Kneron Inc

Document Name: **Kneron Application Library**

Kneron Application Library

Kneron Inc

Engineering Design Document

Table of Contents

1 Introduction 2
1.1 Purpose 2
1.2 Scope 2
2 Reference 2
3 Acronyms, Abbreviations, Definitions 2
4 API Architecture 2
5 Kneron Applications (kdp_app_xxx) 3
5.1 Face identification and recognition function set (kdp_app_fid.h) 3
5.1.1 data structure and error return code 4
5.1.2 provided APIs 5
5.1.3 Example code 6
5.2 Data base operation APIs (kdp_app_db.h) 7
5.2.1 data structure and error return code 7
5.2.2 Provided APIs 7
5.2.3 Example code 11
5.3 Light Weight 3D Face Identification and Recognition (kdp_app_lw3d.h) 12
5.3.1 data structure and error return codes 12
5.3.2 Provided APIs 13
5.3.3 Example code 15
5.4 Dynamic Model Execution (kdp_app_dme.h) 16
5.4.1 data structure and error return codes 16
5.4.2 Provided APIs 16
5.4.3 Example code in DME mode 17
6 Kneron Application Generic (kdp_app.h) 18
6.1 Image Configuration 19
6.1.1 Data Structures 19
6.1.1 Provided APIs 19
6.2 Model Related Configuration 20
6.2.1 Provided APIs 20
6.3 DME Mode 21
6.3.1 data structure 21
6.3.2 Provided APIs 21
6.3.3 Example code 22

1 Introduction

1.1 Purpose

The purpose of this document is to define the high-level APIs that will provide the functionality of face identification, face recognition and limited on-board database management, and object detection.

1.2 Scope

The protocol defined in this document shall covers the functionality and usage of APIs for object detection and classification in DME mode, face identification, digital signature extraction by Kneron algorithm and database management. However, this document does not apply to the dedicated SPI interface used for chip testing or FLASH access purposes.

2 Reference

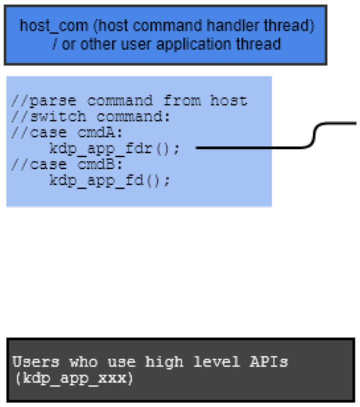
Kneron KL520 Design Specification, Rev. 0.5, (Internal), Feb. 2019
Host and Companion Communication Protocol, draft release, Kneron, Feb. 20, 2019
Host Interface Message Protocol, Rev. 0.27, Kneron, Jul. 24, 2019

3 Acronyms, Abbreviations, Definitions

FID – Face Identification
FR – Face Recognition
OD – Object Detection
DME – Dynamic Model Execution

4 API Architecture

All applications are presented in different APIs. APIs should be called under an osThreadId.
Only kdp_app_xxx is covered in this document


host_com (host command handler thread)
/ or other user application thread
Users who use high level APIs
(kdp_app_xxx)

```
//parse command from host
//switch command:
//case cmdA:
    kdp_app_fdr();
//case cmdB:
    kdp_app_fd();
```

5 Kneron Applications (kdp_app_xxx)

Kneron Applications are provided in a library for specific function set licensed by Kneron. Eg. Face recognition function set with data base, object detection function set, etc.

Available application set are:
- Face Identification and Recognition (kdp_app_fid.h)
- Associate APIs works with Face Identification and Recognition (kdp_app_db.h)
- Light Weight 3D Face Identification and Recognition (kdp_app_lw3d.h)
- Dynamic Model Execution (kdp_app_dme.h)

!!! kdp_app_xxx must be called in a thread.

5.1 Face identification and recognition function set (kdp_app_fid.h)

This function set provides Face Identification and Recognition capacity.
Specification:
- VGA resolution image (640x480)
- RGB565
- YCbCr422(YUV422)
- RAW8

Database Configuration: Refer to 5.2

Limitation:
kdp_app_fid.h and *kdp_app_lw3d.h* cannot work together

5.1.1 data structure and error return code

```
//Error Code Enum:
enum kdp_app_fid_status_code_e {
   KDP_APP_FDR_WRONG_USAGE = 0x100,            ///== KDP_APP_FID_CODES
   KDP_APP_FDR_NO_FACE,                        /// error: no face
   KDP_APP_FDR_BAD_POSE,                       /// error: base face pose
   KDP_APP_FDR_NO_ALIVE,                       /// error: not alive
   KDP_APP_FDR_FR_FAIL,                        /// error: FR failed
   KDP_APP_LM_LOW_CONFIDENT,                   /// error: LM low confident
   KDP_APP_LM_BLUR,                            /// error: LM blur
   KDP_APP_SMALL_FACE                          /// error: face too small
};

enum kdp_app_fid_op_e{
   KDP_APP_FID_OP_EXTRACT,              /// extract feature map only
   KDP_APP_FID_OP_INFERENCE,            /// do inference (do DB comparison)
   KDP_APP_FID_OP_REGISTER              /// do register (insert DB temp data)
};

// data structure
//--------------------------------
typedef struct kdp_app_fid_inout_s {
   enum kdp_app_fid_op_e op;            ///operation mode
   uint16_t face_id;                    ///1-5: faceId (1-5)
   uint16_t user_id;                    ///as input: user id got from Host
                                        ///as output: user id returned by KL520
   float thresh_fid;                     ///threshold value for face recognition
} kdp_app_fid_inout_t;

/** @brief face detectin result
 *
 * facedet_result_s is defined in ipc.h
 * */
typedef struct facedet_result_s kdp_app_face_det_res_t;

/** @brief landmark result
 *
 * landmark_result_s is defined in ipc.h
 * */
typedef struct landmark_result_s kdp_app_face_lm_res_t;

/** @brief face recognition result
 *
 * fr_result_s defined in ipc.h
 * */
typedef struct fr_result_s kdp_app_face_fmap_t;

typedef struct kdp_app_image_buffer_desc_s {
  uint32_t buf_count;                   ///image buffer count
  uint32_t alignment_in_bit;            ///alignment size in bit
  uint32_t rgb_buf_base_addr;           ///rgb image buffer address
  uint32_t rgb_img_size;                          ///rgb image size
  uint32_t nir_buf_base_addr;                     ///nir image buffer address
  uint32_t nir_img_size;                          ///nir image size
} kdp_app_image_buffer_desc_t;
```

5.1.2 provided APIs

```
Int32_t kdp_app_init(void)
```

This function initializes necessary data structure.

Parameters

None

Returns:

registered people count in built-in database

```
int32_t kdp_app_fd(
            kdp_app_face_det_res_t* p_out_p/*out*/)
```

The function performs face detection

Parameters

[out]kdp_app_face_det_res_t* p_out_p;   // face detection result

Returns:

KDP_APP_OK
KDP_APP_UNKNOWN_ERR
KDP_APP_FDR_NO_FACE

```
int32_t kdp_app_fdr(
            kdp_app_fid_inout_t *p_inout_p,
            kdp_app_face_det_res_t *p_fd_out_p,
            kdp_app_face_lm_rest_t* p_lm_out_p,
            kdp_app_face_fmap_t *p_fr_out_p);
```

This function performs both face authentication and registration according to different mode, inference mode and register mode

Parameters

[in] kdp_app_fid_inout_t *p_inout_p	configuration. *Ref: 5.1.1*
[out] kdp_app_face_det_res_t *p_fd_out_p	Face detection result bounding box
[out] kdp_app_face_lm_res_t* p_lm_out_p	Face landmark results
[out] kdp_app_face_fmap_t *p_fr_out_p	Face feature map

Returns:

KDP_APP_OK
KDP_APP_UNKNOWN_ERR

KDP_APP_FDR_WRONG_USAGE
KDP_APP_FDR_NO_FACE
KDP_APP_FDR_BAD_POSE
KDP_APP_FDR_FR_FAIL

KDP_APP_DB_NO_SPACE
KDP_APP_DB_ALREADY_SAVED
KDP_APP_DB_NO_MATCH

5.1.3 Example code

```
#include "kdp_app_fid.h"
#include "kdp_app,h"
#include “cmsis_os2.h”

static void caller_thread(void *argument);

int main(void)
{
   ...
   SystemCoreClockUpdate();                              // System Initialization
   osKernelInitialize();                                 // Initialize CMSIS-RTOS

   ...
   osThreadId_t tid_caller = osThreadNew(caller_thread, NULL, NULL);
   //tid_caller = init_caller_thread();
}

void caller_thread(void *argument)
{
   ...
   Int ret;
   kdp_app_init();                                       // init data

   kdp_app_fid_inout_t  app_data;
   app_data.mode = 0;                                    // inference mode
   app_data.thresh_fid = 0.4;

   kdp_app_face_det_res_t fd_out;
   kdp_app_face_lm_res_t lm_out;
   kdp_app_face_fmap_t  fr_out;

   uint8_t* image = capture_image();                     // get image
   uint32_t image_size = get_image_size();               // get image size
   memcpy((void*)(KDP_DDR_BASE_IMAGE_BUF),  image, image_size);

   ret = kdp_app_fdr(app_data, &fd_out, &lm_out, &fr_out);      // do inference
   ret = kdp_app_fd(&fd_out);                            // do face detection
```

```
  if(ret  == KDP_APP_OK) {
     ...
  }
  else {
     ...
  }
```

5.2 Data base operation APIs (kdp_app_db.h)

On chip data base operations are provided

Configuration:

```
#define KDP_APP_DB_FLASH_ADDR   0x80000            // the physical flash address of built in database
#define MAX_USER       20                          // the max USER count in database
#define MAX_FID        5                           // the max face count for each user
#define DB_NUM         2                           // DB count(for 1 camera or 2 camera)
#define KDP_APP_DB_DEFAULT_GUARD_THRESHOLD 0.425 // the default threshold value for FR
                                                 //value range: [0-1] means [strict – loose]
```

5.2.1 data structure and error return code

```
enum kdp_app_db_status_code_e {
   KDP_APP_DB_NO_SPACE = 0x200,
   KDP_APP_DB_ALREADY_SAVED,                 // identity is already saved
   KDP_APP_DB_DEL_NOT_VALID,                 // wrong delete setting
   KDP_APP_DB_NO_MATCH,                      // no found
   KDP_APP_DB_REG_FIRST,                     // no register data before adding to DB
   KDP_APP_DB_USER_NOT_REG,                  // queried user is not regitered
   KDP_APP_DB_DEL_FAIL                       // delete operation failed
};

typedef struct kdp_app_db_user_data_s {
   uint16_t user_id_in;          // user ID
   uint16_t user_id_out;         // useless
   uint16_t user_idx;            // user index: 0 – (MAX_USER - 1)
   uint16_t fm_idx;              // face index: 0 – (MAX_FID - 1)
   uint16_t del_all;             // if delete all database
} kdp_app_db_user_data_t;
```

5.2.2 Provided APIs

```
int32_t kdp_app_db_init(uint32_t flash_db_addr_p)
```

To initialize data base from existed data in flash

Parameters

[in] input   flash address of database

Return

Valid user count in database

```
void kdp_app_db_switch(uint32_t db_index)
```

To switch database by database index

Parameters

[in] db_index database index, [0 | 1]

Return

N/A

```
//    uint16_t user_id_in;
//    uint16_t user_id_out;  //not use
//    uint16_t user_idx;       //not use
//    uint16_t fm_idx;          //not use
//    uint16_t del_all;           //not use

int32_t kdp_app_db_add(kdp_app_db_user_data_t * input)
```

This function is used to add saved feature maps in ddr to flash

Parameters

[in] input pointer to an instance of "kdp_app_db_user_data_t"

Return

KDP_APP_DB_NO_SPACE,
KDP_APP_DB_ALREADY_SAVED,
KDP_APP_DB_REG_FIRST,

```
//    uint16_t user_id_in;
//    uint16_t user_id_out;   //not use
//    uint16_t user_idx;        // not use
//    uint16_t fm_idx;          // not use
//    uint16_t del_all;

int32_t kdp_app_db_delete(kdp_app_db_user_data_t *input)
```

This function is used to delete specific user id record or clear entire database

Parameters

[in] input pointer to an instance of "kdp_app_db_user_data_t"

Return

KDP_APP_OK
KDP_APP_DB_DEL_NOT_VALID,

KDP_APP_DB_USER_NOT_REG,
KDP_APP_DB_DEL_FAIL

```
int32_t kdp_app_db abort_reg(void)
```

This function is used to abort registration flow

Parameters

NA

Return

KDP_APP_OK

```
int32_t kdp_app_db_compare(
      uint32_t fdr_addr, uint16_t *user_id/*output*/,
      float thresh_guard);
```

compare user fd/fr result to user data in ddr

Parameters

[in] fdr_addr address of fd/fr feature map data
[out] *user_id recognized user ID
[in] thresh_guard comparision threshold for recognition

Return

compare result and user id where find match feature map data

```
int32_t kdp_app_db_lw3d_compare(
      uint32_t rgb_fdr_addr, uint32_t nir_fdr_addr,
      uint16_t *user_id, float thresh_guard);
```

compare user fd/fr result to user data in ddr

Parameters

[in] rgb_fdr_addr address of fd/fr rgb feature map data
[in] nir_fdr_addr address of fd/fr nir feature map data
[out] *user_id recognized user ID
[in] thresh_guard is the threshold

Return

compare result and user id where find match feature map data

```
int32_t kdp_app_db_register(uint32_t fdr_addr, uint16_t user_id, uint16_t fm_idx);
```

register user fm data

Parameters

[in] fdr_addr address of fd/fr feature map data
[in] user_id use id
[in] fm_idx face index

Return

Status. Refer to 5.2.1

```
int32_t kdp_app_db_slot_is_used(uint16_t i);
```

check if a db slot is used

Parameters

[in] i slot index

Return

1: yes 0:no

```
int kdp_app_db_get_available_ID_slot(void);
```

get the next available ID slot for register

Return

slot id for register

```
int kdp_app_db_get_user_id_slot_type(int user_idx);
```

get slot type for a user index

Parameters

[in] user_idx user index

Return

type, 0:invalid, 1:valid, 2:register

```
int kdp_app_db_get_user_id(int user_idx);
```

get user ID for a user index

Parameters

[in] user_idx user index

Return

user ID (could be different from user index)

```
float kdp_app_db_cal_similarity( uint32_t fdr_addr_1, uint32_t fdr_addr_2);
```

Calculate similarity of two feature points

Parameters

[in] fdr_addr address A of user feature map data
[in] fdr_addr address B of user feature map data

Return

similarity score (smaller means more like)

5.2.3 Example code

```
#include "kdp_app_fid.h"
#include “kdp_app_db.h”
#include "kdp_app,h"
#include “cmsis_os2.h”
static void caller_thread(void *argument);

int main(void)
{
   ...
   SystemCoreClockUpdate();          // System Initialization
   osKernelInitialize();             // Initialize CMSIS-RTOS

   ...
   osThreadId_t tid_caller = osThreadNew(caller_thread, NULL, NULL);
   //tid_caller = init_caller_thread();
}

void caller_thread(void *argument)
{
   ...
   Int ret;
   kdp_app_init();                                        // init data
                                                          // kdp_app_db_init() is called inside

   kdp_app_config_image(0, 640, 480, 3,                  //config image
                     KDP_APP_IMAGE_FORMAT_RGB56,
                     KDP_DDR_BASE_IMAGE_BUF);

   kdp_app_fid_inout_t  app_data;
   app_data.mode=1;                                       //register mode, 1st face
   app_data.user_id = 123;                                //register for ID= 123
   app_data.thresh_fid = 0.8;
```

```
kdp_app_face_det_res_t fd_out;
kdp_app_face_lm_res_t lm_out;
kdp_app_face_fmap_t  fr_out;

uint8_t* image = capture_image();                          // to get image
uint32_t image_size = get_image_size();                   // get image size
memcpy((void*)(KDP_DDR_BASE_IMAGE_BUF),  image, image_size);

ret = kdp_app_fdr(app_data, &fd_out, &lm_out, &fr_out);      // do inferenc

 kdp_app_db_user_data_t reg_data;
 app_data.user_id_in = 123;

ret= kdp_app_db_add(&reg_data);
//or
ret= kdp_app_db_del(&reg_data);
// or
ret= kdp_app_db_abort_reg();


if(ret  == KDP_APP_OK) {
   ...
}
else {
   ...
}
```

5.3 Light Weight 3D Face Identification and Recognition (kdp_app_lw3d.h)

Light weight 3D provides higher accuracy face recognition than 2D solution.

Specification:
- VGA resolution image (640x480) RGB565/YCbCr422(YUV422)   : RGB camera
- VGA resolution image (640x480) RAW8   : NIR camera

Database Configuration: Refer to 5.2

Limitation:
kdp_app_fid.h and kdp_app_lw3d.h cannot work together

5.3.1 data structure and error return codes

```
enum kdp_app_lw3d_status_code_e {
   KDP_APP_LW3D_WRONG_USAGE = 0x100,               //KDP_APP_LW3D_CODES
   KDP_APP_LW3D_NO_FACE,                          // no face is detected
  KDP_APP_LW3D_FACE_BOUNDARY_FAIL,               // face region is out of boundary
   KDP_APP_LW3D_BAD_POSE,                         //bad pose
   KDP_APP_LW3D_NO_ALIVE,                                    //not alive
   KDP_APP_LW3D_FR_FAIL,                                     //FR failed
};

typedef struct kdp_app_lw3d_inout_s{
   uint16_t mode;                          //0: inference   1-5: register with faceId(1-5)
   uint16_t user_id;                       //as input: user id got from Host
                                           //as output: user id returned by KL520
   float rgb_thresh_fid;                   //threshold value for face recognition
   float nir_thresh_fid;                   //threshold value for face recognition

} kdp_app_lw3d_inout_t;

typedef struct facedet_result_s kdp_app_lw3d_face_det_res_t;     //fd result
typedef struct landmark_result_s kdp_app_lw3d_face_lm_res_t;  //lm result
typedef struct fr_result_s kdp_app_lw3d_face_fmap_t;                //fr result
```

5.3.2 Provided APIs

```
void kdp_app_lw3d_init(void)
```

This function initializes necessary data structure.

Parameters

None

Returns:

None

```
int32_t kdp_app_lw3d(
            kdp_app_lw3d_inout_t              *p_inout_p,
            kdp_app_lw3d_face_det_res_t  *p_fd_out_p,
            kdp_app_lw3d_face_lm_ras_t   *p_lm_out_p,
            kdp_app_lw3d_face_fmap_t     *p_fr_out_p,
            kdp_app_lw3d_face_det_res_t *p_nir_fd_out_p,
            kdp_spp_lw3d_face_lm_res_t   *p_nir_lm_out_p,
            kdp_app_lw3d_face_fmap_t     *p_nir_fr_out_p
);
```

This function performs both face authentication and registration according to different mode, inference mode and register mode

Parameters

[in] *p_inout_p Configuration
[out] *p_fd_out_p Face detection result
[out] *p_lm_out_p Face landmark result
[out] *p_fr_out_p Face feature map
[out] *p_nir_fd_out_p Face detection result(NIR)
[out] *p_nir_lm_out_p Face landmark result(NIR)
[out] *p_nir_fr_out_p Face feature map(NIR)

Returns:

KDP_APP_OK
KDP_APP_UNKNOWN_ERR

KDP_APP_LW3D_WRONG_USAGE
KDP_APP_LW3D_NO_FACE,
KDP_APP_LW3D_FACE_BOUNDARY_FAIL
KDP_APP_LW3D_BAD_POSE
KDP_APP_LW3D_NO_ALIVE
KDP_APP_LW3D_FR_FAIL,

KDP_APP_DB_NO_SPACE
KDP_APP_DB_ALREADY_SAVED
KDP_APP_DB_NO_MATCH

```
int32_t kdp_app_nir_fdr_only(
            kdp_app_fid_inout_t     *p_inout_p,
            kdp_app_face_det_res_t  *p_fd_out_p,
            kdp_app_face_lm_res_t   *p_lm_out_p,
            kdp_app_face_fmap_t     *p_fr_out_p);
```

get feature map of cam1(nir) only

Parameters

[inout] *p_inout_p image information and settings
[out] *p_fd_out_p FD result
[out] *p_lm_out_p LM result
[out] *p_fr_out_p FR result

Return

KDP_APP_OK
KDP_APP_UNKNOWN_ERR

KDP_APP_LW3D_WRONG_USAGE
KDP_APP_LW3D_NO_FACE,
KDP_APP_LW3D_FACE_BOUNDARY_FAIL
KDP_APP_LW3D_BAD_POSE
KDP_APP_LW3D_NO_ALIVE
KDP_APP_LW3D_FR_FAIL,

KDP_APP_DB_NO_SPACE
KDP_APP_DB_ALREADY_SAVED
KDP_APP_DB_NO_MATCH

5.3.3 Example code

```
#include "kdp_app_lw3d.h"
#include "kdp_app,h"
#include “cmsis_os2.h”

static void caller_thread(void *argument);

int main(void)
{
   ...
   SystemCoreClockUpdate();                         // System Initialization
   osKernelInitialize();                            // Initialize CMSIS-RTOS

   ...
   osThreadId_t tid_caller = osThreadNew(caller_thread, NULL, NULL);
   //tid_caller = init_caller_thread();
}

void caller_thread(void *argument)
{
   ...
   Int ret;
   Int offset = 640*480*3;
   kdp_app_lw3d_init();                             // init data
   kdp_app_config_image(0, 640, 480, 3,             //config image for RGB camera
                     KDP_APP_IMAGE_FORMAT_RGB565,
                     KDP_DDR_BASE_IMAGE_BUF);

   kdp_app_config_image(1, 640, 480, 1,             //config image for NIR camera
                     KDP_APP_IMAGE_FORMAT_RAW8,
                     KDP_DDR_BASE_IMAGE_BUF+offset);


   kdp_app_lw3d_inout_t  app_data;
   app_data.mode = 0;
   app_data.rgb_thresh_fid = 0.8;
   app_data.nir_thresh_fid = 0.75;

   kdp_app_lw3d_face_det_res_t fd_out, nir_fd_out;
   kdp_app_lw3d_face_lm_res_t lm_out, nir_lm_out;
   kdp_app_lw3d_face_fmap_t  fr_out, nir_fr_out;

   //prepare RGB image
   uint8_t* image = capture_image();                // get image
   uint32_t image_size = get_image_size();          // get image size
   memcpy((void*)(KDP_DDR_BASE_IMAGE_BUF),  image, image_size);

   //prepare NIR image
   uint8_t* image = capture_image();                // get image
   uint32_t image_size = get_image_size();          // get image size
   memcpy((void*)(KDP_DDR_BASE_IMAGE_BUF + 3 * KDP_IMAGE_BUF_SIZE),  image, image_size);
```

```
ret = kdp_app_lw3d_fdr(app_data, &fd_out, &lm_out, &fr_out,
                                    &nir_fd_out, &nir_lm_out, &nir_fr_out);    // do inference

if(ret  == KDP_APP_OK) {
   ...
}
else {
   ...
}
```

5.4 Dynamic Model Execution (kdp_app_dme.h)

Dynamic model execution provides the features to get detection output or raw output with images with format of RGB565, RGBA8888, etc.

Specification:

Binary image (224x224x4 for Tiny YOLO V3) RGBA8888   : RGBA binary data

5.4.1 data structure and error return codes

```
typedef struct kdp_app_bounding_box_s {
   float x1;          // top-left corner:  x
   float y1;          // top-left corner:  y
   float x2;          // bottom-right corner:  x
   float y2;          // bottom-right corner:  y

   float score;       // probability score
   int class_num;     // class # (of many) with highest probability
} kdp_app_bounding_box_t;

#define OBJECT_DETECTION_MAX    80

typedef struct kdp_app_dme_res_s {
   uint32_t    class_count;          // total class count
   uint32_t    box_count;            /* boxes of all classes */
   kdp_app_bounding_box_t boxes[OBJECT_DETECTION_MAX];      /* [box_count] */
} kdp_app_dme_res_t;
```

5.4.2 Provided APIs

```
int32_t kdp_app_dme_init(void);
```

This function initializes necessary data for dynamic model execution (reserved for future usage).

Parameters

None

Returns:

KDP_APP_OK

```
int32_t kdp_app_dme(kdp_app_dme_res_t* p_out_p, struct kdp_dme_conf_s img_conf);
```

This function performs dynamic model execution (DME) to get detection output or raw output.

Parameters

[out]	kdp_app_dme_res_t	*p_out_p	Detection result, or raw output results
[in]	struct kdp_dme_conf_s	img_conf	Image configuration

Returns:

KDP_APP_OK
KDP_APP_UNKNOWN_ERR

5.4.3 Example code in DME mode

The example code shows how to dynamically load model, configuration, and return detection results or raw output results.

Detection Results:

Confirm to the structure of struct kdp_app_dme_res_t

Raw Output Results:

OUTPUT_NUM (4 bytes) +
OUTPUT_NUM * HCW ([height, channel, width, h2, c2, w2, ...]) +
RAW_DATA (RAW1 (float), RAW2, ...)

```
#include "kdp_app_dme.h"
#include "kdp_app.h"
#include “cmsis_os2.h”

#define KDP_DDR_MEDEL_INFO_TEMP 0X63FFC000

static void caller_thread(void *argument);

int main(void)
{
   ...
   SystemCoreClockUpdate();                          // System Initialization
   osKernelInitialize();                             // Initialize CMSIS-RTOS

   ...
   osThreadId_t tid_caller = osThreadNew(caller_thread, NULL, NULL);
   ...
}

void caller_thread(void *argument)
{
   ...
   int ret;

   kdp_app_dme_res_t       *p_od_out_s = 0;
   uint32_t p_raw_out_s = 0;
   struct kdp_dme_conf_s dme_conf;

   // receive firmware info from host and load it
   memcpy((uint32_t*)KDP_DDR_MODEL_INFO_TEMP, &(msgcmd_arg->len), msghdr->mSize - 4);

   // receive model from host and load it
   ret = host_mem_read(cmd_start_addr, model_size);

   // enable DME mode and reset model settings
   kdp_app_dme_mode(1);

   // receive configuration from host and load it
   memcpy(&dme_conf, &(msgcmd_arg->addr), sizeof(struct kdp_dme_conf_s));

   if (!(dme_conf.img_conf.image_format & IMAGE_FORMAT_BYPASS_PRE)) {
       // write RGB565 image into DDR
       ret = host_mem_read(KDP_DDR_BASE_IMAGE_BUF, msgcmd_arg->addr);  // specify how many bytes
   }  else {
       // write RGBA image into DDR
      ret = host_mem_read(DDR_MEM_BASE, msgcmd_arg->addr);  // specify how many bytes
   }

   if (dme_conf.img_conf.image_format & IMAGE_FORMAT_RAW_OUTPUT) {
      ret = kdp_app_dme((kdp_app_dme_res_t *)p_raw_out_s, dme_conf);
   } else {
      ret = kdp_app_dme(p_od_out_s, dme_conf);
   }   // do inference

   if (dme_conf.img_conf.image_format & IMAGE_FORMAT_RAW_OUTPUT) {
      data_write((u8 *) p_raw_out_s, final_len);
   } else {
      data_write((u8 *) p_od_out_s, final_len);
   }   // send results to host
}
```

6 Kneron Application Generic (kdp_app.h)

Generic data structures and functions are provided in kdp_app.h

6.1 Image Configuration

Image size, format, source address settings

6.1.1 Data Structures

```
#define KDP_APP_IMAGE_FORMAT_RGBA8888   0x80000000   /**< RGBA8888 */
#define KDP_APP_IMAGE_FORMAT_RAW8       0x80000020   /**< RAW8 */
#define KDP_APP_IMAGE_FORMAT_YCBCR422   0x80000037   /**< YCBCR422 */
#define KDP_APP_IMAGE_FORMAT_RGB565     0x80000060   /**< RGB565 */

struct kdp_img_conf_s (
    int32_t col,                             /**< image column size */
    int32_t row,                             /**< image row size */
     int32_t ch,                             /**< image channel size */
   uint32_t image_format,                    /**< image format, eg. KDP_APP_IMAGE_FORMAT_XXXX */
    uint32_t image_memory_address);          /**< image source memory address */
};
```

6.1.1 Provided APIs

```
void kdp_app_config_image(
    int cam_index,        /**< camera sensor index to config*/
    int32_t col,          /**< image column size */
    int32_t row,          /**< image row size */
     int32_t ch,                             /**< image channel size */
   uint32_t image_format,                    /**< image format, eg. KDP_APP_IMAGE_FORMAT_XXXX */
    uint32_t image_memory_address);          /**< image source memory address */
)
```

This function is called to specify image configuration

Parameters

[in] cam_index	cam0=0 or cam1=1
[in] col	column size
[in] row	row size
[in] ch	channel size
[in] image_format	propietory image format
[in] image_memory_address	image DDR address

Returns:

N/A

```
void kdp_app_config_image_memory_address(
    int cam_index,                                /**< camera sensor index to config*/
    int32_t image_memory_address);                /**< image source memory address */
)
```

This function is called to change image source(frame buffer)

Parameters

[in] cam_index cam0 or cam1
[in] image_memory_address image DDR address

Returns:

N/A

```
uint32_t kdp_app_config_get_channel_size(
   uint32_t  format                                          /**< image format*/
)
```

This function is called to get image channel size by image format

Parameters

[in] format see definition, eg. KDP_APP_IMAGE_FORMAT_XXXX

Returns:

Channel size

```
uint32_t kdp_app_config_get_pixel_size(
   uint32_t  format                                          /**< image format*/
)
```

This function is called to get image pixel size by image format

Parameters

[in] format see definition, eg. KDP_APP_IMAGE_FORMAT_XXXX

Returns:

pixel size

6.2 Model Related Configuration

APIs to load model, config user defined model, and run model

6.2.1 Provided APIs

```
void kdp_app_load_model(void);
```

load model from flash

```
void kdp_app_config_model_input_output(
       uint32_t model_type_p,
    struct kdp_img_conf_s* p_img_config_p,
    void* p_output_p);
```

Config model/input/output information before run model

Parameters

[in] model_type (Defined in common/include/model_type.h)

[in] kdp_img_config_s for image format information and memory address

[in] output memory address

```
int32_t kdp_app_run_model(void);
```

To run configured model.
Must be called after kdp_app_config_model_input_output();

Return

Status(controlled by user defined post process functions)

6.3 DME Mode

Dynamic model load is supported. Models could be uploaded externally.

6.3.1 data structure

```
/* KDP image configuration structure */
struct kdp_img_conf_s {
   int32_t     image_col;
   int32_t     image_row;
   int32_t     image_ch;
   uint32_t    image_format;
};

/* KDP dme configuration structure */
struct kdp_dme_conf_s {
  int32_t     models_selection;
   int32_t     output_num;
   struct kdp_img_conf_s img_conf;
};
```

6.3.2 Provided APIs

```
void kdp_app_dme_mode(int is_dme_mode);
```

This function is called to specify or change DME mode.

Default: non-DME mode

Parameters

[in] is_dme_mode : is DME mode or not

Returns:

N/A

6.3.3 Example code

Refer to 5.4.3