1 СТРУКТУРНАЯ СХЕМА И ОБОБЩЕННЫЙ АЛГОРИТМ РАБОТЫ

Раздел «Структурная схема и обобщенный алгоритм работы» содержит структурную схему, краткое описание назначения ее блоков, функции, выполняемые разрабатываемым устройством.

Устройство разработано на базе платформы EasyAVR v7 Development System фирмы mikroElektronika, на которой установлены микроконтроллер, графический ЖКИ, сенсорная панель.

1.1 Работа программы измерительного устройства

Измерительное устройство предназначено для измерения массы.

Интерфейс программы измерительного устройства представлен на рисунках 1-4.

После включения на дисплее отображаются три кнопки, масса, которая определяется с помощью потенциометра и АЦП и изображение рычажных весов, положение чаш которых будет меняться в зависимости от изменения измеряемого параметра.

Пользователь может с помощью сенсорных кнопок выбрать единицу измерения массы, которая будет отображаться на дисплее устройства. Доступны три единицы измерения: килограммы, фунты и стоуны.

Для определения массы объекта используется АЦП (аналого-цифровой преобразователь). Значение массы отображается на дисплее устройства и изменяется линейно вместе с значением напряжения АЦП. При изменении положения ручки потенциометра меняется значение массы.

При изменении значения массы меняется положение чаш весов, склоняя их вверх или вниз, в зависимости от значения массы. При максимальном значении массы чаша с объектом перевешивает влево, а при минимальном – вправо.

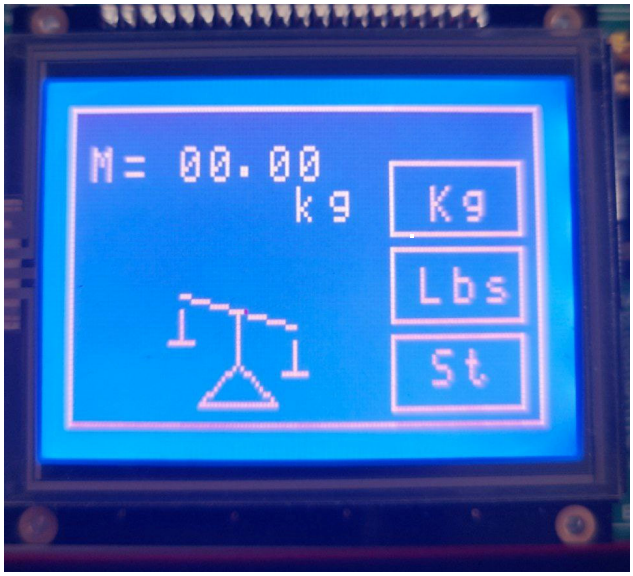

Рисунок 1 – Интерфейс программы (минимальное значение)

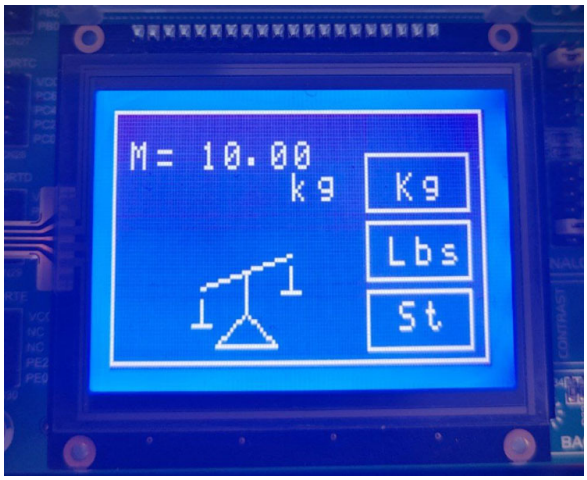

Рисунок 2 – Интерфейс программы (максимальное значение)

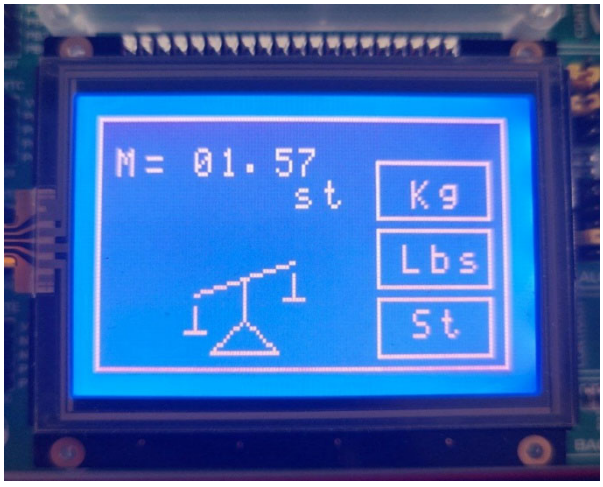

Рисунок 3 – Интерфейс программы (стоуны)

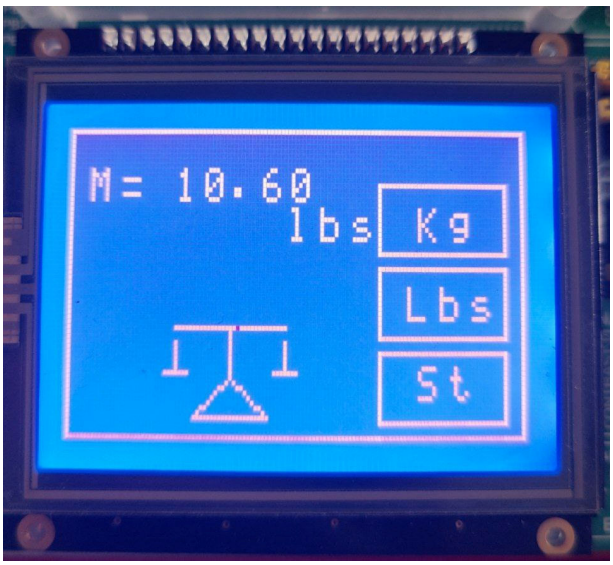

Рисунок 4 – Интерфейс программы (фунты)

1.2 Структурная схема измерительного устройства

Для успешной работы прибора используются следующие компоненты: потенциометр, АЦП, графический ЖКИ, сенсорная панель, микроконтроллер Atmega32 (МК), внутрисхемный программатор для загрузки программы в микроконтроллер (ВП) и источник питания (ИП).

Структурная схема представлена на рисунке 5.

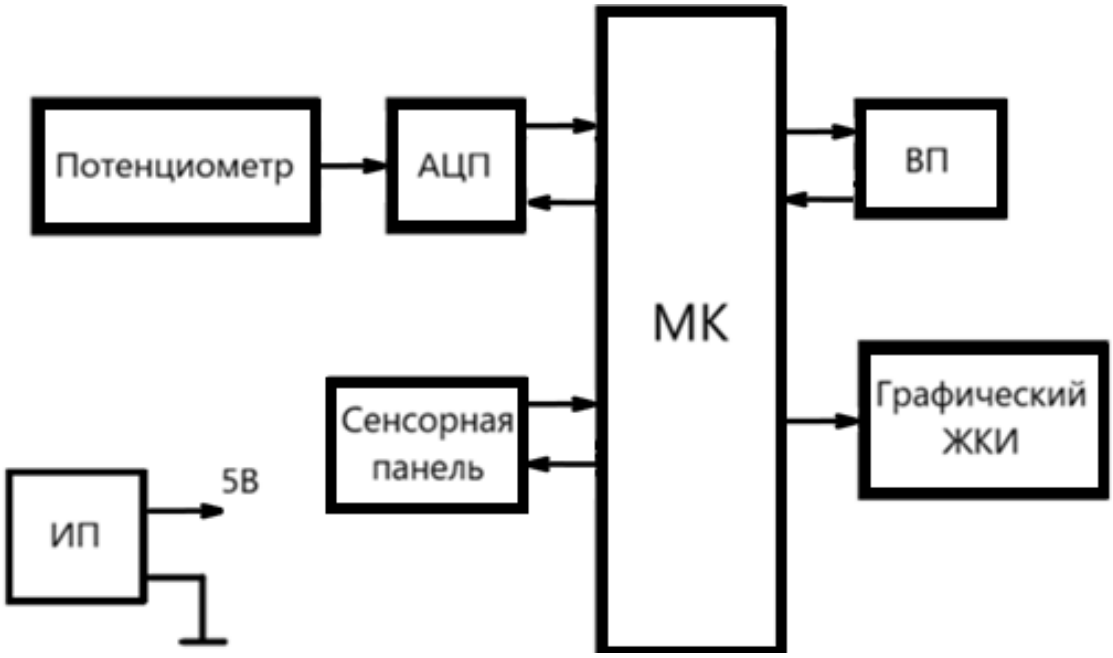

Рисунок 5 – Структурная схема

1.3 Описание назначения блоков структурной схемы

Данный раздел предназначен для описания составляющих элементов устройства.

1.3.1 Аналого-цифровой преобразователь

АЦП (Аналого-цифровой преобразователь) – это специализированная схема, которая может преобразовывать аналоговые сигналы (напряжение) в цифровое представление, обычно в виде целого числа. Значение этого числа линейно зависит от значения входного напряжения. Одними из наиболее важных параметров аналого-цифровых преобразователей являются время преобразования и разрешение. Время преобразования определяет, насколько быстро аналоговое напряжение может быть представлено в виде цифрового числа. Это важный параметр, если вам нужно быстрое получение данных. Второй параметр — разрешающая способность. Разрешающая способность представляет собой количество дискретных шагов, на которые можно разделить поддерживаемый диапазон напряжения. Он определяет чувствительность аналого-цифрового преобразователя. Разрешающая способность представлена максимальным количеством битов, которое занимает результирующее число. АЦП преобразовывает напряжение потенциометра в двоичный код. Схема подключения потенциометра представлена на рисунке 6.

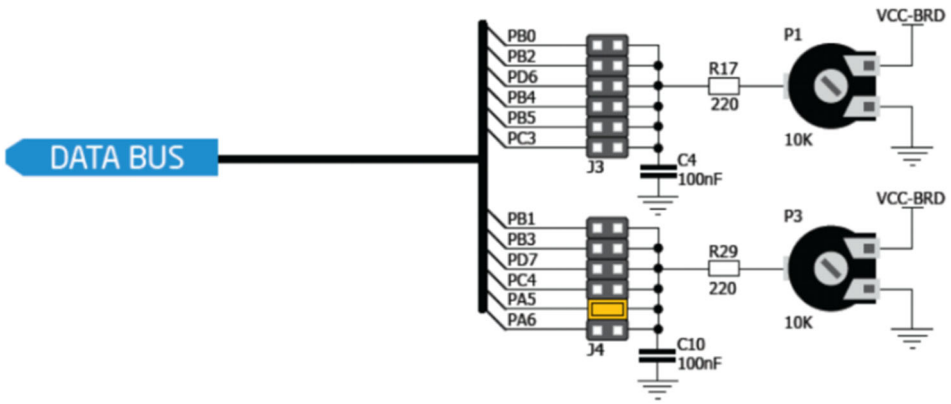

Рисунок 6 – Схема подключения потенциометра

1.3.2 Графический ЖКИ ME-GLCD 128X64

Графический жидкокристаллический дисплей используются для отображения текста, изображения. Связь с модулем дисплея осуществляется с помощью разъема дисплея CN16.

Разъем дисплея направляется к PB0, PB1, PA2, PA3, PD6, PD7 (линии управления) и PORTC (линии данных) гнезд микроконтроллера.

Порты PB0 и PB1подключаются к линиям за CS1 и CS2 соответственно. Они в свою очередь отвечают за сегменты с первого по шестьдесят четвертый и с шестьдесят пятого по сто двадцать восьмой соответственно.

Порт PA2 подключается к линии RS, которая отвечает за выбор регистра индикатора.

Порт PA3 подключается к линии R/W, которая отвечает за выбор типа операции (чтение или запись).

Порт PD6 подключается к линии E, которая отвечает за разрешение сигналов.

Порт PD7 подключается к линии RST, которая отвечает за сбрасывание LCM.

Порты PC0-PC7 отвечают за линии шин данных.

Схема подключения представлена на рисунке 7.

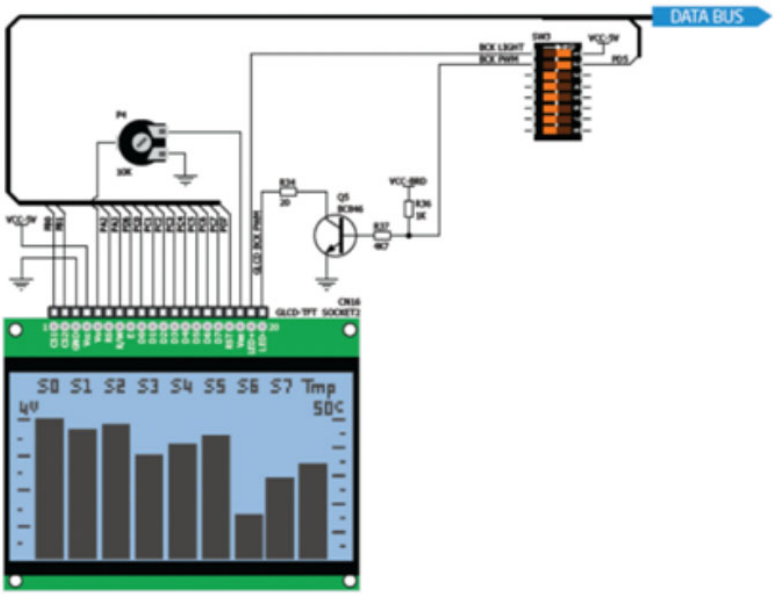

Рисунок 7 – Схема подключения графического ЖКИ

1.3.3 Сенсорная панель

Сенсорная панель – это стеклянная поверхность, покрытая двумя слоями резистивного материала. Когда на нее оказывается давление, слои сжимаются, и контакты на них сближаются, что позволяет определить координаты нажатия. При нажатии на сенсорную панель электрический ток проходит через верхний слой и находит заземление через нижний слой. Чтобы работать с сенсорной панелью необходимо:

а) установить логическую "единицу" на линию PA2 и логический "ноль" на линию PA3;

б) Обеспечить задержку более двух миллисекунд;

в) Запустить АЦП канал "ноль", полученный код АЦП будет координатой x;

г) Установить логический "ноль" на линию PA2 и логическую "единицу" на линию PA3;

д) Обеспечить задержку более двух миллисекунд;

е) Запустить АЦП канал "единица", полученный код будет координатой y;
Схема подключения сенсорной панели представлена на рисунке 8.

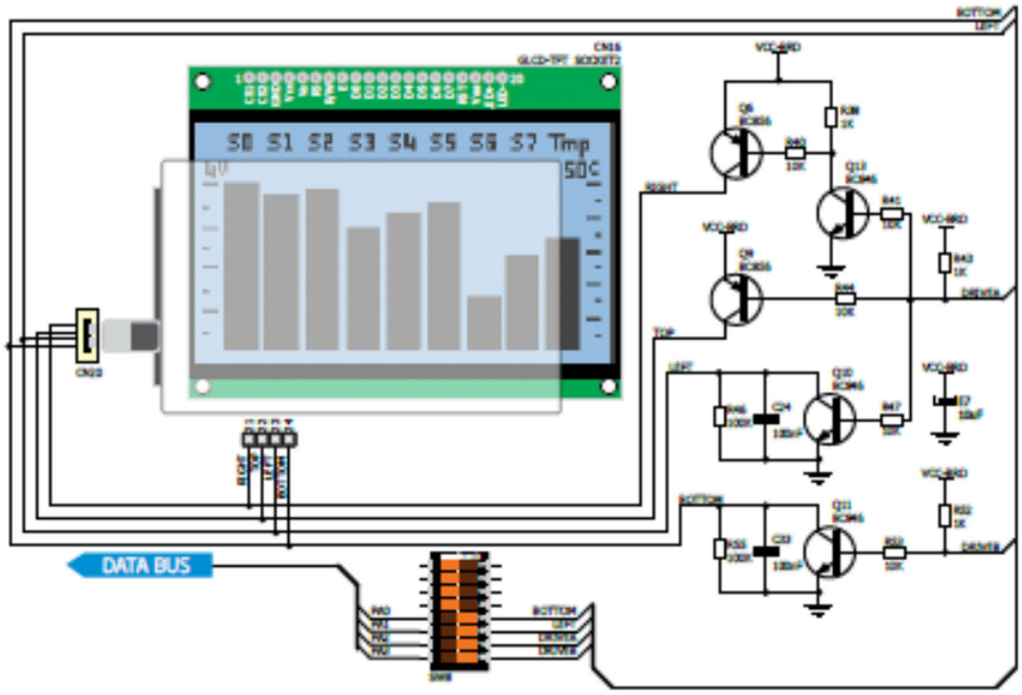

Рисунок 8 – Схема подключения сенсорной панели

1.3.4 Микроконтроллер Atmega32

Atmel AVR ATmega32 представляет собой микроконтроллер с низким энергопотреблением, основанный на улучшенной RISC-архитектуре AVR. Он также имеет две килобайта SRAM и один килобайт EEPROM для хранения данных. Схема расположения выводов микроконтроллера представлена на рисунке 9.

ATmega32 поддерживает различные интерфейсы, такие как SPI, I2C, USART и USB. Выполняя мощные инструкции за один такт, ATmega32 достигает мощности, приближающейся к одному миллиону инструкций в секунду на мегагерц, что позволяет разработчику системы оптимизировать энергопотребление в зависимости от скорости обработки. Все тридцать два регистра напрямую подключены к арифметико-логическому устройству (АЛУ), что позволяет использовать два независимых регистра, доступ к которым осуществляется одной командой, выполняемой за один такт. Получившаяся архитектура более эффективна в коде, обеспечивая при этом пропускную способность в десять раз быстрее, чем обычные микроконтроллеры CISC.

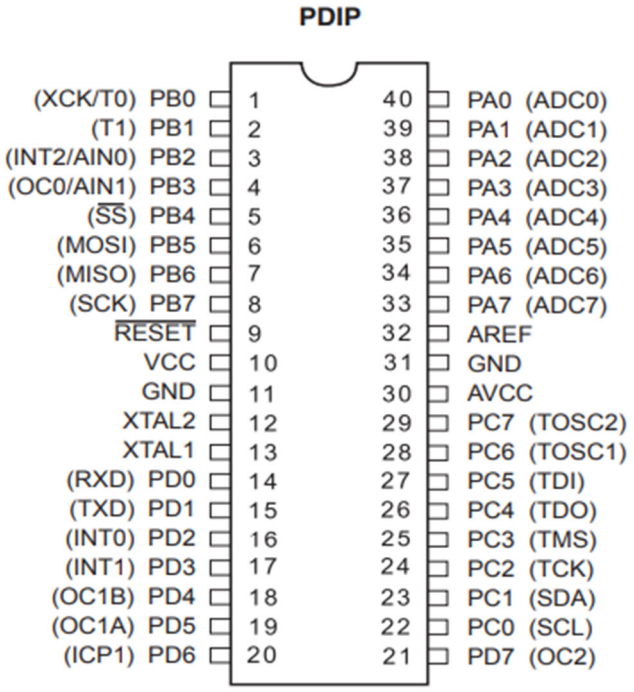

Рисунок 9 – Схема расположения выводов Atmega32

2 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИЗМЕРИТЕЛЬНОГО УСТРОЙСТВА

Раздел «Разработка программного обеспечения измерительного устройства» содержит описание всех компонентов программы измерительного устройства.

2.1 Начальная настройка

Начальная настройка в коде необходима для подготовки программы к работе. Она включает в себя подключение необходимых библиотек, инициализацию функций и переменных и другие действия.

2.1.1 Подключение библиотек

Подключение библиотек позволяет использовать готовые функции и методы. Без подключения библиотек некоторые функции не будут доступны для использования.

Библиотека «avr/pgmspace.h» позволяет экономнее использовать оперативную память микроконтроллера и работать с большими объемами данных. Библиотека «avr/io.h» позволяет работать с портами ввода и вывода микроконтроллера. Библиотека «util/delay.h» позволяет выполнять задержку по времени. Библиотека «avr/interrupt.h» позволяет обрабатывать прерывания. Библиотека «glcd.h» – библиотека функций графического дисплея.

Определение библиотек представлено на рисунке 10.

```
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "glcd.h"
```

Рисунок 10 – Определение библиотек

2.1.2 Инициализация функций

Для работы программы измерительного устройства необходимы следующие функции:

- «void init(void)» – инициализирует порты;
- «void timer_init(void)» – инициализирует таймеры;
- «void init_adc(void)» – инициализирует АЦП;
- «void get_mass(void)»–опрашивает АЦП;
- «void show_mass(void)»–отображает массу на графическом ЖКИ;
- «void scan_key(void)» – сканирует сенсорные кнопки;

Инициализация функций представлена на рисунке 11.

```
void init(void);
void timer_init(void);
void init_adc(void);
void get_mass(void);
void show_mass(void);
void scan_key(void);
```

Рисунок 11 – Инициализация функций

2.1.3 Объявление глобальных переменных

Глобальная переменная – это переменная, которая объявлена вне какой-либо функции и доступна для использования во всех функциях программы. Она используется для хранения данных, которые доступны в разных частях программы. Объявление глобальных переменных представлено на рисунке 12.

```
volatile unsigned char sec_flag=1, new_temp_flag=0;
volatile unsigned int ms_counter=0;
unsigned long long int temp_code_ADC0=0;
volatile unsigned char temp_ADC=0;
volatile unsigned int t_ADC=0;
volatile unsigned char mode=0; // переменная режима измерений
unsigned char adc[4]={0,0,0,0};
// Переменные функции опроса кнопок
volatile unsigned char gread_key=0, gkey_up_counter=0, gkey_status=0, gkey_down_counter=0;
volatile unsigned char gkey=0,isOn=0,isOn1=0,isOn2=0;
volatile int yl = 0, yr = 0, prov=0;
// Переменные для хранения текущих координат точки касания
volatile unsigned int x_coordinate=0;
volatile unsigned int y_coordinate=0;
```

Рисунок 12 – Глобальные переменные

2.2 Основная функция программы

Функция main является точкой входа в программу на языке С. Она должна быть определена в программе и содержать основной код, который будет выполняться при запуске программы.

2.2.1 Вызов функций и запрет прерываний

Сначала с помощью команды «asm («cli»)» запрещаем прерывания. Запрет прерываний используется для того, чтобы защитить микроконтроллер от сигналов, исходящих из вне. Эти сигналы могут навредить записи данных.

Затем инициализируем порты с помощью функции«init()».

После этого следует выполнить рекомендованную последовательность инициализации дисплея. Сначала используется функция «_delay_ms(100)», которая создает задержку в выполнении программы на сто миллисекунд. Затем дисплей выключается при помощи функции «glcd_off()» и снова происходит задержка на 100 миллисекунд. После этого дисплей включается функцией «glcd_on()» и очищается при помощи «glcd_clear()».

Затем происходит инициализация таймера счетчика с помощью функции «timer_init()», что позволяет совершать прерывания с интервалом в 8 миллисекунд.

После, необходимо вызвать функцию «init_adc()» для инициализации АЦП.

Вызов функций и запрет прерываний представлены на рисунке 13.

```
int main(void)
{
    asm ("cli");        //запрет прерываний асемблера
    init();             //инициализация портов
    // реализуем рекомендованную последовательность инициализации дисплея
    _delay_ms(100);
    glcd_off();         //выключаем дисплей
    _delay_ms(100);     //задержка
    glcd_on();          //включаем после задержки
    glcd_clear();       //очистка дисплея
    //
    timer_init();       //инициализация таймера счетчика (прерывания при совпадении, интервал 8 мс)
    init_adc();         //инициализация АЦП (как в предыдущем примере)
```

Рисунок 13 – Вызов функций и запрет прерываний

2.2.2 Отображение заставки

Для того, чтобы отобразить внешний контур интерфейса, кнопки и значения внутри кнопок, а также отображаемое значение необходимы следующие функции: «rectangle()», «glcd_puts()», «glcd_putchar()».Отображение заставки представлено на рисунке 14.

Функция «rectangle()» отображает прямоугольник на дисплее. В нее задаются следующие параметры: unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, byte s, byte c,где:

- unsigned int x1 – координатах для левого верхнего угла;
- unsigned int y1 – координатау для левого верхнего угла;
- unsigned int x2 – координатах для правого нижнего угла;
- unsigned int y2 – координатау для правого нижнего угла;
- byte s – тип линии (0 – сплошная, 1 – точечная и 2 –пунктирная);
- byte c – цвет линии (0 –прозрачная, 1 – непрозрачная);

Функция «glcd_puts()» отображает данные, записанные в флэш-памяти на дисплей. В нее задаются следующие параметры: byte *c, int x, int y, unsigned char l, byte sz, signed char space, где:

- byte *c – указатель на строку, которая будет записана на дисплее;
- int x – номер столбца, с которого нужно начать ввод символа (один символ занимает 8 столбцов);
- int y – номер строки, в которой нужно ввести символ;
- unsigned char l–язык выведенного текста (0 – английский, 1 – арабский или фарси);
- byte sz – размер шрифта (от нуля до семи);
- signed char space - интервалы:

Английский: Интервал между символами;

Арабский и фарси: интервалы между словами;

Функция «glcd_putchar()» записывает символ в указанную позицию с указанным размером. Работает по схожему принципу с «glcd_puts()», принимая те же значения.

```
rectangle(0,0,126,62,0,1);  //отрисовка внешнего контура

// вывод прямоугольников виртуальных "кнопок"
rectangle(86, 11, 121, 25, 0, 1); //kg
rectangle(86, 28, 121, 42, 0, 1); //lbs
rectangle(86, 45, 121, 59, 0, 1); //ct

// вывод текстовой заставки
glcd_puts("M=",base,1,0,1,1);

//Вывод числового значения величины
glcd_putchar((adc[3]+48),base+8*3,1,0,1);
glcd_putchar((adc[2]+48),base+8*4,1,2,1);
glcd_putchar((adc[1]+48),base+8*5,1,2,1);
glcd_puts(".",base+8*5,1,0,1,1);
glcd_putchar((adc[0]+48),base+8*7,1,2,1);
glcd_puts("kg ",60,2,0,1,1);

// Размещаем буквы в центр виртуальных "кнопок"
glcd_puts("Kg",96,2,0,1,1);
glcd_puts("Lbs",93,4,0,1,1);
glcd_puts("St",96,6,0,1,1);
```

Рисунок 14 – Отображение заставки

2.2.3 Отображение основания «весов» и разрешение прерываний

В целях улучшения дизайна интерфейса измерительного устройства были добавлены визуализированные весы, которые изменяют положение чаш в зависимости от значения измеряемой величины. В данном фрагменте кода мы отображаем основание весов, которое не будет изменяться с изменением значения.

Команда «asm («sei»)» разрешает прерывания. Отображение основания «весов» и разрешение прерываний представлено на рисунке 15.

```
line(45, 40, 45, 50, 0, 1);
line(45, 50, 35, 58, 0, 1);
line(45, 50, 55, 58, 0, 1);
line(35, 58, 55, 58, 0, 1);

asm ("sei");    //разрешаем прерывания
```

Рисунок 15 – Отображение основания «весов» и разрешение прерываний

2.2.4 Основная программа

В данном фрагменте кода происходит бесконечный цикл «while(1)», в котором проверяются два условия.

Первое условие «if ((new_temp_flag==1))» проверяет, изменилась ли масса, и если да, то вызывает функцию «show_mass()» для отображения значения массы на дисплее, а также сбрасывает флаг «new_temp_flag».

Второе условие «if ((gkey_status&0b10000000)!=0)» проверяет, была ли нажата какая-либо кнопка. Если да, то сбрасывает старший бит «gkey_status» и проверяет, какая именно кнопка была нажата (gkey). В зависимости от того, какая кнопка была нажата, меняется режим измерения массы (mode) и выводится соответствующее обозначение на дисплей. Основной цикл программы представлен на рисунке 16.

```
    while(1)
    {
        if ((new_temp_flag==1)){
            show_mass();
            new_temp_flag=0;
        }

        if ((gkey_status&0b10000000)!=0)
        {
            gkey_status=gkey_status&0b01000000;
            if (gkey==0){
                PORTB^=(1<<PB7);
                mode=0; // Килограммы
                glcd_puts("kg ",60,2,0,1,1);
            }
            if (gkey==1){
                PORTB^=(1<<PB6);
                mode=1; // Фунты
                glcd_puts("lbs",60,2,0,1,1);
            }
            if (gkey==2){
                PORTB^=(1<<PB5);
                mode=2; // Стоуны
                glcd_puts("st ",60,2,0,1,1);
            }
        }
    }
    return 0;
}
```

Рисунок 16 – Основной цикл программы

2.3 Обработчик прерываний

Этот код является обработчиком прерывания от события совпадения таймера TIMER0. Когда происходит это прерывание, переменная «ms_counter» увеличивается на единицу. Далее проверяется, равна ли переменная «ms_counter» десяти. Если это так, то переменная «ms_counter» сбрасывается в ноль, устанавливается глобальный флаг времени «sec_flag» и вызывается функция «scan_key()» для сканирования нажатий.

Далее проверяется переменная «new_temp_flag», которая указывает, было ли отображено предыдущее значение массы. Если значение равно нулю, то вызывается функция «get_mass()», которая считывает значение кода АЦП канала измерения массы и устанавливает значение «new_temp_flag» в единицу.

Таким образом, этот код используется для периодического сканирования клавиатуры и считывания значения массы с АЦП. Фрагмент кода с обработчиком прерываний представлен на рисунке 17.

```
ISR (TIMER0_COMP_vect)
{
    ms_counter++; // Счетчик прерываний

    if(ms_counter==10)
    {
        ms_counter=0;
        sec_flag=1; //глобальный флаг времени
        scan_key();
        if (new_temp_flag==0){ //проверяем факт отображения предыдущего значения массы
            get_mass(); // считать значение кода АЦП канала измерения массы
            new_temp_flag=1;
        }
    }
}
```

Рисунок 17 - Обработчик прерываний

2.4 Функция отображения массы и чаш «весов»

2.4.1 Вычисление значений

Этот фрагмент кода осуществляет преобразование аналогового сигнала, который поступает на вход АЦП (аналогово-цифровой преобразователь), в цифровое значение.

Сначала устанавливается регистр ADMUX равным 0b00000101, что означает, что АЦП будет использовать вход ADC5 для преобразования.

Затем устанавливается бит ADSC в регистре ADCSRA, что запускает процесс преобразования. Далее происходит ожидание конца преобразования АЦП, пока бит ADSC не станет равным нулю.

Затем, в зависимости от выбранного режима работы (mode), вычисляется значение массы в выбранных единицах измерений (килограммах, фунтах или стоунах). Для этого используется переменная «t_ADC», которая содержит результат преобразования АЦП, а также соответствующие коэффициенты пересчета для каждого режима работы.

В результате выполнения этого кода переменная «temp_code_ADC0» будет содержать значение массы в выбранных единицах измерений. Вычисление значений представлено на рисунке 18.

```
void show_mass (void){
    ADMUX=0b00000101;
    ADCSRA|=(1<<ADSC);      // запускаем АЦП ( аналогично ADCSRA=((ADCSRA|0b01000000));
    while ((ADCSRA&(1<<ADSC))!=0); //ожидаем конца преобразования АЦП

    // Проверка режима работы и вычисление значения массы в выбранных единицах измерений
    if (mode==0) temp_code_ADC0=t_ADC * 0.9775171065493646; //Вычисляем kg

    if (mode==1) temp_code_ADC0=t_ADC * 2.155425219941349;  //Вычисляем lbs

    if (mode==2) temp_code_ADC0=t_ADC * 0.15393970181879757;    //Вычисляем st
```

Рисунок 18 – Вычисление значений

2.4.2 Отображение изображения «весов»

Переменная «yl» отвечает за координату по оси улевой чаши, «yr» за за координату по оси управой чаши.

Значения тридцать семь и сорок три – начальные точки, при нулевом напряжении АЦП. В зависимости от изменения значения напряжения потенциометра изменяются значения переменных «yl» и «yr».

Далее проверяется переменная prov, которая отвечает за проверку изменения значения напряжения АЦП. Если значение меняется, она закрашивает ближайшие к

положению стержня и чаш клетки в прозрачный цвет. Это необходимо для того, чтобы предыдущая линия стерлась и не мешала восприятию картины.

Далее рисуются сами стержень и чаши. Стержень разделен на две части – правую и левую и создается функцией «line()».Отображение изображения «весов» представлено на рисунке 19.

```
yl = 37 + t_ADC / 139;
yr = 43 - t_ADC / 139;

if (prov != yl) {
    for (int i = 1; i < 5; i++) {
        line(45, 40, 30, yl + i, 0, 0);
        line(45, 40, 30, yl - i, 0, 0);

        line(45, 40, 60, yr + i, 0, 0);
        line(45, 40, 60, yr - i, 0, 0);

    }
    for (int w = 5; w < 13; w++) {
        line(27, yl + w, 33, yl + w, 0, 0);

        line(57, yr + w, 63, yr + w, 0, 0);
    }
    line(60, yr + 2, 60, yr + 10, 0, 0);

    line(30, yl + 2, 30, yl + 10, 0, 0);
}
line(47, 40, 47, 50, 0, 0);

line(45, 40, 30, yl, 0, 1);

line(45, 40, 60, yr, 0, 1);

line(60, yr + 3, 60, yr + 9, 0, 1);

line(30, yl + 3, 30, yl + 9, 0, 1);

line(27, yl + 9, 33, yl + 9, 0, 1);

line(57, yr + 9, 63, yr + 9, 0, 1);
```

Рисунок 19 –Отображение изображения «весов»

2.4.3 Отображение значения результата измерений

С помощью ранее упомянутой функции «glcd_putchar()» мы выводим на дисплей измеряемый параметр массы. Т.к используется эта команда, выводится только по 1 символу, поэтому нужно разбить число на четыре символа. Сначала находим остаток от деления найденной величины «temp_code_ADC0» на десять и

добавляем сорок восемь, т.к код цифр от нуля до девяти находится с сорок восьмого места в таблице символов библиотеки для графического ЖКИ.

Затем делим число на десять и повторяем эту операцию еще три раза, чтобы вывести все значение измеряемой величины.

Отображение значения результата измерений представлено на рисунке 20.

```
// Отображение цифр результата измерений ("заставка" выведена заранее)
glcd_putchar((temp_code_ADC0%10+48),base+8*7,1,0,1); //младший десятичный разряд результата измерений
temp_code_ADC0/=10;
glcd_putchar((temp_code_ADC0%10+48),base+8*6,1,0,1);
temp_code_ADC0/=10;
glcd_putchar((temp_code_ADC0%10+48),base+8*4,1,0,1);
temp_code_ADC0/=10;
glcd_putchar((temp_code_ADC0%10+48),base+8*3,1,0,1); //старший десятичный разряд результата измерений

prov = yl;
```

Рисунок 20 – Отображение значения результата измерений

2.5 Инициализация портов

Данный фрагмент кода инициализирует порты ввода-вывода микроконтроллера. Эта инициализация необходима для подключения и использования графического ЖКИ.

Первые две строки устанавливают порт A таким образом, что на выводе PA2 устанавливается логическая "единица", а на выводах PA3 и PA4 - логические "нули". То есть, на выводе PA2 устанавливается высокий уровень напряжения, а на выводах PA3 и PA4 - низкий уровень напряжения.

Следующие две строки устанавливают порт B таким образом, что на выводах с PB0 по PB2 и с PB5 по PB7 устанавливаются логические "нули", а на выводах с PB3 по PB4 - логические "единицы". То есть, на выводах с PB0 по PB2 и с PB5 по PB7 устанавливается низкий уровень напряжения, а на выводах с PB3 по PB4 - высокий уровень напряжения.

Следующие две строки устанавливают порт D таким образом, что на выводе PD7 устанавливается логическая "единица", а на выводах PD6 и PD5 - логические

"нули". То есть, на выводе PD7 устанавливается высокий уровень напряжения, а на выводах PD6 и PD5 - низкий уровень напряжения.

Последняя строка устанавливает порт C как порт вывода, то есть все его выводы будут использоваться для выдачи сигналов. Инициализация портов представлена на рисунке 21.

```
void init()
{
    PORTA=0b00000100;
    DDRA=0b00001100;
    PORTB=0b00000000;
    DDRB=0b11100011;
    PORTD=0b10000000;
    DDRD=0b11000000;
    DDRC=0b11111111;

}
```

Рисунок 21 – Инициализация портов

2.6 Инициализация таймера ноль

Данный фрагмент кода инициализирует таймер ноль микроконтроллера.

Для нормальной работы таймера/счётчика C0 необходимо настроить период срабатывания таймера на восемь миллисекунд. Для этого необходимо провести следующие расчеты:

Для начала найдем частоту срабатывания таймера по формуле (1):

$$F_{TC0} = \frac{8000000}{256 \times 250} = 125 \text{ Гц}, \quad (1)$$

где восемь миллионов герц – частота синхронизации микроконтроллера.

Далее найдем период срабатывания таймера по формуле (2):

$$T = \frac{1}{F_{TC0}} = 0{,}008\ c, \qquad (2)$$

таким образом мы получили период срабатывания таймера/счётчика C0 равный восьми миллисекундам.

Строка «TCNT0=0b00000000» устанавливает начальное значение счетчика таймера ноль в ноль.

Далее строка «OCR0=249» устанавливает значение регистра сравнения таймера ноль в двести сорок девять. Это означает, что когда счетчик таймера достигнет значения двести сорок девять, будет сгенерировано прерывание.

После, строка «TCCR0=0b00001100» устанавливает режим работы таймера ноль. В данном случае это режим, при котором при достижении значения регистра сравнения, счетчик таймера будет обнулен.

Затем строка «TIMSK=0b00000010» устанавливает разрешения на прерывания таймера ноль при совпадении значения счетчика с значением регистра сравнения.

Последняя строка «TIFR=0b00000011» сбрасывает флаги прерываний от таймера ноль.

Инициализация таймера ноль представлена на рисунке 22.

```
void timer_init(void)
{
    //настройка на срабатывание T/C0 с интервалом 8 мс
    TCNT0=0b00000000;   //очистка T/C0
    OCR0=249;           //запись константы 249 в регистр сравнения T/C0
    TCCR0=0b00001100;   //режим "сброс при совпадении" и делитель на 256
    TIMSK=0b00000010;  //разрешение прерывания по совпадению от T/C0
    TIFR=0b00000011;   //очистка флагов прерываний T/C0
}
```

Рисунок 22 – Инициализация таймера ноль

2.7 Инициализация АЦП

Данный фрагмент кода инициализирует аналого-цифровой преобразователь микроконтроллера.

Строка «ADMUX=0b00000000» устанавливает регистр ADMUX в ноль. Этот регистр отвечает за выбор источника напряжения и входа для преобразования.

Строка«ADCSRA=0b10000111» устанавливает регистр ADCSRA. Этот регистр отвечает за настройку параметров АЦП, таких как скорость преобразования, режим работы и т.д. В данном случае, установка битов означает следующее:

- седьмой бит установлен в «единицу», что включает АЦП;

- шестой бит установлен в «ноль», что означает, что преобразование не запущено;

- пятый бит отвечает за выбор режима автоматического повтора запуска преобразования;

- четвертый бит отвечает за флаг срабатывания при срабатывании АЦП;

- третий бит установлен в «ноль», что означает, что прерывания АЦП запрещены;

- второй, первый и нулевой порты установлены в «111» – выбор предделителя равен двенадцать, частота преобразования равна шестьдесят две тысячи пятьсот герц;

Инициализация АЦП представлена на рисунке 23.

```
void init_adc(void)
{
    ADMUX=0b00000000;    //
    ADCSRA=0b10000111;
}
```

Рисунок 23 – Инициализация АЦП

2.8 Сканирование сенсорных кнопок

Данный фрагмент кода отвечает за сканирование координат на клавиатуре.

Для того, чтобы работать с сенсорной панель необходимо настроить порты PA2 и PA3 так, как было описано в пункте 1.3.3.

Сначала устанавливается порт A в значение «0b00000100», что означает, что только вывод PA2 будет установлен в высокий уровень, а остальные выводы будут в низком уровне. Затем происходит задержка в две миллисекунды.

Далее устанавливается регистр ADMUX в значение «0b00000000», что означает, что используется вход ADC0 для измерения напряжения. Затем запускается преобразование АЦП с помощью установки бита ADSC в регистре ADCSRA. Пока происходит преобразование, программа ожидает окончания с помощью цикла «while», который проверяет, равен ли ноль бит ADSC в регистре ADCSRA. Когда преобразование закончится, результат сохраняется в переменную «x_coordinate». Затем порт A устанавливается в значение «0b00001000», что означает, что только вывод PA3 будет установлен в высокий уровень, а остальные выводы будут в низком уровне. Затем происходит задержка в две миллисекунды.

Далее регистр ADMUX устанавливается в значение «0b00000001», что означает, что используется вход ADC1 для измерения напряжения. Запускается преобразование АЦП с помощью установки бита ADSC в регистре ADCSRA.

Пока происходит преобразование, программа ожидает окончания с помощью цикла «while», который проверяет, равен ли ноль бит ADSC в регистре ADCSRA. Когда преобразование закончится, результат сохраняется в переменную «y_coordinate».

Таким образом, после выполнения этого фрагмента кода переменные «x_coordinate» и «y_coordinate» будут содержать координаты, которые были нажаты на клавиатуре.

Первая часть сканирования сенсорных кнопок указана на рисунке 24.

```
void scan_key(void){
    PORTA=0b00000100;       //
    _delay_ms(2);           // Задержка !!! (паразитная емкость должна зарядиться)
    ADMUX=0b00000000;       // Выбираем нулевой канал АЦП
    ADCSRA|=(1<<ADSC);      // запускаем АЦП ( аналогично ADCSRA=((ADCSRA|0b01000000));
    while ((ADCSRA&(1<<ADSC))!=0); //ожидаем конца преобразования АЦП

    x_coordinate=ADC;

    PORTA=0b00001000;
    _delay_ms(2);
    ADMUX=0b00000001;
    ADCSRA|=(1<<ADSC);      // запускаем АЦП ( аналогично ADCSRA=((ADCSRA|0b01000000));
    while ((ADCSRA&(1<<ADSC))!=0); //ожидаем конца преобразования АЦП

    y_coordinate=ADC;
```

Рисунок 24 - Сканирование сенсорных кнопок (часть 1)

Этот фрагмент кода содержит несколько условных операторов if, которые проверяют значения переменных y_coordinate и x_coordinate, а также gkey_status. При выполнении условий по обозначенным координатам нажатия устанавливаются соответствующие значения переменных gkey_status и gkey в зависимости от точки нажатия на сенсорную панель.

Вторая часть сканирования сенсорных кнопок указана на рисунке 25.

```
    if (((y_coordinate>480)&&(y_coordinate<620))&&((x_coordinate>550)&&(x_coordinate<770))&&(gkey_status==0)){
        gkey_status=0b11000000;
        gkey=0;
    }
    if (((y_coordinate>320)&&(y_coordinate<460))&&((x_coordinate>550)&&(x_coordinate<770))&&(gkey_status==0)){
        gkey_status=0b11000000;
        gkey=1;
    }
    if (((y_coordinate>160)&&(y_coordinate<290))&&((x_coordinate>550)&&(x_coordinate<770))&&(gkey_status==0)){
        gkey_status=0b11000000;
        gkey=2;
    }
    if (((y_coordinate<20)&&(x_coordinate<20))&&(gkey_status&0b01000000)!=0){
        gkey_status=gkey_status&0b10000000;
    }

}
```

Рисунок 25 - Сканирование сенсорных кнопок (часть 2)

2.9 Чтение значения с канала АЦП

Этот фрагмент кода представляет собой функцию, которая используется для чтения значения с канала АЦП (аналого-цифрового преобразователя). Чтение значения с канала АЦП представлено на рисунке 26.

Сначала устанавливается значение регистра ADMUX в «0b00000101», что означает, что мы будем считывать значение с пятого канала АЦП.

Затем устанавливается бит ADSC в регистре ADCSRA, чтобы запустить АЦП. Это означает, что АЦП начнет преобразование аналогового сигнала в цифровое значение. Далее выполняется цикл «while», который ожидает, пока преобразование не будет завершено.

Это происходит путем проверки значения бита ADSC в регистре ADCSRA. Пока этот бит не станет равным нулю, цикл будет продолжаться. Когда преобразование завершено, значение АЦП сохраняется в переменной t_ADC. Таким

образом, этот фрагмент кода позволяет получить значение с определенного канала АЦП и сохранить его для дальнейшей обработки.

```
void get_mass (void){ //функция чтения канала АЦП
    // координата Y
    ADMUX=0b00000101;
    ADCSRA|=(1<<ADSC);      // запускаем АЦП ( аналогично ADCSRA=((ADCSRA|0b01000000));
    while ((ADCSRA&(1<<ADSC))!=0); //ожидаем конца преобразования АЦП
    t_ADC=ADC;
}
```

Рисунок 26 – Чтение значения с канала АЦП

3 ОПИСАНИЕ ПРОГРАММ ДЛЯ ОТЛАДКИ ПРОГРАММНОГО КОДА ИЗМЕРИТЕЛЬНОГО УСТРОЙСТВА

3.1 Microchip Studio

Microchip Studio — это интегрированная среда разработки (IDE) для программирования микроконтроллеров AVR от компании Microchip. Эта программа предоставляет разработчикам все необходимые инструменты для создания, отладки и тестирования приложений на микроконтроллерах AVR.

Microchip Studio включает в себя мощный компилятор C/C++, ассемблер, отладчик и симулятор, а также библиотеки и драйверы для работы с периферийными устройствами, такими как АЦП, ШИМ, UART и другие.

Кроме того, Microchip Studio предоставляет разработчикам возможность использовать интегрированные средства отладки и анализа производительности, чтобы ускорить процесс разработки и улучшить качество кода.

Интерфейс программы Microchip Studio представлен на рисунке 23.

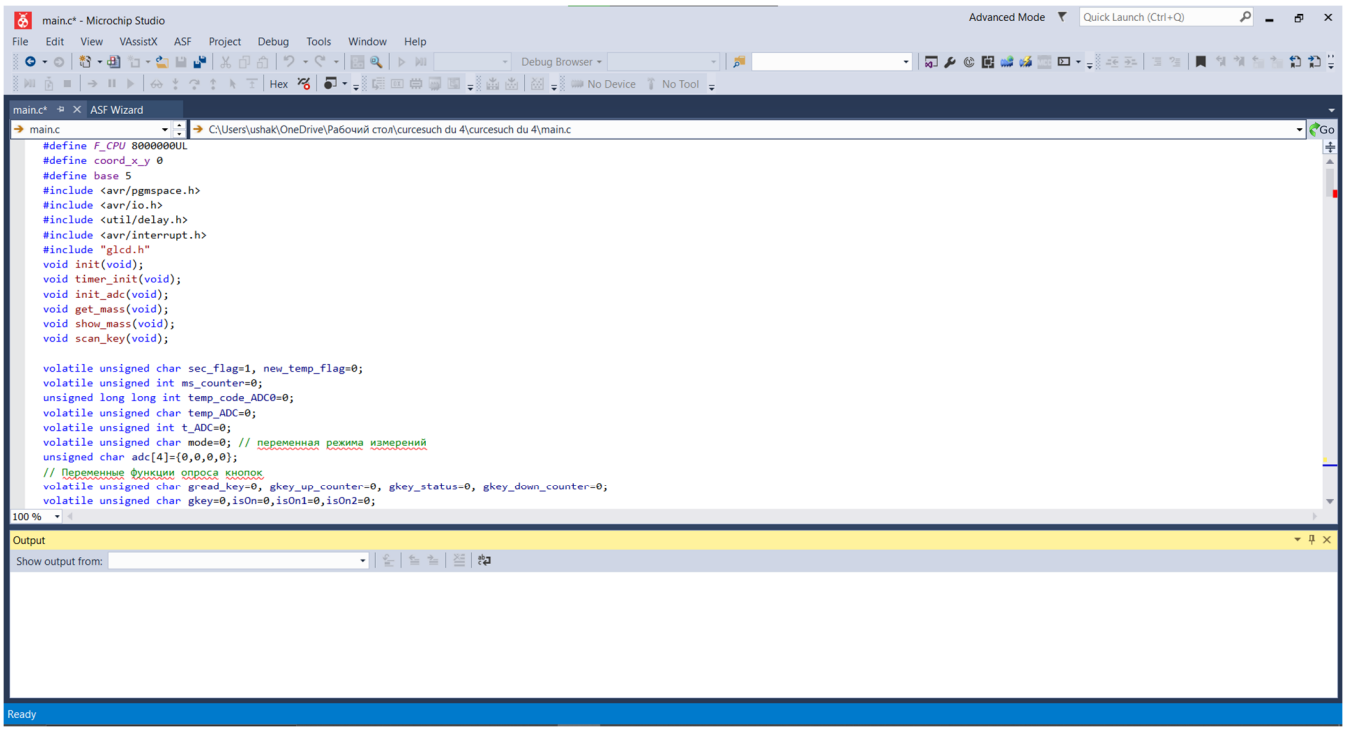

Рисунок 27 – Интерфейс программы Microchip Studio

3.2 AVRFLASH

AVRFLASH – это программа, которая позволяет загружать в микроконтроллер шестнадцатеричный код, который был сгенерирован в компиляторе. Для этого сначала код загружается в буфер программатора, а затем уже передается в микроконтроллер.

Для настройки AVRFLASH нужно:

а)Выбрать микроконтроллер ATmega32;

б)Установить частоту работы восемь МГц;

в)Настроить FUSE биты как показано на рисунке.

г) Загрузить hex-файл в программу.

Интерфейс программы AVRFLASH представлен на рисунке 24.

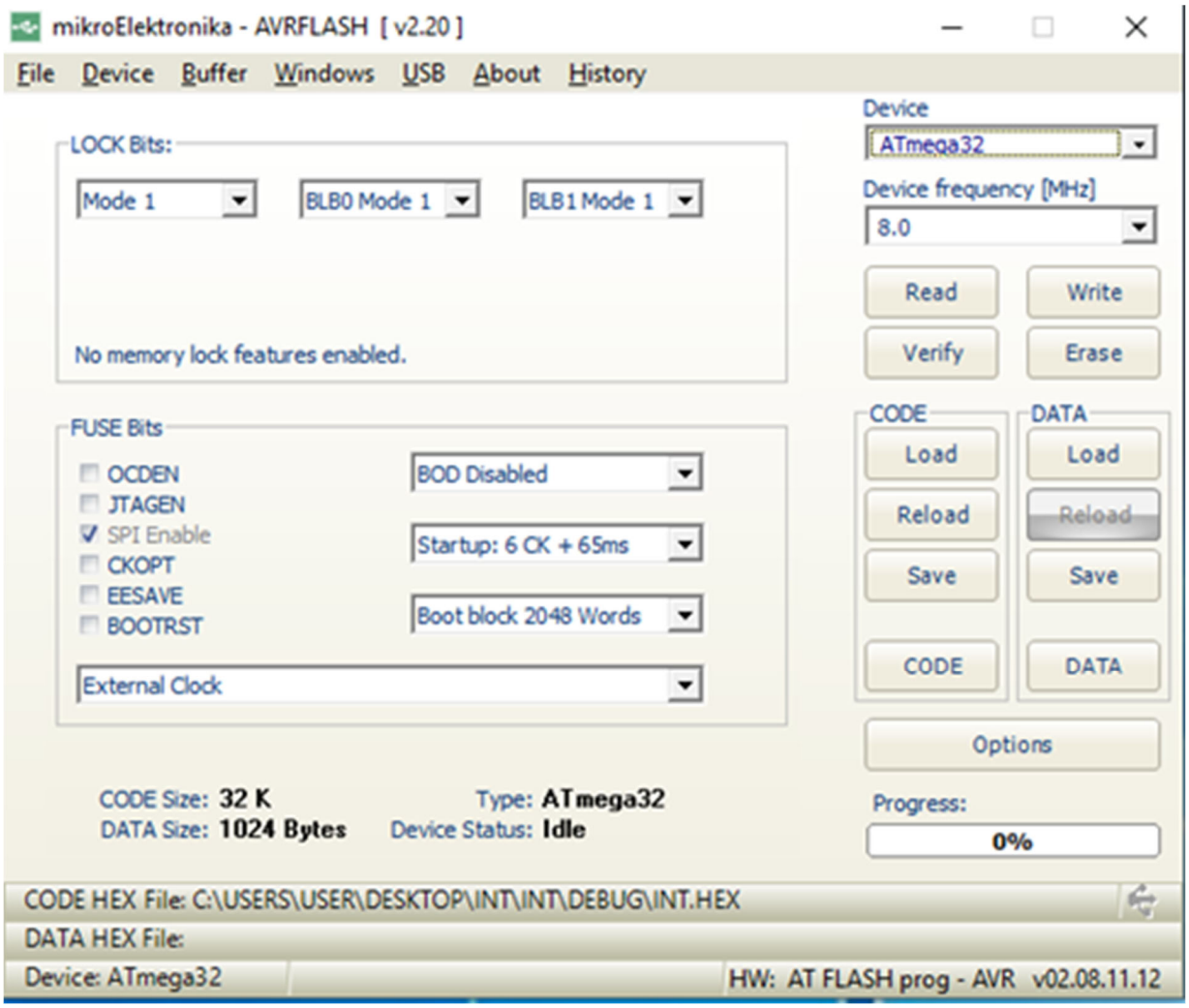

Рисунок 28 – Интерфейс AVRFLASH