Infonique
iSEB Door V1.0

Prepared by	Date	Version
Bing Ran	24/2/2024	1.0

Abstract

This document provides detailed of Infonique iSEB Door specification.

Document History

Date	Rev	Modifier	Changes
24-Feb-2024	1.0	Bing Ran	First Draft

Contents

Table of Figures

Table of table

1 Introduction

This document will discuss the details of the iSEB door.

2 iSEB Door

ISEB Door contains several module such as Door control unit, ESP32-S3-EYE,Relay module,voltage regulator module , Arduino Uno with iSEB expansion board and limit switch. ISEB Door able to control the door , perform human recognition and record the usage of the door usage. The following is showing the overview of the iSEB Door.

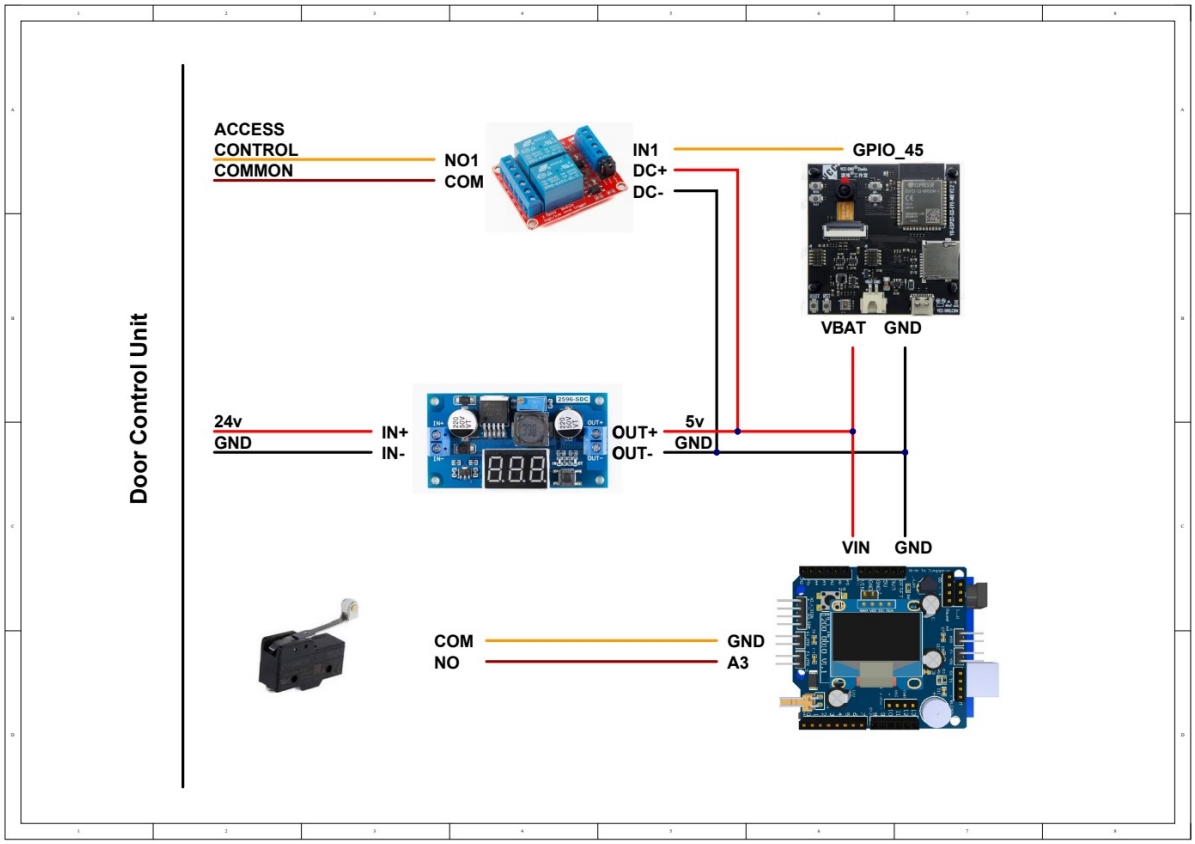

Figure 1: Overview of ISEB Door

3 Door Control Unit

ISEB door is using the DSW-60 from Deper. It is a slim and small size. It is a automatic swing door openers with motion sensors. The figure below is showing DSW-60 from Deper.

Figure 2: DSW-60 from Deper

We have to power up it with 240ac and there is a switch for us to swich on and off the door control unit. For iSEB door we only send signal to connector 1 of the door control unit hence we will focus on connector 1 only. The details of connector 1 is showing in the figure below.

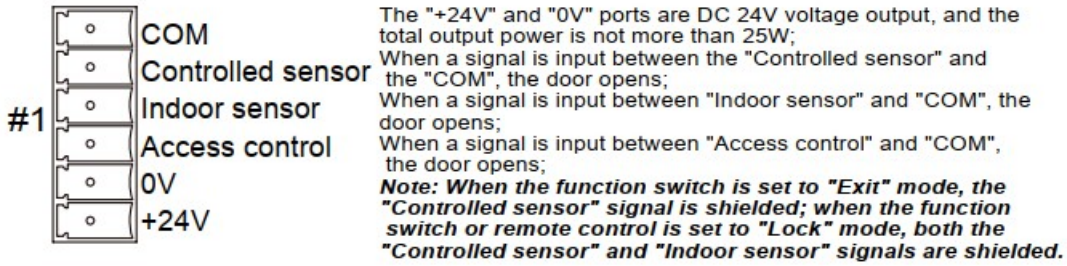

Figure 3: Connector 1

ISEB door will open the door by shorting COM to Access control. Beside that, 0v and +24v will supply power to voltage regulator module. Beside that to configure the other parameter of the door control unit we will need setting panel to configure it.The details of the setting panel is showing in the figure below.

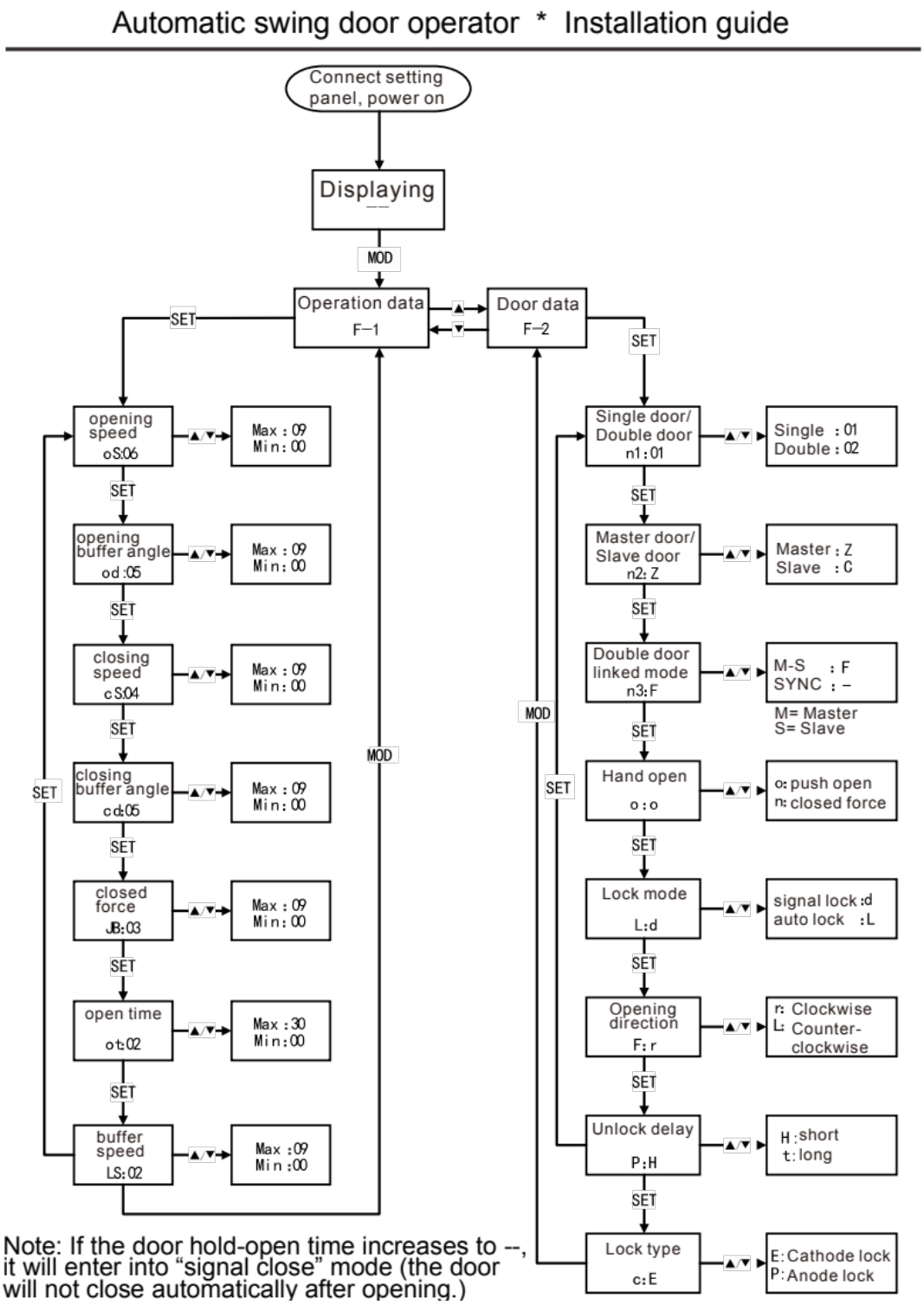

Figure 4: Detail of setting panels

4 Voltage regulator module

ISEB door is using LM2596 module as the voltage regulator module. Voltage regulator module will conver 24v to 5v because the operationg voltage of other module is 5v. The following figure is the showing LM2596 module.

Figure 5: LM2596 module

There is variable resistor , 7 segment display and a switch on the LM2596. The variable resistor is to adjust the output voltage. 7 segment display is showing the input or output voltage of the LM25965. The switch is to toggle the 7 segment display to show input or output voltage.

5 ESP32-S3-EYE

ESP32-S3-EYE is a small-sized AI development board. It is based on the ESP32-S3 Soc and ESP-WHO, Espressfig's AI development framework It features a 2-Megapixel camera, an LCD display , and a microphone, which are used for image reconition and audio processing. ESP32-S3-EYE also offers plenty of storage, with an 8MB octal PSRAM and a 8MB flash. It also supports image transmission via WiFI and debugging through a usb port.

For iSEB door , we will involve only switches , 2-megapixel camera and an LCD display. The figure below is showing the ESP32-S3-EYE that used by iSEB door.

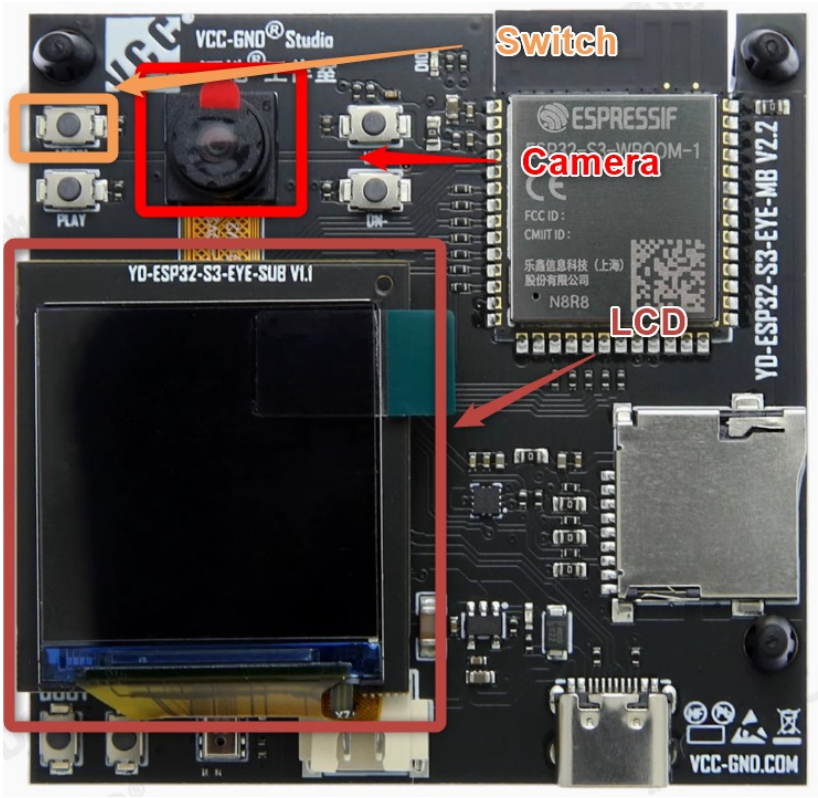

Figure 6: ESP32-S3-EYE

During start up , ESP32-S3-EYE will be in default mode. We have to press the switch to enter face recognition mode. ESP32-S3 EYE will detect human face and send signal to door control unit to open the door through GPIO45.

5.1 Coding

The sample code is available at https://github.com/bingran/esp-who/tree/master. The details of ESP32-S3-EYE such as setting up the debug environment and schematic diagram are mentioned in the github page also. The sample code used for iSEB door will be esp-who/example/esp32-s3-eye.

5.1.1 Code for Main task

The following is the main loop of esp32-s3-eye

```
#include "driver/gpio.h"
#include "app_button.hpp"
#include "app_camera.hpp"
#include "app_lcd.hpp"
#include "app_led.hpp"
#include "app_motion.hpp"
#include "app_speech.hpp"
#include "app_face.hpp"

extern "C" void app_main()
{
    gpio_config_t gpio_conf;
    gpio_conf.mode = GPIO_MODE_OUTPUT_OD;
    gpio_conf.pull_up_en = GPIO_PULLUP_ENABLE;

    gpio_conf.intr_type = GPIO_INTR_DISABLE;
    gpio_conf.pin_bit_mask = 1LL << GPIO_NUM_45;
    gpio_config(&gpio_conf);

    gpio_set_level(GPIO_NUM_45, 1);

    QueueHandle_t xQueueFrame_0 = xQueueCreate(2, sizeof(camera_fb_t *));
    QueueHandle_t xQueueFrame_1 = xQueueCreate(2, sizeof(camera_fb_t *));
    QueueHandle_t xQueueFrame_2 = xQueueCreate(2, sizeof(camera_fb_t *));

    AppButton *key = new AppButton();
    AppSpeech *speech = new AppSpeech();
    AppCamera *camera = new AppCamera(PIXFORMAT_RGB565, FRAMESIZE_240X240, 2,
xQueueFrame_0);
    AppFace *face = new AppFace(key, speech, xQueueFrame_0, xQueueFrame_1);
    AppMotion *motion = new AppMotion(key, speech, xQueueFrame_1, xQueueFrame_2);
    AppLCD *lcd = new AppLCD(key, speech, xQueueFrame_2);
    AppLED *led = new AppLED(GPIO_NUM_3, key, speech);

    key->attach(face);
    key->attach(motion);
    key->attach(led);
    key->attach(lcd);

    speech->attach(face);
```

```
    speech->attach(motion);
    speech->attach(led);
    speech->attach(lcd);

    lcd->run();
    motion->run();
    face->run();
    camera->run();
    speech->run();
    key->run();
}
```

5.1.2 Code for face recognition task

```
#include "app_face.hpp"
#include <list>

#include "esp_log.h"
#include "esp_camera.h"

#include "dl_image.hpp"
#include "fb_gfx.h"

#include "who_ai_utils.hpp"

static const char TAG[] = "App/Face";

#define RGB565_MASK_RED 0xF800
#define RGB565_MASK_GREEN 0x07E0
#define RGB565_MASK_BLUE 0x001F

#define FRAME_DELAY_NUM 16

static void rgb_print(camera_fb_t *fb, uint32_t color, const char *str)
{
    fb_gfx_print(fb, (fb->width - (strlen(str) * 14)) / 2, 10, color, str);
}

static int rgb_printf(camera_fb_t *fb, uint32_t color, const char *format, ...)
{
    char loc_buf[64];
    char *temp = loc_buf;
    int len;
    va_list arg;
    va_list copy;
    va_start(arg, format);
    va_copy(copy, arg);
    len = vsnprintf(loc_buf, sizeof(loc_buf), format, arg);
    va_end(copy);
```

```
    if (len >= sizeof(loc_buf))
    {
        temp = (char *)malloc(len + 1);
        if (temp == NULL)
        {
            return 0;
        }
    }
    vsnprintf(temp, len + 1, format, arg);
    va_end(arg);
    rgb_print(fb, color, temp);
    if (len > 64)
    {
        free(temp);
    }
    return len;
}

AppFace::AppFace(AppButton *key,
                 AppSpeech *speech,
                 QueueHandle_t queue_i,
                 QueueHandle_t queue_o,
                 void (*callback)(camera_fb_t *)) : Frame(queue_i, queue_o, callback),
                                                    key(key),
                                                    speech(speech),
                                                    detector(0.3F, 0.3F, 10, 0.3F),
                                                    detector2(0.4F, 0.3F, 10),
                                                    state(FACE_IDLE),
                                                    switch_on(false),
                                                    gpio_on(false)
{
#if CONFIG_MFN_V1
#if CONFIG_S8
    this->recognizer = new FaceRecognition112V1S8();
#elif CONFIG_S16
    this->recognizer = new FaceRecognition112V1S16();
#endif
#endif

    this->recognizer->set_partition(ESP_PARTITION_TYPE_DATA, ESP_PARTITION_SUBTYPE_ANY,
"fr");
    this->recognizer->set_ids_from_flash();
}

AppFace::~AppFace()
{
    delete this->recognizer;
}
```

```
void AppFace::update()
{
  // Parse key
  if (this->key->pressed > BUTTON_IDLE)
  {
    if (this->key->pressed == BUTTON_MENU)
    {
      this->state = FACE_IDLE;
      this->switch_on = (this->key->menu == MENU_FACE_RECOGNITION) ? true : false;
      ESP_LOGD(TAG, "%s", this->switch_on ? "ON" : "OFF");
    }
    else if (this->key->pressed == BUTTON_PLAY)
    {
      this->state = FACE_RECOGNIZE;
    }
    else if (this->key->pressed == BUTTON_UP)
    {
      this->state = FACE_ENROLL;
    }
    else if (this->key->pressed == BUTTON_DOWN)
    {
      this->state = FACE_DELETE;
    }
  }

  // Parse speech recognition
  if (this->speech->command > COMMAND_NOT_DETECTED)
  {
    if (this->speech->command >= MENU_STOP_WORKING && this->speech->command <=
MENU_MOTION_DETECTION)
    {
      this->state = FACE_IDLE;
      this->switch_on = (this->speech->command == MENU_FACE_RECOGNITION) ? true : false;
      ESP_LOGD(TAG, "%s", this->switch_on ? "ON" : "OFF");
    }
    else if (this->speech->command == ACTION_ENROLL)
    {
      this->state = FACE_ENROLL;
    }
    else if (this->speech->command == ACTION_RECOGNIZE)
    {
      this->state = FACE_RECOGNIZE;
    }
    else if (this->speech->command == ACTION_DELETE)
    {
      this->state = FACE_DELETE;
    }
```

```
    }
    ESP_LOGD(TAG, "Human face recognition state = %d", this->state);
}

static void task(AppFace *self)
{
    ESP_LOGD(TAG, "Start");
    camera_fb_t *frame = nullptr;

    while (true)
    {
        if (self->queue_i == nullptr)
            break;

        if (xQueueReceive(self->queue_i, &frame, portMAX_DELAY))
        {
            if (self->switch_on)
            {
                std::list<dl::detect::result_t> &detect_candidates = self->detector.infer((uint16_t *)frame->buf, {(int)frame->height, (int)frame->width, 3});
                std::list<dl::detect::result_t> &detect_results = self->detector2.infer((uint16_t *)frame->buf, {(int)frame->height, (int)frame->width, 3}, detect_candidates);

                if (detect_results.size())
                {
                    // print_detection_result(detect_results);
                    self->gpio_on = true;
                    draw_detection_result((uint16_t *)frame->buf, frame->height, frame->width, detect_results);
                }

                if (self->state)
                {
                    if (detect_results.size() == 1)
                    {
                        if (self->state == FACE_ENROLL)
                        {
                            self->recognizer->enroll_id((uint16_t *)frame->buf, {(int)frame->height, (int)frame->width, 3}, detect_results.front().keypoint, "", true);
                            ESP_LOGI(TAG, "Enroll ID %d", self->recognizer->get_enrolled_ids().back().id);
                        }
                        else if (self->state == FACE_RECOGNIZE)
                        {
                            self->recognize_result = self->recognizer->recognize((uint16_t *)frame->buf, {(int)frame->height, (int)frame->width, 3}, detect_results.front().keypoint);
                            // print_detection_result(detect_results);
                            ESP_LOGI(TAG, "Similarity: %f", self->recognize_result.similarity);
                            if(-1 !=  self->recognize_result.similarity)
```

```
                {
                    if ((self->recognize_result.id > 0) && (self->recognize_result.similarity > 0.5))
                    {
                        ESP_LOGI(TAG, "Match ID: %d", self->recognize_result.id);
                    }
                    else
                        ESP_LOGI(TAG, "Match ID: %d", self->recognize_result.id);
                }
                else
                {
                    self->recognize_result.id = 0;
                }
            }
        }
        else
        {
            self->recognize_result.id = 0xFF;
        }

        if (self->state == FACE_DELETE)
        {
            vTaskDelay(10);
            self->recognizer->delete_id(true);
            ESP_LOGI(TAG, "%d IDs left", self->recognizer->get_enrolled_id_num());
        }

        self->state_previous = self->state;
        self->state = FACE_IDLE;
        self->frame_count = FRAME_DELAY_NUM;
    }

    // Write result on several frames of image
    if (self->frame_count)
    {
        switch (self->state_previous)
        {
        case FACE_DELETE:
            rgb_printf(frame, RGB565_MASK_RED, "%d IDs left", self->recognizer-
>get_enrolled_id_num());
            break;

        case FACE_RECOGNIZE:
            if(0xFF == self->recognize_result.id)
            {
                rgb_print(frame, RGB565_MASK_RED, "No face detected!");
            }
            else if (self->recognize_result.id > 0)
            {
```

```
                rgb_printf(frame, RGB565_MASK_GREEN, "ID %d", self->recognize_result.id);
              }
              else
                rgb_print(frame, RGB565_MASK_RED, "who ?");
              break;

            case FACE_ENROLL:
              rgb_printf(frame, RGB565_MASK_BLUE, "Enroll: ID %d", self->recognizer->get_enrolled_ids().back().id);
              break;

            default:
              break;
            }

            self->frame_count--;
          }
        }

        if (self->queue_o)
          xQueueSend(self->queue_o, &frame, portMAX_DELAY);
        else
          self->callback(frame);
      }
    }

    ESP_LOGD(TAG, "Stop");
    vTaskDelete(NULL);
}

static void task2(AppFace *self)
{
    while(1)
    {
      if(true == self->gpio_on)
      {
        gpio_set_level(GPIO_NUM_45, 0);
        ESP_LOGI(TAG, "GPIO45 set to 0");
        vTaskDelay(500 / portTICK_PERIOD_MS);
        gpio_set_level(GPIO_NUM_45, 1);
        ESP_LOGI(TAG, "GPIO45 set to 1");
        self->gpio_on = false;
      }
      else
      {
        vTaskDelay(500 / portTICK_PERIOD_MS);
      }
    }
```

```
}
void AppFace::run()
{
   xTaskCreatePinnedToCore((TaskFunction_t)task, TAG, 5 * 1024, this, 5, NULL, 1);
   xTaskCreatePinnedToCore((TaskFunction_t)task2, TAG, 5 * 1024, this, 5, NULL,0);
}
```

5.1.3 Code for Camera task

```
#include "app_camera.hpp"
#include "esp_log.h"
#include "esp_system.h"
const static char TAG[] = "App/Camera";
AppCamera::AppCamera(const pixformat_t pixel_fromat,
                const framesize_t frame_size,
                const uint8_t fb_count,
                QueueHandle_t queue_o) : Frame(nullptr, queue_o, nullptr)
{
   ESP_LOGI(TAG, "Camera module is %s", CAMERA_MODULE_NAME);

#if CONFIG_CAMERA_MODEL_ESP_EYE || CONFIG_CAMERA_MODEL_ESP32_CAM_BOARD
   /* IO13, IO14 is designed for JTAG by default,
    * to use it as generalized input,
    * firstly declair it as pullup input */
   gpio_config_t conf;
   conf.mode = GPIO_MODE_INPUT;
   conf.pull_up_en = GPIO_PULLUP_ENABLE;
   conf.pull_down_en = GPIO_PULLDOWN_DISABLE;
   conf.intr_type = GPIO_INTR_DISABLE;
   conf.pin_bit_mask = 1LL << 13;
   gpio_config(&conf);
   conf.pin_bit_mask = 1LL << 14;
   gpio_config(&conf);
#endif

   camera_config_t config;
   config.ledc_channel = LEDC_CHANNEL_0;
   config.ledc_timer = LEDC_TIMER_0;
   config.pin_d0 = CAMERA_PIN_D0;
   config.pin_d1 = CAMERA_PIN_D1;
   config.pin_d2 = CAMERA_PIN_D2;
   config.pin_d3 = CAMERA_PIN_D3;
   config.pin_d4 = CAMERA_PIN_D4;
   config.pin_d5 = CAMERA_PIN_D5;
   config.pin_d6 = CAMERA_PIN_D6;
   config.pin_d7 = CAMERA_PIN_D7;
   config.pin_xclk = CAMERA_PIN_XCLK;
   config.pin_pclk = CAMERA_PIN_PCLK;
   config.pin_vsync = CAMERA_PIN_VSYNC;
   config.pin_href = CAMERA_PIN_HREF;
   config.pin_sscb_sda = CAMERA_PIN_SIOD;
   config.pin_sscb_scl = CAMERA_PIN_SIOC;
   config.pin_pwdn = CAMERA_PIN_PWDN;
   config.pin_reset = CAMERA_PIN_RESET;
   config.xclk_freq_hz = XCLK_FREQ_HZ;
```

```
    config.pixel_format = pixel_fromat;
    config.frame_size = frame_size;
    config.jpeg_quality = 12;
    config.fb_count = fb_count;
    config.fb_location = CAMERA_FB_IN_PSRAM;
    config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;

    // camera init
    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK)
    {
        ESP_LOGE(TAG, "Camera init failed with error 0x%x", err);
        return;
    }

    sensor_t *s = esp_camera_sensor_get();
    s->set_vflip(s, 1); // flip it back
    // initial sensors are flipped vertically and colors are a bit saturated
    if (s->id.PID == OV3660_PID)
    {
        s->set_brightness(s, 1);  // up the blightness just a bit
        s->set_saturation(s, -2); // lower the saturation
    }
    s->set_sharpness(s, 2);
    s->set_awb_gain(s, 2);
}

static void task(AppCamera *self)
{
    ESP_LOGD(TAG, "Start");
    while (true)
    {
        if (self->queue_o == nullptr)
            break;

        camera_fb_t *frame = esp_camera_fb_get();
        if (frame)
            xQueueSend(self->queue_o, &frame, portMAX_DELAY);
    }
    ESP_LOGD(TAG, "Stop");
    vTaskDelete(NULL);
}

void AppCamera::run()
{
    xTaskCreatePinnedToCore((TaskFunction_t)task, TAG, 2 * 1024, this, 5, NULL, 0);
}
```

5.1.3 Code for LCD task

```
#include "app_lcd.hpp"
#include <string.h>
#include "esp_log.h"
#include "esp_camera.h"
#include "logo_en_240x240_lcd.h"
static const char TAG[] = "App/LCD";

AppLCD::AppLCD(AppButton *key,
           AppSpeech *speech,
           QueueHandle_t queue_i,
           QueueHandle_t queue_o,
           void (*callback)(camera_fb_t *)) : Frame(queue_i, queue_o, callback),
                                    key(key),
                                    speech(speech),
                                    panel_handle(NULL),
                                    switch_on(false)
{
  do
  {
    ESP_LOGI(TAG, "Initialize SPI bus");
    spi_bus_config_t bus_conf = {
      .mosi_io_num = BOARD_LCD_MOSI,
      .miso_io_num = BOARD_LCD_MISO,
      .sclk_io_num = BOARD_LCD_SCK,
      .quadwp_io_num = -1,
      .quadhd_io_num = -1,
      .max_transfer_sz = BOARD_LCD_H_RES * BOARD_LCD_V_RES * sizeof(uint16_t),
    };
    ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &bus_conf, SPI_DMA_CH_AUTO));

    ESP_LOGI(TAG, "Install panel IO");
    esp_lcd_panel_io_handle_t io_handle = NULL;
    esp_lcd_panel_io_spi_config_t io_config = {
      .cs_gpio_num = BOARD_LCD_CS,
      .dc_gpio_num = BOARD_LCD_DC,
      .spi_mode = 0,
      .pclk_hz = BOARD_LCD_PIXEL_CLOCK_HZ,
      .trans_queue_depth = 10,
      .lcd_cmd_bits = BOARD_LCD_CMD_BITS,
      .lcd_param_bits = BOARD_LCD_PARAM_BITS,
    };
    // Attach the LCD to the SPI bus
    ESP_ERROR_CHECK(esp_lcd_new_panel_io_spi((esp_lcd_spi_bus_handle_t)SPI2_HOST,
&io_config, &io_handle));

    // ESP_LOGI(TAG, "Install ST7789 panel driver");
```

```
        esp_lcd_panel_dev_config_t panel_config = {
            .reset_gpio_num = BOARD_LCD_RST,
            .rgb_endian = LCD_RGB_ENDIAN_RGB,
            .bits_per_pixel = 16,
        };
        ESP_ERROR_CHECK(esp_lcd_new_panel_st7789(io_handle, &panel_config, &panel_handle));
        ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
        ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));
        esp_lcd_panel_invert_color(panel_handle, true);// Set inversion for esp32s3eye

        // turn on display
        esp_lcd_panel_disp_on_off(panel_handle, true);

        this->draw_color(0x000000);
        vTaskDelay(pdMS_TO_TICKS(500));
        this->draw_wallpaper();
        vTaskDelay(pdMS_TO_TICKS(1000));
    } while (0);
}

void AppLCD::draw_wallpaper()
{
    uint16_t *pixels = (uint16_t *)heap_caps_malloc((logo_en_240x240_lcd_width *
logo_en_240x240_lcd_height) * sizeof(uint16_t), MALLOC_CAP_8BIT | MALLOC_CAP_SPIRAM);
    if (NULL == pixels)
    {
        ESP_LOGE(TAG, "Memory for bitmap is not enough");
        return;
    }
    memcpy(pixels, logo_en_240x240_lcd, (logo_en_240x240_lcd_width *
logo_en_240x240_lcd_height) * sizeof(uint16_t));
    esp_lcd_panel_draw_bitmap(panel_handle, 0, 0, logo_en_240x240_lcd_width,
logo_en_240x240_lcd_height, (uint16_t *)pixels);
    heap_caps_free(pixels);

    this->paper_drawn = true;
}

void AppLCD::draw_color(int color)
{
    uint16_t *buffer = (uint16_t *)malloc(BOARD_LCD_H_RES * sizeof(uint16_t));
    if (NULL == buffer)
    {
        ESP_LOGE(TAG, "Memory for bitmap is not enough");
    }
    else
    {
        for (size_t i = 0; i < BOARD_LCD_H_RES; i++)
```

```
      {
        buffer[i] = color;
      }

      for (int y = 0; y < BOARD_LCD_V_RES; y++)
      {
        esp_lcd_panel_draw_bitmap(panel_handle, 0, y, BOARD_LCD_H_RES, y+1, buffer);
      }

      free(buffer);
    }
}

void AppLCD::update()
{
    if (this->key->pressed > BUTTON_IDLE)
    {
      if (this->key->pressed == BUTTON_MENU)
      {
        this->switch_on = (this->key->menu == MENU_STOP_WORKING) ? false : true;
        ESP_LOGD(TAG, "%s", this->switch_on ? "ON" : "OFF");
      }
    }

    if (this->speech->command > COMMAND_NOT_DETECTED)
    {
      if (this->speech->command >= MENU_STOP_WORKING && this->speech->command <=
MENU_MOTION_DETECTION)
      {
        this->switch_on = (this->speech->command == MENU_STOP_WORKING) ? false : true;
        ESP_LOGD(TAG, "%s", this->switch_on ? "ON" : "OFF");
      }
    }

    if (this->switch_on == false)
    {
      this->paper_drawn = false;
    }
}

static void task(AppLCD *self)
{
    ESP_LOGD(TAG, "Start");

    camera_fb_t *frame = nullptr;
    while (true)
    {
      if (self->queue_i == nullptr)
```

```
            break;

        if (xQueueReceive(self->queue_i, &frame, portMAX_DELAY))
        {
            if (self->switch_on)
                esp_lcd_panel_draw_bitmap(self->panel_handle, 0, 0, frame->width, frame->height,
(uint16_t *)frame->buf);
            else if (self->paper_drawn == false)
                self->draw_wallpaper();

            if (self->queue_o)
                xQueueSend(self->queue_o, &frame, portMAX_DELAY);
            else
                self->callback(frame);
        }
    }
    ESP_LOGD(TAG, "Stop");
    self->draw_wallpaper();
    vTaskDelete(NULL);
}

void AppLCD::run()
{
    xTaskCreatePinnedToCore((TaskFunction_t)task, TAG, 2 * 1024, this, 5, NULL, 1);
}
```

5.4 Code for button task

```
#include "app_button.hpp"
#include <stdio.h>
#include <stdlib.h>

#include "esp_timer.h"
#include "esp_log.h"
#include "soc/soc_caps.h"
#include "esp_adc/adc_oneshot.h"
#include "esp_adc/adc_cali.h"
#include "esp_adc/adc_cali_scheme.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

// ADC Channels
#define ADC1_EXAMPLE_CHAN0 ADC_CHANNEL_0
// ADC Attenuation
#define ADC_EXAMPLE_ATTEN ADC_ATTEN_DB_11
// ADC Calibration
#if CONFIG_IDF_TARGET_ESP32
#define ADC_EXAMPLE_CALI_SCHEME ESP_ADC_CAL_VAL_EFUSE_VREF
#elif CONFIG_IDF_TARGET_ESP32S2
#define ADC_EXAMPLE_CALI_SCHEME ESP_ADC_CAL_VAL_EFUSE_TP
#elif CONFIG_IDF_TARGET_ESP32C3
#define ADC_EXAMPLE_CALI_SCHEME ESP_ADC_CAL_VAL_EFUSE_TP
#elif CONFIG_IDF_TARGET_ESP32S3
#define ADC_EXAMPLE_CALI_SCHEME ESP_ADC_CAL_VAL_EFUSE_TP_FIT
#endif

static adc_oneshot_unit_handle_t adc1_handle = NULL;

#define PRESS_INTERVAL 500000

static const char *TAG = "App/Button";

AppButton::AppButton() : key_configs({{BUTTON_MENU, 2800, 3000}, {BUTTON_PLAY, 2250, 2450},
{BUTTON_UP, 300, 500}, {BUTTON_DOWN, 850, 1050}}),
                         pressed(BUTTON_IDLE), menu(0)
{
   if (adc1_handle){
      ESP_LOGE(TAG, "Button adc has been initialized");
   }

   adc_oneshot_unit_init_cfg_t init_config1 = {
      .unit_id = ADC_UNIT_1,
      .ulp_mode = ADC_ULP_MODE_DISABLE,
   };
```

```
    ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &adc1_handle));

    adc_oneshot_chan_cfg_t config = {
        .atten = ADC_ATTEN_DB_11,
        .bitwidth = ADC_BITWIDTH_DEFAULT,
    };
    ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, ADC1_EXAMPLE_CHAN0,
&config));
}

static void task(AppButton *self)
{
    int64_t backup_time = esp_timer_get_time();
    int64_t last_time = esp_timer_get_time();

    uint8_t menu_count = 0;

    while (true)
    {
        int voltage = 0;
        ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, ADC1_EXAMPLE_CHAN0, &voltage));
        backup_time = esp_timer_get_time();
        for (auto key_config : self->key_configs)
        {
            if ((voltage >= key_config.min) && (voltage <= key_config.max))
            {
                if (((backup_time - last_time) > PRESS_INTERVAL))
                {
                    self->pressed = key_config.key;
                    ESP_LOGI(TAG, "Button[%d] is clicked", self->pressed);

                    if (self->pressed == BUTTON_MENU)
                    {
                        if(MENU_FACE_RECOGNITION == self->menu)
                        {
                            self->menu = MENU_STOP_WORKING;
                        }
                        else
                        {
                            self->menu = MENU_FACE_RECOGNITION;
                        }
                    }

                    last_time = backup_time;
                    self->notify();

                    self->pressed = BUTTON_IDLE;
                    break;
```

```
            }
          }
        }
        vTaskDelay(pdMS_TO_TICKS(10));
        if(!((MENU_FACE_RECOGNITION == self->menu)&&(MENU_STOP_WORKING == self->menu)))
        {
          self->menu = MENU_STOP_WORKING;
        }
    }
}

void AppButton::run()
{
    xTaskCreatePinnedToCore((TaskFunction_t)task, TAG, 3 * 1024, this, 5, NULL, 0);
}
```

6 Relay module

ISEB door is using relay module to trasnfer signal from ESP32-S3-EYE to door control unit. This is because door control unit is working at higher voltage. The gpio of ESP32-S3 is only tolerate 3.3v, hence, a relay module is require to send signal from ESP32-S3-EYE to door control unit. The figure below is showing the relay module used in iSEB door.

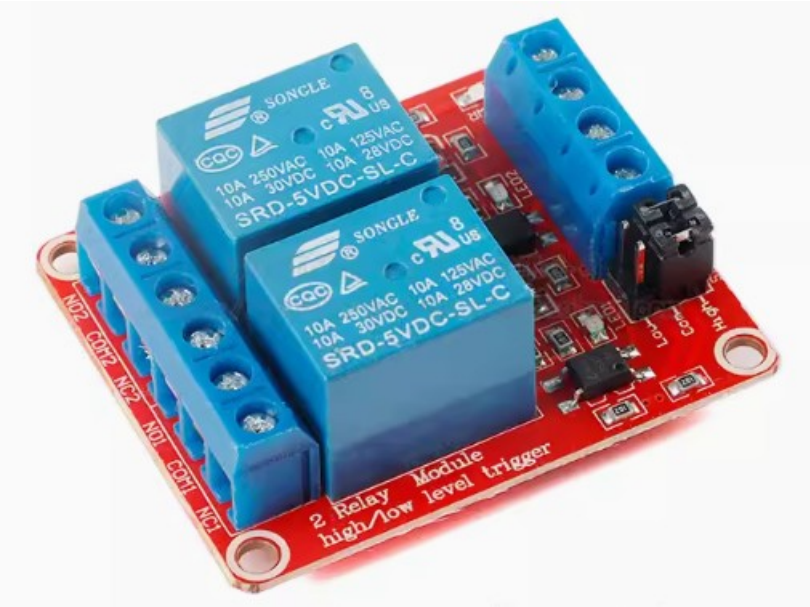

Figure 7: Relay module

It is a two way relay module but only one relay is used in iSEB door. It is a 5v relay because the system is working in 5v. It able to configure to receive high/low signal to turn on with the jumper. For iSEB door, we are using send low signal to turn on the relay to ensure the current require to source the opto isolator is enough.

7 Adruino UNO with iSEB expansion card

ISEB door is using Arduino UNO with iSEB expansion card to record the usage of the door , sound the buzzer when the door is not close and display the cout and status of the door. The figure below is showing the buzzer and switch of the iSEB expansion card.

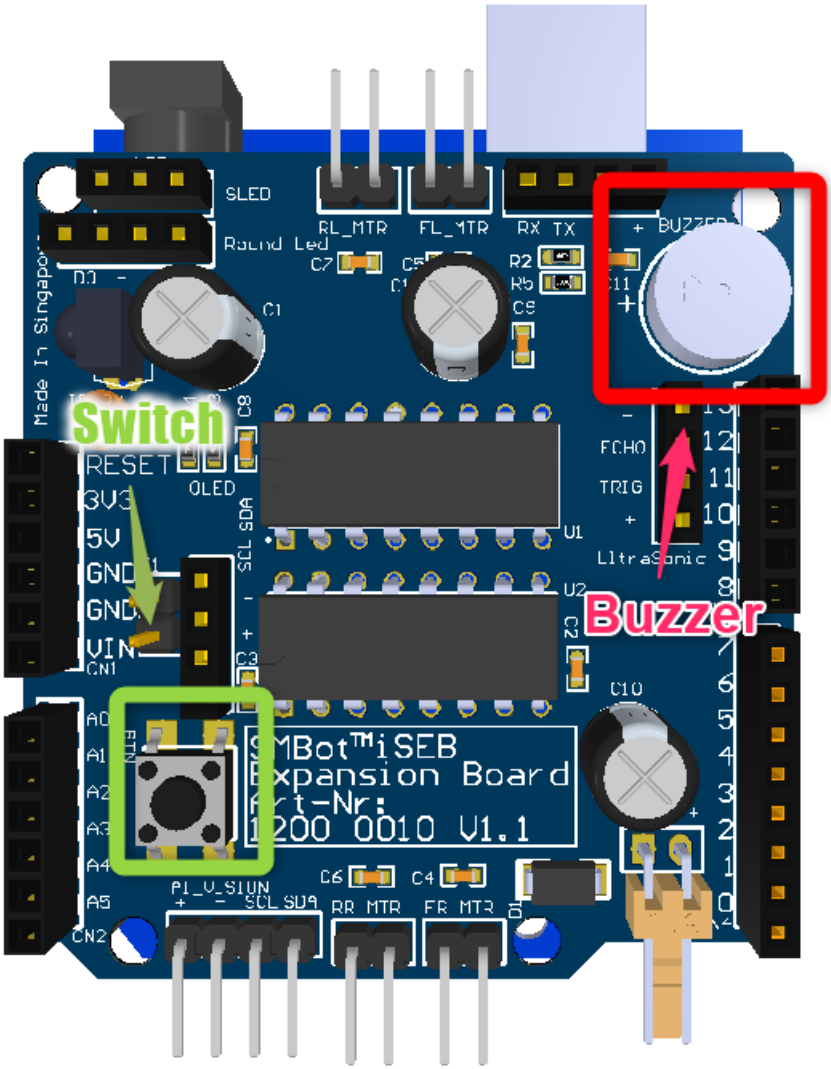

Figure 8: Buzzer and switch of the iSEB expansion card.

The user can clear the count of the usage of the iSEB door by pressing the switch of iSEB expansion card. The cout of the usage is stored in the eeprom hence the value is retained during power cycle.

ISEB door will display the count of the usage and status of the door with LCD. The figure below is showing the LCD of iSEB expansion card.

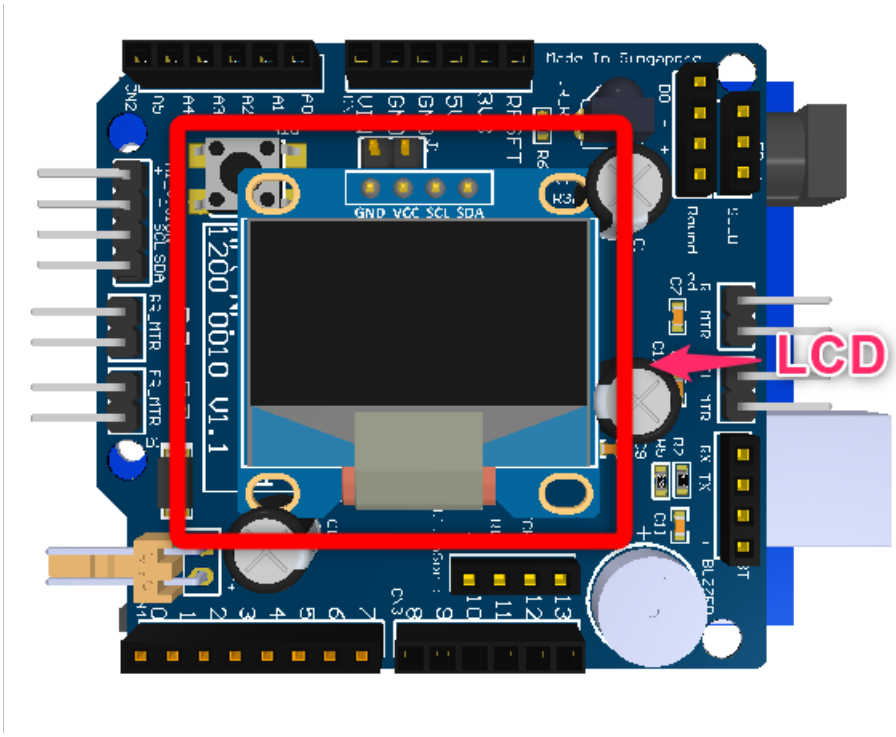

Figure 9: LCD of iSEB expansion card

Beside that arduino UNO is used to detect the status of the door with limit switch. The figure below is showing the limit switch .

Figure 10: limit switch

7.2 Arduino UNO with iSEB expansion card pinout

We will only list down the pinout invovled in iSEB door.

Pin	Function	Pin	Function
D0	UART Rx	D10	N/A
D1	UART Tx	D11	Buzzer
D2	N/A	D12	N/A
D3	N/A	D13	N/A
D4	N/A	A0	N/A
D5	N/A	A1	N/A
D6	N/A	A2	Button Input
D7	N/A	A3	Limit switch
D8	N/A	A4	N/A
D9	N/A	A5	N/A

Table 1 Expansion board pinout

7.2 Arduino Uno Coding

The sample code for the arduin UNO with iSEB exapnsoin card will be available at https://github.com/bingran/iSEB-Door.

The following is the sample code

```
#include "U8glib.h"
#include "pitches.h"
#include <EEPROM.h>

int addr = 0;

/* Pinout Definition */
#define buzzerPin 11
#define btnPin A2
#define signalPin  A3

U8GLIB_SSD1306_128X64 u8g(U8G_I2C_OPT_NONE|U8G_I2C_OPT_DEV_0);      // I2C / TWI

int buzzerTimeout = 0;
int ledTimeout = 0;
int doorCount  = 0;
int signalDebounce = 50;
char buffer[15];

/* button */
bool bButton = false;
bool bSignal = true; /* by default put close to prevent buzzer beep */

/* buzzer */
bool bfBuzzer = 1;
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};

void buzzerPlay ()
{
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    // to calculate the note duration, take one second divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(buzzerPin, melody[thisNote], noteDuration);
    // to distinguish the notes, set a minimum time between them.
```

```
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(buzzerPin);
  }

}

void draw(void) {
  // graphic commands to redraw the complete screen should be placed here
  u8g.setFont(u8g_font_unifont);
  //u8g.setFont(u8g_font_osb21);
  if(false == bSignal)
  {
    u8g.drawStr( 0, 22, "Door: Open" );
    buzzerTimeout++;
    if(20 < buzzerTimeout)
    {
      buzzerTimeout = 0;
    }
    else if(10 < buzzerTimeout)
    {
      Serial.println("No Buzzer");
      // stop the tone playing:
      noTone(buzzerPin);
    }
    else
    {
      Serial.println("Buzzer");
      tone(buzzerPin, melody[3], 2000);
    }
  }
  else
  {
    u8g.drawStr( 0, 22, "Door: Close" );
    buzzerTimeout = 0;
    noTone(buzzerPin);
  }
  sprintf(buffer, "Count :%d", doorCount);
  u8g.drawStr( 0, 44, buffer );

}

void setup(void) {
  Serial.begin(9600);
  // flip screen, if required
  // u8g.setRot180();
```

```
  // set SPI backup if required
  //u8g.setHardwareBackup(u8g_backup_avr_spi);

  // assign default color value
  if ( u8g.getMode() == U8G_MODE_R3G3B2 ) {
    u8g.setColorIndex(255);     // white
  }
  else if ( u8g.getMode() == U8G_MODE_GRAY2BIT ) {
    u8g.setColorIndex(3);         // max intensity
  }
  else if ( u8g.getMode() == U8G_MODE_BW ) {
    u8g.setColorIndex(1);         // pixel on
  }
  else if ( u8g.getMode() == U8G_MODE_HICOLOR ) {
    u8g.setHiColorByRGB(255,255,255);
  }


  pinMode(btnPin, INPUT);  // sets the digital pin 13 as output
  pinMode(signalPin, INPUT_PULLUP);    // sets the digital pin 7 as input

  buzzerPlay();

  doorCount = EEPROM.read(addr);
}

void loop(void) {

  if (bButton != digitalRead(btnPin)) {
    bButton = digitalRead(btnPin);
    if (0 == bButton) {
      Serial.println("Button is pressed");
      doorCount = 0;
      EEPROM.write(addr, doorCount);
    } else {
      Serial.println("Button is released");
    }
  }

  if (bSignal != digitalRead(signalPin))
  {
    signalDebounce--;
    if(0 == signalDebounce)
    {
      bSignal = digitalRead(signalPin);
      if (0 == bSignal) {
        Serial.println("Door is open");
```

```
    }
    else {
      doorCount++;
      EEPROM.write(addr, doorCount);
      Serial.println("Door is close");
    }
  }
 }
 else
 {
  signalDebounce = 50;
 }

 if(0 != ledTimeout)
 {
  ledTimeout--;
 }

 if(0 == ledTimeout)
 {
  ledTimeout = 50;
  // picture loop
  u8g.firstPage();
  do {
    draw();
  } while( u8g.nextPage() );
 }
 // rebuild the picture after some delay
 delay(1);
}
```