**Kneron Inc**

Document Name: **KDP KDPIO Library**

# KDP KDPIO Library

## Kneron Inc

Engineering Design Document

# Table of Contents

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to define the API and data structures of KDPIO (Kneron NPU Processor IO) Library and how to use the API in associated CPU environment. It is intended as reference for the development of Kneron NPU based model drivers for pre-processing, post-processing, cpu-operation and also system tuning.

## 1.2 Scope

The scope of this document is mostly the perspective of the CPU (NCPU) with Kneron NPU.

# 2 Reference

Kneron KL520 Design Specification, Rev. 0.5, Feb. 2019 (Internal)

# 3 Acronyms, Abbreviations, Definitions

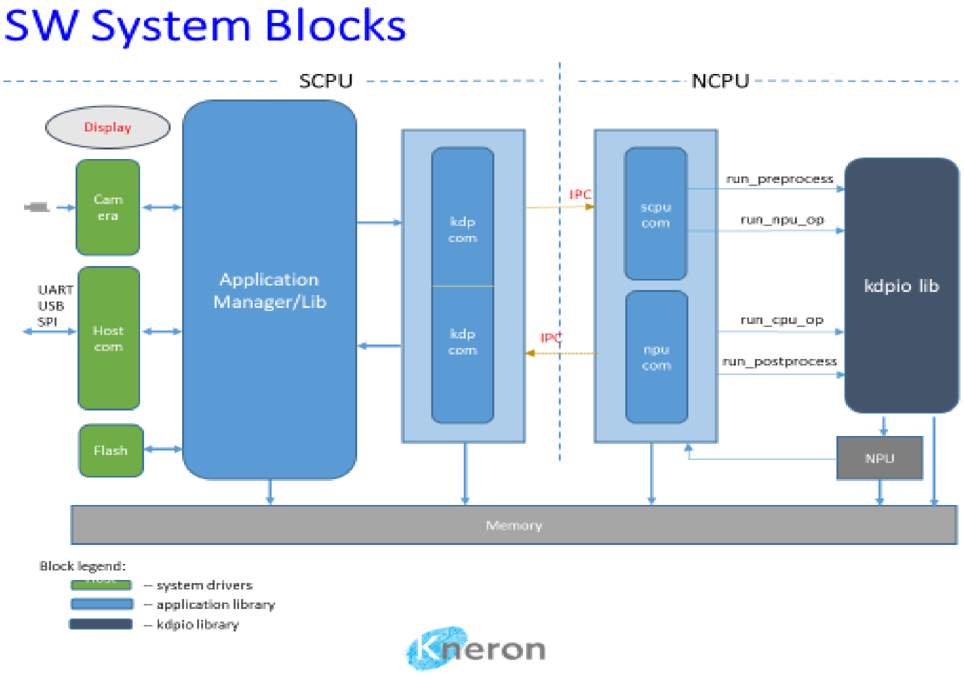
KDPIO – Kneron Development Platform/Processor Input/Output
KL520 – Kneron SOC 520
NCPU – NPU CPU
SCPU – System CPU
FID – Face Identification
FD – Face Detection
LM – Landmark
LV – Liveness
FR – Face Recognition
CNN – Convolutional Neuron Network

# 4 Architecture

## 4.1 Overall Architecture

KDPIO library is inner-most layer close to Kneron NPU in overall architecture. It is responsible to take in CNN model, input frame of image and parameters, and pass them to Kneron NPU processor to run. After NPU is done, KDPIO library or external post-processing code will process and pass the result out to SCPU.

Memories to save all models' data like weight.bin, command.bin, setup.bin, input/output buffers, and raw images are all allocated in DDR memory block by SCPU, and their addresses passed to NCPU through IPC shared memory.



## 4.2 NCPU Architecture

On NCPU there are two main parts of software blocks, one is kdpio library, and another part is for the communication with SCPU and interrupt/OS handling.

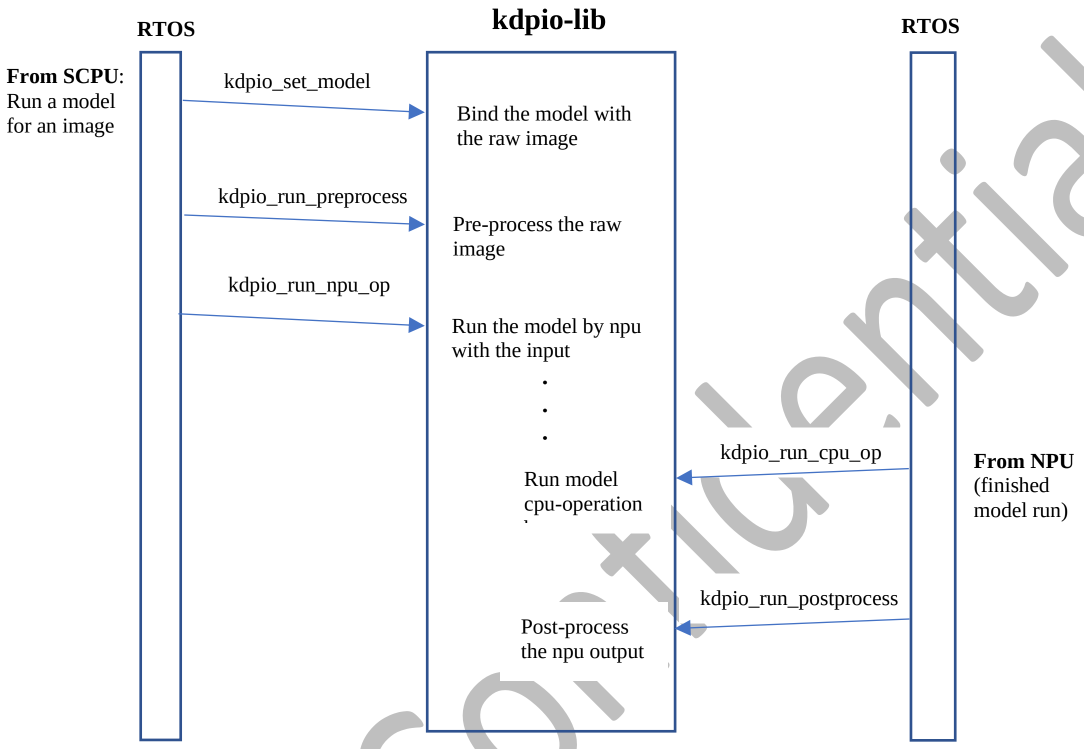
Current design has one thread to listen to the commands from SCPU and to call APIs of kdpio lib; another thread to listen to NPU notification and to decide to do cpu operation or post processing, eventually to notify SCPU on the result.

Communication with SCPU (IPC – Inter Processor Communication) is through two interrupt lines and two shared memory blocks, each for one direction of the communication.

Interrupts from SCPU and from NPU are for notifications. Corresponding processing is handled by their respective threads. If needed for the design of an application, NPU processing and CPU post processing can be parallelized, and the thread priorities could be adjusted for best performance.

For power saving purposes, OS is made tickless and will enter idle mode as often as possible.

## 4.3 KDPIO Operational Call Flow


RTOS
kdpio-lib
RTOS
From SCPU: Run a model for an image
kdpio_set_model
Bind the model with the raw image
kdpio_run_preprocess
Pre-process the raw image
kdpio_run_npu_op
Run the model by npu with the input
.
.
.
kdpio_run_cpu_op
Run model cpu-operation
From NPU (finished model run)
kdpio_run_postprocess
Post-process the npu output

# 5 API

## 5.1 Init API

### 5.1.1 kdpio_init

```
/**
 * kdpio_init() - initialize kdpio
 *
 * This function tells kdpio to initialize the platform and
 * resources for NPU support.
 *
 */
void kdpio_init(void);
```

### 5.1.2 kdpio_exit

```
/**
 * kdpio_exit() - exit kdpio
 *
 * This function tells kdpio to free allocated resources and
 * quit from NPU support.
 *
 */
void kdpio_exit(void);
```

## 5.2 Runtime API

### 5.2.1 kdpio_set_model

```
/**
 * kdpio_set_model() - set the model for an input image
 *
 * @image_p: pointer to struct kdp_image with buffer of raw image
 *          and dimension, and data for pre/cpu/post processing.
 *
 * @model_p: pointer to struct kdp_model with buffers of setup,
 *          command, weights, input and output for npu.
 *
 * This function tells kdpio the CNN model to use for processing next
 * image(s).
 *
 * Return value:
 *  none
 */
void kdpio_set_model(struct kdp_image *image_p, struct kdp_model *model_p);
```

### 5.2.2 kdpio_run_preprocess

```
/**
 * kdpio_run_preprocess() - run preprocessing for the image
 *
 * @image_p: pointer to struct kdp_image with buffer of raw image
 *          and dimension, and data for pre/cpu/post processing.
 *
 * This function tells kdpio to pre-process the raw image for npu
 * before npu running the model. Its output will be put in model's
 * input buffer for npu.
 *
 * Return value:
 * 0 : success
 * <0 : error
 */
int kdpio_run_preprocess(struct kdp_image *image_p);
```

### 5.2.3 kdpio_run_npu_op

```
/**
 * kdpio_run_npu_op() - run cnn model in npu for the image
 *
 * @image_p: pointer to struct kdp_image with buffer of raw image
 *         and dimension, and data for pre/cpu/post processing.
 *
 * This function tells kdpio to run NPU to process the input data
 * with the cnn model previously set.
 *
 * Return value:
 * 0 : success
 * <0 : error
 */
int kdpio_run_npu_op(struct kdp_image *image_p);
```

### 5.2.4 kdpio_run_cpu_op

```
/**
 * kdpio_run_cpu_op() - run cpu operation for the image
 *
 * @image_p: pointer to struct kdp_image with buffer of raw image
 *         and dimension, and data for pre/cpu/post processing.
 *
 * This function tells kdpio to let cpu run the input image when
 * npu finishes its running. If there is no cpu operation to run as
 * specified by model compiler, the function will still return
 * success so that next step (postprocess) can continue.
 *
 * Return value:
 * 0 : success
 * <0 : error
 */
int kdpio_run_cpu_op(struct kdp_image *image_p);
```

### 5.2.5 kdpio_run_postprocess

```
/**
 * kdpio_run_postprocess() - run postprocessing for the image
 *
 * @image_p: pointer to struct kdp_image with buffer of raw image
 *         and dimension, and data for pre/cpu/post processing.
 *
 * @perf_improv_fn: pointer to pre_post_fn callback function to move
 *         npu output to additional buffer for postprocessing while
 *         the original one could be used by npu again.
 *         This is intended to improve fps performance if possible
 *         and desired. Additional output buffer needs to be allocated
 *         for the purpose, and DMA could be used in the callback.
 *
 * This function tells kdpio to post-process the output data from npu
 * before returning to the calling system.
 *
 * Return value:
 * 0 : success
 * <0 : error
 */
int kdpio_run_postprocess(struct kdp_image *image_p, pre_post_fn perf_improv_fn);
```

## 5.3 Registration API

### 5.3.1 kdpio_pre_processing_register

Register a preprocessing callback for a model type. The callback will be passed in with the registered model type and a pointer to struct kdp_image with needed parameters.

### 5.3.2 kdpio_pre_processing_unregister

Unregister a preprocessing callback for a model type.

### 5.3.3 kdpio_cpu_op_register

Register a cpu operation callback for a cpu-op type. The callback will be passed in with the registered cpu-op type and a pointer to struct kdp_image with needed parameters.

### 5.3.4 kdpio_cpu_op_unregister

Unregister a cpu-operation callback for a cpu-op type.

### 5.3.5 kdpio_post_processing_register

Register a postprocessing callback for a model type. The callback will be passed in with the registered model type and a pointer to struct kdp_image with needed parameters.

### 5.3.6 kdpio_post_processing_unregister

Unregister a postprocessing callback for a model type.

## 5.4 Data structures

### 5.4.1 System IPC data

**Structure of SCPU to NCPU Message:**

```
struct scpu_to_ncpu {
        uint32_t    id;         /* = 'scpu' */
        uint32_t    version;
        uint32_t    cmd;            // Run / Stop
        uint32_t    input_count;    // # of input image
        uint32_t    input_flags;    // reserved

        /*
         * debug control flags (dbg.h):
         *   bits 19-16: scpu debug level
         *   bits 03-00: ncpu debug level
         */
        uint32_t    debug_flags;

        /* Active models in memory and running */
        int32_t     num_models;  //usually, num_models=1 (only one active model)
        struct kdp_model    models[MULTI_MODEL_MAX];            //to save active modelInfo
        int32_t     models_type[MULTI_MODEL_MAX];       //to save model type
        int32_t     model_slot_index;  //usually, model_slot_index = 0 (index for above 2 arrays)

        /* Images being processed via IPC */
        int32_t             active_img_index;
        struct kdp_img_raw  raw_images[IPC_IMAGE_MAX];

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

        uint32_t    output_mem_addr2;
        int32_t     output_mem_len2;
};
```

**Structure of NCPU to SCPU Message:**

```
struct ncpu_to_scpu {
        uint32_t    id;         /* = 'ncpu' */
        uint32_t    version;
        int32_t     status;             // status for the cmd from scpu

        /* Images result info corresponding to raw_images[] */
        int32_t                 img_index_done;
        struct kdp_img_result   img_results[IPC_IMAGE_MAX];
};
```

### 5.4.2 kdpio-lib data

**CNN model setup data structures to be parsed by kdpio-lib:**

```
struct setup_struct {
        struct cnn_header       header;

        union {
            struct in_node      in_nd;
            struct out_node     out_nd;
            struct cpu_node     cpu_nd;
        } nodes[1];
};
```

**Data structures passed to preprocessing, postprocessing or cpu-operation:**

```
struct kdp_image {
        /* Original image and model */
        struct kdp_img_raw          *raw_img_p;
        struct kdp_model            *model_p;

        int         model_id;
        char        *setup_mem_p;

        /* Model dimension */
        struct kdp_model_dim    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;
};
```

# 6 OS Processing

## 6.1 Initialize kdpio-lib

OS needs to initialize kdpio library before using it:

```
kdpio_init();
```

## 6.2 Any model with any input image

Compiled CNN model (multiple files) for Kneron SOC needs to be saved in memory from flash or from host. An AI application could have multiple models, and all of them should be saved in memory for performance of inference.

When NCPU receives a CMD_RUN_NPU command, it should find the model:

```
model_slot_index = in_comm_p->model_slot_index;
model_p = &in_comm_p->models[model_slot_index];
```

move model data to data structs:

```
PREPROC_INPUT_MEM_ADDR(npu_img_p) = model_p->input_mem_addr;
PREPROC_INPUT_MEM_LEN(npu_img_p) = model_p->input_mem_len;
POSTPROC_RESULT_MEM_ADDR(npu_img_p) = raw_img_p->results[model_slot_index].result_mem_addr;
POSTPROC_RESULT_MEM_LEN(npu_img_p) = raw_img_p->results[model_slot_index].result_mem_len;
```

and bind the model with the input image:

```
kdpio_set_model(npu_img_p, model_p);
```

finally call pre-processing API and npu operation API in this order:

```
kdpio_run_preprocess(npu_img_p);
kdpio_run_npu_op(npu_img_p);
```

## 6.3 Continuous and parallel processing

Raw images can be passed in to NCPU to run continuously. NCPU will use the newest active image index to get the raw image and pass to kdpio lib.

```
image_index = in_comm_p->active_img_index;
raw_img_p = &in_comm_p->raw_images[image_index];
npu_img_p = &image_s[npu_img_idx];
npu_img_p->raw_img_p = raw_img_p;
```

Since postprocessing is done on NCPU while CNN model is processed by NPU, these two processing could be parallelized for better performance.

To achieve this, two sets of data structures and variables are used alternatively for npu (and pre-processing) and for post-processing.

```
struct kdp_image image_s[IPC_IMAGE_ACTIVE_MAX];
int npu_img_idx;
struct kdp_image *npu_img_p;
struct kdp_image *pp_img_p;
```

and the index is toggled for next input image:

```
npu_img_idx = !npu_img_idx;
```

## 6.4 CPU and Post Processing

When NCPU receives a notification from NPU, NPU may be completely done with model running or a special CPU operation is needed. In any ways, we should call cpu-op API:

```
rc = kdpio_run_cpu_op(npu_img_p);
```

With no error return code *RET_NO_ERROR* from cpu-op, this post-processing can be started with optional secondary output buffer to move the original output data out for parallel processing:

```
out_comm_p->img_index_done = i;
out_comm_p->img_results[i].status = IMAGE_STATE_NPU_DONE;
POSTPROC_OUTPUT_MEM_ADDR(npu_img_p) = in_comm_p->output_mem_addr2;
pp_img_p = npu_img_p;

rc = kdpio_run_postprocess(pp_img_p, npu_out_data_move);
```

# 7 Adding new model

Adding new model for Kneron chip will need

1) Kneron SDK Compiler to generate or convert the model (ONNX or other supported types) to Kneron chip recognizable files.
2) Adding the model’s pre-/post-/cpu-op processing in NCPU firmware
    - Optionally, these processing could be done on Host side (like PC) side with NPU output data being transferred to Host.

Adding a new model’s pre-/post-/cpu-op processing needs to call registration API. Call the unregistration API to remove pre-registered callback.

## 7.1 Model types

Kneron supported model types are already defined by *enum model_type* in model_type.h. New model types can be added with higher enums, such as 1000.

```
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,
};
```

## 7.2 Image format

Following image formats are supported by KL520 and SDK.

```
#define NPU_FORMAT_RGBA8888        0x00
#define NPU_FORMAT_YUV422          0x10
#define NPU_FORMAT_NIR             0x20
#define NPU_FORMAT_YCBCR422        0x30
#define NPU_FORMAT_YUV444          0x40
#define NPU_FORMAT_YCBCR444        0x50
#define NPU_FORMAT_RGB565          0x60
```

This one byte format value should be passed in ‘format’ field of IPC raw image struct to KDPIO-Lib in NCPU. Corresponding model’s preprocessing and postprocessing can use to retrieve the format value: *RAW_FORMAT(image_p)*.

## 7.3 Preprocess example

```
static int kdp_preproc_fd(int model_id, struct kdp_image *image_p)
{
}

kdpio_pre_processing_register(KNERON_FDSMALLBOX, kdp_preproc_fd);
```

## 7.4 Postprocess example

```
static int post_face_detection(int model_id, struct kdp_image *image_p)
{
}

kdpio_post_processing_register(KNERON_FDSMALLBOX, post_face_detection);
```

## 7.5 cpu-op example

```
static int nearest_upsample_cpu(int cpu_op, struct kdp_image *image_p)
{
}

kdpio_cpu_op_register(NEAREST_UPSAMPLE, nearest_upsample_cpu);
```

## 7.6 Helper Macros

Use the helper macros in kdpio.h to get the parameters of raw input image, model dimension, preprocess data, cpu-op data, postprocess and its output node data.

```
#define RAW_IMAGE_MEM_ADDR(image_p) (image_p->raw_img_p->image_mem_addr)
#define RAW_IMAGE_MEM_LEN(image_p)  (image_p->raw_img_p->image_mem_len)
#define RAW_FORMAT(image_p)         (image_p->raw_img_p->format)
#define RAW_INPUT_ROW(image_p)      (image_p->raw_img_p->input_row)
#define RAW_INPUT_COL(image_p)      (image_p->raw_img_p->input_col)
#define RAW_CROP_TOP(image_p)       (image_p->raw_img_p->params_s.crop_top)
#define RAW_CROP_BOTTOM(image_p)    (image_p->raw_img_p->params_s.crop_bottom)
#define RAW_CROP_LEFT(image_p)      (image_p->raw_img_p->params_s.crop_left)
#define RAW_CROP_RIGHT(image_p)     (image_p->raw_img_p->params_s.crop_right)
#define RAW_PAD_TOP(image_p)        (image_p->raw_img_p->params_s.pad_top)
#define RAW_PAD_BOTTOM(image_p)     (image_p->raw_img_p->params_s.pad_bottom)
#define RAW_PAD_LEFT(image_p)       (image_p->raw_img_p->params_s.pad_left)
#define RAW_PAD_RIGHT(image_p)      (image_p->raw_img_p->params_s.pad_right)
#define RAW_OTHER_PARAMS(image_p)   (image_p->raw_img_p->params_s.params)

#define DIM_INPUT_ROW(image_p)      (image_p->dim.input_row)
#define DIM_INPUT_COL(image_p)      (image_p->dim.input_col)
#define DIM_INPUT_CH(image_p)       (image_p->dim.input_channel)

#define PREPROC_INPUT_MEM_ADDR(image_p)     (image_p->preproc.input_mem_addr)
#define PREPROC_INPUT_MEM_LEN(image_p)      (image_p->preproc.input_mem_len)
#define PREPROC_INPUT_MEM_ADDR2(image_p)    (image_p->preproc.input_mem_addr2)
#define PREPROC_INPUT_MEM_LEN2(image_p)     (image_p->preproc.input_mem_len2)
#define PREPROC_INPUT_RADIX(image_p)        (image_p->preproc.input_radix)
#define PREPROC_PARAMS_P(image_p)           (image_p->preproc.params_p)

#define POSTPROC_OUTPUT_FORMAT(image_p)     (image_p->postproc.output_format)
#define POSTPROC_OUTPUT_MEM_ADDR(image_p)   (image_p->postproc.output_mem_addr)
#define POSTPROC_OUTPUT_MEM_LEN(image_p)    (image_p->postproc.output_mem_len)
#define POSTPROC_RESULT_MEM_ADDR(image_p)   (image_p->postproc.result_mem_addr)
#define POSTPROC_RESULT_MEM_LEN(image_p)    (image_p->postproc.result_mem_len)
#define POSTPROC_PARAMS_P(image_p)          (image_p->postproc.params_p)

#define POSTPROC_OUT_NODE(image_p)          (image_p->postproc.node_p)
#define POSTPROC_OUT_NODE_COL(image_p)      (image_p->postproc.node_p->col_length)
#define POSTPROC_OUT_NODE_ROW(image_p)      (image_p->postproc.node_p->row_length)
```

```
#define POSTPROC_OUT_NODE_CH(image_p)       (image_p->postproc.node_p->ch_length)
#define POSTPROC_OUT_NODE_RADIX(image_p)    (image_p->postproc.node_p->output_radix)
#define POSTPROC_OUT_NODE_SCALE(image_p)    (image_p->postproc.node_p->output_scale)

#define CPU_OP_NODE(image_p)            (image_p->cpu_op.node_p)
#define CPU_OP_NODE_OP_TYPE(image_p)    (image_p->cpu_op.node_p->op_type)

#define MODEL_P(image_p)                (image_p->model_p)
#define MODEL_ID(image_p)               (image_p->model_id)
#define MODEL_SETUP_MEM_P(image_p)      (image_p->setup_mem_p)
#define MODEL_CMD_MEM_ADDR(image_p)     (MODEL_P(image_p)->cmd_mem_addr)
#define MODEL_CMD_MEM_LEN(image_p)      (MODEL_P(image_p)->cmd_mem_len)
#define MODEL_WEIGHT_MEM_ADDR(image_p)  (MODEL_P(image_p)->weight_mem_addr)
#define MODEL_BUF_ADDR(image_p)         (MODEL_P(image_p)->buf_addr)
#define MODEL_SETUP_MEM_ADDR(image_p)   (MODEL_P(image_p)->setup_mem_addr)
```