北京大學
PEKING UNIVERSITY

K210 FACE DETECT实验

汇报人：黄森

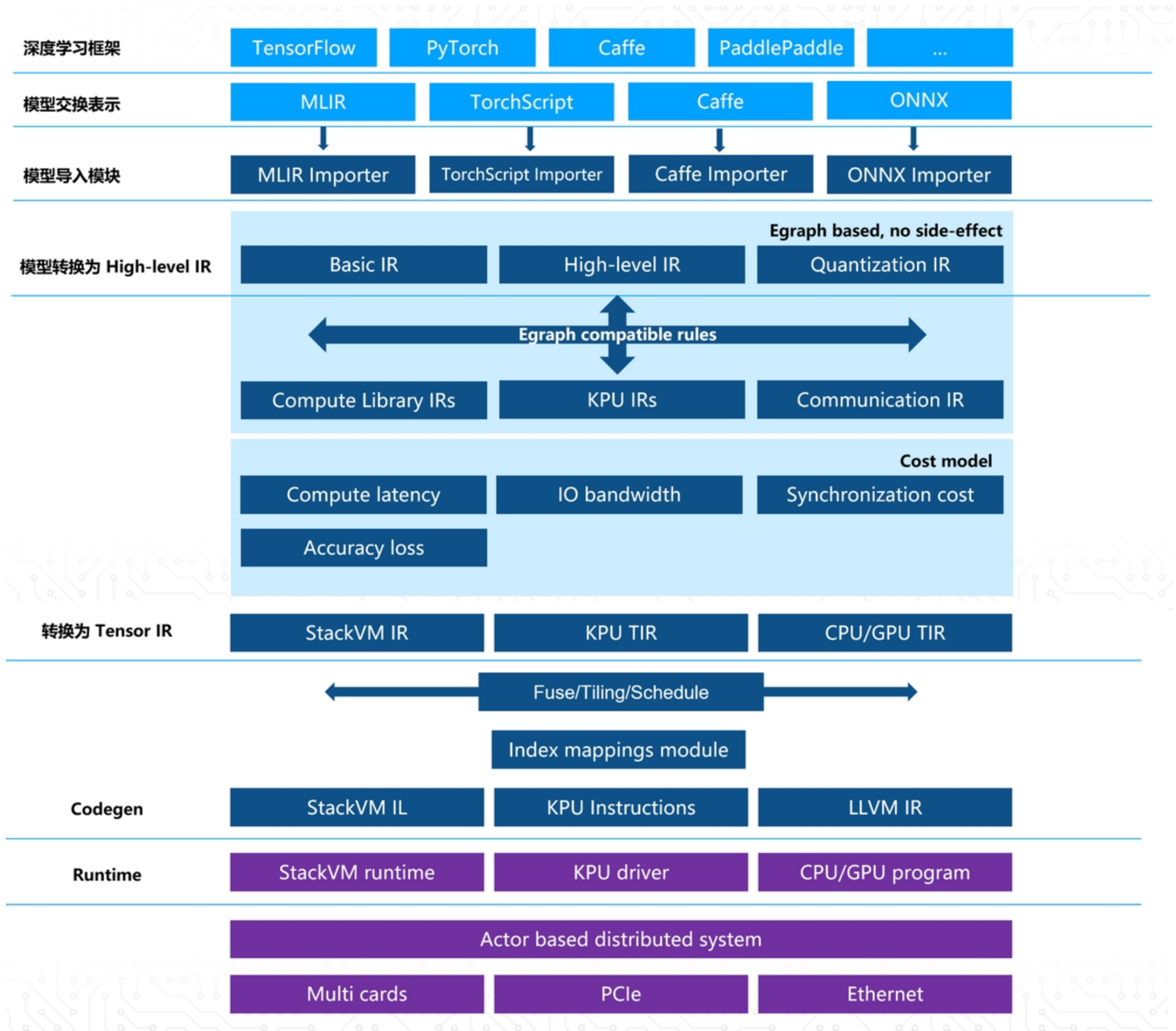
2024/12/6

目录

- K210模型转换工具
- FACE DETECT实验例程介绍
- 代码烧录和实验结果

K210模型转换工具--nncase

➤ nncase架构&转换示例



深度学习框架: TensorFlow | PyTorch | Caffe | PaddlePaddle | ...

模型交换表示: MLIR | TorchScript | Caffe | ONNX

模型导入模块: MLIR Importer | TorchScript Importer | Caffe Importer | ONNX Importer

模型转换为 High-level IR: Egraph based, no side-effect

Basic IR | High-level IR | Quantization IR

Egraph compatible rules

Compute Library IRs | KPU IRs | Communication IR

Cost model

Compute latency | IO bandwidth | Synchronization cost

Accuracy loss

转换为 Tensor IR: StackVM IR | KPU TIR | CPU/GPU TIR

Fuse/Tiling/Schedule

Index mappings module

Codegen: StackVM IL | KPU Instructions | LLVM IR

Runtime: StackVM runtime | KPU driver | CPU/GPU program

Actor based distributed system

Multi cards | PCIe | Ethernet

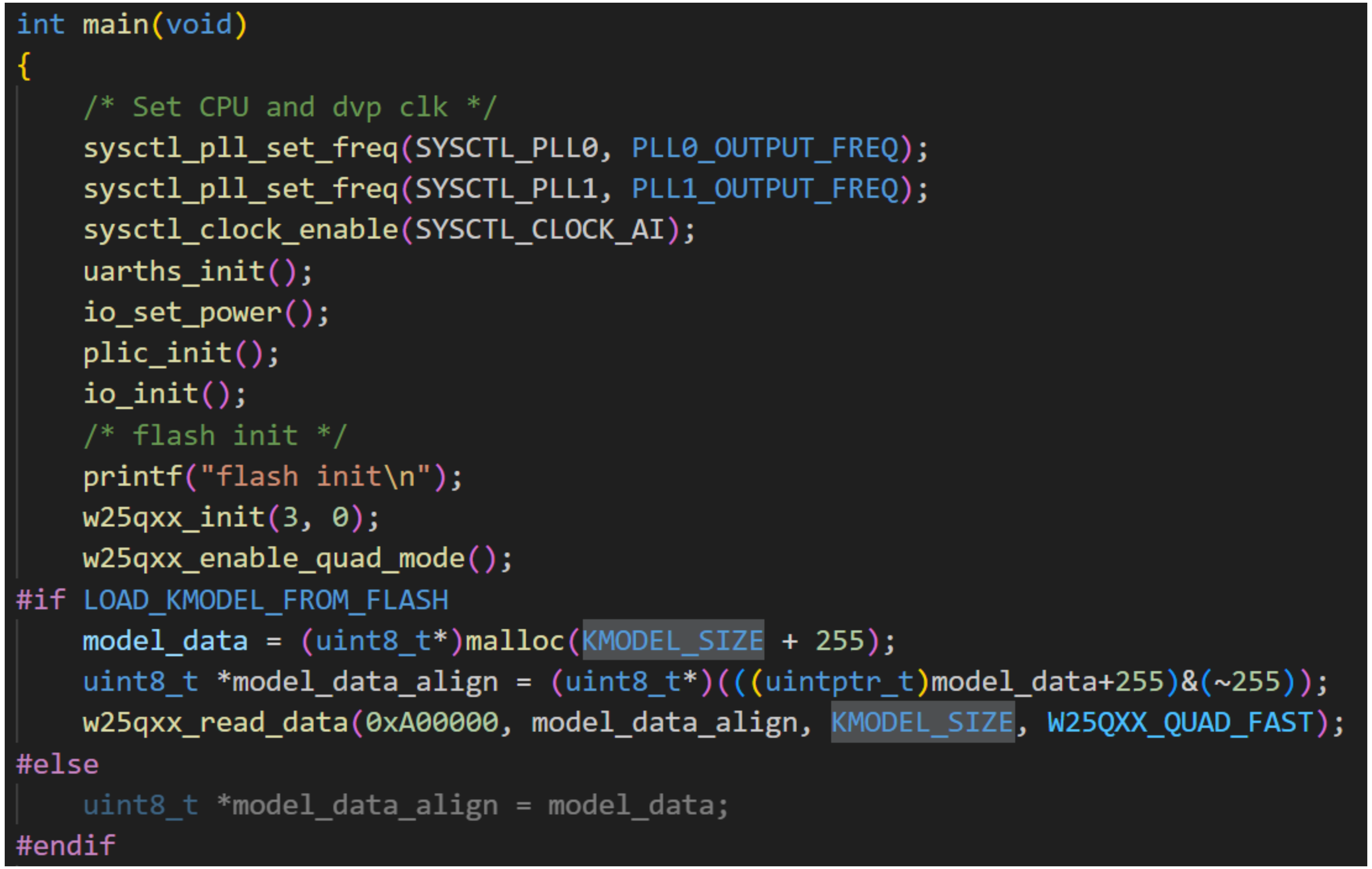
- .tflite转换为kmodel

```
./ncc compile ./model/mobilenetv1_1.0.tflite
./model/mobilenetv1_1.0.kmodel -i tflite -o kmodel --dataset ./dataset/
```

- 模拟运行kmodel

```
./ncc infer ./model/mobilenetv1_1.0.kmodel ./result
--dataset ./testset
```

FACE DETECT实验例程介绍

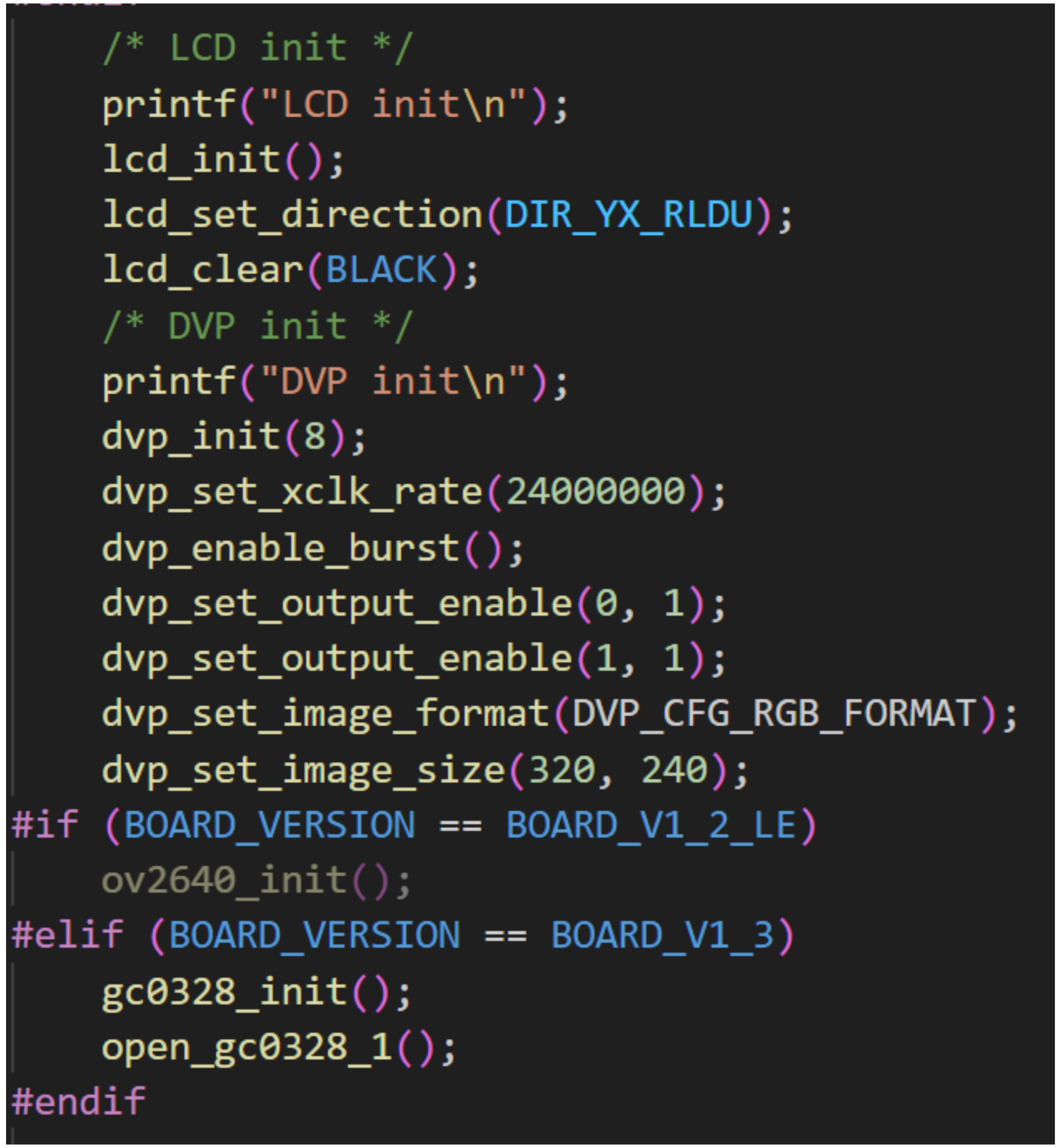


```c
int main(void)
{
    /* Set CPU and dvp clk */
    sysctl_pll_set_freq(SYSCTL_PLL0, PLL0_OUTPUT_FREQ);
    sysctl_pll_set_freq(SYSCTL_PLL1, PLL1_OUTPUT_FREQ);
    sysctl_clock_enable(SYSCTL_CLOCK_AI);
    uarths_init();
    io_set_power();
    plic_init();
    io_init();
    /* flash init */
    printf("flash init\n");
    w25qxx_init(3, 0);
    w25qxx_enable_quad_mode();
#if LOAD_KMODEL_FROM_FLASH
    model_data = (uint8_t*)malloc(KMODEL_SIZE + 255);
    uint8_t *model_data_align = (uint8_t*)(((uintptr_t)model_data+255)&(~255));
    w25qxx_read_data(0xA00000, model_data_align, KMODEL_SIZE, W25QXX_QUAD_FAST);
#else
    uint8_t *model_data_align = model_data;
#endif
```

```c
#define KMODEL_SIZE (380 * 1024)

/* Types for `void *' pointers.  */
#if __WORDSIZE == 64
#ifndef __intptr_t_defined
typedef long int                intptr_t;
#define __intptr_t_defined
#endif
typedef unsigned long int       uintptr_t;
#else
#ifndef __intptr_t_defined
typedef int                     intptr_t;
#define __intptr_t_defined
#endif
typedef unsigned int            uintptr_t;
#endif
```

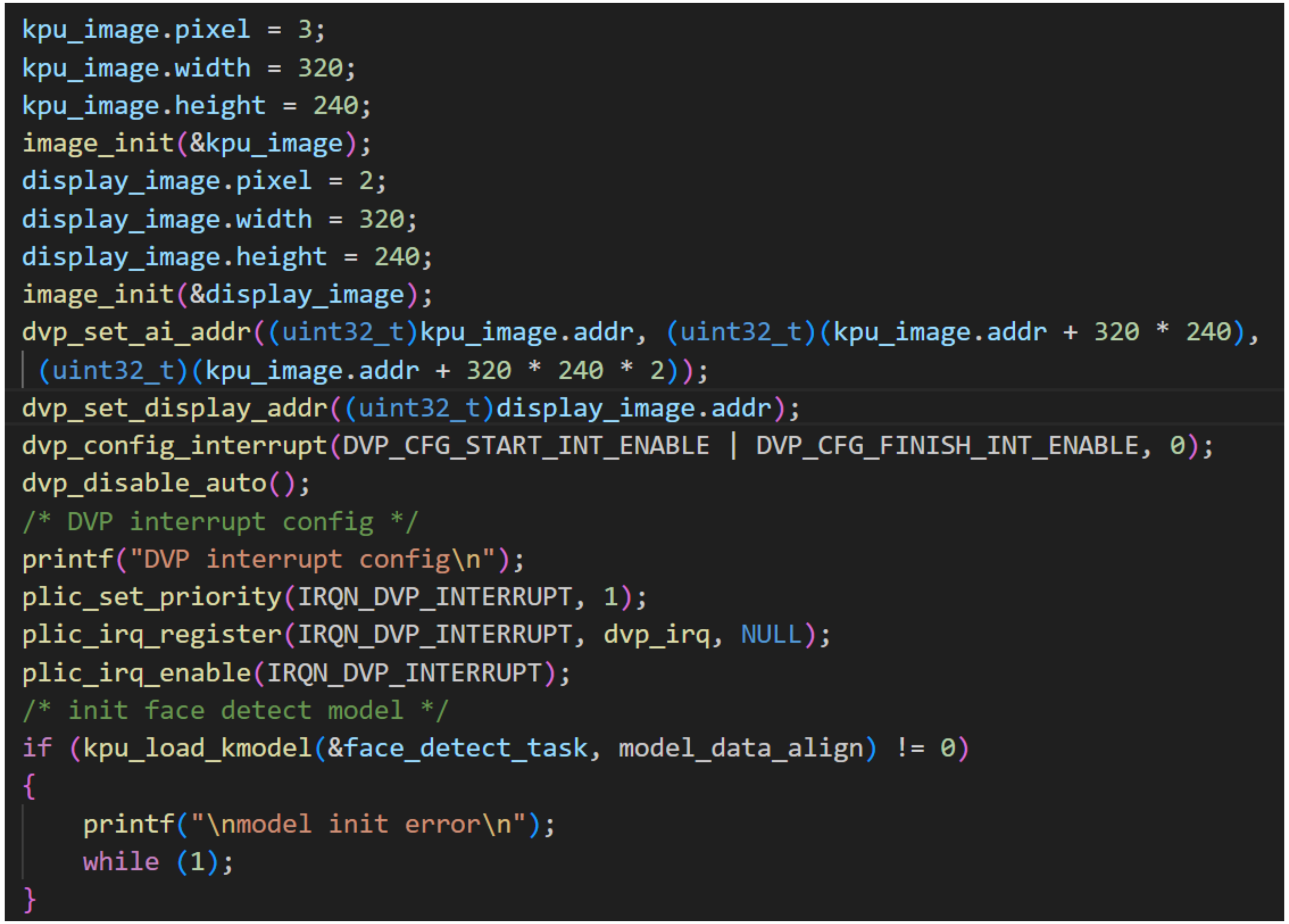
FACE DETECT实验例程介绍



```
    /* LCD init */
    printf("LCD init\n");
    lcd_init();
    lcd_set_direction(DIR_YX_RLDU);
    lcd_clear(BLACK);
    /* DVP init */
    printf("DVP init\n");
    dvp_init(8);
    dvp_set_xclk_rate(24000000);
    dvp_enable_burst();
    dvp_set_output_enable(0, 1);
    dvp_set_output_enable(1, 1);
    dvp_set_image_format(DVP_CFG_RGB_FORMAT);
    dvp_set_image_size(320, 240);
#if (BOARD_VERSION == BOARD_V1_2_LE)
    ov2640_init();
#elif (BOARD_VERSION == BOARD_V1_3)
    gc0328_init();
    open_gc0328_1();
#endif
```

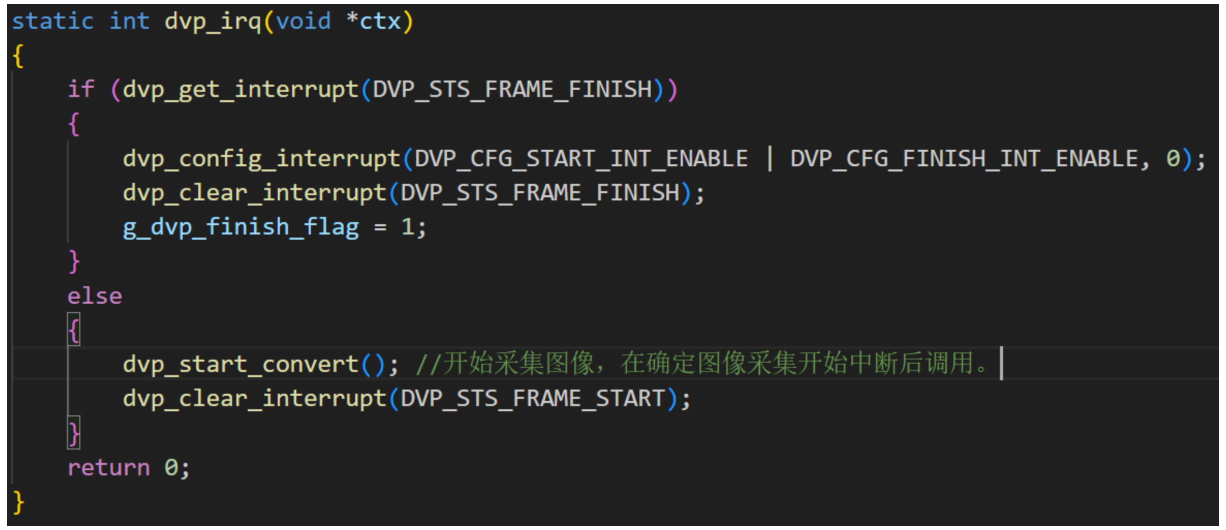
- LCD初始化 & DVP初始化
- DVP使能输出到AI和内存
- 图像格式：RGB
- 图像大小：320 * 240
- 根据开发板版本初始化相机模组

FACE DETECT实验例程介绍



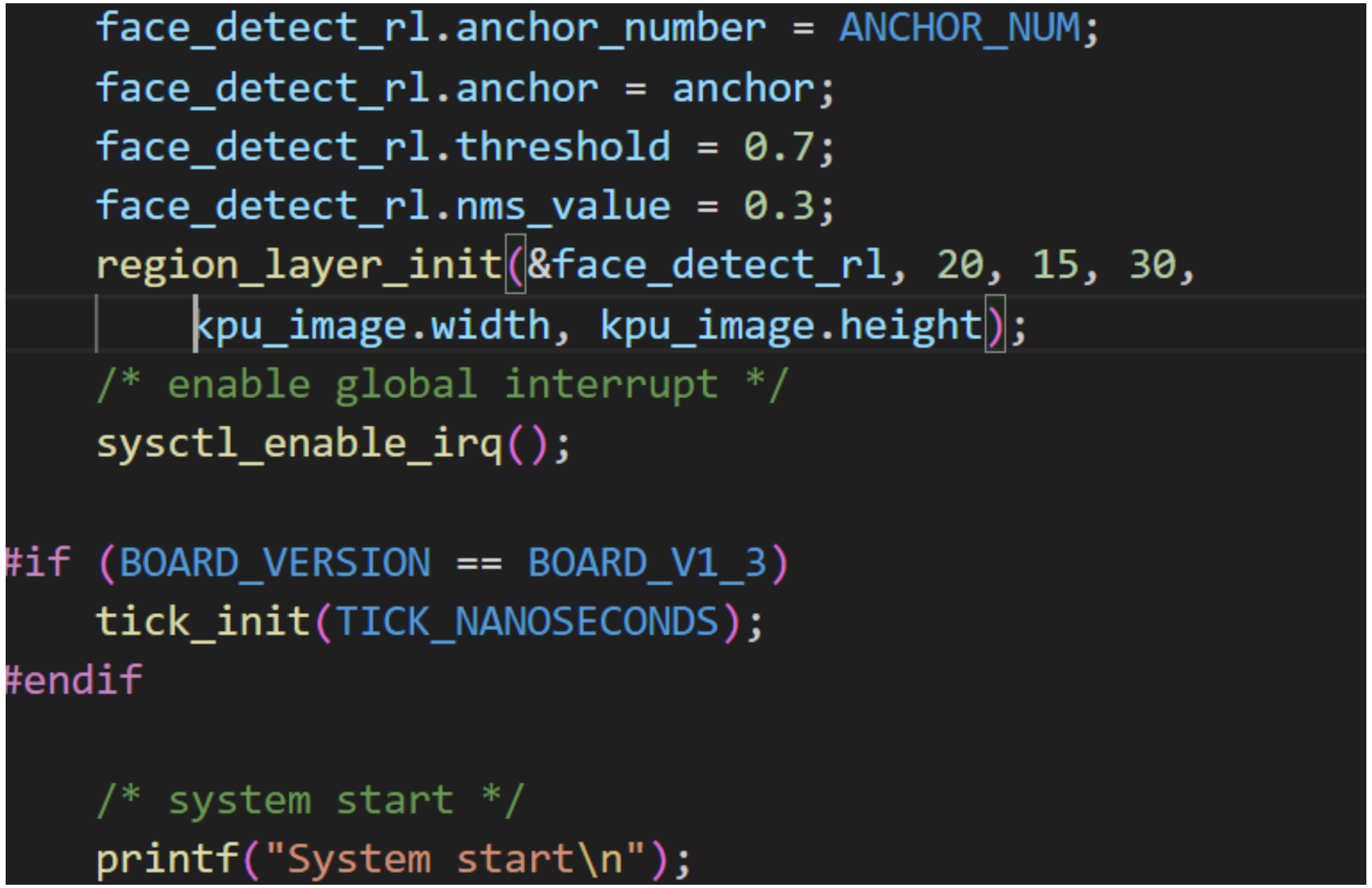
```c
kpu_image.pixel = 3;
kpu_image.width = 320;
kpu_image.height = 240;
image_init(&kpu_image);
display_image.pixel = 2;
display_image.width = 320;
display_image.height = 240;
image_init(&display_image);
dvp_set_ai_addr((uint32_t)kpu_image.addr, (uint32_t)(kpu_image.addr + 320 * 240),
  (uint32_t)(kpu_image.addr + 320 * 240 * 2));
dvp_set_display_addr((uint32_t)display_image.addr);
dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 0);
dvp_disable_auto();
/* DVP interrupt config */
printf("DVP interrupt config\n");
plic_set_priority(IRQN_DVP_INTERRUPT, 1);
plic_irq_register(IRQN_DVP_INTERRUPT, dvp_irq, NULL);
plic_irq_enable(IRQN_DVP_INTERRUPT);
/* init face detect model */
if (kpu_load_kmodel(&face_detect_task, model_data_align) != 0)
{
    printf("\nmodel init error\n");
    while (1);
}
```

- dvp_set_ai_addr()：设置 AI 存放图像的地址，供 AI 模块进行算法处理
- dvp_set_display_addr()：设置采集图像在内存中的存放地址，可以用来显示
- dvp_config_interrupt()：禁用中断
- dvp_disable_auto()：禁用自动接收图像模式
- 设置DVP中断
- 加载kmodel



```c
static int dvp_irq(void *ctx)
{
    if (dvp_get_interrupt(DVP_STS_FRAME_FINISH))
    {
        dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 0);
        dvp_clear_interrupt(DVP_STS_FRAME_FINISH);
        g_dvp_finish_flag = 1;
    }
    else
    {
        dvp_start_convert(); //开始采集图像，在确定图像采集开始中断后调用。
        dvp_clear_interrupt(DVP_STS_FRAME_START);
    }
    return 0;
}
```

FACE DETECT实验例程介绍



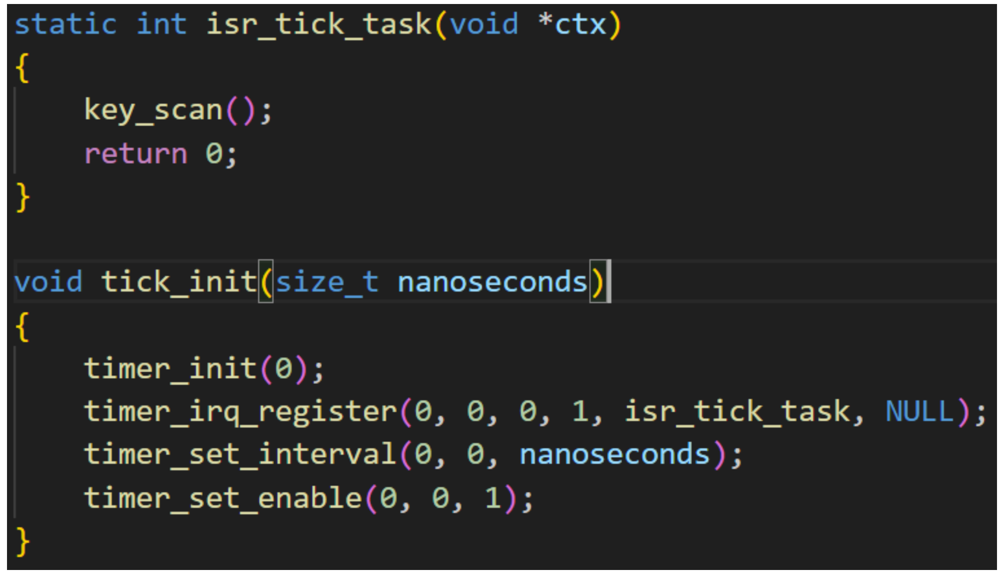
```
    face_detect_rl.anchor_number = ANCHOR_NUM;
    face_detect_rl.anchor = anchor;
    face_detect_rl.threshold = 0.7;
    face_detect_rl.nms_value = 0.3;
    region_layer_init(&face_detect_rl, 20, 15, 30,
        kpu_image.width, kpu_image.height);
    /* enable global interrupt */
    sysctl_enable_irq();

#if (BOARD_VERSION == BOARD_V1_3)
    tick_init(TICK_NANOSECONDS);
#endif

    /* system start */
    printf("System start\n");
```

typedef struct
{
float threshold; **// 置信度阈值**
float nms_value; **// 非极大值抑制(NMS)阈值，用于过滤掉重复检测框**
uint32_t coords; // 每个框的坐标数，通常是4（表示框的x, y, w, h）
uint32_t anchor_number; **// 锚点的数量**
float *anchor; **// 锚点的坐标（每个锚点的宽和高）**
……
} region_layer_t;

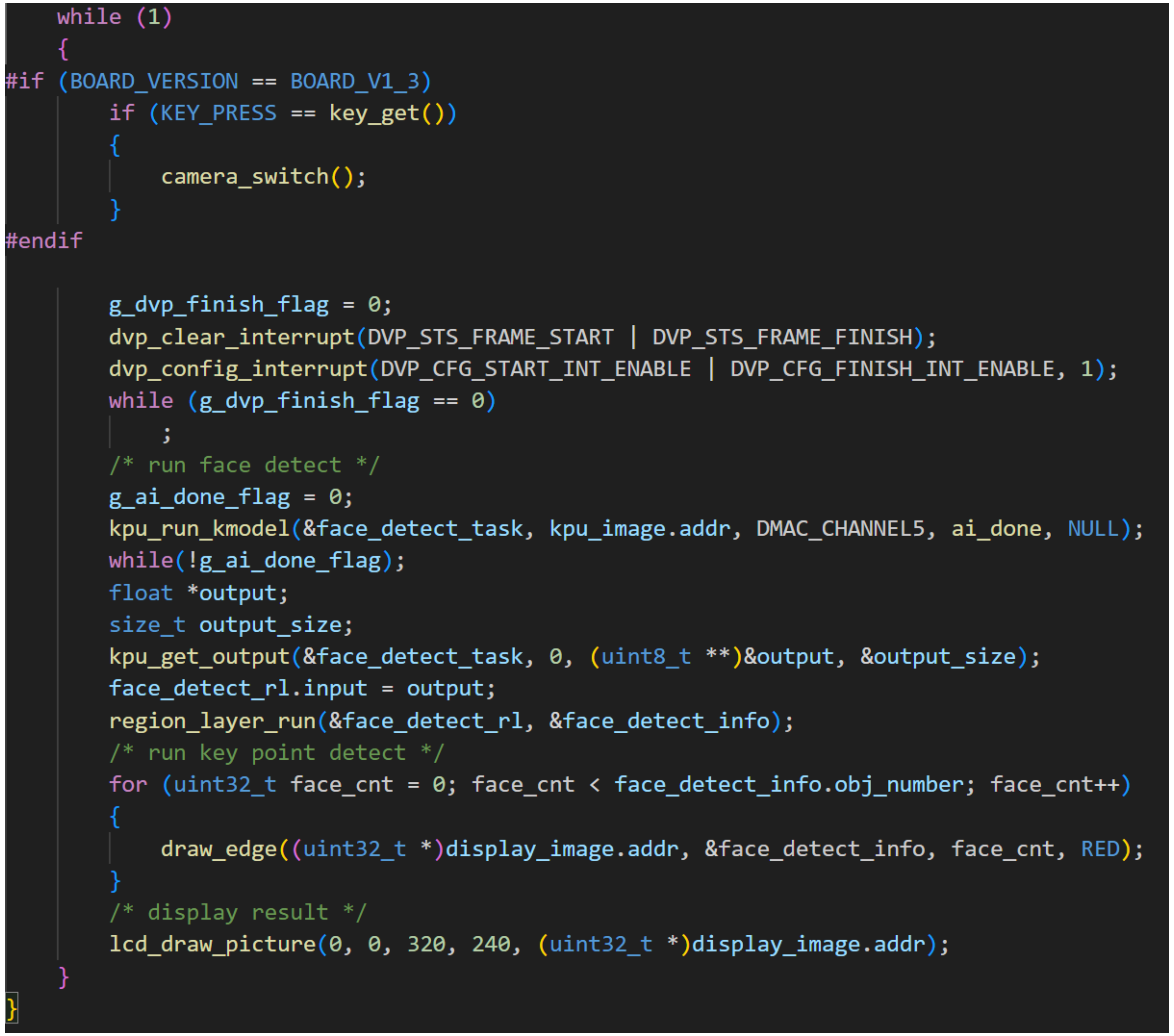
region_layer_t *rl：指向 region_layer_t 结构体的指针，表示一个区域层对象
int width：区域层的宽度
int height：区域层的高度
int channels：通道数，通常用于表示检测模型的输出通道数
int origin_width：原始图像的宽度（例如，输入图像的宽度）
int origin_height：原始图像的高度（例如，输入图像的高度）



```
static int isr_tick_task(void *ctx)
{
    key_scan();
    return 0;
}

void tick_init(size_t nanoseconds)
{
    timer_init(0);
    timer_irq_register(0, 0, 0, 1, isr_tick_task, NULL);
    timer_set_interval(0, 0, nanoseconds);
    timer_set_enable(0, 0, 1);
}
```

FACE DETECT实验例程介绍

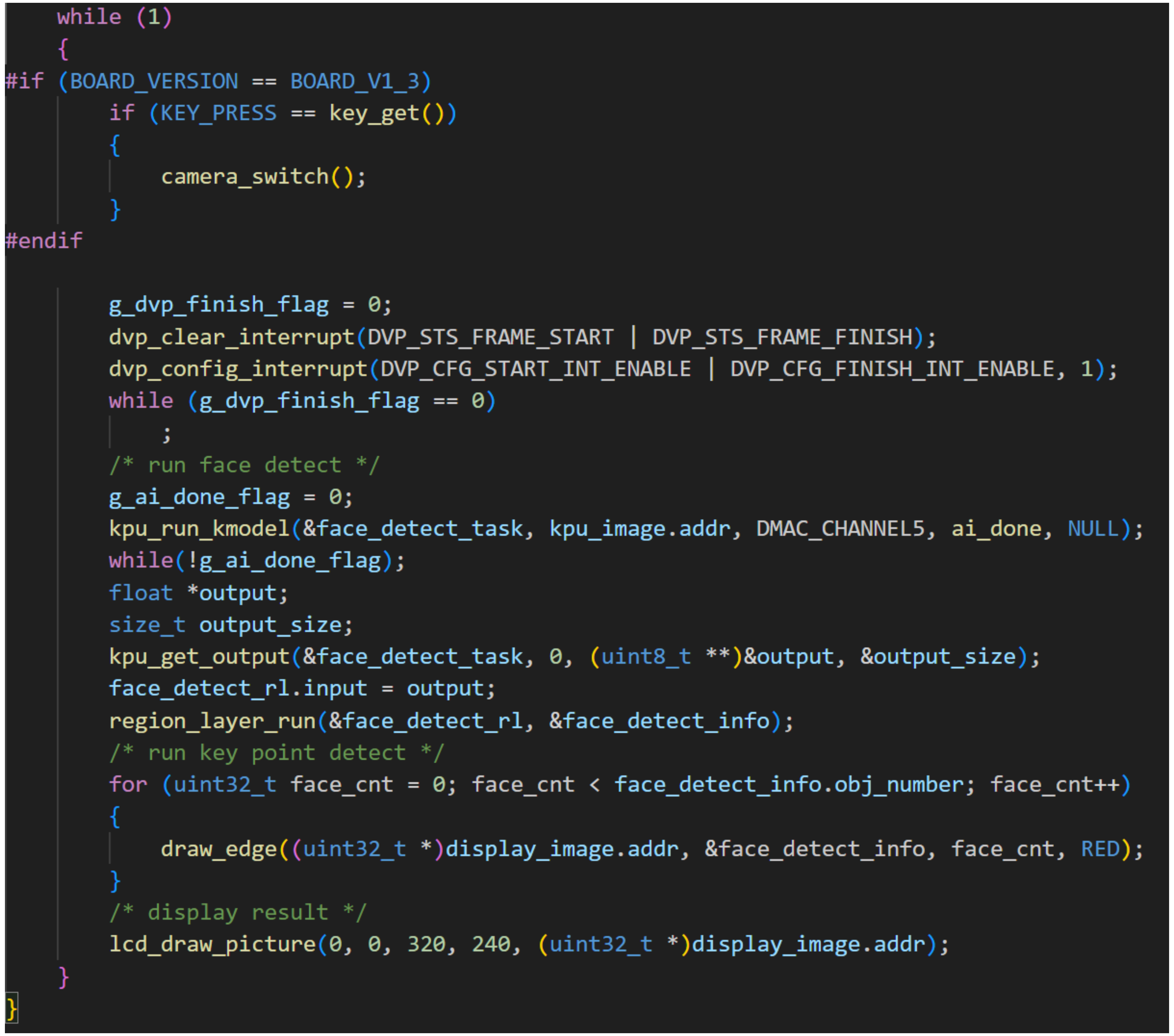


```
    while (1)
    {
#if (BOARD_VERSION == BOARD_V1_3)
        if (KEY_PRESS == key_get())
        {
            camera_switch();
        }
#endif

        g_dvp_finish_flag = 0;
        dvp_clear_interrupt(DVP_STS_FRAME_START | DVP_STS_FRAME_FINISH);
        dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 1);
        while (g_dvp_finish_flag == 0)
            ;
        /* run face detect */
        g_ai_done_flag = 0;
        kpu_run_kmodel(&face_detect_task, kpu_image.addr, DMAC_CHANNEL5, ai_done, NULL);
        while(!g_ai_done_flag);
        float *output;
        size_t output_size;
        kpu_get_output(&face_detect_task, 0, (uint8_t **)&output, &output_size);
        face_detect_rl.input = output;
        region_layer_run(&face_detect_rl, &face_detect_info);
        /* run key point detect */
        for (uint32_t face_cnt = 0; face_cnt < face_detect_info.obj_number; face_cnt++)
        {
            draw_edge((uint32_t *)display_image.addr, &face_detect_info, face_cnt, RED);
        }
        /* display result */
        lcd_draw_picture(0, 0, 320, 240, (uint32_t *)display_image.addr);
    }
}
```

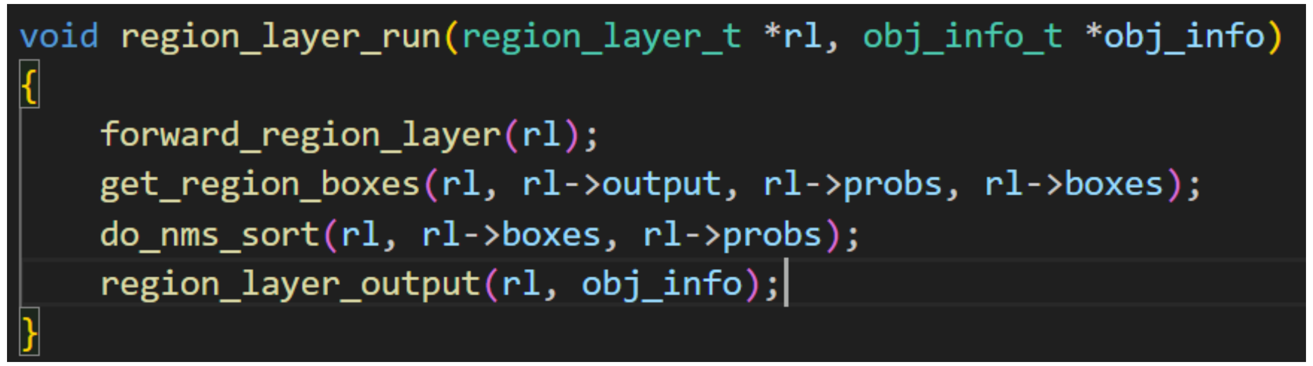
```
typedef struct
{
    int is_nncase;
    union
    {
        struct
        {
            const uint8_t *model_buffer;
            uint8_t *main_buffer;
            uint32_t output_count;
            const kpu_model_output_t *outputs;
            const kpu_model_layer_header_t *layer_headers;
            const uint8_t *body_start;
            uint32_t layers_length;
            volatile uint32_t current_layer;
            const uint8_t *volatile current_body;
            dmac_channel_number_t dma_ch;
            kpu_done_callback_t done_callback;
            void *userdata;
        };
        struct
        {
            void* nncase_ctx;
            uint32_t nncase_version;
        };
    };
} kpu_model_context_t;
```

FACE DETECT实验例程介绍



```
    while (1)
    {
#if (BOARD_VERSION == BOARD_V1_3)
        if (KEY_PRESS == key_get())
        {
            camera_switch();
        }
#endif

        g_dvp_finish_flag = 0;
        dvp_clear_interrupt(DVP_STS_FRAME_START | DVP_STS_FRAME_FINISH);
        dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 1);
        while (g_dvp_finish_flag == 0)
            ;
        /* run face detect */
        g_ai_done_flag = 0;
        kpu_run_kmodel(&face_detect_task, kpu_image.addr, DMAC_CHANNEL5, ai_done, NULL);
        while(!g_ai_done_flag);
        float *output;
        size_t output_size;
        kpu_get_output(&face_detect_task, 0, (uint8_t **)&output, &output_size);
        face_detect_rl.input = output;
        region_layer_run(&face_detect_rl, &face_detect_info);
        /* run key point detect */
        for (uint32_t face_cnt = 0; face_cnt < face_detect_info.obj_number; face_cnt++)
        {
            draw_edge((uint32_t *)display_image.addr, &face_detect_info, face_cnt, RED);
        }
        /* display result */
        lcd_draw_picture(0, 0, 320, 240, (uint32_t *)display_image.addr);
    }
}
```

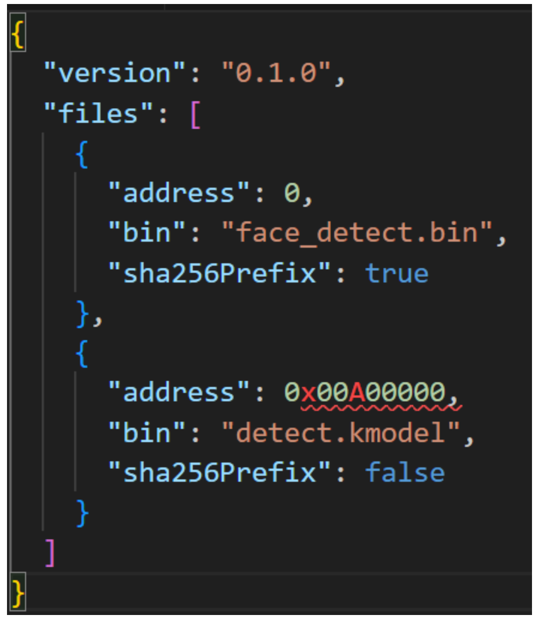


```
void region_layer_run(region_layer_t *rl, obj_info_t *obj_info)
{
    forward_region_layer(rl);
    get_region_boxes(rl, rl->output, rl->probs, rl->boxes);
    do_nms_sort(rl, rl->boxes, rl->probs);
    region_layer_output(rl, obj_info);
}
```

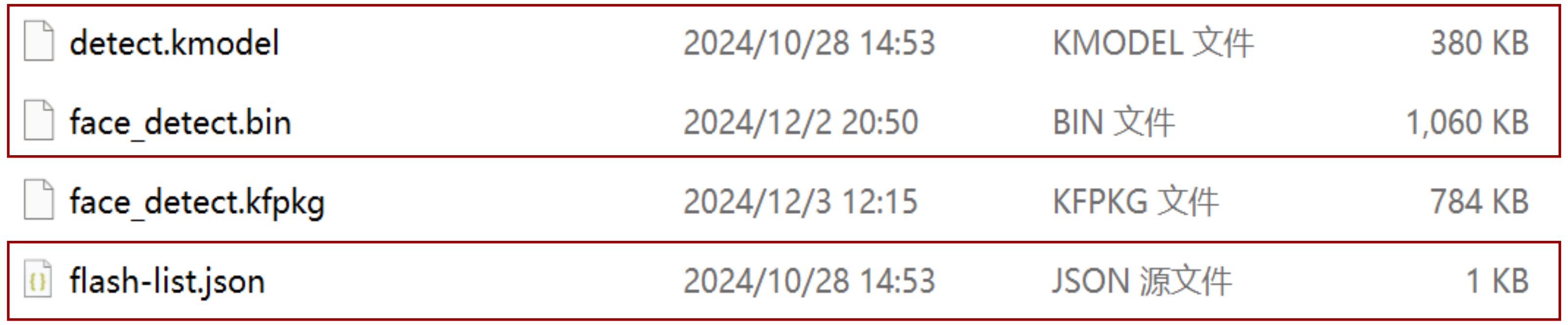
- 前向计算：通过 forward_region_layer，将输入数据通过激活函数（如 sigmoid）处理，计算边界框的坐标和置信度。
- 提取边界框：get_region_boxes 从模型输出中提取每个位置的边界框，包括每个框的坐标、类别概率等。
- 非极大值抑制（NMS）：do_nms_sort 对所有边界框进行排序，并根据一定的重叠度阈值（IoU），删除那些冗余的、重叠过大的框。
- 输出目标信息：在 region_layer_output 中，将剩下的边界框（经过 NMS 后）转换为最终的目标检测信息（如边界框坐标、类别、置信度等）。

代码烧录和实验结果

➤ 生成kfpkg文件



```
{
  "version": "0.1.0",
  "files": [
    {
      "address": 0,
      "bin": "face_detect.bin",
      "sha256Prefix": true
    },
    {
      "address": 0x00A00000,
      "bin": "detect.kmodel",
      "sha256Prefix": false
    }
  ]
}
```

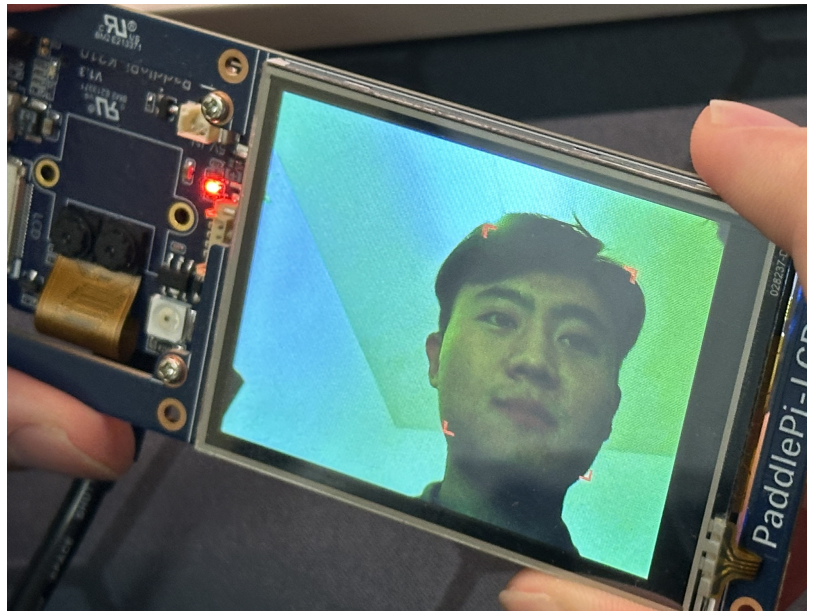


detect.kmodel	2024/10/28 14:53	KMODEL 文件	380 KB
face_detect.bin	2024/12/2 20:50	BIN 文件	1,060 KB
face_detect.kfpkg	2024/12/3 12:15	KFPKG 文件	784 KB
flash-list.json	2024/10/28 14:53	JSON 源文件	1 KB

Zip

face_detect.kfpkg

➤ 实验结果



汇报结束

感谢聆听