You can see the full project on our GitHub:
https://github.com/youefkh05/ArduScope

Main code:

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <avr/pgmspace.h>
#include "Adafruit_SH1106.h"
#include "bitmaps.h"
#include "application.h"
#include "Multi_Metre_Sig.h"
#include "Osci.h"
#include "eerom_map.h"
//#include <stdint.h>
//#include <stdlib.h>

extern Adafruit_SH1106 oled;        // oled handeler

void setup()
{
  MM_Init();
  Serial.begin(9600);

  oled.begin(SH1106_SWITCHCAPVCC, OLED_I2C_ADDRESS);  // use this when SH1106

  oled.clearDisplay();
  oled.display();  // Initial display update
  // Set the text color to white
  oled.setTextColor(WHITE);

  // define pins for buttons
  // INPUT_PULLUP means the button is HIGH when not pressed, and LOW when pressed
  // since it´s connected between some pin and GND
  pinMode(BUTTON_UP_PIN, INPUT_PULLUP); // up button
  pinMode(BUTTON_SELECT_PIN, INPUT_PULLUP); // select button
  pinMode(BUTTON_DOWN_PIN, INPUT_PULLUP); // down button


  Osci_Init();
}

void loop() {
  // Buffer to hold the bitmap read from EEPROM
  //uint8_t bitmapBuffer[BIT_MAP_SIZE];
  flag_type flags = { };

  //Global Variables that are local
  uint8_t NUM_ITEMS = 6;

  menus selected_menu = MainMenu;

  uint8_t item_selected = 0; // which item in the menu is selected

  uint8_t item_sel_previous; // previous item - used in the menu screen to draw
the item before the selected one
  uint8_t item_sel_next; // next item - used in the menu screen to draw next item
after the selected one

  uint8_t device_selected = Voltmeter;
  uint8_t range_selected = range1;
  uint8_t mode_selected = DC_MODE;

  float device_reading = 0;

  while(1)
  {


  // Updates number of items according to selected menu
  switch (selected_menu)
  {
      case MainMenu:
      NUM_ITEMS = 6;
      break;

      case SigGenMenu:
      NUM_ITEMS = 4;

      case ConfigMenu:
      NUM_ITEMS = 2;

      default:
      NUM_ITEMS = 1;
  }
  /******************************************************************************
*****************/
          /*Multi-Meter Code*/
  Select_Mux(device_selected, range_selected);  //Choose device

  // Update Reading
  switch (device_selected)
  {
    case Voltmeter:
      device_reading = Read_Volt(range_selected, mode_selected);
      break;

    case Ammeter:
      device_reading = Read_Amp(range_selected, mode_selected);
      break;

    case Ohmeter:
      device_reading = Read_Ohm(range_selected);
      break;
  }

  char reading_arr[MAX_ITEM_LENGTH];
  itoa(device_reading, reading_arr,10); //Change reading to str to be printed on
screen

  /*******************************************************************************
*********/
    //Selection Buttons

    // Up
Button  **************************************************************************
*******
    if((digitalRead(BUTTON_UP_PIN) == LOW))
    {
      delay(30); // Rebounce Delay
      if((digitalRead(BUTTON_UP_PIN) == LOW)  && (flags.button_up_f == 0))
      {
        item_selected--;
        if(item_selected == 255)
        {
          item_selected = NUM_ITEMS - 1;
        }
        flags.button_up_f = 1;
      }
    }
    else
    {
      flags.button_up_f = 0;
    }

    // Down
Button  *************************************************************************
*******
    if((digitalRead(BUTTON_DOWN_PIN) == LOW))
    {
      delay(30); // Rebounce Delay
      if((digitalRead(BUTTON_DOWN_PIN) == LOW)  && (flags.button_down_f == 0))
      {
        item_selected++;
        if(item_selected >= NUM_ITEMS)
        {
          item_selected = 0;
        }


        flags.button_down_f = 1;
      }
    }
    else
    {
      flags.button_down_f = 0;
    }

    // Selection Button
********************************************************************************
    if((digitalRead(BUTTON_SELECT_PIN) == LOW))
    {
      delay(30); // Rebounce Delay
      if((digitalRead(BUTTON_SELECT_PIN) == LOW)  && (flags.button_select_f ==
0))
      {
        switch (selected_menu)
        {
          case MainMenu:
            switch (item_selected)
            {
              case 0:
                selected_menu = VoltmeterMenu;
                device_selected = Voltmeter;
                break;

              case 1:
                selected_menu = AmmeterMenu;
                device_selected = Ammeter;
                break;

              case 2:
                selected_menu = OhmmeterMenu;
                device_selected = Ohmeter;
                break;

              case 3:
                selected_menu = SigGenMenu;
                item_selected = 0;
                NUM_ITEMS = 4;
                break;

              case 4:
                selected_menu = Scope;
                item_selected = 0;
                NUM_ITEMS = 3;
                break;

              case 5:
                selected_menu = ConfigMenu;
                item_selected = 0;
                NUM_ITEMS = 3;
                break;
            }
          break; // End of Case MainMenu
********************************************************************************

          case VoltmeterMenu:
            selected_menu = MainMenu;
            item_selected = 0;
            break; // End of Case Voltmeter
********************************************************************************

          case AmmeterMenu:
            selected_menu = MainMenu;
            item_selected = 0;
            break;  // End of Case Ammeter
********************************************************************************

          case OhmmeterMenu:
            selected_menu = MainMenu;
            item_selected = 0;
            break;  // End of Case Ohmmeter
********************************************************************************

          case SigGenMenu:
            selected_menu = MainMenu;
            item_selected = 0;
            break;  // End of Case SigGen
********************************************************************************

          case Scope:
            selected_menu = MainMenu;
            item_selected = 0;
            break;  // End of Case SigGen
********************************************************************************

          case ConfigMenu:
            break;
        }


        flags.button_select_f = 1;
      }
    }
    else
    {
      flags.button_select_f = 0;
    }


  /******************************************************************************
**/
    //Update Selected Item
    item_sel_previous = item_selected - 1;
    if(item_sel_previous == 255)
    {
      item_sel_previous = NUM_ITEMS - 1;
    }

    item_sel_next = item_selected + 1;
    if(item_sel_next >= NUM_ITEMS)
    {
      item_sel_next = 0;
    }
  /******************************************************************************
**/
        /*Signal Generator Selection*/

    if(selected_menu == SigGenMenu)
    {
      switch (item_selected)
            {
              case 0:
              break;

              case 1:
              break;

              case 2:
              break;

              case 3:
              break;
            }
    }

  /******************************************************************************
********************/
                  /*OLED Main Menu*/
      oled.clearDisplay(); // Clear the display before drawing each frame
      switch (selected_menu)
      {
        case MainMenu:
          // Selection box and scroll bar
          oled.drawBitmap(0, 0, epd_bitmap_bg, SCREEN_WIDTH, SCREEN_HEIGHT,
TEXT_COLOR);

          // Draw Menu Items
          drawMenuItem(15,  item_sel_previous, MENU_ITEMS_TYPE ); // Menu Item 1
          drawMenuItem(37,  item_selected, MENU_ITEMS_TYPE );         // Menu
Item 2
          drawMenuItem(59,  item_sel_next, MENU_ITEMS_TYPE );         // Menu
Item 3

        break;

        case VoltmeterMenu:
        /*
        u8g.setFont(u8g_font_7x14B);
        u8g.drawStr(26,37, "Ammeter Reading");
        u8g.setFont(u8g_font_7x14);
        u8g.drawStr(26,59, reading_arr);
        */

        //Voltmeter Menu code
        oled.setTextSize(TEXT_SIZE);       // Keep default text size
        oled.setTextColor(WHITE);
        oled.setCursor(26, 37);            // Set position
        oled.print("Voltmeter Reading");  // Display text
        oled.setCursor(26, 50);            // Set position
        oled.print(reading_arr);           // Display text

        break;

        case AmmeterMenu:

        //Ammeter Menu code
        oled.setTextSize(TEXT_SIZE);       // Keep default text size
        oled.setTextColor(WHITE);
        oled.setCursor(26, 37);            // Set position
        oled.print("Ammeter Reading");    // Display text
        oled.setCursor(26, 50);            // Set position
        oled.print(reading_arr);           // Display text

        break;

        case OhmmeterMenu:
        //Ohmmeter Menu Code
        oled.setTextSize(TEXT_SIZE);       // Keep default text size
        oled.setTextColor(WHITE);
        oled.setCursor(26, 37);            // Set position
        oled.print("Ohmmeter Reading");   // Display text
        oled.setCursor(26, 50);            // Set position
        oled.print(reading_arr);           // Display text

        break;

        case SigGenMenu:
        // Selection box and scroll bar
        oled.drawBitmap(0, 0, epd_bitmap_bg, SCREEN_WIDTH, SCREEN_HEIGHT,
TEXT_COLOR);
        //u8g.drawBitmapP(0, 0, 128/8, 64, epd_bitmap_bg);
```

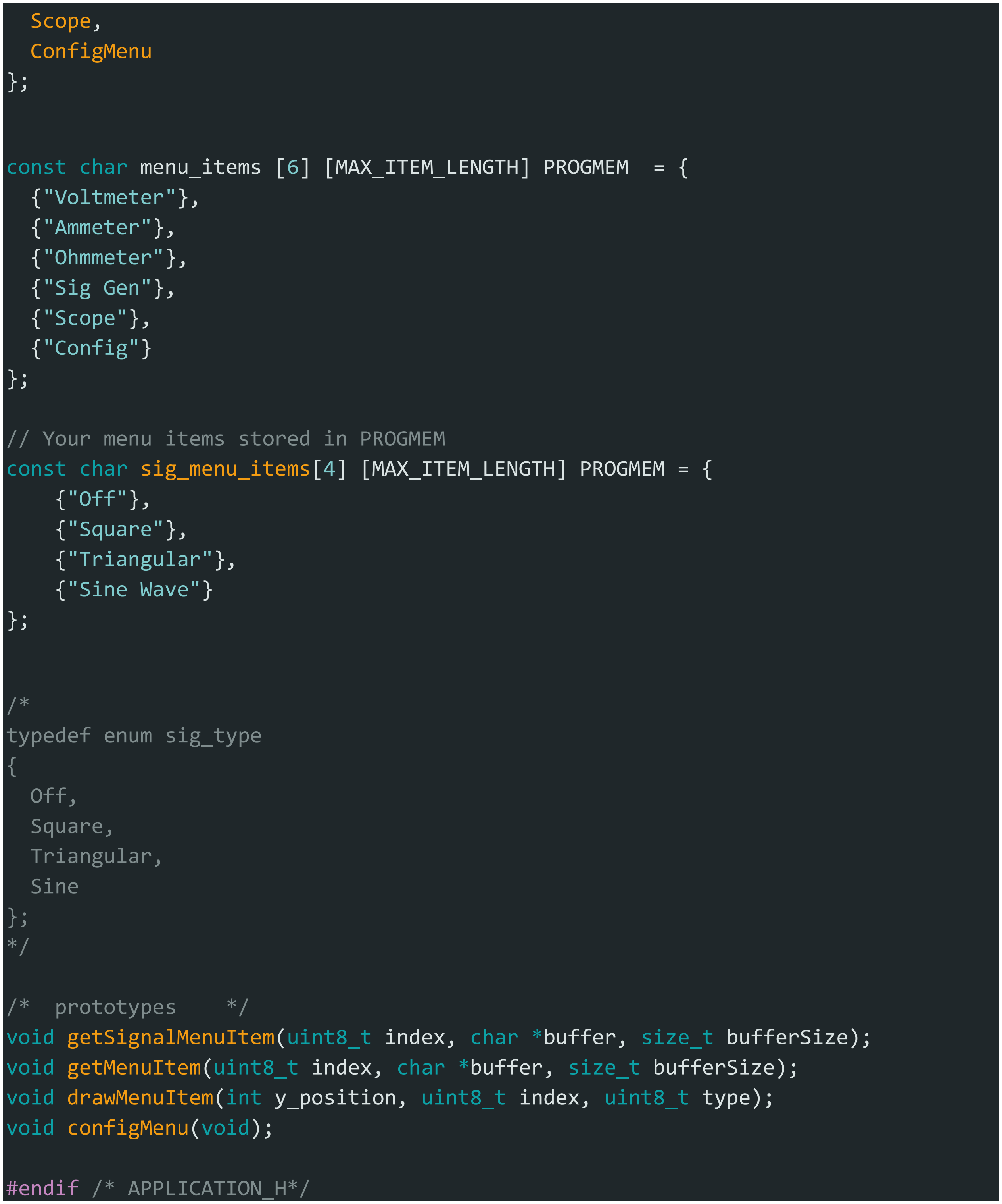
```
          drawMenuItem(15,  item_sel_previous, SIG_MENU_ITEMS_TYPE );     // Menu Item 1
          drawMenuItem(37,  item_selected, SIG_MENU_ITEMS_TYPE );         // Menu Item 2
          drawMenuItem(59,  item_sel_next, SIG_MENU_ITEMS_TYPE );         // Menu Item 3

          /*
          //Signal Generator Menu Code
          //Menu Item 1
          oled.setTextSize(TEXT_SIZE);                 // Keep default text size
          oled.setTextColor(WHITE);
          oled.setCursor(26, 15);                      // Set position
          oled.print(sig_menu_items[item_sel_previous]);   // Display text
          //u8g.setFont(u8g_font_7x14);
          //u8g.drawStr(26,15, sig_menu_items[item_sel_previous]);

          //Menu Item 2
          oled.setTextSize(TEXT_SIZE);                 // Keep default text size
          oled.setTextColor(WHITE);
          oled.setCursor(26, 37);                      // Set position
          oled.print(sig_menu_items[item_selected]);   // Display text

          //Menu Item 3
          oled.setTextSize(TEXT_SIZE);                 // Keep default text size
          oled.setTextColor(WHITE);
          oled.setCursor(26,59);                       // Set position
          oled.print(sig_menu_items[item_sel_next]);   // Display text
          */

        break;

        case Scope:

        Osci_Run();

        NUM_ITEMS = 6;

        selected_menu = MainMenu;

        item_selected = 0; // which item in the menu is selected

        item_sel_previous=-1; // previous item - used in the menu screen to draw the item before the selected one
        item_sel_next=1; // next item - used in the menu screen to draw next item
after the selected one
        //Signal Generator Menu Code

        break;
      }

    oled.display(); // Update the display with the new content
    delay(50); // Optional: Delay for stability or to control the refresh rate


    } /* while(1)  */

}
```

Bitmaps.h:

```
/*
  Contains Bitmap codes for arduino Oled
*/
#ifndef BITMAPS_H
#define BITMAPS_H

/*******************************************************************************
*****************/
/*
// 'oled_display_menus-Recovered_0004_V', 16x16px
const unsigned char volt_bitmap [] PROGMEM = {
  0xfc, 0x3f, 0x60, 0x0c, 0x60, 0x08, 0x30, 0x10, 0x30, 0x10, 0x18, 0x10, 0x18,
0x20, 0x0c, 0x20,
  0x0c, 0x40, 0x0e, 0x40, 0x06, 0x80, 0x06, 0x80, 0x03, 0x80, 0x03, 0x00, 0x01,
0x00, 0x01, 0x00
};
// 'oled_display_menus-Recovered_0006_A', 16x16px
const unsigned char amm_bitmap [] PROGMEM = {
  0x00, 0x00, 0x03, 0xc0, 0x07, 0xe0, 0x0c, 0x30, 0x0c, 0x30, 0x0c, 0x30, 0x0c,
0x30, 0x0c, 0x30,
  0x0f, 0xf0, 0x0f, 0xf0, 0x0c, 0x30, 0x0c, 0x30, 0x0c, 0x30, 0x0c, 0x30, 0x0c,
0x30, 0x00, 0x00
};
// 'oled_display_menus-Recovered_0005_Ohm', 16x16px
const unsigned char ohm_bitmap [] PROGMEM = {
  0x00, 0x00, 0x07, 0xe0, 0x1f, 0xf8, 0x3c, 0x3c, 0x78, 0x1e, 0x60, 0x06, 0xc0,
0x03, 0xc0, 0x03,
  0xc0, 0x03, 0xc0, 0x03, 0x70, 0x0e, 0x70, 0x0e, 0x38, 0x1c, 0xf8, 0x1f, 0xf8,
0x1f, 0x00, 0x00
};

// 'oled_display_menus-Recovered_0003_sig_gen', 16x16px
const unsigned char sig_gen_bitmap [] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0x87, 0x20, 0x84, 0x20, 0x84, 0x20,
0x84, 0x20, 0x84,
  0x20, 0x84, 0x20, 0x84, 0x20, 0x84, 0x20, 0x84, 0x20, 0x84, 0xe0, 0xfc, 0x00,
0x00, 0x00, 0x00
};

// 'oled_display_menus-Recovered_0006_config', 16x16px
const unsigned char config_bitmap [] PROGMEM = {
  0x01, 0x80, 0x01, 0x80, 0x1a, 0x58, 0x36, 0x6c, 0x20, 0x04, 0x11, 0x88, 0x33,
0xcc, 0xc6, 0x63,
  0xc6, 0x63, 0x33, 0xcc, 0x11, 0x88, 0x20, 0x04, 0x36, 0x6c, 0x1a, 0x58, 0x01,
0x80, 0x01, 0x80
};

// Array of all bitmaps for convenience. (Total bytes used to store images in
PROGMEM = 192)
const unsigned char* bitmap_arr[5] = {
  volt_bitmap,
  amm_bitmap,
  ohm_bitmap,
  sig_gen_bitmap,
  config_bitmap
};
*/

/********************************************************************************
**********************/

const unsigned char epd_bitmap_bg [] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x3f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc2,
  0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x22,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12,
  0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x22,
  0x3f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc2,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x02,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00
};

#endif /* BITMAPS_H */
```

Appilcation.h:

```
#ifndef APPLICATION_H
#define APPLICATION_H

/*  Inculdes  */
#include <avr/pgmspace.h>
#include "Osci.h"
#include "bitmaps.h"
#include "eerom_map.h"

//#include <stdint.h>

// Definitions
typedef unsigned char   uint8_t;
#define BUTTON_UP_PIN           (6)
#define BUTTON_SELECT_PIN       (3)
#define BUTTON_DOWN_PIN         (5)

#define MAX_ITEM_LENGTH     (10)

// Define constants for text size and color
#define TEXT_SIZE 1
#define TEXT_COLOR WHITE

/*  Definitions */
#define MENU_ITEMS_TYPE       (1)
#define SIG_MENU_ITEMS_TYPE   (2)

//  Flags type
typedef struct flag_type
{
  uint8_t button_up_f: 1;
  uint8_t button_down_f: 1;
  uint8_t button_select_f: 1;
};

// Custom Data Types
typedef enum menus
{
  MainMenu,
  VoltmeterMenu,
  AmmeterMenu,
  OhmmeterMenu,
  SigGenMenu,
```

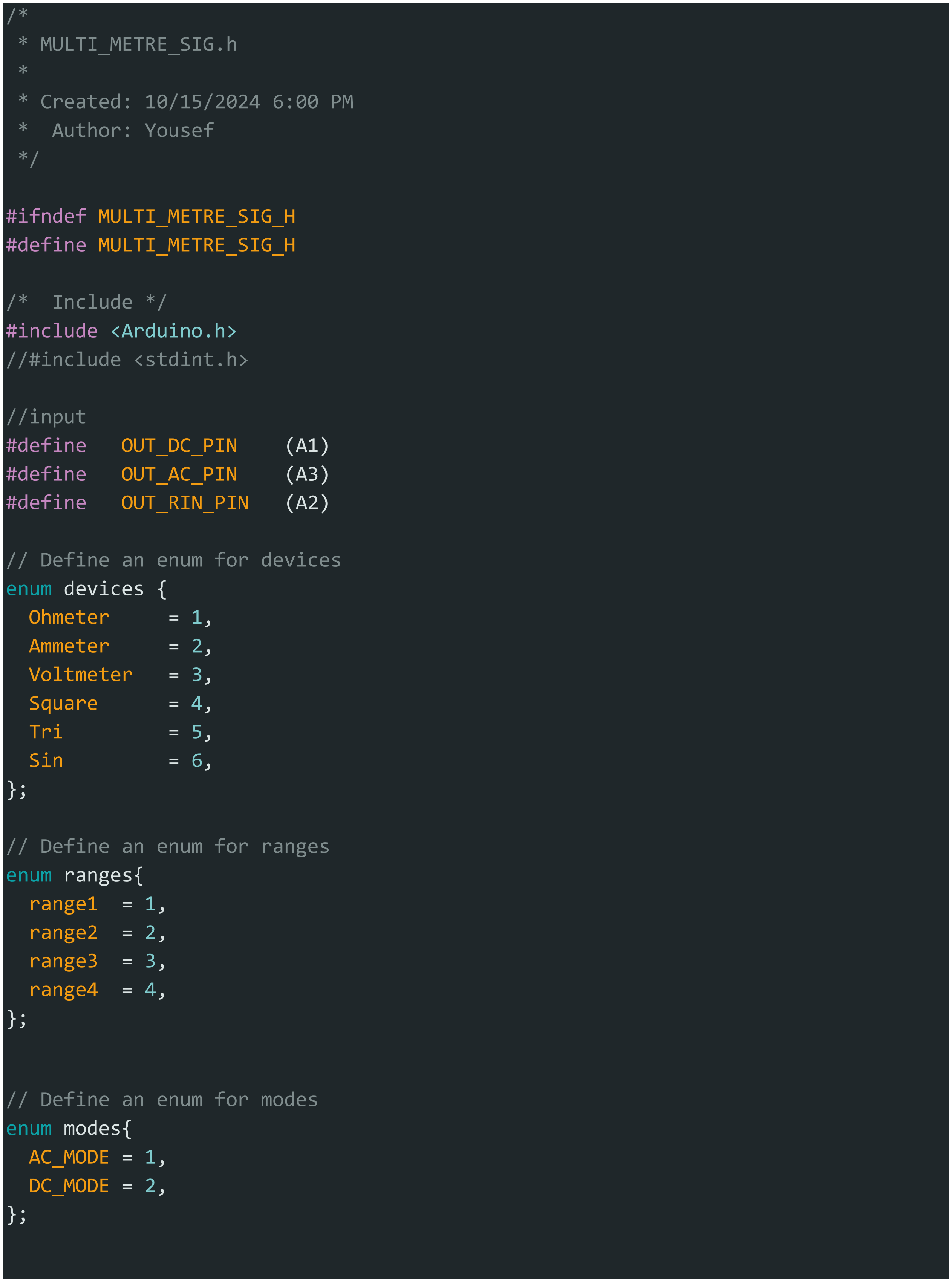


```
  Scope,
  ConfigMenu
};


const char menu_items [6] [MAX_ITEM_LENGTH] PROGMEM  = {
  {"Voltmeter"},
  {"Ammeter"},
  {"Ohmmeter"},
  {"Sig Gen"},
  {"Scope"},
  {"Config"}
};

// Your menu items stored in PROGMEM
const char sig_menu_items[4] [MAX_ITEM_LENGTH] PROGMEM = {
    {"Off"},
    {"Square"},
    {"Triangular"},
    {"Sine Wave"}
};


/*
typedef enum sig_type
{
  Off,
  Square,
  Triangular,
  Sine
};
*/

/*  prototypes    */
void getSignalMenuItem(uint8_t index, char *buffer, size_t bufferSize);
void getMenuItem(uint8_t index, char *buffer, size_t bufferSize);
void drawMenuItem(int y_position, uint8_t index, uint8_t type);
void configMenu(void);

#endif /* APPLICATION_H*/
```

Application.cpp:

```
#include "application.h"

extern Adafruit_SH1106 oled;        // oled handeler

// Function to read a string from PROGMEM
void getSignalMenuItem(uint8_t index, char *buffer, size_t bufferSize) {
    if (index < 4) { // Adjust the range according to your items count
        strncpy_P(buffer, (PGM_P)&sig_menu_items[index], bufferSize);
        buffer[bufferSize - 1] = '\0'; // Ensure null-termination
    }
}

// Function to read a string from PROGMEM
void getMenuItem(uint8_t index, char *buffer, size_t bufferSize) {
    // Make sure the index is within bounds
    if (index < 6) {
        // Read the string from PROGMEM into the buffer
        strncpy_P(buffer, (PGM_P)&menu_items[index], bufferSize);
        buffer[bufferSize - 1] = '\0'; // Ensure null-termination
    }
}

// Function to draw a menu item
void drawMenuItem(int y_position, uint8_t index, uint8_t type) {
  char itemText[MAX_ITEM_LENGTH];
  switch(type){
    case  MENU_ITEMS_TYPE :
      // Buffer to hold the bitmap read from EEPROM
      uint8_t bitmapBuffer[BIT_MAP_SIZE];

      getMenuItem(index, itemText, sizeof(itemText)); // Read from PROGMEM
      //delay(1);
      oled.setTextSize(TEXT_SIZE);
      oled.setTextColor(TEXT_COLOR);
      oled.setCursor(26, y_position-8);
      oled.print(itemText);
      readBitmapFromEEPROM(bitmapBuffer, index, BIT_MAP_SIZE);
      //oled.drawBitmap(3, y_position - 13, item_bitmap, 16, 16, TEXT_COLOR); // Adjusted y position for bitmap
      oled.drawBitmap(3, y_position - 13,  bitmapBuffer, 16, 16, TEXT_COLOR); // Adjusted y position for bitmap
      delay(1);
    break;
    case SIG_MENU_ITEMS_TYPE:
      getSignalMenuItem(index, itemText, sizeof(itemText)); // Read from PROGMEM

      oled.setTextSize(TEXT_SIZE);
      oled.setTextColor(TEXT_COLOR);
      oled.setCursor(26, y_position-8);
      oled.print(itemText);

    break;
  }


}

void configMenu(void){

}
```

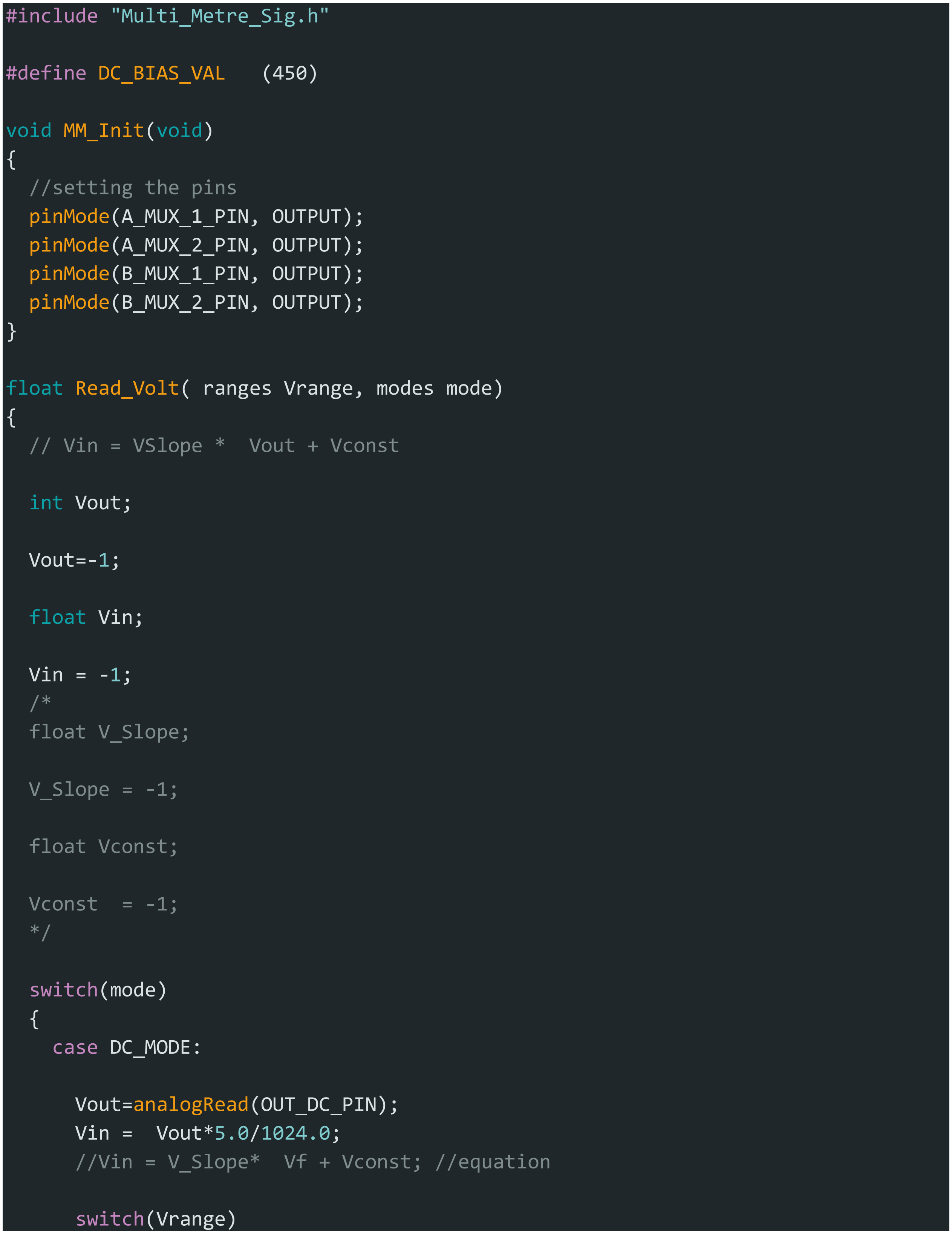
Multi_Metre_Sig.h:



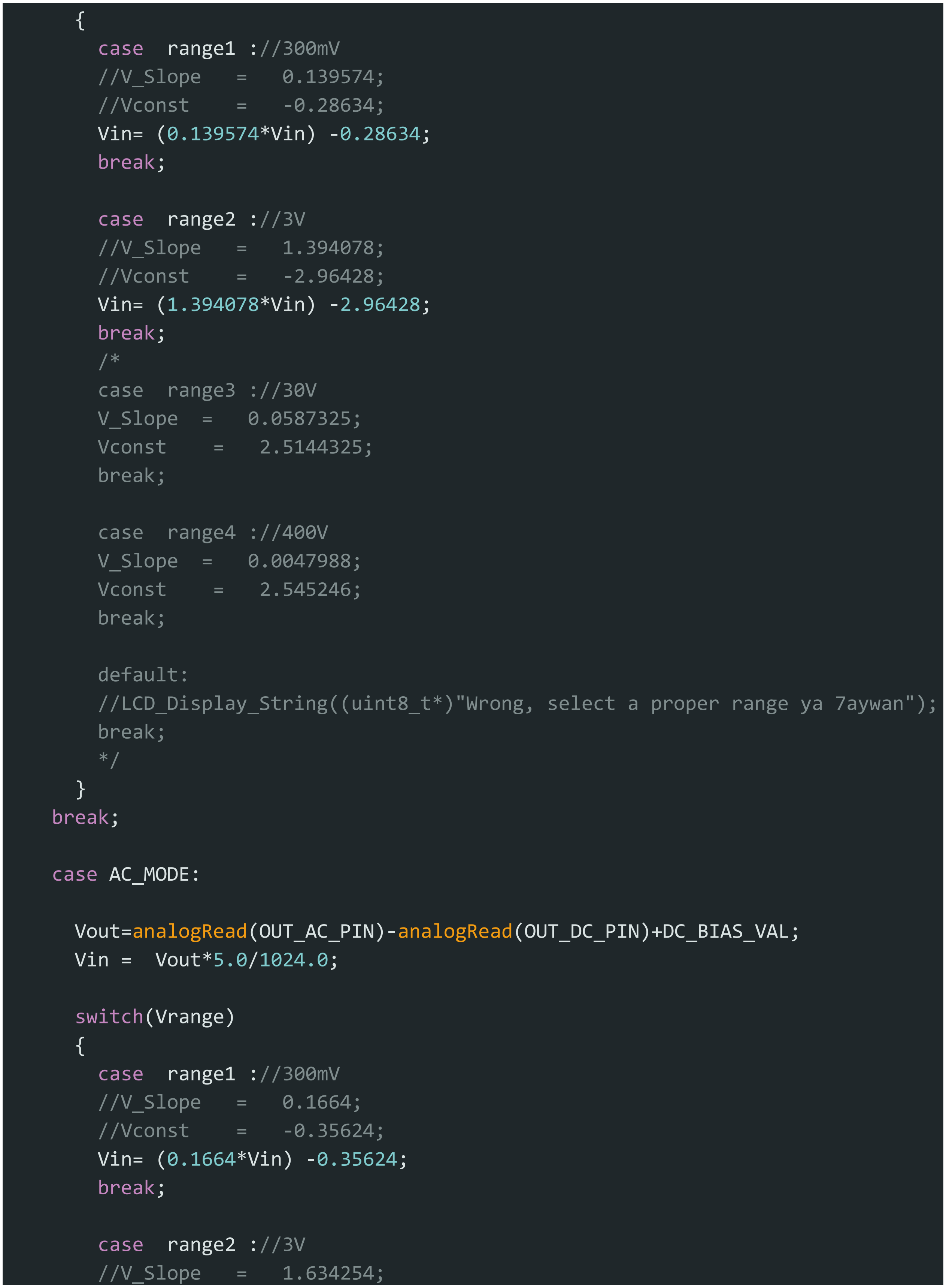
```
/*
 * MULTI_METRE_SIG.h
 *
 * Created: 10/15/2024 6:00 PM
 *  Author: Yousef
 */

#ifndef MULTI_METRE_SIG_H
#define MULTI_METRE_SIG_H

/*  Include */
#include <Arduino.h>
//#include <stdint.h>

//input
#define   OUT_DC_PIN    (A1)
#define   OUT_AC_PIN    (A3)
#define   OUT_RIN_PIN   (A2)

// Define an enum for devices
enum devices {
  Ohmeter     = 1,
  Ammeter     = 2,
  Voltmeter   = 3,
  Square      = 4,
  Tri         = 5,
  Sin         = 6,
};

// Define an enum for ranges
enum ranges{
  range1  = 1,
  range2  = 2,
  range3  = 3,
  range4  = 4,
};


// Define an enum for modes
enum modes{
  AC_MODE = 1,
  DC_MODE = 2,
};
```

```
//MUX1
#define   A_MUX_1_PIN       (12)
#define   B_MUX_1_PIN       (7)

//MUX2
#define   A_MUX_2_PIN       (13)
#define   B_MUX_2_PIN       (4)

// Function Prototypes
void MM_Init(void);

//mode:AC,DC  Range:300mV, 3v, 30v, 400v
float Read_Volt( ranges Vrange, modes mode);

//mode:AC,DC  Range:2mAmp, 20mAmp, 200mAmp, 1Amp
float Read_Amp( ranges Irange, modes mode);

//Range:10k, 100k, 1M
float Read_Ohm( ranges range);

//Devices: Ohmeter, Ammeter, Voltmeter, Square, Tri, Sin    Ranges: 1, 2, 3, 4
void Select_Mux( devices device, ranges range);

//uint8_t Ask_To_Return( uint8_t return_key);





#endif /* MULTI_METRE__SIGH*/
```

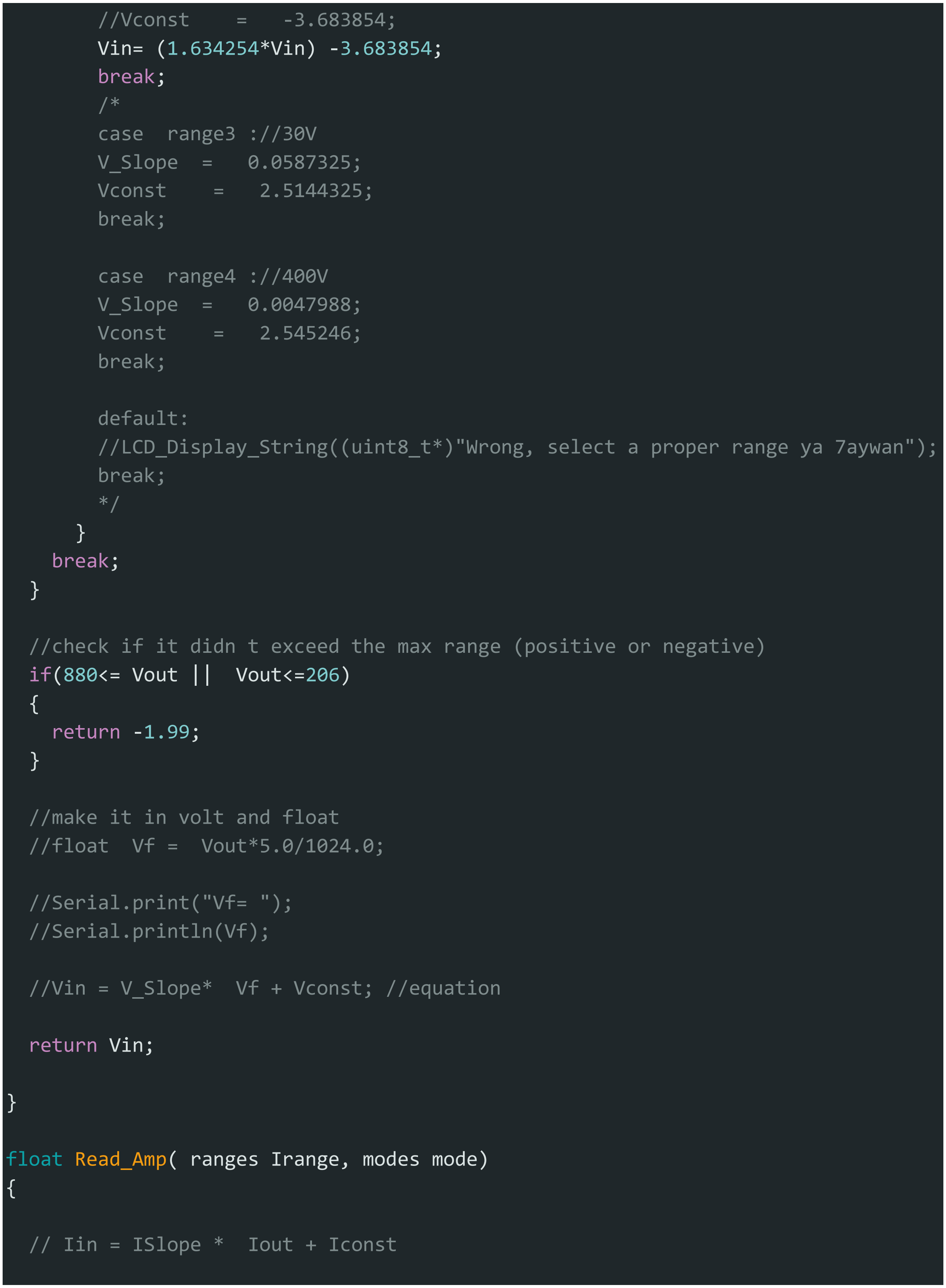
Multi_Metre_Sig.cpp:



```
#include "Multi_Metre_Sig.h"

#define DC_BIAS_VAL   (450)

void MM_Init(void)
{
  //setting the pins
  pinMode(A_MUX_1_PIN, OUTPUT);
  pinMode(A_MUX_2_PIN, OUTPUT);
  pinMode(B_MUX_1_PIN, OUTPUT);
  pinMode(B_MUX_2_PIN, OUTPUT);
}

float Read_Volt( ranges Vrange, modes mode)
{
  // Vin = VSlope *  Vout + Vconst

  int Vout;

  Vout=-1;

  float Vin;

  Vin = -1;
  /*
  float V_Slope;

  V_Slope = -1;

  float Vconst;

  Vconst  = -1;
  */

  switch(mode)
  {
    case DC_MODE:

      Vout=analogRead(OUT_DC_PIN);
      Vin =  Vout*5.0/1024.0;
      //Vin = V_Slope*  Vf + Vconst; //equation

      switch(Vrange)
```

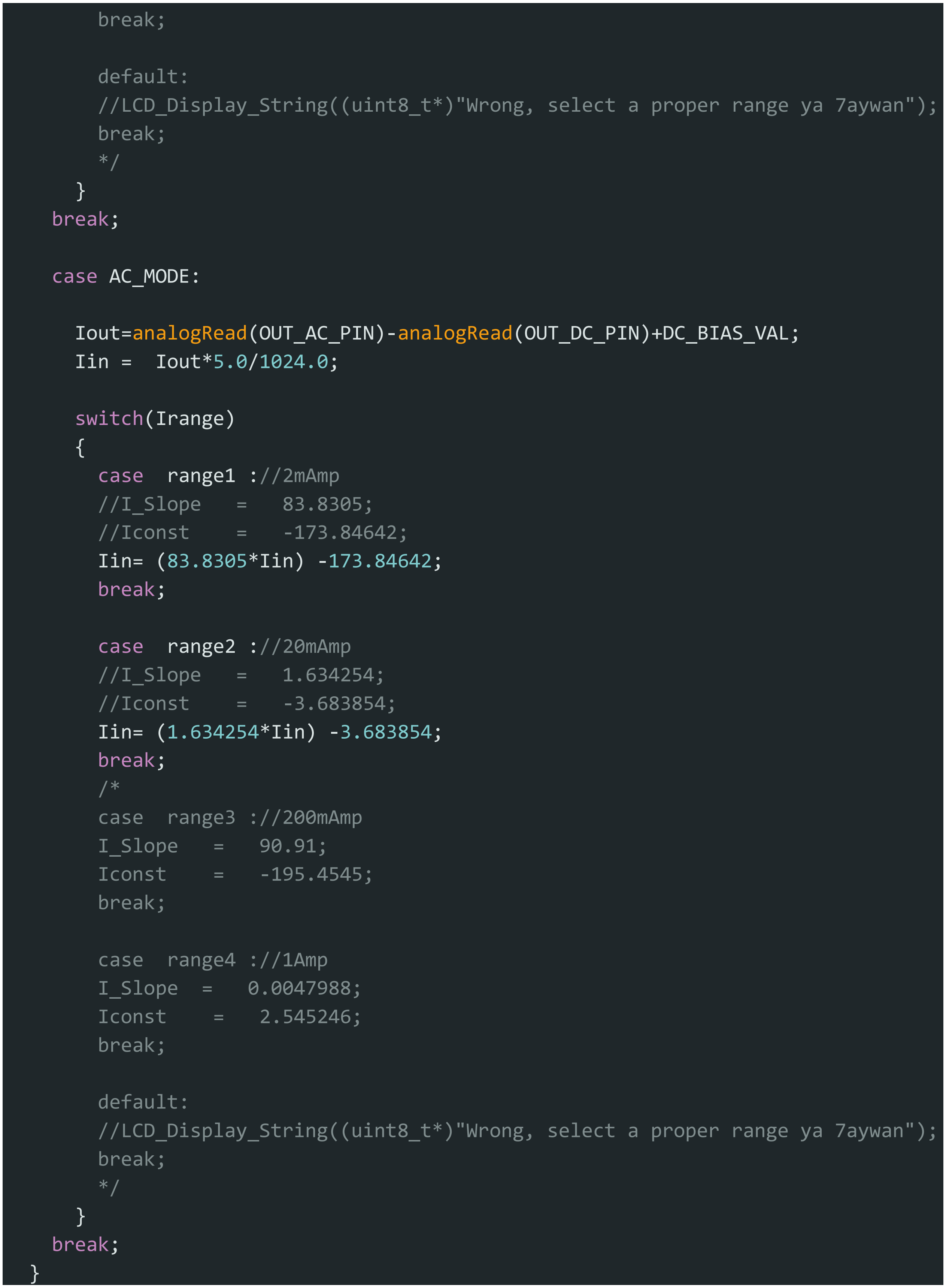


```
      {
        case  range1 ://300mV
        //V_Slope   =   0.139574;
        //Vconst    =   -0.28634;
        Vin= (0.139574*Vin) -0.28634;
        break;

        case  range2 ://3V
        //V_Slope   =   1.394078;
        //Vconst    =   -2.96428;
        Vin= (1.394078*Vin) -2.96428;
        break;
        /*
        case  range3 ://30V
        V_Slope  =   0.0587325;
        Vconst    =   2.5144325;
        break;

        case  range4 ://400V
        V_Slope  =   0.0047988;
        Vconst    =   2.545246;
        break;

        default:
        //LCD_Display_String((uint8_t*)"Wrong, select a proper range ya 7aywan");
        break;
        */
      }
    break;

    case AC_MODE:

      Vout=analogRead(OUT_AC_PIN)-analogRead(OUT_DC_PIN)+DC_BIAS_VAL;
      Vin =  Vout*5.0/1024.0;

      switch(Vrange)
      {
        case  range1 ://300mV
        //V_Slope   =   0.1664;
        //Vconst    =   -0.35624;
        Vin= (0.1664*Vin) -0.35624;
        break;

        case  range2 ://3V
        //V_Slope   =   1.634254;
```

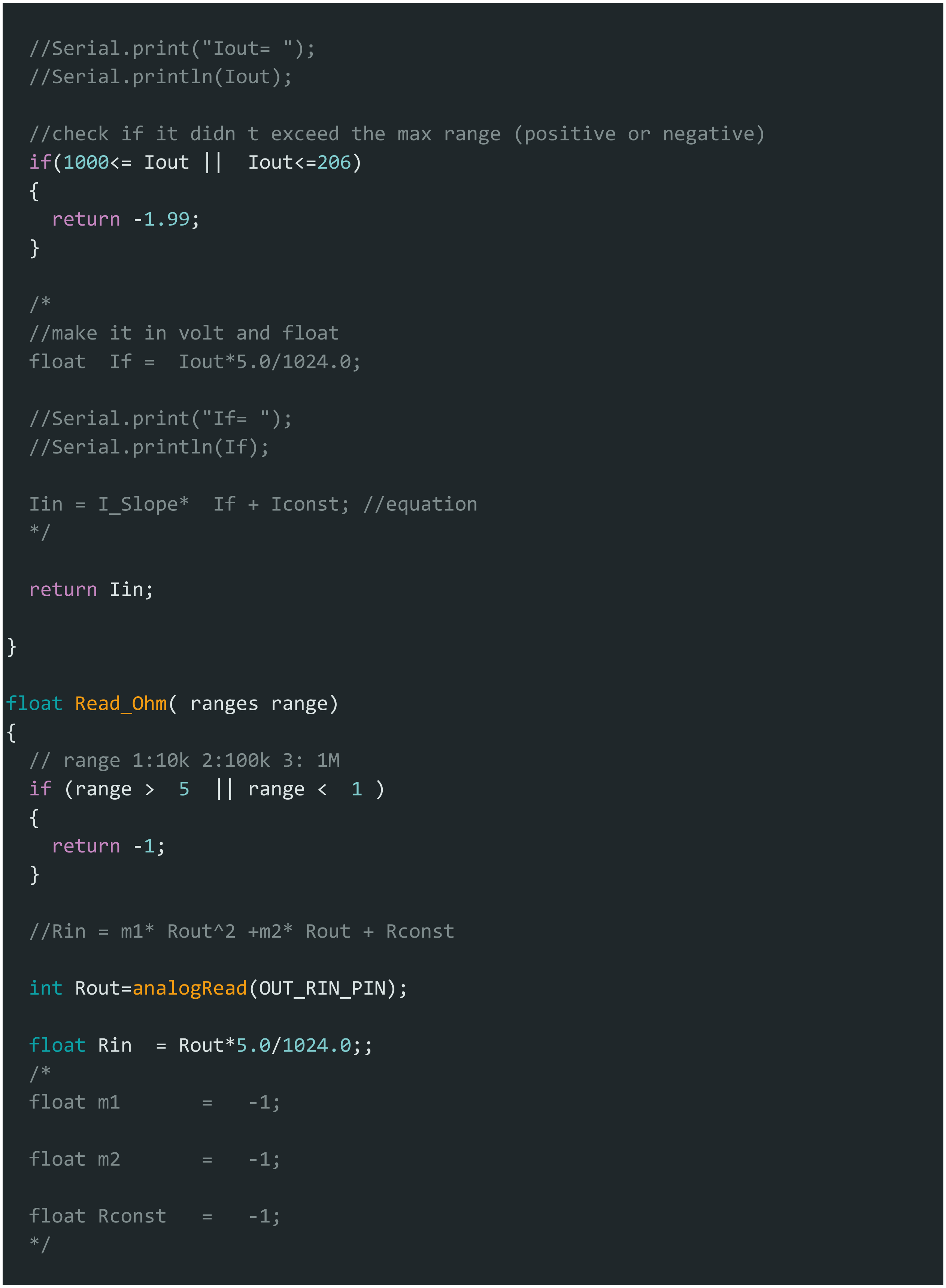


```
        //Vconst    =   -3.683854;
        Vin= (1.634254*Vin) -3.683854;
        break;
        /*
        case  range3 ://30V
        V_Slope  =   0.0587325;
        Vconst    =   2.5144325;
        break;

        case  range4 ://400V
        V_Slope  =   0.0047988;
        Vconst    =   2.545246;
        break;

        default:
        //LCD_Display_String((uint8_t*)"Wrong, select a proper range ya 7aywan");
        break;
        */
      }
    break;
  }

  //check if it didn t exceed the max range (positive or negative)
  if(880<= Vout ||  Vout<=206)
  {
    return -1.99;
  }

  //make it in volt and float
  //float  Vf =  Vout*5.0/1024.0;

  //Serial.print("Vf= ");
  //Serial.println(Vf);

  //Vin = V_Slope*  Vf + Vconst; //equation

  return Vin;

}

float Read_Amp( ranges Irange, modes mode)
{

  // Iin = ISlope *  Iout + Iconst
```

```
int Iout;

Iout=-1;

float Iin;

Iin = -1;
/*
float I_Slope;

I_Slope = -1;

float Iconst;

Iconst  = -1;
*/
 switch(mode)
{
  case DC_MODE:

    Iout=analogRead(OUT_DC_PIN);
    Iin =  Iout*5.0/1024.0;

    switch(Irange)
    {
      case  range1 ://2mAmp
      //I_Slope   =   0.139574;
      //Iconst    =   -0.28634;
      Iin= (0.139574*Iin) -0.28634;
      break;

      case  range2 ://20mAmp
      //I_Slope   =   83.8305;
      //Iconst    =   -173.84642;
      Iin= (83.8305*Iin) -173.84642;
      break;
      /*
      case  range3 ://200mAmp
      I_Slope   =   83.8305;
      Iconst    =   -173.84642;
      break;

      case  range4 ://1Amp
      I_Slope  =   0.0047988;
      Iconst    =   2.545246;
```



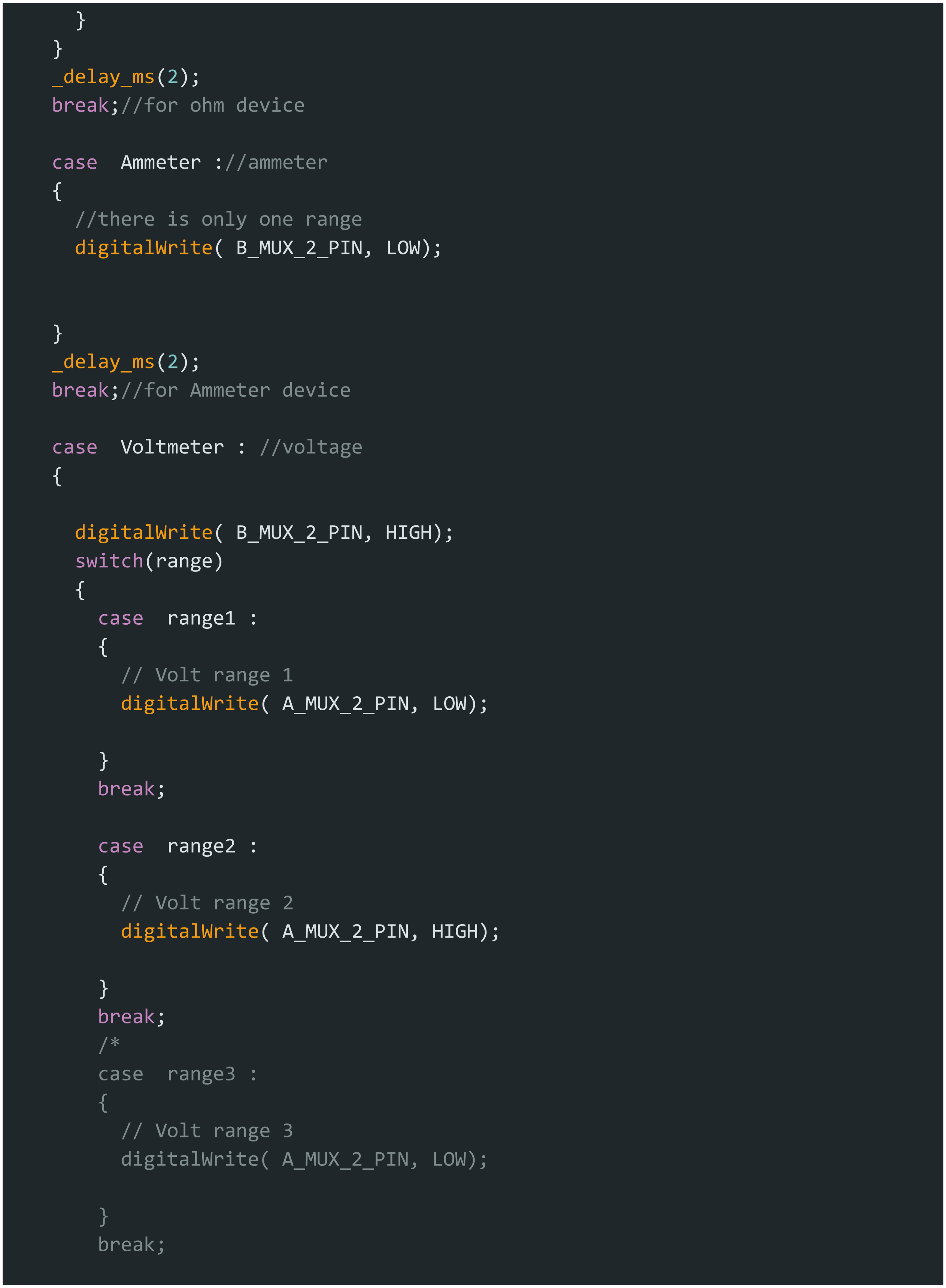
```
        break;

        default:
        //LCD_Display_String((uint8_t*)"Wrong, select a proper range ya 7aywan");
        break;
        */
      }
    break;

    case AC_MODE:

      Iout=analogRead(OUT_AC_PIN)-analogRead(OUT_DC_PIN)+DC_BIAS_VAL;
      Iin =  Iout*5.0/1024.0;

      switch(Irange)
      {
        case  range1 ://2mAmp
        //I_Slope   =   83.8305;
        //Iconst    =   -173.84642;
        Iin= (83.8305*Iin) -173.84642;
        break;

        case  range2 ://20mAmp
        //I_Slope   =   1.634254;
        //Iconst    =   -3.683854;
        Iin= (1.634254*Iin) -3.683854;
        break;
        /*
        case  range3 ://200mAmp
        I_Slope   =   90.91;
        Iconst    =   -195.4545;
        break;

        case  range4 ://1Amp
        I_Slope  =   0.0047988;
        Iconst    =   2.545246;
        break;

        default:
        //LCD_Display_String((uint8_t*)"Wrong, select a proper range ya 7aywan");
        break;
        */
      }
    break;
  }
```



```
  //Serial.print("Iout= ");
  //Serial.println(Iout);

  //check if it didn t exceed the max range (positive or negative)
  if(1000<= Iout ||  Iout<=206)
  {
    return -1.99;
  }

  /*
  //make it in volt and float
  float  If =  Iout*5.0/1024.0;

  //Serial.print("If= ");
  //Serial.println(If);

  Iin = I_Slope*  If + Iconst; //equation
  */

  return Iin;

}

float Read_Ohm( ranges range)
{
  // range 1:10k 2:100k 3: 1M
  if (range >  5  || range <  1 )
  {
    return -1;
  }

  //Rin = m1* Rout^2 +m2* Rout + Rconst

  int Rout=analogRead(OUT_RIN_PIN);

  float Rin  = Rout*5.0/1024.0;;
  /*
  float m1       =   -1;

  float m2       =   -1;

  float Rconst   =   -1;
  */
```

```
switch(range)
{
  case  range1 ://10k  Ohm

    //m1      =   7.408666;

    //m2      =   -58.75646;

    //Rconst  =   118.38925;

    Rin=(7.408666* Rin* Rin) +(-58.75646* Rin) +118.38925;
  break;

  case  range2 ://100k Ohm

  //m1      =   874.13015;

  //m2      =   -8139.9605;

  //Rconst  =   18961.3339;

  Rin=(874.13015* Rin* Rin) +(-8139.9605* Rin) +18961.3339;
  break;

  case  range3 ://1M   Ohm

  //m1      =   17433.97271;

  //m2      =   -165272.9212;

  //Rconst  =   391784.0959;

  Rin=(17433.97271* Rin* Rin) +(-165272.9212* Rin) +391784.0959;

  break;

  default:
  //LCD_Display_String((uint8_t*)"Wrong, select a proper range ya 7aywan");
  break;
}

/*
//make it in volt and float
float  Rf =  Rout*5.0/1024.0;
```

```
  //Serial.print("Rf= ");
  //Serial.println(Rf);

  Rin=m1* Rf* Rf + m2* Rf + Rconst;
  */

  return Rin;

}

void Select_Mux(devices device, ranges range)
{

  switch(device)
  {
    case  Ohmeter :
    { //Ohm
      switch(range)
      {
        case  range1  :
        {
          // Ohm range 1
          digitalWrite( A_MUX_1_PIN, LOW);
          digitalWrite( B_MUX_1_PIN, LOW);

        }
        break;

        case  range2  :
        {
          // Ohm range 2
          digitalWrite( A_MUX_1_PIN, HIGH);
          digitalWrite( B_MUX_1_PIN, LOW);

        }
        break;

        case  range3  :
        {
          // Ohm range 3
          digitalWrite(  A_MUX_1_PIN,  LOW);
          digitalWrite(  B_MUX_1_PIN, HIGH);

        }
        break;
```



```
      }
    }
    _delay_ms(2);
    break;//for ohm device

    case  Ammeter ://ammeter
    {
      //there is only one range
      digitalWrite( B_MUX_2_PIN, LOW);


    }
    _delay_ms(2);
    break;//for Ammeter device

    case  Voltmeter : //voltage
    {

      digitalWrite( B_MUX_2_PIN, HIGH);
      switch(range)
      {
        case  range1 :
        {
          // Volt range 1
          digitalWrite( A_MUX_2_PIN, LOW);

        }
        break;

        case  range2 :
        {
          // Volt range 2
          digitalWrite( A_MUX_2_PIN, HIGH);

        }
        break;
        /*
        case  range3 :
        {
          // Volt range 3
          digitalWrite( A_MUX_2_PIN, LOW);

        }
        break;
        case  range4 :
        {
          // Volt range 4
          digitalWrite( A_MUX_2_PIN, HIGH);

        }
        break;
        */

      }
    }
    _delay_ms(2);
    break;//for volt device

    case  Square : //Square
    {

      digitalWrite( A_MUX_1_PIN, LOW);
      digitalWrite( B_MUX_1_PIN, LOW);

    }
    _delay_ms(2);
    break;//for Square device

    case  Tri  : //Tri
    {

      digitalWrite( A_MUX_1_PIN, HIGH);
      digitalWrite( B_MUX_1_PIN, LOW);

    }
    _delay_ms(2);
    break;//for Tri  device

    case  Sin  : //Sin
    {

      digitalWrite( A_MUX_1_PIN, LOW);
      digitalWrite( B_MUX_1_PIN, HIGH);


    }
    _delay_ms(2);
    break;//for Sin  device
  }
```

```

  return;
}
```

Ocsi.h:

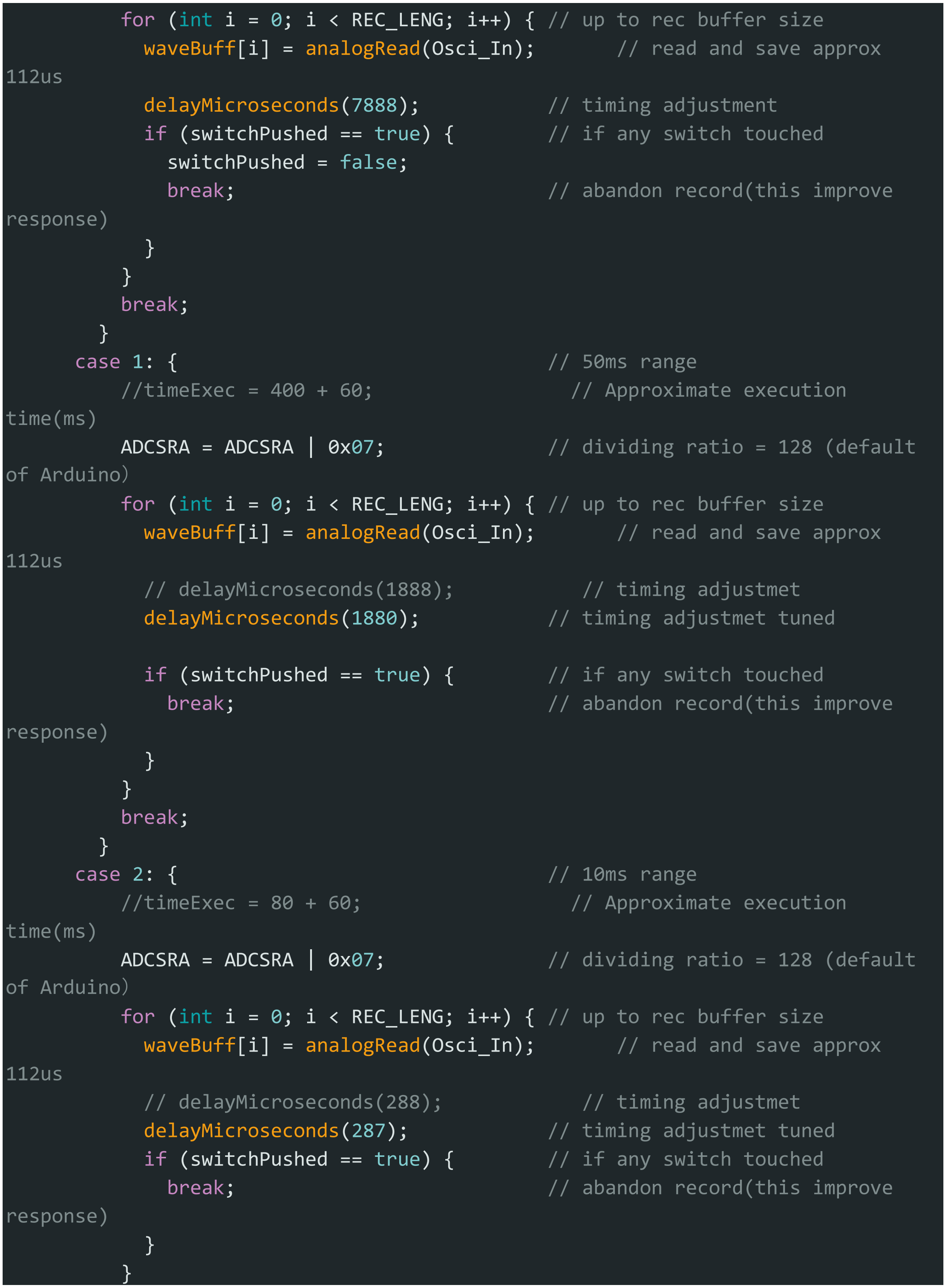
```
/*

 * Osci.h
 *
 * Created: 10/26/2024 2:48 PM
 *  Author: Yousef
 */

#ifndef OSCI_H
#define OSCI_H

/*  Include */
#include <Wire.h>
#include <Adafruit_GFX.h>
//#include <Adafruit_SSD1306.h>
#include "Adafruit_SH1106.h"   // https://github.com/wonho-maker/Adafruit_SH1106
#include "application.h"
//#include <EEPROM.h>

//OLED Definitions
#define SCREEN_WIDTH      (128)   // OLED display width
#define SCREEN_HEIGHT     (64)    // OLED display height
#define OLED_RESET        (-1)    // Reset pin # (or -1 if sharing Arduino reset
pin)
#define REC_LENG          (100)   // size of wave data buffer
#define MIN_TRIG_SWING    (5)     // minimum trigger swing.(Display "Unsync" if
swing smaller than this value
#define OLED_I2C_ADDRESS  (0x3C)

//Pins Number
#define   Osci_Input_Bot    (2)
#define   Select_Bot        (8)
#define   Exit_Bot          (3)
#define   Up_Bot            (9)
#define   Down_Bot          (10)
#define   Hold_Bot          (11)
#define   Osci_In           (A0)
#define   Sig_In

/*
typedef struct flag_type
{
  uint8_t button_up_f: 1;
  uint8_t button_down_f: 1;
  uint8_t button_select_f: 1;
};
*/

void Osci_Init(void);

void Osci_Run(void);

static void setConditions(char* hScale, char* vScale,int &rangeMax,int
&rangeMaxDisp,int &rangeMin,int &rangeMinDisp);

static void writeCommonImage(void);

static void readWave(int *waveBuff);

static void dataAnalize(int &dataMin,int &dataMax,int &dataAve,int *waveBuff,int
&trigP,boolean &trigSync,float &waveFreq,int &waveDuty);

static void freqDuty(int &dataMin,int &dataMax,int &dataAve,float &waveFreq,int
&waveDuty,int *waveBuff);

static int sum3(int k,int *waveBuff);

static void startScreen(void);

static void dispHold(void);

static void dispInf(char* hScale, char* vScale,int &dataMin,int &dataMax,int
&dataAve,char *chrBuff,
int &rangeMax,int &rangeMaxDisp,int &rangeMin,int &rangeMinDisp,int
&trigP,boolean &trigSync,float &waveFreq,int &waveDuty);

static void plotData(int *waveBuff,int &rangeMax,int &rangeMaxDisp,int
&rangeMin,int &rangeMinDisp,
int &trigP,boolean &trigSync,float &waveFreq,int &waveDuty);

static void saveEEPROM(void);

static void loadEEPROM(void);

static void auxFunctions(void);

static void uuPinOutputLow(unsigned int d, unsigned int a);

static void pin2IRQ(void);
static void pin3IRQ(void);




#endif /* OSCI_H  */
```

Ocsi.cpp:

```
#include "Osci.h"


#define MAX_VRANGE      (5)
#define MAX_HRANGE      (6)
#define LSB_5V          (0.00566826f)  // Sensitivity coefficient of 5V range.
std=0.00563965, 1.1*630/(1024*120)
//float LSB_5V = 0.00566826;      // sensivity coefficient of 5V range.
std=0.00563965 1.1*630/(1024*120)
//float lsb50V = 0.05243212;     // sensivity coefficient of 50V range.
std=0.0512898 1.1*520.91/(1024*10.91)


// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
//Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);   //
device name is oled
Adafruit_SH1106 oled(OLED_RESET);        // use this when SH1106

// Range name table (those are stored in flash memory)
//const char vRangeName[10][5] PROGMEM = {"A50V", "A 5V", " 50V", " 20V", " 10V",
"  5V", "  2V", "  1V", "0.5V", "0.2V"}; // Vertical display character (number of
characters including \ 0 is required)
const char vstring_table[MAX_VRANGE] [5] PROGMEM = { "  5V", "  2V", "  1V",
"0.5V", "0.2V"};
//const char hRangeName[10][6] PROGMEM = {"200ms", "100ms", " 50ms", " 20ms", "
10ms", "  5ms", "  2ms", "  1ms", "500us", "200us"};  //  Hrizontal display
characters
const char hstring_table[MAX_HRANGE] [6] PROGMEM = {"200ms", " 50ms", " 10ms",
"  2ms", "500us", "200us"};
const PROGMEM float hRangeValue[] = { 0.2, 0.05, 0.01, 0.002, 0.5e-3, 0.2e-3}; //
horizontal range value in second. ( = 25pix on screen)
flag_type flags1 = { };

//volatile char trigD;                   // trigger slope flag,     0:positive
1:negative
volatile boolean scopeP;                 // operation scope position number.
0:Veratical, 1:Hrizontal, 2:Trigger slope
volatile boolean hold = false;           // hold flag
volatile boolean exitflag = false;       // hold flag
volatile boolean switchPushed = false;   // flag of switch pusshed !
//volatile int saveTimer;                // remaining time for saving EEPROM
//int timeExec;                          // approx. execution time of current
range setting (ms)
```

```
volatile char vRange;                   // V-range number
0:5V,   1:2V,  2:1V,  3:0.5V,  4:0.2V
volatile char hRange;                   // H-range nubmer 0:200ms,  1:50ms,
2:10ms, 3;2ms, 4:500us, 5;200us

void Osci_Init(void)
{
  pinMode(Osci_Input_Bot, INPUT_PULLUP);             // button pussed interrupt
(int.0 IRQ)
  pinMode(Select_Bot, INPUT_PULLUP);          // Select button
  pinMode(Up_Bot , INPUT_PULLUP);             // Up
  pinMode(Down_Bot, INPUT_PULLUP);            // Down
  pinMode(Hold_Bot, INPUT_PULLUP);            // Hold
  pinMode(Exit_Bot, INPUT_PULLUP);            // Hold
  //pinMode(12, INPUT);                   // 1/10 attenuator(Off=High-Z,
Enable=Output Low)
  pinMode(LED_BUILTIN, OUTPUT);


  //oled.begin(SSD1306_SWITCHCAPVCC, 0x3D);  // select 3C or 3D (set your OLED
I2C address)
  oled.begin(SH1106_SWITCHCAPVCC, OLED_I2C_ADDRESS);  // use this when SH1106

  //auxFunctions();                       // Voltage measure (never return)
  //loadEEPROM();                         // read last settings from EEPROM
  analogReference(INTERNAL);            // ADC full scale = 1.1V
  attachInterrupt(0, pin2IRQ, FALLING); // activate IRQ at falling edge mode
  //attachInterrupt(1, pin3IRQ, LOW);

}


void Osci_Run(void) {

  int waveBuff[REC_LENG];        // wave form buffer (RAM remaining capacity is
barely)
  char chrBuff[8];               // display string buffer
  char hScale[] = "xxxAs";       // horizontal scale character
  char vScale[] = "xxxx";        // vartical scale


  int dataMin;                   // buffer minimum value (smallest=0)
  int dataMax;                   //        maximum value (largest=1023)
  int dataAve;                   // 10 x average value (use 10x value to keep
accuracy. so, max=10230)
```

```
  int rangeMax;                  // buffer value to graph full swing
  int rangeMin;                  // buffer value of graph botto
  int rangeMaxDisp;              // display value of max. (100x value)
  int rangeMinDisp;              // display value if min.
  int trigP;                     // trigger position pointer on data buffer
  boolean trigSync;              // flag of trigger detected
  //int att10x;                    // 10x attenetor ON (effective when 1)

  float waveFreq;                // frequency (Hz)
  int waveDuty;                // duty ratio (%)

  hold = false;          // hold flag
  exitflag = false;      // hold flag
  switchPushed = false;  // flag of switch pusshed !

  vRange=1;             // V-range number 0:5V,   1:2V,  2:1V,  3:0.5V,  4:0.2V
  hRange=1;             // H-range nubmer 0:200ms,  1:50ms, 2:10ms, 3;2ms,
4:500us, 5;200us
  //trigD=1;            // trigger slope flag,     0:positive 1:negative
  scopeP=false;

  startScreen(); // display start message
  flags1.button_select_f = 0;


  while(flags1.button_select_f == 0){

    setConditions(hScale,vScale,rangeMax,rangeMaxDisp,rangeMin,rangeMinDisp);
                 // set measurment conditions
    readWave(waveBuff);      // read wave form and store into buffer memory
    setConditions(hScale,vScale,rangeMax,rangeMaxDisp,rangeMin,rangeMinDisp);
       // set measurment conditions again (reflect change during measure)
    dataAnalize(dataMin,dataMax,dataAve,waveBuff,trigP,trigSync,waveFreq,waveDuty
);                      // analize data
    writeCommonImage();                      // write fixed screen image (2.6ms)
    plotData(waveBuff,rangeMax,rangeMaxDisp,rangeMin,rangeMinDisp,trigP,trigSync,
waveFreq,waveDuty);                          // plot waveform (10-18ms)
    dispInf(hScale,
vScale,dataMin,dataMax,dataAve,chrBuff,rangeMax,rangeMaxDisp,rangeMin,rangeMinDis
p,trigP,trigSync,waveFreq,waveDuty);                            // display
information (6.5-8.5ms)
    oled.display();                          // send screen buffer to OLED (37ms)
    //saveEEPROM();                          // save settings to EEPROM if necessary
    while (hold == true) {                  // wait if Hold flag ON
      dispHold();
      delay(10);                            // loop cycle speed = 60-470ms (buffer
size = 200)
    }

    if((digitalRead(Exit_Bot) == LOW))
    {
      delay(30); // Rebounce Delay
      if((digitalRead(Exit_Bot) == LOW)  && (flags1.button_select_f == 0))
      {
        flags1.button_select_f = 1;
      }
    }
    else
    {
      flags1.button_select_f = 0;
    }

  }
  analogReference(DEFAULT);  // Set the ADC reference to the default (AVCC or
5V/3.3V)

  delay(500); // exit Delay

}

  static void setConditions(char* hScale, char* vScale,int &rangeMax,int
&rangeMaxDisp,int &rangeMin,int &rangeMinDisp) {           // measuring condition
setting
    // get range name from PROGMEM
    strcpy_P(hScale, hstring_table[hRange]);  // H range name
    strcpy_P(vScale, vstring_table[vRange]);  // Directly read from PROGMEM

    switch (vRange) {              // setting of Vrange
      case 0: {                    // 5V range
          rangeMax = 5 / LSB_5V;    // set full scale pixcel count number
          rangeMaxDisp = 500;
          rangeMin = 0;
          rangeMinDisp = 0;
          //att10x = 0;              // no input attenuator
          break;
        }
      case 1: {                    // 2V range
          rangeMax = 2 / LSB_5V;    // set full scale pixcel count number
          rangeMaxDisp = 200;
          rangeMin = 0;
          rangeMinDisp = 0;
          //att10x = 0;              // no input attenuator
          break;
        }
      case 2: {                    // 1V range
          rangeMax = 1 / LSB_5V;    // set full scale pixcel count number
          rangeMaxDisp = 100;
          rangeMin = 0;
          rangeMinDisp = 0;
          //att10x = 0;              // no input attenuator
          break;
        }
      case 3: {                    // 0.5V range
          rangeMax = 0.5 / LSB_5V;  // set full scale pixcel count number
          rangeMaxDisp = 50;
          rangeMin = 0;
          rangeMinDisp = 0;
          //att10x = 0;              // no input attenuator
          break;
        }
      case 4: {                    // 0.5V range
          rangeMax = 0.2 / LSB_5V;  // set full scale pixcel count number
          rangeMaxDisp = 20;
          rangeMin = 0;
          rangeMinDisp = 0;
          //att10x = 0;              // no input attenuator
          break;
        }
    }
}

  static void writeCommonImage() {                 // Common screen image drawing
    oled.clearDisplay();                    // erase all(0.4ms)
    oled.setTextColor(WHITE);               // write in white character
    oled.setCursor(85, 0);                  // Start at top-left corner
    oled.println(F("av    v"));             // 1-st line fixed characters
    oled.drawFastVLine(26, 9, 55, WHITE);   // left vartical line
    oled.drawFastVLine(127, 9, 3, WHITE);   // right vrtical line up
    oled.drawFastVLine(127, 61, 3, WHITE);  // right vrtical line bottom

    oled.drawFastHLine(24, 9, 7, WHITE);    // Max value auxiliary mark
    oled.drawFastHLine(24, 36, 2, WHITE);
    oled.drawFastHLine(24, 63, 7, WHITE);
    oled.drawFastHLine(51, 9, 3, WHITE);     // Max value auxiliary mark
    oled.drawFastHLine(51, 63, 3, WHITE);

    oled.drawFastHLine(76, 9, 3, WHITE);     // Max value auxiliary mark
    oled.drawFastHLine(76, 63, 3, WHITE);

    oled.drawFastHLine(101, 9, 3, WHITE);    // Max value auxiliary mark
    oled.drawFastHLine(101, 63, 3, WHITE);

    oled.drawFastHLine(123, 9, 5, WHITE);    // right side Max value auxiliary mark
    oled.drawFastHLine(123, 63, 5, WHITE);

    for (int x = 26; x <= 128; x += 5) {
      oled.drawFastHLine(x, 36, 2, WHITE);   // Draw the center line (horizontal line) with a dotted line
    }
    for (int x = (127 - 25); x > 30; x -= 25) {
      for (int y = 10; y < 63; y += 5) {
        oled.drawFastVLine(x, y, 2, WHITE); // Draw 3 vertical lines with dotted lines
      }
    }
}

  static void readWave(int *waveBuff) {                            // Record waveform to memory array

    //if (att10x == 1) {                             // if 1/10 attenuator required
      //pinMode(12, OUTPUT);                         // assign attenuator controle pin to OUTPUT,
      //digitalWrite(12, LOW);                       // and output LOW (output 0V)
    //} else {                                       // if not required
      //pinMode(12, INPUT);                          // assign the pin input (Hi-z)
    //}
    switchPushed = false;                          // Clear switch operation flag
    ADCSRA = ADCSRA & 0xf8;
    switch (hRange) {                              // set recording conditions in accordance with the range number
      case 0: {                                    // 200ms range
          //timeExec = 1600 + 60;                    // Approximate execution time(ms) Used for countdown until saving to EEPROM
          ADCSRA = ADCSRA | 0x07;                  // dividing ratio = 128 (default of Arduino)
```



```
          for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
            waveBuff[i] = analogRead(Osci_In);       // read and save approx
112us
            delayMicroseconds(7888);           // timing adjustment
            if (switchPushed == true) {        // if any switch touched
              switchPushed = false;
              break;                           // abandon record(this improve
response)
            }
          }
          break;
        }
      case 1: {                                // 50ms range
          //timeExec = 400 + 60;                 // Approximate execution
time(ms)
          ADCSRA = ADCSRA | 0x07;              // dividing ratio = 128 (default
of Arduino)
          for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
            waveBuff[i] = analogRead(Osci_In);       // read and save approx
112us
            // delayMicroseconds(1888);           // timing adjustmet
            delayMicroseconds(1880);           // timing adjustmet tuned

            if (switchPushed == true) {        // if any switch touched
              break;                           // abandon record(this improve
response)
            }
          }
          break;
        }
      case 2: {                                // 10ms range
          //timeExec = 80 + 60;                  // Approximate execution
time(ms)
          ADCSRA = ADCSRA | 0x07;              // dividing ratio = 128 (default
of Arduino)
          for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
            waveBuff[i] = analogRead(Osci_In);       // read and save approx
112us
            // delayMicroseconds(288);            // timing adjustmet
            delayMicroseconds(287);            // timing adjustmet tuned
            if (switchPushed == true) {        // if any switch touched
              break;                           // abandon record(this improve
response)
            }
          }
          break;
        }
      case 3: {                                // 2ms range
          //timeExec = 16 + 60;                  // Approximate execution
time(ms)
          ADCSRA = ADCSRA | 0x06;              // dividing ratio = 64 (0x1=2,
0x2=4, 0x3=8, 0x4=16, 0x5=32, 0x6=64, 0x7=128)
          for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
            waveBuff[i] = analogRead(Osci_In);       // read and save approx 56us
            // delayMicroseconds(24);            // timing adjustmet
            delayMicroseconds(23);             // timing adjustmet tuned
          }
          break;
        }
      case 4: {                                // 500us range
          //timeExec = 4 + 60;                   // Approximate execution
time(ms)
          ADCSRA = ADCSRA | 0x04;              // dividing ratio = 16(0x1=2,
0x2=4, 0x3=8, 0x4=16, 0x5=32, 0x6=64, 0x7=128)
          for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
            waveBuff[i] = analogRead(Osci_In);       // read and save approx 16us
            delayMicroseconds(4);              // timing adjustmet
            // time fine adjustment 0.0625 x 8 = 0.5us (nop=0.0625us @16MHz)
            asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
asm("nop"); asm("nop"); asm("nop");
          }
          break;
        }
      case 5: {                                // 200us range
          //timeExec = 2 + 60;                   // Approximate execution
time(ms)
          ADCSRA = ADCSRA | 0x02;              // dividing ratio = 4(0x1=2,
0x2=4, 0x3=8, 0x4=16, 0x5=32, 0x6=64, 0x7=128)
          for (int i = 0; i < REC_LENG; i++) { // up to rec buffer size
            waveBuff[i] = analogRead(Osci_In);       // read and save approx 6us
            // time fine adjustment 0.0625 * 20 = 1.25us (nop=0.0625us @16MHz)
            asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
            asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop");
          }
          break;
        }
    }
}
```

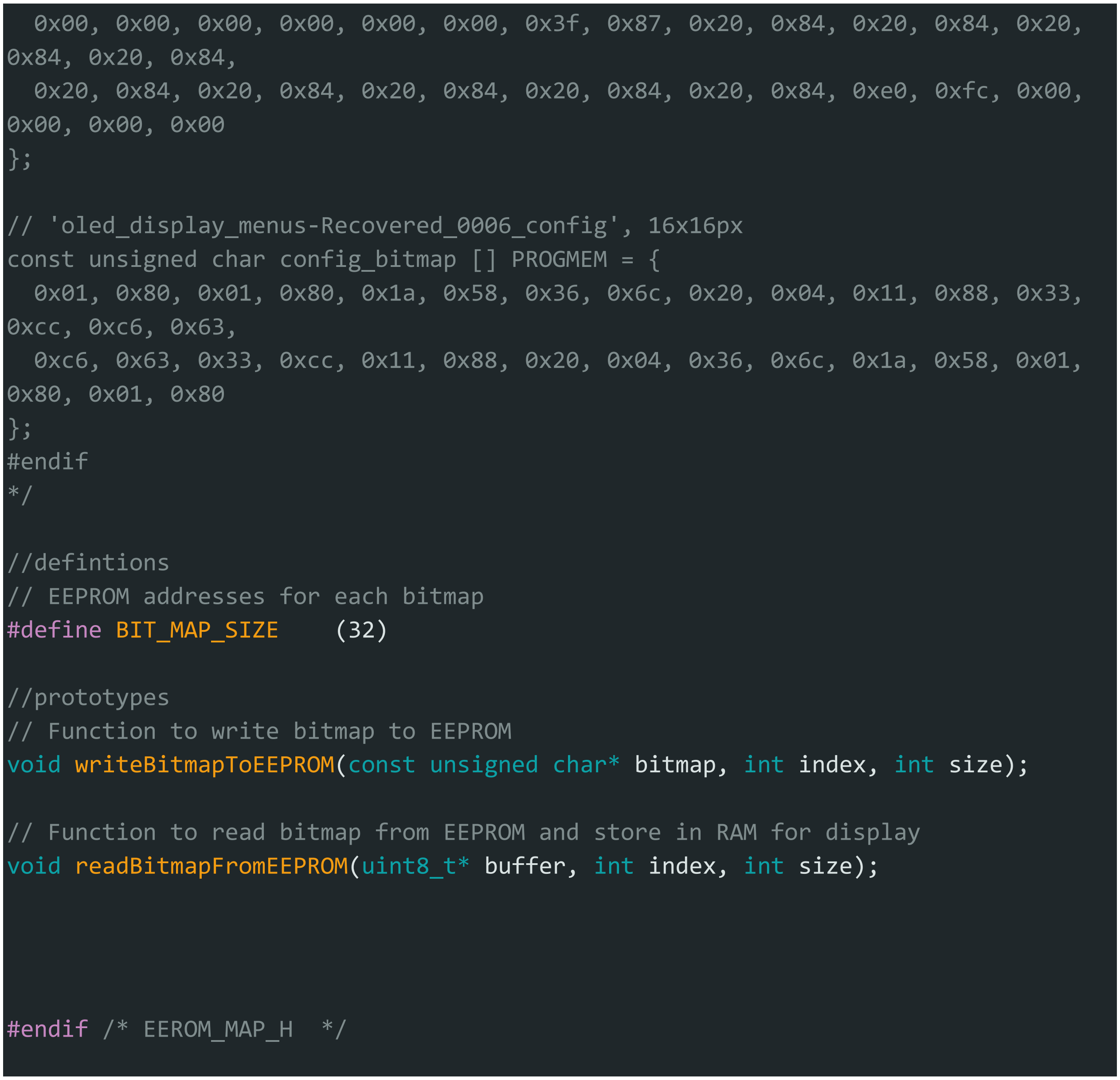
```
static void dataAnalize(int &dataMin,int &dataMax,int &dataAve,int *waveBuff,int &trigP,boolean &trigSync,float &waveFreq,int &waveDuty) {
  // get various information from wave form

  #define MAX_ADC_VALUE         (1023)                    // Maximum ADC value for 10-bit resolution
  #define MIN_ADC_VALUE         (0)                       // Minimum ADC value
  #define AVERAGE_DIVISOR       (20)                      // Divisor for average calculation
  #define TRIG_CENTER           (REC_LENG / 2)            // Center point in the data range
  #define MEDIAN_SEARCH_RADIUS  (REC_LENG/4)              // Range to search around center
  #define TRIG_OFFSET           (MEDIAN_SEARCH_RADIUS+1)  // Range offset for trigger search (adjusted to 51 based on new size)


  int d;
  long sum = 0;

  // search max and min value
  dataMin = 1023;                          // min value initialize to big number
  dataMax = 0;                             // max value initialize to small number
  for (int i = 0; i < REC_LENG; i++) {     // serach max min value
    d = waveBuff[i];
    sum = sum + d;
    if (d < dataMin) {                     // update min
      dataMin = d;
    }
    if (d > dataMax) {                     // updata max
      dataMax = d;
    }
  }

  // calculate average
  dataAve = (sum + 10) / 20;               // Average value calculation (calculated by 10 times to improve accuracy)

  // Trigger position search
  for (trigP = ((REC_LENG / 2) - TRIG_OFFSET); trigP < ((REC_LENG / 2) + MEDIAN_SEARCH_RADIUS); trigP++) { // Find the points that straddle the median at the center ± 50 of the data range
      // if trigger direction is positive
      if ((waveBuff[trigP - 1] < (dataMax + dataMin) / 2) && (waveBuff[trigP] >=
(dataMax + dataMin) / 2)) {
        break;                                    // positive trigger position
found !
      }

  }
  trigSync = true;
  if (trigP >= ((REC_LENG / 2) + MEDIAN_SEARCH_RADIUS)) {           // If the
trigger is not found in range
    trigP = (REC_LENG / 2);                       // Set it to the center for the
time being
    trigSync = false;                             // set Unsync display flag
  }
  if ((dataMax - dataMin) <= MIN_TRIG_SWING) {    // amplitude of the waveform
smaller than the specified value
    trigSync = false;                             // set Unsync display flag
  }
  freqDuty(dataMin,dataMax,dataAve,waveFreq,waveDuty,waveBuff);
}

static void freqDuty(int &dataMin,int &dataMax,int &dataAve,float &waveFreq,int
&waveDuty,int *waveBuff) {                               // detect frequency and
duty cycle value from waveform data
  int swingCenter;                              // center of wave (half of p-p)
  float p0 = 0;                                 // 1-st posi edge
  float p1 = 0;                                 // total length of cycles
  int p2 = 0;                                 // total length of pulse high time
  float pFine = 0;                              // fine position (0-1.0)
  float lastPosiEdge;                           // last positive edge position

  float pPeriod;                                // pulse period
  float pWidth;                                 // pulse width

  int p1Count = 0;                              // wave cycle count
  int p2Count = 0;                              // High time count

  boolean a0Detected = false;
  //  boolean b0Detected = false;
  boolean posiSerch = true;                      // true when serching posi edge

  swingCenter = (3 * (dataMin + dataMax)) / 2;   // calculate wave center value

  for (int i = 1; i < REC_LENG - 2; i++) {       // scan all over the buffer
    if (posiSerch == true) {   // posi slope (frequency serch)
```

```
      if ((sum3(i,waveBuff) <= swingCenter) && (sum3(i + 1,waveBuff) >
swingCenter)) {  // if across the center when rising (+-3data used to eliminate
noize)
        pFine = (float)(swingCenter - sum3(i,waveBuff)) / ((swingCenter -
sum3(i,waveBuff)) + (sum3(i + 1,waveBuff) - swingCenter) );  // fine cross point
calc.
        if (a0Detected == false) {               // if 1-st cross
          a0Detected = true;                     // set find flag
          p0 = i + pFine;                        // save this position as
startposition
        } else {
          p1 = i + pFine - p0;                   // record length (length of
n*cycle time)
          p1Count++;
        }
        lastPosiEdge = i + pFine;                // record location for Pw
calcration
        posiSerch = false;
      }
    } else {   // nega slope serch (duration serch)
      if ((sum3(i,waveBuff) >= swingCenter) && (sum3(i + 1,waveBuff) <
swingCenter)) {  // if across the center when falling (+-3data used to eliminate
noize)
        pFine = (float)(sum3(i,waveBuff) - swingCenter) / ((sum3(i,waveBuff) -
swingCenter) + (swingCenter - sum3(i + 1,waveBuff)) );
        if (a0Detected == true) {
          p2 = p2 + (i + pFine - lastPosiEdge);  // calucurate pulse width and
accumurate it
          p2Count++;
        }
        posiSerch = true;
      }
    }
  }

  pPeriod = p1 / p1Count;                  // pulse period
  pWidth = p2 / p2Count;                   // palse width

  waveFreq = 1.0 / ((pgm_read_float(hRangeValue + hRange) * pPeriod) / 25.0); //
frequency
  waveDuty = 100.0 * pWidth / pPeriod;                                       //
duty ratio
}
```

```
static int sum3(int k,int *waveBuff) {       // Sum of before and after and own
value
  int m = waveBuff[k - 1] + waveBuff[k] + waveBuff[k + 1];
  return m;
}

static void startScreen() {                      // Staru up screen
  oled.clearDisplay();
  oled.setTextSize(1);                    // at double size character
  oled.setTextColor(WHITE);
  //for(int i=0;i++;i<=40){
    oled.setCursor(40, 0);
    oled.println(F("ArduScope"));
    oled.setCursor(30, 20);
    oled.println(F("Oscilloscope"));
    oled.setCursor(55, 42);
    oled.println(F(";)"));
    oled.display();
    delay(50);
  //}

  delay(1500);
  oled.clearDisplay();
  oled.setTextSize(1);                    // After this, standard font size
}

static void dispHold() {                         // display "Hold"
  oled.fillRect(42, 11, 24, 8, BLACK);    // black paint 4 characters
  oled.setCursor(42, 11);
  oled.print(F("Hold"));                  // Hold
  oled.display();                         //
}

static void dispInf(char* hScale, char* vScale,int &dataMin,int &dataMax,int
&dataAve,char *chrBuff,
int &rangeMax,int &rangeMaxDisp,int &rangeMin,int &rangeMinDisp,int
&trigP,boolean &trigSync,float &waveFreq,int &waveDuty)
{                          // Display of various information
  float voltage;
  // display vertical sensitivity
  oled.setCursor(2, 0);                   // around top left
  oled.print(vScale);                     // vertical sensitivity value
  if (scopeP == false) {                      // if scoped
    oled.drawFastHLine(0, 7, 27, WHITE);  // display scoped mark at the bottom
    oled.drawFastVLine(0, 5, 2, WHITE);
    oled.drawFastVLine(26, 5, 2, WHITE);
  }

  // horizontal sweep speed
  oled.setCursor(34, 0);                  //
  oled.print(hScale);                     // display sweep speed (time/div)
  if (scopeP == true) {                      // if scoped
    oled.drawFastHLine(32, 7, 33, WHITE); // display scoped mark at the bottom
    oled.drawFastVLine(32, 5, 2, WHITE);
    oled.drawFastVLine(64, 5, 2, WHITE);
  }

  // trigger polarity
  oled.setCursor(75, 0);                  // at top center
  oled.print(char(0x18));                 // up mark

  // average voltage
  voltage = dataAve * LSB_5V / 10.0;          // 5V range value
  if (voltage < 10.0) {                      // if less than 10V
    dtostrf(voltage, 4, 2, chrBuff);         // format x.xx
  } else {                                   // no!
    dtostrf(voltage, 4, 1, chrBuff);         // format xx.x
  }
  oled.setCursor(98, 0);                     // around the top right
  oled.print(chrBuff);                       // display average voltage圧の平均値
を表示
  //  oled.print(saveTimer);                 // use here for debugging

  // vartical scale lines
  voltage = rangeMaxDisp / 100.0;            // convart Max voltage
  dtostrf(voltage, 4, 2, chrBuff);
  oled.setCursor(0, 9);
  oled.print(chrBuff);                       // display Max value

  voltage = (rangeMaxDisp + rangeMinDisp) / 200.0; // center value calculation
  dtostrf(voltage, 4, 2, chrBuff);
  oled.setCursor(0, 33);
  oled.print(chrBuff);                       // display the value

  voltage = rangeMinDisp / 100.0;            // convart Min vpltage
  dtostrf(voltage, 4, 2, chrBuff);
  oled.setCursor(0, 57);
  oled.print(chrBuff);                       // display the value

  // display frequency, duty % or trigger missed
  if (trigSync == false) {                   // If trigger point can't found
    oled.fillRect(92, 14, 24, 8, BLACK);     // black paint 4 character
    oled.setCursor(92, 14);                  //
    oled.print(F("unSync"));                 // dosplay Unsync
  } else {
    oled.fillRect(90, 12, 25, 9, BLACK);    // erase Freq area
    oled.setCursor(91, 13);                  // set display locatio
    if (waveFreq < 100.0) {                  // if less than 100Hz
      oled.print(waveFreq, 1);               // display 99.9Hz
      oled.print(F("Hz"));
    } else if (waveFreq < 1000.0) {          // if less than 1000Hz
      oled.print(waveFreq, 0);               // display 999Hz
      oled.print(F("Hz"));
    } else if (waveFreq < 10000.0) {         // if less than 10kHz
      oled.print((waveFreq / 1000.0), 2);    // display 9.99kH
      oled.print(F("kH"));
    } else {                                 // if more
      oled.print((waveFreq / 1000.0), 1);    // display 99.9kH
      oled.print(F("kH"));
    }
    oled.fillRect(96, 21, 25, 10, BLACK);    // erase Freq area (as small as
possible)
    oled.setCursor(105, 23);                  // set location
    oled.print(waveDuty, 1);                 // display duty (High level ratio)
in %
    oled.print(F("%"));
  }
}

static void plotData(int *waveBuff,int &rangeMax,int &rangeMaxDisp,int
&rangeMin,int &rangeMinDisp,
int &trigP,boolean &trigSync,float &waveFreq,int &waveDuty)
{                    // plot wave form on OLED
  long y1, y2;
  for (int x = 0; x < REC_LENG/2-1 ; x++) {
    int plotX = map(2*x, 0, REC_LENG - 1, 27, 124);  // Scale the x-coordinate to
fit the full width of the display (128 pixels)

    // Map the data point to vertical positions (y-values) and constrain them to
valid OLED range
    y1 = map(waveBuff[x + trigP - REC_LENG / 4], rangeMin, rangeMax, 63, 9);  //
convert to plot address
    y1 = constrain(y1, 9, 63);  // Constrain to OLED screen height
    y2 = map(waveBuff[x + trigP - REC_LENG / 4 + 1], rangeMin, rangeMax, 63,
9);  // next data point
    y2 = constrain(y2, 9, 63);  // Constrain to OLED screen height

    oled.drawLine(plotX, y1, plotX + 1, y2, WHITE);  // Connect the points with a
line
    //oled.drawLine(plotX, y1, plotX + 1, y2, WHITE);  // Connect the points with
a line
  }
}

/*
static void saveEEPROM() {                    // Save the setting value in EEPROM
after waiting a while after the button operation.
  if (saveTimer > 0) {                 // If the timer value is positive,
    saveTimer = saveTimer - timeExec;  // Timer subtraction
    if (saveTimer < 0) {               // if time up
      EEPROM.write(0, vRange);         // save current status to EEPROM
      EEPROM.write(1, hRange);
      EEPROM.write(2, trigD);
      EEPROM.write(3, scopeP);
    }
  }
}

static void loadEEPROM() {                    // Read setting values from EEPROM
(abnormal values will be corrected to default)
  int x;
  x = EEPROM.read(0);                  // vRange
  if ((x < 0) || (7 < x)) {            // if out side 0-9
    x = 3;                             // default value
  }
  vRange = x;

  x = EEPROM.read(1);                  // hRange
  if ((x < 0) || (9 < x)) {            // if out of 0-9
    x = 3;                             // default value
  }
  hRange = x;
  x = EEPROM.read(2);                  // trigD
  if ((x < 0) || (1 < x)) {            // if out of 0-1
    x = 1;                             // default value
  }
  trigD = x;
  x = EEPROM.read(3);                  // scopeP
  if ((x < 0) || (2 < x)) {            // if out of 0-2
    x = 1;                             // default value
  }
  scopeP = x;
}


static void uuPinOutputLow(unsigned int d, unsigned int a) { // 指定ピンを出力、
LOWに設定
  // PORTx =0, DDRx=1
  unsigned int x;
  x = d & 0x00FF; PORTD &= ~x; DDRD |= x;
  x = d >> 8;     PORTB &= ~x; DDRB |= x;
  x = a & 0x003F; PORTC &= ~x; DDRC |= x;
}
*/

static void pin2IRQ() {                   // Pin2(int.0) interrupr handler
  // Pin8,9,10,11 buttons are bundled with diodes and connected to Pin2.
  // So, if any button is pressed, this routine will start.
  int x;                           // Port information holding variable

  x = PINB;                        // read port B status

  if ( (x & 0x07) != 0x07) {       // if bottom 3bit is not all Hi(any wer
pressed)
    //saveTimer = 5000;              // set EEPROM save timer to 5 secnd
    switchPushed = true;           // switch pushed falag ON
  }
  if ((x & 0x01) == 0) {           // if select button(Pin8) pushed,
    scopeP=!scopeP;                      // forward scope position
  }

  if ((x & 0x02) == 0) {           // if UP button(Pin9) pusshed, and
    if (scopeP == false) {             // scoped vertical range
      vRange++;                    // V-range up !
      if (vRange > (MAX_VRANGE-1)) {            // if upper limit
        vRange = MAX_VRANGE-1;                // stay as is
      }
    }
    if (scopeP == true) {             // if scoped hrizontal range
      hRange++;
                     // H-range up !
      if (hRange > (MAX_HRANGE-1)) {            // if upper limit
        hRange = MAX_HRANGE-1;                // stay as is
      }
    }
  }

  if ((x & 0x04) == 0) {           // if DOWN button(Pin10) pusshed, and
    if (scopeP == false) {             // scoped vertical range
      vRange--;                    // V-range DOWN
      if (vRange < 0) {            // if bottom
        vRange = 0;                // stay as is
      }
    }
    if (scopeP == true) {             // if scoped hrizontal range
      hRange--;                    // H-range DOWN
      if (hRange < 0) {            // if bottom
        hRange = 0;                // satay as is
      }
    }
  }

  if ((x & 0x08) == 0) {           // if HOLD button(pin11) pushed
    hold = ! hold;                 // revers the flag
  }
}
```

eerom_map.h:

```
/*

 * eerom_map.h
 *
 * Created: 10/26/2024 2:48 PM
 *  Author: Yousef
 */

#ifndef EEROM_MAP_H
#define EEROM_MAP_H

/*  Include */
#include <Arduino.h>
#include <EEPROM.h>
/*
//#define SAVE_BIT 1    //if you want to save the bit maps
#ifdef SAVE_BIT
#include <avr/pgmspace.h>  // For PROGMEM

// 'oled_display_menus-Recovered_0004_V', 16x16px
const unsigned char volt_bitmap [] PROGMEM = {
  0xfc, 0x3f, 0x60, 0x0c, 0x60, 0x08, 0x30, 0x10, 0x30, 0x10, 0x18, 0x10, 0x18,
0x20, 0x0c, 0x20,
  0x0c, 0x40, 0x0e, 0x40, 0x06, 0x80, 0x06, 0x80, 0x03, 0x80, 0x03, 0x00, 0x01,
0x00, 0x01, 0x00
};
// 'oled_display_menus-Recovered_0006_A', 16x16px
const unsigned char amm_bitmap [] PROGMEM = {
  0x00, 0x00, 0x03, 0xc0, 0x07, 0xe0, 0x0c, 0x30, 0x0c, 0x30, 0x0c, 0x30, 0x0c,
0x30, 0x0c, 0x30,
  0x0f, 0xf0, 0x0f, 0xf0, 0x0c, 0x30, 0x0c, 0x30, 0x0c, 0x30, 0x0c, 0x30, 0x0c,
0x30, 0x00, 0x00
};
// 'oled_display_menus-Recovered_0005_Ohm', 16x16px
const unsigned char ohm_bitmap [] PROGMEM = {
  0x00, 0x00, 0x07, 0xe0, 0x1f, 0xf8, 0x3c, 0x3c, 0x78, 0x1e, 0x60, 0x06, 0xc0,
0x03, 0xc0, 0x03,
  0xc0, 0x03, 0xc0, 0x03, 0x70, 0x0e, 0x70, 0x0e, 0x38, 0x1c, 0xf8, 0x1f, 0xf8,
0x1f, 0x00, 0x00
};

// 'oled_display_menus-Recovered_0003_sig_gen', 16x16px
const unsigned char sig_gen_bitmap [] PROGMEM = {
```



```
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0x87, 0x20, 0x84, 0x20, 0x84, 0x20,
0x84, 0x20, 0x84,
  0x20, 0x84, 0x20, 0x84, 0x20, 0x84, 0x20, 0x84, 0x20, 0x84, 0xe0, 0xfc, 0x00,
0x00, 0x00, 0x00
};

// 'oled_display_menus-Recovered_0006_config', 16x16px
const unsigned char config_bitmap [] PROGMEM = {
  0x01, 0x80, 0x01, 0x80, 0x1a, 0x58, 0x36, 0x6c, 0x20, 0x04, 0x11, 0x88, 0x33,
0xcc, 0xc6, 0x63,
  0xc6, 0x63, 0x33, 0xcc, 0x11, 0x88, 0x20, 0x04, 0x36, 0x6c, 0x1a, 0x58, 0x01,
0x80, 0x01, 0x80
};
#endif
*/

//defintions
// EEPROM addresses for each bitmap
#define BIT_MAP_SIZE    (32)

//prototypes
// Function to write bitmap to EEPROM
void writeBitmapToEEPROM(const unsigned char* bitmap, int index, int size);

// Function to read bitmap from EEPROM and store in RAM for display
void readBitmapFromEEPROM(uint8_t* buffer, int index, int size);




#endif /* EEROM_MAP_H  */
```

Eerom.cpp:

```
#include "eerom_map.h"


void writeBitmapToEEPROM(const unsigned char* bitmap, int index, int size) {
    for (int i = 0; i < size; i++) {
        // Read byte from PROGMEM and write to EEPROM
        EEPROM.write((index*BIT_MAP_SIZE)+i, pgm_read_byte(&bitmap[i]));
    }
}


void readBitmapFromEEPROM(uint8_t* buffer, int index, int size) {
    for (int i = 0; i < size; i++) {
        buffer[i] = EEPROM.read((index*BIT_MAP_SIZE) + i);
    }
}
```