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

[Abstract 2](#__RefHeading___Toc1131_1720759766)

[Document History 2](#__RefHeading___Toc1133_1720759766)

[Contents 3](#__RefHeading___Toc1135_1720759766)

[Table of Figures 4](#__RefHeading___Toc1137_1720759766)

[Table of table 4](#__RefHeading___Toc1139_1720759766)

[1 Introduction 5](#__RefHeading___Toc1141_1720759766)

[2 iSEB Door 5](#__RefHeading___Toc1143_1720759766)

[3 Door Control Unit 6](#__RefHeading___Toc2639_2924196363)

[4 Voltage regulator module 8](#__RefHeading___Toc2641_2924196363)

[5 ESP32-S3-EYE 9](#__RefHeading___Toc2643_2924196363)

[5.1 Coding 10](#__RefHeading___Toc930_582364585)

[5.1.1 Code for Main task 10](#__RefHeading___Toc932_582364585)

[5.1.2 Code for face recognition task 11](#__RefHeading___Toc934_582364585)

[5.1.3 Code for Camera task 18](#__RefHeading___Toc936_582364585)

[5.1.3 Code for LCD task 20](#__RefHeading___Toc938_582364585)

[5.4 Code for button task 24](#__RefHeading___Toc940_582364585)

[6 Relay module 27](#__RefHeading___Toc2645_2924196363)

[7 Adruino UNO with iSEB expansion card 28](#__RefHeading___Toc2647_2924196363)

[7.2 Arduino UNO with iSEB expansion card pinout 30](#__RefHeading___Toc2649_2924196363)

[7.2 Arduino Uno Coding 31](#__RefHeading___Toc942_582364585)

# Table of Figures

[Figure 1: Overview of ISEB Door 5](#Figure!0|sequence)

[Figure 2: DSW-60 from Deper 6](#Figure!11|sequence)

[Figure 3: Connector 1 6](#Figure!1|sequence)

[Figure 4: Detail of setting panels 7](#Figure!2|sequence)

[Figure 5: LM2596 module 8](#Figure!3|sequence)

[Figure 6: ESP32-S3-EYE 9](#Figure!4|sequence)

[Figure 7: Relay module 10](#Figure!5|sequence)

[Figure 8: Buzzer and switch of the iSEB expansion card. 11](#Figure!6|sequence)

[Figure 9: LCD of iSEB expansion card 12](#Figure!7|sequence)

[Figure 10: limit switch 12](#Figure!8|sequence)

# Table of table

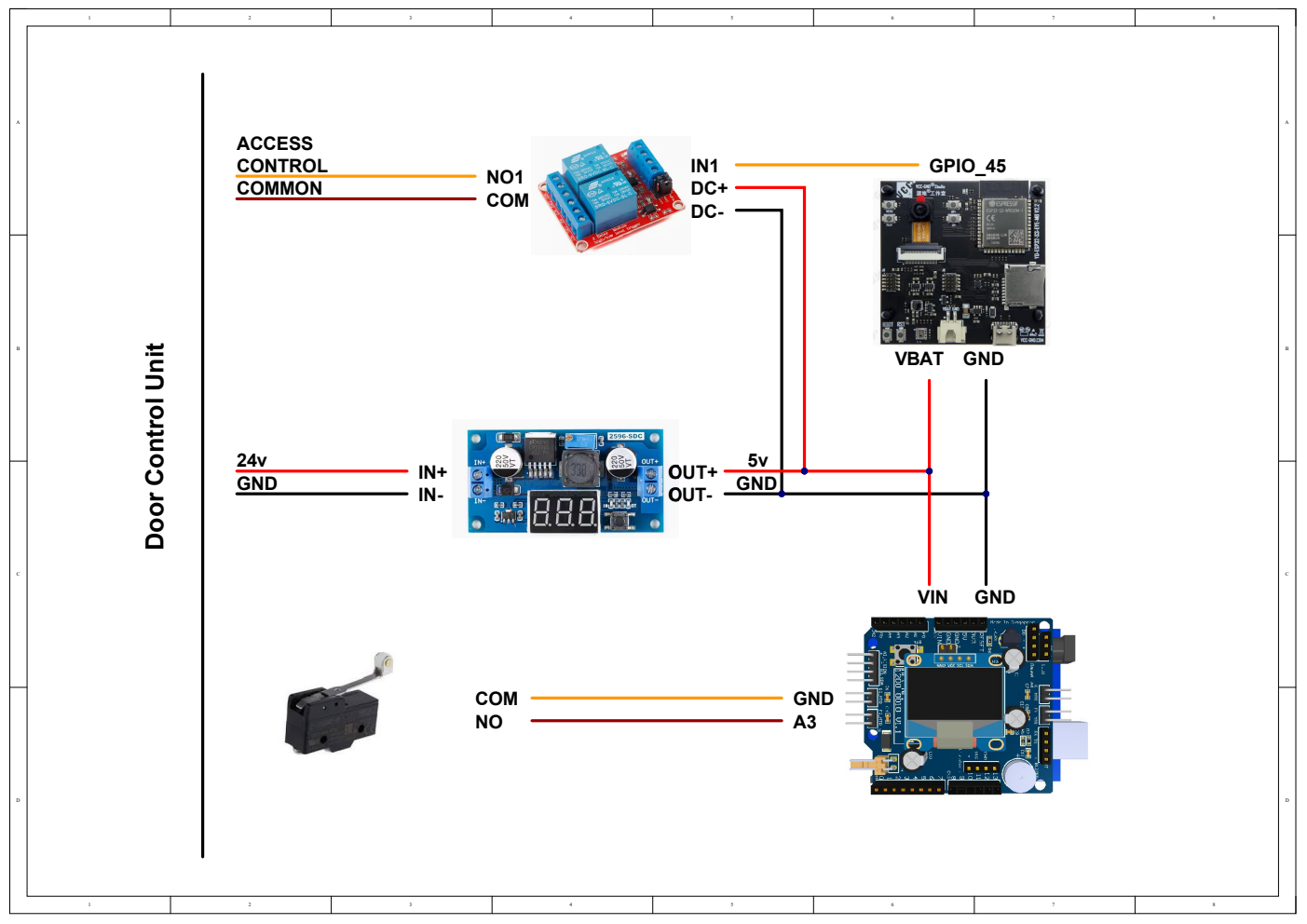
[Table 1 Expansion board pinout 13](#Table!0|sequence)

# 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.

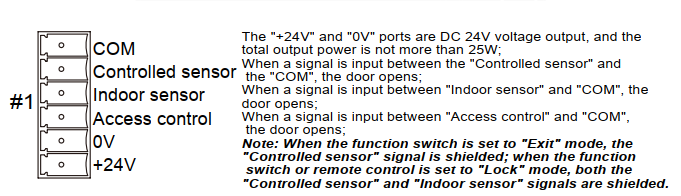
Figure 1: Overview of ISEB Door

# 3 Door Control Unit

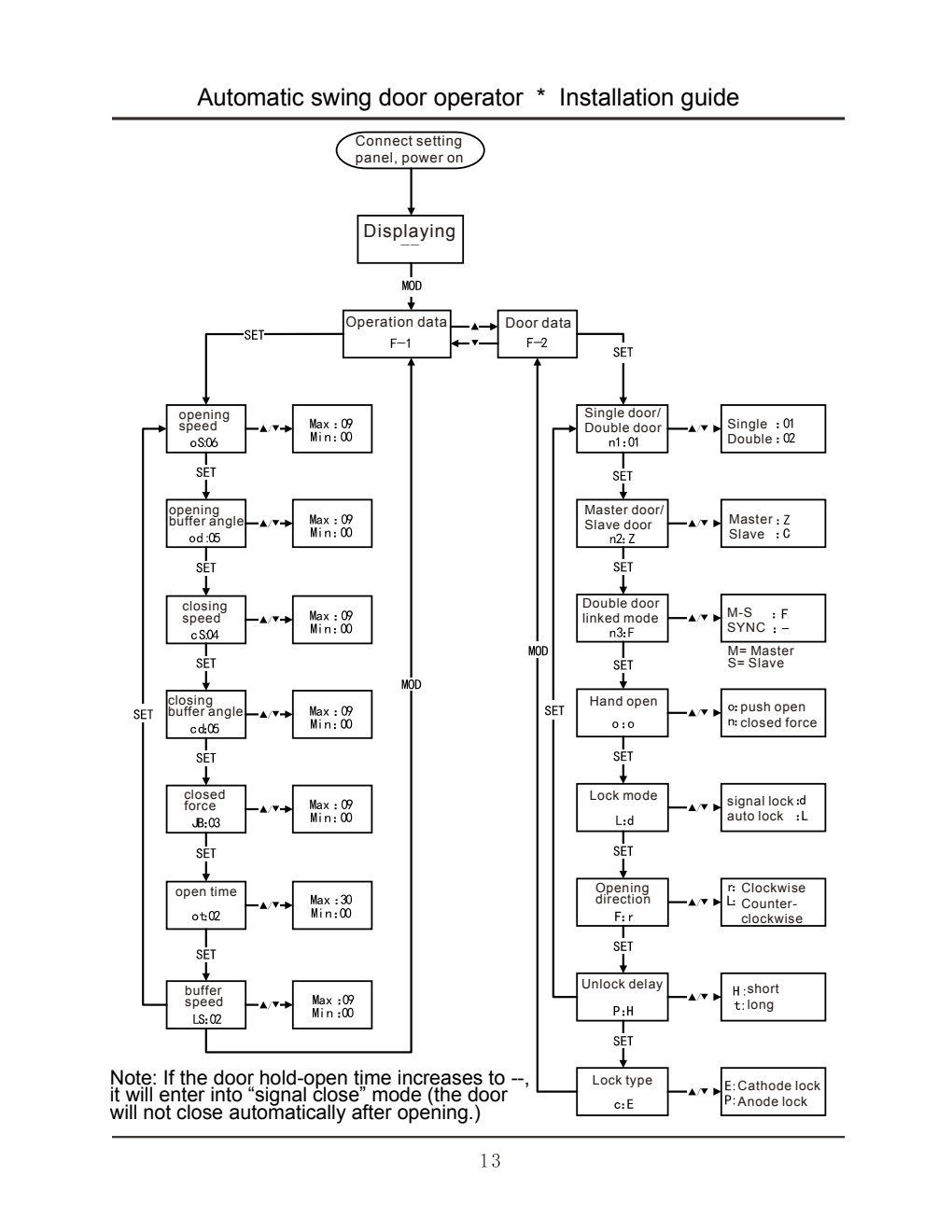
ISEB door is using the DSW-60 from Deper. It is a slim and small size. It is a automatic swing door openers with motion sensors. The figure below is showing DSW-60 from Deper.

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 2: DSW-60 from Deper

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.

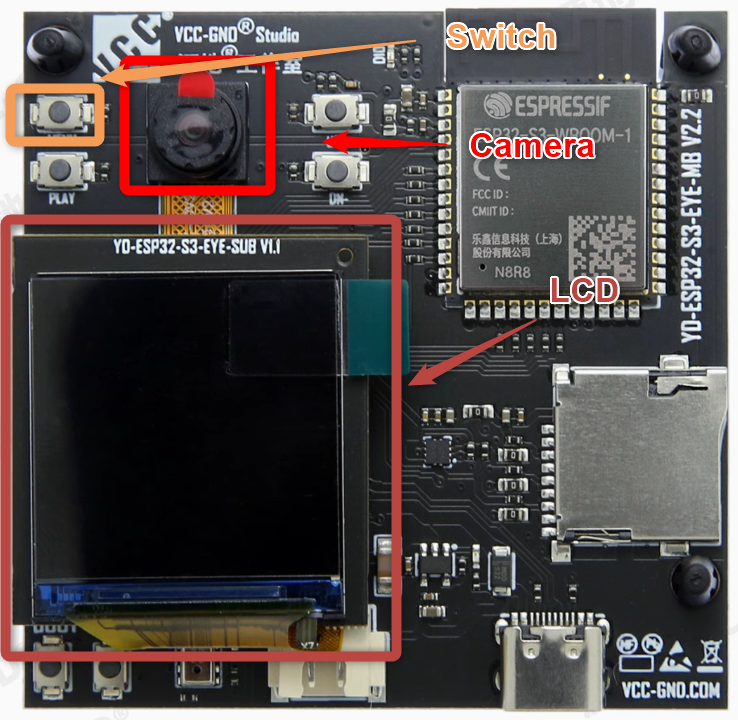
Figure 5: LM2596 module

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()

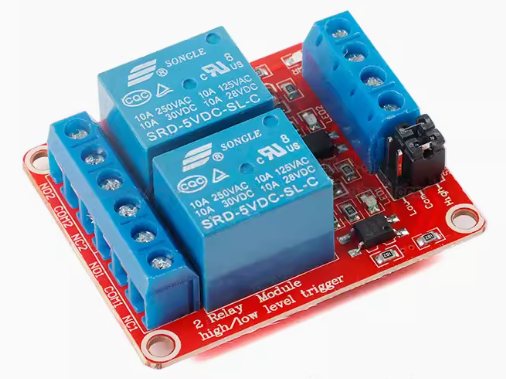
{

xTaskCreatePinnedToCore((TaskFunction\_t)task, TAG, 3 \* 1024, this, 5, NULL, 0);

}

# 6 Relay module

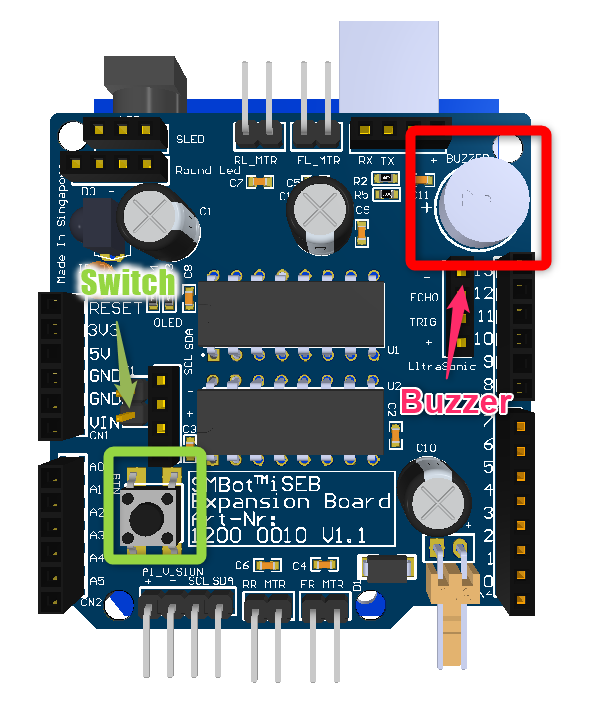
ISEB door is using relay module to trasnfer signal from ESP32-S3-EYE to door control unit. This is because door control unit is working at higher voltage. The gpio of ESP32-S3 is only tolerate 3.3v, hence, a relay module is require to send signal from ESP32-S3-EYE to door control unit. The figure below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

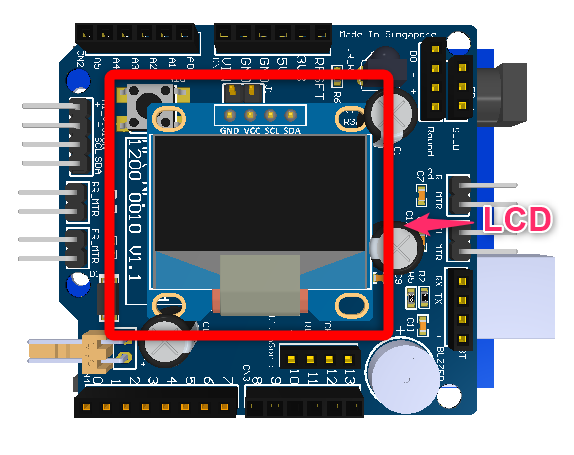
# 7 Adruino UNO with iSEB expansion card

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);  
}