UM-SJTU Joint Institute

Design of Microprocessor based Systems (VE373)

Final Project Report
An Embedded System of Color Extractor

Chen Yuchi	ID: 518021910935
Lin Yipeng	ID: 518370910060
Zhu Zhengping	ID: 518370910068

December 1, 2021

Contents

1 Introduction to the Overall Design

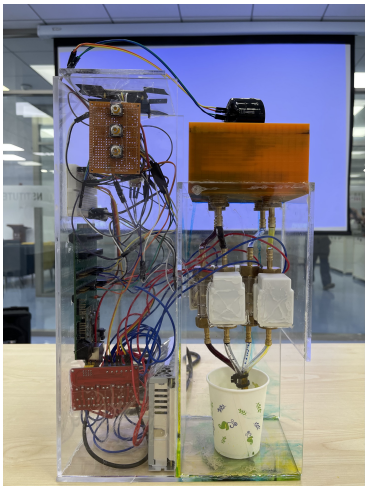
Today, more and more artwork is finished on electronic devices. Digital software allows everyone to be an artist and create beautiful drawings everywhere.

However, when an "RGB artist" decides to hang out and draw the fantastic nature with physical brush and pigment, he or she may be hard to pick a color from environment, to convert RGB values to a formal printable color description, or to mix pigments to achieve a desired color. To overcome these problems, we made the Color Extractor shown in Figure 1, which is able to:

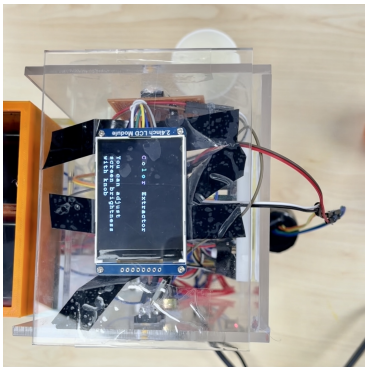
- sense and pick color from environment.
- adjust the picked color according to users' will.
- convert RGB values to a printable color description.
- mix pigments according to printable color description, so that the final color is the same as desired one.

The top level block diagram is shown in Figure 2. The most important three states are **Sample**, **Color Preview & Adjustment and Volume Adjustment** (for simplicity, **Adjustment** is used afterwards), and **Release Ink**. When the system comes to **Sample** state, the color sensor read the RGB value of the presented color, and automatically move forward to **Adjustment** state. In **Adjustment** state, the color just measured, as well as its corresponding RGB values, is displayed on the screen. The users are able to increase or decrease the RGB values according to their preferences, the resulting color will be simultaneously displayed on the screen. They can also choose the volume of ink to be released. From **Adjustment** state, the system can either return to **Sample** state again, or move forward to **Release Ink** state. During **Release Ink** state, the system will release cyan, magenta, yellow, and black (CMYK) ink, with which the target color can be created.

Detailed operation of this embedded system is shown in the video we uploaded. Since it was recorded during demonstration, we are very sorry for the unsatisfying quality.



(a) Side View



(b) Top View

Figure 1: Color Extractor

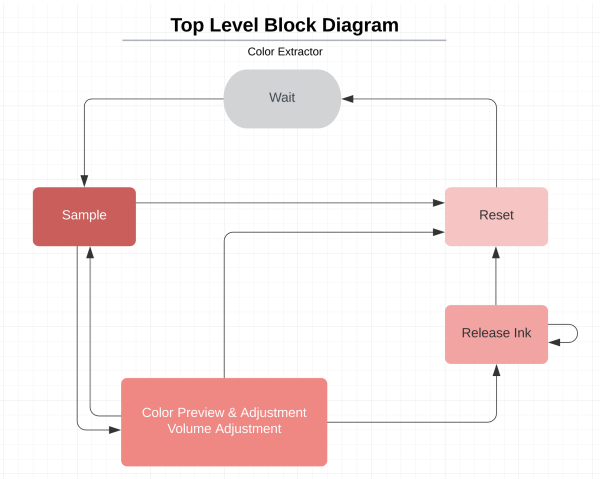


Figure 2: Top Level Block Diagram

2 Project Timeline

Our Team followed the time arrangement as shown in the Gant Diagram in Figure 3. We finished most of the programming that properly drives every single component (e.g. color sensors, screen, valves, etc.,) before July 29. We spent next five days to assemble everything together and finished mechanical setup. The embedded system of color extractor is finally born on August 3.

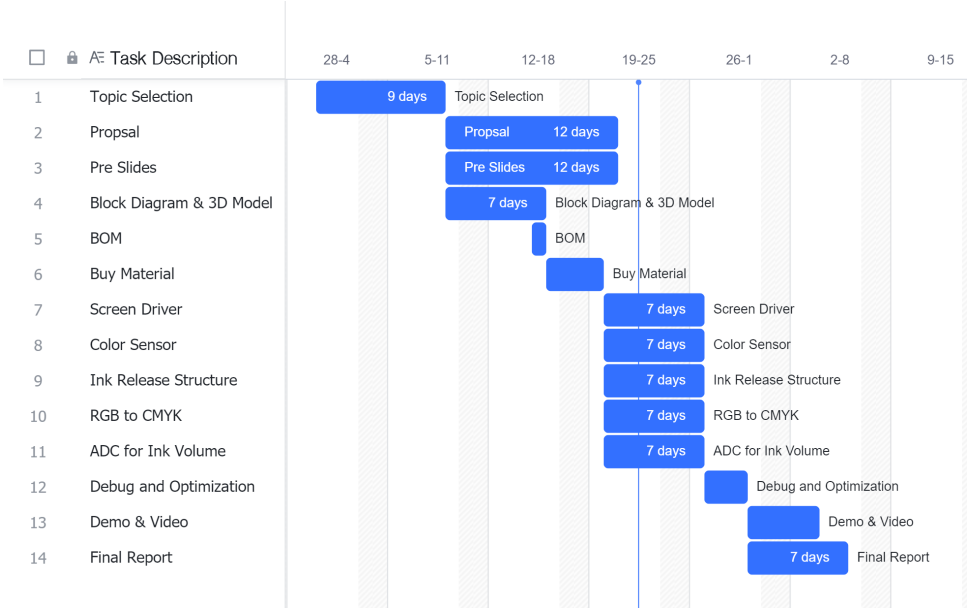


Figure 3: Gant Diagram

3 Software Design

3.1 Color Sensor

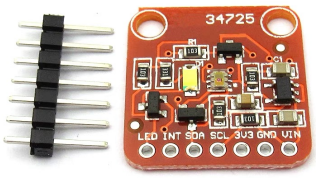


Figure 4: Color Sensor: TCS34725

We chose the color sensor with TCS34725 chip, shown by Figure 4. It communicates with the PIC32 MCU by I^2C module. Actually, the sensor itself is an embedded system that consists of multiple SFRS. We use I^2C module to communicate between PIC32 and TCS34725, and to read and write certain SFRs in both PIC32 and TCS34725. The sensor

is supposed to ultimately come up with three integers of R, G, and B values ranging from 0-255. We applied the following strategies after careful tests to make the output by sensor as accurate as possible.

3.2 LCD Screen

We selected a 2.4 inch LCD screen with ILI9341 chip to produce graphical displays. The screen communicates with the PIC32 with SPI module. The middle layer of the programming is done by the screen vendor (see "lcd_driver.c" in Appendix), which is supposed to be a library for Arduino. The upper layer and bottom layer of the programming are done by our team (see "GUI.c" and "dev_config.c" in Appendix).

3.2.1 SPI Communication

Some important configurations of SPI1 used for screen-MCU communication is listed below:

1. Master mode is selected.
2. Input data is sampled at middle of data output time.
3. Idle state for clock is a low level and active state is a high level.
4. Serial output data changes on transition from active clock state to idle clock state.

3.2.2 Font Display

The font we use is an open source library. It is composed of all ASCII characters. Each character is encoded as a 14×20 matrix. All 1s in the matrix mean to turn on corresponding pixels, while 0s mean to turn off. Each character, or each matrix, is stored as 40 8-bit binary numbers. And all these binary numbers for all fonts are stored in a list.

To display a character, we looked it up in the list to find the starting position of corresponding binary numbers indexed by ASCII codes. Then colors the screen according to the pixels labeled by 1s in the matrix.

3.3 Potentiometer

In this project, we enabled a potentiometer to produce an analog input. The analog input has difference usages in different stages:

1. In **Wait** state, the analog input produced by the potentiometer is used to control the duty cycle of the PWM signal that can adjust the brightness of the LCD screen.
2. In **Adjustment** state, the analog input can be used to adjust the level of ink volume ranging from 1 to 10. Higher level of volume will make the system release more ink.

3.4 RGB to CMYK Conversion

With the RGB values ranging from 0 to 255 measured from color sensor, we performed an RGB-to-CMYK conversion to get the four CMYK values. The corresponding proportion of CMYK ink is released based on the proportion of CMYK values. RGB-to-CMYK conversion is performed based on the formula below:

$$RGB_{max} = \max(R, G, B)$$
$$\text{C} = 100 \times (-R + RGB_{max})/RGB_{max}$$
$$\text{M} = 100 \times (-G + RGB_{max})/RGB_{max}$$
$$\text{Y} = 100 \times (-B + RGB_{max})/RGB_{max}$$
$$\text{K} = 100 \times (1 - RGB_{max}/255)$$

3.5 State Transition

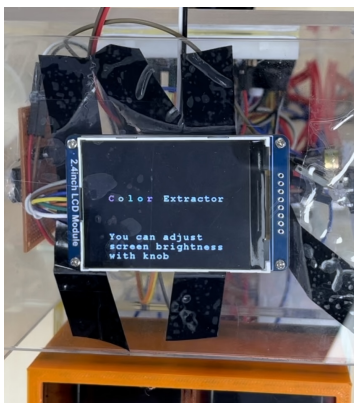


Figure 5: Screen in Wait State

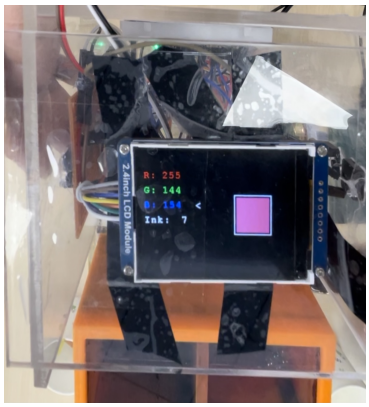


Figure 6: Screen in Adjustment State

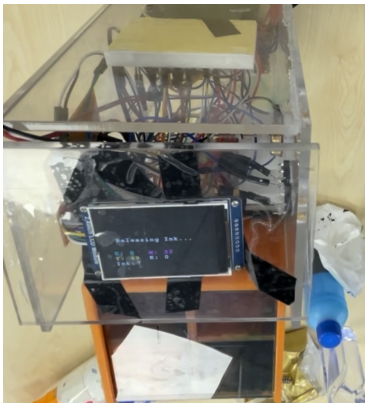


Figure 7: Screen in Release Ink State

The transitions between states are mostly triggered by pressing the five buttons. The five buttons are **Confirm**, **Reset**, **Select**, **Up**, and **Down**.

1. In Wait state, the screen will display as shown in Figure 5. User can rotate the potentiometer to adjust the screen brightness. Pressing the **Confirm** button will make the system move into Sample state.

2. In Sample state, the system will start sampling and automatically move forward to Adjustment state.

3. In Adjustment state, the color preview will be displayed, shown in Figure 6. Rotating the potentiometer can select the level of ink volume to be released. By pressing the **Select** button, user can select one of the RGB value that it is going to adjust. By short pressing the **Up** and **Down** button, the selected RGB value will increase or decrease by 1 at a time. If one keep pressing the **Up** or **Down** button for a while, the selected RGB value will increase or decrease 10 at a time. A short press of the **Reset** button will bring the system back to the Sample state again.

4. In Release Ink state, the screen will display the notice "Releasing Ink ..." and the CMYK values, as well as the volume level, shown as Figure 7. After the releasing process, pressing the **Confirm** button will release the same ink again, while pressing the **Reset** will end the releasing state and turn to Reset state.

5. By long pressing the **Reset** button, the system will be forced into the Reset state.

4 Hardware Design

4.1 Components

To achieve above mentions functions, we choose these components: We should have a color sensor to pick color. We should have a screen to preview and adjust the picked color. We should have electromagnetic valves to control the flow of ink. Relays and 12V power supply are then necessary. To adjust the amount of ink, we need a potentiometer. Finally, all kinds of wires, buttons, and ink are needed. We also need a 3-D printed ink container. We then generate the BOM as shown in Table 1.

Name	Description	Quantity	Source
TCS34725 Color Sensor	A widely used color sensor	1	Taobao
TFT LCD	A SPI driven 16-bit screen	1	Taobao
Electromagnetic Valves	Can control fluids	4	Taobao
4-Way Relays	Can control high voltage circuit	1	Taobao
12V Power Supply	Can provide 12V 6A power	1	Taobao
CMYK Ink	Ink	1	Taobao
Potentiometer	An adjustable resistor	1	JI Lab
Ink Container	A 3-D printed container	1	SJTU 3-D Printing Center
Wires, Buttons	-	-	-
Simple Circuit Board	-	-	-

Table 1: BOM of the Product

4.2 Block Diagram

After the decision of the necessary components for out product, we can then design the interaction among these components. Therefore, we have the block diagram of out product, which is shown in Fig. 8.

4.3 Product Structure

We then design the physical structure of out product. The color extractor include four layers. Each of them has distinguished functions and will be assembled individually. All layers except from the Ink layer will be assembled using acrylic board. Ink layer will be 3D printed.

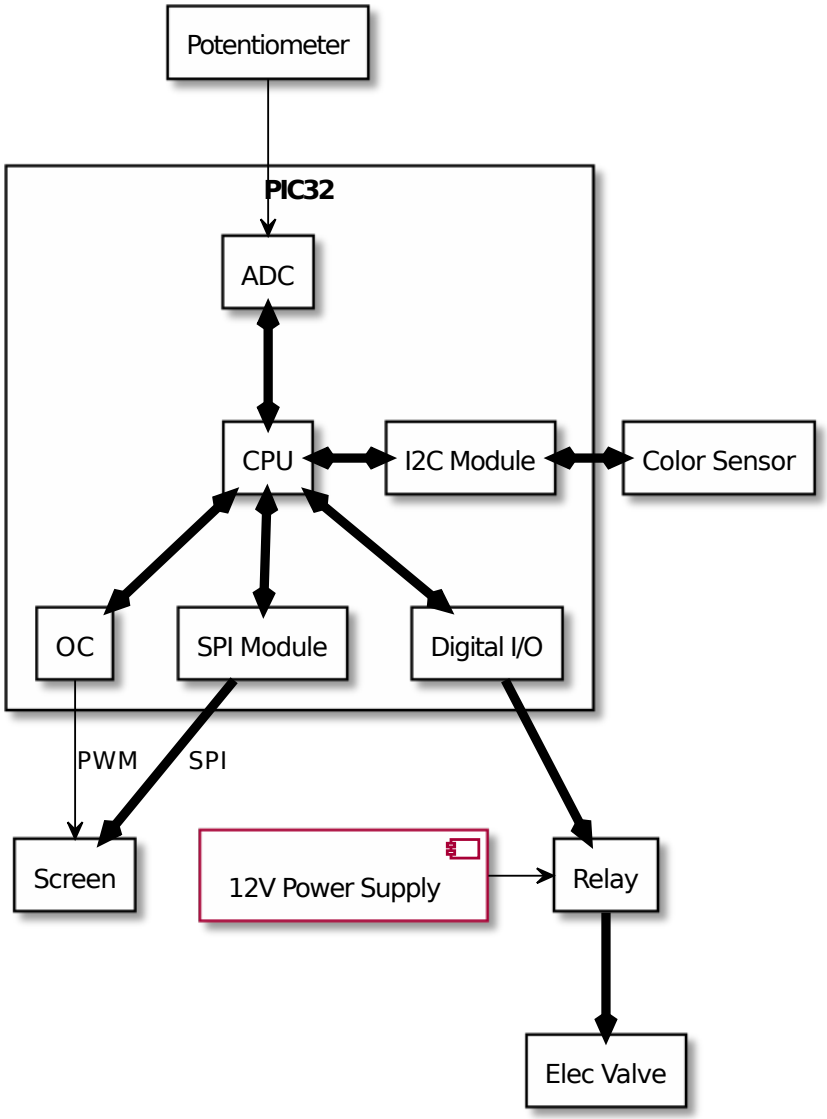


Figure 8: Block Diagram of the Product

4.3.1 Ink layer

Ink layer contains four isolated blocks. Each block will be used to hold around 100ml CMYK ink. There will be a 20mm diameter hole at the bottom of each block allowing ink to flow through tubes toward the solenoid valve. The lid can be slide into the notch on each sides of the ink box to seal the ink box.

4.3.2 Valve layer

Four solenoid valves will be installed on each sides of the inner surface, with one end connected to the ink block and the other end to the container on the stirring plate.

4.3.3 Stir layer

Stir layer has one panel removed so that the container containing the mixed ink can be taken out easily.

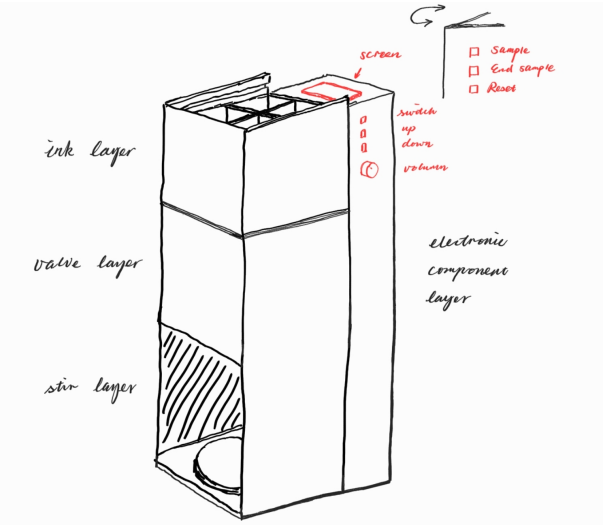


Figure 9: Prototype of the Product

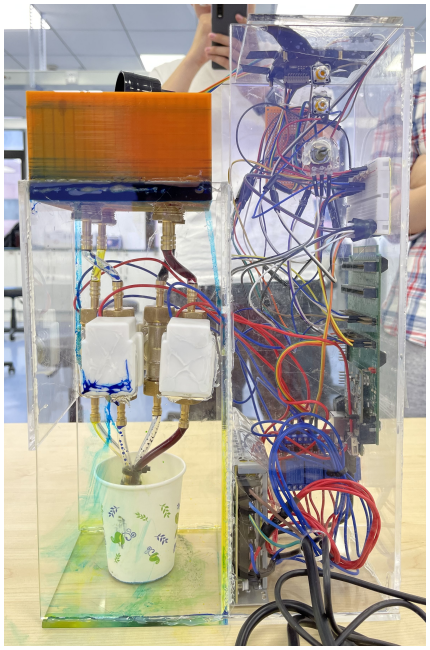


Figure 10: Real Product

4.3.4 Electronic component layer

This layer is separated from other parts to prevent ink from coming in. All electronic components except the solenoid valves will be installed here. There will be a screen at the top displaying the current color RGB value. Five buttons and a knob will be used to adjust RGB colors and ink volume and control the color sensor. The color sensor will be connected to the board using wires so that it can be pulled out to sense color from other objects.

5 Testing

1. The color sensor is easily influenced by the ambient lights. Two options can be made to solve this problem. The first one is to measure the ambient lights and make color balance adjustment, and then measure the RGB values of the target. However, such solution was objected because there would be too much uncertainty caused by the ever-changing circumstance. The solution we applied was to make a **light-tight shield** that formed a sealing environment for the color sensor, so that it can always measure the target color only lightened by its own LED.

2. The measurement of the color sensor is quite inconsistent. On account of the color extractor's function, we require a high precision of the sensor. Therefore, **multiple linear regression (MLR)** was applied to solve the relation between measured RGB values and real ones. The data set (colors) we used for regression is shown in Figure 11. In most cases, the MLR results enable the sensor to provide acceptable RGB values.

3. Despite using MLR, it still couldn't work perfectly for dark colors in our testing. There. might be two reasons. The first one is that the LED on the color

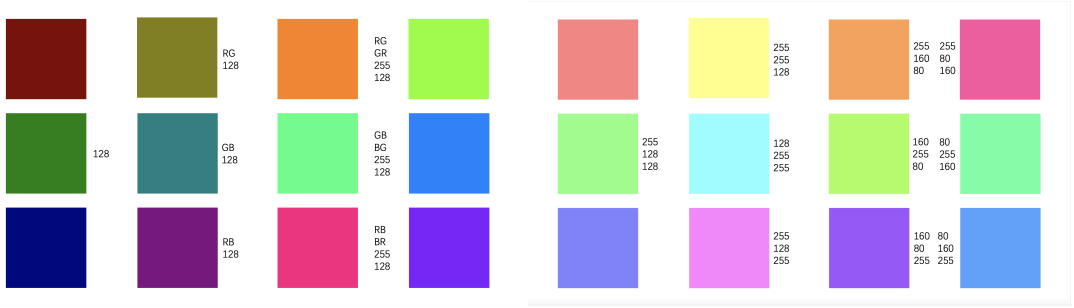


Figure 11: Data Set for MLR

sensor produces a warm white light, which means the amount of blue light is relatively low. By checking the document of the corresponding LED (45-21/LK2C-B38452C4CB2/2T), it can be confirmed that the lights of 400-500nm wavelength has relatively low intensity for warm white light, which is shown in Figure 12. The second reason is that, the relative responsivity of blue light is low for the color sensor, shown by Figure 13. As the sensor determines the intensity of RGB lights by measuring certain waves passing the filters, it should have more blue light with wavelength between 400-500nm produced by the LED. The two reasons together cause the sensor to perform imprecisely for dark colors.

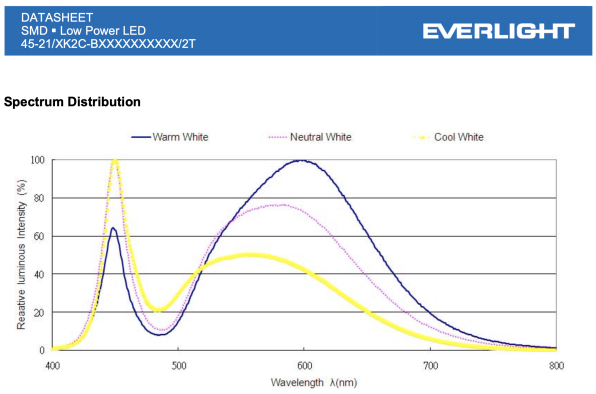


Figure 12: Spectrum Distribution of LED

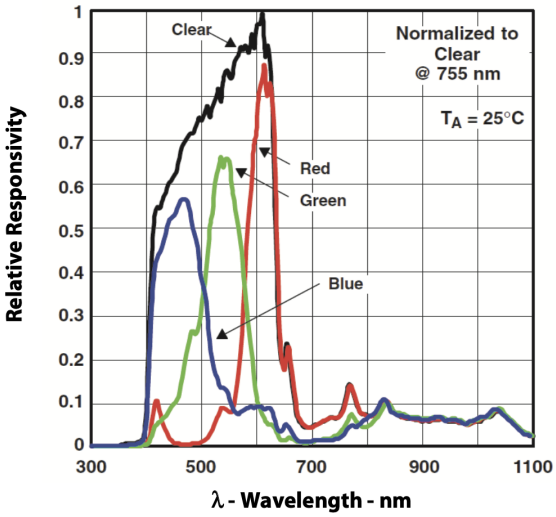


Figure 13: RPhotodiode Spectral Responsivity RGBC

6 Conclusion and Further Improvements

6.1 Conclusion

In general, we have successfully completed and demonstrated the project the way we initially designed. The embedded system of Color Extractor is able to measure colors from real life and the user is also able to make adjustment to the measured color. The system can convert the RGB values to CMYK values, and release corresponding proportion of CMYK ink of user-selected volume that can mix to get the target color. As demonstrated in JI lab, the system works properly.

6.2 Further Improvements

The following are the improvements we can make if more time is given:

1. Change the LED lights on the color sensor, or add a blue light to it. By doing this way, the color sensor can have more precise measurement.

2. Enable the dilution. Now the color can only be manually diluted to get the target color. If more time is given, we are planning to add a container that contains water and add a dilution function to the software design. By doing so, the mixture of ink and water released will directly be the target color.

3. Update the material used for 3-D printing. The container we are using now is not 100% waterproof. As you can figure out from the pictures that the ink will seeping through the container slowly. This problem can be easily improved by changing the material of 3-D printing.

Appendices

ADC.c

```c
#include <xc.h>
#include "utils.h"

//#pragma interrupt adcISR ipl2 vector 27 // 27 for ADC1

int adcVal;

//void adcISR(){
//
//    IFS1bits.AD1IF = 0;
//}

void init_ADC(){
    TRISBSET = 1;

    AD1CON1 = 0x04E4;
    AD1CON2 = 0x0002;
    AD1CON3 = 0x0818;
    AD1CHS = 0x00000000;
    AD1PCFG = 0xFFFE;
    /*
     * ADCS=24
     * sample = 8
     * convert = 12
     * 8 buffer mode
     * SMPI = 0
     */

    //IPC6bits.AD1IP = 2;
    //IPC6bits.AD1IS = 0;
    //IFS1bits.AD1IF = 0;
    //IEC1bits.AD1IE = 1;

}

int read_ADC_val() {
    AD1CON1SET = 0x8000;
    while (!IFS1bits.AD1IF){
        if(AD1CON2bits.BUFS == 1){
            adcVal = ADC1BUF0;
        } else {
            adcVal = ADC1BUF8;
        }
    }
    AD1CON1CLR = 0x8000;
    IFS1bits.AD1IF = 0;
    return adcVal;
```

```
}
```

ADC.h

```
/*
 * File:   ADC.h
 * Author: Lin
 *
 * Created on 2021?8?2?, ??11:30
 */

#ifndef ADC_H
#define        ADC_H

#ifdef        __cplusplus
extern "C" {
#endif

    void init_ADC();

    void start_ADC();

    void stop_ADC();

    int read_ADC_val();

#ifdef        __cplusplus
}
#endif

#endif        /* ADC_H */
```

GUI.c

```
#include "GUI.h"

void init_LCD(){
    DEV_Pin_Init();
    DEV_SPI_Init();
    asm("ei");
    LCD_Init();
    LCD_Clear(0x0000);
}

void draw_color_block(const uint16_t x_start, const uint16_t y_start,
    const uint16_t x_end, const uint16_t y_end, const uint16_t color,
    const int frame){
    uint16_t xls = LCD_WIDTH - y_end - 1;
    uint16_t yls = x_start;
    uint16_t xle = LCD_WIDTH - y_start - 1;
    uint16_t yle = x_end;
```

```
    uint16_t tmp;
    if (!frame){
        LCD_ClearWindow(xls, yls, xle, yle, color);
        return;
    }
    for (tmp = xls; tmp <= xle; tmp ++) {
        LCD_DrawPaint(tmp, yls, 0xFFFF);
        LCD_DrawPaint(tmp, yls + 1, 0xFFFF);
        LCD_DrawPaint(tmp, yle, 0xFFFF);
        LCD_DrawPaint(tmp, yle - 1, 0xFFFF);
    }
    for (tmp = yls; tmp <= yle; tmp ++){
        LCD_DrawPaint(xls, tmp, 0xFFFF);
        LCD_DrawPaint(xls + 1, tmp, 0xFFFF);
        LCD_DrawPaint(xle, tmp, 0xFFFF);
        LCD_DrawPaint(xle - 1, tmp, 0xFFFF);
    }

    LCD_ClearWindow(xls + 5, yls + 5, xle - 3, yle - 3, color);
}

void set_LCD_brightness(const uint16_t val){
    OC5RS = val * 4;
}

void clear_LCD(const uint16_t color){
    LCD_Clear(color);
}

void draw_interface(const uint16_t R, const uint16_t G,
    const uint16_t B, const uint16_t inkLevel){
    char str[100];
    write_str(5, 35, "R:", 0xF800);
    sprintf(str, "%d", R);
    write_str(45, 35, str, 0xF800);
    write_str(5, 65, "G:", 0x07E0);
    sprintf(str, "%d", G);
    write_str(45, 65, str, 0x07E0);
    write_str(5, 95, "B:", 0x001F);
    sprintf(str, "%d", B);
    write_str(45, 95, str, 0x001F);
    write_str(5, 125, "Ink:", 0xFFFF);
    sprintf(str, "%d", inkLevel);
    write_str(85, 125, str, 0xFFFF);
    draw_color_block(200, 80, 280, 160, ((R>>3)<<11) +
        ((G>>2)<<5) + ((B>>3)), 1);
}

void write_val(const uint16_t x, const uint16_t y, const uint8_t val) {
```

```
    if (x > LCD_WIDTH || y > LCD_HEIGHT) {
        return;
    }
    uint16_t fw = Font20.Width;
    uint8_t n2 = (val / 100);
    uint8_t n1 = (val - n2 * 100) / 10;
    uint8_t n0 = (val - n2 * 100 - n1 * 10);
    write_char(x, y, n2 + '0', 0xFFFF);
    write_char(x + fw, y, n1 + '0', 0xFFFF);
    write_char(x + 2 * fw, y, n0 + '0', 0xFFFF);
    return;
}

void write_str(const uint16_t x, const uint16_t y, const char *str,
    const uint16_t color) {
    if (x > LCD_WIDTH || y > LCD_HEIGHT) {
        return;
    }
    uint16_t fw = Font20.Width;
    int pos = 0;
    for (pos = 0; pos < strlen(str); pos ++){
        write_char(x + pos * fw, y, str[pos], color);
    }
    return;
}

void write_char(const uint16_t x, const uint16_t y, const char val,
    const uint16_t color) {
    const uint8_t *ft = Font20.table;
    uint16_t fw = 16;
    uint16_t fh = 20;
    uint32_t s_addr = (val - 32) * 2 * 20;
    uint16_t yl = 0;
    for (yl = 0; yl < fh; yl ++) {
        uint16_t xl = 0;
        for (xl = 0; xl < fw; xl ++) {
            if (ft[s_addr + xl / 8] & (0x80 >> (xl % 8))) {
                LCD_DrawPaint(LCD_WIDTH - y - yl, x + xl, color);
            }
            else {
                LCD_DrawPaint(LCD_WIDTH - y - yl, x + xl, 0x0000);
            }
        }
        s_addr += 2;
    }
}
```

GUI.h

```c
/*
 * File:   GUI.h
 * Author: Lin
 *
 * Created on 2021?7?30?, ??11:15
 */

#ifndef GUI_H
#define        GUI_H

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "lcd_driver.h"
#include "dev_config.h"
#include "font.h"

void init_LCD();

void draw_interface(const uint16_t R, const uint16_t G,
    const uint16_t B, const uint16_t inkLevel);

void set_LCD_brightness(const uint16_t val);

void draw_color_block(const uint16_t x_start, const uint16_t y_start,
    const uint16_t x_end, const uint16_t y_end, const uint16_t color, int frame);

void clear_LCD(const uint16_t color);

void write_str(const uint16_t x, const uint16_t y, const char *str,
    const UWORD color);

void write_val(const uint16_t x, const uint16_t y, const uint8_t val);

#endif        /* GUI_H */
```

color_sensor.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "color_sensor.h"
#include "utils.h"
#include "dev_config.h"
#include "lcd_driver.h"

#define SENSOR_ADDR 0x29
```

```
#define R_CO_C 0.02465
#define R_CO_R 0.00465
#define R_CO_G -0.0611
#define R_CO_B -0.0128
#define R_INTE -11.911
#define G_CO_C -0.2058
#define G_CO_R 0.18774
#define G_CO_G 0.31938
#define G_CO_B 0.15645
#define G_INTE -42.349
#define B_CO_C 0.08618
#define B_CO_R -0.0927
#define B_CO_G -0.1753
#define B_CO_B 0.10713
#define B_INTE 57.202

void wait4Nms(int n){
    int pr4Backup = PR4;
    int t4conBackup = T4CON;
    T4CON = 0x0040;
    TMR4 = 0x0;
    PR4 = 4999;
    T4CONSET = 0x8000;
    int i;
    for(i = 0; i < n; i++){
        while(!IFS0bits.T4IF);
        IFS0bits.T4IF = 0;
    }
    T4CON = 0;
    TMR4 = 0;
    PR4 = pr4Backup;
    T4CON = t4conBackup;
}

uint8_t readOneByte(uint8_t addr){
    // PBCLK = 20MHz
    I2C2BRG = 0x060;
    // Baud Rate 100kHz
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.ON = 1;
    I2C2CONbits.SEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = SENSOR_ADDR << 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = 0b10000000 | addr;
```

```
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.RSEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = (SENSOR_ADDR << 1) | 0b1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.RCEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.ACKDT = 1;
    I2C2CONbits.ACKEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.PEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    if(I2C2STATbits.RBF == 1){
        return I2C2RCV;
    }

    return 0xFF;
}

void readNBytes(uint8_t startAddr, int n, uint8_t* result){
    // PBCLK = 20MHz
    I2C2BRG = 0x060;
    // Baud Rate 100kHz
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.ON = 1;
    I2C2CONbits.SEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = SENSOR_ADDR << 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = 0b10100000 | startAddr;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;
```

```
    I2C2CONbits.RSEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = (SENSOR_ADDR << 1) | 0b1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    int i;
    for(i = 0; i < n - 1; i++){
        I2C2CONbits.RCEN = 1;
        while(!IFS1bits.I2C2MIF);
        IFS1bits.I2C2MIF = 0;

        I2C2CONbits.ACKDT = 0;
        I2C2CONbits.ACKEN = 1;
        result[i] = I2C2RCV;
        while(!IFS1bits.I2C2MIF);
        IFS1bits.I2C2MIF = 0;
    }
    I2C2CONbits.RCEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.ACKDT = 1;
    I2C2CONbits.ACKEN = 1;
    result[n - 1] = I2C2RCV;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.PEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;
}

void writeOneByte(uint8_t addr, uint8_t value){
    // PBCLK = 20MHz
    I2C2BRG = 0x060;
    // Baud Rate 100kHz
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.ON = 1;
    I2C2CONbits.SEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = SENSOR_ADDR << 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;
```

```
    I2C2TRN = 0b10000000 | addr;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2TRN = value;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;

    I2C2CONbits.PEN = 1;
    while(!IFS1bits.I2C2MIF);
    IFS1bits.I2C2MIF = 0;
}

void getRGB(int* R, int* G, int* B){
    writeOneByte(0x01, 0xF6);
    writeOneByte(0x03, 0xAB);
    writeOneByte(0x0F, 0x03);
    writeOneByte(0x00, 0b00001011);
    delay_ms(40);
    uint8_t bytes[9];
    readNBytes(0x13, 9, bytes);
    writeOneByte(0x01, 0b00000000);
    int cValue, rValue, gValue, bValue;
    cValue = (bytes[2] << 8) + bytes[1];
    rValue = (bytes[4] << 8) + bytes[3];
    gValue = (bytes[6] << 8) + bytes[5];
    bValue = (bytes[8] << 8) + bytes[7];
    double rDouble, gDouble, bDouble;
    rDouble = R_CO_C * cValue + R_CO_R * rValue +
        R_CO_G * gValue + R_CO_B * bValue + R_INTE;
    gDouble = G_CO_C * cValue + G_CO_R * rValue +
        G_CO_G * gValue + G_CO_B * bValue + G_INTE;
    bDouble = B_CO_C * cValue + B_CO_R * rValue +
        B_CO_G * gValue + B_CO_B * bValue + B_INTE;
    if(rDouble < 0){
        *R = 0;
    } else if(rDouble > 255){
        *R = 255;
    } else {
        *R = (int)rDouble;
    }
    if(gDouble < 0){
        *G = 0;
    } else if(gDouble > 255){
        *G = 255;
    } else {
        *G = (int)gDouble;
    }
    if(bDouble < 0){
        *B = 0;
```

```
    } else if(bDouble > 255){
        *B = 255;
    } else {
        *B = (int)bDouble;
    }
}
```

color_sensor.h

```
/*
 * File:   color_sensor.h
 * Author: citrate
 *
 * Created on 2021?7?27?, ??8:58
 */

#ifndef COLOR_SENSOR_H
#define        COLOR_SENSOR_H
#include <xc.h>

void getRGB(int* R, int* G, int* B);
// Blocking.
#endif        /* COLOR_SENSOR_H */
```

dev_config.c

```
#include "dev_config.h"



void initTimerAndOC() {
    T2CON = 0x0000; // prescale = 1
    PR2 = 0x1387; // period = 5000
    TMR2 = 0;

    OC5CON = 0x0000;
    OC5R = 0x0;
    OC5RS = 0x0;
    OC5CON = 0x0006;
    T2CONSET = 0x8000;
    OC5CONSET = 0x8000;
}

void DEV_SPI_Init(){
    int rData;
    IEC0CLR=0x03800000;
    SPI1CON = 0x0;
    rData=SPI1BUF;
    SPI1BRG=0x0;
    SPI1STATCLR=0x40;
    SPI1CONbits.MSTEN = 1;
```

```
    SPI1CONbits.CKP = 0;
    SPI1CONbits.SMP = 0;
    //SPI1CONbits.DISSDO = 1;
    SPI1CONbits.CKE = 1;
    SPI1CONbits.ON = 1;

}

void DEV_Pin_Init() {
    INTCONbits.MVEC = 1;
    //TRISDCLR = 0x0E;
    TRISDCLR = 0xFF;
    //LATDCLR = 0x0E;
    LATDCLR = 0xFF;

    delay_init();

    initTimerAndOC();

    OC5RS = 4000;


}

void DEV_Digital_Write(int pin, UBYTE value) {
    if (value)
        LATDSET = 1 << (pin + 1);
    else
        LATDCLR = 1 << (pin + 1);
}

void DEV_SPI_WRITE(UBYTE dat) {
    UBYTE tmp;
    SPI1BUF = dat;
    //DEV_Delay_us(1);
    while(SPI1STATbits.SPITBF){}
    tmp = SPI1BUF;

}

void DEV_Delay_ms(UBYTE xms) {
    delay_ms(xms);
}


void DEV_Delay_us(UBYTE xus) {
    delay_us(xus);
}
```

dev_config.h

```
/*
 * File:   dev_config.h
 * Author: Lin
 *
 * Created on 2021?7?20?, ??4:56
 */



#ifndef DEV_CONFIG_H
#define        DEV_CONFIG_H

#include "p32xxxx.h"

#define UBYTE   uint8_t
#define UWORD   uint16_t
#define UDOUBLE uint32_t

/*
 * DEV_CLK_PIN // ???
 *
 */

#define DEV_CS_PIN  0 //LATDbits.LATD1 // control signal
#define DEV_DC_PIN  1 //LATDbits.LATD2 // command:0 data:1
#define DEV_RST_PIN 2 //LATDbits.LATD3 // reset, put 1
//#define DEV_BL_PIN  LATDbits.LATD4 // backlight

void DEV_SPI_Init(void);

void DEV_Pin_Init(void);

void DEV_Digital_Write(int pin, UBYTE value);
//void DEV_Digital_Read(UBYTE pin);

/**
 * SPI
**/
void DEV_SPI_WRITE(UBYTE dat);

/**
 * delay x ms
**/
void DEV_Delay_ms(UBYTE xms);

void DEV_Delay_us(UBYTE xus);

/**
 * PWM_BL
```

```
**/
//void DEV_Set_PWM(_Value)  analogWrite(DEV_BL_PIN, _Value)


#endif        /* DEV_CONFIG_H */
```

font.c

```
#include "font.h"

const uint8_t Font20_Table[] = {
    // @0 ' ' (14 pixels wide)
  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, //

  // @40 '!' (14 pixels wide)
  0x00, 0x00, //
  0x07, 0x00, //      ###
  0x07, 0x00, //      ###
  0x07, 0x00, //      ###
  0x07, 0x00, //      ###
  0x07, 0x00, //      ###
  0x07, 0x00, //      ###
  0x07, 0x00, //      ###
  0x02, 0x00, //       #
  0x02, 0x00, //       #
  0x00, 0x00, //
  0x00, 0x00, //
  0x07, 0x00, //      ###
  0x07, 0x00, //      ###
  0x00, 0x00, //
  0x00, 0x00, //
```

```
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @80 '"' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x1C, 0xE0, //    ###  ###
0x1C, 0xE0, //    ###  ###
0x1C, 0xE0, //    ###  ###
0x08, 0x40, //     #    #
0x08, 0x40, //     #    #
0x08, 0x40, //     #    #
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @120 '#' (14 pixels wide)
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x0C, 0xC0, //     ##  ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @160 '£' (14 pixels wide)
```

```
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x07, 0xE0, //      ######
0x0F, 0xE0, //     #######
0x18, 0x60, //    ##    ##
0x18, 0x00, //    ##
0x1F, 0x00, //    #####
0x0F, 0xC0, //     ######
0x00, 0xE0, //         ###
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
0x1F, 0xC0, //    #######
0x1F, 0x80, //    ######
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @200 '%' (14 pixels wide)
0x00, 0x00, //
0x1C, 0x00, //    ###
0x22, 0x00, //   #   #
0x22, 0x00, //   #   #
0x22, 0x00, //   #   #
0x1C, 0x60, //    ###   ##
0x01, 0xE0, //        ####
0x0F, 0x80, //     #####
0x3C, 0x00, //   ####
0x31, 0xC0, //   ##   ###
0x02, 0x20, //       #   #
0x02, 0x20, //       #   #
0x02, 0x20, //       #   #
0x01, 0xC0, //        ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @240 '&' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x03, 0xE0, //       #####
0x0F, 0xE0, //     #######
0x0C, 0x00, //     ##
```

```
    0x0C, 0x00, //     ##
    0x06, 0x00, //      ##
    0x0F, 0x30, //     ####  ##
    0x1F, 0xF0, //    #########
    0x19, 0xE0, //    ##  ####
    0x18, 0xC0, //    ##   ##
    0x1F, 0xF0, //    #########
    0x07, 0xB0, //      #### ##
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @280 ''' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x01, 0x00, //        #
    0x01, 0x00, //        #
    0x01, 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, //

    // @320 '(' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0xC0, //         ##
    0x00, 0xC0, //         ##
    0x01, 0x80, //        ##
    0x01, 0x80, //        ##
    0x01, 0x80, //        ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
```

```
0x01, 0x80, //        ##
0x01, 0x80, //        ##
0x01, 0x80, //        ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @360 ')' (14 pixels wide)
0x00, 0x00, //
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @400 '*' (14 pixels wide)
0x00, 0x00, //
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x1B, 0x60, //    ## ## ##
0x1F, 0xE0, //    ########
0x07, 0x80, //      ####
0x07, 0x80, //      ####
0x0F, 0xC0, //     ######
0x0C, 0xC0, //     ##  ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
```

```
0x00, 0x00, //
0x00, 0x00, //

// @440 '+' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @480 ',' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x03, 0x80, //       ###
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x04, 0x00, //      #
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @520 '-' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
```

```
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x3F, 0xE0, //   #########
    0x3F, 0xE0, //   #########
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @560 '.' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @600 '/' (14 pixels wide)
    0x00, 0x60, //          ##
    0x00, 0x60, //          ##
    0x00, 0xC0, //         ##
    0x00, 0xC0, //         ##
    0x00, 0xC0, //         ##
    0x01, 0x80, //        ##
    0x01, 0x80, //        ##
    0x03, 0x00, //       ##
```

```
0x03, 0x00, //       ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x18, 0x00, //    ##
0x18, 0x00, //    ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @640 '0' (14 pixels wide)
0x00, 0x00, //
0x0F, 0x80, //     #####
0x1F, 0xC0, //    #######
0x18, 0xC0, //    ##   ##
0x30, 0x60, //   ##     ##
0x30, 0x60, //   ##     ##
0x30, 0x60, //   ##     ##
0x30, 0x60, //   ##     ##
0x30, 0x60, //   ##     ##
0x30, 0x60, //   ##     ##
0x30, 0x60, //   ##     ##
0x18, 0xC0, //    ##   ##
0x1F, 0xC0, //    #######
0x0F, 0x80, //     #####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @680 '1' (14 pixels wide)
0x00, 0x00, //
0x03, 0x00, //       ##
0x1F, 0x00, //    #####
0x1F, 0x00, //    #####
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x1F, 0xE0, //    ########
0x1F, 0xE0, //    ########
```

```
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @720 '2' (14 pixels wide)
0x00, 0x00, //
0x0F, 0x80, //     #####
0x1F, 0xC0, //    #######
0x38, 0xE0, //   ###   ###
0x30, 0x60, //   ##     ##
0x00, 0x60, //          ##
0x00, 0xC0, //         ##
0x01, 0x80, //        ##
0x03, 0x00, //       ##
0x06, 0x00, //      ##
0x0C, 0x00, //     ##
0x18, 0x00, //    ##
0x3F, 0xE0, //   #########
0x3F, 0xE0, //   #########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @760 '3' (14 pixels wide)
0x00, 0x00, //
0x0F, 0x80, //     #####
0x3F, 0xC0, //   ########
0x30, 0xE0, //   ##    ###
0x00, 0x60, //          ##
0x00, 0xE0, //         ###
0x07, 0xC0, //      #####
0x07, 0xC0, //      #####
0x00, 0xE0, //         ###
0x00, 0x60, //          ##
0x00, 0x60, //          ##
0x60, 0xE0, //  ##     ###
0x7F, 0xC0, //  #########
0x3F, 0x80, //   #######
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
```

```
// @800 '4' (14 pixels wide)
0x00, 0x00, //
0x01, 0xC0, //         ###
0x03, 0xC0, //        ####
0x03, 0xC0, //        ####
0x06, 0xC0, //       ## ##
0x0C, 0xC0, //      ##  ##
0x0C, 0xC0, //      ##  ##
0x18, 0xC0, //     ##   ##
0x30, 0xC0, //    ##    ##
0x3F, 0xE0, //    #########
0x3F, 0xE0, //    #########
0x00, 0xC0, //          ##
0x03, 0xE0, //        #####
0x03, 0xE0, //        #####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @840 '5' (14 pixels wide)
0x00, 0x00, //
0x1F, 0xC0, //     #######
0x1F, 0xC0, //     #######
0x18, 0x00, //     ##
0x18, 0x00, //     ##
0x1F, 0x80, //     ######
0x1F, 0xC0, //     #######
0x18, 0xE0, //     ##   ###
0x00, 0x60, //          ##
0x00, 0x60, //          ##
0x00, 0x60, //          ##
0x30, 0xE0, //    ##    ###
0x3F, 0xC0, //    ########
0x1F, 0x80, //     ######
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @880 '6' (14 pixels wide)
0x00, 0x00, //
0x03, 0xE0, //        #####
0x0F, 0xE0, //      #######
0x1E, 0x00, //     ####
```

```
0x18, 0x00, //    ##
0x38, 0x00, //   ###
0x37, 0x80, //   ## ####
0x3F, 0xC0, //   ########
0x38, 0xE0, //   ###   ###
0x30, 0x60, //   ##     ##
0x30, 0x60, //   ##     ##
0x18, 0xE0, //    ##   ###
0x1F, 0xC0, //    #######
0x07, 0x80, //      ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @920 '7' (14 pixels wide)
0x00, 0x00, //
0x3F, 0xE0, //   #########
0x3F, 0xE0, //   #########
0x30, 0x60, //   ##     ##
0x00, 0x60, //          ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x01, 0x80, //        ##
0x01, 0x80, //        ##
0x01, 0x80, //        ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @960 '8' (14 pixels wide)
0x00, 0x00, //
0x0F, 0x80, //     #####
0x1F, 0xC0, //    #######
0x38, 0xE0, //   ###   ###
0x30, 0x60, //   ##     ##
0x38, 0xE0, //   ###   ###
0x1F, 0xC0, //    #######
0x1F, 0xC0, //    #######
0x38, 0xE0, //   ###   ###
0x30, 0x60, //   ##     ##
```

```
    0x30, 0x60, //   ##     ##
    0x38, 0xE0, //   ###   ###
    0x1F, 0xC0, //    #######
    0x0F, 0x80, //     #####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1000 '9' (14 pixels wide)
    0x00, 0x00, //
    0x0F, 0x00, //     ####
    0x1F, 0xC0, //    #######
    0x38, 0xC0, //   ###   ##
    0x30, 0x60, //   ##     ##
    0x30, 0x60, //   ##     ##
    0x38, 0xE0, //   ###   ###
    0x1F, 0xE0, //    ########
    0x0F, 0x60, //     #### ##
    0x00, 0xE0, //         ###
    0x00, 0xC0, //         ##
    0x03, 0xC0, //       ####
    0x3F, 0x80, //   #######
    0x3E, 0x00, //   #####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1040 ':' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x03, 0x80, //       ###
    0x00, 0x00, //
    0x00, 0x00, //
```

```
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1080 ';' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x01, 0xC0, //         ###
    0x01, 0xC0, //         ###
    0x01, 0xC0, //         ###
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x03, 0x80, //        ###
    0x03, 0x00, //        ##
    0x06, 0x00, //       ##
    0x06, 0x00, //       ##
    0x04, 0x00, //       #
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1120 '<' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x30, //           ##
    0x00, 0xF0, //         ####
    0x03, 0xC0, //        ####
    0x07, 0x00, //       ###
    0x1C, 0x00, //    ###
    0x78, 0x00, //  ####
    0x1C, 0x00, //    ###
    0x07, 0x00, //       ###
    0x03, 0xC0, //        ####
    0x00, 0xF0, //         ####
    0x00, 0x30, //           ##
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1160 '=' (14 pixels wide)
```

```
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x7F, 0xF0, //  ###########
0x7F, 0xF0, //  ###########
0x00, 0x00, //
0x00, 0x00, //
0x7F, 0xF0, //  ###########
0x7F, 0xF0, //  ###########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1200 '>' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x30, 0x00, //   ##
0x3C, 0x00, //   ####
0x0F, 0x00, //     ####
0x03, 0x80, //       ###
0x00, 0xE0, //         ###
0x00, 0x78, //          ####
0x00, 0xE0, //         ###
0x03, 0x80, //       ###
0x0F, 0x00, //     ####
0x3C, 0x00, //   ####
0x30, 0x00, //   ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1240 '?' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x0F, 0x80, //     #####
0x1F, 0xC0, //    #######
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
```

```
    0x00, 0x60, //          ##
    0x01, 0xC0, //        ###
    0x03, 0x80, //       ###
    0x03, 0x00, //       ##
    0x00, 0x00, //
    0x00, 0x00, //
    0x07, 0x00, //      ###
    0x07, 0x00, //      ###
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1280 '@' (14 pixels wide)
    0x00, 0x00, //
    0x03, 0x80, //       ###
    0x0C, 0x80, //     ##  #
    0x08, 0x40, //     #    #
    0x10, 0x40, //    #     #
    0x10, 0x40, //    #     #
    0x11, 0xC0, //    #   ###
    0x12, 0x40, //    #  #  #
    0x12, 0x40, //    #  #  #
    0x12, 0x40, //    #  #  #
    0x11, 0xC0, //    #   ###
    0x10, 0x00, //    #
    0x08, 0x00, //     #
    0x08, 0x40, //     #    #
    0x07, 0x80, //      ####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1320 'A' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x1F, 0x80, //    ######
    0x1F, 0x80, //    ######
    0x03, 0x80, //       ###
    0x06, 0xC0, //      ## ##
    0x06, 0xC0, //      ## ##
    0x0C, 0xC0, //     ##  ##
    0x0C, 0x60, //     ##   ##
    0x1F, 0xE0, //    ########
    0x1F, 0xE0, //    ########
    0x30, 0x30, //   ##      ##
```

```
0x78, 0x78, //  ####    ####
0x78, 0x78, //  ####    ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1360 'B' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x3F, 0x80, //   #######
0x3F, 0xC0, //   ########
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
0x18, 0xE0, //    ##   ###
0x1F, 0xC0, //    #######
0x1F, 0xE0, //    ########
0x18, 0x70, //    ##    ###
0x18, 0x30, //    ##     ##
0x18, 0x30, //    ##     ##
0x3F, 0xF0, //   ##########
0x3F, 0xE0, //   #########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1400 'C' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x07, 0xB0, //      #### ##
0x0F, 0xF0, //     ########
0x1C, 0x70, //    ###   ###
0x38, 0x30, //   ###     ##
0x30, 0x00, //   ##
0x30, 0x00, //   ##
0x30, 0x00, //   ##
0x30, 0x00, //   ##
0x38, 0x30, //   ###     ##
0x1C, 0x70, //    ###   ###
0x0F, 0xE0, //     #######
0x07, 0xC0, //      #####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
```

```
0x00, 0x00, //
0x00, 0x00, //

// @1440 'D' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x7F, 0x80, //  ########
0x7F, 0xC0, //  #########
0x30, 0xE0, //   ##    ###
0x30, 0x70, //   ##     ###
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x70, //   ##     ###
0x30, 0xE0, //   ##    ###
0x7F, 0xC0, //  #########
0x7F, 0x80, //  ########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1480 'E' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x18, 0x30, //    ##     ##
0x18, 0x30, //    ##     ##
0x19, 0x80, //    ##  ##
0x1F, 0x80, //    ######
0x1F, 0x80, //    ######
0x19, 0x80, //    ##  ##
0x18, 0x30, //    ##     ##
0x18, 0x30, //    ##     ##
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1520 'F' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
```

```
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x18, 0x30, //    ##     ##
0x18, 0x30, //    ##     ##
0x19, 0x80, //    ##  ##
0x1F, 0x80, //    ######
0x1F, 0x80, //    ######
0x19, 0x80, //    ##  ##
0x18, 0x00, //    ##
0x18, 0x00, //    ##
0x3F, 0x00, //   ######
0x3F, 0x00, //   ######
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1560 'G' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x07, 0xB0, //      #### ##
0x1F, 0xF0, //    #########
0x18, 0x70, //    ##    ###
0x30, 0x30, //   ##      ##
0x30, 0x00, //   ##
0x30, 0x00, //   ##
0x31, 0xF8, //   ##   ######
0x31, 0xF8, //   ##   ######
0x30, 0x30, //   ##      ##
0x18, 0x30, //    ##     ##
0x1F, 0xF0, //    #########
0x07, 0xC0, //      #####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1600 'H' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x3C, 0xF0, //   ####  ####
0x3C, 0xF0, //   ####  ####
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
0x1F, 0xE0, //    ########
```

```
    0x1F, 0xE0, //    ########
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x3C, 0xF0, //   ####  ####
    0x3C, 0xF0, //   ####  ####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1640 'I' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x1F, 0xE0, //    ########
    0x1F, 0xE0, //    ########
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x1F, 0xE0, //    ########
    0x1F, 0xE0, //    ########
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @1680 'J' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x03, 0xF8, //       #######
    0x03, 0xF8, //       #######
    0x00, 0x60, //          ##
    0x00, 0x60, //          ##
    0x00, 0x60, //          ##
    0x00, 0x60, //          ##
    0x30, 0x60, //   ##     ##
    0x30, 0x60, //   ##     ##
    0x30, 0x60, //   ##     ##
    0x30, 0xE0, //   ##    ###
    0x3F, 0xC0, //   ########
    0x0F, 0x80, //     #####
```

```
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1720 'K' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x3E, 0xF8, //   ##### #####
0x3E, 0xF8, //   ##### #####
0x18, 0xE0, //    ##   ###
0x19, 0x80, //    ##  ##
0x1B, 0x00, //    ## ##
0x1F, 0x00, //    #####
0x1D, 0x80, //    ### ##
0x18, 0xC0, //    ##   ##
0x18, 0xC0, //    ##   ##
0x18, 0x60, //    ##    ##
0x3E, 0x78, //   #####  ####
0x3E, 0x38, //   #####   ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1760 'L' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x3F, 0x00, //   ######
0x3F, 0x00, //   ######
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x0C, 0x30, //     ##    ##
0x0C, 0x30, //     ##    ##
0x0C, 0x30, //     ##    ##
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
```

```
// @1800 'M' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x78, 0x78, //  ####    ####
0x78, 0x78, //  ####    ####
0x38, 0x70, //   ###    ###
0x3C, 0xF0, //   ####  ####
0x34, 0xB0, //   ## #  # ##
0x37, 0xB0, //   ## #### ##
0x37, 0xB0, //   ## #### ##
0x33, 0x30, //   ##  ##  ##
0x33, 0x30, //   ##  ##  ##
0x30, 0x30, //   ##      ##
0x7C, 0xF8, //  #####  #####
0x7C, 0xF8, //  #####  #####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1840 'N' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x39, 0xF0, //   ###  #####
0x3D, 0xF0, //   #### #####
0x1C, 0x60, //    ###   ##
0x1E, 0x60, //    ####  ##
0x1E, 0x60, //    ####  ##
0x1B, 0x60, //    ## ## ##
0x1B, 0x60, //    ## ## ##
0x19, 0xE0, //    ##  ####
0x19, 0xE0, //    ##  ####
0x18, 0xE0, //    ##   ###
0x3E, 0xE0, //   ##### ###
0x3E, 0x60, //   #####  ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1880 'O' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x07, 0x80, //      ####
0x0F, 0xC0, //     ######
```

```
0x1C, 0xE0, //    ###  ###
0x38, 0x70, //   ###    ###
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x38, 0x70, //   ###    ###
0x1C, 0xE0, //    ###  ###
0x0F, 0xC0, //     ######
0x07, 0x80, //      ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1920 'P' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x3F, 0xC0, //   ########
0x3F, 0xE0, //   #########
0x18, 0x70, //    ##    ###
0x18, 0x30, //    ##     ##
0x18, 0x30, //    ##     ##
0x18, 0x70, //    ##    ###
0x1F, 0xE0, //    ########
0x1F, 0xC0, //    #######
0x18, 0x00, //    ##
0x18, 0x00, //    ##
0x3F, 0x00, //   ######
0x3F, 0x00, //   ######
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @1960 'Q' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x07, 0x80, //      ####
0x0F, 0xC0, //     ######
0x1C, 0xE0, //    ###  ###
0x38, 0x70, //   ###    ###
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
```

```
0x38, 0x70, //   ###    ###
0x1C, 0xE0, //    ###  ###
0x0F, 0xC0, //     ######
0x07, 0x80, //      ####
0x07, 0xB0, //      #### ##
0x0F, 0xF0, //     ########
0x0C, 0xE0, //     ##  ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2000 'R' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x3F, 0xC0, //   ########
0x3F, 0xE0, //   #########
0x18, 0x70, //    ##    ###
0x18, 0x30, //    ##     ##
0x18, 0x70, //    ##    ###
0x1F, 0xE0, //    ########
0x1F, 0xC0, //    #######
0x18, 0xE0, //    ##   ###
0x18, 0x60, //    ##    ##
0x18, 0x70, //    ##    ###
0x3E, 0x38, //   #####   ###
0x3E, 0x18, //   #####    ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2040 'S' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x0F, 0xB0, //     ##### ##
0x1F, 0xF0, //    #########
0x38, 0x70, //   ###    ###
0x30, 0x30, //   ##      ##
0x38, 0x00, //   ###
0x1F, 0x80, //    ######
0x07, 0xE0, //      ######
0x00, 0x70, //         ###
0x30, 0x30, //   ##      ##
0x38, 0x70, //   ###    ###
0x3F, 0xE0, //   #########
0x37, 0xC0, //   ## #####
0x00, 0x00, //
0x00, 0x00, //
```

```
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2080 'T' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x3F, 0xF0, //   ##########
    0x3F, 0xF0, //   ##########
    0x33, 0x30, //   ##  ##  ##
    0x33, 0x30, //   ##  ##  ##
    0x33, 0x30, //   ##  ##  ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x0F, 0xC0, //     ######
    0x0F, 0xC0, //     ######
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2120 'U' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x3C, 0xF0, //   ####  ####
    0x3C, 0xF0, //   ####  ####
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x1C, 0xE0, //    ###  ###
    0x0F, 0xC0, //     ######
    0x07, 0x80, //      ####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2160 'V' (14 pixels wide)
```

```
    0x00, 0x00, //
    0x00, 0x00, //
    0x78, 0xF0, //  ####   ####
    0x78, 0xF0, //  ####   ####
    0x30, 0x60, //   ##     ##
    0x30, 0x60, //   ##     ##
    0x18, 0xC0, //    ##   ##
    0x18, 0xC0, //    ##   ##
    0x0D, 0x80, //     ## ##
    0x0D, 0x80, //     ## ##
    0x0D, 0x80, //     ## ##
    0x07, 0x00, //      ###
    0x07, 0x00, //      ###
    0x07, 0x00, //      ###
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2200 'W' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x7C, 0x7C, //  #####   #####
    0x7C, 0x7C, //  #####   #####
    0x30, 0x18, //   ##       ##
    0x33, 0x98, //   ##  ###  ##
    0x33, 0x98, //   ##  ###  ##
    0x33, 0x98, //   ##  ###  ##
    0x36, 0xD8, //   ## ## ## ##
    0x16, 0xD0, //    # ## ## #
    0x1C, 0x70, //    ###   ###
    0x1C, 0x70, //    ###   ###
    0x1C, 0x70, //    ###   ###
    0x18, 0x30, //    ##     ##
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2240 'X' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x78, 0xF0, //  ####   ####
    0x78, 0xF0, //  ####   ####
    0x30, 0x60, //   ##     ##
    0x18, 0xC0, //    ##   ##
```

```
    0x0D, 0x80, //     ## ##
    0x07, 0x00, //      ###
    0x07, 0x00, //      ###
    0x0D, 0x80, //     ## ##
    0x18, 0xC0, //    ##   ##
    0x30, 0x60, //   ##     ##
    0x78, 0xF0, //  ####   ####
    0x78, 0xF0, //  ####   ####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2280 'Y' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x3C, 0xF0, //   ####  ####
    0x3C, 0xF0, //   ####  ####
    0x18, 0x60, //    ##    ##
    0x0C, 0xC0, //     ##  ##
    0x07, 0x80, //      ####
    0x07, 0x80, //      ####
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x0F, 0xC0, //     ######
    0x0F, 0xC0, //     ######
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2320 'Z' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x1F, 0xE0, //    ########
    0x1F, 0xE0, //    ########
    0x18, 0x60, //    ##    ##
    0x18, 0xC0, //    ##   ##
    0x01, 0x80, //        ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x06, 0x00, //      ##
    0x0C, 0x60, //     ##   ##
    0x18, 0x60, //    ##    ##
```

```
0x1F, 0xE0, //    ########
0x1F, 0xE0, //    ########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2360 '[' (14 pixels wide)
0x00, 0x00, //
0x03, 0xC0, //       ####
0x03, 0xC0, //       ####
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0xC0, //       ####
0x03, 0xC0, //       ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2400 '\' (14 pixels wide)
0x18, 0x00, //    ##
0x18, 0x00, //    ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x0C, 0x00, //     ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x01, 0x80, //        ##
0x01, 0x80, //        ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x00, 0x60, //          ##
0x00, 0x60, //          ##
0x00, 0x00, //
0x00, 0x00, //
```

```
0x00, 0x00, //
0x00, 0x00, //

// @2440 ']' (14 pixels wide)
0x00, 0x00, //
0x0F, 0x00, //     ####
0x0F, 0x00, //     ####
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x0F, 0x00, //     ####
0x0F, 0x00, //     ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2480 '^' (14 pixels wide)
0x00, 0x00, //
0x02, 0x00, //       #
0x07, 0x00, //      ###
0x0D, 0x80, //     ## ##
0x18, 0xC0, //    ##   ##
0x30, 0x60, //   ##     ##
0x20, 0x20, //   #       #
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2520 '_' (14 pixels wide)
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, //
0xFF, 0xFC, // ##############
0xFF, 0xFC, // ##############

// @2560 '`' (14 pixels wide)
0x00, 0x00, //
0x04, 0x00, //      #
0x03, 0x00, //       ##
0x00, 0x80, //         #
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, //

// @2600 'a' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x0F, 0xC0, //     ######
0x1F, 0xE0, //    ########
0x00, 0x60, //          ##
```

```
0x0F, 0xE0, //     #######
0x1F, 0xE0, //    ########
0x38, 0x60, //   ###    ##
0x30, 0xE0, //   ##    ###
0x3F, 0xF0, //   ##########
0x1F, 0x70, //    ##### ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2640 'b' (14 pixels wide)
0x00, 0x00, //
0x70, 0x00, //  ###
0x70, 0x00, //  ###
0x30, 0x00, //   ##
0x30, 0x00, //   ##
0x37, 0x80, //   ## ####
0x3F, 0xE0, //   #########
0x38, 0x60, //   ###    ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x38, 0x60, //   ###    ##
0x7F, 0xE0, //  ##########
0x77, 0x80, //  ### ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2680 'c' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x07, 0xB0, //      #### ##
0x1F, 0xF0, //    #########
0x18, 0x30, //    ##     ##
0x30, 0x30, //   ##      ##
0x30, 0x00, //   ##
0x30, 0x00, //   ##
0x38, 0x30, //   ###     ##
0x1F, 0xF0, //    #########
0x0F, 0xC0, //     ######
```

```
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2720 'd' (14 pixels wide)
0x00, 0x00, //
0x00, 0x70, //          ###
0x00, 0x70, //          ###
0x00, 0x30, //           ##
0x00, 0x30, //           ##
0x07, 0xB0, //      #### ##
0x1F, 0xF0, //    #########
0x18, 0x70, //    ##    ###
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x38, 0x70, //   ###    ###
0x1F, 0xF8, //    ##########
0x07, 0xB8, //      #### ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2760 'e' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x07, 0x80, //      ####
0x1F, 0xE0, //    ########
0x18, 0x60, //    ##    ##
0x3F, 0xF0, //   ##########
0x3F, 0xF0, //   ##########
0x30, 0x00, //   ##
0x18, 0x30, //    ##     ##
0x1F, 0xF0, //    #########
0x07, 0xC0, //      #####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
```

```
// @2800 'f' (14 pixels wide)
0x00, 0x00, //
0x03, 0xF0, //       ######
0x07, 0xF0, //      #######
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x1F, 0xE0, //    ########
0x1F, 0xE0, //    ########
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x1F, 0xE0, //    ########
0x1F, 0xE0, //    ########
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @2840 'g' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x07, 0xB8, //      #### ###
0x1F, 0xF8, //    ##########
0x18, 0x70, //    ##    ###
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x30, 0x30, //   ##      ##
0x18, 0x70, //    ##    ###
0x1F, 0xF0, //    #########
0x07, 0xB0, //      #### ##
0x00, 0x30, //           ##
0x00, 0x70, //          ###
0x0F, 0xE0, //     #######
0x0F, 0xC0, //     ######
0x00, 0x00, //
0x00, 0x00, //

// @2880 'h' (14 pixels wide)
0x00, 0x00, //
0x38, 0x00, //   ###
0x38, 0x00, //   ###
0x18, 0x00, //    ##
```

```
    0x18, 0x00, //    ##
    0x1B, 0xC0, //    ## ####
    0x1F, 0xE0, //    ########
    0x1C, 0x60, //    ###   ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x3C, 0xF0, //   ####  ####
    0x3C, 0xF0, //   ####  ####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2920 'i' (14 pixels wide)
    0x00, 0x00, //
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x00, 0x00, //
    0x00, 0x00, //
    0x1F, 0x00, //    #####
    0x1F, 0x00, //    #####
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x1F, 0xE0, //    ########
    0x1F, 0xE0, //    ########
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @2960 'j' (14 pixels wide)
    0x00, 0x00, //
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x00, 0x00, //
    0x00, 0x00, //
    0x1F, 0xC0, //    #######
    0x1F, 0xC0, //    #######
    0x00, 0xC0, //         ##
    0x00, 0xC0, //         ##
    0x00, 0xC0, //         ##
```

```
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x00, 0xC0, //         ##
0x01, 0xC0, //        ###
0x3F, 0x80, //   #######
0x3F, 0x00, //   ######
0x00, 0x00, //
0x00, 0x00, //

// @3000 'k' (14 pixels wide)
0x00, 0x00, //
0x38, 0x00, //   ###
0x38, 0x00, //   ###
0x18, 0x00, //    ##
0x18, 0x00, //    ##
0x1B, 0xE0, //    ## #####
0x1B, 0xE0, //    ## #####
0x1B, 0x00, //    ## ##
0x1E, 0x00, //    ####
0x1E, 0x00, //    ####
0x1B, 0x00, //    ## ##
0x19, 0x80, //    ##  ##
0x39, 0xF0, //   ###  #####
0x39, 0xF0, //   ###  #####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3040 'l' (14 pixels wide)
0x00, 0x00, //
0x1F, 0x00, //    #####
0x1F, 0x00, //    #####
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x1F, 0xE0, //    ########
0x1F, 0xE0, //    ########
0x00, 0x00, //
0x00, 0x00, //
```

```
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3080 'm' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x7E, 0xE0, //  ###### ###
0x7F, 0xF0, //  ###########
0x33, 0x30, //   ##  ##  ##
0x33, 0x30, //   ##  ##  ##
0x33, 0x30, //   ##  ##  ##
0x33, 0x30, //   ##  ##  ##
0x33, 0x30, //   ##  ##  ##
0x7B, 0xB8, //  #### ### ###
0x7B, 0xB8, //  #### ### ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3120 'n' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x3B, 0xC0, //   ### ####
0x3F, 0xE0, //   #########
0x1C, 0x60, //    ###   ##
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
0x18, 0x60, //    ##    ##
0x3C, 0xF0, //   ####  ####
0x3C, 0xF0, //   ####  ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3160 'o' (14 pixels wide)
```

```
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x07, 0x80, //      ####
    0x1F, 0xE0, //    ########
    0x18, 0x60, //    ##    ##
    0x30, 0x30, //   ##      ##
    0x30, 0x30, //   ##      ##
    0x30, 0x30, //   ##      ##
    0x18, 0x60, //    ##    ##
    0x1F, 0xE0, //    ########
    0x07, 0x80, //      ####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @3200 'p' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x77, 0x80, //  ### ####
    0x7F, 0xE0, //  ##########
    0x38, 0x60, //   ###    ##
    0x30, 0x30, //   ##      ##
    0x30, 0x30, //   ##      ##
    0x30, 0x30, //   ##      ##
    0x38, 0x60, //   ###    ##
    0x3F, 0xE0, //   #########
    0x37, 0x80, //   ## ####
    0x30, 0x00, //   ##
    0x30, 0x00, //   ##
    0x7C, 0x00, //  #####
    0x7C, 0x00, //  #####
    0x00, 0x00, //
    0x00, 0x00, //

    // @3240 'q' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x07, 0xB8, //      #### ###
```

```
    0x1F, 0xF8, //    ##########
    0x18, 0x70, //    ##    ###
    0x30, 0x30, //   ##      ##
    0x30, 0x30, //   ##      ##
    0x30, 0x30, //   ##      ##
    0x18, 0x70, //    ##    ###
    0x1F, 0xF0, //    #########
    0x07, 0xB0, //      #### ##
    0x00, 0x30, //           ##
    0x00, 0x30, //           ##
    0x00, 0xF8, //         #####
    0x00, 0xF8, //         #####
    0x00, 0x00, //
    0x00, 0x00, //

    // @3280 'r' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x3C, 0xE0, //   ####  ###
    0x3D, 0xF0, //   #### #####
    0x0F, 0x30, //     ####  ##
    0x0E, 0x00, //     ###
    0x0C, 0x00, //     ##
    0x0C, 0x00, //     ##
    0x0C, 0x00, //     ##
    0x3F, 0xC0, //   ########
    0x3F, 0xC0, //   ########
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @3320 's' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x07, 0xE0, //      ######
    0x1F, 0xE0, //    ########
    0x18, 0x60, //    ##    ##
    0x1E, 0x00, //    ####
    0x0F, 0xC0, //     ######
    0x01, 0xE0, //        ####
    0x18, 0x60, //    ##    ##
```

```
    0x1F, 0xE0, //    ########
    0x1F, 0x80, //    ######
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @3360 't' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x0C, 0x00, //     ##
    0x0C, 0x00, //     ##
    0x0C, 0x00, //     ##
    0x3F, 0xE0, //   #########
    0x3F, 0xE0, //   #########
    0x0C, 0x00, //     ##
    0x0C, 0x00, //     ##
    0x0C, 0x00, //     ##
    0x0C, 0x00, //     ##
    0x0C, 0x30, //     ##    ##
    0x0F, 0xF0, //     ########
    0x07, 0xC0, //      #####
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @3400 'u' (14 pixels wide)
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x38, 0xE0, //   ###   ###
    0x38, 0xE0, //   ###   ###
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0x60, //    ##    ##
    0x18, 0xE0, //    ##   ###
    0x1F, 0xF0, //    #########
    0x0F, 0x70, //     #### ###
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
```

```
0x00, 0x00, //
0x00, 0x00, //

// @3440 'v' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x78, 0xF0, //  ####   ####
0x78, 0xF0, //  ####   ####
0x30, 0x60, //   ##     ##
0x18, 0xC0, //    ##   ##
0x18, 0xC0, //    ##   ##
0x0D, 0x80, //     ## ##
0x0D, 0x80, //     ## ##
0x07, 0x00, //      ###
0x07, 0x00, //      ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3480 'w' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x78, 0xF0, //  ####   ####
0x78, 0xF0, //  ####   ####
0x32, 0x60, //   ##  #  ##
0x32, 0x60, //   ##  #  ##
0x37, 0xE0, //   ## ######
0x1D, 0xC0, //    ### ###
0x1D, 0xC0, //    ### ###
0x18, 0xC0, //    ##   ##
0x18, 0xC0, //    ##   ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3520 'x' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
```

```
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x3C, 0xF0, //   ####  ####
0x3C, 0xF0, //   ####  ####
0x0C, 0xC0, //     ##  ##
0x07, 0x80, //      ####
0x03, 0x00, //       ##
0x07, 0x80, //      ####
0x0C, 0xC0, //     ##  ##
0x3C, 0xF0, //   ####  ####
0x3C, 0xF0, //   ####  ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3560 'y' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x78, 0xF0, //  ####   ####
0x78, 0xF0, //  ####   ####
0x30, 0x60, //   ##     ##
0x18, 0xC0, //    ##   ##
0x18, 0xC0, //    ##   ##
0x0D, 0x80, //     ## ##
0x0F, 0x80, //     #####
0x07, 0x00, //      ###
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x0C, 0x00, //     ##
0x7F, 0x00, //  #######
0x7F, 0x00, //  #######
0x00, 0x00, //
0x00, 0x00, //

// @3600 'z' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x1F, 0xE0, //    ########
0x1F, 0xE0, //    ########
0x18, 0xC0, //    ##   ##
```

```
    0x01, 0x80, //        ##
    0x03, 0x00, //       ##
    0x06, 0x00, //      ##
    0x0C, 0x60, //     ##   ##
    0x1F, 0xE0, //    ########
    0x1F, 0xE0, //    ########
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @3640 '{' (14 pixels wide)
    0x00, 0x00, //
    0x01, 0xC0, //        ###
    0x03, 0xC0, //       ####
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x07, 0x00, //      ###
    0x0E, 0x00, //     ###
    0x07, 0x00, //      ###
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0xC0, //       ####
    0x01, 0xC0, //        ###
    0x00, 0x00, //
    0x00, 0x00, //
    0x00, 0x00, //

    // @3680 '|' (14 pixels wide)
    0x00, 0x00, //
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
    0x03, 0x00, //       ##
```

```
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x03, 0x00, //       ##
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3720 '}' (14 pixels wide)
0x00, 0x00, //
0x1C, 0x00, //    ###
0x1E, 0x00, //    ####
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x07, 0x00, //      ###
0x03, 0x80, //       ###
0x07, 0x00, //      ###
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x06, 0x00, //      ##
0x1E, 0x00, //    ####
0x1C, 0x00, //    ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //

// @3760 '~' (14 pixels wide)
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x0E, 0x00, //     ###
0x3F, 0x30, //   ######  ##
0x33, 0xF0, //   ##  ######
0x01, 0xE0, //        ####
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00 //
```

```
};

sFONT Font20 = {
  Font20_Table,
  14, /* Width */
  20, /* Height */
};
```

font.h

```
/*
 * File:   font.h
 * Author: Lin
 *
 * Created on 2021?7?30?, ??11:02
 */

#ifndef FONT_H
#define        FONT_H

#include <stdint.h>

typedef struct _tFont
{
  const uint8_t *table;
  uint16_t Width;
  uint16_t Height;

} sFONT;

extern sFONT Font20;


#endif        /* FONT_H */
```

lcd_driver.c

```
/*****************************************************************
* | File                :        LCD_Driver.c
* | Author      :   Waveshare team
* | Function    :   LCD driver
* | Info        :
*----------------
* |        This version:   V1.0
* | Date        :   2018-12-18
* | Info        :
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documnetation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
```

```
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to  whom the Software is
# furished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS OR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
#
******************************************************************************/
#include "lcd_driver.h"
#include <stdio.h>
#include <stdlib.h>
/******************************************************************************
function:
        Hardware reset
******************************************************************************/
static void LCD_Reset(void)
{
        DEV_Delay_ms(200);
        DEV_Digital_Write(DEV_RST_PIN, 0);
        DEV_Delay_ms(200);
        DEV_Digital_Write(DEV_RST_PIN, 1);
        DEV_Delay_ms(200);
}

/******************************************************************************
function:
                Write data and commands
******************************************************************************/
static void LCD_Write_Command(UBYTE data)
{
        DEV_Digital_Write(DEV_CS_PIN, 0);
        DEV_Digital_Write(DEV_DC_PIN, 0);
        DEV_SPI_WRITE(data);
}

static void LCD_WriteData_Byte(UBYTE data)
{
        DEV_Digital_Write(DEV_CS_PIN, 0);
        DEV_Digital_Write(DEV_DC_PIN, 1);
        DEV_SPI_WRITE(data);
        DEV_Digital_Write(DEV_CS_PIN,1);
}
```

```c
void LCD_WriteData_Word(UWORD data)
{
        DEV_Digital_Write(DEV_CS_PIN, 0);
        DEV_Digital_Write(DEV_DC_PIN, 1);
        DEV_SPI_WRITE((data>>8) & 0xff);
        DEV_SPI_WRITE(data);
        DEV_Digital_Write(DEV_CS_PIN, 1);
}


/******************************************************************************
function:
                Common register initialization
******************************************************************************/
void LCD_Init(void)
{
    //DEV_Pin_Init();
        LCD_Reset();

        LCD_Write_Command(0x11); //Sleep out

        LCD_Write_Command(0xCF);
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(0xC1);
        LCD_WriteData_Byte(0X30);
        LCD_Write_Command(0xED);
        LCD_WriteData_Byte(0x64);
        LCD_WriteData_Byte(0x03);
        LCD_WriteData_Byte(0X12);
        LCD_WriteData_Byte(0X81);
        LCD_Write_Command(0xE8);
        LCD_WriteData_Byte(0x85);
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(0x79);
        LCD_Write_Command(0xCB);
        LCD_WriteData_Byte(0x39);
        LCD_WriteData_Byte(0x2C);
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(0x34);
        LCD_WriteData_Byte(0x02);
        LCD_Write_Command(0xF7);
        LCD_WriteData_Byte(0x20);
        LCD_Write_Command(0xEA);
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(0x00);
        LCD_Write_Command(0xC0); //Power control
        LCD_WriteData_Byte(0x1D); //VRH[5:0]
        LCD_Write_Command(0xC1); //Power control
        LCD_WriteData_Byte(0x12); //SAP[2:0];BT[3:0]
```

```
        LCD_Write_Command(0xC5); //VCM control
        LCD_WriteData_Byte(0x33);
        LCD_WriteData_Byte(0x3F);
        LCD_Write_Command(0xC7); //VCM control
        LCD_WriteData_Byte(0x92);
        LCD_Write_Command(0x3A); // Memory Access Control
        LCD_WriteData_Byte(0x55);
        LCD_Write_Command(0x36); // Memory Access Control
        LCD_WriteData_Byte(0x08);
        LCD_Write_Command(0xB1);
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(0x12);
        LCD_Write_Command(0xB6); // Display Function Control
        LCD_WriteData_Byte(0x0A);
        LCD_WriteData_Byte(0xA2);

        LCD_Write_Command(0x44);
        LCD_WriteData_Byte(0x02);

        LCD_Write_Command(0xF2); // 3Gamma Function Disable
        LCD_WriteData_Byte(0x00);
        LCD_Write_Command(0x26); //Gamma curve selected
        LCD_WriteData_Byte(0x01);
        LCD_Write_Command(0xE0); //Set Gamma
        LCD_WriteData_Byte(0x0F);
        LCD_WriteData_Byte(0x22);
        LCD_WriteData_Byte(0x1C);
        LCD_WriteData_Byte(0x1B);
        LCD_WriteData_Byte(0x08);
        LCD_WriteData_Byte(0x0F);
        LCD_WriteData_Byte(0x48);
        LCD_WriteData_Byte(0xB8);
        LCD_WriteData_Byte(0x34);
        LCD_WriteData_Byte(0x05);
        LCD_WriteData_Byte(0x0C);
        LCD_WriteData_Byte(0x09);
        LCD_WriteData_Byte(0x0F);
        LCD_WriteData_Byte(0x07);
        LCD_WriteData_Byte(0x00);
        LCD_Write_Command(0XE1); //Set Gamma
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(0x23);
        LCD_WriteData_Byte(0x24);
        LCD_WriteData_Byte(0x07);
        LCD_WriteData_Byte(0x10);
        LCD_WriteData_Byte(0x07);
        LCD_WriteData_Byte(0x38);
        LCD_WriteData_Byte(0x47);
        LCD_WriteData_Byte(0x4B);
        LCD_WriteData_Byte(0x0A);
```

```
        LCD_WriteData_Byte(0x13);
        LCD_WriteData_Byte(0x06);
        LCD_WriteData_Byte(0x30);
        LCD_WriteData_Byte(0x38);
        LCD_WriteData_Byte(0x0F);
        LCD_Write_Command(0x29); //Display on
}

/******************************************************************
function:        Set the cursor position
parameter        :
          Xstart:         Start UWORD x coordinate
          Ystart:        Start UWORD y coordinate
          Xend   :        End UWORD coordinates
          Yend   :        End UWORD coordinatesen
******************************************************************/
void LCD_SetWindow(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD  Yend)
{
        LCD_Write_Command(0x2a);
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(Xstart & 0xff);
        LCD_WriteData_Byte((Xend - 1) >> 8);
        LCD_WriteData_Byte((Xend - 1) & 0xff);

        LCD_Write_Command(0x2b);
        LCD_WriteData_Byte(0x00);
        LCD_WriteData_Byte(Ystart & 0xff);
        LCD_WriteData_Byte((Yend - 1) >> 8);
        LCD_WriteData_Byte((Yend - 1) & 0xff);

        LCD_Write_Command(0x2C);
}

/******************************************************************
function:        Settings window
parameter        :
          Xstart:         Start UWORD x coordinate
          Ystart:        Start UWORD y coordinate

******************************************************************/
void LCD_SetCursor(UWORD X, UWORD Y)
{
        LCD_Write_Command(0x2a);
        LCD_WriteData_Byte(X >> 8);
        LCD_WriteData_Byte(X);
        LCD_WriteData_Byte(X >> 8);
        LCD_WriteData_Byte(X);

        LCD_Write_Command(0x2b);
        LCD_WriteData_Byte(Y >> 8);
```

```
        LCD_WriteData_Byte(Y);
        LCD_WriteData_Byte(Y >> 8);
        LCD_WriteData_Byte(Y);

        LCD_Write_Command(0x2C);
}

/******************************************************************
function:       Clear screen function, refresh the screen to a certain color
parameter       :
          Color :               The color you want to clear all the screen
******************************************************************/
void LCD_Clear(UWORD Color)
{
        unsigned int i,j;
        LCD_SetWindow(0, 0, LCD_WIDTH, LCD_HEIGHT);
        DEV_Digital_Write(DEV_DC_PIN, 1);
        for(i = 0; i < LCD_WIDTH; i++){
                for(j = 0; j < LCD_HEIGHT; j++){
            //printf("i: %d, j: %d", i, j);
                        DEV_SPI_WRITE((Color>>8)&0xff);
                        DEV_SPI_WRITE(Color);
                }
         }
}

/******************************************************************
function:       Refresh a certain area to the same color
parameter       :
          Xstart: Start UWORD x coordinate
          Ystart:       Start UWORD y coordinate
          Xend  :       End UWORD coordinates
          Yend  :       End UWORD coordinates
          color :       Set the color
******************************************************************/
void LCD_ClearWindow(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend,UWORD color)
{
        UWORD i,j;
        LCD_SetWindow(Xstart, Ystart, Xend-1,Yend-1);
        for(i = Ystart; i <= Yend-1; i++){
                for(j = Xstart; j <= Xend-1; j++){
                        LCD_WriteData_Word(color);
                }
        }
}

/******************************************************************
function: Draw a point
parameter       :
            X       :       Set the X coordinate
```

```
            Y        :        Set the Y coordinate
          Color :        Set the color
******************************************************************/
void LCD_DrawPaint(UWORD x, UWORD y, UWORD Color)
{
        LCD_SetCursor(x, y);
        LCD_WriteData_Word(Color);
}
```

lcd_driver.h

```
/*
 * File:   lcd_driver.h
 * Author: Lin
 *
 * Created on 2021?7?20?, ??4:52
 */

#ifndef LCD_DRIVER_H
#define        LCD_DRIVER_H

#include "dev_config.h"
#include <p32xxxx.h>

#define LCD_WIDTH   240 //LCD width
#define LCD_HEIGHT  320 //LCD height

void LCD_WriteData_Word(UWORD da);

void LCD_SetCursor(UWORD X, UWORD Y);
void LCD_SetWindow(UWORD Xstart, UWORD Ystart, UWORD Xend,
    UWORD Yend);
void LCD_DrawPaint(UWORD x, UWORD y, UWORD Color);

void LCD_Init(void);
void LCD_SetBackLight(UWORD Value);

void LCD_Clear(UWORD Color);
void LCD_ClearWindow(UWORD Xstart, UWORD Ystart, UWORD Xend,
    UWORD Yend, UWORD color);

#endif        /* LCD_DRIVER_H */
```

main.c

```
#include <stdio.h>
#include <stdlib.h>

#include "GUI.h"
#include "utils.h"
#include "color_sensor.h"
```

```c
#include "ADC.h"

#pragma config FCKSM = CSECME

// TODO: define corresponding buttons
#define resetButton PORTEbits.RE4
#define confirmButton PORTEbits.RE5
#define selectButton PORTEbits.RE6
#define plusButton PORTEbits.RE7
#define minusButton PORTGbits.RG14

#define inkLevelMulti

#define c_id 0
#define m_id 1
#define y_id 2
#define k_id 3

int init_button() {
    LATECLR = 0xFF;
    LATGbits.LATG14 = 0;
    TRISECLR = 0xF;
    TRISESET = 0xF0;
    TRISGbits.TRISG14 = 1;

}

void rgb2cmyk(const int r, const int g, const int b, int *c, int *m, int *y, int *k) {
    int rgb_max = max(max(r, b), g);
    *c = (int) (100 * (-(double)r + (double)rgb_max) / (double)rgb_max);
    *m = (int) (100 * (-(double)g + (double)rgb_max) / (double)rgb_max);
    *y = (int) (100 * (-(double)b + (double)rgb_max) / (double)rgb_max);
    *k = (int) (100 * (1.0 - (rgb_max / 255.0)));
}

void release_ink(const int val, int id){
    LATESET = 0x1 << id;
    delay_ms(val);
    //delay_ms(1000);
    LATECLR = 0x1 << id;
}

int main(){
    setSYSCLK80MHzAndPBDIV(0b10);
    init_LCD();
    init_ADC();
    init_button();
    int resetFlag = 0;
    while(1){
        // Init State
```

```
        if(resetFlag == 1){
            resetFlag = 0;
        }
        LCD_Clear(0x0000);
        write_str(40, 100, "C", 0xFC10);
        write_str(60, 100, "o", 0xFFF0);
        write_str(80, 100, "l", 0x87F0);
        write_str(100, 100, "o", 0x87FF);
        write_str(120, 100, "r", 0xFC1F);
        write_str(140, 100, " Extractor", 0xFFFF);
        write_str(40, 180, "You can adjust", 0xFFFF);
        write_str(40, 200, "screen brightness", 0xFFFF);
        write_str(40, 220, "with knob", 0xFFFF);
        //LCD_Clear(0x0000);
        while(!confirmButton){
            if (resetButton) {
                delay_ms(1);
                if (resetButton) {
                    delay_ms(300);
                    if (resetButton) {
                        while (resetButton) {}
                        release_ink(500, c_id);
                        release_ink(500, m_id);
                        release_ink(500, y_id);
                        release_ink(500, k_id);
                    }
                }
            }

            int adc_val = read_ADC_val();
            set_LCD_brightness(adc_val);
#if 0
            char adc_str[20];
            sprintf(adc_str, "%d", adc_val);
            write_str(80, 100, adc_str, 0xFFFF);
            delay_ms(10);
            write_str(80, 100, "    ", 0xFFFF);
            delay_ms(10);
#endif
        }

        // Color Extraction State
        int R, G, B, inkLevel;
        inkLevel = 10;
        getRGB(&R, &G, &B);
        //int i = 0;
        //for (i = 0; i < 3; i ++) {
        //    inkLevel = read_ADC_val() / 100.0 + 1;
        //}
        // TODO: Read ADC into inkLevel
```

```
        //inkLevel = (inkLevel / 100) + 1;
        //inkLevel = 5; // only for test
        char str[100];
        LCD_Clear(0x0000);
        draw_interface(R, G, B, inkLevel);
        int adjustingColor = 0;
        write_str(115, 35 + 30 * adjustingColor, "<", 0xFFFF);
        while(1){
            if(resetButton){
                delay_ms(300);
                if (resetButton) {
                    resetFlag = 1;
                    break;
                }
                else {
                    getRGB(&R, &G, &B);
                    draw_interface(R, G, B, inkLevel);
                }
            }
            if(selectButton){
                draw_color_block(115, 30 + 30 * adjustingColor,
                    135, 65 + 30 * adjustingColor, 0x0000, 0);
                adjustingColor = (adjustingColor + 1) % 3;
                write_str(115, 35 + 30 * adjustingColor, "<", 0xFFFF);
                while(selectButton);
                delay_ms(1);
                while(selectButton);
            }
            if(plusButton | minusButton){
                int count = 0;
                while(plusButton | minusButton){
                    count++;
                    int increment = count > 5 ? 10 : 1;
                    switch(adjustingColor){
                        case 0:
                            R = plusButton ? (R + increment > 255 ? 255
                                : R + increment) : (R - increment < 0 ? 0
                                : R - increment);
                            //draw_color_block(45, 30, 105, 60, 0x0000, 0);
                            write_str(45, 35, "     ", 0x0000);
                            sprintf(str, "%d", R);
                            write_str(45, 35, str, 0xF800);
                            break;
                        case 1:
                            G = plusButton ? (G + increment > 255 ? 255
                                : G + increment) : (G - increment < 0 ? 0
                                : G - increment);
                            //draw_color_block(45, 60, 105, 90, 0x0000, 0);
                            write_str(45, 65, "     ", 0x0000);
                            sprintf(str, "%d", G);
```

```
                            write_str(45, 65, str, 0x07E0);
                            break;
                        case 2:
                            B = plusButton ? (B + increment > 255 ? 255
                                : B + increment) : (B - increment < 0 ? 0
                                : B - increment);
                            //draw_color_block(45, 90, 105, 120, 0x0000, 0);
                            write_str(45, 95, "     ", 0x0000);
                            sprintf(str, "%d", B);
                            write_str(45, 95, str, 0x001F);
                            break;
                    }
                    draw_color_block(200, 80, 280, 160,
                    ((R>>3)<<11) + ((G>>2)<<5) + ((B>>3)), 1);
                    T4CON = 0x0060; // PBCLK DIV by 64
                    PR4 = 62499;    // period = 62500, freq = 5Hz
                    TMR4 = 0;
                    T4CONSET = 0x8000;
                    while(!IFS0bits.T4IF){
                        if(!plusButton & !minusButton){
                            break;
                        }
                    }
                    T4CONCLR = 0x8000;
                    PR4 = 0;
                    IFS0bits.T4IF = 0;
                }
            }
            if(confirmButton){
                break;
            }
            // TODO: Read ADC
            int newInkLevel = 0;
            int i = 0;
            for (i = 0; i < 5; i ++) {
                newInkLevel += read_ADC_val() / 100;
#if 0
                char adc_str[20];
                sprintf(adc_str, "%d", inkLevel);
                write_str(80, 100, adc_str, 0xFFFF);
                delay_ms(10);
#endif
            }
            newInkLevel = newInkLevel / 5 + 1;
            if (newInkLevel == 11){
                newInkLevel -= 1;
            }
            if(newInkLevel != inkLevel){
                //draw_color_block(85, 120, 105, 155, 0x0000, 0);
                write_str(85, 125, "     ", 0xFFFF);
```

```
            sprintf(str, "%d", newInkLevel);
            write_str(85, 125, str, 0xFFFF);
            inkLevel = newInkLevel;
            //delay_ms(10);
        }
    }
    if(resetFlag == 1){
        continue;
    }

    // Inking State
    LCD_Clear(0x0000);
    write_str(40, 100, "Releasing Ink...", 0xFFFF);

    //break;
    // TODO: Inking..............
    int C, M, Y, K;
    rgb2cmyk(R, G, B, &C, &M, &Y, &K);
    write_str(30, 150, "C: ", 0x07FF);
    sprintf(str, "%d", C);
    write_str(75, 150, str, 0x07FF);
    write_str(130, 150, "M: ", 0xF81F);
    sprintf(str, "%d", M);
    write_str(175, 150, str, 0xF81F);
    write_str(30, 170, "Y: ", 0xFFE0);
    sprintf(str, "%d", Y);
    write_str(75, 170, str, 0xFFE0);
    write_str(130, 170, "K: ", 0xFFFF);
    sprintf(str, "%d", K);
    write_str(175, 170, str, 0xFFFF);
    write_str(30, 190, "Ink: ", 0xFFFF);
    sprintf(str, "%d", inkLevel);
    write_str(90, 190, str, 0xFFFF);

    //write_str(200, 175, "id: ", 0xFFFF);
    release_ink(C * inkLevel, c_id);
    delay_ms(100);
    //sprintf(str, "%d", c_id);
    //write_str(250, 175, str, 0xFFFF);
    release_ink(M * inkLevel, m_id);
    delay_ms(100);
    //sprintf(str, "%d", m_id);
    //write_str(250, 175, str, 0xFFFF);
    release_ink(Y * inkLevel, y_id);
    delay_ms(100);
    //sprintf(str, "%d", y_id);
    //write_str(250, 175, str, 0xFFFF);
    release_ink(K * inkLevel, k_id);
    delay_ms(100);
    //sprintf(str, "%d", k_id);
```

```
        //write_str(250, 175, str, 0xFFFF);

        while(!confirmButton){

            int newInkLevel = 0;
            int i = 0;
            for (i = 0; i < 5; i ++) {
                newInkLevel += read_ADC_val() / 100;
            }
            newInkLevel = newInkLevel / 5 + 1;
            if (newInkLevel == 11){
                newInkLevel -= 1;
            }
            if(newInkLevel != inkLevel){
                //draw_color_block(85, 120, 105, 155, 0x0000, 0);
                write_str(75, 190, "     ", 0xFFFF);
                sprintf(str, "%d", newInkLevel);
                write_str(75, 190, str, 0xFFFF);
                inkLevel = newInkLevel;
                //delay_ms(10);
            }

            if (resetButton) {
                delay_ms(300);
                if (resetButton){
                    //while (resetButton) {}
                    break;
                }
                else {
                    release_ink(C * inkLevel, c_id);
                    delay_ms(100);
                    release_ink(M * inkLevel, m_id);
                    delay_ms(100);
                    release_ink(Y * inkLevel, y_id);
                    delay_ms(100);
                    release_ink(K * inkLevel, k_id);
                    delay_ms(100);
                }
            }
        }
        delay_ms(500);

    }
//    init_LCD();
//    TRISECLR = 0xFF;
//    LATE = 0x0;
//    while(1){
//        int R, G, B;
//        getRGB(&R, &G, &B);
//        delay_ms(10000);
```

```
//    }
}
```

utils.c

```
#include "utils.h"

#define CLK_FREQ_MS 20000 // 20Mhz / 1000 (1ms)
#define CLK_FREQ_US 20 // 20Mhz / 1000000 (1us)

int timer_n, timer_flag = 0;

/* timer interrupt handler */
#pragma interrupt T3_ISR ipl3AUTO vector 12
void T3_ISR (void) {
    IFS0bits.T3IF = 0;
    timer_n--;
    if(timer_n > 0)
        return;
    T3CONCLR = 0x8000;
    TMR3 = 0x0;
    timer_flag = 1;
}

void delay_init() {
    T3CON = 0x0;
    TMR3 = 0x0;

    IPC3bits.T3IP = 3;
    IPC3bits.T3IS = 0;
    IFS0bits.T3IF = 0;
    IEC0bits.T3IE = 1;
}

void delay_ms(int xms) {
    timer_n = xms;
    timer_flag = 0;
    PR3 = CLK_FREQ_MS;
    T3CONSET = 0x8000;
    while (!timer_flag) {}
}


void delay_us(int xus) {
    timer_n = xus;
    timer_flag = 0;
    PR3 = CLK_FREQ_US;
    T3CONSET = 0x8000;
    while (!timer_flag) {}
}
```

```
void setSYSCLK80MHzAndPBDIV(uint8_t n){
    SYSKEY = 0x0;
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    OSCCONbits.PLLODIV = 0b000; // PLL output divisor = 1
    OSCCONbits.PBDIV = n;       // PBCLK = SYSCLK / 4
    OSCCONbits.PLLMULT = 0b101; // PLL output multiplier = 20
    OSCCONbits.NOSC = 0b001;    // FRCPLL
    OSCCONbits.OSWEN = 0b1;
    while(OSCCONbits.OSWEN);
    SYSKEY = 0x0;
}
```

utils.h

```
/*
 * File:   utils.h
 * Author: Lin
 *
 * Created on 2021?7?30?, ??4:51
 */

#ifndef UTILS_H
#define        UTILS_H

#include <stdint.h>
#include <p32xxxx.h>

void delay_init();

void delay_ms(int xms);

void delay_us(int xus);

void setSYSCLK80MHzAndPBDIV(uint8_t n);

#endif        /* UTILS_H */
```