Kneron Inc

Document Name: **Host Library Software Design**

Host Library Software design

Kneron Inc

Engineering Design Document

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the software design of host library, which is used by Applications running on the host controller to communicate with the Kneron devices.

1.2 Scope

This document covers the software design for the host library. In the current design, the command messages are exchanged by UART or USB interface and the image files are transferred from the host to Kneron devices via USB interface. If the message transferring interface has to be changed, the host library has to be changed accordingly.

The library can support two modes:
1. Command message in UART + image file in USB;
2. Both Command message and image file in USB.
If using UART mode, UART device should be added to the library by calling API kdp_add_dev(int type, const char* name). If using USB mode, USB device (indicated by device type passed to the API parameter) should be added to the library.

Although the host library support multiple devices, mix usage of UART device and USB device is not supported till now. It means that you can add multiple USB devices or add multiple UART devices. But one UART device and one USB device could not be added to the library for one same Application.

2 Reference

Host Interface Message Protocol, Rev. 0.28, Kneron D-0002.

3 Acronyms, Abbreviations, Definitions

FD	Face Detection
FR	Face Recognition
FDR	Face Detection and Recognition
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
Host (controller)	A PC or Server running FD or FR Applications
Kneron device	Hardware device with NPU

4 System Architecture

The system is composed of host controller and Kneron devices. Variants of Applications could be run on the host controller, such as FD, FR, FDR, object recognition, etc. Each Application has a source of pictures either from a live video camera or from a photo album and it need send the pictures to Kneron device to identify or recognize the contents of those pictures.

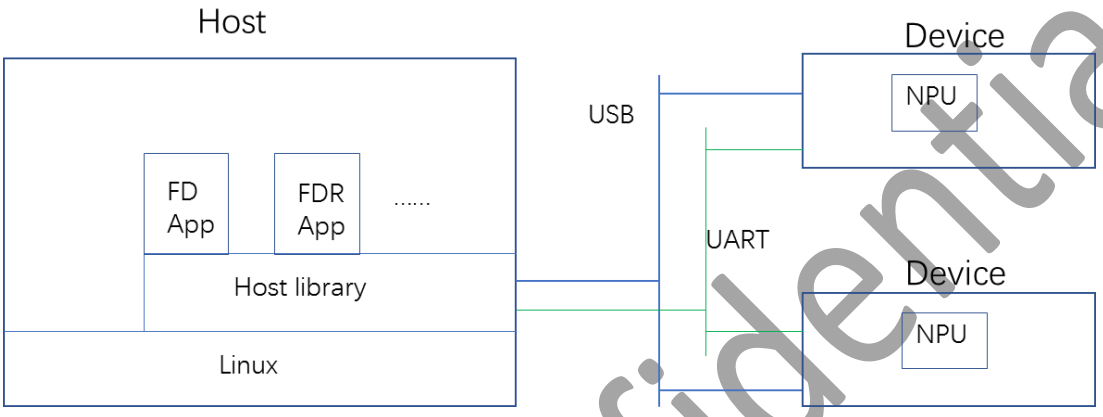

Figure 1: System Architecture

Kneron devices generally contain NPU, which is used to process the images transferred from the host. According to the running Application configuration, the appropriate model needs to be loaded to the device first, and then the NPU will calculate the output result with the inputted image by the loaded model.

For example, if running the FDR App, first the face rectangle is calculated by the FD model on the NPU, and then the face features are extracted from the face image by the FR model. And at last, the face features will be compared with the local database in the device to detect whether this face is matched with any one in the database. The matched result will be sent back to host to return to the end user.

There are two interfaces connected to each device from the host: UART and USB. Currently the command messages are exchanged via UART interface and the image file is transferred to device by USB interface in running mode 1. And both command messages and image file are transferred in USB in running mode 2. The supported command messages are described on the “Host Interface Message Protocol”.

5 Software Design

The host library software provides API services to the Applications, which does not need to care about the hardware and OS details. The host library is built with Linux system call and libusb 3rd party open source library. If running on Windows or other OS, the appropriate system calls need to be changed accordingly.

To provide such services, the host library is decomposed into the following 5 sub modules: interface sub module, message receiver, message sender, message handler and device handler. The details of each sub module are described as follows.

Thinking of the host could connect multiple devices and the input / output by UART interface is too slow, two modes of message communication between host and device are provided: sync mode and async mode.

In sync mode communication, the API need wait for the running result from device. The main thread can not do other API call before this API returns. To avoid deadlock, the API returns timeout in case of it could not get the response from device for a specified time slot.

In async mode communication, the API returns immediately. The running result is returned to the calling Application in registered callback functions. The host library checks for each pending command call from the Application, in case of it could not get the response for a command call for a specified time slot, it will return timeout to the Application too. In the current implementation, the framework for async mode communication is done, but the code needs to be modified according to the real user scenario.

5.1 Interface Sub Module

This sub module provides interface APIs to the Applications. It abstracts the functionalities provided by the host library and provides one interface API for each specific functionality from user point of view. It separates the specific Application requirements from the actual backend message exchanging between host and devices.

The header file “kdp_host.h” in this sub module is exported to the end user, and its implementations are provided in library format. The real implementation is designed in C++ concepts, but the header file is provided in C format, which allows both the C developers and C++ developers.

The C++ header file will be provided later.

5.1.1 Header file

Header file “kdp_host.h” is exported to the end user. The Application developers need to include this header file to their project and link to the host library in the build script.

By now, there are 4 parts in this file: library init, SFID APIs, system operation APIs and dynamic mode execution APIs.

Before calling the APIs in the host library, appropriate initialization is required: log init, library init, library start and device adding. For the detailed description of each API, refer to section 7.

The SFID is the first Application supported by host library. For this Application, there are two modes: face verifying mode and face registration mode. In face registration mode, the device extracts face features from the input image and save them to local database.

In face verifying mode, the device extracts face features from the input image and compare them to the local database. Then it returns the matched user ID to the host.

The end user need call kdp_start_verify_mode() to enter face verifying mode and call kdp_start_reg_user_mode() to enter face registration mode.

The system operation APIs contain utilities for the operation of device such as reset of the device, status query for the device, firmware update and model update for the device.

The dynamic model execution APIs provide utilities for the user to specify the model and transfer it to the device and related setup data. And then the device does inference using the given model and configuration data and send the inference result back.

For the detailed description of each API, refer to the section 7.

5.1.2 Implementation file in sync mode

File “KdpHostApi.cpp” contains the real implementation of each API working in sync mode. It forwards the API call from end user to the appropriate function in “SyncMsgHandler.cpp”.

The function in “SyncMsgHandler.cpp” generates command message according to the end user input value and send it to device. Then it need wait for the response message from device.

If the response message arrives in a specified time slot, it decodes the result from the response message and return it to end user. Otherwise, it returns timeout to the end user.

5.1.3 Implementation file in async mode

File “KdpHostApi_async.cpp” and file “AsyncMsgHandler.cpp” contain the framework for APIs working in async mode.

In case of working in async mode, the API returns immediately after sending the messages to the device. When the end user call all the required APIs, a background handler thread needs to be run to handle the result messages returned from the device.

For each API call, a callback function needs to be provided, which will be called in case of the response message arrives. Generally, the callback function needs to process the result message.

Since currently there is no Application for this mode, it needs to be modified according to the real Application scenario in the future.

5.2 Message Receiver

Since the receiving of messages in UART interface is too slow, a separate thread is created to do the message receiving. It monitors the file descriptors registered by all the devices by “select” system call. In case of one message is coming from any file descriptor, this thread will “read” all the message content and put it in the message queue.

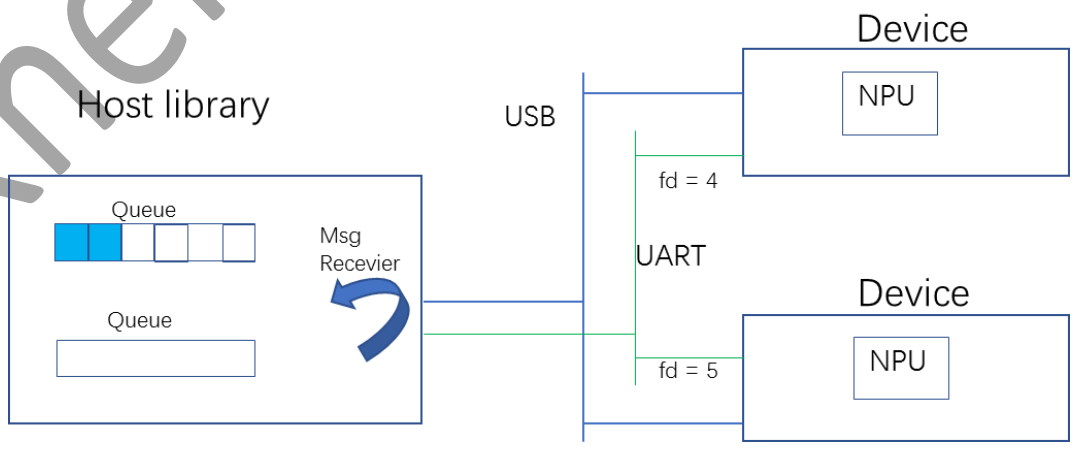

Figure 2: Message Receiver

Since the UART message comes slowly, there are possibly two kinds of exceptions: message receiver returns a part of message, message receiver returns one message and a part of next message. To avoid the message chaos, the receiver read the message header (4 bytes) firstly, and then it decodes the message length stored in the header. At last, it read exactly the specified number of bytes from the file descriptor.

In case of the queue is full, it will throw the received messages. The message handler periodically got the messages from the queue and process them. Refer to the section 5.4 for detail.

In USB only mode, currently libusb_bulk_transfer is used for receiving messages from device, which costs too much CPU resources, so it is not applicable to make it running all the time. Only when command messages are initiated from host, this thread will be in running state. It periodically calls libusb_bulk_transfer to wait for response messages from device. If not in command message processing state, this thread will be blocked.

5.3 Message Sender

Since the sending of message via UART interface is slow, in async mode, a separate thread is created to do the message sending. It monitors the message sending queues periodically (for each device, there is a separate message sending queue), in case of one UART message is available in the queue, this thread will send it via UART interface to the specific device. The calling thread returns after pushing the messages to the queue.

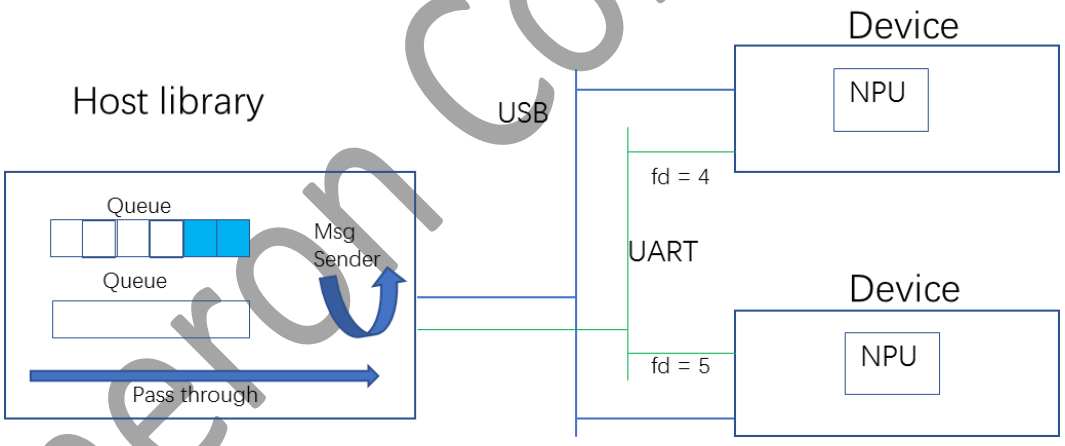

Figure 3: Message Sender

In sync mode, the message sending thread is not used. The calling thread sends the UART messages directly without using the message sending queue. The speed is not an issue, since it need wait for the result response anyway.

In USB only mode, this thread is not used either since only sync message transfer is used.

5.4 Message Handler

The message handler is the main controller for the host library. It holds references to message receiver and message sender. It also holds references to sync message handler and async message handler.

For message sending, it provides sending APIs to the interface sub module for both sync mode and async mode. For message receiving, it creates a separate thread which monitors the message receiving queues periodically. In case of any message is received, it decodes the message and dispatch it to sync message handler or async message handler accordingly.

The sync message handler generally wakes up the waiting user thread and notify it about the arriving of its response message. The async message handler generally wakes up the waiting user thread and calls the registered callback functions.

5.5 Device Handler

The device handler is responsible for the creating of specific devices and configure their parameters. Currently, only UART device and USB device are supported.

When creating UART devices, its name has to be provided correctly. Generally, for USB cable UART in Linux, the name is "/dev/ttyUSB0". And the baud rate and parity need to be configured too (no parity, 1 stop bit, 8 bits).

In UART mode, the USB device acts as a helper device for a created UART device. In the "send SFID image" command handling, the host send the "CMD_SFID_SEND_IMAGE" command via UART to the device first, then it wait for the ACK from the device. After receiving the ACK, it calls the USB interface to send the image to the device.

The sending of image on USB interface relies on the 3rd party libusb library. The version used in the host library is libubs 1.0. API libusb_bulk_transfer() in libusb is called to send the image to device from the host.

If all command messages and image files have to be transferred in USB device, USB device can not act as a helper device. The UART device is not supported in this mode. The USB device sends and receives all the messages and do decoding and encoding. All the messages are exchanged via libusb API libusb_bulk_transfer().

6 Host Library APIs

This section summarizes the list of APIs that provided by host library. The functionality and usage for each API are described in detail including the input and output parameters.

6.1 Library Init API

- kdp_lib_init():

Description:
initialize the host library

Return value:
return 0 on succeed, -1 on failure

- kdp_lib_start():

Description:
start the host library to wait for messages

Return value:
return 0 on succeed, -1 on failure

- kdp_lib_de_init():

Description:
free the resources used by host library

Return value:
return 0 on succeed, -1 on failure

- kdp_init_log(const char* dir, const char* name):

Description:
initialize the host lib internal log

Parameters:
dir: the directory name of the log file
name: the log file name

Return value:
return 0 on succeed, -1 on failure

- kdp_add_dev(int type, const char* name):

Description:
add com device to the host lib

Parameters:
type: the device type, only KDP_USB_DEV supported now
name: Currently, only “” is required

Return value:
return dev index on succeed, -1 on failure

6.2 System operation API

- kdp_reset_sys(int dev_idx, uint32_t reset_mode);

Description:

request for doing system reset, for details, please refer to the document 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.3.1.

Parameters:
dev_idx: connected device ID. A host can connect several devices
reset_mode: specifies the reset mode
0 - no operation
1 - reset message protocol
3 - switch to suspend mode
4 - switch to active mode
255 - reset whole system
256 - system shutdown (RTC)

0x2000xxxx -reset/switch active boot partition, please refer 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.3.1

Reset mode 3 and 4 are not supported for KL520 USB mode.

Return value:
0 on succeed, else for error code ,please refer to《KL520 Host Interface Message Protocol v1_5.pdf》 section6.3.1

- kdp_report_sys_status(int dev_idx, uint32_t* device, uint32_t* firmware_id, uint16_t* sys_status, uint16_t* app_status);

Description:
request for system status

Parameters:
dev_idx: connected device ID. A host can connect several devices

device: the name of the device,current hard code is “0x0520”(KL520)

firmware_id: the id of the firmware in device
Firmware id is the KL520 firmware release version number. It is encoded as major, minor, patch & edit number. The most recent number should be 0.9.4.2 but will be continuously updated as engineering development proceeds.

sys_status: system status

System status refers to the power management status of KL520. Only bit 0 is currently defined. If bit 0 is set, it means KL520 is placed in a low power inactive state. This state can be invoked by issuing a RESET command with reset mode control = 3, (enter suspended mode).

However, this suspended mode is only valid if the host interface is UART(reserved) since the USB interface will be disabled in this low-power mode.

For the USB-based host, the only low power mode is the system shut down (reset mode control = 256) command

app_status: application status

Application status refers to the application that is currently active and ready to run. The status returned corresponds to the application id of the active application. Currently, applications defined are 0 = NO APPLICATION ADDED, 1 = LW3D, 2 = SFID and 3 = DME.

Applications are activated by sending the Start.LW3D FID, or Start SFID, or Start DME command, respectively

Return value:
0 on succeed, else –1 on failure

- kdp_update_fw(int dev_idx, uint32_t* module_id, char* img_buf, int buf_len);

Description:
request for update firmware

Parameters:
dev_idx: connected device ID. A host can connect several devices
module_id: the module id of which the firmware to be updated
1 - scpu module
2 - ncpu module
img_buf, buf_len: the fw image buffer and file size

Return value:
0 if succeed, else error code， the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.3.3

- int kdp_update_model(int dev_idx, uint32_t* model_id, uint32_t model_size, \
  char* img_buf, int buf_len);

Description:
request for update model

Parameters:
dev_idx: connected device ID. A host can connect several devices
model_id (reserved): the model id to be updated
model_size: the size of the model
img_buf: the model file buffer
buf_len: the size of model

Return value:
0 if succeed, else error code, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.3.4

6.3 SFID API

In this section, two suites of APIs are described: kdp_verify_user_id* and kdp_extract_feature*. They are used to analyze the images host sent to device.

When in user verification mode, host send an image to KL520 to check whether there is a user id which has the same face feature as the given image. If yes, KL520 returns user id to host. In this case kdp_verify_user_id* should be used.

When in user registration mode, host asks the KL520 to start to register user id i image index j. And then host send this user's face image to KL520, which extracts the face feature and save it to flash DB. In this case, kdp_extract_feature* should be used.

And for each suite of API, two kinds of APIs are provided: *all_res and *generic.
*all_res API need user give all the required parameters. The API parses the fd result, fr result, etc from response message (from KL520) and copy them to the appropriate return parameters. Users can use the returned fd result, fr result, etc directly.

For *generic API, user only need provide mask and a buffer. User need to parse the fd result, fr result, landmark result, liveness result from the buffer according to the mask value by himself/herself.

- Generic data structure for fd result/ fr result/ landmark result/ liveness result

fd result: 8 bytes. It stands for (x, y, width, height), and unsigned short(uint16_t) for each value
fr result: 2048 bytes. It stands for 512 feature map values, and float type for each value
landmark result: 20 bytes. It stands for five coordinates(x,y), and unsigned short(uint16_t) for each value. The first coordinate (x,y) is for left eye, and the following are for right eye, for nose, for left mouth corner, and for right mouth corner.
Liveness result: 2 bytes. It stands for alive or not(reserved).

The following sample code is to get value,
e.g.
To get "x" and "y" from "char *fd_res"

```
coor_x = ((fd_res[1] & 0xff) << 8 )| (fd_res[0] & 0xff);
coor_y = ((fd_res[3] & 0xff) << 8 )| (fd_res[2] & 0xff);
```

- kdp_start_verify_mode(int dev_idx, uint32_t* img_size):

Description:
start the user verification mode and use default threshold, 0.4

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_size: the required image file size will be returned.

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.1

- kdp_start_verify_mode_thresh(int dev_idx, uint32_t* img_size, float thresh):

Description:

start the user verification mode with specified threshold
It is same as calling kdp_start_sfid_mode with image width 640, image height 480, image format RGB565

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_size: the required image file size will be returned.
thresh: the threshold used to match the face recognition result. Range: 0.0-1.0

Return value:
return 0 on succeed, error code on failure, he details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.1

- kdp_start_sfid_mode(int dev_idx, uint32_t* img_size, float thresh, uint16_t width, uint16_t height, uint32_t format):

Description:
start the user sfid mode with specified threshold, image format

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_size: the required image file size will be returned.
thresh: the threshold used to match the face recognition result. Range: 0.0-1.0f
img_width: the width of input image
img_height: the height of input image
format: the input image format: IMG_FORMAT_RGBA8888, IMG_FORMAT_RAW8, IMG_FORMAT_YCbCr422, IMG_FORMAT_RGB565

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.1

- kdp_verify_user_id(int dev_idx, uint16_t* user_id, char* img_buf, int buf_len):

Description:
extract the face feature from input image, and compare it with DB and return the matched user ID.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the input image memory and length
user_id: matched user ID in DB will be returned.

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.5

- kdp_verify_user_id_fdfr_res(int dev_idx, uint16_t* user_id, char* img_buf, int buf_len, \
  bool* fd_flag, char* fd_res, bool* fr_flag, char* fr_res):

Description:
extract the face feature from input image, and compare it with DB and return the matched user ID and face detection result, face recognition result.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the input image memory and length
user_id: matched user ID in DB will be returned.
fd_flag: whether fd result is requested and whether it is available
fr_flag: whether fr result is requested and whether it is available
fd_res: fd result, vaild only set fd_flag true
fr_res: fr_result, vaild only set fr_flag true

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.5

- kdp_verify_user_id_all_res(int dev_idx, uint16_t* user_id, char* img_buf, int buf_len, \
  bool* fd_flag, char* fd_res, bool* fr_flag, char* fr_res, \
  bool* lm_flag, char* lm_res, bool* live_flag, char* live_res);

Description:
extract the face feature from input image, and compare it with DB,
return the matched user ID, and face analysis related result.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the image buffer and file size
user_id: matched user ID in DB will be returned.
fd_flag: indicate whether fd result is requested and whether fd result is available.
fd_res: contains fd result
fr_flag: indicate whether fr result is requested and whether fr result is available.
fr_res: contains fr result
lm_flag: indicate whether lm result is requested and whether lm result is available
lm_res: landmark result 20 bytes:
live_flag (reserved): indicate whether liveness result is requested and whether liveness result is available
live_res (reserved): liveness detection result

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《 KL520 Host Interface Message Protocol v1_5.pdf 》 section6.5.5

- kdp_verify_user_id_generic(int dev_idx, uint16_t* user_id, char* img_buf, int buf_len, \
  uint16_t* mask, char* res);

Description:
extract the face feature from input image, and compare it with DB,
return the matched user ID, and face analysis related result.

Parameters:
dev_idx: connected device ID. A host can connect several devices
user_id: matched user ID in DB will be returned.
img_buf, buf_len: the image buffer and file size
mask: indicate the requested and responsed flags
  bit 0 - FD result
  bit 1 - LM data
  bit 2 - FM feature map
  bit 3 – liveness(reserved)
res: contains all the related result
 FD result: 8 byte
 LM result: 20 bytes
 FR result: 2048 bytes
 liveness: two uint16_t value(reserved)

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《 KL520 Host Interface Message Protocol v1_5.pdf 》 section6.5.5

- kdp_start_reg_user_mode(int dev_idx, uint16_t usr_id, uint16_t img_idx):

Description:
start the user register mode

Parameters:
dev_idx: connected device ID. A host can connect several devices
user_id: the user id that will be registered,user id range 1~20
img_idx: the image index that will be saved. (a user could have 5 images),img_idx range 1~5

Return value:

return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.2

- kdp_extract_feature(int dev_idx, char* img_buf, int buf_len):

Description:
extract face feature from input image and save it in device
before calling this API, host must call kdp_start_reg_user_mode to enable register mode,
and specified for which user and which index to be extracted.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_byf, buf_len: the input image memory and length

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.5

- kdp_extract_feature_fdfr_res(int dev_idx, char* img_buf, int buf_len, \
  bool* fd_flag, char* fd_res, bool* fr_flag, char* fr_res);:

Description:
extract face feature from input image and save it in device. Returns face detection and face recognition result.
before calling this API, host must call kdp_start_reg_user_mode to enable register mode,
and specified for which user and which index to be extracted.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the input image memory and length
fd_flag: whether fd result is requested and whether it is available
fr_flag: whether fr result is requested and whether it is available
fd_res: fd result,vaild when fd_flag is true.
fr_res: fr_result, vaild when fr_flag is true

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.5

- kdp_extract_feature_all_res(int dev_idx, char* img_buf, int buf_len, \
  bool* fd_flag, char* fd_res, bool* fr_flag, char* fr_res, \
  bool* lm_flag, char* lm_res, bool* live_flag, char* live_res);

Description:
extract face feature from input image and save it in device,
returns face detection, face recognition result.
before calling this API, host must call kdp_start_reg_user_mode to enable register mode,
and specified for which user and which index to be extracted.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the image buffer and file size
fd_flag: indicate whether fd result is requested and whether fd result is available.
fd_res: contains fd result, 8 bytes
fr_flag: indicate whether fr result is requested and whether fr result is available.
fr_res: contains fr result, 2048 bytes
lm_flag: indicate whether lm result is requested and whether lm result is available
lm_res: landmark result 20 bytes:
live_flag: indicate whether liveness result is requested and whether liveness result is available
live_res: liveness detection result

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《 KL520 Host Interface Message Protocol v1_5.pdf 》 section6.5.5

- kdp_extract_feature_generic(int dev_idx, char* img_buf, int buf_len, \
  uint16_t* mask, char* res);

Description:
extract face feature from input image and save it in device,
returns face detection, face recognition result.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the image buffer and file size
mask: indicate the requested and response flags
  bit 0 - FD result
  bit 1 - LM data
  bit 2 - FM feature map
  bit 3 - liveness
res: contains all the related result
  FD result: 8 bytes
  LM result: 20 bytes
  FR result: 2048 bytes
  liveness: two uint16_t value

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.5

- kdp_register_user(int dev_idx, uint32_t user_id):

Description:
register the face features to device DB
before calling this API, host must have called kdp_extract_feature
at least once successfully, otherwise, no features could be saved.

Parameters:
dev_idx: connected device ID. A host can connect several devices
user_id: the user id that be registered. must be same as the kdp_start_reg_user_mode.Range:1~20

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.3

- kdp_remove_user(int dev_idx, uint32_t user_id):

Description:
remove users from device DB, if user_id is 0, remove all registered users
It needs to be called after start lw3d mode or start verify mode

Parameters:
dev_idx: connected device ID. A host can connect several devices
user_id: the user to be removed. 0 for all users,Range:1~20

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.4

- kdp_start_lw3d_mode(int dev_idx, uint32_t* rgb_size, uint32_t* nir_size,
  float rgb_thresh, float nir_thresh, uint16_t rgb_width, uint16_t rgb_height,
  uint16_t nir_width, uint16_t nir_height, uint32_t rgb_fmt, uint32_t nir_fmt);

Description:
start the light weight 3D mode with specified threshold, img format

Parameters:
dev_idx: connected device ID. A host can connect several devices
rgb_size: the required rgb image file size will be returned.
nir_size: the required nir image file size will be returned.

rgb_thresh: the threshold used to match rgb face recognition result.
if 0, the default threshold is used.
nir_thresh: the threshold used to match nir face recognition result.
if 0, the default threshold is used.
rgb_width: the width of rgb image
rgb_height: the height of rgb image
nir_width: the width of nir image
nir_height: the height of nir image
rgb_fmt: the format of rgb image
nir_fmt: the format of nir image

Return value:
return 0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.6

- kdp_start_lw3d_mode_thresh(int dev_idx, uint32_t* rgb_size, uint32_t* nir_size, float rgb_thresh, float nir_thresh);

Description:
start the light weight 3D mode with specified threshold
It is same as calling kdp_start_lw3d_mode with image width 640, image height 480, RGB image format RGB565, NIR image format RAW8.

Parameters:
dev_idx: connected device ID. A host can connect several devices
rgb_size: the required rgb image file size will be returned.
nir_size: the required nir image file size will be returned.
rgb_thresh: the threshold used to match rgb face recognition result.
if 0, the default threshold, 0.4, is used. Range: 0.0-1.0
nir_thresh: the threshold used to match nir face recognition result.
if 0, the default threshold,0.4, is used. Range: 0.0-1.0

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.6

- kdp_extract_lw3D_feature(int dev_idx, char* img_buf, int buf_len, \ char* img_buf_nir, int buf_len_nir);

Description:
extract the face feature from input rgb/nir image, and save it in device,
return the face analysis related result if required.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the rgb image buffer and file size
img_buf_nir, buf_len_nir: the nir image buffer and file size

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.7

- kdp_extract_lw3D_feature_all_res(int dev_idx, char* img_buf, int buf_len, \
  char* img_buf_nir, int buf_len_nir, bool* fd_flag, char* fd_res, \
  bool* fr_flag, char* fr_res, bool* lm_flag, char* lm_res, \
  bool* fd_flag_nir, char* fd_res_nir, bool* fr_flag_nir, char* fr_res_nir, \
  bool* lm_flag_nir, char* lm_res_nir, bool* live_flag, char* live_res);

Description:
extract the face feature from input rgb/nir image, and save it in device,
return the face analysis related result if required.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the rgb image buffer and file size
img_buf_nir, buf_len_nir: the nir image buffer and file size
fd_flag/fd_flag_nir: indicate whether fd result is requested and whether fd result is available.
fd_res/fd_res_nir: contains fd result, 8 bytes
fr_flag/fr_flag_nir: indicate whether fr result is requested and whether fr result is available.
fr_res/fr_res_nir: contains fr result, 2048 bytes
lm_flag/lm_flag_nir: indicates whether lankmwark result is requested and whether result is available.
lm_res/lm_res_nir: contains landmark result, 20 bytes
live_flag: indicates whether liveness detection is requested and whether result is available
live_res: contains liveness detection result, two uint16_t

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.7

- kdp_extract_lw3D_feature_generic(int dev_idx, char* img_buf, int buf_len, \
  char* img_buf_nir, int buf_len_nir, uint16_t* mask, char* res);

Description:
extract the face feature from input rgb/nir image, and save it in device,
return the face analysis related result if required.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the rgb image buffer and file size
img_buf_nir, buf_len_nir: the nir image buffer and file size

mask: indicate the requested and response flags
  bit 0 - FD result

bit 1 - LM data
bit 2 - FM feature map
bit 4 - NIR FD result
bit 5 - NIR LM data
bit 6 - NIR feature map
bit 8 - Final liveness
res: contains all the related result
FD result: 8 bytes
LM result: 20 bytes
FR result: 2048 bytes
NIR FD result: 8 bytes
NIR LM result: 20 bytes
NIR FR result: 2048 bytes
liveness: two uint16_t

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.7

- kdp_verify_lw3D_user(int dev_idx, uint16_t* user_id, char* img_buf, int buf_len, \
  char* img_buf_nir, int buf_len_nir);

Description:
extract the face feature from input rgb/nir image, and compare it with DB,
return the matched user ID, and face analysis related result.

Parameters:
dev_idx: connected device ID. A host can connect several devices
user_id: matched user ID in DB will be returned.
img_buf, buf_len: the rgb image buffer and file size
img_buf_nir, buf_len_nir: the nir image buffer and file size

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.5.7

- kdp_verify_lw3D_image_all_res(int dev_idx, uint16_t* user_id, char* img_buf, int buf_len,
  char* img_buf_nir, int buf_len_nir, bool* fd_flag, char* fd_res, \
  bool* fr_flag, char* fr_res, bool* lm_flag, char* lm_res, \
  bool* fd_flag_nir, char* fd_res_nir, bool* fr_flag_nir, char* fr_res_nir, \
  bool* lm_flag_nir, char* lm_res_nir, bool* live_flag, char* live_res);

Description:
extract the face feature from input rgb/nir image, and compare it with DB,
return the matched user ID, and face analysis related result.

Parameters:
dev_idx: connected device ID. A host can connect several devices
user_id: matched user ID in DB will be returned.
img_buf, buf_len: the rgb image buffer and file size
img_buf_nir, buf_len_nir: the nir image buffer and file size
fd_flag/fd_flag_nir: indicate whether fd result is requested and whether fd result is available.
fd_res/fd_res_nir: contains fd result, 8 bytes
fr_flag/fr_flag_nir: indicate whether fr result is requested and whether fr result is available.
fr_res/fr_res_nir: contains fr result, 2048 bytes
lm_flag/lm_flag_nir: indicates whether lankmwark result is requested and whether result is available.
lm_res/lm_res_nir: contains landmark result, 20 bytes
live_flag: indicates whether liveness detection is requested and whether result is available
live_res: contains liveness detection result, two uint16_t

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《 KL520 Host Interface Message Protocol v1_5.pdf 》 section6.5.7

- kdp_verify_lw3D_image_generic(int dev_idx, uint16_t* user_id, char* img_buf, int buf_len, char* img_buf_nir, int buf_len_nir, uint16_t* mask, char* res);

Description:
extract the face feature from input rgb/nir image, and compare it with DB,
return the matched user ID, and face analysis related result.

Parameters:
dev_idx: connected device ID. A host can connect several devices
user_id: matched user ID in DB will be returned.
img_buf, buf_len: the rgb image buffer and file size
img_buf_nir, buf_len_nir: the nir image buffer and file size

mask: indicate the requested and response flags
  bit 0 - FD result
  bit 1 - LM data
  bit 2 - FM feature map
  bit 4 - NIR FD result
  bit 5 - NIR LM data
  bit 6 - NIR feature map
  bit 8 - Final liveness
res: contains all the related result
  FD result: 8 bytes
  LM result: 20 bytes
  FR result: 2048 bytes
  NIR FD result: 8 bytes
  NIR LM result: 20 bytes
  NIR FR result: 2048 bytes

liveness: two uint16_t

Refer to sub-section: Generic data structure for fd result/ fr result/ landmark result/ liveness result of #6.3 for details

Return value:
0 on succeed, error code on failure, the details please refer to 《 KL520 Host Interface Message Protocol v1_5.pdf 》 section6.5.7

6.4 Dynamic model API

- kdp_start_dme(int dev_idx, uint32_t model_size, char* data, int dat_size, \
  uint32_t* ret_size, char* img_buf, int buf_len);

Description:
request for starting dynamic model execution

Parameters:
dev_idx: connected device ID. A host can connect several devices
model_size: size of inference model
data: firmware setup data
dat_size: setup data size
ret_size: returned model size
img_buf, buf_len: the model file buffer and file size

Return value:
0 on succeed, error code on failure, the details please refer to 《 KL520 Host Interface Message Protocol v1_5.pdf 》 section6.6.1

- kdp_dme_configure(int dev_idx, char* data, int dat_size);

Description:
request for configuring dme
It is same as calling kdp_dme_configure_param with setup_conf = 1.

Parameters:
dev_idx: connected device ID. A host can connect several devices
data: inference setup data
dat_size: the size of setup data

Return value:
0 on succeed, error code on failure, the details please refer to 《 KL520 Host Interface Message Protocol v1_5.pdf 》 section6.6.2

- kdp_dme_configure_param (int dev_idx, uint32_t setup_conf, char* data, int dat_size);

Description:

request for configuring dme

Parameters:
dev_idx: connected device ID. A host can connect several devices
setup_conf: specifies how the setup data to be used.
data: inference setup data
dat_size: the size of setup data

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.6.2

- kdp_dme_inference(int dev_idx, char* img_buf, int buf_len, uint32_t* inf_size, \
  bool* res_flag, char* inf_res);

Description:
calling device to do inference with provided model
before calling this API, host must call kdp_start_dme and kdp_dme_configure to configure the dme model.

Parameters:
dev_idx: connected device ID. A host can connect several devices
img_buf, buf_len: the image buffer and file size
inf_size: the size of inference result
res_flag: indicate whether result is requested and available
inf_res: contains the returned inference result

Return value:
0 on succeed, error code on failure, the details please refer to 《KL520 Host Interface Message Protocol v1_5.pdf》 section6.6.3

6.5 Feature map comparison API

- float kdp_FM_comparison(unsigned char *user_fm_a, unsigned char *user_fm_b, int fm_size);

Description:
Calculate similarity of two feature points .
before calling this API, host must ensure buffer A and B are the same size

Parameters:
user_fm_a: buffer A of user feature map data
user_fm_b: buffer B of user feature map data
fm_size: size of user feature map data

Return value:
-1 on failure, similarity score on succeed(smaller score are more similar).

7 Unit testing

This section describes the methods for the unit testing of host library.

If there is no KDP520 board available, a simulator (Linux PC) could be used to testing the UART message transferring, receiving, decoding and encoding.

For an example, the RK3399 Dev board could be used as a simulator. It has a UART port, which could be connected to the UART port in the Host PC (also running Linux). Refer to the source file uart_test.cpp in directory tester for detail.

If connected to a KDP520 EVB board, the demo programs in directory example could be used to test the APIs listed in section 7. The library initialization and de-initialization are called only once, while the other APIs could be called many times sequentially.

In main.cpp, the definition MAX_TIMES specifies how many times the list of APIs will be run in the unit test. It could be set to value 1000,000 for the aging test. The CPU performance and memory usage status need to be monitored by Linux tool “top”.

There are several demo programs: deluser, reguser, veruser, test, l3wd, dme, udt_fw, udt_md.
Program deluser is used for remove a specific user from device database. If 0, remove all users.
Program reguser is used to register the uid (given by argument) to database.
Program veruser is used to extract the face feature and match it in the device database.
Program test is used for all the APIs contained in this library.
Program lw3d is used for the testing of light weight 3D functionalities.
Program dme is used for the testing of dynamic model execution functionalities.
Program udt_fw is used for the testing of firmware update functionalities.
Program udt_md is used for the testing of update model functionalities.
Program fmcmp is used for the testing of feature map comparison.

For the detailed usage of those demo programs, please check the following sections.

7.1 Program deluser

Program deluser can be used to remove a specific user or all users from the KL520 flash database. By default, it removes the user(s) for SFID mode. If it is required to remove user in LW3D mode, the LW3D related source code in file user_test_del.cpp needs to be enabled by change “#if 0” to “#if 1”. (The SFID mode code needs to be disabled accordingly)

Usage: deluser user_id
For example:
To remove all users
sudo ./deluser
sudo ./deluser 0

To remove a specific user
sudo ./deluser 1

APIs used:
kdp_remove_user()

7.2 Program reguser

Program reguser can be used to register a user to KL520 flash database, the image used for the face registration is "../test_image/img/ u1_f1_rgb.bin".

Usage: reguser user_id
For example:
To register user id 2
sudo ./reguser 2

APIs used:
kdp_start_reg_user_mode (), kdp_extract_feature_fdfr_res(), kdp_extract_feature(), kdp_register_user()

7.3 Program veruser

Program veruser can be used to search a user from KL520 flash database by his/her face image, the image used for the face verification is "../test_image/img/ u1_f1_rgb.bin".

Usage: veruser
For example:
To search the user
sudo ./veruser

APIs used:
kdp_verify_user_id (), kdp_verify_user_id_fdfr_res ()

7.4 Program test

Program test contains a suite of test cases for user removal, user registration, user verification. The image used for the face registration/verification is "../test_image/img/ u1_f1_rgb.bin".

It also contains test cases for system reset. Since the reset mode 255, 256 and firmware bank switch need operator manual intervention, for long run purpose, those case are commented out. If need those kind of test cases, the related code needs to be enabled. (by changing "#if 0" to "#if 1")

Usage: test user_id
For example:
To test the removal, registration, verification of user id 2:
sudo ./test 2

APIs used:
kdp_remove_user(), kdp_verify_user_id(), kdp_verify_user_id_fdfr_res(), kdp_verify_user_id_all_res(), kdp_verify_user_id_generic(), kdp_start_reg_user_mode(), kdp_extract_feature(), kdp_extract_feature_all_res(), kdp_extract_feature_generic(), kdp_register_user(), kdp_reset_sys(), kdp_report_sys_status()

7.5 Program lw3d

Program lw3d contains lw3d test cases for user removal, user registration, user verification. The image used for the face registration/verification is "../test_image/img/u18_f1_rgb.bin and u18_f1_nir.bin".

Usage: lw3d user_id
For example:
To test the removal, registration, verification of user id 1:
sudo ./lw3d 1

APIs used:
kdp_remove_user(), kdp_extract_lw3D_feature_all_res(), kdp_extract_lw3D_feature_generic(), kdp_extract_lw3D_feature(), kdp_verify_lw3D_image_all_res(), kdp_verify_lw3D_image_generic(), kdp_verify_lw3D_user()

7.6 Program dme

Program dme can be used to test the dynamic mode execution feature. The dme related test files are in directory "../test_image/dme/". For dme, there are two modes: raw mode and detect mode.

Usage: dme mode
For example:
To run dme raw mode
sudo ./dme 2

To run dme detect mode
sudo ./dme 1

APIs used:
kdp_start_dme(), kdp_dme_configure(), kdp_dme_inference()

7.7 Program udt_md

Program udt_md can be used to test the update model feature. The model file is "../test_image/ota/ model_ota.bin".

Note:
model_ota.bin can be generated as following (windows only):
kl520_sdk\utils\ota\gen_ota_binary_for_win.exe -model fw_info.bin all_models.bin model_ota.bin

model_ota.bin can be generated as following (linux version):
chmod +x kl520_sdk\utils\ota\gen_ota_binary_for_linux
kl520_sdk\utils\ota\gen_ota_binary_for_linux -model fw_info.bin all_models.bin model_ota.bin

Usage: udt_md model_id (0 – no operation, other value – model id)

For example:
To run update model with model id 1
sudo ./udt_md 1

To run update model with empty operation
sudo ./udt_md 0

Note：After update model finishes, the KL520 is doing reset. The KL520 needs to be re-started manually, either by SPI or by JTAG. After the system is started successfully, KL520 can send back the response.

APIs used:
kdp_update_model()

7.8 Program udt_fw

Program udt_fw can be used to test the update firmware feature. The firmware files are in "../test_image/ota/work1". An alternate directory "work2" exists in the same directory, which could be used for testing of the switch of two banks.

Usage: udt_fw fw_id (0 – no operation, 1 – scpu, 2 - ncpu)

For example:
To run update scpu firmware
sudo ./udt_fw 1

To run update ncpu firmware
sudo ./udt_fw 2

Note：After firmware update finishes, the KL520 is doing reset. The KL520 needs to be re-started manually, either by SPI or by JTAG. After the system is started successfully, KL520 can send back the response.

APIs used:
kdp_update_fw()

7.9 Program fmcmp

Program fmcmp can be used to calculate similarity of two feature points, the image used for the face verification is "../test_image/img/ u1_f1_rgb.bin", "../test_image/img/ u18_f1_rgb.bin"..

Usage: fmcmp
For example:

To search the user
sudo ./fmcmp

APIs used:
kdp_start_verify_mode_thresh (), kdp_start_reg_user_mode
(),kdp_remove_user(),kdp_extract_feature_all_res(),kdp_FM_comparison().

8 Example code

This section lists the example code, which could be used as the code base for real Applications. It also demonstrates the usages for each API listed in section 7.

Refer to the following code segment for the usage of library initialization and de-initialization:

8.1 UART device example

Line 23 is to initialization the debug log to specified directory /tmp/mzt.log. If other file name or directory is preferred, it should be modified accordingly.

Line 25 is to initialize the host library, after it, the UART device could be added (like line 31). According to the equipment status, multiple UART devices could be added after it. After the devices have been added, the library can be started as line 38, which will start the message receiver, message sender and message handler.

The message API testing is in line 45, in the real case, it could be infinite loop for getting images from user and send it to KDP for the inferring.

After all the calling has been done, line 50 is used to de-initialize the host library to free the resources.

```
23  kdp_init_log("/tmp/", "mzt.log");
24
25  if(kdp_lib_init() < 0) {
26      printf("init for kdp host lib failed.\n");
27      return -1;
28  }
29
30  printf("adding devices....\n");
31  int dev_idx = kdp_add_dev(KDP_UART_DEV, "/dev/ttyUSB0");
32  if(dev_idx < 0 ) {
33      printf("add device failed.\n");
34      return -1;
35  }
36
37  printf("start kdp host lib....\n");
38  if(kdp_lib_start() < 0) {
39      printf("start kdp host lib failed.\n");
40      return -1;
41  }
42
43  for(int i = 0; i < MAX_TIMES; i++) {
44      printf("doing test :%d....\n", i);
45      user_test(dev_idx);
46      usleep(10000);
47  }
48
49  printf("de init kdp host lib....\n");
50  kdp_lib_de_init();
```

8.2 USB device example

Line 23 is to initialization the debug log to specified directory /tmp/mzt.log. If other file name or directory is preferred, it should be modified accordingly.

Line 25 is to initialize the host library, after it, the USB device could be added (like line 31). According to the equipment status, multiple USB devices could be added after it. After the devices have been added, the library can be started as line 38, which will start the message receiver, message sender and message handler.

The message API testing is in line 45, in the real case, it could be infinite loop for getting images from user and send it to KDP for the inferring.

After all the calling has been done, line 50 is used to de-initialize the host library to free the resources.

```
23   kdp_init_log("/tmp/", "mzt.log");
24
25   if(kdp_lib_init() < 0) {
26       printf("init for kdp host lib failed.\n");
27       return -1;
28   }
29
30   printf("adding devices....\n");
31   int dev_idx = kdp_add_dev(KDP_USB_DEV, "");
32   if(dev_idx < 0 ) {
33       printf("add device failed.\n");
34       return -1;
35   }
36
37   printf("start kdp host lib....\n");
38   if(kdp_lib_start() < 0) {
39       printf("start kdp host lib failed.\n");
40       return -1;
41   }
42
43   for(int i = 0; i < MAX_TIMES; i++) {
44       printf("doing test :%d....\n", i);
45       user_test(dev_idx);
46       usleep(10000);
47   }
48
49   printf("de init kdp host lib....\n");
50   kdp_lib_de_init();
```

8.3 User verification

For the user verification, please refer to the following code segment:

```
int ret = kdp_start_verify_mode(dev_idx, &img_size);
if(ret != 0 || img_size == 0) {
    printf("start verify mode failed :%d, buf size:%d.\n", ret, img_size);
    return -1;
}
printf("starting verify mode successfuly, img size:%d...\n", img_size);

usleep(SLEEP_TIME);

char img_buf[IMAGE_SIZE];
uint16_t u_id = 0;

for(int i = 0; i < 1; i++) { //do 100 times
    memset(img_buf, 0, sizeof(img_buf));

    printf("verifying for image ...\n");
    int n_len = read_file_to_buf(img_buf, "/home/ygchen/1.bin", IMAGE_SIZE);
    if(n_len <= 0) {
        printf("reading file to buf failed:%d...\n", n_len);
        n_len = IMAGE_SIZE;
    }

    u_id = 0;
    ret = kdp_verify_user_id(dev_idx, &u_id, img_buf, n_len);
    if(ret != 0) {
        printf("could not find matched user id.\n");
    } else {
        printf("find matched user id:%d.\n", u_id);
    }
}
```

8.4 User registration

For user registration, please refer to the following code segment:

```
ret = kdp_start_reg_user_mode(dev_idx, u_id, img_idx);
if(ret != 0) {
    printf("could not set to register user mode..\n");
    return -1;
}
printf("register mode succeeded...\n");

usleep(SLEEP_TIME);

if(1) {
    //TODO need set image name here
    memset(img_buf, 0, sizeof(img_buf));

    printf("registering with image...\n");
    int n_len = read_file_to_buf(img_buf, "/home/ygchen/1.bin", IMAGE_SIZE);
    if(n_len <= 0) {
        printf("reading file to buf failed:%d...\n", n_len);
        n_len = IMAGE_SIZE;
    }

    ret = kdp_extract_feature(dev_idx, img_buf, n_len);
    if(ret != 0) {
        printf("could not extract feature for.\n");
    }  else {
        printf("feature extracted successfully for img index.\n");
    }
}

usleep(SLEEP_TIME);

uint32_t user_id = 5;
printf("registering user :%d ...\n", user_id);
ret = kdp_register_user(dev_idx, user_id);
if(ret != 0) {
    printf("register user failed.\n");
    //return -1;
}
printf("registered with user id:%d.\n", user_id);

usleep(SLEEP_TIME);

//delete all user
printf("removing user ...\n");
ret = kdp_remove_user(dev_idx, 0);
```