Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский Томский политехнический Университет»



Центр цифровых

образовательных технологий

15.03.06 «Мехатроника и робототехника»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛАБОРАТОРНАЯ РАБОТА №1

ЗНАКОМСТВО С SPI. ПРИМЕНЕНИЕ БИБЛИОТЕК ПРИ РАЗРАБОТКЕ ПО ПОД STM32

по дисциплине:

**Встраиваемые системы**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Исполнитель:** | |  | | | | |
| студент группы | | 8Е12 |  | Федоренко Д. В. |  | 02.11.2023 |
|  | |  |  |  |  |  |
| **Руководитель:** | |  | | | | |
| преподаватель | |  |  | Поберёзкин Н. И. |  | 02.11.2023 |
|  | |  |  |  |  |  |
|  |

Томск – 2023

# Оглавление

[Оглавление 2](#_Toc152205425)

[Цель работы 3](#_Toc152205426)

[Задание 3](#_Toc152205427)

[Описание протокола SPI 3](#_Toc152205428)

[Строение светодиодной матрицы LEDP10 4](#_Toc152205429)

[Практическая работа с матрицей LEDP10 5](#_Toc152205430)

[Схема подключения 9](#_Toc152205431)

[Используемые переменные 11](#_Toc152205432)

[Блок схемы алгоритмов 12](#_Toc152205433)

[Листинги кода 13](#_Toc152205434)

[Пример работы и ссылка на проект 23](#_Toc152205435)

[Выводы по работе 24](#_Toc152205436)

[Приложение А 25](#_Toc152205437)

# Цель работы

Научиться использовать последовательный синхронный стандарт передачи данных в режим полного дуплекса (SPI) и изучить строение LEDP10.

# Задание

Реализовать видео игру в жанре гонки с применением матрицы LEDP10 и использованием стандарта передачи данных SPI.

# Описание протокола SPI

Serial Peripheral Interface (SPI) — это последовательный, синхронный, полнодуплексный протокол передачи данных между главным (master) и несколькими ведомым/ведомыми (slave) устройством/устройствами.

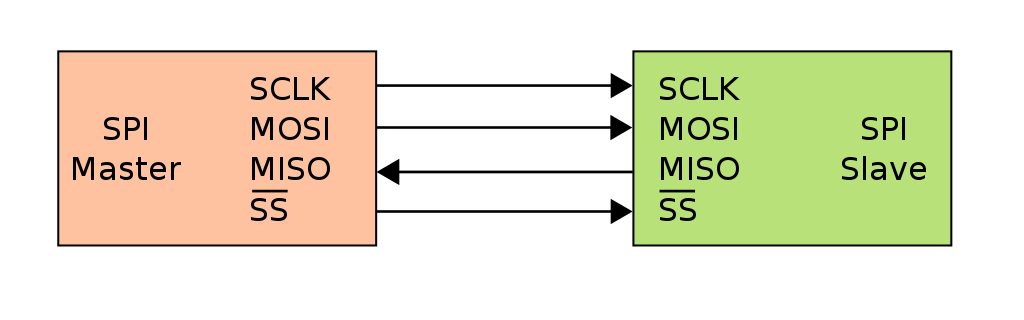


Рисунок 1- Подключение по протоколу SPI

В данном примере MOSI – канал передачи данных от ведущего устройства к ведомому. MISO – канал передачи данных от ведомого устройства к ведущем. SCLK – канал для передачи тактового сигнала от ведущего устройства к ведомому. SS – канал для выбора ведомого.

Так же иногда в схемах можно встречать иные обозначения тех или иных каналов.

Канал для передачи тактового сигнала необходим для синхронизации двух устройств между собой.

# Строение светодиодной матрицы LEDP10

Поиск информации в интернете позволяет найти принципиальную схему данной светодиодной матрицы.

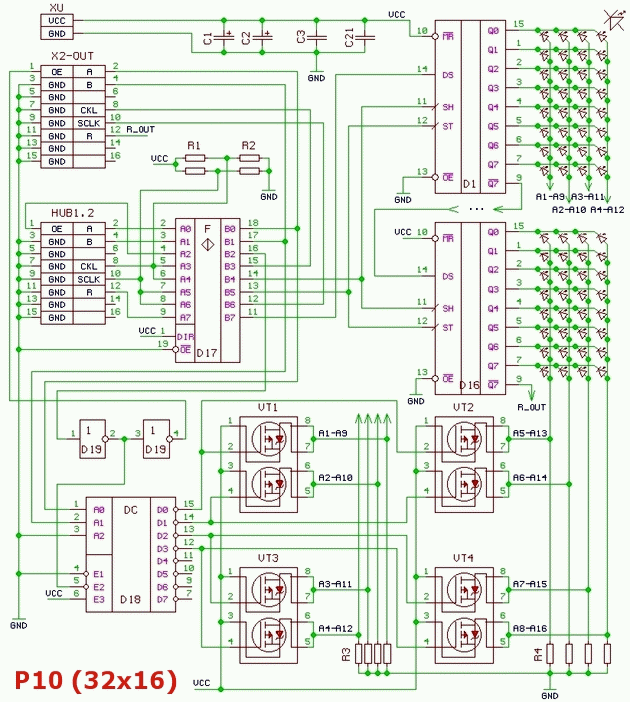


Рисунок 2- Принципиальная схема

HUB 1 – входной, в него будет происходить подключение STM32, так же есть HUB 2, который в теории является дублирующем первого порта, но он не принимает сигналы, а выдаёт, в него подключается шлейф, соединённый с следующей матрицей.

С хаба сигналы расходятся к элементу D17, являющимся не инвертирующий буффер.

# Практическая работа с матрицей LEDP10

Схема подключения STM32 к данной матрице через специальный шлейф происходит по следующему принципу.

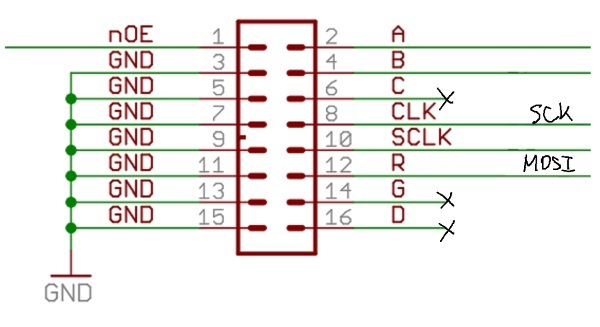


Рисунок 3- Схема подключения STM32 к HUB 1

Здесь приведена распиновка контактов на светодиодной матрице, подключение возможно осуществить напрямую или по средствам шлейфа.

Пин OE отвечает за подачу питания на матрицу, если на нём будет сигнал 0, то матрица гореть не будет, если же будет 1, то матрица зажгётся. Подача ШИМ на контакт позволяет регулировать яркость свечения матрицы, что может быть полезно при длительной работе с ней.

Пины A и B позволяют переключаться между строками матрицы в блоках путём путём комбинаций, с ними можно ознакомиться в таблице 1.

Пин R, (MOSI) , канал для передачи данных.

Пин CLK (SCK) осуществляет передачу тактового сигнала для синхронизации.

Пин SCLK отвечает за подачу сигнала о применении загруженных данных.

Таблица 1 – Связь комбинаций и номера строка.

|  |  |  |
| --- | --- | --- |
| Номер строки | A | B |
| 4 | 0 | 0 |
| 3 | 1 | 0 |
| 2 | 0 | 1 |
| 1 | 1 | 1 |

Структура матрицы следующая. Вся матрица имеет размерность 16 на 32 пикселя. Вся матрица разбита на ячейки размерностью 4 на 8 пикселей. То есть вся матрица состоит из 16-ти подобных ячеек.

Каждая строка в ячейки кодируется 1 байтом данных, где каждый бит отвечает за соответствующий светодиод. Бит может иметь лишь два состояния 0 и 1, при этом, светодиод загорается при значении 0, не горит при значении 1.

Ниже, на рисунке 4, представлена визуализация битов с строками

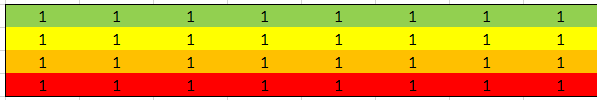


Рисунок 4- Положение битов в ячейке матрицы

Каждая ячейка (1-ца на рисунке 3) соответствует светодиоду, если в строках будет содержаться подобная битовая последовательность, то вся ячейка (матрицы) не будет гореть.

Зелёный цвет – четвёртая строка, жёлтый цвет – третья строка, оранжевый – вторая строка, красный – первая строка.

На изображении 5 представлена матрица, разбитная на 16-ть ячеек (высотой в 4 ячейки экселя и шириной в 1).

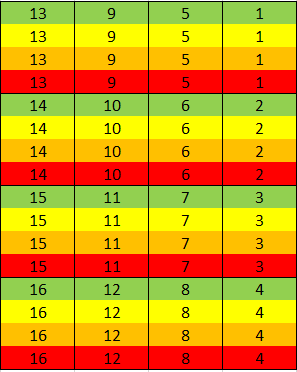


Рисунок 5- Положение байтов в матрице для загрузки

Цифра в ячейке обозначает номер байта, отвечающего за каждый из секторов. При этом стоит отметить, что загрузка происходит последовательно и с так называемым “проталкиванием” байтов.

Если мы хотим поместить информацию в 16-тую ячейку зелёного цвета, нам необходимо выставить пины A и B при отправке данных в положении 0 и 0. Первым направится байт, который мы хотим установить в ячейку 16, он будет находиться в ячейки 1, после мы отправляем последовательно ещё 15-ть байтов, чтобы протолкнуть первый отправляемый байт до позиции 16.

Переключившись на следующую комбинацию пинов А и В мы можем подтворить прошлый алгоритм, чтобы установить байт скажем в 16-тую жёлтую ячейку.

Если договориться, что мы всегда загружаем в любую из 4-рёх байтовых последовательностей (4 цвета на изображении) по 16 байт данных, то можно представить матрицу в следующем виде.

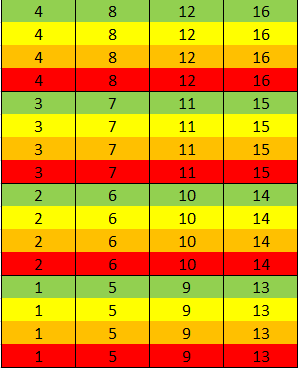


Рисунок 6- Матрица с загрузкой байтов

Где 1 – положение, где окажется первый байт данных, загружаемый в последовательность. А 16 – последний. С этой таблицей уже гораздо удобнее работать.

Дальше необходимо лишь переключаться между каналами А и B, заполняя их данными по 16 байт, чтобы поддерживать иллюзию статичного изображений.

Предположительно, если подключить ещё одну матрицу, то рисунок 5 можно будет продлить дальше, появятся ячейки с 17-тую по 32-ую. Принцип управления останется тем же, только нужно будет загружать уже не по 16-дь байт данных, а по 32 байта на строку.

## Схема подключения

На рисунке 2 можно увидеть скриншот из программы CubaMX, на которой показы все задействованные в проекте пины микроконтроллера.

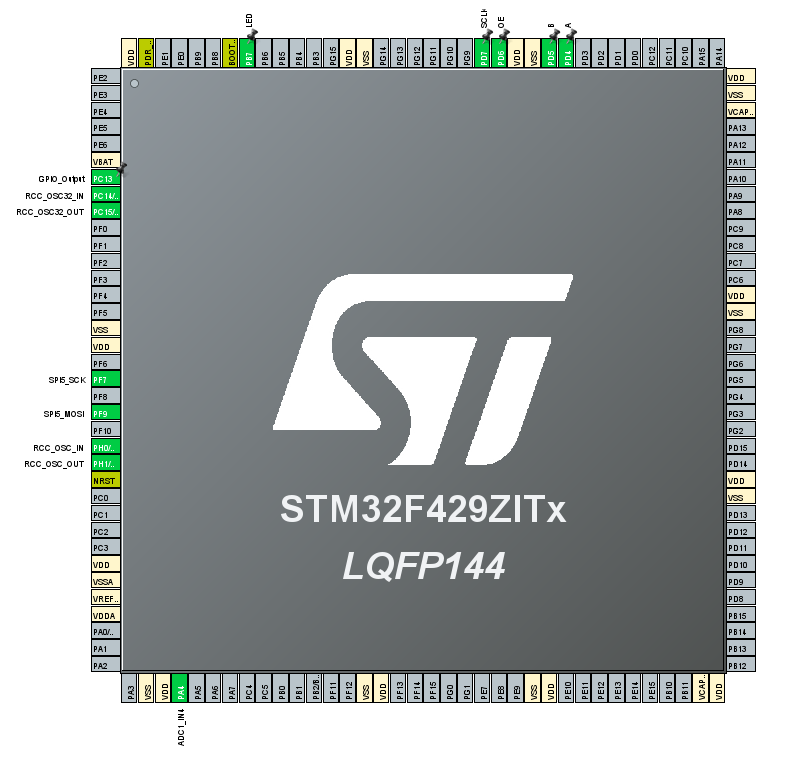


Рисунок 7 – Используемые пины микроконтроллера

В данном проекте не используются таймеры, ввиду расчёта на оставление за пользователем возможности динамического изменения скорости путём вращения реостата.

В таблице 2 возможно ознакомиться с используемыми пинами более подробно.

Таблица 2 – Задействованные входы и выходы

|  |  |  |
| --- | --- | --- |
| Номер вывода | Функция | Пользовательские метки |
| PD4 | Подача сигнала на матрицу в пин А | A |
| PD5 | Подача сигнала на матрицу в пин B | B |
| PA4 | Приём сигнала с реостата | Нет |
| PD6 | Подача сигнала включения/выключения панели | OE |
| PF7 | Генерация тактового сигнала | Нет |
| PF9 | Передача данных на матрицу | Нет |
| P14 | Вход для тактирования | Нет |
| P15 | Выход для тактирования | Нет |
| PH0 | Вход для тактирования | Нет |
| PH1 | Выход для тактирования | Нет |

## Используемые переменные

В таблице 3 представлена информация о всех переменных, встречающихся в проекте.

Часть из них используется только внутри функций, часть из них применяются по всему проекту.

Таблица 3 – Используемые переменные

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Назначение | Тип | Диапазон измерений |
| cadr\_cloud | Хранить информацию о текущем кадре облака | uint16\_t | 0….4 |
| cadr\_prov | Номер кадра облака внутри функции | uint16\_t | 0….4 |
| cadr\_show | Отчёт отрисованных кадров (для регулировки скорости) | uint16\_t | 0….16 |
| move | Смещение относительно базовой координаты | uint16\_t | 0….12 |
| point | Очки за прохождение препядствий | uint16\_t | 0…. |
| old\_car\_x | Координата машины на прошлом кадре | uint16\_t | 0….20 |
| x\_car | Координата машины на новом кадре | uint16\_t | 8….20 |
| flag | Метка о отрисовки самого первого кадра в игре | uint16\_t | 0….1 |
| j | Строка, с которой изменяется облако | uint16\_t | 0….6 |
| pos | Определение момента для замены кадра облака | uint16\_t | 0….4 |
| displ | Кадр для отрисовки | int | 0….1 |
| live | Состояние игрока | uint16\_t | 0….1 |
| Danger | Координата для отрисовки препятствия | uint16\_t | 0….18 |

Таблица 3 – Используемые переменные

|  |  |  |  |
| --- | --- | --- | --- |
| displ\_over | Заставка проигрыша | int | 0….1 |
| a | Регулировка скорости игры | uint16\_t | 0….16 |
| b | Счётчик препятствий для генерации | uint16\_t | 0….44 |
| g | Значение с АЦП | uint16\_t | 0….4095 |

## Блок схемы алгоритмов

Ниже приведены алгоритмы используемых в проекте функций и их взаимодействие между друг другом.

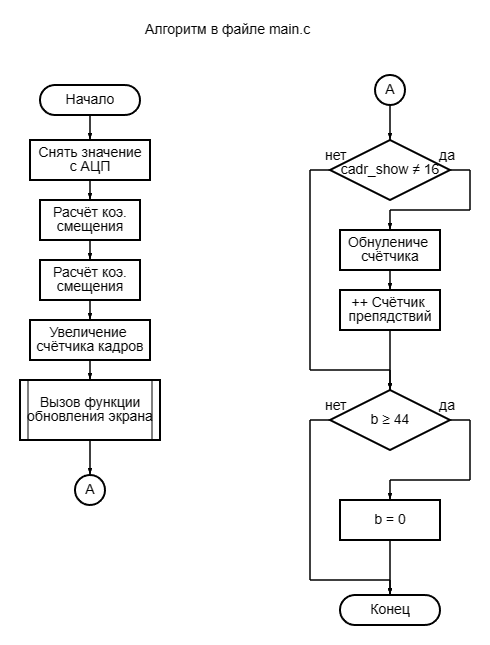


Рисунок 8- Алгоритм в main,c

Следующий алгоритм показывает работу кода, наисанного в разделе main.c. С дополнительными пояснениями можно ознакомиться внутри кода проекта, так как он содержит внутри себя раличные коментарии.

С алгоритмами всех остальных функций, показаных на рисунке 8, можно ознакомиться в приложении Б.

## Листинги кода

Ниже приведены листинги кода, в которых возможно ознакомиться с кодом, находящимся в проекте.

Листинг 1 – Код в файле main.c

|  |
| --- |
| uint8\_t cadr\_show = 0;  uint8\_t b = 0; //Количесто  **while** (1){  HAL\_ADC\_Start(&hadc1);  HAL\_ADC\_PollForConversion(&hadc1,10);  g = HAL\_ADC\_GetValue(&hadc1);  move = g/292;  HAL\_ADC\_Stop(&hadc1);  cadr\_show++;  Display\_Test(cadr\_show,b);  **if** (cadr\_show == 16) {  cadr\_show = 0;  b++;  };  //Тут число 44 - число элементов в массиве препядствия  **if** (b >= 44) {  b = 0;  }; |

В листинге 1 содержится обращение к АЦП, чтобы получать информацию об изменении положения реостата для дальнейшего перемещения персонажа игрока.

Листинг 2 – Код в файле display.c

|  |
| --- |
| **#include** "display.h"  **extern** uint8\_t test;  **extern** P;  uint16\_t cadr\_cloud = 1;  uint16\_t move = 0;  uint16\_t point = 0;  uint16\_t old\_car\_x = 0;  uint16\_t flag = 0;  uint16\_t j = 0;  uint16\_t pos = 1;  **int** displ[16][32] = {  {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, 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, 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, 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, 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, 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, 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, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  {1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1}  }; |

Листинг 2 содержит внутри себя объявление основных переменных, с которыми будет происходить дальнейшее взаимодействие.

Листинг 3 – Код в файле display.c

|  |
| --- |
| //Функция очистки матрицы  **void** **Clear\_matrix**(**void**){  **for** (uint8\_t i = 0; 4 > i; i++ ){  **for** (uint8\_t n = 0; 16 > n; n++ ){  Display\_Write\_1(0xFF, i);  }  }  }  //Алгоритм вывода информации  **void** **Display\_Write\_1**(uint8\_t pData, uint8\_t s) {  **switch**(s+1){  **case** 1:  HAL\_GPIO\_WritePin(GPIOD, A\_Pin, 1);  HAL\_GPIO\_WritePin(GPIOD, B\_Pin, 1);  **break**;  **case** 2:  HAL\_GPIO\_WritePin(GPIOD, A\_Pin, 0);  HAL\_GPIO\_WritePin(GPIOD, B\_Pin, 1);  **break**;  **case** 3:  HAL\_GPIO\_WritePin(GPIOD, A\_Pin, 1);  HAL\_GPIO\_WritePin(GPIOD, B\_Pin, 0);  **break**;  **case** 4:  HAL\_GPIO\_WritePin(GPIOD, A\_Pin, 0);  HAL\_GPIO\_WritePin(GPIOD, B\_Pin, 0);  **break**;  }  HAL\_GPIO\_WritePin(GPIOD, OE\_Pin, *GPIO\_PIN\_RESET*);  HAL\_GPIO\_WritePin(GPIOD, SCLK\_Pin, *GPIO\_PIN\_RESET*);  HAL\_SPI\_Transmit(&hspi5, (uint8\_t\*) &pData, 1, 10);  HAL\_GPIO\_WritePin(GPIOD, SCLK\_Pin, *GPIO\_PIN\_SET*);  HAL\_GPIO\_WritePin(GPIOD, OE\_Pin, *GPIO\_PIN\_SET*);  } |

В листинги 3 представлен код функции для очистки экрана и функции отправки одного байта информации на панель путём выбора комбинации пинов A и B в зависимости от желаемой строки для вывода.

Следом происходит отключение питания матрицы (OE), отключение контакта SLCK, для загрузки данных, включение пина говорит о завершении загрузки данных, после чего происходи подача питания на матрицу.

Функция очистки экрана необходима для затирания всей выводимой информации путём заполнения всей матрицы 1-ми.

Листинг 4 – Код в файле display.c

|  |
| --- |
| **void** **Show\_matrix**( **int** displ[16][32] ){  //Очистка экрана  Clear\_matrix();//Возможно есть смысл удалить  //Матрица для байтов  **int** bytesMatrix[16][4] = {  {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}};  //Алгоритм сборки байтовой матрицы  **for** (**int** i = 0; i < 16; i++) {  **for** (**int** j = 0; j < 4; j++) {  uint8\_t byte = 0;  **for** (**int** k = 0; k < 8; k++) {  // Собираем байт из битов, начиная с самого младшего  byte |= (displ[i][j \* 8 + k] << (7-k)); //лучший преобразователь  }  bytesMatrix[i][j] = byte;  }  }  //Алгоритм выгрузки байтов на экран.  **for** (uint8\_t n = 0; 4 > n; n++ ) {  **for** (**int** x = 0; x < 4; x++){  **for** (**int** y = 15-n; y > -1; y = y-(4)){  Display\_Write\_1(bytesMatrix[y][x],n);  }  }  HAL\_Delay(1);  Clear\_matrix(); //Возможно есть смысл удалить  }  } |

В листинге 4 содержится пример преобразования матрицы битов (которая наглядно отображает информацию, которая будет выведена на экран) в байтовую матрицу, которая будет загружена на панель с помощью специального алгоритма, суть которого заключается в правильном обходе матрицы байтов, чтобы каждый байт информации оказался на нужной позиции при последовательной загрузки.

Листинг 5 – Код в файле display.c

|  |
| --- |
| **int** **Callyzi\_Car**(**void**) {  uint8\_t live = 1;  //Проверяем, была ли машина до этого или это первая итерация  **if** (old\_car\_x != 0) {  flag = 1;  }  //Затираем прошлое положение машины, если она была  **if** (flag == 1) {  **for** (**int** i = old\_car\_x-1; i < old\_car\_x + 2; i++) {  displ[14][i] = 1;  }  **for** (**int** i = old\_car\_x-2; i < old\_car\_x + 3; i++) {  displ[15][i] = 1;  }  }  //Определяем новые координаты центра машины  uint8\_t x\_car = 8+move; //предел - 23, то есть move - от 0 до 16  old\_car\_x = x\_car;  //Определяем координаты всей машины для дальнейшем отрисовки и проверяем, все ли свободны (кроме столбца 15)  **for** (**int** i = x\_car-1; i < x\_car + 2; i++) {  **if** ((displ[14][i] == 0)&(i != 15)) {  live = 0;  }  displ[14][i] = 0;  }  **for** (**int** i = x\_car-2; i < x\_car + 3; i++) {  **if** ((displ[15][i] == 0)&(i != 15)) {  live = 0;  }  displ[15][i] = 0;  }  flag = 0;  **return** (live);  } |

В листинге 5 описывается возможность определения коллизии машины. Сперва происходит проверка, первая ли это отрисовка машины или нет. Если это не первая отрисовка, то тогда происходит затирание прошлого положения машины, используя старую координату центра машины.

После происходит обновление актуальной координаты машины с помощью переменной move и базовым положением машины 8. Следом происходит копирование актуальной переменной с координатой в переменную, хранящую старую координату, при следующей итерации это позволит точечно затереть старое положение машины.

Следом происходит отрисовка машины на новом положении параллельно с проверкой, нет ли там препятствия, которое может убить персонажа игрока. Однако, нахождение в 15-том столбце матрицы посторонних объектов не убивает персонажа, так как там постоянно находится разметка. Препятствия там не появляются, чтобы не сбивать разметку.

Если же машина должна быть в месте, где уже есть препятствие, то происходит отображение флага live на 0, что в дальнейшем запустит код с проигрышем надписи “game over”. Функция возвращает переменную live.

Листинг 6 – Код в файле display.c

|  |
| --- |
| **int** **Danger\_offset**(**void**) {  uint8\_t live = 1;  //Проверка опасных зон, если будет смещение, то окажется ли что-то на координатах машины  uint16\_t x\_car = 8+move;  **if** (((displ[13][x\_car] == 0) || (displ[13][x\_car + 1] == 0)  || (displ[13][x\_car - 1] == 0) || (displ[14][x\_car - 2] == 0)  || (displ[14][x\_car + 2] == 0))  & !(((x\_car == 15) || (x\_car + 1 == 15) || (x\_car + 2 == 15)  || (x\_car - 1 == 15) || (x\_car - 2 == 15)))) {  live = 0;  **return** (live);  }  //Алгоритм сдвига всех препядствий  **for** (**int** i = 15; i >= 7; i--) {  **for** (**int** j = 9; j < 22; j++) {  **if** ((displ[i][j] == 0)&(j != 15)) {  displ[i][j] = 1;  **if** (i+1 < 16) {  displ[i+1][j] = 0;  point++;  }  }  }  }  **return**(live);  } |

В листинге 6 описывается функция, назначение которой смещение препятствий и проверка зон вокруг персонажа игрока и, если там есть препятствие, то, при смещении, оно станет причиной проигрыша игрока, следовательно, игрока проиграл уже на этом этапе и дальнейшее смещение препятствий не имеет смысла. Функция вернёт переменную live, с значением 0 – проигрыш.

Если же препятствий нет около игрока, то происходит смешение всех препятствий начиная снизу вверх на одну ячейку вниз. Движение снизу вверх необходимо, так как иначе за одну итерацию все смещения происходили бы с места возникновения вплоть до самого конца. После смещения происходит возврат переменной live с значением 1 – не проиграл.

Листинг 7 – Код в файле display.c

|  |
| --- |
| **void** **Game\_Over**() {  **for** (**int** i = old\_car\_x-1; i < old\_car\_x + 2; i++) {  displ[14][i] = 1;  }  **for** (**int** i = old\_car\_x-2; i < old\_car\_x + 3; i++) {  displ[15][i] = 1;  }  **int** displ\_over[16][32] = {  {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, 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, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 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, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 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, 1, 1, 1, 1, 1, 1, 1}};  uint16\_t a = 1000;  **while** (a > 2) {  Show\_matrix(displ\_over);  a--;  }} |

Листинг 7 описывается процесс завершения игры проигрышем, если игрок сталкивается с препятствием из-за неудачного смещения машины или из-за смещение преграды вниз.

Сперва происходит затирание положения машины, чтобы можно было начать игру снова.

После вводится экран завершения игры, происходит задержка, после чего игра запускается заново, обнуляя все значимые переменные.

Листинг 8 – Код в файле display.c

|  |
| --- |
| **void** **Way\_Run**(**int** displ[16][32]) {  //Определяем текущий кадр, чтобы верно сдвинуть разметку  **switch**(displ[7][15]+displ[8][15]){  **case** 0:  //Был кадр 2, значит  //Вывод кадра 3  displ[7][15] = 1;  displ[8][15] = 0;  displ[9][15] = 0;  displ[10][15] = 1;  displ[11][15] = 0;  displ[12][15] = 0;  displ[13][15] = 0;  displ[14][15] = 1;  displ[15][15] = 1;  **break**;  **case** 1:  **if** (displ[7][15] == 1){  //Был кадр 3, значит  //Вывод кадра 1  displ[7][15] = 0;  displ[8][15] = 1;  displ[9][15] = 0;  displ[10][15] = 0;  displ[11][15] = 1;  displ[12][15] = 1;  displ[13][15] = 0;  displ[14][15] = 0;  displ[15][15] = 0;  }  **else** {  //Вывод кадра 2  displ[7][15] = 0;  displ[8][15] = 0;  displ[9][15] = 1;  displ[10][15] = 0;  displ[11][15] = 0;  displ[12][15] = 1;  displ[13][15] = 1;  displ[14][15] = 0;  displ[15][15] = 0;  }  **break**;  }  } |

Листинг 8 описывает алгоритм изображения движения разметки дорожной полосы. Определение кадра происходит по первым двум пикселям разметки. После определения кадра, происходит замена на следующий. Так происходит каждую итерацию.

Листинг 9 – Код в файле display.c

|  |
| --- |
| **void** **Display\_Test**(uint8\_t a, uint8\_t b) {  Show\_matrix(displ);  **if** (a == 16) {  Way\_Run(displ);  Cloud\_2(cadr\_cloud);  //Счётчик кадров облаокв и задержка вывода ( смещение должно быть медленее, чем дви  //жение дорожной разметки)  **if** (pos == 4){  pos = 0;  **if** (cadr\_cloud == 4) {  cadr\_cloud = 1;  }  **else** {  cadr\_cloud++;  }  }  pos++;  Danger\_spawn(b);  Callyzi\_Car(); //Можно попробовать удалить  **if** ((Danger\_offset() == 0)||(Callyzi\_Car() == 0)){  Game\_Over();  displ[16][32] = {  {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, 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, 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, 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, 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, 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, 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, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  {1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1},  {1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1}  };  move = 0;  flag = 0;  old\_car\_x = 0;  }  }  } |

Листинг 9 описывает основную функцию, вызываемую из файла main.c, в ней в определённой последовательности вызываются все описанные ранее функции.

В приложении А возможно ознакомиться с листингом кода для отрисовки облаков.

Сперва происходит отрисовка кадра, после, каждый 16-ый отрисованный кадр происходит изменение положение разметки и отрисовка облака, отрисовка нового кадра облака происходит каждый 64-ый кадр.

Следом происходит сдвиг всех препятствий, после него – отрисовка нового положения машины.

Потом происходит проверка условий, не проиграл ли игрок, если он проиграл, то вызывается функция проигрыша, а после очищается экран, путём замены его на начальный игровой. Поле происходит обнуление всех важных переменных.

## Пример работы и ссылка на проект

С видео работы игры возможно ознакомиться по [ссылке](https://drive.google.com/drive/folders/1w_eB4Z3NGAlPlP0F2J-kuE7DudRr45T-?usp=sharing).

Здесь приведена [ссылка](https://github.com/GreyPaldin/LB_1_ORKMIR_) на GITHAB, где возможно ознакомиться с исходником проекта и частью сопутствующей документации.

В случай ознакомления с документом в печатаном виде, предусмотрен QR код на репрезторий.



Рисунок 9- QR код на репрезиторий

# Выводы по работе

В рамках лабораторной работы была изучена схема светодиодной матрицы LEDP10, особенности протокола общения SPI. Реализована игра гонок на базе отладочной платы Nuсleo и светодиодной матрицы.

# Приложение А

(Обязательное)

Листинг кода для отрисовки облаков

|  |
| --- |
| **void** **Cloud\_2**(**int** cadr\_prov) {  **switch**(cadr\_prov){  **case**(1):  //Чистка прошлой области\_2 с кадра 4  j = 1;  **for** (**int** i = 19; i < 30; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 18; i < 32; i++) {  displ[j - 1][i] = 1;  }  //Отрисовка нового облака\_2 кадра 1  j = 6;  **for** (**int** i = 22; i < 30; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 23; i < 28; i++) {  displ[j-1][i] = 0;  }  //Затирание облака\_1 на кадре 2  j = 4;  **for** (**int** i = 3; i < 5; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 2; i < 6; i++) {  displ[j - 1][i] = 1;  }  **for** (**int** i = 2; i < 8; i++) {  displ[j - 2][i] = 1;  }  //Отрисовка облака\_1 на кадре 3  j = 3;  **for** (**int** i = 0; i < 10; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 1; i < 8; i++) {  displ[j - 1][i] = 0;  }  **for** (**int** i = 2; i < 6; i++) {  displ[j - 2][i] = 0;  }  **for** (**int** i = 2; i < 5; i++) {  displ[j - 3][i] = 0;  }  **break**;  **case**(2):  //Чистка прошлой области\_2 с кадра 1  j = 6;  **for** (**int** i = 22; i < 30; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 23; i < 28; i++) {  displ[j-1][i] = 1;  }  //Отрисовка нового облака\_2 кадра 2  j = 4;  **for** (**int** i = 21; i < 31; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 22; i < 29; i++) {  displ[j-1][i] = 0;  }  **for** (**int** i = 23; i < 28; i++) {  displ[j-2][i] = 0;  }  //Чистка облака\_1 с кадре 3  j = 3;  **for** (**int** i = 0; i < 10; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 1; i < 8; i++) {  displ[j - 1][i] = 1;  }  **for** (**int** i = 2; i < 6; i++) {  displ[j - 2][i] = 1;  }  **for** (**int** i = 2; i < 5; i++) {  displ[j - 3][i] = 1;  }  //Отрисовка облака\_1 с кадре 4  j = 1;  **for** (**int** i = 0; i < 12; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 0; i < 10; i++) {  displ[j-1][i] = 0;  }  **break**;  **case**(3):  //Чистка прошлой области\_2 с кадра 2  j = 4;  **for** (**int** i = 21; i < 31; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 22; i < 29; i++) {  displ[j-1][i] = 1;  }  **for** (**int** i = 23; i < 28; i++) {  displ[j-2][i] = 1;  }  //Отрисовка нового облака\_2 кадра 3  j = 3;  **for** (**int** i = 18; i < 32; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 19; i < 30; i++) {  displ[j-1][i] = 0;  }  **for** (**int** i = 20; i < 28; i++) {  displ[j-2][i] = 0;  }  **for** (**int** i = 22; i < 28; i++) {  displ[j-3][i] = 0;  }  //Чистка прошлой области\_1 с кадра 4  j = 1;  **for** (**int** i = 0; i < 12; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 0; i < 10; i++) {  displ[j - 1][i] = 1;  }  //Орисовка нового облака\_1 кадр 1  j = 6;  **for** (**int** i = 2; i < 7; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 3; i < 5; i++) {  displ[j - 1][i] = 0;  }  **break**;  **case**(4):  //Затирание прошлого облака\_2 кадр 3  j = 3;  **for** (**int** i = 18; i < 32; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 19; i < 30; i++) {  displ[j - 1][i] = 1;  }  **for** (**int** i = 20; i < 28; i++) {  displ[j - 2][i] = 1;  }  **for** (**int** i = 22; i < 28; i++) {  displ[j - 3][i] = 1;  }  //Отрисовка облака\_2 кадра 4  j = 1;  **for** (**int** i = 19; i < 30; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 18; i < 32; i++) {  displ[j - 1][i] = 0;  }  //Затирание облака\_1 с кадра 1  j = 6;  **for** (**int** i = 2; i < 7; i++) {  displ[j][i] = 1;  }  **for** (**int** i = 3; i < 5; i++) {  displ[j - 1][i] = 1;  }  //Отрисовка облака\_1 на кадре 2  j = 4;  **for** (**int** i = 3; i < 5; i++) {  displ[j][i] = 0;  }  **for** (**int** i = 2; i < 6; i++) {  displ[j - 1][i] = 0;  }  **for** (**int** i = 2; i < 8; i++) {  displ[j - 2][i] = 0;  }  }  } |

# Приложение Б

(Обязательное)

Алгоритмы программы

