ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

Курсовой проект по дисциплине

«Объектно-ориентированное программирование»

«VGA-консоль на основе Arduino»

Пояснительная записка

Выполнил студент

гр. 3331506/20401 _____ Леонтьев М. Л.

Работу принял _____ Ананьевский М.С.

Санкт-Петербург

2025 г.

Оглавление

Техническое задание 3

1. Введение 3

2. Теоретические сведения 3

2.1 Аппаратная часть VGA-консоли 3

2.2 Принципы формирования VGA-сигнала на Arduino 4

3. Подготовительные работы 5

3.1 Разработка схемы и пайка компонентов 5

3.2 Проектирование и 3D-печать корпуса 6

4. Ход работы 10

5. Программный код 10

5.1 Тетрис 10

5.2 Змейка 18

6. Результаты работы программы 25

7. Заключение 25

8. Список литературы 25

Техническое задание

Разработать простую игровую VGA-консоль на базе микроконтроллера Arduino. Реализовать аппаратную часть, позволяющую выводить изображение на VGA-монитор, управлять играми с помощью кнопок. Написать две игры на языке программирования C — «Тетрис» и «Змейка».

1. Введение

Современные микроконтроллеры предоставляют возможности для создания полноценных проектов в области цифровой электроники. Цель данного курсового проекта — собрать игровую VGA-консоль с помощью микроконтроллера Arduino Uno и разработать программное обеспечение для двух классических игр. Такой проект позволяет закрепить знания в области электроники, программирования на языке C и 3D-моделирования.

2. Теоретические сведения

2.1 Аппаратная часть VGA-консоли

VGA (Video Graphics Array) — стандарт аналогового видеовыхода, позволяющий выводить изображение на монитор с использованием сигналов синхронизации и аналоговых RGB-сигналов. Несмотря на возраст стандарта, его простота делает его удобным для реализации на микроконтроллерах.

Для формирования сигнала VGA использована плата Arduino Uno. VGA-разъем подключен через резистивный делитель к цифровым пинам Arduino, что позволяет формировать видеосигнал с минимальным количеством компонентов. Управление осуществляется с помощью тактовых кнопок, подключенных к цифровым входам с использованием подтягивающих резисторов. Схема подключения VGA представлена на рисунке 2.1.1.

Wheel	Pin Ax
1	A2
2	A1

Button	Pin Dx	Move
1	12	Left
2	11	Up (rotate)
3	10	Right
4	13	Down (P1)
5	5	P2

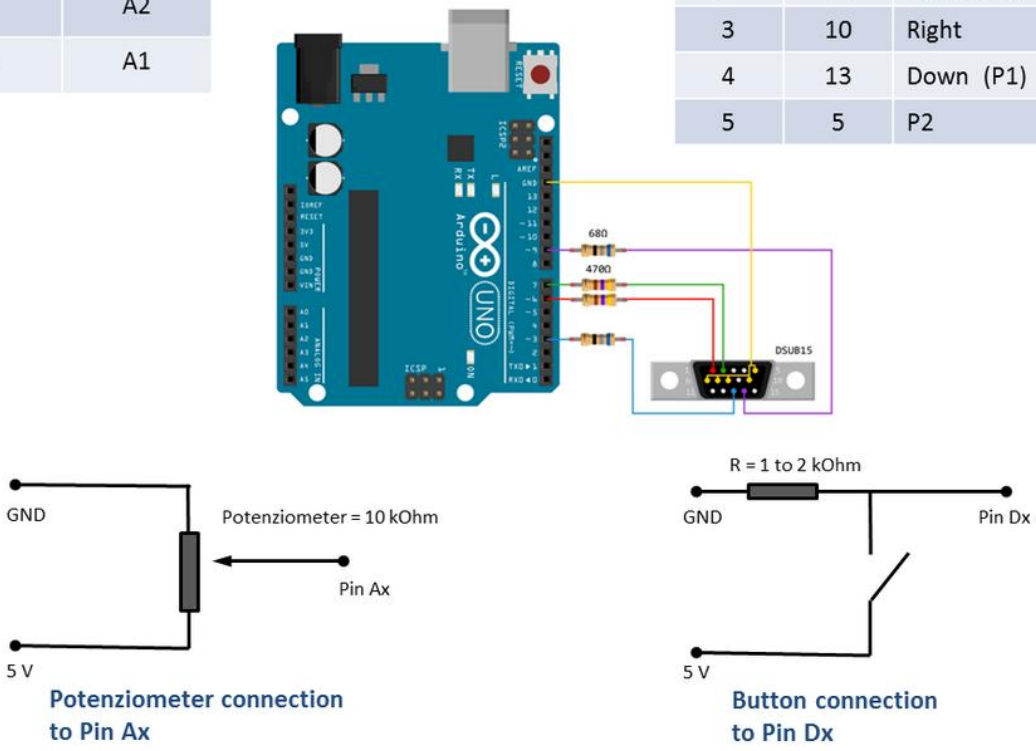

Рисунок 2.1.1 – Схема подключения

2.2 Принципы формирования VGA-сигнала на Arduino

Arduino работает на частоте 16 МГц, что накладывает определённые ограничения на разрешение и частоту обновления изображения. Однако, благодаря использованию низкоуровневого программирования и оптимизации кода, можно сформировать устойчивый VGA-сигнал, достаточный для отображения простой графики и текстов.

В рамках проекта используется специализированная библиотека **VGAX**, предназначенная для вывода изображения через VGA с микроконтроллеров семейства AVR, таких как Arduino Uno. Данная библиотека позволяет использовать **разрешение 120×60 пикселей** и поддерживает **4 фиксированных цвета**:

- чёрный (фон),

- красный,
- зелёный,
- жёлтый (как смесь красного и зелёного).

Такая цветовая схема достигается благодаря формированию сигналов по двум цветовым каналам — **R (красный)** и **G (зелёный)** — без использования синего, что упрощает схему вывода. Каждый пиксель кодируется двумя битами, что позволяет добиться простоты и высокой скорости обработки на Arduino.

Библиотека VGAX работает напрямую с портами микроконтроллера и использует прерывания таймера, чтобы точно соблюдать тайминги VGA-сигнала. Это делает её идеальным выбором для создания простых игр с минимальной графикой.

3. Подготовительные работы

3.1 Разработка схемы и пайка компонентов

Схема VGA-консоли была разработана вручную на основе базовых принципов формирования VGA-сигнала с использованием микроконтроллера Arduino. Конструкция проектировалась с учётом минимализма, надёжности и компактности — все компоненты были припаяны напрямую к плате Arduino, без использования макетной платы.

Формирование видеосигнала реализовано через два цветовых канала — **красный (R)** и **зелёный (G)** — что позволяет отобразить **четыре фиксированных цвета** (чёрный, красный, зелёный и жёлтый). Сигналы синхронизации (HSync и VSync) формируются программно с использованием встроенных таймеров микроконтроллера.

Вывод изображения осуществляется на стандартный VGA-разъём. Для согласования уровней сигнала используются **резистивные делители напряжения**, собранные из доступных резисторов — конкретные номиналы подбирались из имеющихся в наличии и обеспечивают приемлемое качество

сигнала на экране. Использование стандартных значений, таких как 330 Ом, не потребовалось — схема работоспособна и с другими значениями в допустимом диапазоне.

В качестве органов управления используются **четыре тактовые кнопки**, расположенные на корпусе устройства. Кнопки подключены к цифровым пинам Arduino и обеспечивают управление персонажем или курсором в играх. **Потенциометр в схеме не использовался**, так как аналоговый ввод не был необходим — все взаимодействие осуществляется цифровыми средствами.

Основные элементы схемы:

- Arduino Uno**,**
- VGA-разъём**,**
- Резисторы (номиналы подобраны из доступных компонентов)**,**
- 4 тактовые кнопки**,**
- Монтажные провода**,**
- Ручная пайка без макетной платы**,**
- Корпус, напечатанный на 3D-принтере**,** в который элементы были установлены вручную.

Собранная схема была тщательно проверена на работоспособность. Все соединения выполнены аккуратно, видеосигнал стабилен, нажатия кнопок корректно обрабатываются. Компактная конструкция позволила удобно разместить устройство внутри корпуса.

3.2 Проектирование и 3D-печать корпуса

Для надёжной фиксации компонентов VGA-консоли и обеспечения удобства пользования был разработан и напечатан на 3D-принтере индивидуальный корпус. Конструкция учитывает размеры платы Arduino, расположение кнопок и разъёмов, а также обеспечивает доступ к элементам управления.

Корпус состоит из двух частей — основания и крышки, которые плотно соединяются между собой. Внутренние элементы корпуса включают:

Крепёжные направляющие под плату Arduino — чтобы исключить её смещение;

Выемки под контакты и кабели — позволяют легко подключать VGA-разъём и питание;

Платформу для кнопок — плоскость с отверстиями, в которые фиксируются 4 тактовые кнопки;

Пазы для проводов и контактов, чтобы удобно расположить пайку и исключить повреждение.

На верхней крышке предусмотрены:

Четыре отверстия под кнопки, которые аккуратно выходят на поверхность;

Гладкая поверхность корпуса без лишних деталей;

Вентиляционное отверстие / декоративный логотип (вырез на боковой стенке), что добавляет дизайну индивидуальность.

Все элементы корпуса были смоделированы в CAD-программе с точным учётом размеров компонентов. Расположение кнопок удобно для управления в играх — они организованы в форме креста, имитируя стандартный D-pad.

Печать производилась на 3D-принтере методом FDM с использованием PLA-пластика. После печати детали были обработаны, собраны, и все компоненты аккуратно установлены внутрь. Плата Arduino была припаяна снаружи, затем аккуратно вставлена в корпус, при этом все провода и кнопки были заранее соединены и выведены наружу.

Результат — компактный, прочный и эстетичный корпус, подходящий для самостоятельной VGA-консоли, легко переносимый и устойчивый в эксплуатации. Корпус устройства представлен на рисунках 3.2.1–3.2.4**.**

Рисунок 3.2.1 – Корпус разъем под VGA

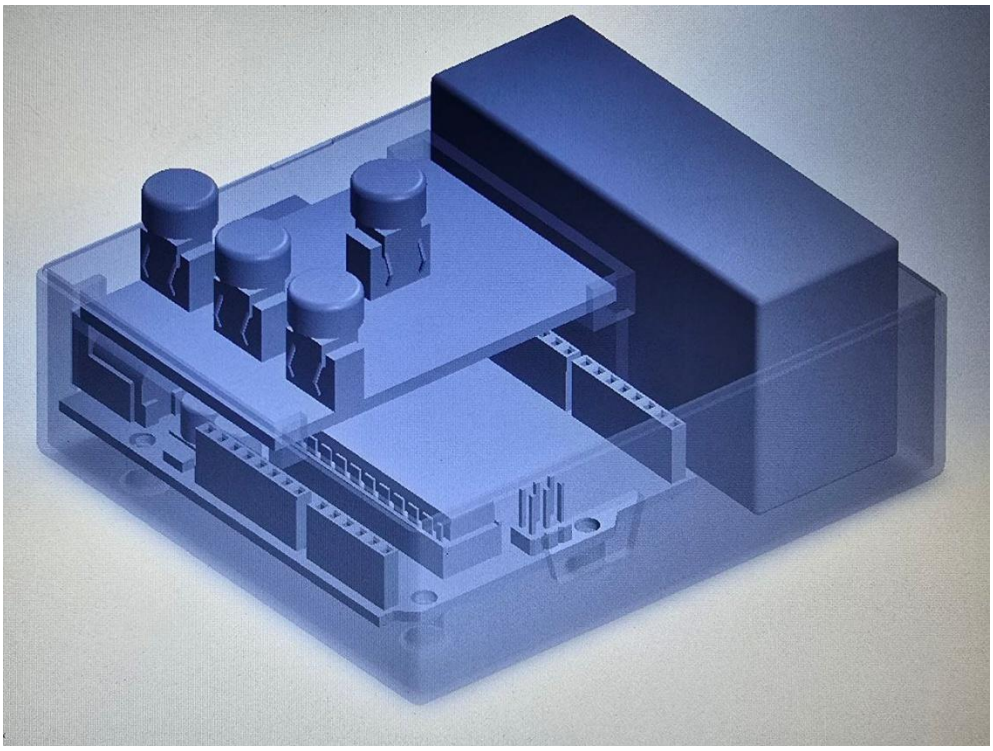

Рисунок 3.2.2 – Нижняя часть корпуса с расположенной электроникой

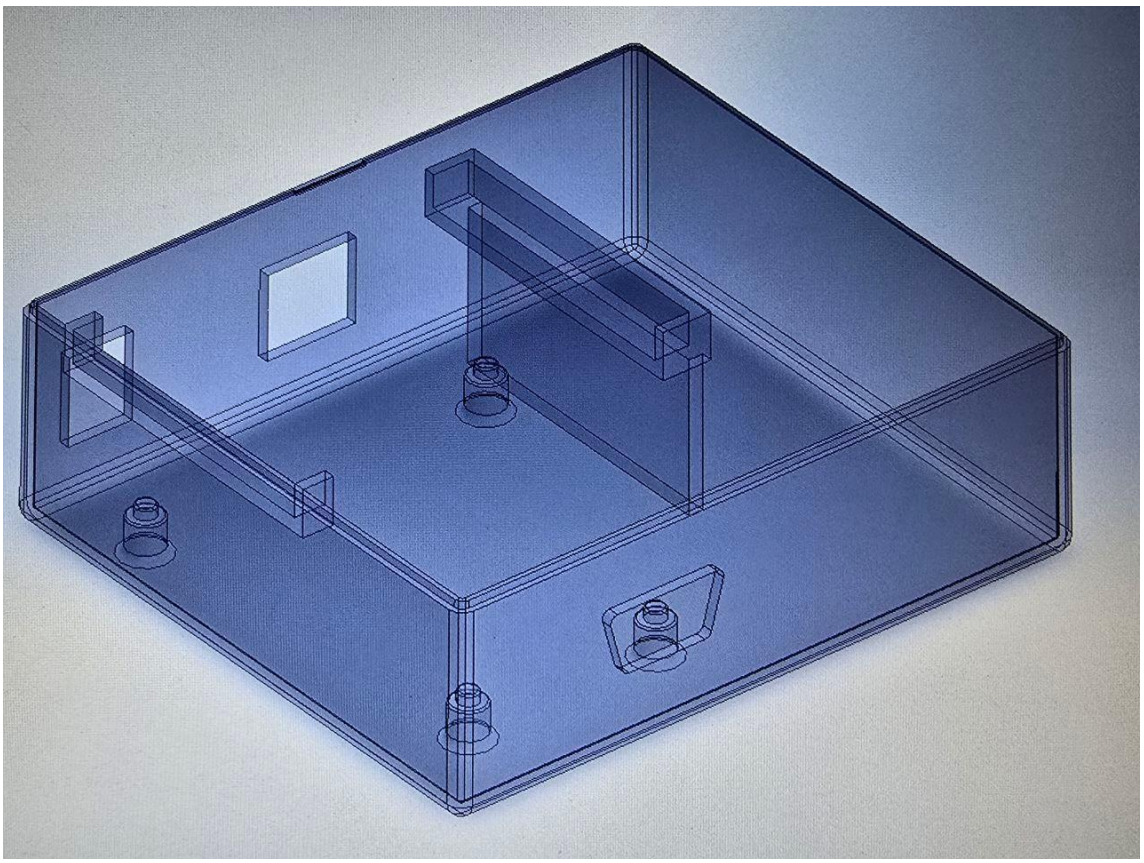

Рисунок 3.2.3 – Корпус изнутри

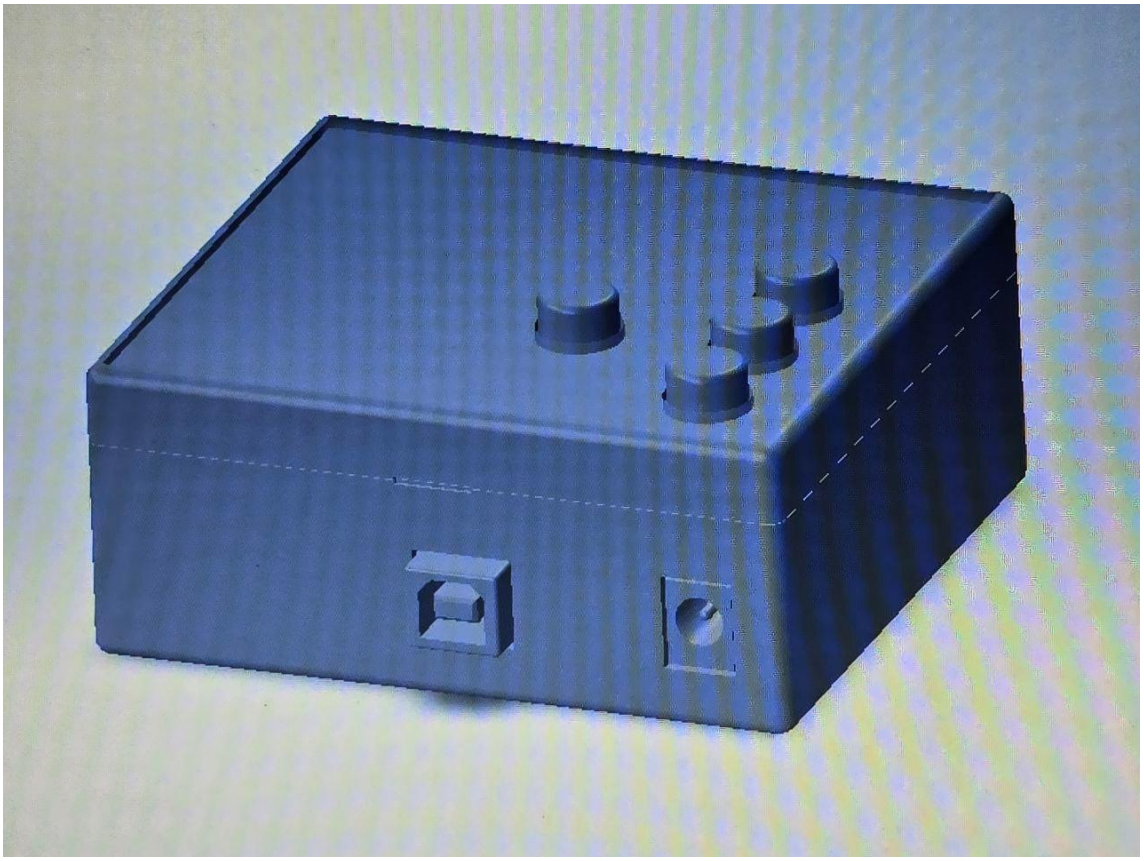

Рисунок 3.2.4 – Разъем под USB

4. Ход работы

- Разработана схема консоли.
- Подобраны необходимые компоненты (Arduino, VGA-разъем, резисторы, кнопки, макетная плата).
- Сформирована плата, выполнена пайка.
- Написан тестовый код для проверки VGA-сигнала.
- С использованием VGA-библиотеки реализована базовая графическая оболочка.
- Написаны игры:

Тетрис — управление падением фигур, проверка на заполнение строк, увеличение скорости.

Змейка — управление направлением, генерация еды, увеличение длины змейки.

- Отладка, устранение багов, тестирование всех режимов.
- Финальная сборка устройства, проверка работоспособности.

5. Программный код

5.1 Тетрис

Разбор программного кода для игры тетрис.

Данный код реализует классическую игру "Тетрис" для вывода на VGA дисплей с использованием Arduino. В отчете будет представлен подробный разбор структуры кода, основных функций и алгоритмов.

1. Подключение библиотек и определение констант

```
#include <math.h> // для математических функций
#define FNT_NANOFONT_HEIGHT 6 // Определяются константы для шрифт, Высота символов
#define FNT_NANOFONT_SYMBOLS_COUNT 95 // Количество символов
#include <VGAXUtils.h> // библиотека для работы с VGA выводом

VGAX vga;
VGAXUtils vgaU;
```

2. Определение шрифта

```
const unsigned char
fnt_nanofont_data[FNT_NANOFONT_SYMBOLS_COUNT][1+FNT_NANOFONT_HEIGHT]
PROGMEM={
{ 1, 128, 128, 128, 0, 128, 0, }, //glyph '!' code=0
{ 3, 160, 160, 0, 0, 0, 0, }, //glyph '"' code=1
 // ⋯ другие символы ⋯
};
```

Шрифт определен в виде массива данных, где каждый символ представлен битовой картой. Особенность - символ '#' заменен на полный прямоугольник для тестирования.

3. Текстовые строки

```
static const char str0[] PROGMEM="0";
static const char str1[] PROGMEM="1";
static const char str2[] PROGMEM="2";
static const char str3[] PROGMEM="3";
static const char str4[] PROGMEM="4";
static const char str5[] PROGMEM="5";
static const char str6[] PROGMEM="6";
static const char str7[] PROGMEM="7";
static const char str8[] PROGMEM="8";
static const char str9[] PROGMEM="9";
static const char str10[] PROGMEM="#";
static const char str11[] PROGMEM="Arduino VGA Tetris";
static const char str13[] PROGMEM="Game";
static const char str14[] PROGMEM="Over!";
```

Текстовые строки хранятся в памяти программ (PROGMEM) для экономии оперативной памяти. PROGMEM (от Program Memory) — это ключевое слово в

Arduino, которое указывает компилятору размещать данные не в оперативной памяти (RAM), а во флеш-памяти (Flash) микроконтроллера.

4. Функция setup()

```
void setup() {
  vga.begin();
  randomSeed(analogRead(5));
}
```

Инициализация:

- Запуск VGA
- Инициализация генератора случайных чисел

5. Глобальные переменные

Определены многочисленные переменные для управления игрой:

- Состояния кнопок (button, button_1 и т.д.)
- Массивы для хранения текущего, следующего и временных блоков
- Координаты и параметры игры (x, y, score и т.д.)
- Скорость игры (fast)
- Таймер (time)

6. Основные функции игры

6.1. Обработка ввода

```
void processInputs() {
  if(button_1 == 1) {
     button_2 = digitalRead(11);
     button_3 = digitalRead(12);
     button_4 = digitalRead(13);
     button_1 = 0;
       vga.delay(25);
  }
// ··· аналогично для других кнопок ···
    button = button_2 || button_4;

}
```

Эта функция обрабатывает ввод с кнопок, используя механизм антидребезга. Функция использует каскадную проверку условий, чтобы обрабатывать только одну кнопку за вызов:

1. Если button_1 была нажата (значение 1):

- Считываются состояния остальных кнопок (11-13)
- button_1 сбрасывается в 0
- Добавляется задержка 25 мс для антидребезга
- Функция завершает работу (не проверяет другие кнопки)

2. Если button_1 не нажата, проверяется button_2 и так далее по цепочке.

Переменная button примет значение 1 (true), если:

- button_2 нажата (равна 1) **ИЛИ**
- button_4 нажата (равна 1)

Если обе кнопки не нажаты, button получит 0 (false)

6.2. Меню игры

```
void drawMenu() {
  while (button_1 == 0 && button_2 == 0 && button_3 == 0 && button_4 == 0) {
     processInputs();
     vga.printPROGMEM((byte*)fnt_nanofont_data, FNT_NANOFONT_SYMBOLS_COUNT,
FNT_NANOFONT_HEIGHT, 3, 1, str11, 26, 16, (counterMenu%3) + 1);
     vga.delay(1000);
     counterMenu++;
  }
  vga.clear(0);
  drawGameScreen();
  drawScore(score);
}
```

Отображает анимированное меню с названием игры, пока пользователь не нажмет любую кнопку.

6.3. Отрисовка игровых элементов

Функции для рисования границ игрового поля и отображения счета.

```
void drawScore(int i) {
    if (i > 39){
       score = 0; // Сброс счёта при превышении 39
       i = 0;
```

```
                fast = fast - 4; // Увеличение скорости игры
                if (fast < 3) {fast = 2;} // Ограничение минимальной скорости
            }
            vgaU.draw_line(20, 10, 20, 50, 3); // Основная линия индикатора
            vgaU.draw_line(20, 50, 20, 50 - i, 1); // Заливка (текущий уровень)
            vgaU.draw_line(19, 10, 22, 10, 3); // Верхняя отметка
            vgaU.draw_line(19, 50, 22, 50, 3); // Нижняя отметка
        }

        void drawBorder() { // // Отрисовывает прямоугольную рамку игрового поля
        для тетриса
           // total screen size = 120/60
           // tetris game board: width = 30; heigh = 60
    vgaU.draw_line(44,0,78,0,3); // // Верхняя граница (x1=44,y1=0 → x2=78,y2=0, цвет=3)

    vgaU.draw_line(44,59,78,59,3); // Нижняя граница (x1=44,y1=59 → x2=78,y2=59, цвет=3)

    vgaU.draw_line(44,0,44,59,3); // Левая граница (x1=44,y1=0 → x2=44,y2=59, цвет=3)

    vgaU.draw_line(78,0,78,60,3); // Правая граница (x1=78,y1=0 → x2=78,y2=60, цвет=3)
}


        }
```

6.4. Работа с блоками

Определяет формы фигурок тетриса.

```
        void blockDef(int i) { // Принимает параметр i (от 1 до 7), который
        определяет тип фигуры
          if (i == 1){
          // O
          block[0][0] = 0;
          block[0][1] = 0;
          block[1][0] = 1;
          block[1][1] = 0;
          block[2][0] = 0;
          block[2][1] = 1;
          block[3][0] = 1;
          block[3][1] = 1;
          color = 1;
          }
        // ... другие формы ...
    }
```

Каждая фигура состоит из 4 блоков, где:

- block[n][0] — x-координата (относительно центра)
- block[n][1] — y-координата (относительно центра)

Выполняет масштабирование координат тетромино для корректного отображения на VGA-экране.

```
void blockExtension() {
   for (int i = 0; i < 4; i++){
      blockExt[0][0] = block[0][0]*3;
      blockExt[0][1] = block[0][1]*2;
      blockExt[1][0] = block[1][0]*3;
      blockExt[1][1] = block[1][1]*2;
      blockExt[2][0] = block[2][0]*3;
      blockExt[2][1] = block[2][1]*2;
      blockExt[3][0] = block[3][0]*3;
      blockExt[3][1] = block[3][1]*2;
   }
}
```

- Преобразует координаты из "логических" (относительных) в "физические" (пиксельные)
- Учитывает **пропорции 4:3** типичного VGA-экрана

6.5. Управление блоками

Функции для вращения и перемещения блоков на игровом поле.

```
void blockRotation(int clock){
// Вращение блока
  for (int i = 0; i < 4; i++){
// Сохранение старой позиции
      blockOld[0][0] = block[0][0];
      blockOld[0][1] = block[0][1];
      blockOld[1][0] = block[1][0];
      blockOld[1][1] = block[1][1];
      blockOld[2][0] = block[2][0];
      blockOld[2][1] = block[2][1];
      blockOld[3][0] = block[3][0];
      blockOld[3][1] = block[3][1];
  }
// Применение вращения
  for (int i = 0; i < 4; i++){
      block[0][0] = blockOld[0][1]*clock;
      block[0][1] = -blockOld[0][0]*clock;
      block[1][0] = blockOld[1][1]*clock;
      block[1][1] = -blockOld[1][0]*clock;
      block[2][0] = blockOld[2][1]*clock;
      block[2][1] = -blockOld[2][0]*clock;
      block[3][0] = blockOld[3][1]*clock;
      block[3][1] = -blockOld[3][0]*clock;
  }
```

```
}


void blockTranslation(int x, int y) {
// Перемещение блока
   for (int i = 0; i < 4; i++){
      blockTr[0][0] = blockExt[0][0] + x;
      blockTr[0][1] = blockExt[0][1] + y;
      blockTr[1][0] = blockExt[1][0] + x;
      blockTr[1][1] = blockExt[1][1] + y;
      blockTr[2][0] = blockExt[2][0] + x;
      blockTr[2][1] = blockExt[2][1] + y;
      blockTr[3][0] = blockExt[3][0] + x;
      blockTr[3][1] = blockExt[3][1] + y;
   }
}
```

6.6. Проверки и логика игры

Функции проверки игровых условий и обработки заполненных линий.

```
    void checkBlock(){
    // Проверка возможности размещения блока
      busy = 0;
      for (int i = 0; i < 4; i++){
         busy = busy + vga.getpixel(blockTr[i][0], blockTr[i][1])
    vga.getpixel(blockTr[i][0] + 2, blockTr[i][1]);
     }
    }

void checkForFullLine() {   // Проверка заполненных линий
   for (int i = 0; i < 4; i++){
      for (int j = 45; j < 76; j += 3) {
         if (vga.getpixel(j, blockTmp[i][1]) >0){k++; }
      }
      if (k == 11) {
// Удаление заполненной линии
         vgaU.draw_line(45, blockTmp[i][1], 78, blockTmp[i][1], 0);
// ... другие действия ...
      }
      k = 0;
    }
// Обновление счета
   if (yCounter != 0) {score = score + 2*int(pow(2, yCounter));}
}
```

7. Главный игровой цикл

```
void loop() {
  processInputs();
```

```
// Генерация нового блока
  if (noLoop < 1){ // Сгенерировать новую фигуру
     blockN = blockNext;
     if (noLoop == -1 ) { // Только в начале игры
        drawMenu();
        while (button_1 == 0 && button_2 == 0 && button_3 == 0 && button_4 == 0) {
           blockN = int(random(7)) + 1;
           processInputs();
        }
      // Инициализация нового блока
     // ... код инициализации ...
     }
// Обработка управления
  if (button_2 == 1){ // Вращение
     if (button_2 == 1){clock = -1;}
     //if (button_5 == 1){clock = 1;}
     delBlock();
     blockRotation(clock);
     checkBlockRotation();
  }
  if (button_1 == 1 || button_3 == 1){ // Перемещение
     if (button_1 == 1){delta = 3;}
     if (button_3 == 1){delta = -3;}
     delBlock();
     checkBlockTranslation();
  }
  time++;
// Падение блока
  if (time % fast > fast - 2 || button_4 == 1){ //

     if (fast < 3) {fast = 2;}
     y = y + 2;
     delBlock();
     replaceBlock();
  }
  vga.delay(10 + 2*fast);
}
```

Главный цикл игры обрабатывает:

- Ввод пользователя
- Генерацию новых блоков
- Управление блоками (вращение, перемещение)
- Автоматическое падение блоков
- Проверку игровых условий

5.2 Змейка

Разбор программного кода для игры "Змейка".

Данный код реализует классическую игру "Змейка" для вывода на VGA дисплей с использованием Arduino. В отчете будет представлен подробный разбор структуры кода, основных функций и алгоритмов.

1. Подключение библиотек, определение глобальных констант и пинов для кнопок управления.

```
#include <VGAX.h>          // Основная библиотека для работы с VGA
#include <math.h>          // Математические функции
#include <VGAXUtils.h>     // Утилиты для VGA (линии, фигуры)

#define FNT_NANOFONT_HEIGHT 6  // Высота шрифта
#define FNT_NANOFONT_SYMBOLS_COUNT 95  // Количество символов в шрифте

#define BUTTON_1 12  // Цифровой пин для кнопки 1
#define BUTTON_2 11  // Цифровой пин для кнопки 2
#define BUTTON_3 10  // Цифровой пин для кнопки 3
#define BUTTON_4 13  // Цифровой пин для кнопки 4
```

2. Определим массив символов, отображение которых задается побитово.

```
const unsigned char
fnt_nanofont_data[FNT_NANOFONT_SYMBOLS_COUNT][1+FNT_NANOFONT_HEIGHT] PROGMEM = {
    // Каждый символ задается битовой картой
    { 1, 128, 128, 128, 0, 128, 0, }, // Символ '!'
    { 3, 160, 160, 0, 0, 0, 0, }, // Символ '"'
    // ... остальные символы ...
};
```

3. Теперь определим текстовые строки.

```
// Текстовые строки в памяти программ
static const char str0[] PROGMEM="0";
static const char str1[] PROGMEM="1";
static const char str2[] PROGMEM="2";
static const char str3[] PROGMEM="3";
static const char str4[] PROGMEM="4";
static const char str5[] PROGMEM="5";
static const char str6[] PROGMEM="6";
static const char str7[] PROGMEM="7";
static const char str8[] PROGMEM="8";
static const char str9[] PROGMEM="9";
```

```
static const char str10[] PROGMEM="#";
static const char str20[] PROGMEM="Arduino VGA Snake";
static const char str22[] PROGMEM="Game Over";
static const char str23[] PROGMEM="Score";
```

4. Инициализация VGA и генератора случайных чисел.

```
void setup() {
  vga.begin();  // Инициализация VGA
  randomSeed(analogRead(5)); // Инициализация генератора случайных чисел
}
```

5. Запишем флаги для состояния кнопок, счетчик и состояние игры. Также определим начальные параметры.

```
// Состояние кнопок
boolean button1 = 0;
boolean button2 = 0;
boolean button3 = 0;
boolean button4 = 0;
boolean button;

// Счетчики и состояние игры
byte counterMenu = 0;     // Счетчик анимации меню
byte counterMenu2 = 0;    // Второй счетчик меню
byte state = 1;           // Текущее состояние игры (1-меню, 2-ожидание, 3-игра)
byte score = 0;           // Текущий счет
byte scoreMax = 10;       // Максимальный счет до увеличения скорости

// Позиция еды
byte foodX = 60;
byte foodY = 30;

// Данные змейки
byte snakeMaxLength = 55;  // Максимальная длина змейки
byte sx[55];               // Массив X-координат сегментов змейки
byte sy[55];               // Массив Y-координат сегментов змейки
byte slength = 3;          // Начальная длина змейки
byte delta = 5;            // Прирост длины при съедании еды
byte wleft = 36;           // Левая граница игрового поля

// Вспомогательные переменные
int i;                    // Индекс текущего сегмента
byte x, y;                // Текущее направление движения
byte direct = 3;          // Направление (1-вправо, 2-вверх, 3-влево, 4-вниз)
int speedDelay = 100;     // Задержка между движениями (скорость игры)
```

6. Функция, которая генерирует координаты еды в случайном месте, с учетом положения змейки.

```
void foodIni() {
  do {
     // Генерация случайных координат в пределах игрового поля
     foodX = random(VGAX_WIDTH - 4 - wleft) + 2 + wleft;
     foodY = random(VGAX_HEIGHT - 4) + 2;
  } while (vga.getpixel(foodX, foodY) > 1); // Проверка, чтобы еда не попала на
змейку
}
```

7. Функция обработки нажатий на кнопки.

```
void processInputs() {
  button1 = digitalRead(BUTTON_1);
  button2 = digitalRead(BUTTON_2);
  button3 = digitalRead(BUTTON_3);
  button4 = digitalRead(BUTTON_4);
  button = button1 | button2 | button3 | button4; // Любая кнопка нажата
}
```

8. Функция отрисовки меню.

```
void drawMenu() {
  counterMenu2++;
  vga.delay(10);
  if (counterMenu2 > 50) {  // Анимация мигания текста
    counterMenu++;
    vgaPrint(str20, 26, 16, (counterMenu%3) + 1); // Текст меняет цвет
    counterMenu2 = 0;
  }
}
```

9. Функция отрисовки границ игрового поля.

```
void drawBorder() {
    // Рисуем границы игрового поля разными линиями
    vgaU.draw_line(wleft, 0, VGAX_WIDTH-1, 0, 3);          // Верхняя
    vgaU.draw_line(wleft, VGAX_HEIGHT-1, VGAX_WIDTH-1, VGAX_HEIGHT-1, 3); // Нижняя
    vgaU.draw_line(wleft, 0, wleft, VGAX_HEIGHT-1, 3);     // Левая
    vgaU.draw_line(VGAX_WIDTH-1, 0, VGAX_WIDTH-1, VGAX_HEIGHT, 3); // Правая
}
```

10. Функция отрисовки счетчика игры.

```
void drawScore() {
  vgaPrint(str23, 10, 3, 2);  // Надпись "Score"
  vgaPrint(str10, 20, 10, 0); // Очистка места для цифры
  if (score < 10) {
      vgaPrintNumber(score, 20, 10, 2); // Однозначное число
  } else {
      // Для двузначных чисел
      vgaPrint(str10, 15, 10, 0);  // Очистка
      vgaPrintNumber(1, 15, 10, 2); // Первая цифра "1"
```

```
      // Вторая цифра
      if (score == 10) vgaPrintNumber(0, 20, 10, 2);
      if (score == 11) vgaPrintNumber(1, 20, 10, 2);
      if (score == 12) vgaPrintNumber(2, 20, 10, 2);
  }
}
```

11. Инициализация самой игры с отображением на экране положения змейки и еды. Змейка создается длиной в три сегмента, еда появляется по заданным координатам.

```
void drawSnakeIni() {
   // Инициализация видимой части змейки
   for (byte i = 0; i < slength; i++) {
      sx[i] = 80 + i;  // Горизонтальное расположение
      sy[i] = 30;      // Фиксированная Y-координата
      vga.putpixel(int(sx[i]), int(sy[i]), 2); // Отрисовка сегмента
   }

   // Инициализация неиспользуемых сегментов
   for (byte i = slength; i < snakeMaxLength; i++) {
     sx[i] = 1;
     sy[i] = 1;
   }

   vga.putpixel(foodX, foodY, 1); // Отрисовка еды
}
```

12. Функция нового раунда, в которой задаются начальные параметры.

```
void newMatch(){
  score = 0;
  slength = 3;
  i = slength - 1;
  vga.clear(0);
  drawBorder();
  drawScore();
  vga.putpixel(foodX, foodY, 1);
}
```

13. Основной цикл игры.

```
void loop() {
  processInputs(); // Считываем состояние кнопок
```

14. Игра работает в трех режимах: «Меню», «Ожидание старта» и «Игра». Эти режимы определяет состояние *state*. Реализовано это через условия *if*, в каждом из которых написана своя реализация игры.

15. Состояние 1 – Меню. Отображается название игры. Переход в следующий режим осуществляется через нажатие любой кнопки.

```
// Состояние 1 - Меню
if(state == 1) {
   drawMenu(); // Отрисовываем меню
   vga.delay(10);
   processInputs();
   if (button == 1) { // Если нажата любая кнопка
      button = 0;
      vga.clear(0);
      drawStartScreen();
      state = 2; // Переходим в состояние ожидания старта
   }
}
```

16. Состояние 2 – Ожидание старта. Задаются начальные параметры игры, счет. Ожидание нажатия кнопки для начала игры.

```
// Состояние 2 - Ожидание старта
if(state == 2) {
   if(score == scoreMax || score == 0) {
      processInputs();
   }
   if (button == 1) { // Старт игры
      score = 0;
      drawScore();
      button = 0;
      // Сброс всех флагов кнопок
      button1 = 0; button2 = 0; button3 = 0; button4 = 0;
      direct = 3; // Начальное направление - влево
      x = -1; y = 0; // Вектор движения
      i = slength - 1;
      state = 3; // Переход в игровое состояние
   }
}
```

17. Состояние 3 – Игровой процесс. В зависимости от нажатой кнопки меняется направление движения змейки. В нашем случае змейка может поворачивать только на 90°. С каждым циклом удаляется последний сегмент змейки и добавляется новый в ее начале.

```
// Состояние 3 - Игровой процесс
if(state == 3) {
   processInputs();

   // Обработка управления - изменение направления
   if (direct == 1) { // Движение вправо
```

```
   if (button2 == 1) { x = 0; y = -1; direct = 2; button4 = 0; } // Вверх
   if (button4 == 1) { x = 0; y = +1; direct = 4; } // Вниз
}
else {

   if (direct == 2){
      if (button1 == 1){ x = +1; y = 0; direct = 1; button3 = 0;}
      if (button3 == 1){ x = -1; y = 0; direct = 3;}
   }
   else {
      if (direct == 3){
         if (button2 == 1){ x = 0; y = -1; direct = 2; button4 = 0;}
         if (button4 == 1){ x = 0; y = +1; direct = 4;}
      }
      else {
         if (direct == 4){
            if (button1 == 1){ x = +1; y = 0; direct = 1; button3 = 0;}
            if (button3 == 1){ x = -1; y = 0; direct = 3;}
         }
      }
   }
}
// Удаление хвоста
vga.putpixel(int(sx[i]), int(sy[i]), 0);

// Добавление новой головы
if (i == slength - 1) {
   sx[i] = sx[0] + x;
   sy[i] = sy[0] + y;
} else {
   sx[i] = sx[i + 1] + x;
   sy[i] = sy[i + 1] + y;
}
```

18. Проверка на столкновение с едой. После столкновения с едой она появляется в новом месте, а сама змейка увеличивает длину на один сегмент. Счет увеличивается на единицу. Также осуществляется проверка счета: в случае если счет *score* превышает *scoreMsx*, то начинается новый раунд.

```
// Проверка столкновения с едой
if (sx[i] == foodX && sy[i] == foodY) {
   foodIni(); // Новая еда
   drawBorder();
   vga.putpixel(foodX, foodY, 1);
   score++;
   if (score > scoreMax) {
      speedDelay = speedDelay - 20; // Увеличение скорости
```

```
            newMatch(); // Новый раунд
        }
        drawScore();
    }
```

19. Проверка на столкновение с границами игрового поля или со своим телом. После появляется надпись «Game over» и игра начинается с начала.

```
    // Проверка столкновений
    if (vga.getpixel(int(sx[i]), int(sy[i])) == 0 ||
        vga.getpixel(int(sx[i]), int(sy[i])) == 1) {
        vga.putpixel(int(sx[i]), int(sy[i]), 2); // Нормальное движение
    } else {
        // Обработка Game Over
        vgaPrint(str22, 58, 24, 1); // Надпись "Game Over"
        vga.delay(300);
        button == 0;
        // Ожидание нажатия кнопки
        while(button == 0) { processInputs(); }
        newMatch(); // Новая игра
        drawSnakeIni(); // Отрисовка змейки
    }

    i--; // Переход к следующему сегменту
    if (i < 0) { i = slength - 1; } // Зацикливание индекса

    vga.delay(speedDelay); // Задержка для управления скоростью
  }
}
```

20. Функция, которая выводит текстовую строку.

```
// Вывод строки из PROGMEM
void vgaPrint(const char* str, byte x, byte y, byte color) {
   vga.printPROGMEM((byte*)fnt_nanofont_data, FNT_NANOFONT_SYMBOLS_COUNT,
                    FNT_NANOFONT_HEIGHT, 3, 1, str, x, y, color);
}
```

21. Функция, которая выводит числа.

```
// Вывод числа
void vgaPrintNumber(byte number, byte x, byte y, byte color) {
   char scoreChar[2];
   sprintf(scoreChar,"%d",number);
   vga.printSRAM((byte*)fnt_nanofont_data, FNT_NANOFONT_SYMBOLS_COUNT,
                 FNT_NANOFONT_HEIGHT, 1, 1, scoreChar, x, y, color);
}
```

6. Результаты работы программы

- Устройство корректно выводит VGA-сигнал на монитор.
- Управление осуществляется кнопками.
- В «Тетрисе» реализована смена фигур, подсчет очков.
- В «Змейке» корректно работает увеличение тела, игра завершается при столкновении.

7. Заключение

В ходе курсового проекта была собрана и запрограммирована простая игровая консоль на основе Arduino с VGA-выходом. Реализованы две игры, что позволило продемонстрировать возможности микроконтроллера в обработке графики и управления вводом. Проект помог развить навыки работы с аппаратной частью, пайкой, 3D-моделированием и программированием на языке C.

8. Список литературы

1. Документация Arduino Uno (официальный сайт)
2. Описание VGA сигнала: https://tinyvga.com
3. Библиотека VGAx: https://github.com/smaffer/vgax
4. Учебные материалы по программированию на языке C
5. Видеоуроки и статьи по пайке и 3D-моделированию