Санкт-Петербургский политехнический университет Петра Великого
Институт металлургии, машиностроения и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Программирование на языках высокого уровня
Тема: Создание ПО для трехколесного робота

Выполнил студент гр. 3331506/10401 Папасимеониди А.С.

Преподаватель Ананьевский М.С.

«___»_____ 2024 г.

Санкт-Петербург
2024

Содержание

1. Формулировка задачи, которую решает студент

Задача заключается в создании ПО для трехколесного робота. Задача студента на начальном этапе создать драйвера для работы микроконтроллера с экраном SSD1306 (вывод бинаризированных изображений, текста, различных фигур), адресной светодиодной лентой WS2812B (должна гореть различными цветами, мигать, изображать анимацию).

Ссылка на проект с экраном в гите: https://github.com/simeonidi03/hal

Ссылка на проект со led в гит: https://github.com/simeonidi03/Led_strip_for403a

2. Среда разработки и микроконтроллер

В связи с изменившимися логистическими цепочками, микроконтроллеры серии STM32 стали недоступны для студенческих проектов в связи с высокой стоимостью (1600 рублей за микроконтроллер). Для создания проекта был использован аналог STM32 от компании Artery. Я работаю с платой для стартовой отладки (схожа с Arduino), называется плата Artery-START-F413.

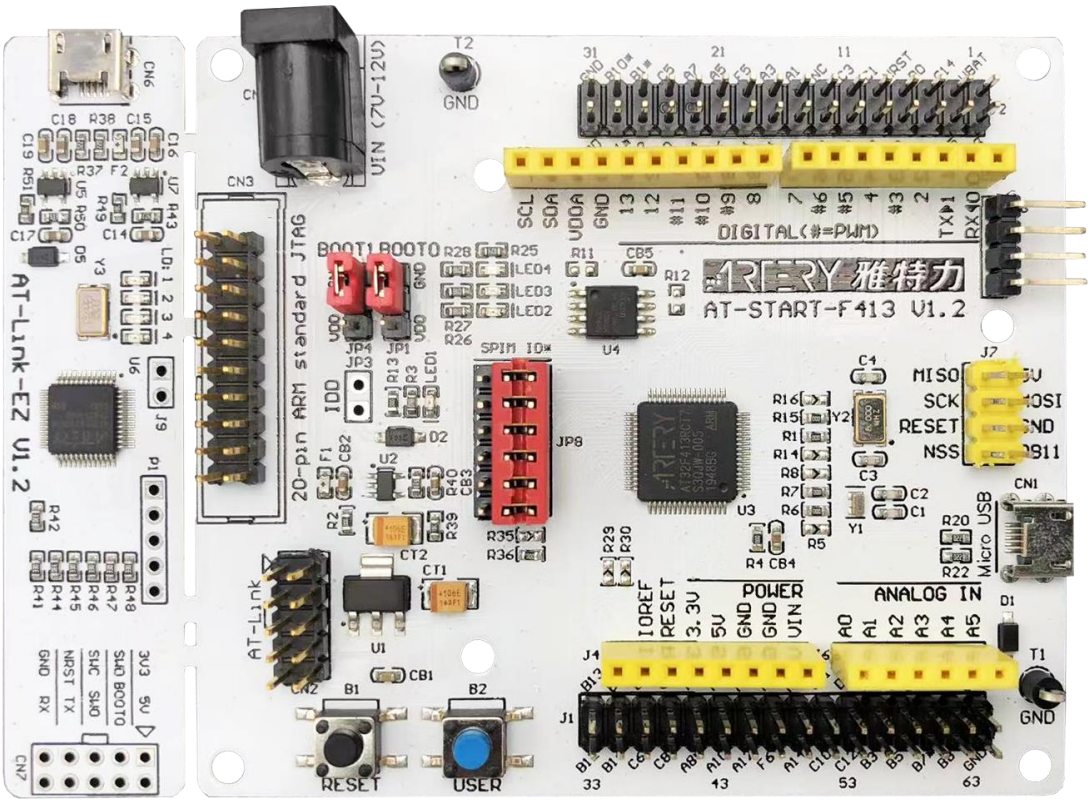

Рисунок 2.1 – Фотография платы

В данную плату встроен программатор (левая часть платы). Работу произвожу в проприетарной IDE под название AT32IDE. Также для программирования данной платы возможно использование Keil 5 или 4

версии. Доступен кодогенератор (аналог CubeMX от STM). IDE базируется на Eclipse.

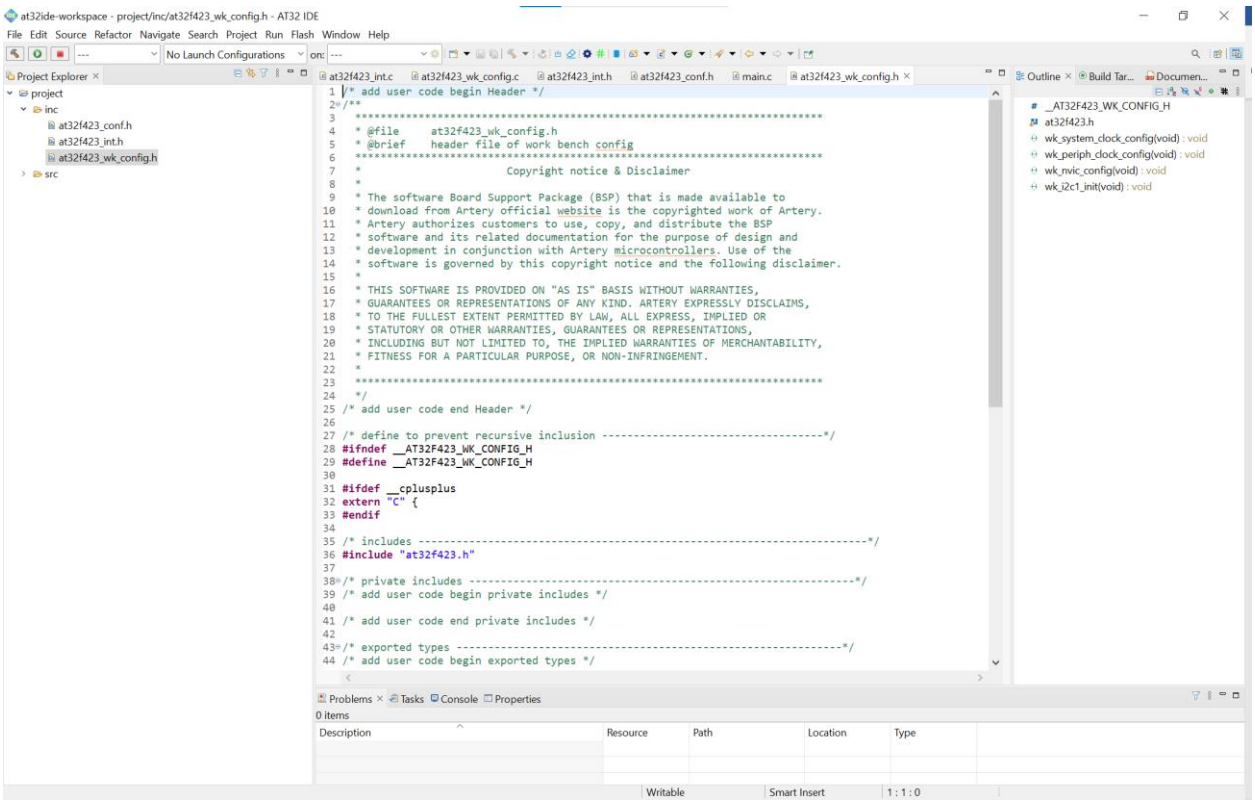

Рисунок 2.2 – AT32IDE

На этом краткое описание среды работы заканчивается.

3. Словесное описание алгоритма работы дисплея SSD1306

Дисплей с микроконтроллером соединяется по шине I2C. В идеологии этой шины есть главное и управляемое устройство. Главным будет являться микроконтроллер, управляемым будет экран. Шина I2C имеет две линии передачи данных.

Линия SCL – линия для тактирования (т.е. управления передачей данных и согласования всех устройств между собой).

Линия SDA – передача данных.

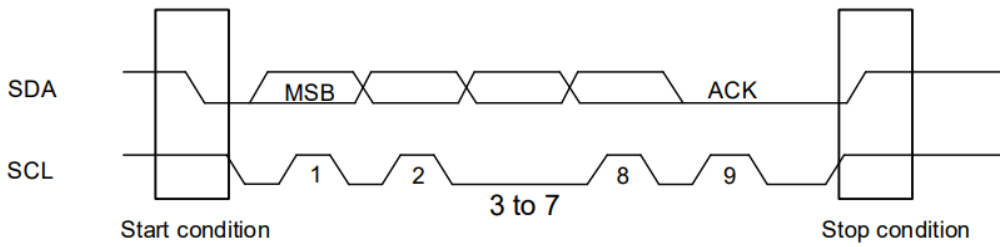

Рисунок 3.1 – Пример передачи данных по I2C

Начальное условие: Когда SCL установлен на высоком уровне, SDA переключается с высокого на низкий

Условие остановки: Когда SCL установлен на высоком уровне, SDA переключается с низкого на высокий.

В данном проекте я не использую прямой доступ к памяти (DMA), так как установка картинки на экран будет производиться в начале, когда процессор не будет загружен другими задачами.

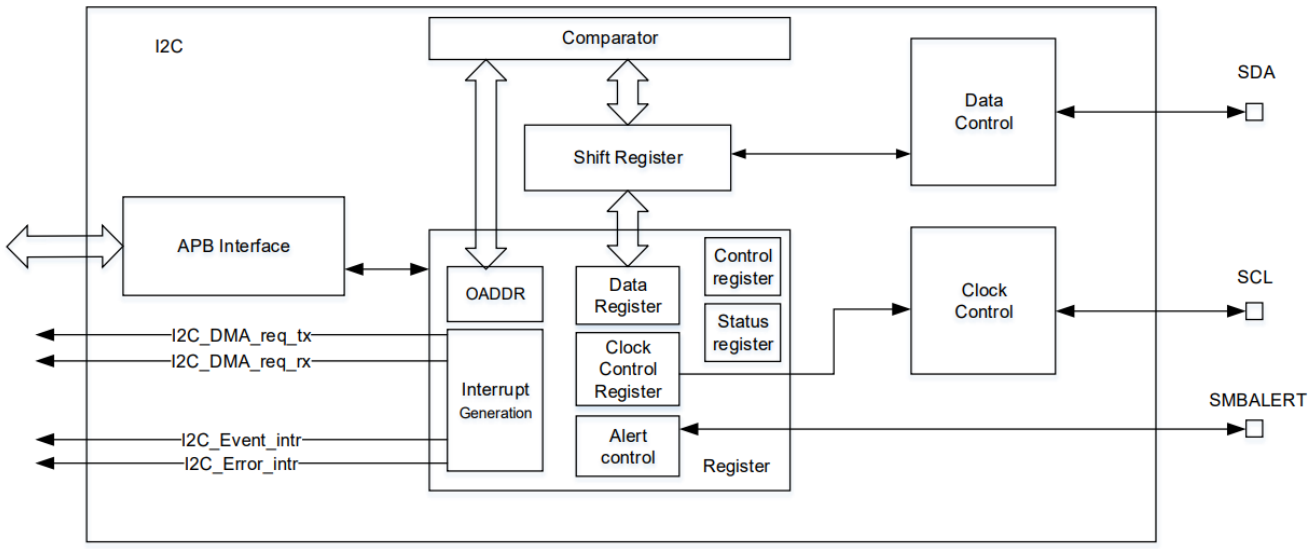

Рисунок 3.2 – Схема работы шины I2C

Пояснения:

Shift register – сдвиговый регист

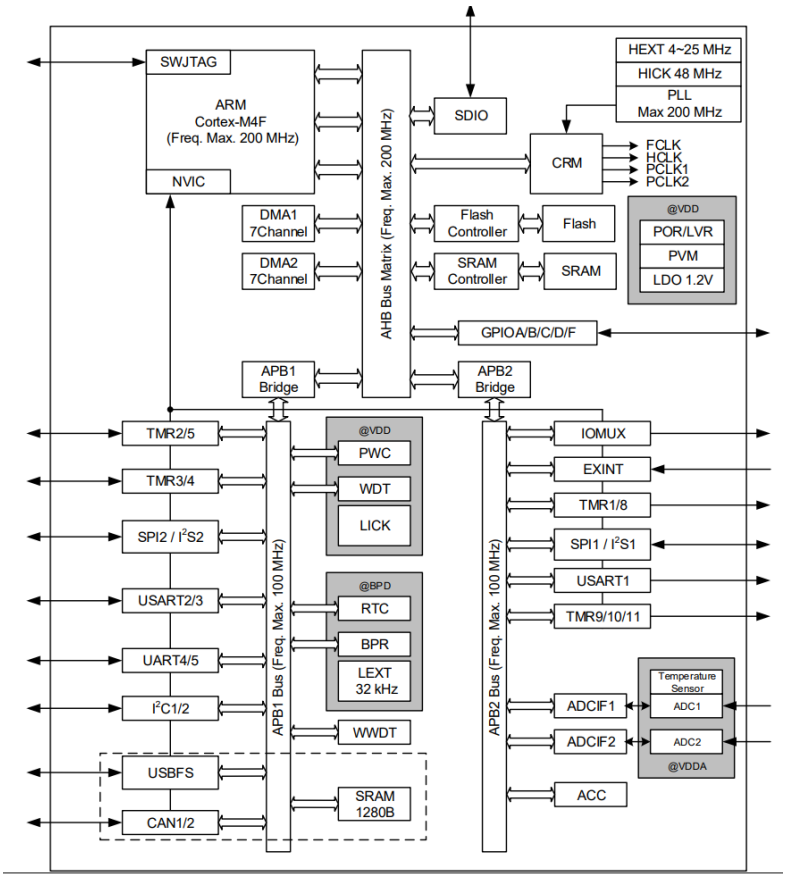

Рисунок 3.3 – Архитектура микроконтроллера

Шина I2C берёт тактирование с шины APB1 через шину AHB. Максимальная частота данной серии микроконтроллеров равна 200 МГц.

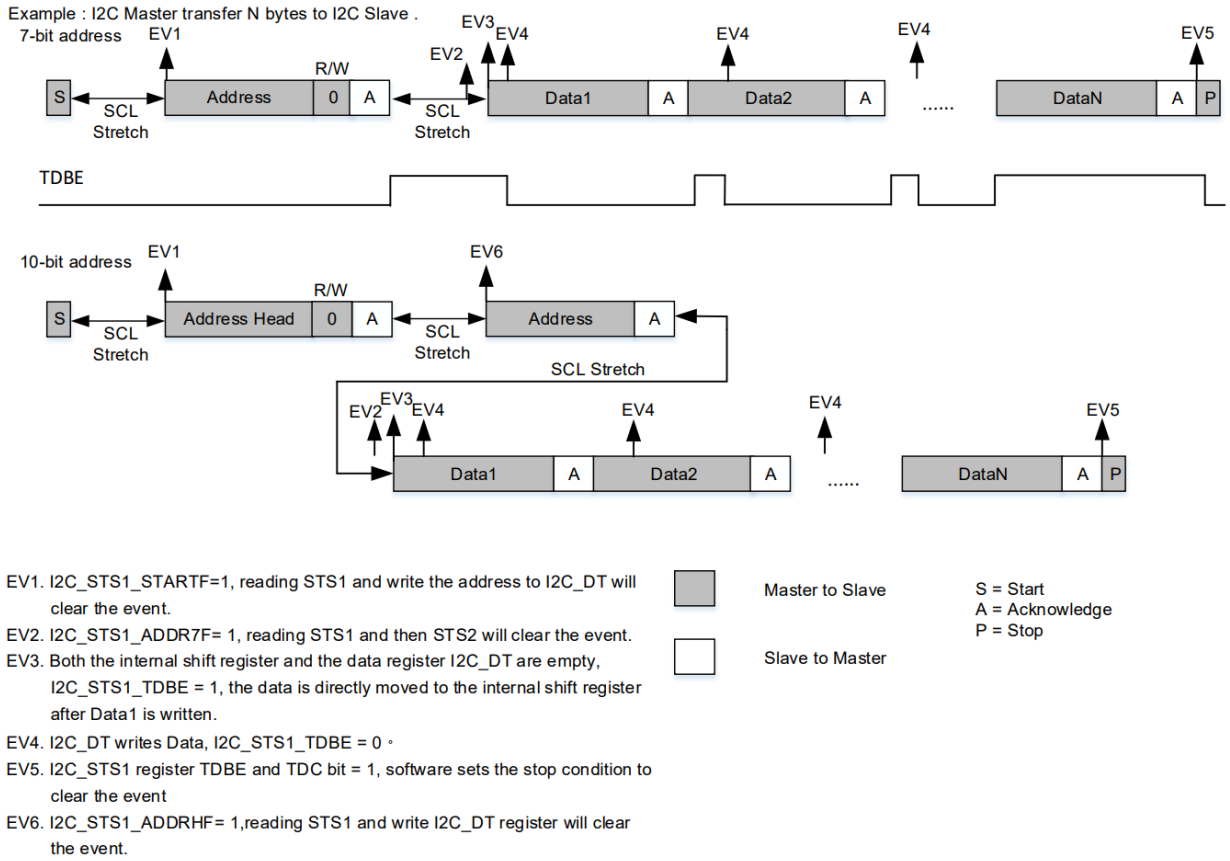

Рисунок 3.4 – Схема работы шины I2C

Сначала в шину передаётся адрес устройства (slave). В нашем случае это 0x3C (16-ричная система).

Также пару слов скажу о библиотеках в экосистеме Artery. Они используют union – объединения. Это участок памяти, который используется несколькими разными переменными, которые могут быть даже различного типа. Часто используются макросы (стандартно для библиотек для встроенных систем). Используются шаблоны структур для упрощения кода (без использования ключевого слова struct).

Ход работы:

- 1. Для использования пинов нужно настроить пин с помощью GPIO. Изначально пытался реализовать с помощью переписывания библиотеки, написанной под STM32, затем изучил аналог библиотеки HAL в Artery, с помощью неё настроил порты 6 и 7 GPIOB на SDA и SCL соответственно. Порты настроены в режиме GPIO_MODE_MUX, что обозначает порты как мультиплексоры. Этот режим позволяет использовать один и тот же порт GPIO для различных целей, переключаясь между ними с помощью ПО.

- Мы используем I2C1 в основном режиме. Дисплей не будет передавать какие-либо данные на микроконтроллер. I2C будет работать в быстром режиме, 400 кГц.
- Параметр DutyCycle, определяющий коэффициент заполнения, те времени в течении которого сигнал на шине данных будет на высоком уровне, ставим в режим 16/9.
- Также отмечу, что в библиотеке данного микроконтроллера используется идеология структурного выбора, в отличии от STM, где зачастую используют побитовые операции.
- Для начала передачи по I2C мы должны установить бит START (см. рис. 3.4 в единицу). Это запустит передачу данных (при успешном подтверждении адреса Slave устройством).

Для программирования микроконтроллером используют различные идеологии. Любо программирование с помощью встроенных функций (часто используется начинающими), либо программирование на уровне битов.

Рассмотрим регистр ctrl1_bit. Это структура с union внутри. Эта структура позволяет управлять передачей данных по I2C. Она позволяет обращаться к отдельным флагам управления для удобной постановки и снятия флагов настройки I2C. Каждый бит в этой структуре соответствует определенной настройке или флагу управления в I2C, позволяя программисту легко манипулировать отдельными настройками шины через их битовое представление в регистре ctrl1.

NB: побитовое ИЛИ отличается от обычной логической операции ИЛИ.

```
a |= b;
```

Побитовое ИЛИ представляет собой логические операции между переменными в их двоичном выражении. Сравниваются значения каждого разряда данных чисел по правилам обычного логического ИЛИ.

Для оценки статуса передачи используют статусный регистр. Статусный регистр представляет собой выделенную область памяти, в которой каждый бит отвечает за состояние передачи в I2C. Для начала передачи по I2C используют стартовый бит (I2C_SR1_SB). Он запускают передачи и при ответе контроллера дисплея (отправки подтверждения) начинается передача данных на шину.

Часть встроенной библиотеки I2C от компании Artery представлена ниже:

```
union
  {
    __IO uint32_t sts1;
    struct
    {
      __IO uint32_t startf              : 1; /* [0] */
      __IO uint32_t addr7f              : 1; /* [1] */
      __IO uint32_t tdc                 : 1; /* [2] */
      __IO uint32_t addrhf              : 1; /* [3] */
      __IO uint32_t stopf               : 1; /* [4] */
      __IO uint32_t reserved1           : 1; /* [5] */
      __IO uint32_t rdbf                : 1; /* [6] */
      __IO uint32_t tdbe                : 1; /* [7] */
      __IO uint32_t buserr              : 1; /* [8] */
      __IO uint32_t arlost              : 1; /* [9] */
      __IO uint32_t ackfail             : 1; /* [10] */
      __IO uint32_t ouf                 : 1; /* [11] */
      __IO uint32_t pecerr              : 1; /* [12] */
      __IO uint32_t reserved2           : 1; /* [13] */
      __IO uint32_t tmout               : 1; /* [14] */
      __IO uint32_t alertf              : 1; /* [15] */
      __IO uint32_t reserved3           : 16; /* [31:16] */
    } sts1_bit;
        };
```

Код представляет собой цикл while, в котором проверяется не отправили ли мы стартовый бит в шину.

Если подтверждение успешно отправлено (I2C_CR1_ACK = = 1), значит, что ведомое устройство (контроллер экрана) ожидает получения следующего пакета данных.

В библиотеках компании Artery бит, отвечающий за генерацию подтверждений (ACK) в объединении находится под именем acken и расположен в структуре ctrl1_bit (представлена ниже).

```
  union
  {
    __IO uint32_t ctrl1;
    struct
    {
      __IO uint32_t i2cen               : 1; /* [0] */
      __IO uint32_t permode             : 1; /* [1] */
      __IO uint32_t reserved1           : 1; /* [2] */
      __IO uint32_t smbmode             : 1; /* [3] */
      __IO uint32_t arpen               : 1; /* [4] */
      __IO uint32_t pecen               : 1; /* [5] */
      __IO uint32_t gcaen               : 1; /* [6] */
      __IO uint32_t stretch             : 1; /* [7] */
      __IO uint32_t genstart            : 1; /* [8] */
      __IO uint32_t genstop             : 1; /* [9] */
      __IO uint32_t acken               : 1; /* [10] */
      __IO uint32_t mackctrl            : 1; /* [11] */
      __IO uint32_t pecten              : 1; /* [12] */
      __IO uint32_t smbalert            : 1; /* [13] */
      __IO uint32_t reserved2           : 1; /* [14] */
      __IO uint32_t reset               : 1; /* [15] */
```

```
    __IO uint32_t reserved3            : 16;/* [31:16] */
  } ctrl1_bit;
};
```

Мы используем режим I2C при котором микроконтроллер является ведущим, а контроллер экрана – ведомым.

`I2C_OADR1_ADDO` представляет собой бит в регистре расширенного адреса. Обычно используется для управления битом адреса. Когда устройство I2C в режиме мастера передаёт вой адрес на шину, бит ADDO указывает, будет ли в конце установлен 0 или 1. В контексте спецификации протокола I2C, этот бит предназначен для указания операции чтения или записи.

`I2C_SR1_TXE` (Transmit Data Register Empty) является одним из битов в регистре состояния I2C (SR1).

Бит `I2C_SR1_BTF` (Byte Transfer Finished) это бит в регистре состояния I2C (SR1) и используется для определения завершения передачи байта. Когда бит “BTF” устанавливается в 1, это означает, что байт передан полностью (считан мастером и записан на шину).

После завершения передачи текущего байта BTF устанавливается в 0, что теперь можно подготовить следующий байт для передачи или принять следующий байт данных.

`I2C_SR1_BTF` указывает на завершение передачи текущего байта.

`I2C_SR1_TXE` указывает на готовность к передаче нового байта.

Немного про реализацию в микроконтроллерах Artery. В регистре статуса есть бит tdc (transfer data complete). Если установлен 0, то передача информации еще не завершена.

Тактирование шины I2C может происходит либо по шине APB1, APB2.

APB(Advanced Peripheral Bus) – протокол ARM, маломощный недорогой протокол шины для подключения низкоскоростных периферийных устройств в конструкции системы на кристалле. APB – это простая шина с протоколом с одним краем (тактированием), что упрощает его реализации и снижает сложность системы.

Деление частоты (тактовой) I2C достигается путём установки ctrlfreq[7:0] в ctrl2_bit регистре. Минимальная частота меняется от режима, по крайней мере, с 2 МГц в стандартном режиме и 4 МГц в быстром режиме.

2. Режим работы

I2C может работать как master, а также как slave.

Переключение с режима master в режим slave, и наоборот также поддерживается.

Когда бит genstart == 1 (начальное условие активировано) I2C шина переключается из slave режима в master, и возвращается обратно в slave mode автоматически в конце передачи данных (условие остановки вызвано).

3. Алгоритм работы:

- Создание стартового условия
- Передача адреса
- Передача данных
- Создание условия остановки
- Окончание взаимодействия\

4. Контроль адресации

Как ведущее, так и ведомое устройство поддерживает 7-битный и 10-битный режимы адресации.

- 7-битный режим

Бит addr2en = 0 расшифровывается как режим только с одним адресом: соответствует только OADDR1.

Бит dualen ==1 означает режим двойной адресации, адрес соответствует OADDR1 и OADDR2.

- 10-битный режим

Только возможна адресация OADDR

Поддержка специальных адресов ведомого:

- Адрес широковещательного вызова (0b0000000x). Этот адрес включен, когда GCAEN = 1.
- Адрес устройства SMBus по умолчанию (0b1100001x): этот адрес включен для разрешения протокола адресов SMBus в режиме устройства SMBus.

Есть также другие настройки шин, но при конфигурации шины мы используем настройки по умолчанию.

Процедура сопоставления адресов подчинённых:

- Получите начальное условие
- Сопоставление адресов
- Подчиненное устройство отправляет подтверждение, если адрес совпадает.
- ADDR7F устанавливается в 1 с DIFF, указывающем направление передачи.

Когда бит DIFF = 0, ведомое устройство переходит в режим приёмника, начиная приём данных.

Когда бит DIFF = 1, ведомый переходит в режим передачи, начинает передачу данных.

5. Возможность растягивания тактовой частоты

А) Растягивание частоты возможно с установкой бита stretch (растягивания) в регистре ctrl1_bit.

После включения, когда подчиненное устройство не может своевременно обрабатывать данные при определенных условиях, оно переводит SCL (линию тактирования) на низкий уровень, чтобы прекратить связь во избежание потери данных.

Б) Режим передачи

Растягивание частоты включено:

Если данные не записываются в регистр dt_bit до передачи следующего (первый восходящий фронт SCL следующих данных), интерфейс I2C отключит шину SCL и будет ждать, пока данные не будут записаны в dt.

В) Режим приёмника

- Растягивание частоты включено: когда регистр сдвига получает ещё один байт до считывания данных из регистра dt, I2C будет удерживать SCL на низком уровне, ожидая, пока ПО прочитает регистр dt.
- Если отключено растягивание тактовой частоты, то данные в регистре будут ещё не считаны, когда регистр сдвига получает ещё один байт. В этом случае, если получены другие данные, возникает ошибка.

Г) Поток связи подчиненного устройства

Включите периферийную тактовую частоту I2C и настройте биты, связанные с тактовой частотой в регистре ctrl2_bit для правильной

синхронизации, а затем подождите, пока ведущий I2C отправит условие запуска.

! Случай, когда ведомый является передатчиком не написан, т.к. он не реализуется (не используется).

Д) I2C master communication flow

Инициализация:

1. Тактирование ввода программы должно быть установлено, чтобы сгенерировать правильную синхронизацию с помощью бита clcfreq в регистре ctrl2_bit.
2. Запрограммировать скорость передачи данных I2C с помощью бита в регистре clkctrl.
3. Запрограммируйте максимальное время шины с помощью регистра tmrise_bit
4. Запрограммируйте управляющий регистр: ctrl1_bit
5. Включите периферийное устройство, если установлен бит genstart, на шине генерируется условие запуска, и устройство переходит в главный режим.

Е) Передача адреса подчиненного устройства

Мы выбираем 7-битный режим адресации.

Режим 7-разрядного адреса:

Передатчик: когда младший бит отправленного адреса 0, ведущий переходит в режим передатчика.

Приёмник: когда младший бит отправленного адреса 1, ведущий переходит в режим приёмника.

! Достаточно подробно было описан алгоритм работы шины на рис. 3.4

6. В ходе работы был использован готовый код, написанный для stm32f103, который переписывался для работы на платформе Artery.

При побитовом написании программы (в стиле CMSIS) была проблема с битом tdbe (transmit data buffer register – флаг проверки пустоты буфера).

tdbe равен:

0: данные передаются из регистра DT в регистр сдвига (на данный момент данные ещё загружаются)

1: данные были переданы из регистра DT в сдвиговый регистр. Регистр данных теперь пуст.

Этот регистр устанавливается, когда регистр DT пуст, и очищается при записи в регистр DT.

! Бит TDBE не очищается при записи первых передаваемых данных, когда установлен tdc, поскольку регистр данных в это время всё ещё пуст.

При программировании были трудности, связанные с тем, что биты tdc и tdbe не переключаются в нужные режимы. Это было связано с отсутствием подтверждения от контроллера экрана. Решил проблему с помощью функции `i2c_memory_write,` в которой был более корректный код работы I2C для Artery. После перевода на данную функцию биты стали обновляться корректно.

Данная функция является ключевой при передаче данных по шине, позволяя отправлять данные на шину. Полный код представлен ниже:

```
i2c_status_type i2c_memory_write(i2c_handle_type* hi2c, i2c_mem_address_width_type
mem_address_width, uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t
size, uint32_t timeout)
{
  /* initialization parameters */
  hi2c->pbuff = pdata;
  hi2c->pcount = size;

  hi2c->error_code = I2C_OK;

  /* wait for the busy flag to be reset */
  if(i2c_wait_flag(hi2c, I2C_BUSYF_FLAG, I2C_EVENT_CHECK_NONE, timeout) != I2C_OK)
  {
    return I2C_ERR_STEP_1;
  }

  /* ack acts on the current byte */
  i2c_master_receive_ack_set(hi2c->i2cx, I2C_MASTER_ACK_CURRENT);

  /* send slave address */
  if(i2c_master_write_addr(hi2c, address, timeout) != I2C_OK)
  {
    /* generate stop condtion */
    i2c_stop_generate(hi2c->i2cx);

    return I2C_ERR_STEP_2;
  }

  /* clear addr flag */
  i2c_flag_clear(hi2c->i2cx, I2C_ADDR7F_FLAG);

  /* wait for the tdbe flag to be set */
  if(i2c_wait_flag(hi2c, I2C_TDBE_FLAG, I2C_EVENT_CHECK_ACKFAIL, timeout) != I2C_OK)
  {
    /* generate stop condtion */
    i2c_stop_generate(hi2c->i2cx);

    return I2C_ERR_STEP_3;
  }
```

```
  /* send memory address */
  if(i2c_memory_address_send(hi2c, mem_address_width, mem_address, timeout) !=
I2C_OK)
  {
    return I2C_ERR_STEP_4;
  }

  while(size > 0)
  {
    /* wait for the tdbe flag to be set */
    if(i2c_wait_flag(hi2c, I2C_TDBE_FLAG, I2C_EVENT_CHECK_ACKFAIL, timeout) !=
I2C_OK)
    {
      /* generate stop condtion */
      i2c_stop_generate(hi2c->i2cx);

      return I2C_ERR_STEP_5;
    }

    /* write data */
    i2c_data_send(hi2c->i2cx, (*pdata++));
    size--;
  }

  /* wait for the tdc flag to be set */
  if(i2c_wait_flag(hi2c, I2C_TDC_FLAG, I2C_EVENT_CHECK_ACKFAIL, timeout) != I2C_OK)
  {
    /* generate stop condtion */
    i2c_stop_generate(hi2c->i2cx);

    return I2C_ERR_STEP_6;
  }

  /* generate stop condtion */
  i2c_stop_generate(hi2c->i2cx);

  return I2C_OK;
}
```

Возвращает функция перечислимый тип даных enum статуса работы I2C.

```
typedef enum
{
  I2C_OK = 0,           /*!< no error */
  I2C_ERR_STEP_1,       /*!< step 1 error */
  I2C_ERR_STEP_2,       /*!< step 2 error */
  I2C_ERR_STEP_3,       /*!< step 3 error */
  I2C_ERR_STEP_4,       /*!< step 4 error */
  I2C_ERR_STEP_5,       /*!< step 5 error */
  I2C_ERR_STEP_6,       /*!< step 6 error */
  I2C_ERR_STEP_7,       /*!< step 7 error */
  I2C_ERR_STEP_8,       /*!< step 8 error */
  I2C_ERR_STEP_9,       /*!< step 9 error */
  I2C_ERR_STEP_10,      /*!< step 10 error */
  I2C_ERR_STEP_11,      /*!< step 11 error */
  I2C_ERR_STEP_12,      /*!< step 12 error */
  I2C_ERR_START,        /*!< start error */
  I2C_ERR_ADDR10,       /*!< addr10 error */
  I2C_ERR_ADDR,         /*!< addr error */
  I2C_ERR_STOP,         /*!< stop error */
```

```
I2C_ERR_ACKFAIL,     /*!< ackfail error */
I2C_ERR_TIMEOUT,     /*!< timeout error */
I2C_ERR_INTERRUPT,   /*!< interrupt error */

    } i2c_status_type;
```

Бит startf находится в регистре status register1.

0: нет сгенерированного условия старта

1: стартовое условие сгенерировано

! Очищается доступом для записи в регистр DT после того, как ПО прочитает регистр sts1_bit.

7. Рассмотрим control_bit

Bit10: acken (acknowledge enable – подтверждение возможно)

0: Disabled (нет оправленного подтверждения)

1: Enabled (подтверждение отправлено)

Бит устанавливается и убирается программно.

8. Рассмотрим status register

ADDR7F: 0-7 битный флаг соответствия адреса.

0: адрес не принимается в режиме подчинённого устройство и не отправляется в режиме хоста.

1: адрес отправляется в режиме хоста или адрес принимается в режиме подчиненного устройства.

Бит сбрасывается доступом для чтения к регистру status2 после того, как ПО прочитает регистр status1.

! Бит ADDR7F не устанавливается после приёма NACK

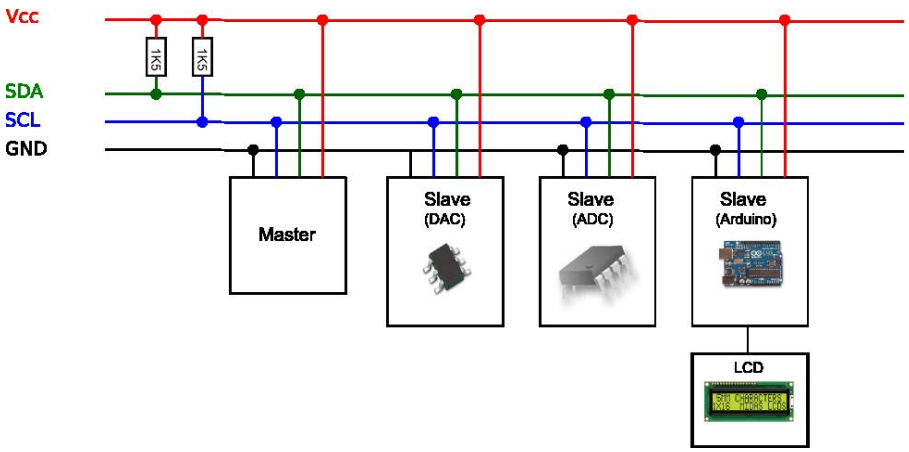

Рисунок 3.5 – Пример подключения по шине I2C

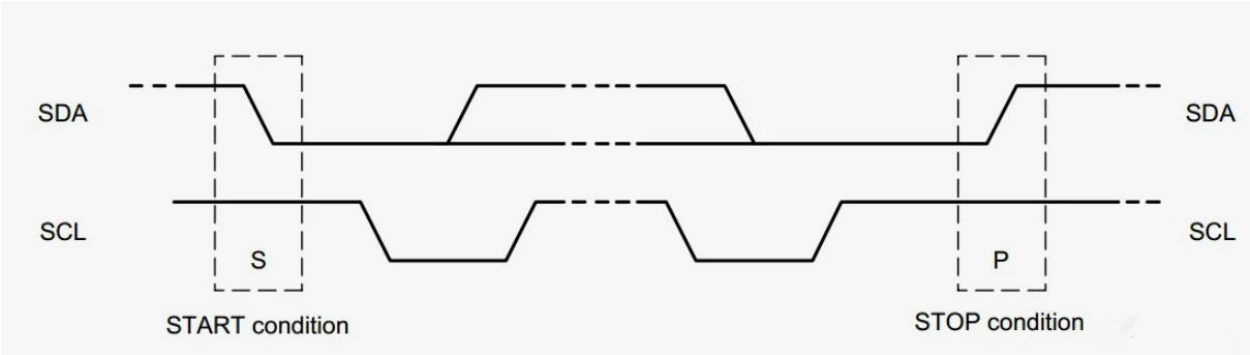

Рисунок 3.6 – Работа шины в виде графика напряжения

SDA (линия данных) меняется только при SCL = 0.

S – старт, мастер передаёт, slave в ожидании

P – признак конца передачи

Если после после переданного байта 9 битом (его выставляет приёмник) стоит:

0: приёмник понял и подтверждает приём сигнала

1: приёмник не понял или не желает обмениваться информацией

На данном этапе с помощью логического анализатора я смог точно определить, что передача данных по I2C ведется успешно. Я перешел к задаче формирования и отправки корректного пакета данных для отображения корректных данных на устройтсве.

4. Реализация работы вывода на экран

Экран SSD1306 - это маленький графический дисплей, который широко используется в электронике для отображения текста и графики. Этот тип дисплея использует технологию OLED (Organic Light-Emitting Diode), что позволяет каждому пикселю быть своим собственным источником света. В случае SSD1306, OLED-экран работает по принципу управления точечной матрицей.

Вот основные особенности и принципы работы экрана SSD1306: Контроллер SSD1306: Экран управляется микросхемой-контроллером SSD1306. Этот контроллер является монокристаллическим дисплеем на основе CMOS, спроектированным для использования с дисплеями OLED.

Разрешение: Экраны SSD1306 обычно имеют небольшое разрешение, например, 128x64 или 128x32 пикселя. Это означает, что они состоят из указанного числа пикселей по горизонтали и вертикали.

Светодиоды OLED: Каждый пиксель на экране представляет собой маленький светодиод OLED, который может светиться или быть выключенным независимо от других пикселей. Это позволяет создавать высококонтрастное и яркое изображение.

Интерфейс I2C: Экраны SSD1306 могут быть подключены к микроконтроллеру с использованием интерфейсов I2C или SPI. Интерфейс

I2C обеспечивает простое соединение с использованием всего двух проводов (SDA и SCL).

Команды и данные: Для управления содержимым дисплея, микроконтроллер отправляет команды и данные через соответствующий интерфейс. Команды используются для настройки параметров дисплея (например, яркость, режим отображения), а данные используются для передачи информации о содержимом экрана (текст, графика и т.д.).

Графическое программирование: Для отображения информации на экране, программисты могут использовать специальные библиотеки и API, предоставляемые производителями, чтобы упростить работу с дисплеем. Обычно это включает в себя установку пикселей, отображение текста и рисование форм.

С использованием данных принципов и команд, микроконтроллер может легко управлять содержимым OLED-экрана SSD1306 и создавать различные графические интерфейсы или отображать информацию в реальном времени.

Для отправки данных на OLED-экран SSD1306, вы обычно используете команды и байты данных через интерфейс I2C или SPI. Вот общий процесс отправки данных на экран SSD1306:

Выбор буфера данных: Экран SSD1306 обычно имеет два буфера данных: один для текущего отображения и один для буферизации новых данных. Вы отправляете данные в буфер, который в данный момент не отображается на экране.

Выбор координат: Вы указываете координаты (x, y) для позиции пикселя или области данных на экране.

Отправка команды и данных: После установки координат вы отправляете команду, указывающую на то, что сейчас будут отправлены данные (обычно команда 0x40). Затем вы отправляете фактические данные в буфер пикселей или символов.

Формат данных: Формат данных может варьироваться в зависимости от режима работы (горизонтальный, вертикальный, страничный и т.д.) и цветовой глубины. Однако для монохромных (ч/б) OLED-экранов, таких как SSD1306, каждый байт представляет 8 пикселей в виде битовой маски. Каждый бит в байте соответствует одному пикселю, и установка бита в 1 включает пиксель, а установка в 0 - выключает.

Например, если у вас есть байт 0b00101011, это означает, что три пикселя будут включены (в позициях 2, 4 и 7), а остальные пиксели будут выключены.

Повторение процесса: Вы повторяете этот процесс для каждой строки или страницы экрана в зависимости от режима работы.

Я реализовал несколько функций, формирующих пакеты данных для отправки на дисплей.

Например, SSD1306_PutsE может рисовать текст на экране в определенном месте и нужным шрифтом (из набора).

Листинг 4.1 – Пример вызова функций вывода текста

```
SSD1306_PutsE("GEOSCAN,", &Font_16x26, SSD1306_COLOR_WHITE, 0, 0);
SSD1306_UpdateScreen();

delay_ms(g_speed * DELAY * 10);
SSD1306_PutsE("RUSSIA,", &Font_7x10, SSD1306_COLOR_WHITE, 10, 31);
SSD1306_PutsE("St. Peterburg", &Font_7x10, SSD1306_COLOR_WHITE, 10, 41);
```

Реализация функции представлена ниже, возврат значения char является номинальным. Также представлена функция, вызываемая функцией SSD1306_PutsE.

Листинг 4.2 – Реализация функции вывода текста

```
char SSD1306_PutsE(char* str, FontDef_t* Font, SSD1306_COLOR_t color, uint16_t x,
uint16_t y) {
	/* Set write pointers */
	SSD1306.CurrentX = x;
	SSD1306.CurrentY = y;

	/* Write characters */
	while (*str) {
		/* Write character by character */
		if (SSD1306_Putc(*str, Font, color) != *str) {
			/* Return error */
			return *str;
		}

		/* Increase string pointer */
		str++;
	}

	/* Everything OK, zero should be returned */
	return *str;

	}

char SSD1306_Putc(char ch, FontDef_t* Font, SSD1306_COLOR_t color) {
	uint32_t i, b, j;

	/* Check available space in LCD */
	if (
		SSD1306_WIDTH <= (SSD1306.CurrentX + Font->FontWidth) ||
		SSD1306_HEIGHT <= (SSD1306.CurrentY + Font->FontHeight)
```

```
	) {
		/* Error */
		return 0;
	}

	/* Go through font */
	for (i = 0; i < Font->FontHeight; i++) {
		b = Font->data[(ch - 32) * Font->FontHeight + i];
		for (j = 0; j < Font->FontWidth; j++) {
			if ((b << j) & 0x8000) {
				SSD1306_DrawPixel(SSD1306.CurrentX + j, (SSD1306.CurrentY +
i), (SSD1306_COLOR_t) color);
			} else {
				SSD1306_DrawPixel(SSD1306.CurrentX + j, (SSD1306.CurrentY +
i), (SSD1306_COLOR_t)!color);
			}
		}
	}

	/* Increase pointer */
	SSD1306.CurrentX += Font->FontWidth;

	/* Return character written */
	return ch;
}
```

Данная функция использует функцию SSD1306_DrawPixel. SSD1306_DrawPixel является основной при рисовании, она «зажигает» или «тушит» конкретный пиксель.

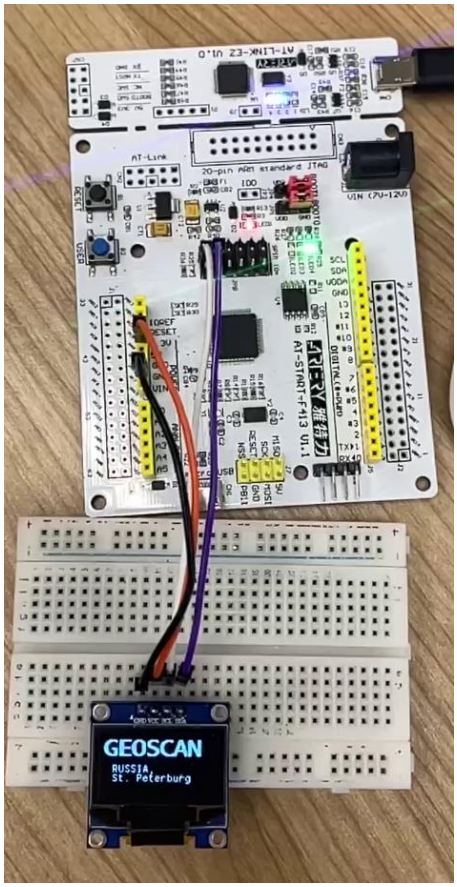

Рисунок 4.1 – Результат работы кода вывода на экран текста

Листинг 4.3 – Реализация функции включения и выключения пикселя экрана

```
void SSD1306_DrawPixel(uint16_t x, uint16_t y, SSD1306_COLOR_t color) {
	if (
		x >= SSD1306_WIDTH ||
		y >= SSD1306_HEIGHT
	) {
		/* Error */
		return;
	}

	/* Check if pixels are inverted */
	if (SSD1306.Inverted) {
		color = (SSD1306_COLOR_t)!color;
	}

	/* Set color */
	if (color == SSD1306_COLOR_WHITE) {
		SSD1306_Buffer[x + (y / 8) * SSD1306_WIDTH] |= 1 << (y % 8);
	} else {
		SSD1306_Buffer[x + (y / 8) * SSD1306_WIDTH] &= ~(1 << (y % 8));
	}
}
```

С помощью функции **SSD1306_DrawPixel** включается или выключается отдельный пиксель. С помощью данного функционала мы можем рисовать линии, треугольники, квадраты и круги и других геометрических фигур. Приведу пример реализации для круга.

Листинг 4.4 – Реализация функции отрисовки круга с заданным радиусом и центром

```
void SSD1306_DrawCircle(int16_t x0, int16_t y0, int16_t r, SSD1306_COLOR_t c) {
	int16_t f = 1 - r;
	int16_t ddF_x = 1;
	int16_t ddF_y = -2 * r;
	int16_t x = 0;
	int16_t y = r;

    SSD1306_DrawPixel(x0, y0 + r, c);
    SSD1306_DrawPixel(x0, y0 - r, c);
    SSD1306_DrawPixel(x0 + r, y0, c);
    SSD1306_DrawPixel(x0 - r, y0, c);

    while (x < y) {
        if (f >= 0) {
            y--;
            ddF_y += 2;
            f += ddF_y;
        }
        x++;
        ddF_x += 2;
        f += ddF_x;

        SSD1306_DrawPixel(x0 + x, y0 + y, c);
        SSD1306_DrawPixel(x0 - x, y0 + y, c);
        SSD1306_DrawPixel(x0 + x, y0 - y, c);
        SSD1306_DrawPixel(x0 - x, y0 - y, c);

        SSD1306_DrawPixel(x0 + y, y0 + x, c);
```

```
        SSD1306_DrawPixel(x0 - y, y0 + x, c);
        SSD1306_DrawPixel(x0 + y, y0 - x, c);
        SSD1306_DrawPixel(x0 - y, y0 - x, c);
    }
}
```

Основная сложность была в выводе изображений. Была использована функция включения и выключения пикселя. Т.к. пакеты данных передаются горизонтальными полосами на дисплей, то требовался соответствующий массив данных для использования в функции вывода изображения.

Листинг 4.5 – Реализация функции вывода изображения

```
void ssd1306_DrawBitmap(uint8_t x, uint8_t y, const unsigned char* bitmap, uint8_t w,
uint8_t h, SSD1306_COLOR_t color) {
    int16_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte
    uint8_t byte = 0;

    if (x >= SSD1306_WIDTH || y >= SSD1306_HEIGHT) {
        return;
    }

    for (uint8_t j = 0; j < h; j++, y++) {
        for (uint8_t i = 0; i < w; i++) {
            if (i & 7) {
                byte <<= 1;
            } else {
                byte = (*(const unsigned char *)(&bitmap[j * byteWidth + i / 8]));
            }

            if (byte & 0x80) {
             SSD1306_DrawPixel(x + i, y, color);
            }
        }
    }
    return;
}
```

Алгоритм для вывода изображения:

1. Сначала на сайте https://www.resizepixel.com/ru преобразуй картинку до размера 128*64
2. Затем бинаризируй её в приложении LCD Vision
3. Последний шаг: получи код в приложении Image2Code и загрузи полученный массив в AT32IDE в файлы с названием images.c и images.h

! Более подробная документация и приложения расположены в моём репозитории: https://github.com/simeonidi03/Docs_embd

Листинг 4.6 – массив для вывода бинаризированного изображения на экран

```
const unsigned char Bus [] = {
0x00, 0x00, 0x15, 0x55, 0x55, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x0A, 0xCA, 0x22, 0xAA, 0xDD, 0xAA, 0xA8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x29, 0x37, 0xFD, 0x55, 0x6B, 0x55, 0x55, 0x55, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x56, 0xDF, 0x57, 0xFF, 0x95, 0xBE, 0xAA, 0xAA, 0xA8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xB9, 0x1F, 0x80, 0xFF, 0xFA, 0x6F, 0xFF, 0xD5, 0x55, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x3F, 0xE9, 0xFF, 0xFF, 0x97, 0xF6, 0xBA, 0xAA, 0xA0, 0x00, 0x00, 0x00, 0x00, 0x00,
0x01, 0x50, 0x5F, 0xFD, 0xFF, 0xFC, 0xF5, 0xFD, 0x55, 0x01, 0x50, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xA0, 0x7F, 0xFF, 0xFF, 0xFE, 0xFA, 0xBC, 0x3E, 0x00, 0xAA, 0x00, 0x00, 0x00, 0x00, 0x00,
0x01, 0x60, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x6E, 0x1A, 0x80, 0x15, 0x00, 0x00, 0x00, 0x00, 0x00,
0x02, 0xA2, 0xFF, 0xFF, 0xFF, 0xFF, 0xF7, 0x9E, 0xB7, 0x00, 0x22, 0x80, 0x00, 0x00, 0x00, 0x00,
0x05, 0x57, 0xFF, 0xFF, 0xFF, 0xFF, 0xF7, 0xEB, 0x1E, 0x00, 0x08, 0x50, 0x00, 0x00, 0x00, 0x00,
0x05, 0x5F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFB, 0xF5, 0x5D, 0x00, 0x02, 0x88, 0x00, 0x00, 0x00, 0x00,
0x0A, 0xA5, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFA, 0xBF, 0x00, 0x01, 0x16, 0x00, 0x00, 0x00, 0x00,
0x05, 0x52, 0x5F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0xFF, 0x00, 0x00, 0x21, 0x00, 0x00, 0x00, 0x00,
0x0A, 0xAD, 0x23, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x5F, 0x81, 0x00, 0x80, 0xC0, 0x00, 0x00, 0x00,
0x05, 0x55, 0xD4, 0x1B, 0xFF, 0xFF, 0xFF, 0xFF, 0xB7, 0xA6, 0x40, 0x00, 0x20, 0x00, 0x00, 0x00,
0x02, 0xAA, 0xAB, 0x45, 0xFF, 0xFF, 0xFF, 0xFF, 0xDF, 0xEC, 0x40, 0x20, 0x18, 0x00, 0x00, 0x00,
0x09, 0x55, 0x6D, 0x5A, 0x17, 0xFF, 0xFF, 0xFF, 0xEB, 0xFB, 0xB4, 0x80, 0x04, 0x00, 0x00, 0x00,
0x04, 0xAA, 0xB5, 0x6A, 0xC9, 0x7F, 0xCD, 0xFF, 0xF5, 0xFE, 0xD5, 0x20, 0x03, 0x00, 0x00, 0x00,
0x0A, 0xAA, 0x4A, 0xAA, 0xB4, 0x97, 0xC6, 0xFF, 0xFA, 0xF7, 0x6A, 0x80, 0x01, 0x80, 0x00, 0x00,
0x04, 0xB5, 0x69, 0x55, 0x57, 0x40, 0x95, 0xFF, 0xFD, 0x7E, 0xDD, 0x54, 0xA0, 0x80, 0x00, 0x00,
0x09, 0x56, 0x94, 0x8A, 0xAA, 0xBA, 0x3F, 0xFF, 0xFF, 0x5F, 0xF6, 0xBF, 0x55, 0x20, 0x00, 0x00,
0x00, 0xAB, 0x52, 0xA4, 0x15, 0x55, 0xA1, 0x7F, 0xFF, 0xBF, 0xFD, 0xE9, 0x22, 0x48, 0x00, 0x00,
0x11, 0x55, 0x4A, 0x52, 0xA2, 0xEA, 0xAE, 0x9F, 0xFF, 0xAE, 0xAA, 0x94, 0x88, 0x92, 0x00, 0x00,
0x02, 0xAA, 0xA5, 0x29, 0x49, 0x2A, 0xAA, 0xA9, 0x7F, 0xD5, 0xD5, 0x6A, 0x52, 0x24, 0x80, 0x00,
0x0B, 0xFA, 0xD2, 0x94, 0x94, 0xC0, 0x55, 0x76, 0x87, 0xDB, 0x75, 0x55, 0x24, 0x41, 0x20, 0x00,
0x17, 0xFD, 0x49, 0x22, 0x49, 0x28, 0x0A, 0xAB, 0x68, 0xAD, 0xAD, 0x5A, 0x90, 0x82, 0x40, 0x00,
0x07, 0xFE, 0xA4, 0x49, 0x24, 0x92, 0x80, 0x55, 0x57, 0x56, 0xDB, 0x47, 0x6A, 0x00, 0x14, 0x00,
0x0F, 0x2E, 0xA8, 0x02, 0x49, 0x24, 0x50, 0x0A, 0xAA, 0xAB, 0x6E, 0xA9, 0xB5, 0x20, 0x08, 0x00,
0x0E, 0xB7, 0x50, 0x00, 0x00, 0x09, 0x25, 0x41, 0x55, 0x55, 0xB5, 0xF4, 0x6D, 0x48, 0x02, 0x00,
0x0E, 0xCB, 0x22, 0x00, 0x00, 0x00, 0x88, 0x94, 0x2A, 0xBA, 0xDB, 0x5A, 0x9B, 0xA4, 0x01, 0x00,
0x0C, 0xDF, 0x89, 0x50, 0x00, 0x00, 0x02, 0x24, 0x0A, 0xD7, 0x6D, 0x07, 0x56, 0xDA, 0x80, 0x80,
0x0A, 0xDD, 0x80, 0x2A, 0x80, 0x00, 0x00, 0x08, 0x05, 0x6D, 0xB6, 0xA9, 0xEB, 0xB5, 0x51, 0x40,
0x04, 0x5B, 0x82, 0x85, 0x50, 0x00, 0x00, 0x02, 0x0A, 0xB6, 0xDA, 0x92, 0x3D, 0x7F, 0xAA, 0x20,
0x03, 0x2B, 0xC5, 0x52, 0x2A, 0x00, 0x00, 0x00, 0x15, 0x4B, 0x6D, 0x49, 0x2B, 0xED, 0x75, 0x50,
0x05, 0xC2, 0xD2, 0xA9, 0x55, 0xA0, 0x00, 0x00, 0x28, 0x05, 0xB6, 0xA4, 0x15, 0xBF, 0xEE, 0xF8,
0x03, 0xD3, 0xD5, 0x55, 0x2A, 0x55, 0x00, 0x00, 0x42, 0xE0, 0xDB, 0x49, 0x08, 0x7F, 0xFF, 0xF8,
0x01, 0xAC, 0xEA, 0xAA, 0xA5, 0x2A, 0xA0, 0x00, 0x8F, 0xFC, 0x6E, 0xA2, 0x29, 0x2F, 0xFE, 0xF8,
0x02, 0x6D, 0xED, 0x55, 0x5A, 0xA5, 0x5A, 0x80, 0x3F, 0xFF, 0x1B, 0x50, 0xB2, 0x4F, 0xFF, 0xC4,
0x02, 0x6D, 0xF2, 0xAA, 0xAA, 0xAA, 0xA5, 0x54, 0x5F, 0xFF, 0x8D, 0xAB, 0x48, 0x0F, 0xFF, 0x28,
0x02, 0xE3, 0xFD, 0x57, 0x55, 0x55, 0x52, 0xA8, 0x3F, 0x57, 0xEB, 0x55, 0x76, 0xBF, 0xFA, 0xA4,
0x01, 0x6F, 0xFF, 0xAA, 0xBA, 0xAA, 0xAA, 0x55, 0x7C, 0x93, 0xF5, 0xEA, 0xAF, 0xEA, 0xEA, 0x48,
0x01, 0xBB, 0xFF, 0xF5, 0x57, 0x75, 0x55, 0x54, 0xFB, 0x39, 0xF3, 0x55, 0x5A, 0xD1, 0xD4, 0x80,
0x00, 0xDF, 0xFF, 0xFE, 0xAA, 0xDB, 0xAA, 0xAA, 0xF6, 0x7A, 0xFA, 0xEA, 0xEA, 0xA1, 0x20, 0x04,
0x00, 0x7F, 0xFD, 0xFF, 0xD5, 0x56, 0xD5, 0x55, 0x70, 0x79, 0x7D, 0xB5, 0x55, 0x52, 0xA0, 0x00,
0x00, 0x05, 0x56, 0xBF, 0xFE, 0xAB, 0x7B, 0xAA, 0xE6, 0x28, 0xBD, 0x6A, 0xAA, 0xAD, 0x50, 0x00,
0x00, 0x00, 0x01, 0x55, 0xFF, 0xD5, 0x56, 0xDD, 0xEF, 0x38, 0x3E, 0xD5, 0x55, 0x52, 0xB0, 0x58,
0x00, 0x00, 0x00, 0x0B, 0x6F, 0xFE, 0xAB, 0xB6, 0xEF, 0xB3, 0x1E, 0xAA, 0xAA, 0xAA, 0x60, 0xF8,
0x00, 0x00, 0x00, 0x00, 0xB7, 0xFF, 0xD5, 0x6D, 0xE7, 0xD7, 0xDF, 0x55, 0x7D, 0x55, 0x1F, 0xF8,
0x00, 0x00, 0x00, 0x00, 0x0A, 0xBF, 0xFA, 0x9B, 0xE7, 0xE7, 0x4E, 0xAB, 0x2A, 0xAA, 0xFF, 0xF0,
0x00, 0x00, 0x00, 0x00, 0x01, 0x57, 0xFF, 0xE5, 0xE8, 0x07, 0x9F, 0x73, 0x2A, 0xA4, 0xFF, 0xE0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x2A, 0xBF, 0xFF, 0x70, 0x6B, 0xCF, 0xA9, 0x15, 0x51, 0xFF, 0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x56, 0xFF, 0xEB, 0xE8, 0x3F, 0x55, 0x0A, 0x8B, 0xFA, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09, 0xBF, 0xF7, 0xDC, 0x7F, 0xE8, 0x2A, 0x53, 0xA0, 0x40,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x55, 0xF3, 0xDE, 0xFF, 0x5C, 0x95, 0x40, 0x02, 0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0A, 0xBB, 0xCF, 0x7F, 0xF7, 0x5A, 0xAA, 0xA8, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x7D, 0x9F, 0xBF, 0xBD, 0xF7, 0x55, 0x54, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0D, 0x4F, 0x7F, 0xF7, 0xBD, 0xFE, 0xA0, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0E, 0xF6, 0xFF, 0xFF, 0x7B, 0xAA, 0x80, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x51, 0xFF, 0xFE, 0xEE, 0xD4, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xAB, 0x51, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x5F, 0xFA, 0xD4, 0x88, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x05, 0x21, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

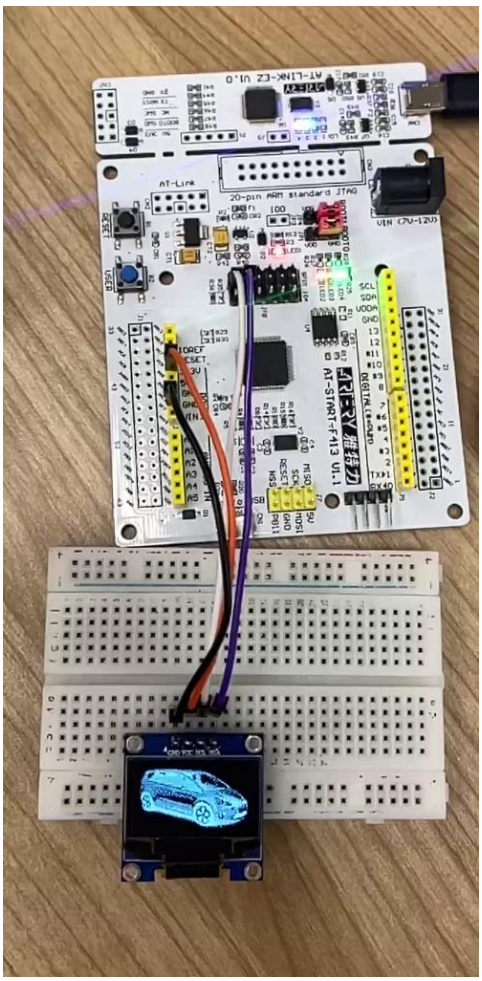

Рисунок 4.2 – Результат работы кода вывода на экран бинаризированной фотографии

Бинаризация изображений - это процесс преобразования изображения из градаций серого в бинарное изображение, где каждый пиксель может быть только черным или белым. Этот процесс широко используется для анализа изображений, распознавания образов и других приложений. Я не стал писать код для бинаризации изображения, предлагая пользователю самим получить массив с помощью специализированного ПО, прикладываемого в репозитории.

На этом разработанный функционал экрана на данный момент заканчивается.

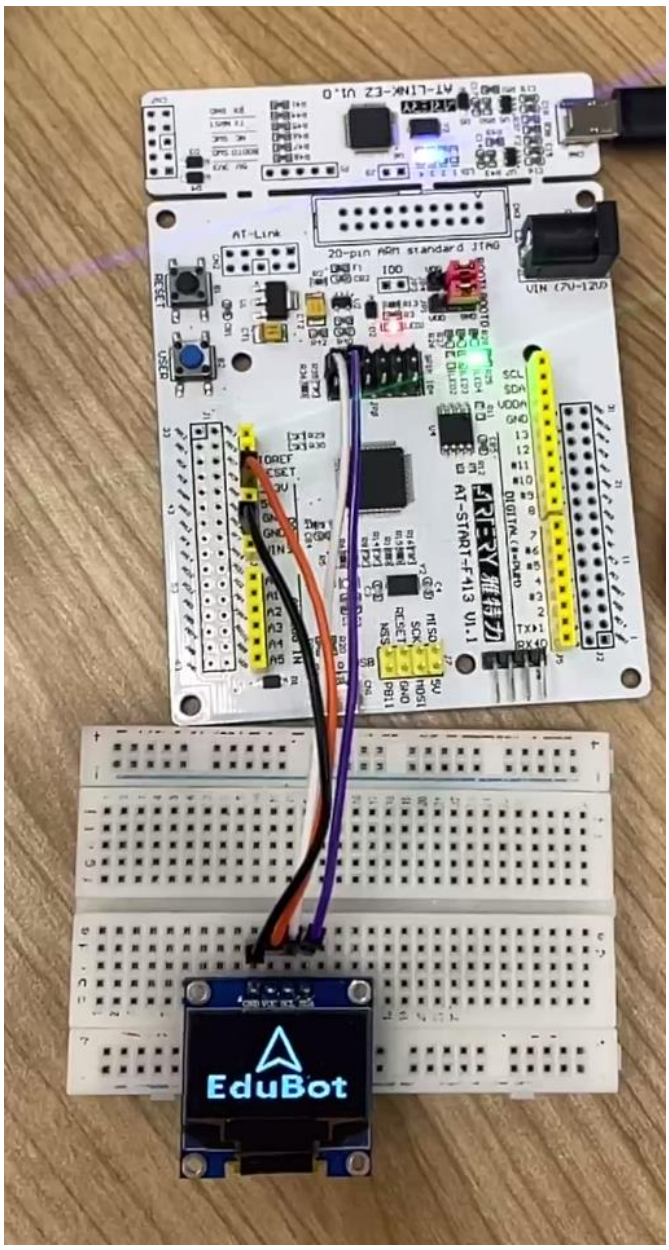

Рисунок 4.3 – Результат работы кода вывода на экран логотипа проекта

5. Реализация работы экрана SSD1306

Листинг 5.1 - Программы с реализацией работы дисплея

main.c

```
#include "at32f413_board.h"
#include "at32f413_clock.h"
#include "at32f413_wk_config.h"
#include "ssd1306.h"
#include <stdio.h>
#include "i2c_at_lib.h"
#include "images.h"
/** @addtogroup AT32F413_periph_template
  * @{
  */
```

```
/** @addtogroup 413_LED_toggle LED_toggle
  * @{
  */

#define DELAY                            100
#define FAST                             1
#define SLOW                             4

uint8_t g_speed = FAST;

void button_exint_init(void);
void button_isr(void);

/**
  * @brief  configure button exint
  * @param  none
  * @retval none
  */
void button_exint_init(void)
{
  exint_init_type exint_init_struct;

  crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
  gpio_exint_line_config(GPIO_PORT_SOURCE_GPIOA, GPIO_PINS_SOURCE0);

  exint_default_para_init(&exint_init_struct);
  exint_init_struct.line_enable = TRUE;
  exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_select = EXINT_LINE_0;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);

  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  nvic_irq_enable(EXINT0_IRQn, 0, 0);
}

/**
  * @brief  button handler function
  * @param  none
  * @retval none
  */
void button_isr(void)
{
  /* delay 5ms */
  delay_ms(5);

  /* clear interrupt pending bit */
  exint_flag_clear(EXINT_LINE_0);
```

```
  /* check input pin state */
  if(SET == gpio_input_data_bit_read(USER_BUTTON_PORT, USER_BUTTON_PIN))
  {
    if(g_speed == SLOW)
      g_speed = FAST;
    else
      g_speed = SLOW;
  }
}

/**
  * @brief  exint0 interrupt handler
  * @param  none
  * @retval none
  */
void EXINT0_IRQHandler(void)
{
  button_isr();
}

/**
  * @brief  main function.
  * @param  none
  * @retval none
  */
int main(void)
{
  /* add user code begin 1 */

  /* add user code end 1 */

  /* system clock config. */
  wk_system_clock_config();

  /* config periph clock. */
  wk_periph_clock_config();

  /* nvic config. */
  wk_nvic_config();

  /* init i2c1 function. */
  wk_i2c1_init();

  /* init i2c1 function. */

  uint8_t ssd1306_int_ok = SSD1306_Init();

//  system_clock_config();

  at32_board_init();
```

```
  button_exint_init();

  if(ssd1306_int_ok){
         at32_led_toggle(LED4);
  } else{
         at32_led_toggle(LED2);
  }


  SSD1306_PutsE("GEOSCAN,", &Font_16x26, SSD1306_COLOR_WHITE, 0, 0); //пишем
надпись в выставленной позиции шрифтом "Font_7x10" белым цветом.
  SSD1306_UpdateScreen();

  delay_ms(g_speed * DELAY * 10);
  SSD1306_PutsE("RUSSIA,", &Font_7x10, SSD1306_COLOR_WHITE, 10, 31);
  SSD1306_PutsE("St. Peterburg", &Font_7x10, SSD1306_COLOR_WHITE, 10, 41);
  SSD1306_UpdateScreen();

  delay_ms(g_speed * DELAY * 10);
  SSD1306_Fill(SSD1306_COLOR_BLACK);
  SSD1306_Geoscan();
  SSD1306_UpdateScreen();

  delay_ms(g_speed * DELAY * 10);
  SSD1306_Fill(SSD1306_COLOR_BLACK);
  SSD1306_UpdateScreen();


  ssd1306_DrawBitmap(0, 0, image1_bits, 128, 64, SSD1306_COLOR_WHITE);
  SSD1306_UpdateScreen();
  delay_ms(g_speed * DELAY * 10);


  ssd1306_DrawBitmap(0, 0, image1_bits, 128, 64, SSD1306_COLOR_WHITE);
  SSD1306_UpdateScreen();
  delay_ms(g_speed * DELAY * 10);

  SSD1306_Fill(SSD1306_COLOR_BLACK);
  ssd1306_DrawBitmap(0, 0, Bus, 128, 64, SSD1306_COLOR_WHITE);
  SSD1306_UpdateScreen();
  delay_ms(g_speed * DELAY * 10);
  SSD1306_UpdateScreen();

  //SSD1306_DrawCircle(63, 45, 15, SSD1306_COLOR_WHITE); //рисуем белую
окружность в позиции 10;33 и радиусом 7 пикселей
  //SSD1306_UpdateScreen();
  //SSD1306_Fill(SSD1306_COLOR_WHITE);
```

```
  while(1)
  {
    at32_led_toggle(LED2);
    delay_ms(g_speed * DELAY);
    at32_led_toggle(LED3);
    delay_ms(g_speed * DELAY);
    at32_led_toggle(LED4);
    delay_ms(g_speed * DELAY);
  }
}
/*
uint8_t SSD1306_Init(); //Инициализация

SSD1306_UpdateScreen(); //Посылаем данные из буффера в памяти дисплею

SSD1306_ToggleInvert(); //инвертирует цвета изображения в оперативной памяти

SSD1306_Fill(SSD1306_COLOR_t Color); //заполняем дисплей желаемым цветом

SSD1306_DrawPixel(uint16_t x, uint16_t y, SSD1306_COLOR_t color);
//нарисовать один пиксел

SSD1306_GotoXY(uint16_t x, uint16_t y); //установить позицию текстового
курсора

SSD1306_Putc(char ch, FontDef_t* Font, SSD1306_COLOR_t color); //вывести
символ ch в позиции курсора

SSD1306_Puts(char* str, FontDef_t* Font, SSD1306_COLOR_t color); //вывести
строку str в позиции курсора

SSD1306_DrawLine(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,
SSD1306_COLOR_t c); //нарисовать линию

SSD1306_DrawRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h,
SSD1306_COLOR_t c); //наррисовать прямоугольник

SSD1306_DrawFilledRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h,
SSD1306_COLOR_t c); //заполненный прямоугольник

SSD1306_DrawTriangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
uint16_t x3, uint16_t y3, SSD1306_COLOR_t color); //треугольник

SSD1306_DrawCircle(int16_t x0, int16_t y0, int16_t r, SSD1306_COLOR_t c);
//круг радиуса r

SSD1306_DrawFilledCircle(int16_t x0, int16_t y0, int16_t r, SSD1306_COLOR_t
c); //заполненный круг
*/
```

6. Словесное описание алгоритма работы WS2812b

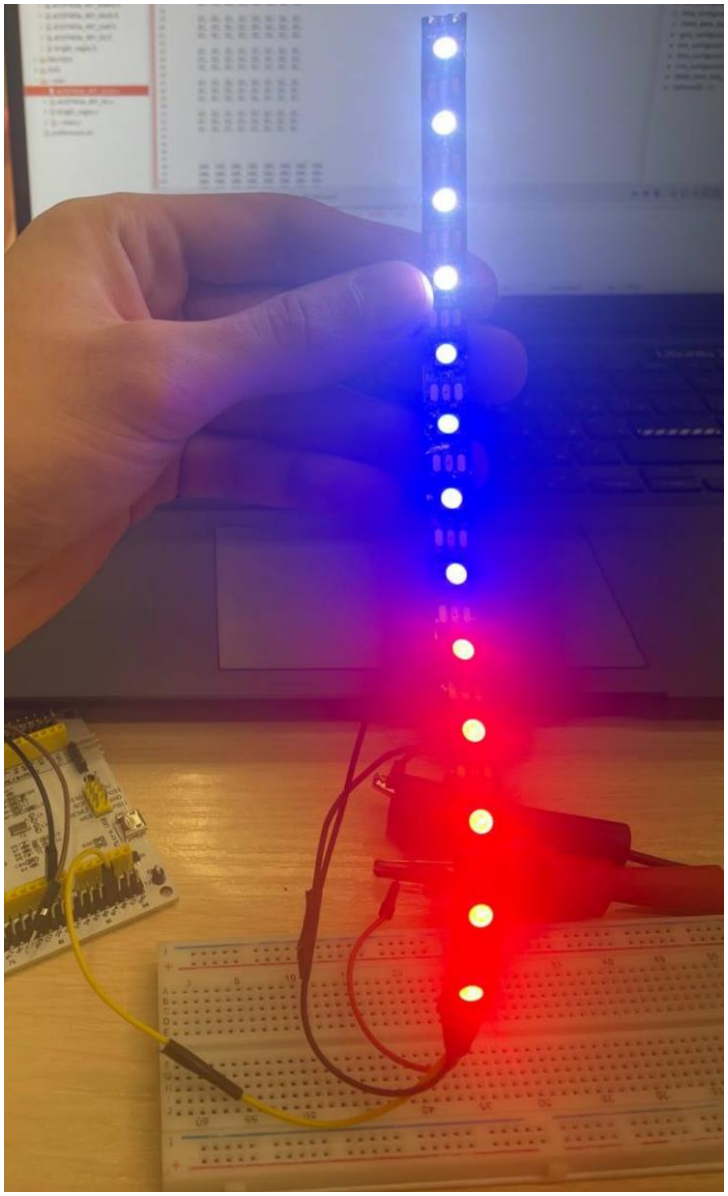

Рисунок 6.1 – Результат работы кода вывода на экран логотипа проекта

Краткий обзор светодиодной ленты:

1. Адресные светодиоды WS2812b имеет две ножки питания, вход и выход.
2. Команда приходит на ножку в виде последовательности 24 битов, по 8 битов на каждый цвет.
3. Синхроимпульс отсутствует
4. Длительность единицы = 0,85 мкс
5. Длительность нуля = 0,4 мкс
6. На один бит приходится 1,25 мкс
7. На один светодиод тратится (8 + 8 + 8) * 1,25 мкс = 30 мкс
8. Скорость передачи 800 кБит/с

9. В каждом чипе данные, которые ему переданы в нём зависают и только последние 24 бита он «забирает» себе.

Технические характеристики:

- Размер 5х5 мм
- Частота ШИМ 400 Гц
- Напряжения питания – 5В
- Потребление при максимальной яркости – 1 мА на светодиод (на практике до 1,2 мА выдавал потребление лабораторный источник питания)
- Цветность RGB, 256 оттенков на канал, 16 миллионов цветов
- Размер данных – 24 бита на светодиод
- Скорость передачи данных – 800 кГц

ШИМ (Широтно-Импульсная Модуляция) — это метод управления аналоговым сигналом с использованием цифровых средств. Он широко применяется в электронике и автоматике для управления мощностью, освещением, скоростью двигателей и другими параметрами.

Основная идея ШИМ заключается в периодическом изменении ширины импульсов в цифровом сигнале. Это позволяет эффективно регулировать среднюю мощность или интенсивность сигнала, сохраняя при этом цифровую природу управления. ШИМ часто используется для создания аналоговых сигналов или управления устройствами, способными работать с аналоговыми сигналами.

Основные характеристики ШИМ включают:
1. Частота (Frequency): Частота ШИМ определяет, как часто происходит переключение между состояниями (вкл/выкл). Измеряется в герцах (Гц).
2. Скважность (Duty Cycle): Скважность представляет собой отношение времени, в течение которого сигнал находится в состоянии "включено" (высокий уровень), к общему периоду. Измеряется в процентах.
Чем выше частота и скважность, тем ближе выходной сигнал приближается к аналоговому сигналу. Путем изменения скважности и/или частоты можно регулировать выходной сигнал ШИМ для управления различными устройствами.

Примеры применения ШИМ включают управление яркостью светодиодов, регулирование скорости двигателей, управление мощностью в электронных устройствах и другие задачи, где требуется аналоговый уровень управления с использованием цифровых средств.

В моей работе я использую ШИМ для отправки данных на адресные светодиоды.

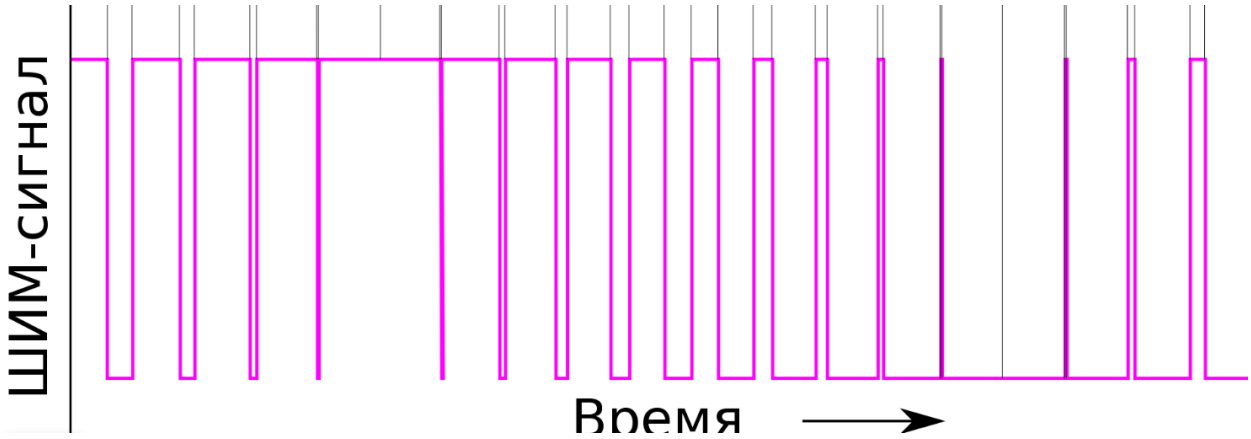

Рисунок 6.2 – Пример передачи ШИМ-сигнала

DMA (Direct Memory Access) - это технология, которая позволяет периферийным устройствам, таким как контроллеры ввода/вывода (I/O), обращаться к системной памяти напрямую, минуя центральный процессор (CPU). DMA улучшает эффективность работы системы, позволяя устройствам передавать или получать данные без прямого участия процессора.

Основные характеристики и преимущества технологии DMA включают:

Без прерывания CPU: Периферийные устройства, такие как сетевые адаптеры, звуковые карты или жесткие диски, могут передавать данные между собой и памятью без постоянного участия процессора. Это позволяет процессору заниматься другими задачами в то время, как DMA управляет передачей данных.

Улучшенная производительность: Передача данных через DMA может быть более эффективной, чем через CPU, так как DMA работает непосредственно с системной шиной и может использовать более высокие скорости передачи.

Сокращение времени задержки: Периферийные устройства могут обмениваться данными с памятью напрямую, что снижает время задержки в сравнении с передачей данных через процессор.

Пакетная передача данных: DMA обычно позволяет передавать данные блоками или пакетами, что может быть более эффективным для некоторых приложений, например, при чтении или записи больших объемов данных.

Расширение возможностей устройств ввода/вывода: DMA позволяет устройствам ввода/вывода более эффективно управлять данными и буферизацией, что особенно важно в высокопроизводительных системах.

Принцип работы DMA зависит от конкретной архитектуры компьютерной системы. В системах с DMA, программирование и конфигурация DMA контроллера обеспечивают обмен данными между периферийными устройствами и системной памятью.

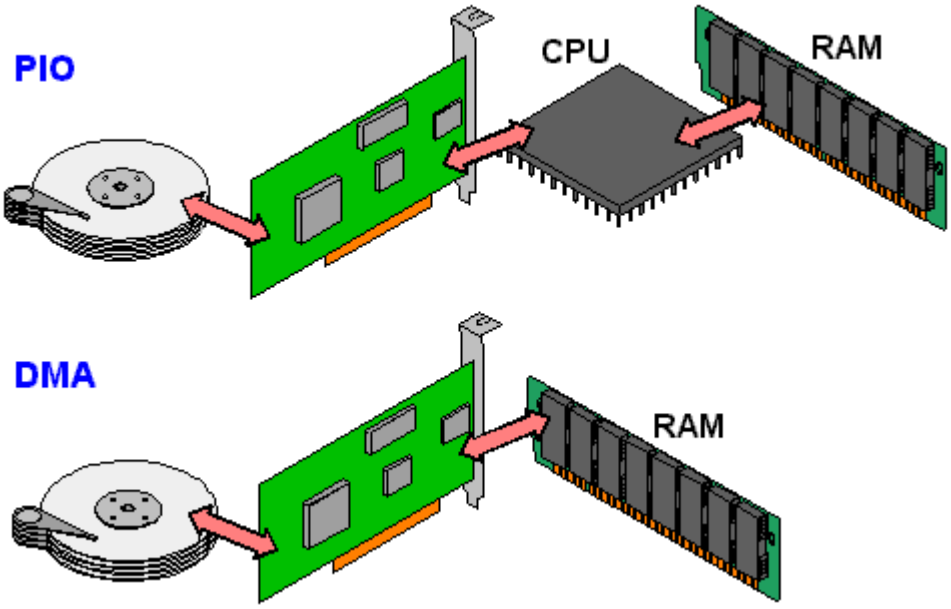

Рисунок 6.3 – Упрощенная схема работы DMA

! Ключевое слово extern – используется для определения глобальных переменных в языке Си.

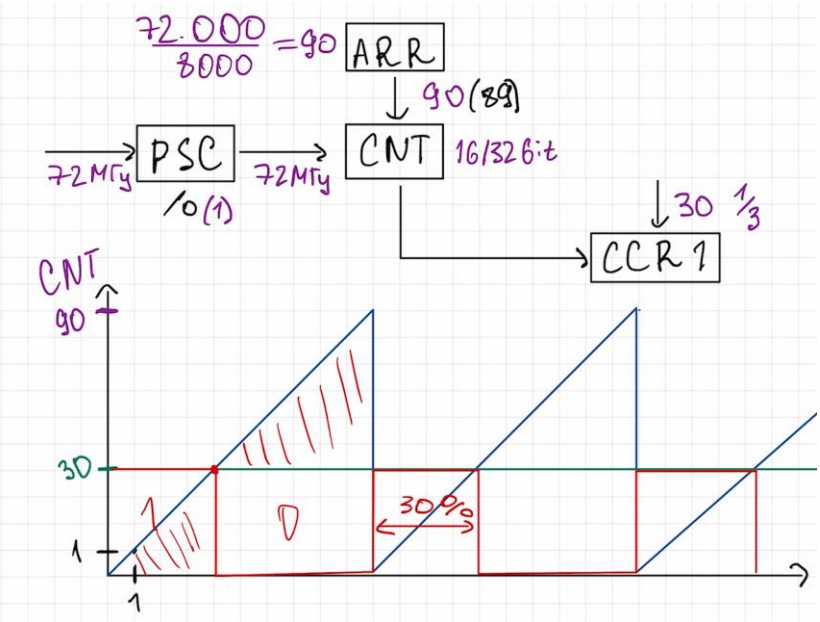

Рисунок 6.4 – Работа таймера с DMA

Для реализации работы использован таймер 4 и второй канал, который работает с первым DMA. Тактовая частота камня – 200 МГц. Были использованы делители напряжения, настроена длительность положительного и отрицательного бита.

Листинг 6.1 – Число в массиве, отвечающее за определенное значение

```
#define BIT_ON 32
#define BIT_OFF 68
```

Также я реализовал выбор цвета из 16 млн цветов с помощью RGB.

Листинг 6.2 – Функция подбора цвета

```
void set_color(uint16_t* one_led, const int red, int green, const int blue){
    int bit_buf = 0;
    int buf = green;
    int number = green;
    int binaryArray[8];
    for (int i = 0; i < 8; ++i) {
        binaryArray[i] = (number >> (7 - i)) & 1;
    }
    for(int i = 0; i < 8; ++i) {
        one_led[i] = BIT_ON * binaryArray[i];
        if(!binaryArray[i]) {
            one_led[i] = BIT_OFF;
        }
    }
    number = red;
    for (int i = 0; i < 8; ++i) {
        binaryArray[i] = (number >> (7 - i)) & 1;
    }
    int i_obr = 8;
    for(int i = 0; i < 8; ++i) {
        one_led[i_obr] = BIT_ON * binaryArray[i];
        if(!binaryArray[i]) {
            one_led[i_obr] = BIT_OFF;
        }
        i_obr++;
    }
     number = blue;
     for (int i = 0; i < 8; ++i) {
         binaryArray[i] = (number >> (7 - i)) & 1;
     }
    i_obr = 16;
    for(int i = 0; i < 8; ++i) {
        one_led[i_obr] = BIT_ON * binaryArray[i];
        if(!binaryArray[i]) {
            one_led[i_obr] = BIT_OFF;
        }
        i_obr++;
    }
}
```

Следующим шагом был выбор, какие светодиоды зажигать (по порядковому номеру).

Листинг 6.3 – Включение определенных светодиодов

```
void set_led_number(uint16_t* arr, int first_led, int last_led){
    int arr_index = 24;
    while(arr_index < BIT_NUMBER - 48){
        for(int i = 0; i < 24; ++i){
            arr[arr_index] = arr[i];
            arr_index++;
        }
    }
    while (arr_index < BIT_NUMBER) {
        arr[arr_index] = 100;
        arr_index++;
    }
}
```

Листинг 6.4 – Мигание светодиодов

```
void led_blink( uint16_t* src_buffer){


    static int blinkCounter = 0;
      if (blinkCounter % 1000 < 333) {
            set_color(src_buffer, 255, 0, 0);
            set_led_number(src_buffer, 0, 0);
      } else if (blinkCounter % 1000 > 333 && blinkCounter % 1000 < 666) {
            set_color(src_buffer, 0, 0, 255);
            set_led_number(src_buffer, 0, 0);
      } else {
            set_color(src_buffer, 255, 255, 255);
            set_led_number(src_buffer, 0, 0);
      }

      blinkCounter++;

      if (blinkCounter >= 1000) {
            blinkCounter = 0;
      }
}
```

Листинг 6.5 – Эффект волны у светодиодов

```
void led_wave( uint16_t* src_buffer){

    static int currentLED = 0;  // Variable to keep track of the current LED
    static unsigned char color[3] = {255, 255, 0};  // Initial color (Red)
    static int numLEDs = 5;  // Number of LEDs to light up consequently
    // Turn off the previously lit LEDs
    for (int i = 0; i < numLEDs; i++) {
      set_single_led_color(src_buffer,(currentLED + i) % 14, 0, 0, 0, 0);
    }

    // Move to the next starting LED
    currentLED = (currentLED + 1) % 14;
      // Turn on the next set of LEDs with the specified color
      for (int i = 0; i < numLEDs; i++) {
            set_single_led_color(src_buffer, (currentLED + i) % 14, color[0],
color[1], color[2], i * i * i * (255 / (numLEDs * numLEDs * numLEDs )));
      }
}
```

7. Реализация работы адресной светодиодной ленты

Листинг 7.1 – массив для вывода бинаризированного изображения на экран

```
#include "at32f403a_407_board.h"
#include "at32f403a_407_clock.h"
#include <bright_regim.h>
#include <string.h>
/* TMR AutoReload Value: for 16 bit tmr 0x0000~0xFFFF */
#define TMRx_PR 100

/** @addtogroup UTILITIES_examples
  * @{
  */
uint16_t PWM_DataLength=0;
tmr_output_config_type tmr_output_struct;

uint16_t src_buffer[360] = {};
/** @addtogroup AT32_TMR_DMA_Update_PWM_Duty_Cycle_Dynamically Duty
  * @{
  */

void crm_configuration(void);
void gpio_configuration(void);
void tmr_configuration(void);
void dma_configuration(void);
void TMR4_DMA_Duty_Cycle(void);

/**
  * @brief  configure the tmr pins.
  * @param  none
  * @retval none
  */
void gpio_configuration(void)
{
  gpio_init_type gpio_init_struct;

  gpio_default_para_init(&gpio_init_struct);

       /* configure PB7 tmr4_ch2 as output*/
  gpio_init_struct.gpio_pins = GPIO_PINS_7;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init(GPIOB, &gpio_init_struct);
}

/**
  * @brief  configure the tmr parameters.
  * @param  none
  * @retval none
  */
void tmr_configuration(void)
{
  /* Init TMR4 */
  tmr_base_init(TMR4, TMRx_PR-1, 2);
  tmr_cnt_dir_set(TMR4, TMR_COUNT_UP);

       /* TMR configuration as output mode */
```

```
  tmr_output_default_para_init(&tmr_output_struct);
  tmr_output_struct.oc_mode = TMR_OUTPUT_CONTROL_PWM_MODE_A;
  tmr_output_struct.oc_output_state = TRUE;
  tmr_output_struct.oc_polarity = TMR_OUTPUT_ACTIVE_LOW;

      /* TMR4 channel 2 configuration */
  tmr_output_channel_config(TMR4, TMR_SELECT_CHANNEL_2, &tmr_output_struct);

      /* enable tmr output channel buffer */
  tmr_output_channel_buffer_enable(TMR4, TMR_SELECT_CHANNEL_2, TRUE);
}


/**
  * @brief  configure the dma parameters.
  * @param  none
  * @retval none
  */
void dma_configuration(void)
{
      dma_init_type dma_init_struct = {0};
      uint16_t index = 0;

      while(index < PWM_DataLength)
      {
            src_buffer[index] = (uint16_t) (((uint32_t) (src_buffer[index]) *
(TMRx_PR)) / (100));
            index++;
      }

  /* dma1 channel7 configuration */
  dma_default_para_init(&dma_init_struct);

  dma_init_struct.buffer_size = PWM_DataLength;
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;
  dma_init_struct.memory_base_addr = (uint32_t)src_buffer;
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;
  dma_init_struct.memory_inc_enable = TRUE;
  dma_init_struct.peripheral_base_addr = (uint32_t)&TMR4->c2dt;
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;
  dma_init_struct.peripheral_inc_enable = FALSE;
  dma_init_struct.priority = DMA_PRIORITY_HIGH;
  dma_init_struct.loop_mode_enable = FALSE;
  dma_init(DMA1_CHANNEL7, &dma_init_struct);
}

/**
  * @brief  configures the different system clocks.
  * @param  none
  * @retval none
  */
void crm_configuration(void)
{
  /* tmr clock enable */
  crm_periph_clock_enable(CRM_TMR4_PERIPH_CLOCK, TRUE);

  /* gpio clock enable */
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

      /* dma clock enable */
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
```

```
}

/**
  * @brief  configures the PWM dma duty cycle parameters.
  * @param  none
  * @retval none
  */
void TMR4_DMA_Duty_Cycle(void)
{
      //led_blink(src_buffer);


      led_wave(src_buffer);
      for (int i = 0; i < 1000000; i++);

      /* set tmr channel CC value */
      tmr_channel_value_set(TMR4, TMR_SELECT_CHANNEL_2, 0);

  /* config set the number of data to be transferred by dma */
  dma_channel_enable(DMA1_CHANNEL7, FALSE);
      dma_data_number_set(DMA1_CHANNEL7, PWM_DataLength);
  dma_channel_enable(DMA1_CHANNEL7, TRUE);

      /* TMR enable counter */
      tmr_counter_enable(TMR4, TRUE);

      /* wait for the end of dma transfer */
      while (!dma_flag_get(DMA1_FDT7_FLAG));
      /* Clear dma flag */
      dma_flag_clear(DMA1_FDT7_FLAG);

      /* Clear TMR4 update Interrupt  pending bit */
      tmr_flag_clear(TMR4, TMR_OVF_FLAG);
  while(SET!=tmr_flag_get(TMR4, TMR_OVF_FLAG));
      /* Clear TMR4 update Interrupt  pending bit */
      tmr_flag_clear(TMR4, TMR_OVF_FLAG);
  while(SET!=tmr_flag_get(TMR4, TMR_OVF_FLAG));

      /* TMR disable counter */
      tmr_counter_enable(TMR4, FALSE);

}

/**
  * @brief  main function.
  * @param  none
  * @retval none
  */
int main(void)
{
      uint16_t new_led[BIT_NUMBER];
      size_t size = 360 * sizeof(uint16_t);
      turn_off_led(new_led);
      //set_color(new_led, 0, 45, 0);
      //set_single_led_color(new_led, 13, 0, 0, 255, 255);

      //set_led_number(new_led, 0, 0);
      memcpy(src_buffer, new_led, size);

      /* the Number of Pwm Output */
      PWM_DataLength = sizeof(src_buffer)/sizeof(src_buffer[0]);
```

```
  /* system clocks configuration */
  system_clock_config();

       /*board initialize interface init led and button    */
       at32_board_init();

       /* turn led2/led3/led4 on */
  at32_led_on(LED2);
  at32_led_on(LED3);
  at32_led_on(LED4);

       /* enable tmr/gpio clock */
       crm_configuration();

  /* tmr gpio configuration */
  gpio_configuration();

       /* dma configuration */
  dma_configuration();

       /* tmr configuration */
  tmr_configuration();

       /* enable tmr4 overflow dma request */
       tmr_dma_request_enable(TMR4, TMR_OVERFLOW_DMA_REQUEST, TRUE);


  while(1)
  {

              TMR4_DMA_Duty_Cycle();
              delay_us(1000);

       }
}
```

8. Заключение

В ходе работы было изучено много новых алгоритмов (например, превращение трех чисел RGB т 0 до 255 в массив, который для адресного светодиода означает определенный цвет), новые построения кода (например, объединения), использование флагов и побитовые операции, работа со статическими массивами и оптимизированная работа с ними, изучены принципы работы микропроцессоров, новые методы оптимизированной работы с производительностью (например, использование DMA), работа с делителями частоты для различных шин, работа с таймерами и принцип формирования ШИМ-сигнала с их помощью, использование переменный extern, которые можно использовать во всём проекте.

Также были изучены принципы сборки проекта в среде AT32IDE (основано на Eclipse), неоднократно была поведена отладка проекта,

изучены компиляторы для ARM-процессоров, а также получены навыки использования логического анализатора и осциллографа.

Список реализованных функций:

1. Вывод текста на экран
2. Вывод фигур на экран
3. Вывод бинаризированного изображения на экран
4. Горение светодиодной ленты определенным цветом, выбранным с помощью RGB
5. Мигание светодиодной ленты с заданной частотой
6. Включение отдельных светодиодов
7. Работа в режиме эффекта волны

В будущем проект будет дорабатываться, была попытка установки на него ОС для микроконтроллеров, но оказалось, что это практически не осуществимо. Поэтому следующая задача (уже вне курсового проекта) – соединение микроконтроллера с компьютером на базе Raspberry Pi по шине I2C.