Kneron Inc

Document Name: **KL520 model on flash memory**

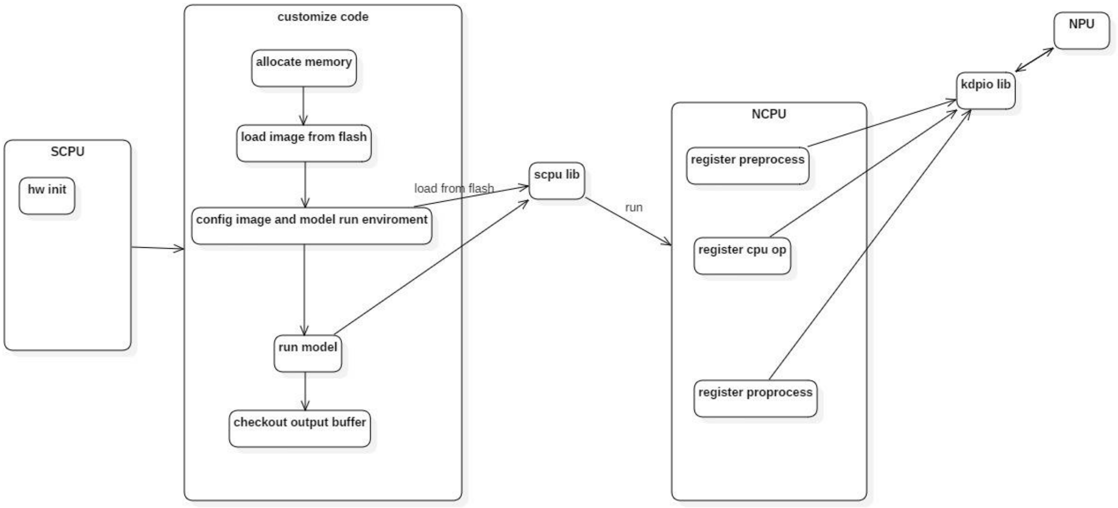
KL520 model on flash memory tutorial

Kneron Inc

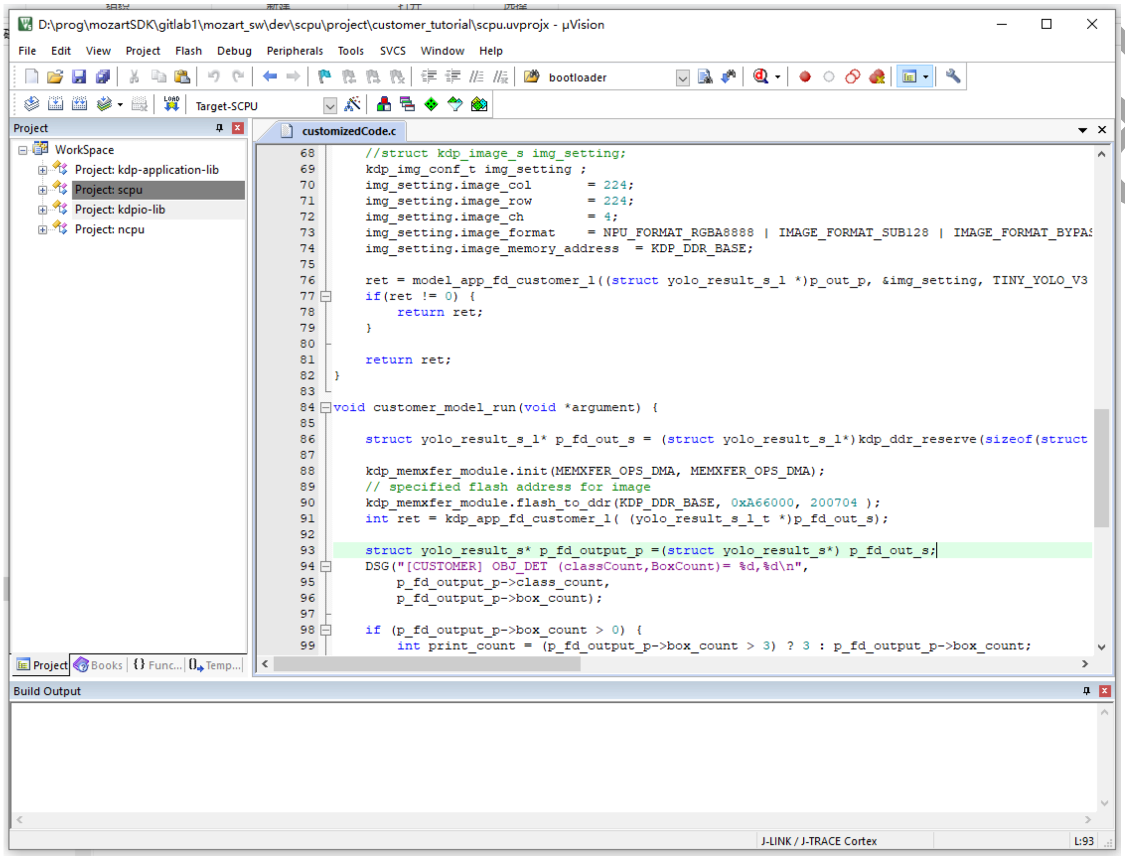
Table of Contents

1. Introduction

This tutorial describes the overview flow of run customer model on chip, and the key points which need to be pay attention to during implementation. The example in this chapter is based on Tiny-YOLO-V3 model, and the dimension of Input Node is 3x224x224 (Channel*Height*Width) in ONNX format.



Start keil uVision tutorial project on example_projects\customer_tutorial\:


D:\prog\mozartSDK\gitlab1\mozart_sw\dev\scpu\project\customer_tutorial\scpu.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
bootloader
Target-SCPU
Project
WorkSpace
Project: kdp-application-lib
Project: scpu
Project: kdpio-lib
Project: ncpu
Project Books {} Func... Temp...
customizedCode.c
Build Output
J-LINK / J-TRACE Cortex
L:93

```c
    //struct kdp_image_s img_setting;
    kdp_img_conf_t img_setting ;
    img_setting.image_col        = 224;
    img_setting.image_row        = 224;
    img_setting.image_ch         = 4;
    img_setting.image_format     = NPU_FORMAT_RGBA8888 | IMAGE_FORMAT_SUB128 | IMAGE_FORMAT_BYPAS
    img_setting.image_memory_address  = KDP_DDR_BASE;

    ret = model_app_fd_customer_l((struct yolo_result_s_l *)p_out_p, &img_setting, TINY_YOLO_V3
    if(ret != 0) {
        return ret;
    }

    return ret;
}

void customer_model_run(void *argument) {

    struct yolo_result_s_l* p_fd_out_s = (struct yolo_result_s_l*)kdp_ddr_reserve(sizeof(struct

    kdp_memxfer_module.init(MEMXFER_OPS_DMA, MEMXFER_OPS_DMA);
    // specified flash address for image
    kdp_memxfer_module.flash_to_ddr(KDP_DDR_BASE, 0xA66000, 200704 );
    int ret = kdp_app_fd_customer_l( (yolo_result_s_l_t *)p_fd_out_s);

    struct yolo_result_s* p_fd_output_p =(struct yolo_result_s*) p_fd_out_s;
    DSG("[CUSTOMER] OBJ_DET (classCount,BoxCount)= %d,%d\n",
        p_fd_output_p->class_count,
        p_fd_output_p->box_count);

    if (p_fd_output_p->box_count > 0) {
        int print_count = (p_fd_output_p->box_count > 3) ? 3 : p_fd_output_p->box_count;
```

1.1 Working Flow

To run customized model and get output in KL520, you need follow these steps:

(1) Compile models: get all_models.bin, fw_info.bin
(2) Convert image into bin file (optional)
(3) Generate flash bin file
(4) Upload bin file into board flash memory
(5) Allocate memory address for model output and input, load image from flash
(6) Config model environment and model run
(7) Checkout network output

2. Compile models: get all_models.bin fw_info.bin

2.1 Prerequisite

For Windows 10 Home system, please install Docker Toolbox
https://download.docker.com/win/stable/DockerToolbox.exe

For Windows 10 Professional system, please install

https://docs.docker.com/docker-for-windows/install/

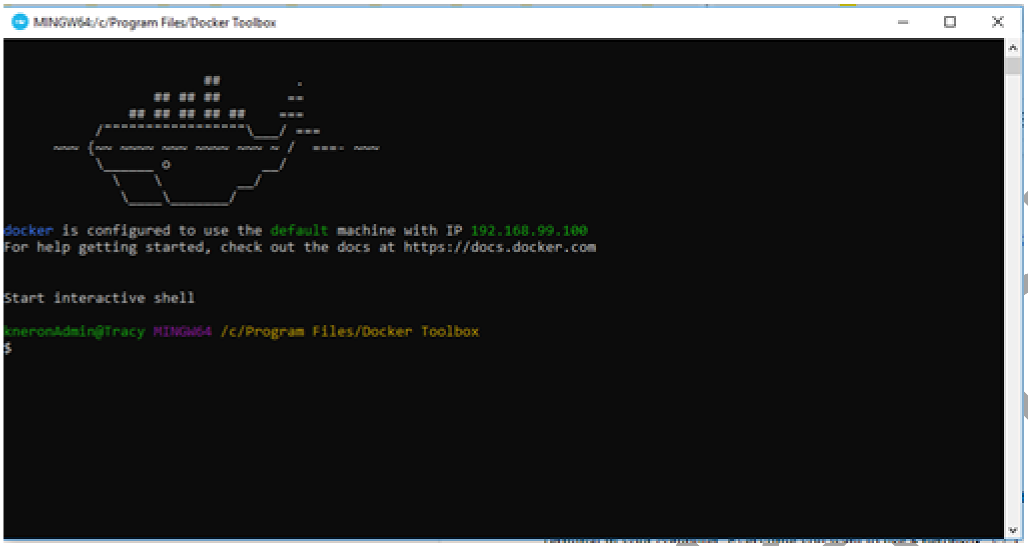
For Ubuntu 16.0.4, please follow these instructions

https://docs.docker.com/install/linux/docker-ce/ubuntu/ to install.

2.2 Docker check

Before using this toolchain, you need to start the docker properly.

For windows 10 home users, if you successfully install Docker Toolbox, which is introduced in the Part 2.1, you can search and find Docker Quickstart Terminal in your computer. Everytime you want to use this toolchain, please remember to start Docker Quickstart Terminal. Figure 1 shows the appearance of Docker Quickstart Terminal which is started properly. If the terminal does not show this appearance, it means there is some problem for the installation of Docker.



MINGW64:/c/Program Files/Docker Toolbox

```
docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com


Start interactive shell

kneronAdmin@Tracy MINGW64 /c/Program Files/Docker Toolbox
$
```

Figure 1 Appearance of Docker Quickstart Terminal

For ubuntu 16.0.4 users, please refer this link

[https://www.digitalocean.com/community/questions/how-to-check-for-docker-installation to check whether docker is installed successfully.](https://www.digitalocean.com/community/questions/how-to-check-for-docker-installation)

2.3 Tutorial Example

During this tutorial, it will use Keras Onet model as an example. For the window users, you need to type the commands by Docker Quickstart terminal or other related Docker tools, and for the ubuntu 16.0.4 users, you need to type the commands by ubuntu terminal.

2.3.1 Pull the docker image

Each time when there is a new version of linux command toolchain releasing, you need to pull the latest docker image.

The command to pull the latest image is:

```
docker pull kneron/toolchain:linux_command_toolchain
```

After finishing this command, the docker image is saved in your local side, and only when you want to update the image to the latest one, you need to connect to the Internet to repeat this command again. Otherwise, you can finish the following parts without the connection to the Internet.

To check whether the docker image is pulled successfully, type in this command:

```
docker image ls
```

If the image is pulled successfully, you will see the image, mrwhoami/kmc_docker:linux_command_toolchain in the docker image list.

2.3.2 Start the docker image

Then you can start the docker image you just pulled, and get a docker container to run the toolchain. When you start it, you need to configure a local folder as the one for communicating between your local environment and the container, let's call it as **Interactive Folder** Assume the absolute path of the folder you configure is absolue_path_of_your_folder.

And the start command is:

```
docker run -it --rm -v absolute_path_of_your_folder:/data1
kneron/toolchain:linux_command_toolchain
```

For example, if the absolute path of the path folder you configure is /home/kneron/Document/test_docker, and then the related command is

```
docker run -it --rm -v /home/kneron/Document/test_docker:/data1
kneron/toolchain:linux_command_toolchain
```

After running the start command, you'll enter into the docker container. Then, copy the example materials to the Interactive Folder by the following command:

```
cp -r /workspace/examples/* /data1/
```

2.3.3 Converter

2.3.3.1 Keras to ONNX

The command format for converting keras model to onnx model is:

```
python /workspace/onnx-keras/generate_onnx.py -o absolute_path_of_output_model_file absolute_path_of_input_model_file -O -C
```

For Onet example here, the detailed command is:

```
python /workspace/onnx-keras/generate_onnx.py -o /data1/onet-0.417197.onnx /data1/keras/onet-0.417197.hdf5 -O -C
```

For tiny_yolo_v3 example, the detailed command is:

```
python /workspace/onnx-keras/generate_onnx.py -o /data1/yolov3-tiny-224.h5.onnx /data1/keras/yolov3-tiny-224.h5 -O -C
```

There might be some warning log when running this problem, and you can check whether the convert works successfully by checking whether the onnx file is generated.

2.3.3.2 Tensorflow to ONNX

```
/workspace/scripts/tf2onnx.sh absolute_path_of_input_model_file absolute_path_of_output_onnx_model_file name_of_input_layer:0 name_of_output_layer:0
```

For the provided example model: mnist.pb

```
/workspace/scripts/tf2onnx.sh /data1/tensorflow/model/mnist.pb /data1/mnist.pb.onnx Placeholder:0 fc2/add:0
```

2.3.3.3 Pytorch to ONNX

```
python /workspace/scripts/pytorch2onnx.py absolute_path_of_input_model_file channel_number model_input_width model_input_height absolute_path_of_output_model_file
```

For the provided example model: resnet34.pth

```
python /workspace/scripts/pytorch2onnx.py /data1/pytorch/models/resnet34.pth 3 224 224 /data1/resnet34.onnx
```

2.3.3.4 Caffe to ONNX

```
python /workspace/onnx-caffe/generate_onnx.py -o absolute_path_of_output_onnx_model_file -w absolute_path_of_input_caffe_weight_file -n absolute_path_of_input_caffe_model_file
```

For the provided example model: mobilenetv2.caffemodel

```
python /workspace/onnx-caffe/generate_onnx.py -o /data1/mobilenetv2.onnx -w /data1/caffe/models/mobilenetv2.caffemodel -n /data1/caffe/models/mobilenetv2.prototxt
```

2.3.4 FpAnalyser, Simulator and Batch-Compile

This part is the instructions for batch-compile, which will generate the binary file requested by firmware.

2.3.4.1 Fill the input parameters

Fill the simulator and emulator input parameters in the /data1/batch_compile_input_params.json in Interactive Folder. Please refer on the FAQ question 4 to fill the related parameters.

And the follow will give two examples for how to configure the batch_compile_input_params.json.

(1) tiny_yolo_v3

```
{
   "input_image_folder": ["/data1/caffe/images"],
   "img_channel": ["RGB"],
   "model_input_width": [224],
   "model_input_height": [224],
   "img_preprocess_method": ["yolo"],
   "input_onnx_file": ["/data1/yolov3-tiny-224.h5.onnx"],
   "command_addr": "0x30000000",
   "weight_addr": "0x40000000",
   "sram_addr": "0x50000000",
   "dram_addr": "0x60000000",
   "whether_encryption": "No",
   "encryption_key": "0x12345678",
   "model_id_list": [19],
   "model_version_list": [1]
}
```

(2) tiny_yolo_v3 and Onet

```
{
    "input_image_folder": ["/data1/caffe/images", "/data1/keras/n000645"],
    "img_channel": ["RGB", "L"],
    "model_input_width": [224, 48],
    "model_input_height": [224, 48],
    "img_preprocess_method": ["yolo", "kneron"],
    "input_onnx_file": ["/data1/yolov3-tiny-224.h5.onnx", "/data1/onet-0.417197.onnx"],
    "command_addr": "0x30000000",
    "weight_addr": "0x40000000",
    "sram_addr": "0x50000000",
    "dram_addr": "0x60000000",
    "whether_encryption": "No",
    "encryption_key": "0x12345678",
    "model_id_list": [19, 20],
    "model_version_list": [1, 1]
}
```

2.3.4.2 Batch compile

For running the compiler and ip evaluator:

```
cd /workspace/scripts && ./fpAnalyserBatchCompile.sh.x
```

And a folder called batch_compile will be generated in Interactive Folder, which stores the result of the fpAnalyer and batch-compile.

2.3.4.3 Simulator and Emulator

Fill the simulator and emulator input parameters in the input_params.json in Interactive Folder.

Running the programs

For running the simulator:

```
cd /workspace/scripts && ./simulator.sh.x
```

And a folder called simulator will be generated in Interactive Folder, which stores the result of the simulator.

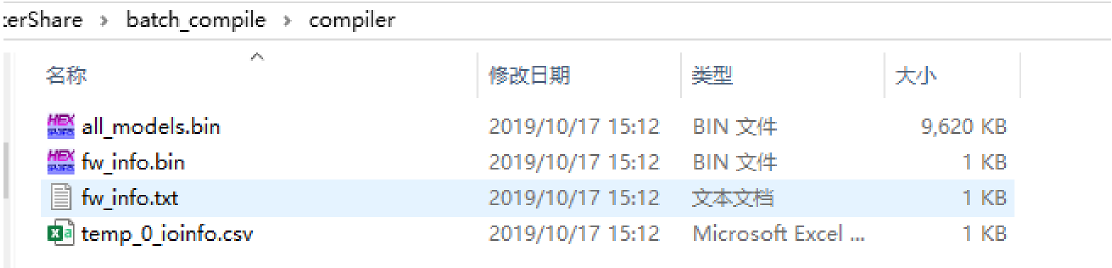
For running the emulator:

```
cd /workspace/scripts && ./emulator.sh.x
```

And a folder called emulator will be generated in Interactive Folder, which stores the result of the emulator.

2.3.4.4 Get the output file from compiler

In Interactive Folder, you'll find a folder called compiler, which contains the output files of the fpAnalyer and batch-compile.



:erShare › batch_compile › compiler

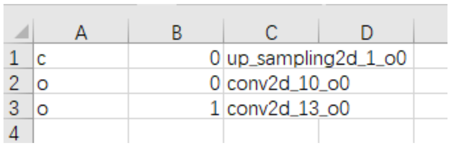
名称	修改日期	类型	大小
all_models.bin	2019/10/17 15:12	BIN 文件	9,620 KB
fw_info.bin	2019/10/17 15:12	BIN 文件	1 KB
fw_info.txt	2019/10/17 15:12	文本文档	1 KB
temp_0_ioinfo.csv	2019/10/17 15:12	Microsoft Excel ...	1 KB

fw_info.txt list readable info of model id, version etc.

all_model.bin and fw_info.bin is for firmware to use

temp_X_ioinfo.csv contains the information that cpu node and output node.

If you find the cpu node in temp_X_ioinfo.csv, whose format is "c,**,**", you need to implement and register this function in SDK.



	A	B	C	D
1	c	0	up_sampling2d_1_o0	
2	o	0	conv2d_10_o0	
3	o	1	conv2d_13_o0	
4				

3. HOST side

Host is responsible for communicating with SCPU, sending commands, firmware information (fw_info.bin), model data (all_models.bin), configuration, and input images to SCPU, and retrieving back the output results from SCPU through USB interface.

3.1 Convert image into bin file (optional)

e.g python source convert image to bin(rgb565) :

```
import numpy as np
from PIL import Image

def normalization(x, mode):
    x = x.astype(float)

    if mode == 'Nothing':
        return x

    if mode == 'yolo':
        x /= 255.
        return x

    if mode == 'kneron':
        x /= 256.
        x -= 0.5
        return x

    if mode == 'tf':
        x /= 127.5
        x -= 1.
        return x

    if mode == 'torch':
        x /= 255.
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]

    if mode == 'caffe':
        ### mean is for BGR format
        mean = [103.939, 116.779, 123.68]
        std = None

    # Zero-center by mean pixel
    x[..., 0] -= mean[0]
    x[..., 1] -= mean[1]
    x[..., 2] -= mean[2]
    if std is not None:
        x[..., 0] /= std[0]
        x[..., 1] /= std[1]
        x[..., 2] /= std[2]
    return x

def resize(x, size, keep_aspect_ratio=True):
    if not keep_aspect_ratio:
        return np.array(x.resize(size))
    else:
        x.thumbnail(size, Image.ANTIALIAS)
        x = x.crop((0, 0, size[0], size[1]))
        return np.array(x)

def convert_to_rgb565(clrspace,imagefile,size,keep_aspect_ratio,mode):
    image = Image.open(imagefile)
    if clrspace == "BGR":
        image = Image.fromarray(np.array(image)[...,::-1])
    else:
        image = image.convert(clrspace)

    img_data = resize(image, size, keep_aspect_ratio)
    img_data = normalization(img_data, mode)
    r = np.left_shift((img_data[...,0] / 8).astype("uint16"), 11)
    g = np.left_shift((img_data[...,1] / 4).astype("uint16"), 5)
    b = (img_data[...,2] / 8).astype("uint16")
    rgb565 = np.bitwise_or(np.bitwise_or(r, g), b)
    rgb565 = rgb565.flatten()
    rgb565.tofile(imagefile+'.rgb565.bin')


if __name__ == '__main__' :
    imagefile = 'mytest1.png'
    convert_to_rgb565("RGB",imagefile,(640,480),False,'Nothing')
```

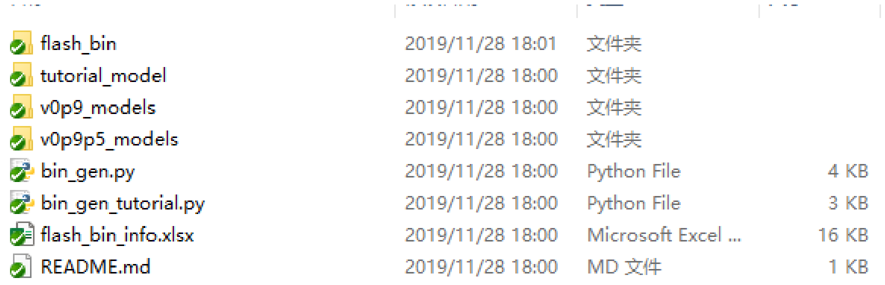
3.2 Generate flash bin file

Generate flash bin for all (firmware, model, model info, test image)

1. Go to utils/bin_gen/ directory
2. Copy all_models.bin and fw_info.bin into flash_bin directory
3. Run bin_gen_tutorial.py to generate flash bin file, the command as follows,

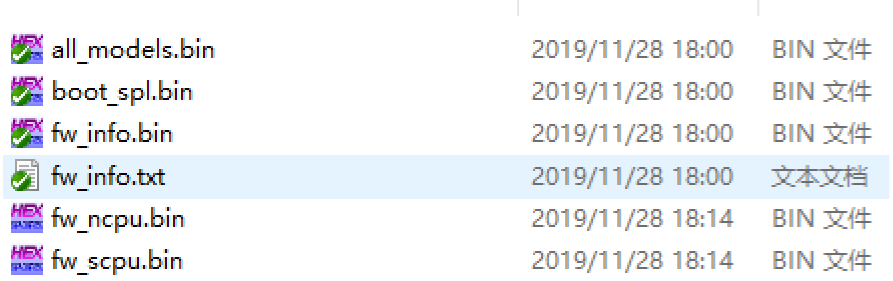
```
python3 bin_gen_tutorial.py
```

4. File ‘flash_fdfr_image.bin’ will be generated



flash_bin	2019/11/28 18:01	文件夹	
tutorial_model	2019/11/28 18:00	文件夹	
v0p9_models	2019/11/28 18:00	文件夹	
v0p9p5_models	2019/11/28 18:00	文件夹	
bin_gen.py	2019/11/28 18:00	Python File	4 KB
bin_gen_tutorial.py	2019/11/28 18:00	Python File	3 KB
flash_bin_info.xlsx	2019/11/28 18:00	Microsoft Excel ...	16 KB
README.md	2019/11/28 18:00	MD 文件	1 KB

Check flash_bin directory which contains all the file need to concatenate:



all_models.bin	2019/11/28 18:00	BIN 文件
boot_spl.bin	2019/11/28 18:00	BIN 文件
fw_info.bin	2019/11/28 18:00	BIN 文件
fw_info.txt	2019/11/28 18:00	文本文档
fw_ncpu.bin	2019/11/28 18:14	BIN 文件
fw_scpu.bin	2019/11/28 18:14	BIN 文件

Source code for reference:

```
flash_fdfr_bin_info = [
                [0x00000000, 8*1024,  './flash_bin/boot_spl.bin'],
                [0x00002000, 88*1024,  './flash_bin/fw_scpu.bin'],
                [0x00018000, 64*1024, './flash_bin/fw_ncpu.bin'],
                [0x00100000, 232, './flash_bin/fw_info.bin'],
                [0x00101000, 9850100, './flash_bin/all_models.bin'],
                [0x00A66000, 200704, './flash_bin/image.bin']
                ]

def file_read_binary(src):
    ''' Load depth file '''
```

```
     with open(src, "rb") as f:
         data = f.read()
         return data

def write_data_to_binary_file(ofile, data):
    with io.open(ofile, 'wb') as fo:
        fo.write(bytearray(data))

def bin_gen(bin_info, ofile):
    bin_len = len(bin_info)
    model_size = 0;
    if os.path.isfile(bin_info[-1][2]):
        model_size = os.path.getsize(bin_info[-1][2])
        #print('model size = %d' %model_size)
        if(model_size >= MAX_MODEL_SIZE):
            print('[ERROR] model file size is too big')
            sys.exit(-1)
    else:
        print('model file is not available')
        sys.exit(-1)

    total_bin_size = bin_info[-1][0] + model_size
    bin_data = np.zeros(total_bin_size, dtype=np.uint8)
    bin_data.fill(0xff)
    for bin_file in bin_info:
        print('start parsing, start: 0x{:08X}, size: {:08X}, file={:s}'.format(bin_file[0],
bin_file[1], bin_file[2]))
        bin_start_addr = bin_file[0]
        bin_file = bin_file[2]

        if os.path.isfile(bin_file) == 0:
            print('{:s} is not exists'.format(bin_file))
        else:
            rdata = file_read_binary(bin_file)
            rdata = list(rdata)
            rdata = np.array(rdata)
            bin_data[bin_start_addr:bin_start_addr + len(rdata)] = rdata

    write_data_to_binary_file(ofile, bin_data)
```

3.3 Upload model into board flash memory

Upload model by the flash_programmer (uart):

Prerequisite:

1. bring board up and PC need connect to uart
2. uart firmware is running on the board
3. go to utils/flash_programmer directory

Command:

```
python3 flash_programmer.py -a xxxx.bin
```

Upload model by the following command (usb):

Prerequisite:

1. bring board up and PC need connect to board through usb cable
2. companion mode firmware is running on the board
3. go to utils/ flash_model directory
4. make sure flash_fdfr_image.bin is exist on the directory

Command:

```
python3 mozart.py -f flash_models.py
```

4. SCPU

SCPU is responsible for loading firmware information, model data, and input images to assigned memory address, launching application, communicating with NCPU, and sending back the output results to Host.

4.1 Sample code

The full version of code in

customer_tutorial: scpu\project\customer_tutorial\main\customizedCode.c,

companion_tutorial: scpu\project\ companion_tutorial \main\customizedCode.c

```
#include "kdp_memxfer.h"    //for flash access
#include "kdp_memory.h"     //for ddr functions
#include "dbg.h"
#include "kdp_ddr_table.h"  //for default ddr table
#include "kdp_app.h"
…
void customer_model_run(void *argument) {

    struct yolo_result_s_l* p_fd_out_s = (struct yolo_result_s_l*)kdp_ddr_reserve(sizeof(struct yolo_result_s_l));

    kdp_memxfer_module.init(MEMXFER_OPS_DMA, MEMXFER_OPS_DMA);
    // specified flash address for image
    kdp_memxfer_module.flash_to_ddr(KDP_DDR_BASE, 0xA66000, 200704 );
    int ret = kdp_app_fd_customer_l( (yolo_result_s_l_t *)p_fd_out_s);

    struct yolo_result_s* p_fd_output_p =(struct yolo_result_s*) p_fd_out_s;
    DSG("[CUSTOMER] OBJ_DET (classCount,BoxCount)= %d,%d\n",
        p_fd_output_p->class_count,
        p_fd_output_p->box_count);

    if (p_fd_output_p->box_count > 0) {
        int print_count = (p_fd_output_p->box_count > 3) ? 3 : p_fd_output_p->box_count;
        DSG("(print 3 boxes at most)\n");

        for (int i = 0; i < print_count; i++) {
            DSG("box[%d]:class=%d, (x1,y1,x2,y2) = (%f, %f, %f, %f) @ score=%f\n", i,
                p_fd_output_p->boxes[i].class_num,
                p_fd_output_p->boxes[i].x1,
                p_fd_output_p->boxes[i].y1,
                p_fd_output_p->boxes[i].x2,
                p_fd_output_p->boxes[i].y2,
                p_fd_output_p->boxes[i].score);
        }
    }
}
```

4.2 Allocate memory address for model output and input, load image from flash memory

Allocate memory by `kdp_ddr_reserve()`.

Load image from flash by `flash_to_ddr()`

```
p_fd_output_p= (struct yolo_result_s_l*) kdp_ddr_reserve (sizeof(struct yolo_result_s_l));
  kdp_memxfer_module.init(MEMXFER_OPS_DMA, MEMXFER_OPS_DMA);
//  0xA66000 : image address in flash
// 200704: size of image(byte)
// KDP_DDR_BASE : ddr address
  kdp_memxfer_module.flash_to_ddr(KDP_DDR_BASE, 0xA66000, 200704 );

```

4.3 Config image

For running customer model, need specify input size, channel and format, address of data.

```
kdp_img_conf_t img_setting ;
  img_setting.image_col      = 224;
  img_setting.image_row      = 224;
```

```
    img_setting.image_ch        = 4;
    img_setting.image_format    = NPU_FORMAT_RGBA8888 | IMAGE_FORMAT_SUB128 | IMAGE_FORMAT_BYPASS_PRE ;
//RGB 565
    img_setting.image_memory_address  = KDP_DDR_BASE;
```

Image Format

Image format flags is defined as below.

```
  // SDK code snippet
/* Image format flags */  #define IMAGE_FORMAT_SUB128                 BIT31
#define IMAGE_FORMAT_ROT_MASK               (BIT30 | BIT29)
#define IMAGE_FORMAT_ROT_SHIFT              29
#define IMAGE_FORMAT_ROT_CLOCKWISE          0x01
#define IMAGE_FORMAT_ROT_COUNTER_CLOCKWISE  0x02

#define IMAGE_FORMAT_RAW_OUTPUT             BIT28

#define IMAGE_FORMAT_CHANGE_ASPECT_RATIO    BIT20

#define IMAGE_FORMAT_BYPASS_PRE             BIT19
#define IMAGE_FORMAT_BYPASS_NPU_OP          BIT18
#define IMAGE_FORMAT_BYPASS_CPU_OP          BIT17
#define IMAGE_FORMAT_BYPASS_POST            BIT16

#define IMAGE_FORMAT_NPU                    0x00FF
#define NPU_FORMAT_RGBA8888         0x00
#define NPU_FORMAT_NIR              0x20
/* Support YCBCR (YUV) */
#define NPU_FORMAT_YCBCR422         0x30
#define NPU_FORMAT_YCBCR444         0x50
#define NPU_FORMAT_RGB565           0x60

/* Determine the exact format with the data byte sequence in DDR memory: [lowest byte]...[highest byte]
*/
#define NPU_FORMAT_YCBCR422_CRY1CBY0 0x30
#define NPU_FORMAT_YCBCR422_CBY1CRY0 0x31
#define NPU_FORMAT_YCBCR422_Y1CRY0CB 0x32
#define NPU_FORMAT_YCBCR422_Y1CBY0CR 0x33
#define NPU_FORMAT_YCBCR422_CRY0CBY1 0x34
#define NPU_FORMAT_YCBCR422_CBY0CRY1 0x35
#define NPU_FORMAT_YCBCR422_Y0CRY1CB 0x36
#define NPU_FORMAT_YCBCR422_Y0CBY1CR 0x37  // Y0CbY1CrY2CbY3Cr...

```

4.4 Config model running environment

Call kdp_app_config_model_input_output() to configure model running environment.

```

/*
* model_type_p : model id
* p_img_setting_p: input image setting
* p_fd_output_p: output address
*/
kdp_app_config_model_input_output(model_type_p, p_img_setting_p ,p_fd_output_p);
```

4.5 Model manager run

Call kdp_app_run_model to send command to NPU for running model.

```
status = kdp_app_run_model();
```

4.6 Checkout network output (depends on post process in ncpu)

```
//output result
struct yolo_result_s* p_fd_output_p =(struct yolo_result_s*) p_fd_out_s;
dbg_msg("[COM LUKE] OBJ_DET (classCount,BoxCount)= %d,%d\n",
  p_fd_output_p->class_count,
  p_fd_output_p->box_count);
dbg_msg("box[%d]:class=%d, (x1,y1,x2,y2) = (%f, %f, %f, %f) @ score=%f\n", i,  p_fd_output_p->boxes[i].class_num,
           p_fd_output_p->boxes[i].x1,
            p_fd_output_p->boxes[i].y1,
            p_fd_output_p->boxes[i].x2,
            p_fd_output_p->boxes[i].y2,
            p_fd_output_p->boxes[i].score);
```

5 NCPU and NPU

NCPU is responsible for communicating with SCPU and NPU, registering new preprocess/cpu_op/postprocess functions, setting model, running preprocess/cpu_op/post_process, and enabling NPU. NPU is responsible for running npu_op, interrupting when meeting cpu_op, and continuing running npu_op.

6 Memory

The DDR memory is the space which can be accessed by SCPU, NCPU and NPU. The firmware information, model data, input images, and output results are located in this space.

7 Limitations for KL520 NPU pre-process

(1) The input width and height of models is limited to be less than 256.

The model’s input width and height need to be less than 256. In this tutorial, the input dimension is 3x224x224 for Tiny-YOLO-V3.

8 CPU and Output Node Info

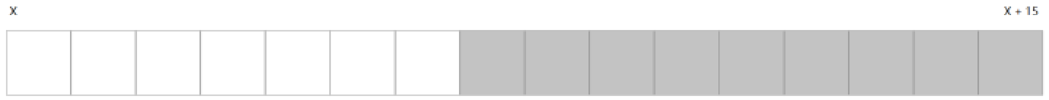
The CPU and Output Node info is shown in ioinfo.csv. For Tiny-YOLO-V3, the ioinfo is summarized as below. From this table, cpu_op upsample need to be implemented and there are 2 output node used for postprocessing in this model.

Node Type	**Index**	**Name**	**Note**
c	0	up_sampling2d_1_o0	Upsample CPU Node
o	0	conv2d_10_o0	First Output Node
o	1	conv2d_13_o0	Second Output Node

9 Node Result Layout in Memory

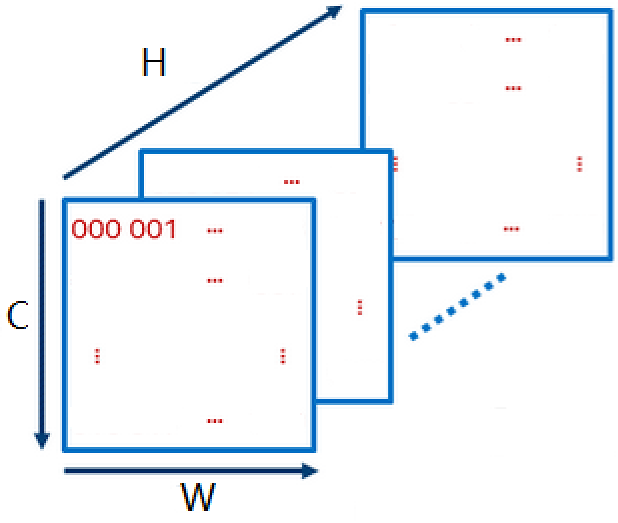
(1) For KL520, the results are in 16-byte aligned format for each row.

The following figure shows that if width mod 16 is equal to 7, then 9 bytes is inserted at the end of each row.



```
// SDK code snippet
/**
 * pad_up_16() - calculate the 16-bytes aligned width
 *
 * @a: the original width
 *
 * Return value:
 * >0 : aligned width
 */
inline static int pad_up_16(int a)
{
    return ceil((float)a / 16) * 16;
}
```

(2) For KL520, the data sequence of the CPU and Output node is HxCxW (Height*Channel*Width), which means that the data is stored in the sequence: row_1 of C1, row_1 of C2, ..., row_1 of Cn, row_2 of C1, row_2 of C2, ..., row_2 of Cn, row_H of C1, row_H of C2, ..., row_H of Cn.



(3)For KL520, the data type is fixed-point. To do postprocess, the fixed-point data need to be converted to float data with do_div_scale() function.

```
// SDK code snippet
/**
 * do_div_scale() - convert fixed point to float
 *
 * @v: fixed point value
 *
 * @div: 2 raised to the power of radix
 *
 * @scale: scale for fixed point
 *
 * Return float value
 */
static float do_div_scale(float v, int div, float scale)
{
    return ((v / div) / scale);
}
```

For example, the dimension of one output node for Tiny-YOLO-V3 is 7x255x7 (HxCxW), the actual memory size is 7x255x16. The dimension of another output node is 14x255x14 (HxCxW), the actual memory size is 14x255x16. The cpu_op and postprocess functions are based on this data layout.

10 Register Preprocess/CPU_Operation/Postprocess Functions

10.1 API Functions

API Functions	**Parameters**
int kdpio_cpu_op_register(int cpu_op, cpu_op_fn cof);	cpu_op: unique ID for this cpu_node cof: CPU operation function, such as **nearest_upsample_cpu**
int kdpio_pre_processing_register(int model_id, pre_proc_fn ppf);	model_id: unique ID for this model ppf: preprocess function, such as **kdp_preproc_inproc**
int kdpio_post_processing_register(int model_id, post_proc_fn ppf);	model_id: unique ID for this model ppf: postprocess function, such as **post_yolo_v3**

int kdp_preproc_inproc(int model_id, struct kdp_image_s *image_p);	model_id: unique ID for this model image_p: kdp_image_s struct with original image input

kdp_preproc_inproc() supports Kneron preprocess method (RGB/256 - 0.5).

10.2 Allocate Unique ID for New Model and New CPU_OP

(1) Allocate new model ID in /common/include/model_type.h.

In SDK, a new model ID 19 is assigned for TINY-YOLO-V3. This ID is consistent with the model ID in fw_info.bin.

```
enum model_type {
    INVALID_ID,
    KNERON_FDSMALLBOX                   = 1,
    KNERON_FDANCHOR                     = 2,
    KNERON_FDSSD                        = 3,
    AVERAGE_POOLING                     = 4,
    KNERON_LM_5PTS                      = 5,
    KNERON_LM_68PTS                     = 6,
    KNERON_LM_150PTS                    = 7,
    KNERON_FR_RES50                     = 8,
    KNERON_FR_RES34                     = 9,
    KNERON_FR_VGG10                     = 10,
    KNERON_TINY_YOLO_PERSON             = 11,
    KNERON_3D_LIVENESS                  = 12,
    KNERON_GESTURE_RETINANET            = 13,
    TINY_YOLO_VOC                       = 14,
    IMAGENET_CLASSIFICATION_RES50       = 15,
    IMAGENET_CLASSIFICATION_RES34       = 16,
    IMAGENET_CLASSIFICATION_INCEPTION_V3= 17,
    IMAGENET_CLASSIFICATION_MOBILENET_V2= 18,
    TINY_YOLO_V3                        = 19,
};
```

(2) Allocate new CPU_OP ID in /ncpu/rtos2/cpu_ops_ex.h.

For a new CPU node, the first step is to get support by Compiler. Then compiler assigns a unique cpu_op_type to this type of CPU node when generating all_models.bin.

As there is a new cpu_op nearest_upsample for TINY-YOLO-V3, cpu_op_type 100 is assigned for it. This op_type is consistent with the one in all_models.bin.

```
/* Type of CPU Operations */
enum cpu_op_type {
    SOFTMAX,
    FLATTERN                = 1,
    DROPOUT                 = 2,
    NEAREST_UPSAMPLE        = 100,
    BILINEAR_UPSAMPLE       = 101,
    MYSTERY                 = 1000,
};
```

10.3 Struct kdp_image_s

The struct kdp_image_s is a key structure which exchange data and parameters between different API functions of NCPU: kdpio_set_model, kdpio_run_preprocess, kdpio_run_npu_op, kdpio_run_cpu_op, and kdpio_run_postprocess.

```
/* Structure of kdp_model_dim */
struct kdp_model_dim_s {
    /* CNN input dimensions */
    uint32_t    input_row;
    uint32_t    input_col;
    uint32_t    input_channel;
};

/* Structure of kdp_pre_proc_s */
struct kdp_pre_proc_s {
    /* input image in memory for NPU */
    uint32_t    input_mem_addr;
    int32_t     input_mem_len;

    /* Input working buffers for NPU */
    uint32_t    input_mem_addr2;
    int32_t     input_mem_len2;

    /* number of bits for input fraction */
    uint32_t    input_radix;

    /* Other parameters for the model */
    void        *params_p;
};

/* Structure of kdp_post_proc_s */
struct kdp_post_proc_s {
    /* output number */
    uint32_t    output_num;

    /* output data memory from NPU */
    uint32_t    output_mem_addr;
    int32_t     output_mem_len;

    /* result data memory from post processing */
    uint32_t    result_mem_addr;
    int32_t     result_mem_len;

    /* data memory for post processing */
    uint32_t    output_mem_addr3;

    /* output data format from NPU
     *     BIT(0): =0, 8-bits
     *             =1, 16-bits
     */
    uint32_t    output_format;

    /* output node parameter */
    struct out_node_s     *node_p;

    /* Other parameters for the model */
    void        *params_p;
};

/* Structure of kdp_cpu_op_s */
struct kdp_cpu_op_s {
    /* cpu op node parameter */
    struct cpu_node_s     *node_p;
};

/* KDP image structure */
struct kdp_image_s {
    /* Original image and model */
    struct kdp_img_raw_s   *raw_img_p;
    struct kdp_model_s     *model_p;

    int         model_id;
    /* Setup memory address contains information for input node, cpu node, output node */
    char        *setup_mem_p;

    /* Model dimension */
    struct kdp_model_dim_s  dim;

    /* Pre process struct */
    struct kdp_pre_proc_s   preproc;

    /* Post process struct */
    struct kdp_post_proc_s   postproc;

    /* CPU operation struct */
    struct kdp_cpu_op_s     cpu_op;
};
```

10.4 Register Preprocess Function

If the new model’s preprocess method is different from Kneron’s, the corresponding preprocess function need to be implemented and registered.

The preprocess function can be implemented in /ncpu/rtos2/pre_process_ex.c and registered in /ncpu/rtos2/main.c. pre_process_ex.c and pre_process_ex.h need to be created.

For Tiny-YOLO-V3, RGBA image input is directly provided.

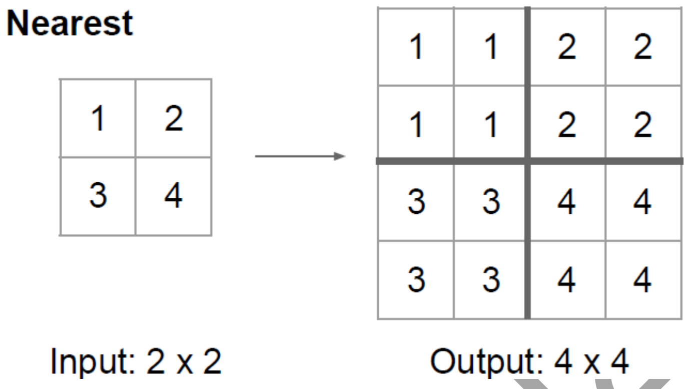
10.5 Register CPU_Operation Function

By following the results data layout in KL520 and cpu_op algorithm, the corresponding CPU_Operation function can be implemented and registered in SDK.

(1) Nearest Upsample

The following figure shows the nearest upsampling with scale factor of 2. For Tiny-YOLO-V3, the cpu_op function nearest_upsample_cpu() is implemented in

/ncpu/rtos2/cpu_ops_ex.c and registered in /ncpu/rtos2/main.c. The example code is in Chapter 6.2.4.



10.6 Register Postprocess Function

Different models have their specific postprocess method to output the detection results. By following results layout in KL520 and the postprocess algorithm, the corresponding postprocess function can be implemented and registered in SDK.

If bypass postprocess, SDK output the raw results of all output nodes. And no postprocess function is needed to registered for any models.

For Tiny-YOLO-V3, the postprocess function post_yolo_v3() is implemented in /ncpu/rtos2/post_processing_ex.c and registered in /ncpu/rtos2/main.c. The example code is in Chapter 6.2.5.

11 CPU_OP

CPU_OP function is registered in scpu_comm_thread(). The implementation of this function follows the rule of 16-bytes alignment and HCW data layout. Kdp_image_s struct provides all information needed for cpu_op: dimension of cpu node, radix, scale, input memory address, output memory address.

For Tiny-YOLO-V3, nearest_upsample_cpu() function is implemented and registered. The example of how to register it is shown in 6.2.1.

```
/**
 * nearest_upsample_cpu() - execute nearest upsampling CPU operations
 *
 * @image_p: pointer to struct kdp_image with buffer of raw image
 *          and dimension, and data for pre/cpu/post processing.
 *
 * @cpu_op: the CPU operation type id this function was registered for
 *
 * This is a cpu function which uses Kneron NCPU to do nearest upsampling.
 *
 * Return value:
 * 0 : success
 * <0 : error
 */
int nearest_upsample_cpu(int cpu_op, struct kdp_image_s *image_p) {
    // Get the input dimension
    int row_num_in = CPU_OP_NODE_INPUT_ROW(image_p);
    int col_num_in = CPU_OP_NODE_INPUT_COL(image_p);
    int chnl_num_in = CPU_OP_NODE_INPUT_CH(image_p);
    // Get the output dimension
    int row_num_out = CPU_OP_NODE_OUTPUT_ROW(image_p);
    int col_num_out = CPU_OP_NODE_OUTPUT_COL(image_p);
    // Calculate the input and output 16-bytes aligned column size
    int col_num_pad_in = pad_up_16(col_num_in);
    int col_num_pad_out = pad_up_16(col_num_out);

    int oi, oj, ii, ij, k;
    // Calculate the upsampling scales
    int h_scale = row_num_out / row_num_in;
    int w_scale = col_num_out / col_num_in;
    int out_index, in_index;
    // Get the input memory address and output memory address
    uint8_t *data_array = (uint8_t*)CPU_OP_NODE_INPUT_ADDR(image_p);
    uint8_t *dst_data_array = (uint8_t*)CPU_OP_NODE_OUTPUT_ADDR(image_p);
    // HCW for DRAM
    for (oj = 0; oj < row_num_out; oj++) {
        for (k = 0; k < chnl_num_in; ++k) {
            for (oi = 0; oi < col_num_out; oi++) {
                ij = oj / h_scale;
                ii = oi / w_scale;
                // Calculate the output index. As each row is 16-bytes aligned, col_num_pad_out is used
                // to calculate the output location
                out_index = oi + col_num_pad_out * (k + chnl_num_in * oj);

                // Calculate the input index. As each row is 16-bytes aligned, col_num_pad_in is used
                // to calculate the input location
                in_index = ii + col_num_pad_in * (k + chnl_num_in * ij);

                // Fill the input data to the output location
                dst_data_array[out_index] = data_array[in_index];
            }
        }
    }
    return 0;
}
```

12 Post-processing

Postprocess function is registered in scpu_comm_thread(). The implementation of this function follows the rule of 16-bytes alignment and HCW data layout.

For Tiny-YOLO-V3, post_yolo_v3() function is implemented and registered. The example of how to register it is shown in 6.2.1.

Part 1 of post_yolo_v3

This part of code do preparation for the postprocessing logic: get the result memory address; calculate total class of this model.

```
/**
 * post_yolo_v3() - post process function for yolo v3
 *
 * @model_id: the model id this function was registered for
 *
 * @image_p: pointer to struct kdp_image with buffer of raw image
 *          and dimension, and data for pre/cpu/post processing.
 *
 * This is a post process function which outputs the detection results.
 *
 * Return value:
 * 0 : success
 * <0 : error
 */
```

```
int post_yolo_v3(int model_id, struct kdp_image_s *image_p)
{
    struct yolo_v3_post_globals_s *gp = &u_globals.yolov3;
    int i, j, k, div, ch, row, col, max_score_class, good_box_count, class_good_box_count, good_result_count, len;
    float box_x, box_y, box_w, box_h, box_confidence, max_score;
    int8_t *src_p, *x_p, *y_p, *width_p, *height_p, *score_p, *class_p;
    struct bounding_box_s *bbox;
    struct yolo_result_s *result;
    struct bounding_box_s *result_box_p, *result_tmp_p, *r_tmp_p;

    int data_size, grid_w, grid_h, class_num, grid_w_bytes_aligned, w_bytes_to_skip;
    int incr_off;

    // Get the data size, the data size is 1 byte for KL520
    data_size = (POSTPROC_OUTPUT_FORMAT(image_p) & BIT(0)) + 1;     /* 1 or 2 in bytes */

    // Get the result memory address
    result = (struct yolo_result_s *)(POSTPROC_RESULT_MEM_ADDR(image_p));
    result_box_p = result->boxes;

    result_tmp_p = gp->result_tmp_s;

    // Calculate the total classes for this model
    class_num = POSTPROC_OUT_NODE_CH(image_p) / YOLO_V3_CELL_BOX_NUM - YOLO_BOX_FIX_CH;

    result->class_count = class_num;
    bbox = gp->bboxes_v3;
    good_box_count = 0;

    float anchers_v[3][2];
    int idx;
    int offset = sizeof(struct out_node_s);
    struct out_node_s *out_p;
```

Part 2 of post_yolo_v3

This part of code calculates all valid detection boxes from the 2 output nodes of Tiny-YOLO-V3: calculate the aligned byte size of a row; get the radix and scale of output node; convert fixed point value back to float; calculate all valid detection boxes. Kdp_image_s struct provides all information needed for postprocessing: dimension of output node, radix, scale, output memory address, result memory address.

```
/* Second part of post_yolo_v3 */
```

```

    // Get the valid detections from the 2 output nodes
    for (idx = 0; idx < POSTPROC_OUTPUT_NUM(image_p); idx++) {
        // Get the pointer to output node
        out_p = (struct out_node_s *)((uint32_t)POSTPROC_OUT_NODE(image_p) + idx * offset);

        // Get the pointer to output memory address of output node
        src_p = (int8_t *)OUT_NODE_ADDR(out_p);

        // Get width and height of output node
        grid_w = OUT_NODE_COL(out_p);
        grid_h = OUT_NODE_ROW(out_p);
        // Calculate the byte size of a row
        len = grid_w * data_size;
        // Calculate the aligned byte size of a row: 16-byte alignment
        grid_w_bytes_aligned = round_up(len);
        // Calculate the skipped bytes for each row
        w_bytes_to_skip = grid_w_bytes_aligned - len;
        len = grid_w_bytes_aligned;

        // Get the start address of x, y, w, h, confidence_score, class probability
        x_p = src_p;
        y_p = x_p + len;
        width_p = y_p + len;
        height_p = width_p + len;
        score_p = height_p + len;
        class_p = score_p + len;

        // Get the radix of output node and calculate 2 raised to the power of radix
        div = 1 << OUT_NODE_RADIX(out_p);
        // Get the scale of output node
        float fScale = *(float *)&OUT_NODE_SCALE(out_p);

        // Set different anchors for output node
        if (0 == idx) {
            memcpy(anchers_v, anchers_v0, 6*sizeof(float));
        } else {
            memcpy(anchers_v, anchers_v1, 6*sizeof(float));
        }

        // The data layout is in HCW
        for (row = 0; row < grid_h; row++) {
            for (ch = 0; ch < YOLO_V3_CELL_BOX_NUM; ch++) {
                for (col = 0; col < grid_w; col++) {
                    if (data_size == 1) {
                        box_x = (float)*x_p;
                        box_y = (float)*y_p;
                        box_w = (float)*width_p;
                        box_h = (float)*height_p;
                        box_confidence = (float)*score_p;
                    } else {
                        box_x = (float)*(uint16_t *)x_p;
                        box_y = (float)*(uint16_t *)y_p;
                        box_w = (float)*(uint16_t *)width_p;
                        box_h = (float)*(uint16_t *)height_p;
                        box_confidence = (float)*(uint16_t *)score_p;
                    }

                    // Calculate the confidence score with sigmoid function
                    box_confidence = sigmoid(do_div_scale(box_confidence, div, fScale));

                    /* Get scores of all class */
                    for (i = 0; i < class_num; i++) {
                        if (data_size == 1)
                            gp->box_class_probs[i] = (float)*(class_p + i * len);
                        else
                            gp->box_class_probs[i] = (float)*(uint16_t *)(class_p + i * len );
                        // Calculate the class probability with sigmoid function
                        gp->box_class_probs[i] = sigmoid(do_div_scale(gp-
>box_class_probs[i], div, fScale));
                    }

                    //increase pointer to next position
                    x_p += data_size;
                    y_p += data_size;
                    width_p += data_size;
                    height_p += data_size;
                    score_p += data_size;
                    class_p += data_size;

                    /* Find highest score class */
                    max_score = 0;
                    max_score_class = -1;
                    for (i = 0; i < class_num; i++) {
                        gp->box_class_probs[i] = gp->box_class_probs[i] * box_confidence;
                        if (gp->box_class_probs[i] >= prob_thresh_yolov3) {
                            if (gp->box_class_probs[i] >= prob_thresh_yolov3 && gp-
>box_class_probs[i] > max_score) {
                                max_score = gp->box_class_probs[i];
                                max_score_class = i;
                            }
                        }
                    }
                    /* Calculate the valid detection box: top-left, right-bottom */
                    if (max_score_class != -1) {
                        box_x = do_div_scale(box_x, div, fScale);
                        box_y = do_div_scale(box_y, div, fScale);
```

```
                      box_w = do_div_scale(box_w, div, fScale);
                      box_h = do_div_scale(box_h, div, fScale);

                        box_x = (sigmoid(box_x) + col) / grid_w;
                        box_y = (sigmoid(box_y) + row) / grid_h;
                        box_w = expf(box_w) * anchers_v[ch][0] / DIM_INPUT_COL(image_p);
                        box_h = expf(box_h) * anchers_v[ch][1] / DIM_INPUT_ROW(image_p);

                        bbox->x1 = (box_x - (box_w / 2)) * DIM_INPUT_COL(image_p);
                        bbox->y1 = (box_y - (box_h / 2)) * DIM_INPUT_ROW(image_p);
                        bbox->x2 = (box_x + (box_w / 2)) * DIM_INPUT_COL(image_p);
                        bbox->y2 = (box_y + (box_h / 2)) * DIM_INPUT_ROW(image_p);
                        bbox->score = gp->box_class_probs[max_score_class];
                        bbox->class_num = max_score_class;

                        bbox++;
                        good_box_count++;
                    }
                }

                /* go to next grid_w*85 values for next anchors*/
                // The output dimension is grid_h * 255 * grid_w, and there are 3 anchors
                // The data sequence is:
                // 1st anchor: row of x, row of y, row of w, row of h, row of confidence score, row of class_prob1, ..., row of class_prob80;
                // 2nd anchor: row of x, row of y, row of w, row of h, row of score, row of class1, ..., row of class80;
                // 3rd anchor: row of x, row of y, row of w, row of h, row of score, row of class1, ..., row of class80;
                incr_off = w_bytes_to_skip + grid_w_bytes_aligned*84; //84 = 85 -1
                x_p += incr_off;
                y_p += incr_off;
                width_p += incr_off;
                height_p += incr_off;
                score_p += incr_off;
                class_p += incr_off;
            }
        }

    }

    good_result_count = 0;
```

Part 3 of post_yolo_v3

This part of code do non max suppression (NMS) on all valid detection boxes. Then final detection boxes are generated.

```
/* Third part of post_yolo_v3 */

    // Do NMS for all valid detection boxes, and get the final good result
    for (i = 0; i < class_num; i++) {
        bbox = gp->bboxes_v3;
        class_good_box_count = 0;
        r_tmp_p = result_tmp_p;

        for (j = 0; j < good_box_count; j++) {
            if (bbox->class_num == i) {
                memcpy(r_tmp_p, bbox, sizeof(struct bounding_box_s));
                r_tmp_p++;
                class_good_box_count++;
            }
            bbox++;
        }

        if (class_good_box_count == 1) {
            memcpy(&result_box_p[good_result_count], &result_tmp_p[0], sizeof(struct bounding_box_s));
            good_result_count++;
        } else if (class_good_box_count >= 2) {
            qsort(result_tmp_p, class_good_box_count, sizeof(struct bounding_box_s), box_score_comparator);
            for (j = 0; j < class_good_box_count; j++) {
                if (result_tmp_p[j].score == 0)
                    continue;
                for (k = j + 1; k < class_good_box_count; k++) {
                    if (box_iou(&result_tmp_p[j], &result_tmp_p[k]) > nms_thresh_yolov3) {
                        result_tmp_p[k].score = 0;
                    }
                }
            }

            for (j = 0; j < class_good_box_count; j++) {
                if (result_tmp_p[j].score > 0) {
                    memcpy(&result_box_p[good_result_count], &result_tmp_p[j], sizeof(struct bounding_box_s));
                    good_result_count++;
                }
            }
        }
    }

    dbg_msg("good_result_count: %d\n", good_result_count);
    result->box_count = good_result_count;
    for (unsigned int i = 0; i < good_result_count; i++) {
        dbg_msg("post_yolo3 %f %f %f %f %f %d\n", result->boxes[i].x1, result->boxes[i].y1, result->boxes[i].x2, result->boxes[i].y2, result->boxes[i].score, result->boxes[i].class_num);
    }
    len = good_result_count * sizeof(struct bounding_box_s);

    return len;
}
```