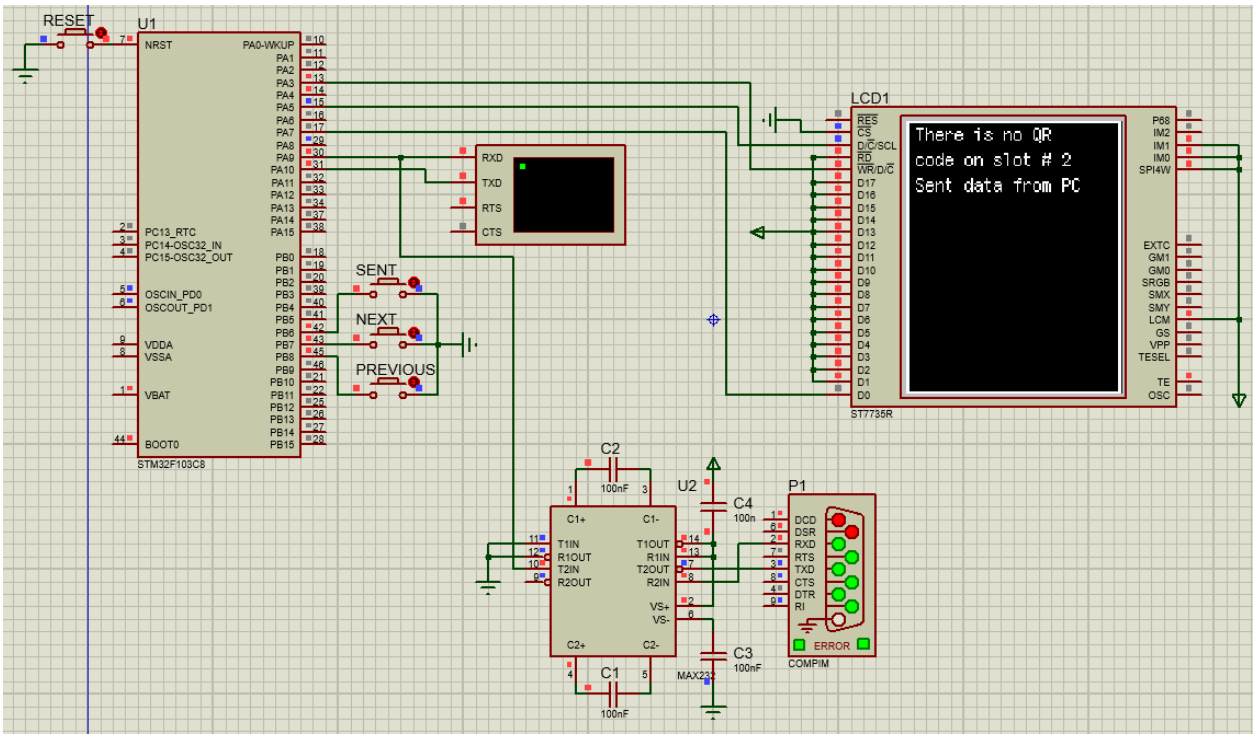

Рисунок 20 – Вывод сообщения о пустой ячейке

Если строка отправлена на МК-систему, то на дисплее будет отображен QR-код, кодирующий эту строку. Пример отображения QR-кода показан на рисунке 21.

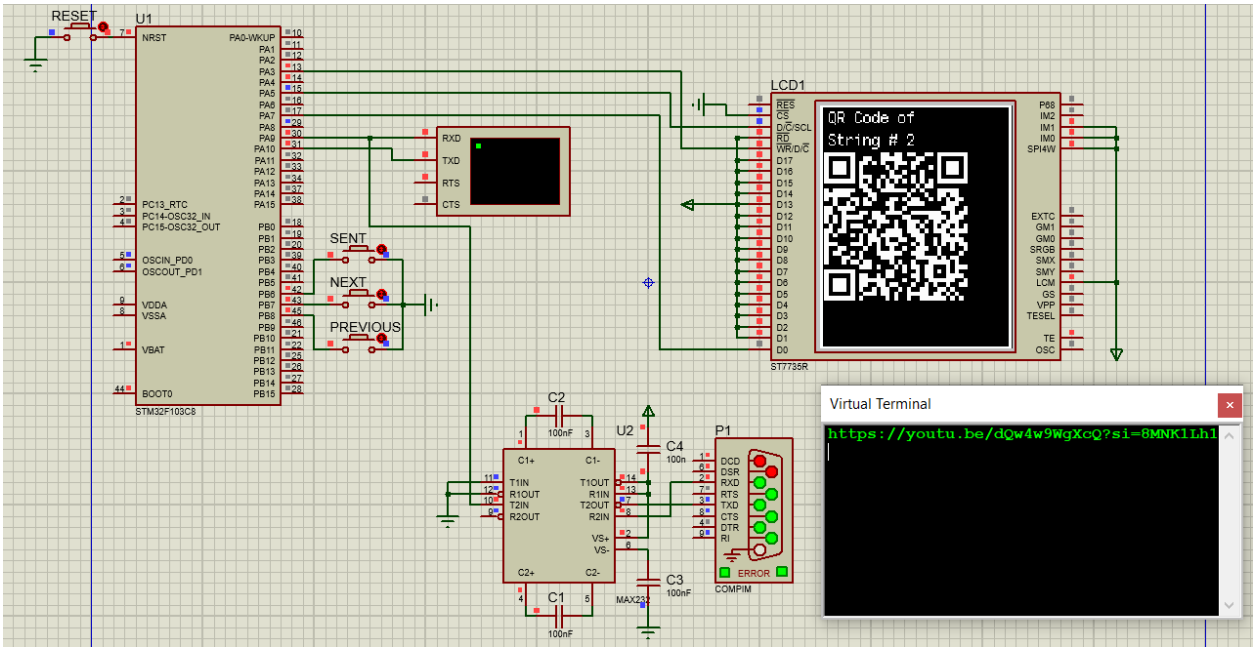

Рисунок 21 – Вывод QR-кода после введенной строки

2.3 Тестирование выходных сигналов МК

На рисунке 22 показана осциллограма сигнала до и после драйвера MAX232.

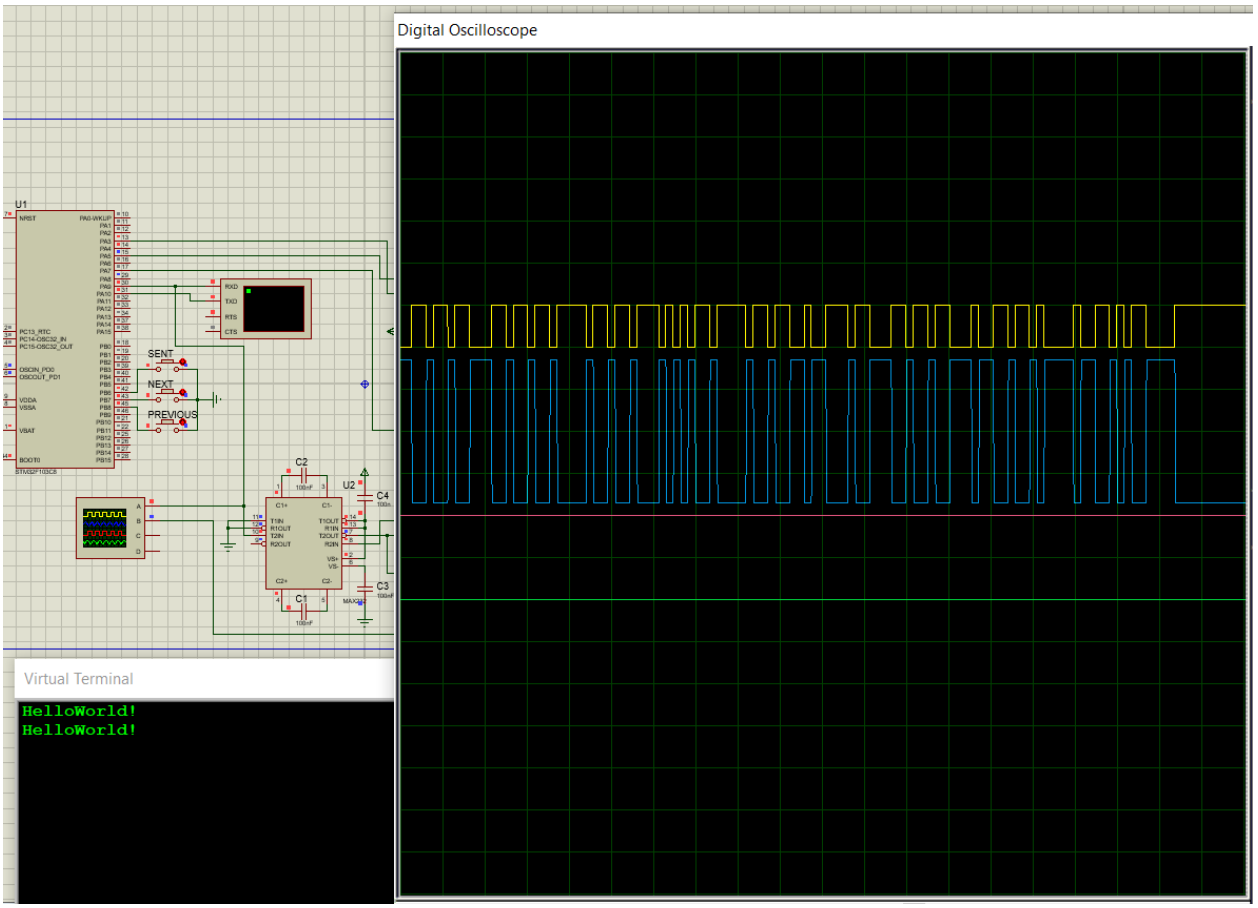

Рисунок 22 – Осциллограмма сигнала до и после драйвера MAX232

2.4 Программирование МК

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана МК-система для генерации QR-кода из строк, вводимых с ПЭВМ и телефона. Система работает на микроконтроллере STM32F103C8T6, она разработана в соответствии с условиями ТЗ.

Код программы для МК написан на языке С в среде программирования STM32CubeIDE, модель отлажена и протестирована в программе Proteus 8.15.

В процессе работы над курсовой работой были разработаны схемы электрические функциональная и принципиальная, спецификация и документация к устройству.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Информация о тонкопленочных транзисторных дисплеях [Электронный ресурс] – URL: https://monitor4ik.com/stati/tft/ (Дата обращения – 18.12.2023).

2 Методические указания к лабораторной работе с МК STM32F103C8T6 [Электронный ресурс] – URL: https://e-learning.bmstu.ru/iu6/pluginfile.php/19506/mod_resource/content/2/%D0%9B%D0%A0%20STM32%202023.pdf (Дата обращения – 18.12.2023).

3 Даташит МК STM32F103C8T6 [Электронный ресурс] – URL: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf (Дата обращения – 18.12.2023).

4 Даташит дисплея ST7735 [Электронный ресурс] – URL: https://www.displayfuture.com/Display/datasheet/controller/ST7735.pdf (Дата обращения – 18.12.2023).

5 Статья про работу интерфейса SPI в МК семейства STM32 [Электронный ресурс] – URL: https://303421.selcdn.ru/soel-upload/clouds/1/iblock/e84/e84786cd2be538cfb9eb44a3938d1dcc/20140118.pdf (Дата обращения – 18.12.2023).

6 Информация об используемых SPI регистрах в МК семейства STM32 [Электронный ресурс] – URL: https://www.rotr.info/electronics/mcu/arm_usart_stm32_reg.htm (Дата обращения – 18.12.2023).

7 Руководство по программированию МК семейства STM32 [Электронный ресурс] – URL: https://portal.tpu.ru/SHARED/t/TORGAEV/academic/Tab4/Posobie3.pdf (Дата обращения – 18.12.2023).

8 Программирование на Си [Электронный ресурс]. – URL: http://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf (дата обращения 27.10.2023).

9 ГОСТ 2.710-81 Обозначения буквенно-цифровые в электрических схемах.

10 ГОСТ 2.721-74 Обозначения условные графические в схемах. Обозначения общего применения.

11 ГОСТ 2.102-68 ЕСКД. Виды и комплектность конструкторских документов.

12 ГОСТ 2.105-95 ЕСКД. Текстовые документы

ПРИЛОЖЕНИЕ А

Текст программы.

Заголовочные файлы:

main.h

```
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
#include "stm32f1xx_hal.h"
void Error_Handler(void);
extern SPI_HandleTypeDef hspi1;
extern UART_HandleTypeDef huart1;
#ifdef __cplusplus
}
#endif
#endif /* __MAIN_H */
```

logic.h

```
#ifndef LOGIC_H_
#define LOGIC_H_
static void createQR(const char* text, int* size, unsigned char qr[]);
#endif  // LOGIC_H_
```

qrcodegen.h

```
#pragma once
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#ifdef __cplusplus
extern "C" {
#endif
enum qrcodegen_Ecc {
	qrcodegen_Ecc_LOW = 0 ,  // The QR Code can tolerate about  7% erroneous
codewords
	qrcodegen_Ecc_MEDIUM  ,  // The QR Code can tolerate about 15% erroneous
codewords
	qrcodegen_Ecc_QUARTILE,  // The QR Code can tolerate about 25% erroneous
codewords
	qrcodegen_Ecc_HIGH    ,  // The QR Code can tolerate about 30% erroneous
codewords
};
enum qrcodegen_Mask {
	qrcodegen_Mask_AUTO = -1,
	qrcodegen_Mask_0 = 0,
	qrcodegen_Mask_1,
	qrcodegen_Mask_2,
	qrcodegen_Mask_3,
```

```
	qrcodegen_Mask_4,
	qrcodegen_Mask_5,
	qrcodegen_Mask_6,
	qrcodegen_Mask_7,
};
enum qrcodegen_Mode {
	qrcodegen_Mode_NUMERIC      = 0x1,
	qrcodegen_Mode_ALPHANUMERIC = 0x2,
	qrcodegen_Mode_BYTE         = 0x4,
	qrcodegen_Mode_KANJI        = 0x8,
	qrcodegen_Mode_ECI          = 0x7,
};
struct qrcodegen_Segment {
	enum qrcodegen_Mode mode;
	int numChars;
	uint8_t *data;
	int bitLength;
};
#define qrcodegen_VERSION_MIN   1  // The minimum version number supported in the
QR Code Model 2 standard
#define qrcodegen_VERSION_MAX  27  // The maximum version number supported in the
QR Code Model 2 standard
#define qrcodegen_BUFFER_LEN_FOR_VERSION(n)  ((((n) * 4 + 17) * ((n) * 4 + 17) + 7)
/ 8 + 1)
#define qrcodegen_BUFFER_LEN_MAX
qrcodegen_BUFFER_LEN_FOR_VERSION(qrcodegen_VERSION_MAX)
bool qrcodegen_encodeText(const char *text, uint8_t tempBuffer[], uint8_t qrcode[],
	enum qrcodegen_Ecc ecl, int minVersion, int maxVersion, enum qrcodegen_Mask
mask, bool boostEcl);
bool qrcodegen_encodeBinary(uint8_t dataAndTemp[], size_t dataLen, uint8_t
qrcode[],
	enum qrcodegen_Ecc ecl, int minVersion, int maxVersion, enum qrcodegen_Mask
mask, bool boostEcl);
bool qrcodegen_encodeSegments(const struct qrcodegen_Segment segs[], size_t len,
	enum qrcodegen_Ecc ecl, uint8_t tempBuffer[], uint8_t qrcode[]);
bool qrcodegen_encodeSegmentsAdvanced(const struct qrcodegen_Segment segs[], size_t
len, enum qrcodegen_Ecc ecl,
	int minVersion, int maxVersion, enum qrcodegen_Mask mask, bool boostEcl,
uint8_t tempBuffer[], uint8_t qrcode[]);
bool qrcodegen_isNumeric(const char *text);
bool qrcodegen_isAlphanumeric(const char *text);
size_t qrcodegen_calcSegmentBufferSize(enum qrcodegen_Mode mode, size_t numChars);
struct qrcodegen_Segment qrcodegen_makeBytes(const uint8_t data[], size_t len,
uint8_t buf[]);
struct qrcodegen_Segment qrcodegen_makeNumeric(const char *digits, uint8_t buf[]);
struct qrcodegen_Segment qrcodegen_makeAlphanumeric(const char *text, uint8_t
buf[]);
struct qrcodegen_Segment qrcodegen_makeEci(long assignVal, uint8_t buf[]);
int qrcodegen_getSize(const uint8_t qrcode[]);
bool qrcodegen_getModule(const uint8_t qrcode[], int x, int y);
#ifdef __cplusplus
}
#endif
```

ST7735.h

```
#ifndef ST7735_H_
#define ST7735_H_
#include <stdbool.h>
#include <stdlib.h>
#include "main.h"
typedef struct {
   const uint8_t width;
   uint8_t height;
   const uint16_t *data;
} FontDef;
extern FontDef Font_7x10;
extern SPI_HandleTypeDef ST7735_SPI_PORT;
void ST7735_Init();
void ST7735_DrawString(uint16_t x, uint16_t y, const char* str, FontDef font);
void ST7735_FillScreen();
void ST7735_DrawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1);
void ST7735_DrawPixel(int16_t x, int16_t y);
void ST7735_WriteImage(uint8_t* qr, size_t size, size_t mult);
/* Color definitions */
#define ST7735_BLACK   0x0000
#define ST7735_BLUE    0x001F
#define ST7735_RED     0xF800
#define ST7735_GREEN   0x07E0
#define ST7735_CYAN    0x07FF
#define ST7735_MAGENTA 0xF81F
#define ST7735_YELLOW  0xFFE0
#define ST7735_WHITE   0xFFFF
#define ST7735_RGB(r, g, b) (((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b & 0xF8) >>
3))
/* ONLY CONFIG BELOW */
#define ST7735_SPI_PORT hspi1 //hspi1, hspi2, hspi3...
//Port and pin connected signal 'DC' (data or command) ST7735 display
#define ST7735_DC_Pin       GPIO_PIN_3
#define ST7735_DC_GPIO_Port   GPIOA
// WaveShare ST7735S-based 1.8" display, default orientation
#define ST7735_IS_160X128     1
#define ST7735_WIDTH        128
#define ST7735_HEIGHT       160
#define ST7735_XSTART       2
#define ST7735_YSTART       1
#define ST7735_DATA_ROTATION  0
#define ST7735_NOP     0x00
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09
#define ST7735_SLPIN   0x10
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
#define ST7735_NORON   0x13
```

```
#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E
#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36
#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6
#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5
#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD
#define ST7735_PWCTR6  0xFC
#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1
#define DELAY 0x80
/* Ports config */
#define TFT_DC_D()  HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin,
GPIO_PIN_SET)
#define TFT_DC_C()  HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin,
GPIO_PIN_RESET)
/* Init comands */
static const uint8_t
init_cmds1[] = {                    // Init for 7735R, part 1 (red or green tab)
     15,                           // 15 commands in list:
     ST7735_SWRESET, DELAY,        //  1: Software reset, 0 args, w/delay
     150,                          //     150 ms delay
     ST7735_SLPOUT, DELAY,         //  2: Out of sleep mode, 0 args, w/delay
     255,                          //     500 ms delay
     ST7735_FRMCTR1, 3,            //  3: Frame rate ctrl - normal mode, 3 args:
     0x01, 0x2C, 0x2D,             //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
     ST7735_FRMCTR2, 3,            //  4: Frame rate control - idle mode, 3 args:
     0x01, 0x2C, 0x2D,             //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
     ST7735_FRMCTR3, 6,            //  5: Frame rate ctrl - partial mode, 6 args:
     0x01, 0x2C, 0x2D,             //     Dot inversion mode
     0x01, 0x2C, 0x2D,             //     Line inversion mode
     ST7735_INVCTR, 1,             //  6: Display inversion ctrl, 1 arg, no delay:
     0x07,                         //     No inversion
     ST7735_PWCTR1, 3, 0xA2,       //  7: Power control, 3 args, no delay:
```

```
    0x02,                           //     -4.6V
    0x84,                           //     AUTO mode
    ST7735_PWCTR2, 1,               //  8: Power control, 1 arg, no delay:
    0xC5,                           //     VGH25 = 2.4C VGSEL = -10 VGH = 3 * AVDD
    ST7735_PWCTR3, 2,               //  9: Power control, 2 args, no delay:
    0x0A,                           //     Opamp current small
    0x00,                           //     Boost frequency
    ST7735_PWCTR4, 2,               // 10: Power control, 2 args, no delay:
    0x8A, 0x2A,                     //     BCLK/2, Opamp current small & Medium low
    ST7735_PWCTR5, 2, 0x8A, 0xEE,   // 11: Power control, 2 args, no delay:
    ST7735_VMCTR1, 1, 0x0E,         // 12: Power control, 1 arg, no delay:
    ST7735_INVOFF, 0,               // 13: Don't invert display, no args, no delay
    ST7735_MADCTL, 1,               // 14: Memory access control (directions), 1
arg:
    ST7735_DATA_ROTATION,           //     row addr/col addr, bottom to top refresh
    ST7735_COLMOD, 1,               // 15: set color mode, 1 arg, no delay:
    0x05                                    //     16-bit color
};
#endif /* ST7735_H_ */
```

Программные файлы:

main.c

```
#include "main.h"

#include "ST7735.h"
#include "qrcodegen.h"

#define BUFFER_SIZE 1
#define STRING_MAX_LEN 1024

SPI_HandleTypeDef hspi1;
UART_HandleTypeDef huart1;

uint8_t current_string;
uint16_t current_byte;
char strings[10][STRING_MAX_LEN];
uint8_t isTaken[10];
uint8_t receiveBuffer[BUFFER_SIZE];


void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);

int main(void)
{
 HAL_Init();
 SystemClock_Config();
```

```
 MX_GPIO_Init();
 MX_SPI1_Init();
 MX_USART1_UART_Init();
 /* USER CODE BEGIN 2 */
 HAL_NVIC_DisableIRQ(EXTI9_5_IRQn);
 NVIC_SetPriority(SysTick_IRQn, 0);
 ST7735_Init();
 current_string = 0;
 current_byte = 0;
 for (int i = 0; i < 10; i++)
        isTaken[i] = 0;
 ResetScreen();
 HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
 HAL_UART_Receive_IT(&huart1, receiveBuffer, BUFFER_SIZE);
 while (1)
 {

 }
}

void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
   Error_Handler();
 }

 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
 {
   Error_Handler();
 }
}
static void MX_SPI1_Init(void)
{
 hspi1.Instance = SPI1;
 hspi1.Init.Mode = SPI_MODE_MASTER;
 hspi1.Init.Direction = SPI_DIRECTION_2LINES;
 hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
 hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
 hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
```

```
 hspi1.Init.NSS = SPI_NSS_SOFT;
 hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
 hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
 hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
 hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
 hspi1.Init.CRCPolynomial = 10;
 if (HAL_SPI_Init(&hspi1) != HAL_OK)
 {
   Error_Handler();
 }
}

static void MX_USART1_UART_Init(void)
{
 huart1.Instance = USART1;
 huart1.Init.BaudRate = 115200;
 huart1.Init.WordLength = UART_WORDLENGTH_8B;
 huart1.Init.StopBits = UART_STOPBITS_1;
 huart1.Init.Parity = UART_PARITY_NONE;
 huart1.Init.Mode = UART_MODE_TX_RX;
 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart1.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart1) != HAL_OK)
 {
   Error_Handler();
 }
}

static void MX_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOD_CLK_ENABLE();
 __HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
 /*Configure GPIO pin : PA3 */
 GPIO_InitStruct.Pin = GPIO_PIN_3;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
 /*Configure GPIO pins : PB6 PB7 PB8 */
 GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8;
 GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
 GPIO_InitStruct.Pull = GPIO_PULLUP;
 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
 /* EXTI interrupt init*/
 HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
```

```
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
 if (huart->Instance == USART1){
        strings[current_string][current_byte] = receiveBuffer[0];
        current_byte++;
        if (receiveBuffer[0] == 0x0D){
              int i = 0;
              uint8_t temp[STRING_MAX_LEN] = "";
              while (strings[current_string][i] != 0x0D){
                    temp[i] = strings[current_string][i];
                    i++;
              }
              temp[i] = '\0';
              ResetScreenWith(temp, current_string);
              current_byte = 0;
              isTaken[current_string] = 1;
        }
        HAL_UART_Receive_IT(&huart1, receiveBuffer, BUFFER_SIZE);
 }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
      HAL_NVIC_DisableIRQ(EXTI9_5_IRQn);
        if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_6) == 0) {
              if (isTaken[current_string]){
                    int i = 0;
                    uint8_t temp[STRING_MAX_LEN] = "";
                    while (strings[current_string][i] != 0x0D){
                          temp[i] = strings[current_string][i];
                          i++;
                    }
                    temp[i] = '\r';
                    i++;
                    HAL_UART_Transmit_IT(&huart1, temp, i);
              }
        } else if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_7) == 0) {
              current_string++;
              if (current_string > 9){
                    current_string = 0;
              }
              if (isTaken[current_string]){
                    int i = 0;
                    uint8_t temp[1024];
                    while (strings[current_string][i] != 0x0D){
                          temp[i] = strings[current_string][i];
                          i++;
                    }
                    temp[i] = '\0';
                    ResetScreenWith(temp, current_string);
              } else {
                    BlankScreen(current_string);
              }
        } else if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8) == 0) {
```

```
                if (current_string == 0)
                        current_string = 9;
                else
                        current_string--;
                if (isTaken[current_string]){
                        int i = 0;
                        uint8_t temp[STRING_MAX_LEN];
                        while (strings[current_string][i] != 0x0D){
                                temp[i] = strings[current_string][i];
                                i++;
                        }
                        temp[i] = '\0';
                        ResetScreenWith(temp, current_string);
                } else {
                        BlankScreen(current_string);
                }
          }
        HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
}
void Error_Handler(void)
{
 __disable_irq();
 while (1)
 {
 }
}
#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif
```

logic.c

```
#include <limits.h>
#include <string.h>
#include <stdio.h>
#include "logic.h"
#include "main.h"
#include "ST7735.h"
#include "qrcodegen.h"
void ResetScreen() {
        ST7735_FillScreen();
        char *text[] = { "QR Code gen", "Enter string", "to continue"};
        for (int i = 0; i < sizeof(text) / sizeof(text[i]); ++i) {
                ST7735_DrawString(3, 3 + i * 15, text[i], Font_7x10);
        }
}
void BlankScreen(uint8_t cur_str){
        ST7735_FillScreen();
```

```
	char temp[18] = "code on slot # ";
	int num = cur_str;
	num++;
	if (num == 10){
		temp [15] = '1';
		temp [16] = '0';
	}
	else{
		num += 0x30;
		temp [15] = num;
	}
	char *text[] = { "There is no QR", temp, "Sent data from PC" };
		for (int i = 0; i < sizeof(text) / sizeof(text[i]); ++i) {
			ST7735_DrawString(3, 3 + i * 15, text[i], Font_7x10);
		}
}
void ResetScreenWith(char * str, uint8_t cur_str) {
	ST7735_FillScreen();
	int num = cur_str;
	num++;
	char temp[11] = "String # ";
	if (num == 10){
		temp [9] = '1';
		temp [10] = '0';
	}
	else{
		num += 0x30;
		temp [9] = num;
	}
	char *text[] = { "QR Code of", temp };
	for (int i = 0; i < sizeof(text) / sizeof(text[i]); ++i) {
		ST7735_DrawString(3, 3 + i * 15, text[i], Font_7x10);
	}
	PrintQR(str);
}
void ReturnQR(char * str, const char* qr, int *size){
	createQR(str, &size, qr);
}
char intToHex(uint8_t num) {
	if (num < 10)
		return num + 48;
	else return num + 55;
}
void ReturnQRBits(char * str, const char* qr, int *size, uint8_t *retBuf){
	createQR(str, &size, qr);
	int k = 0;
	int bit = 0;
	for (int y = 0; y < size; y++)
		for (int x = 0; x < size; x++){
			retBuf[k] *= 2;
			if (qrcodegen_getModule(qr, x, y)){
				retBuf[k]++;
			}
```

```
                    bit++;
                    if (bit == 4){
                          bit = 0;
                          k++;
                    }
             }
      retBuf[(*size)*(*size)] = '\r';
}
void PrintQR(char *str)
{
      uint8_t qr[qrcodegen_BUFFER_LEN_MAX];
      int size = 0;
      createQR(str, &size, qr);
      int mult = 130/size;
      ST7735_WriteImage(qr, size, mult);
}
static void createQR(const char* text, int *size, uint8_t qr[]) {
      enum qrcodegen_Ecc errCorLvl = qrcodegen_Ecc_MEDIUM;  // Error correction
level
      // Make and print the QR Code symbol
      uint8_t tempBuffer[qrcodegen_BUFFER_LEN_MAX];
      bool ok = qrcodegen_encodeText(text, tempBuffer, qr, errCorLvl,
            qrcodegen_VERSION_MIN, qrcodegen_VERSION_MAX, qrcodegen_Mask_AUTO,
true);
      if (ok)
            *size = qrcodegen_getSize(qr);
      else
            *size = 0;
}
```

ST7735.c

```
#include "ST7735.h"
uint16_t fcolor = ST7735_WHITE;  // Background color definition
uint16_t bcolor = ST7735_BLACK;  // Font and lines color definition
/* Helpers prototypes */
static void ST7735_Reset();
static void ST7735_WriteCommand(uint8_t cmd);
static void ST7735_WriteData(uint8_t* buff, size_t buff_size);
static void ST7735_ExecuteCommandList(const uint8_t *addr);
static void ST7735_SetAddressWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t
y1);
static void ST7735_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font,
uint16_t color, uint16_t bgcolor);
/* Main functions */
void ST7735_Init()
{
   ST7735_Reset();
   ST7735_ExecuteCommandList(init_cmds1);
}
void ST7735_DrawString(uint16_t x, uint16_t y, const char* str, FontDef font)
```

```
{
   while(*str)
   {
       ST7735_WriteChar(x, y, *str++, font, fcolor, bcolor);
       x += font.width;
   }
}
void ST7735_FillScreen()
{
      ST7735_SetAddressWindow(0, 0, ST7735_WIDTH - 1, ST7735_HEIGHT - 1);
      uint8_t data[] = { bcolor >> 8, bcolor & 0xFF };
      TFT_DC_D();
      for (int y = ST7735_HEIGHT; y >= 0; y--) {
            for (int x = ST7735_WIDTH; x >= 0; x--) {
                  ST7735_WriteData(data, sizeof(data));
            }
      }
}
void ST7735_DrawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1)
{
      ST7735_SetAddressWindow(x0, y0, x1, y0);
      uint8_t data[] = { fcolor >> 8, fcolor & 0xFF };
      TFT_DC_D();
      for (int i = x0; i < x1; ++i) {
            ST7735_WriteData(data, sizeof(data));
      }
      ST7735_SetAddressWindow(x0, y0, x0, y1);
      for (int i = y0; i < y1; ++i) {
            ST7735_WriteData(data, sizeof(data));
      }
}
/* Helpers */
static void ST7735_Reset()
{
 HAL_Delay(20);
}
static void ST7735_WriteCommand(uint8_t cmd)
{
 TFT_DC_C();
 HAL_SPI_Transmit(&ST7735_SPI_PORT, &cmd, sizeof(cmd), HAL_MAX_DELAY);
}
void ST7735_DrawPixel(int16_t x, int16_t y) {
//  if((x < 0) || (x >= 132) || (y < 0) || (y >= 132)) return;
 ST7735_SetAddressWindow(x,y,x,y);
 pushColor();
}
void ST7735_WriteImage(uint8_t* qr, size_t size, size_t mult) {
      ST7735_SetAddressWindow(1,30,1+size*mult-1,30+size*mult-1);
      uint8_t data[] = { fcolor >> 8, fcolor & 0xFF };
      uint8_t edata[] = { bcolor >> 8, bcolor & 0xFF };
      if (mult == 1){
            for (int y = 0; y < size; y++)
                  for (int x = 0; x < size; x++)
```

```
                        if (qrcodegen_getModule(qr, x, y)){
                                ST7735_WriteData(data, sizeof(data));
                        }
                        else{
                                ST7735_WriteData(edata, sizeof(edata));
                        }
        } else {
                for (int y = 0; y < size; y++)
                        for (int i = y*mult; i < mult*(y+1); i++)
                                for (int x = 0; x < size; x++)
                                        for(int j = x*mult; j < mult*(x+1); j++)
                                                if (qrcodegen_getModule(qr, x, y)){
                                                        ST7735_WriteData(data, sizeof(data));
                                                }
                                                else{
                                                        ST7735_WriteData(edata,
sizeof(edata));
                                                }
        }
}
static void ST7735_WriteData(uint8_t* buff, size_t buff_size)
{
 TFT_DC_D();
 HAL_SPI_Transmit(&ST7735_SPI_PORT, buff, buff_size, HAL_MAX_DELAY);
}
static void ST7735_ExecuteCommandList(const uint8_t *addr)
{
   uint8_t numCommands, numArgs;
   uint16_t ms;
   numCommands = *addr++;
   while(numCommands--)
   {
     uint8_t cmd = *addr++;
       ST7735_WriteCommand(cmd);
       numArgs = *addr++;
       // If high bit set, delay follows args
       ms = numArgs & DELAY;
       numArgs &= ~DELAY;
       if(numArgs)
       {
           ST7735_WriteData((uint8_t*)addr, numArgs);
           addr += numArgs;
       }
       if(ms)
       {
           ms = *addr++;
           if(ms == 255) ms = 500;
           HAL_Delay(ms);
       }
   }
}
static void ST7735_SetAddressWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1)
{
```

```
    // column address set
    ST7735_WriteCommand(ST7735_CASET);
    uint8_t data[] = { 0x00, x0 + ST7735_XSTART, 0x00, x1 + ST7735_XSTART };
    ST7735_WriteData(data, sizeof(data));
    // row address set
    ST7735_WriteCommand(ST7735_RASET);
    data[1] = y0 + ST7735_YSTART;
    data[3] = y1 + ST7735_YSTART;
    ST7735_WriteData(data, sizeof(data));
    // write to RAM
    ST7735_WriteCommand(ST7735_RAMWR);
}
static void ST7735_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font,
uint16_t color, uint16_t bgcolor)
{
    uint32_t i, b, j;
    ST7735_SetAddressWindow(x, y, x+font.width-1, y+font.height-1);
    for(i = 0; i < font.height; i++)
    {
        b = font.data[(ch - 32) * font.height + i];
        for(j = 0; j < font.width; j++)
        {
            if((b << j) & 0x8000)
            {
                uint8_t data[] = { color >> 8, color & 0xFF };
                ST7735_WriteData(data, sizeof(data));
            }
            else
            {
                uint8_t data[] = { bgcolor >> 8, bgcolor & 0xFF };
                ST7735_WriteData(data, sizeof(data));
            }
        }
    }
}
/* Font definition */
static const uint16_t Font7x10 [] = {
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,  //
sp
0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x1000, 0x0000, 0x0000,  //
!
0x2800, 0x2800, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,  //
"
0x2400, 0x2400, 0x7C00, 0x2400, 0x4800, 0x7C00, 0x4800, 0x4800, 0x0000, 0x0000,  //
#
0x3800, 0x5400, 0x5000, 0x3800, 0x1400, 0x5400, 0x5400, 0x3800, 0x1000, 0x0000,  //
$
0x2000, 0x5400, 0x5800, 0x3000, 0x2800, 0x5400, 0x1400, 0x0800, 0x0000, 0x0000,  //
%
0x1000, 0x2800, 0x2800, 0x1000, 0x3400, 0x4800, 0x4800, 0x3400, 0x0000, 0x0000,  //
&
0x1000, 0x1000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,  //
'
```

```
0x0800, 0x1000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x1000, 0x0800,  //
(
0x2000, 0x1000, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x1000, 0x2000,  //
)
0x1000, 0x3800, 0x1000, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,  //
*
0x0000, 0x0000, 0x1000, 0x1000, 0x7C00, 0x1000, 0x1000, 0x0000, 0x0000, 0x0000,  //
+
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x1000, 0x1000,  //
,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x3800, 0x0000, 0x0000, 0x0000, 0x0000,  //
-
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000,  //
.
0x0800, 0x0800, 0x1000, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x0000, 0x0000,  //
/
0x3800, 0x4400, 0x4400, 0x5400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000,  //
0
0x1000, 0x3000, 0x5000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000,  //
1
0x3800, 0x4400, 0x4400, 0x0400, 0x0800, 0x1000, 0x2000, 0x7C00, 0x0000, 0x0000,  //
2
0x3800, 0x4400, 0x0400, 0x1800, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000,  //
3
0x0800, 0x1800, 0x2800, 0x2800, 0x4800, 0x7C00, 0x0800, 0x0800, 0x0000, 0x0000,  //
4
0x7C00, 0x4000, 0x4000, 0x7800, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000,  //
5
0x3800, 0x4400, 0x4000, 0x7800, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000,  //
6
0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000,  //
7
0x3800, 0x4400, 0x4400, 0x3800, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000,  //
8
0x3800, 0x4400, 0x4400, 0x4400, 0x3C00, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000,  //
9
0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000,  //
:
0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x1000, 0x1000, 0x1000,  //
;
0x0000, 0x0000, 0x0C00, 0x3000, 0x4000, 0x3000, 0x0C00, 0x0000, 0x0000, 0x0000,  //
<
0x0000, 0x0000, 0x0000, 0x7C00, 0x0000, 0x7C00, 0x0000, 0x0000, 0x0000, 0x0000,  //
=
0x0000, 0x0000, 0x6000, 0x1800, 0x0400, 0x1800, 0x6000, 0x0000, 0x0000, 0x0000,  //
>
0x3800, 0x4400, 0x0400, 0x0800, 0x1000, 0x1000, 0x0000, 0x1000, 0x0000, 0x0000,  //
?
0x3800, 0x4400, 0x4C00, 0x5400, 0x5C00, 0x4000, 0x4000, 0x3800, 0x0000, 0x0000,  //
@
0x1000, 0x2800, 0x2800, 0x2800, 0x2800, 0x7C00, 0x4400, 0x4400, 0x0000, 0x0000,  //
A
0x7800, 0x4400, 0x4400, 0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x0000, 0x0000,  //
```

```
B
0x3800, 0x4400, 0x4000, 0x4000, 0x4000, 0x4000, 0x4400, 0x3800, 0x0000, 0x0000,  //
C
0x7000, 0x4800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4800, 0x7000, 0x0000, 0x0000,  //
D
0x7C00, 0x4000, 0x4000, 0x7C00, 0x4000, 0x4000, 0x4000, 0x7C00, 0x0000, 0x0000,  //
E
0x7C00, 0x4000, 0x4000, 0x7800, 0x4000, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000,  //
F
0x3800, 0x4400, 0x4000, 0x4000, 0x5C00, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000,  //
G
0x4400, 0x4400, 0x4400, 0x7C00, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000,  //
H
0x3800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x3800, 0x0000, 0x0000,  //
I
0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000,  //
J
0x4400, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4800, 0x4400, 0x0000, 0x0000,  //
K
0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x7C00, 0x0000, 0x0000,  //
L
0x4400, 0x6C00, 0x6C00, 0x5400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000,  //
M
0x4400, 0x6400, 0x6400, 0x5400, 0x5400, 0x4C00, 0x4C00, 0x4400, 0x0000, 0x0000,  //
N
0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000,  //
O
0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000,  //
P
0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x5400, 0x3800, 0x0400, 0x0000,  //
Q
0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4800, 0x4800, 0x4400, 0x0000, 0x0000,  //
R
0x3800, 0x4400, 0x4000, 0x3000, 0x0800, 0x0400, 0x4400, 0x3800, 0x0000, 0x0000,  //
S
0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000,  //
T
0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000,  //
U
0x4400, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x1000, 0x0000, 0x0000,  //
V
0x4400, 0x4400, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800, 0x0000, 0x0000,  //
W
0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x2800, 0x2800, 0x4400, 0x0000, 0x0000,  //
X
0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000,  //
Y
0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x4000, 0x7C00, 0x0000, 0x0000,  //
Z
0x1800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1800,  //
[
0x2000, 0x2000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x0000, 0x0000,  //
/
```

```
0x3000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x3000,  //
]
0x1000, 0x2800, 0x2800, 0x4400, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,  //
^
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFE00,  //
_
0x2000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,  //
`
0x0000, 0x0000, 0x3800, 0x4400, 0x3C00, 0x4400, 0x4C00, 0x3400, 0x0000, 0x0000,  //
a
0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800, 0x0000, 0x0000,  //
b
0x0000, 0x0000, 0x3800, 0x4400, 0x4000, 0x4000, 0x4400, 0x3800, 0x0000, 0x0000,  //
c
0x0400, 0x0400, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0000, 0x0000,  //
d
0x0000, 0x0000, 0x3800, 0x4400, 0x7C00, 0x4000, 0x4400, 0x3800, 0x0000, 0x0000,  //
e
0x0C00, 0x1000, 0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000,  //
f
0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0400, 0x7800,  //
g
0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000,  //
h
0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000,  //
i
0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0xE000,  //
j
0x4000, 0x4000, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4400, 0x0000, 0x0000,  //
k
0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x0000,  //
l
0x0000, 0x0000, 0x7800, 0x5400, 0x5400, 0x5400, 0x5400, 0x5400, 0x0000, 0x0000,  //
m
0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400, 0x0000, 0x0000,  //
n
0x0000, 0x0000, 0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800, 0x0000, 0x0000,  //
o
0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800, 0x4000, 0x4000,  //
p
0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0400, 0x0400,  //
q
0x0000, 0x0000, 0x5800, 0x6400, 0x4000, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000,  //
r
0x0000, 0x0000, 0x3800, 0x4400, 0x3000, 0x0800, 0x4400, 0x3800, 0x0000, 0x0000,  //
s
0x2000, 0x2000, 0x7800, 0x2000, 0x2000, 0x2000, 0x2000, 0x1800, 0x0000, 0x0000,  //
t
0x0000, 0x0000, 0x4400, 0x4400, 0x4400, 0x4400, 0x4C00, 0x3400, 0x0000, 0x0000,  //
u
0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x0000, 0x0000,  //
v
0x0000, 0x0000, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800, 0x0000, 0x0000,  //
```

```
w
0x0000, 0x0000, 0x4400, 0x2800, 0x1000, 0x1000, 0x2800, 0x4400, 0x0000, 0x0000,  //
x
0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000, 0x6000,  //
y
0x0000, 0x0000, 0x7C00, 0x0800, 0x1000, 0x2000, 0x4000, 0x7C00, 0x0000, 0x0000,  //
z
0x1800, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x1000, 0x1000, 0x1000, 0x1800,  //
{
0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,  //
|
0x3000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x1000, 0x1000, 0x1000, 0x3000,  //
}
0x0000, 0x0000, 0x0000, 0x7400, 0x4C00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,  //
~
};
FontDef Font_7x10 = { 6,10,Font7x10 };
```

ПРИЛОЖЕНИЕ Б