# Hardware Embedded Code

```
#include <Ahmed-elmasry-1234566789-project-1_inferencing.h> // modify
with your project title,
Replace the xxxx
#include "edge-impulse-sdk/dsp/image/image.hpp"
#include "esp_camera.h"
// #define CAMERA_MODEL_ESP_EYE  // Has PSRAM
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
#if defined(CAMERA_MODEL_ESP_EYE)
#define PWDN_GPIO_NUM -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 4
#define SIOD_GPIO_NUM 18
#define SIOC_GPIO_NUM 23
#define Y9_GPIO_NUM 36
#define Y8_GPIO_NUM 37
#define Y7_GPIO_NUM 38
#define Y6_GPIO_NUM 39
#define Y5_GPIO_NUM 35
#define Y4_GPIO_NUM 14
#define Y3_GPIO_NUM 13
#define Y2_GPIO_NUM 34
#define VSYNC_GPIO_NUM 5
#define HREF_GPIO_NUM 27
#define PCLK_GPIO_NUM 25
#elif defined(CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM 32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 0
#define SIOD_GPIO_NUM 26
#define SIOC_GPIO_NUM 27
#define Y9_GPIO_NUM 35
#define Y8_GPIO_NUM 34
#define Y7_GPIO_NUM 39
#define Y6_GPIO_NUM 36
```

```
#define Y5_GPIO_NUM 21
#define Y4_GPIO_NUM 19
#define Y3_GPIO_NUM 18
#define Y2_GPIO_NUM 5
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM 23
#define PCLK_GPIO_NUM 22
#else
#error "Camera model not selected"
#endif
/* Constant defines -------------------------------------------------
------ */
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS 320
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS 240
#define EI_CAMERA_FRAME_BYTE_SIZE 3
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
// ESP32-CAM doesn't have dedicated i2c pins, so we define our own.
Let's choose 15 and 14
#define I2C_SDA 15
#define I2C_SCL 14
TwoWire I2Cbus = TwoWire(0);
// Display defines
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &I2Cbus,
OLED_RESET);
/* Private variables -----------------------------------------------
------ */
static bool debug_nn = false;  // Set this to true to see e.g.
features generated from the raw signal
static bool is_initialised = false;
uint8_t *snapshot_buf;  //points to the output of the capture
static camera_config_t camera_config = {
.pin_pwdn = PWDN_GPIO_NUM,
.pin_reset = RESET_GPIO_NUM,
.pin_xclk = XCLK_GPIO_NUM,
.pin_sscb_sda = SIOD_GPIO_NUM,
.pin_sscb_scl = SIOC_GPIO_NUM,
```

```
.pin_d7 = Y9_GPIO_NUM,
.pin_d6 = Y8_GPIO_NUM,
.pin_d5 = Y7_GPIO_NUM,
.pin_d4 = Y6_GPIO_NUM,
.pin_d3 = Y5_GPIO_NUM,
.pin_d2 = Y4_GPIO_NUM,
.pin_d1 = Y3_GPIO_NUM,
.pin_d0 = Y2_GPIO_NUM,
.pin_vsync = VSYNC_GPIO_NUM,
.pin_href = HREF_GPIO_NUM,
.pin_pclk = PCLK_GPIO_NUM,
//XCLK 20MHz or 10MHz for OV2640 double FPS (Experimental)
.xclk_freq_hz = 20000000,
.ledc_timer = LEDC_TIMER_0,
.ledc_channel = LEDC_CHANNEL_0,
.pixel_format = PIXFORMAT_JPEG,  //YUV422,GRAYSCALE,RGB565,JPEG
.frame_size = FRAMESIZE_QVGA,    //QQVGA-UXGA Do not use sizes above
QVGA when not JPEG
.jpeg_quality = 12,  //0-63 lower number means higher quality
.fb_count = 1,
//if more than one, i2s runs in continuous mode. Use only with JPEG
.fb_location = CAMERA_FB_IN_PSRAM,
.grab_mode = CAMERA_GRAB_WHEN_EMPTY,
};
/* Function definitions -------------------------------------------
--------- */
bool ei_camera_init(void);
void ei_camera_deinit(void);
bool ei_camera_capture(uint32_t img_width, uint32_t img_height,
uint8_t *out_buf);
/**
* @brief
*/
Arduino setup function
void setup() {
// put your setup code here, to run once:
Serial.begin(115200);
// Initialize I2C with our defined pins
I2Cbus.begin(I2C_SDA, I2C_SCL, 100000);
// SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V
internally
if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
```

```
Serial.printf("SSD1306 OLED display failed to initalize.\nCheck that
display SDA is connected to pin %d
and SCL connected to pin %d\n", I2C_SDA, I2C_SCL);
while (true)
;
}
//comment out the below line to start inference immediately after
upload
while (!Serial)
;
Serial.println("Edge Impulse Inferencing Demo");
if (ei_camera_init() == false) {
ei_printf("Failed to initialize Camera!\r\n");
} else {
ei_printf("Camera initialized\r\n");
}
ei_printf("\nStarting continious inference in 2 seconds...\n");
display.clearDisplay();
display.setCursor(0, 0);
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);
display.print("Starting continious\n inference in\n 2 seconds...");
display.display();
ei_sleep(2000);
display.clearDisplay();
}
/**
* @brief
*
Get data and run inferencing
* @param[in]  debug  Get debug info if true
*/
void loop() {
display.clearDisplay();
// instead of wait_ms, we'll wait on the signal, this allows threads
to cancel us...
if (ei_sleep(5) != EI_IMPULSE_OK) {
return;
}
snapshot_buf = (uint8_t *)malloc(EI_CAMERA_RAW_FRAME_BUFFER_COLS *
EI_CAMERA_RAW_FRAME_BUFFER_ROWS * EI_CAMERA_FRAME_BYTE_SIZE);
// check if allocation was successful
```

```
if (snapshot_buf == nullptr) {
ei_printf("ERR: Failed to allocate snapshot buffer!\n");
return;
}
ei::signal_t signal;
signal.total_length = EI_CLASSIFIER_INPUT_WIDTH *
EI_CLASSIFIER_INPUT_HEIGHT;
signal.get_data = &ei_camera_get_data;
if (ei_camera_capture((size_t)EI_CLASSIFIER_INPUT_WIDTH,
(size_t)EI_CLASSIFIER_INPUT_HEIGHT,
snapshot_buf) == false) {
ei_printf("Failed to capture image\r\n");
free(snapshot_buf);
return;
}
// Run the classifier
ei_impulse_result_t result = { 0 };
EI_IMPULSE_ERROR err = run_classifier(&signal, &result, debug_nn);
if (err != EI_IMPULSE_OK) {
ei_printf("ERR: Failed to run classifier (%d)\n", err);
return;
}
// print the predictions
ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly:
%d ms.): \n",
result.timing.dsp, result.timing.classification,
result.timing.anomaly);
#if EI_CLASSIFIER_OBJECT_DETECTION == 1
bool bb_found = result.bounding_boxes[0].value > 0;
for (size_t ix = 0; ix < result.bounding_boxes_count; ix++) {
auto bb = result.bounding_boxes[ix];
if (bb.value == 0) {
continue;
}
ei_printf("    %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\n",
bb.label, bb.value, bb.x, bb.y, bb.width,
bb.height);
display.setCursor(0, 20 * ix);
display.setTextSize(2);
display.setTextColor(SSD1306_WHITE);
display.print(bb.label);
display.print("-");
```

```
display.print(int((bb.value)*100));
display.print("%");
display.display();
}
if (!bb_found) {
ei_printf("    No objects found\n");
display.setCursor(0, 16);
display.setTextSize(2);
display.setTextColor(SSD1306_WHITE);
display.print("No objects  found");
display.display();
}
#else
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
ei_printf("    %s: %.5f\n", result.classification[ix].label,
result.classification[ix].value);
}
#endif
#if EI_CLASSIFIER_HAS_ANOMALY == 1
ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif
free(snapshot_buf);
}
/**
* @brief   Setup image sensor & start streaming
*
* @retval  false if initialisation failed
*/
bool ei_camera_init(void) {
if (is_initialised) return true;
#if defined(CAMERA_MODEL_ESP_EYE)
pinMode(13, INPUT_PULLUP);
pinMode(14, INPUT_PULLUP);
#endif
//initialize the camera
esp_err_t err = esp_camera_init(&camera_config);
if (err != ESP_OK) {
Serial.printf("Camera init failed with error 0x%x\n", err);
return false;
}
sensor_t *s = esp_camera_sensor_get();
```

```
// initial sensors are flipped vertically and colors are a bit
saturated
if (s->id.PID == OV3660_PID) {
s->set_vflip(s, 1);
// flip it back
s->set_brightness(s, 1);  // up the brightness just a bit
s->set_saturation(s, 0);  // lower the saturation
}
#if defined(CAMERA_MODEL_M5STACK_WIDE)
s->set_vflip(s, 1);
s->set_hmirror(s, 1);
#elif defined(CAMERA_MODEL_ESP_EYE)
s->set_vflip(s, 1);
s->set_hmirror(s, 1);
s->set_awb_gain(s, 1);
#endif
is_initialised = true;
return true;
}
/**
* @brief
*/
 Stop streaming of sensor data
void ei_camera_deinit(void) {
//deinitialize the camera
esp_err_t err = esp_camera_deinit();
if (err != ESP_OK) {
ei_printf("Camera deinit failed\n");
return;
}
is_initialised = false;
return;
}
/**
* @brief
*
 Capture, rescale and crop image
* @param[in]  img_width     width of output image
* @param[in]  img_height    height of output image
* @param[in]  out_buf
*
pointer to store output image, NULL may be used
```

```
if ei_camera_frame_buffer is to be used for capture and
resize/cropping.
*
* @retval     false if not initialised, image captured, rescaled or
cropped failed
*
*/
bool ei_camera_capture(uint32_t img_width, uint32_t img_height,
uint8_t *out_buf) {
bool do_resize = false;
if (!is_initialised) {
ei_printf("ERR: Camera is not initialized\r\n");
return false;
}
camera_fb_t *fb = esp_camera_fb_get();
if (!fb) {
ei_printf("Camera capture failed\n");
return false;
}
bool converted = fmt2rgb888(fb->buf, fb->len, PIXFORMAT_JPEG,
snapshot_buf);
esp_camera_fb_return(fb);
if (!converted) {
ei_printf("Conversion failed\n");
return false;
}
if ((img_width != EI_CAMERA_RAW_FRAME_BUFFER_COLS)
|| (img_height != EI_CAMERA_RAW_FRAME_BUFFER_ROWS)) {
do_resize = true;
}

  if (do_resize) {
    ei::image::processing::crop_and_interpolate_rgb888(
      out_buf,
      EI_CAMERA_RAW_FRAME_BUFFER_COLS,
      EI_CAMERA_RAW_FRAME_BUFFER_ROWS,
      out_buf,
      img_width,
      img_height);
  }
```

```
  return true;
}

static int ei_camera_get_data(size_t offset, size_t length, float
*out_ptr) {
  // we already have a RGB888 buffer, so recalculate offset into pixel
index
  size_t pixel_ix = offset * 3;
  size_t pixels_left = length;
  size_t out_ptr_ix = 0;

  while (pixels_left != 0) {
    // Swap BGR to RGB here
    // due to https://github.com/espressif/esp32-camera/issues/379
    out_ptr[out_ptr_ix] = (snapshot_buf[pixel_ix + 2] << 16) +
(snapshot_buf[pixel_ix + 1] << 8) +
snapshot_buf[pixel_ix];

    // go to the next pixel
    out_ptr_ix++;
pixel_ix += 3;
pixels_left--;
}
// and done!
return 0;
}
#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR !=
EI_CLASSIFIER_SENSOR_CAMERA
#error "Invalid model for current sensor"
#endif
```