4.4 Neural Network Accelerator

1. Experiment purpose

In this lesson, we mainly learn function of neural network accelerator of K210.

2.Experiment preparation

2.1 components

Neural network accelerator in K210 chip

2.2 Component characteristics

KPU is a general-purpose neural network processor with built-in convolution, batch normalization, activation, and pooling operation units, which can detect faces or objects in real time.

2.3 Corresponding API function in SDK

The corresponding header file kpu.h

We will provide users with the following interfaces:

• kpu_task_init (No longer supported after 0.6.0, please use kpu_single_task_init): Initialize the kpu task handle. This function is implemented in gencode_output.c generated by the model compiler.

• kpu_run (No longer supported after 0.6.0, please use kpu_start): Start KPU to perform AI calculations.

• kpu_get_output_buf (No longer supported after 0.6.0): Get the cache of KPU output results.

• kpu_release_output_buf (No longer supported after 0.6.0): Release the KPU output result buffer.

• kpu_start: Start KPU and perform AI calculations.

• kpu_single_task_init: Initialize the kpu task handle.

• kpu_single_task_deinit: Log off the kpu task.

• kpu_model_load_from_buffer: Parsing kmodel and initialize kpu handle.

• kpu_load_kmodel: Loading kmodel, which needs to be used in conjunction with nncase.

• kpu_model_free: Release kpu resources.

• kpu_get_output: Get the final processing result of KPU.

• kpu_run_kmodel: Run kmodel.

3. Experiment principle

KPU can receive image data, and then through neural network convolution calculation, return the calculated data, LCD display output.

4. Experiment procedure

4.1 Define the mapping relationship between the hardware pins of LCD, camera, and buttons and software GPIO.

```
/*****************************HARDWARE-PIN*********************************/
// Hardware IO port, corresponding Schematic
#define PIN_LCD_CS              (36)
#define PIN_LCD_RST             (37)
#define PIN_LCD_RS              (38)
#define PIN_LCD_WR              (39)

// camera
#define PIN_DVP_PCLK            (47)
#define PIN_DVP_XCLK            (46)
#define PIN_DVP_HSYNC           (45)
#define PIN_DVP_PWDN            (44)
#define PIN_DVP_VSYNC           (43)
#define PIN_DVP_RST             (42)
#define PIN_DVP_SCL             (41)
#define PIN_DVP_SDA             (40)

#define PIN_KEYPAD_MIDDLE       (2)

/*****************************SOFTWARE-GPIO********************************/
//Software GPIO port, corresponding program
#define LCD_RST_GPIONUM         (0)
#define LCD_RS_GPIONUM          (1)

#define KEYPAD_MIDDLE_GPIONUM   (2)

/*****************************FUNC-GPIO************************************/
//Function of GPIO port, bound to hardware IO port
#define FUNC_LCD_CS             (FUNC_SPI0_SS3)
#define FUNC_LCD_RST            (FUNC_GPIOHS0 + LCD_RST_GPIONUM)
#define FUNC_LCD_RS             (FUNC_GPIOHS0 + LCD_RS_GPIONUM)
#define FUNC_LCD_WR             (FUNC_SPI0_SCLK)

#define FUNC_KEYPAD_MIDDLE      (FUNC_GPIOHS0 + KEYPAD_MIDDLE_GPIONUM)
```

```
void hardware_init(void)
{
    /* button */
    fpioa_set_function(PIN_KEYPAD_MIDDLE, FUNC_KEYPAD_MIDDLE);

    /* LCD */
    fpioa_set_function(PIN_LCD_CS,  FUNC_LCD_CS);
    fpioa_set_function(PIN_LCD_RST, FUNC_LCD_RST);
    fpioa_set_function(PIN_LCD_RS,  FUNC_LCD_RS);
    fpioa_set_function(PIN_LCD_WR,  FUNC_LCD_WR);

    // DVP camera
    fpioa_set_function(PIN_DVP_RST,   FUNC_CMOS_RST);
    fpioa_set_function(PIN_DVP_PWDN,  FUNC_CMOS_PWDN);
    fpioa_set_function(PIN_DVP_XCLK,  FUNC_CMOS_XCLK);
    fpioa_set_function(PIN_DVP_VSYNC, FUNC_CMOS_VSYNC);
    fpioa_set_function(PIN_DVP_HSYNC, FUNC_CMOS_HREF);
    fpioa_set_function(PIN_DVP_PCLK,  FUNC_CMOS_PCLK);
    fpioa_set_function(PIN_DVP_SCL,   FUNC_SCCB_SCLK);
    fpioa_set_function(PIN_DVP_SDA,   FUNC_SCCB_SDA);

    // enable SPI0 and DVP
    sysctl_set_spi0_dvp_data(1);
}
```

4.2 Set the level voltage required for the camera and display interface to 1.8V.

```
static void io_set_power(void)
{
    /* Set dvp and spi pin to 1.8V */
    sysctl_set_power_mode(SYSCTL_POWER_BANK6, SYSCTL_POWER_V18);
    sysctl_set_power_mode(SYSCTL_POWER_BANK7, SYSCTL_POWER_V18);
}
```

4.3 Sets the system clock, initializes system interrupts, and enables global system interrupts.

```
/* Set the system clock and the DVP clock*/
sysctl_pll_set_freq(SYSCTL_PLL0, 800000000UL);
sysctl_pll_set_freq(SYSCTL_PLL1, 300000000UL);
sysctl_pll_set_freq(SYSCTL_PLL2, 45158400UL);
uarths_init();

/* System interrupt initialization*/
plic_init();
/* Enable system global interrupt */
sysctl_enable_irq();
```

4.4 Initialize the display screen and display the picture for one second.

```
/* Initialize the display screen and display a one-second image */
printf("LCD init\r\n");
lcd_init();
lcd_draw_picture_half(0, 0, 320, 240, gImage_logo);
sleep(1);
```

4.5 Initialize ov2640 camera

```
/* Initialize ov2640 camera */
printf("ov2640 init\r\n");
ov2640_init();
```

4.6 Initializes the key and sets the key interrupt callback.

```
/* Key interrupt callback*/
int key_irq_cb(void *ctx)
{
    key_flag = 1;
    key_state = gpiohs_get_pin(KEYPAD_MIDDLE_GPIONUM);
    return 0;
}

/* Initializing key */
void init_key(void)
{
    // Set the GPIO mode for the keys to be pull-up input
    gpiohs_set_drive_mode(KEYPAD_MIDDLE_GPIONUM, GPIO_DM_INPUT_PULL_UP);
    // Set the GPIO level trigger mode of the key to be rise edge and fall edge
    gpiohs_set_pin_edge(KEYPAD_MIDDLE_GPIONUM, GPIO_PE_BOTH);
    // Set the interrupt callback for the key GPIO port
    gpiohs_irq_register(KEYPAD_MIDDLE_GPIONUM, 1, key_irq_cb, NULL);
}
```

4.7 KPU initializes and prints the remaining memory.

```
/* Initialize kpu */
kpu_task_t task;
conv_init(&task, CONV_3_3, conv_data);

printf("KPU TASK INIT, FREE MEM: %ld\r\n", get_free_heap_size());
printf("Please press the keypad to switch mode\r\n");
```

4.8 Waiting for the camera to complete the data transmission, kpu starts to calculate the data collected by the camera and outputs it to g_ai_buf_out.

```
while (g_dvp_finish_flag == 0)
    ;
/* Starting */
conv_run(&task, g_ai_buf_in, g_ai_buf_out, kpu_done);
```

```
/* KPU done*/
static int kpu_done(void *ctx)
{
    g_ai_done_flag = 1;
    return 0;
}
```

4.9 Wait for the KPU calculation to complete, and convert the data to RGB565 format, because the display does not support RGB888 format.

```
while (!g_ai_done_flag)
    ;
g_ai_done_flag = 0;
g_dvp_finish_flag = 0;
/* Convert to RGB565 format supported by LCD */
rgb888_to_565(g_ai_buf_out, g_ai_buf_out + 320 * 240, g_ai_buf_out + 320 * 240 * 2,
    (uint16_t *)g_display_buf, 320 * 240);
```

```
/* Convert image data format, because the camera output to AI is in RGB888 format, while the dis
void rgb888_to_565(uint8_t *src_r, uint8_t *src_g, uint8_t *src_b, uint16_t *dst, uint32_t len)
{
    uint32_t i;
    for (i = 0; i < len; i += 2)
    {
        dst[i] = (((uint16_t)(src_r[i + 1] >> 3)) << 11) +
            (((uint16_t)src_g[i + 1] >> 2) << 5) +
            (((uint16_t)src_b[i + 1]) >> 3);
        dst[i + 1] = (((uint16_t)(src_r[i] >> 3)) << 11) +
            (((uint16_t)src_g[i] >> 2) << 5) +
            (((uint16_t)src_b[i]) >> 3);
    }
}
```

4.10 At this point, we need to write each corresponding mode in the upper left corner. The default is the original origin.

```
/* Upper left display mode */
void draw_text(void)
{
    char string_buf[8 * 16 * 2 * 16]; //16 characters
    char title[20];

    switch (demo_index)
    {
    case 0:
        sprintf(title, " origin ");
        lcd_ram_draw_string(title, (uint32_t *)string_buf, BLUE, BLACK);
        lcd_ram_cpyimg((char *)g_display_buf, 320, string_buf, strlen(title) * 8, 16, 0, 0);
        break;
    case 1:
        sprintf(title, "  edge  ");
        lcd_ram_draw_string(title, (uint32_t *)string_buf, BLUE, BLACK);
        lcd_ram_cpyimg((char *)g_display_buf, 320, string_buf, strlen(title) * 8, 16, 0, 0);
        break;
    case 2:
        sprintf(title, " sharp  ");
        lcd_ram_draw_string(title, (uint32_t *)string_buf, BLUE, BLACK);
        lcd_ram_cpyimg((char *)g_display_buf, 320, string_buf, strlen(title) * 8, 16, 0, 0);
        break;
    case 3:
        sprintf(title, "relievos");
        lcd_ram_draw_string(title, (uint32_t *)string_buf, BLUE, BLACK);
        lcd_ram_cpyimg((char *)g_display_buf, 320, string_buf, strlen(title) * 8, 16, 0, 0);
        break;

    default:
        break;
    }
```

4.11 Finally, the data is displayed on the LCD.

```
/* The upper left corner indicates which mode to write letters in */
draw_text();
/* Display image */
lcd_draw_picture(0, 0, 320, 240, g_display_buf);
```

4.12 Change the state of the button by interrupting the button to switch the mode.

```
if (key_flag)
{
    if (key_state == 0) //press
    {
        msleep(20);
        key_flag = 0;
        demo_index = (demo_index + 1) % 4;
        memcpy((void *)conv_data, (void *)(conv_data_demo[demo_index]),
            3 * 3 * 3 * 3 * sizeof(float));
        conv_init(&task, CONV_3_3, conv_data);
    }
    else
    {
        msleep(20);
        key_flag = 0;
    }
}
```

13. Compile and debug, burn and run

Copy the kpu to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

cmake .. -DPROJ=kpu -G "MinGW Makefiles"

make

```
[ 89%] Linking C executable kpu
Generating .bin file ...
[100%] Built target kpu
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **kpu.bin** file will be generated in the build folder.

We need to use the type-C data cable to connect the computer and the K210 development board.

Open kflash, select the corresponding device, and then burn the **kpu.bin** file to the K210 development board.

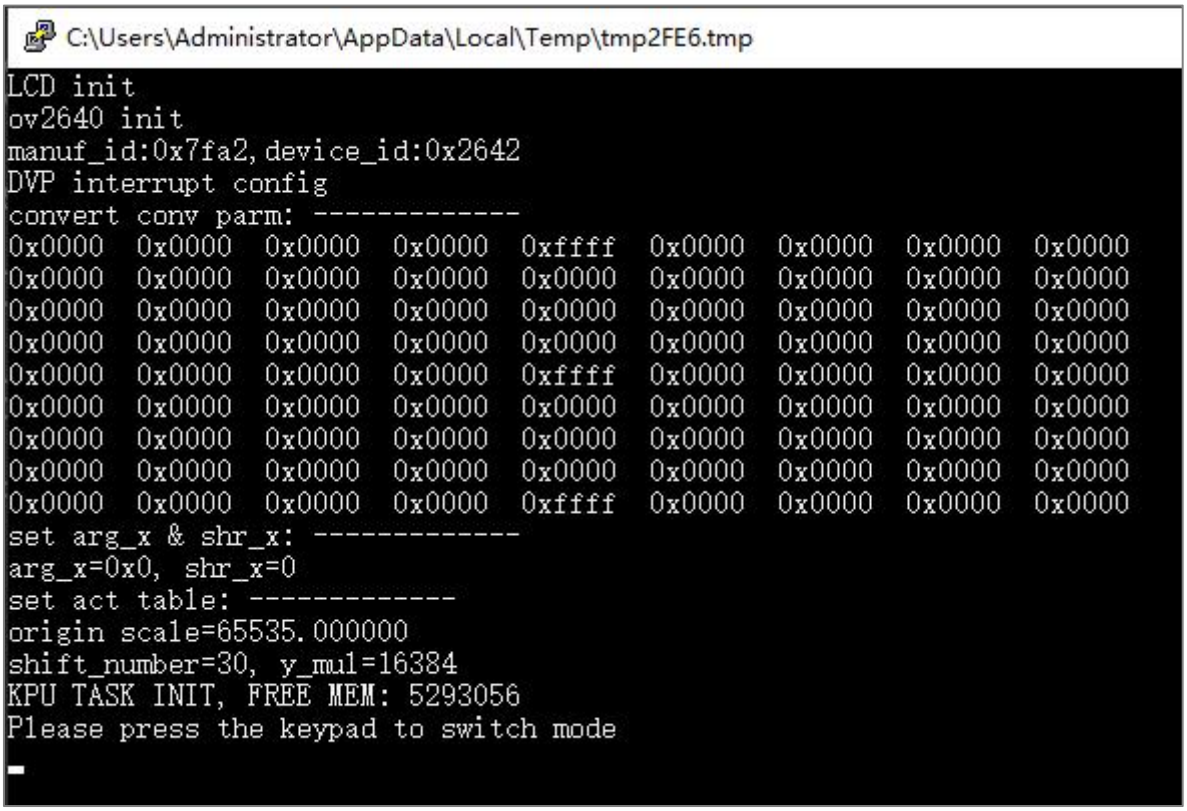
5. Experimental phenomenon

After the firmware is write, a terminal interface will pop up. It will print some initialization information. As shown below.

And print some initialization information. At this time, we see that the monitor has displayed the current captured image of the camera, and there is a word "origin" in the upper left corner.

When we press the middle of the keypad, the mode will be switched. The LCD displays pictures will be changed.

In addition to the original screen, there are three other modes that can be displayed. When you press the keypad every time, you can switch the mode.



6. Experiment summary

6.1 KPU can process images and produce different visual effects.

6.2 The data format of KPU and LCD is different, and it can be displayed normally after conversion.

Appendix -- API

Header file is kpu.h

kpu_task_init

Description: Initializes the KPU task handle.

Function prototype:

kpu_task_t* kpu_task_init(kpu_task_t* task)

Parameter:

Parameter name	Description	Input/Output
task	KPU task handle	Input

Return value: KPU task handle.

kpu_run

Description: Start KPU and perform AI calculation.

Function prototype:

int kpu_run(kpu_task_t* v_task, dmac_channel_number_t dma_ch, const void *src, void* dest, plic_irq_callback_t callback)

Parameter:

Parameter name	Description	Input/Output
task	KPU task handle	Input
dma_ch	DMA channel	Input
src	Input image data	Input
dest	Output operation result	Output
callback	Operation completes the callback function	Input

Return value:

Return value	Description
0	succeed
!0	KPU busy, fail

kpu_get_output_buf

Description: Gets the cache of the KPU output.

Function prototype:

uint8_t *kpu_get_output_buf(kpu_task_t* task)

Parameter:

Parameter name	Description	Input/Output
task	KPU task handle	Input

Return value:

A pointer to a cache of KPU output results.

kpu_release_output_buf

Description: Release the cache of the KPU output.

Function prototype:

void kpu_release_output_buf(uint8_t *output_buf)

Parameter:

Parameter name	Description	Input/Output
output_buf	cache of the KPU output	Input

Return value: no

kpu_start

Description: Start KPU and perform AI calculation.

Function prototype:

int kpu_start(kpu_task_t *task)

Parameter:

Parameter name	Description	Input/Output
task	KPU task handle	Input

Return value:

Return value	Description
0	succeed
!0	KPU busy, fail

kpu_single_task_init

Description: Initializes the KPU task handle.

Function prototype:

int kpu_single_task_init(kpu_task_t *task)

Parameter:

Parameter name	Description	Input/Output
task	KPU task handle	Input

Return value:

Return value	Description
0	succeed
!0	fail

kpu_single_task_deinit

Description: Log off KPU task

Function prototype:

int kpu_single_task_deinit(kpu_task_t *task)

Parameter:

Parameter name	Description	Input/Output
task	KPU task handle	Input

Return value:

Return value	Description
0	succeed
!0	fail

kpu_model_load_from_buffer

Description: Parsing the kmodel and initialize the KPU handle.

Function prototype:

int kpu_model_load_from_buffer(kpu_task_t *task, uint8_t *buffer, kpu_model_layer_metadata_t **meta);

Parameter:

Parameter name	Description	Input/Output
task	KPU task handle	Input
buffer	Kmodel data	Input
meta	Internal test data, user set to NULL	Output

Return value:

Return value	Description
0	succeed
!0	fail

kpu_load_kmodel

Description: Kmodel needs to be used in conjunction with NNCASE to load.

Function prototype:

int kpu_load_kmodel(kpu_model_context_t *ctx, const uint8_t *buffer)

Parameter:

Parameter name	Description	Input/Output
ctx	KPU task handle	Input
buffer	Kmodel data	Input

Return value:

Return value:	Description
0	succeed
!0	fail

kpu_model_free

Description: Release KPU resources.

Function prototype:

void kpu_model_free(kpu_model_context_t *ctx)

Parameter:

Parameter name	Description	Input/Output
ctx	KPU task handle	Input

Return value: no

kpu_get_output

Description: Get the results of the final KPU processing.

Function prototype:

int kpu_get_output(kpu_model_context_t *ctx, uint32_t index, uint8_t **data, size_t *size)

Parameter:

Parameter name	Description	Input/Output
ctx	KPU task handle	Input
index	The index value of the result, such as kmodel related	Input
data	result	Input
size	Size(byte)	Input

Return value:

Return value	Description
0	succeed
!0	fail

kpu_run_kmodel

Description: Running kmodel

Function prototype:

int kpu_run_kmodel(kpu_model_context_t *ctx, const uint8_t *src, dmac_channel_number_t dma_ch, kpu_done_callback_t done_callback, void *userdata)

Parameter:

Parameter name	Description	Input/Output
ctx	KPU task handle	Input
src	source data	Input
dma_ch	DMA channel	Input
done_callback	After it is completes, it will call callback function.	Input
userdata	The parameters of the callback	Input

Return value:

Return value	Description
0	succeed
!0	fail

Eg:

```
/* Generate kpu_task_gencode_output_init through MC, set the source data to g_ai_buf, use DMA5, and call the ai_done function after kpu is completed */
kpu_task_t task;
volatile uint8_t g_ai_done_flag;
static int ai_done(void *ctx)
{
    g_ai_done_flag = 1;
    return 0;
}

/* Initialize the kpu */
kpu_task_gencode_output_init(&task);
task.src = g_ai_buf;
task.dma_ch = 5;
task.callback = ai_done;
kpu_single_task_init(&task);

/*Start up kpu */
kpu_start(&task);
```

Data type

The related data types and data structure are defined as follows:

kpu_task_t：kpu task structure

kpu_task_t

Description: Kpu task structure.

Define

```
typedef struct
{
    kpu_layer_argument_t *layers;
    kpu_layer_argument_t *remain_layers;
    plic_irq_callback_t callback;
    void *ctx;
    uint64_t *src;
    uint64_t *dst;
    uint32_t src_length;
    uint32_t dst_length;
    uint32_t layers_length;
    uint32_t remain_layers_length;
    dmac_channel_number_t dma_ch;
    uint32_t eight_bit_mode;
    float output_scale;
    float output_bias;
    float input_scale;
    float input_bias;
} kpu_task_t;
```

Member

Member name	Description
layers	KPU parameter pointer (MC initialization, users do not need to care)
remain_layers	KPU parameter pointer (used during calculation, users do not need to care)
callback	Operation completion callback function (user setting required)
ctx	Parameters of the callback function (non-empty needs to be set by the user)
src	Calculation source data (requires user settings)
dst	Operation result output pointer (KPU initialization assignment, users do not need to care)
src_length	Source data length (MC initialization, users do not need to care)
dst_length	Operation result length (MC initialization, users don't need to care)
layers_length	Number of layers (MC initialization, users do not need to care)
remain_layers_length	Number of remaining layers (used during calculation, users don't need to care)

Member name	Description
dma_ch	DMA channel number used (user setting required)
eight_bit_mode	Whether it is 8-bit mode (MC initialization, users do not need to care)
output_scale	Output scale value (MC initialization, users do not need to care)
output_bias	Output bias value (MC initialization, users do not need to care)
input_scale	Input scal value (MC initialization, users do not need to care)
input_bias	Input bias value (MC initialization, users do not need to care)