Kneron Plus Introduction

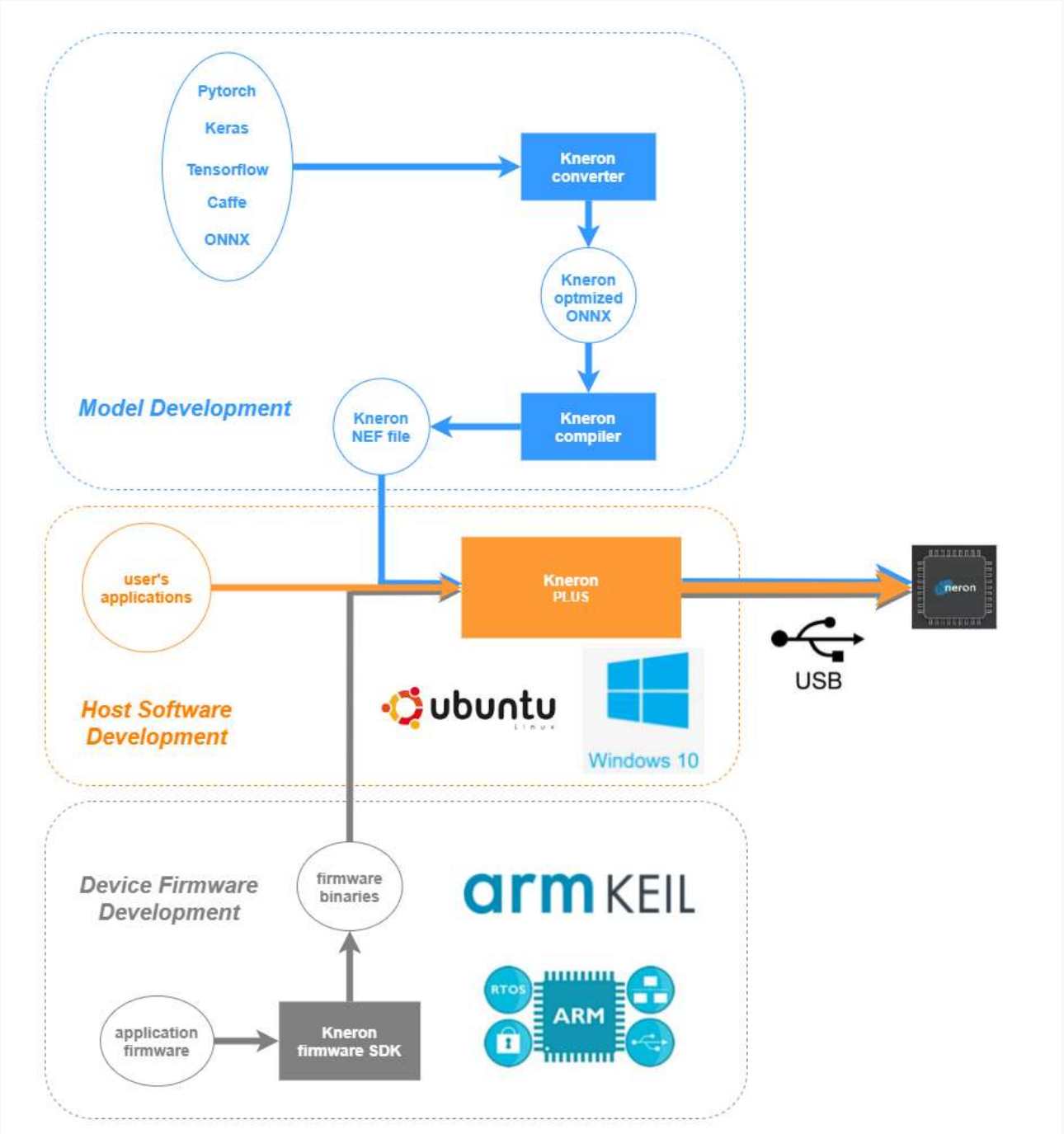
Basic Features

Kneron PLUS stands for *Platform Library Unified Software* which is a framework comprising new software(SW) and firmware(FW) design for KL520 (and alpha for KL720).

In order to run the inference of models on Kneron AI devices, there are three parts of AI application development are required:

- **model development**
- **software development**
- **firmware development**

Below diagram depicts three parts of development in a big picture.



However, this document only focuses on the **software development** and the **firmware development**. For the **model development**, please refer to the Toolchain Docker part.

In comparison with the previous SW/FW framework, this aims to simplify the design flow for AI applications development.

Below gives some definitions regarding the Kneron PLUS:

- **PLUS** is a software library developed by Kneron and it allows users to manipulate the AI device through sophisticated C/C++/Python API.
- **KP** API is part of the PLUS written in C/Python and it provides simplified functions and examples to help users develop their software applications. For the complete list of KP API, please refer to another document.
- **NEF** represents for **NPU Executable Format** which may comprise one or multiple models and it can only work on Kneron's SoC. This package comes with some NEFs for demonstration purposes. We will use the **Tiny Yolo v3** NEF as the model input on KL520 in our inference examples.
- **Firmware** is the code responsible for driving Kneron SoC and make it work with the software library. The KDP2 firmware can work with the PLUS and it has prebuilt images included in the PLUS.

The features which PLUS Supported are listed below:

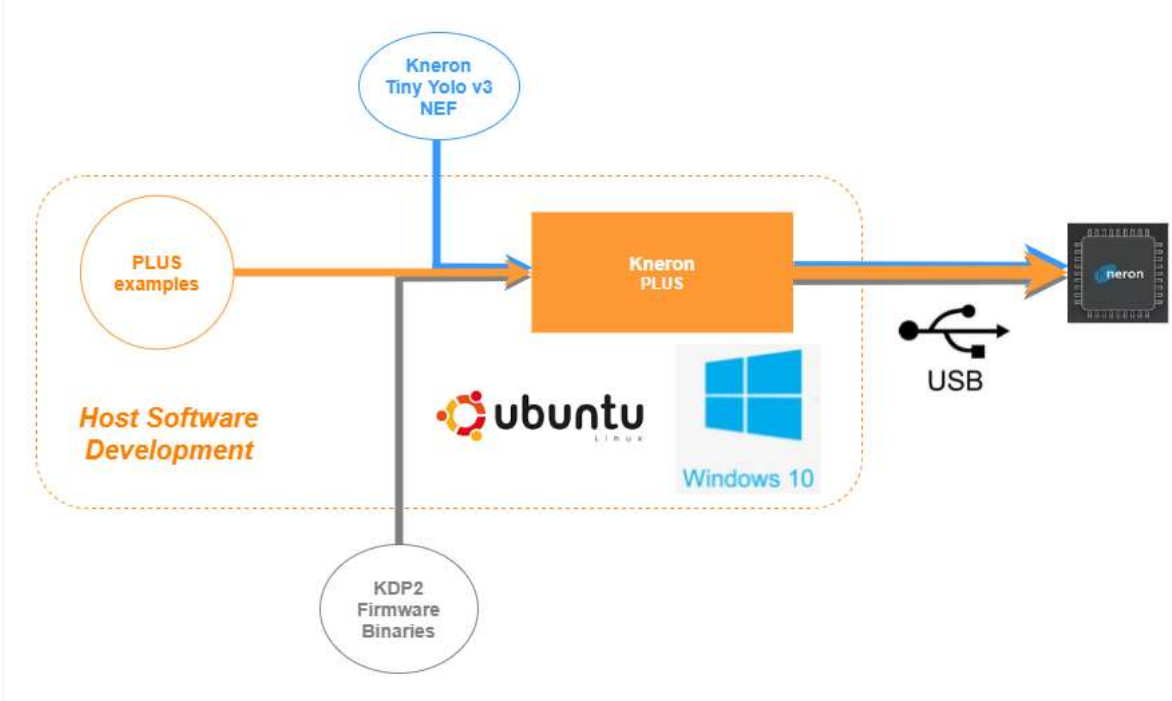
Index	Category	Supported Item	Minimum Version	KL520	KL720
1	Image Format	RGBA8888 , RAW8 , YCbCr422 (YUYV422) , RGB565			
2	System	Firmware In Flash			
3		Model In Flash			
4		Runtime Upload Firmware			X
5		Runtime Upload Model			
6		Software Reset			
7		Software Shutdown (Developing Broad Only)			X
8		Software Reboot			
9		Scan Devices			
10		Device Log via USB			
11		Device Connection : All Devices, Specified Device(s)			

Index	Category	Supported Item	Minimum Version	KL520	KL720
12	Inference	Flexible Send / Receive Inference			
13		Multiple Device Auto Dispatch			
14		Enable / Disable Pre-process on Device			
15		Enable / Disable Post-process on Device			
16		Output Floating Point / Fixed Point Result			
17	System / Model Info	Get Firmware Version			
18		Get KN Number			
19		Get Model CRC			
20		Get Model Info			
21		Install Device Driver for Windows	v1.3.0		
22	Application API	Generic Inference			
23		Customized Inference (C code only)			
24		User Define API (C code only)			
25	System Examples	Get Firmware Info			
26		Get Model Info			
27		Reboot Device			
28		Shutdown Device			X
29	Inference Examples	Generic Inference (Raw Output)			
30		Generic Inference (with Crop)			

Index	Category	Supported Item	Minimum Version	KL520	KL720
31		Generic Inference (with Post Process on Host Side)			
32		Generic Inference (Bypass Pre Process)			
33		Generic Inference (Multiple Threads)	v1.3.0		
34		Generic Inference (Model in Flash)	v1.3.0		
35		Generic Inference (Web Cam with Drop Frame)	v1.3.0		
36		User Define API Inference (Yolo with Config Post Process) (C code only)	v1.3.0		
37		Customized Inference with Single Model (C code only)			X
38		Customized Inference with Multiple Models (C code only)			X
39	Debug Examples	Debug Checkpoints Example	v1.3.0		X
40		Execution Time Profiling Example	v1.3.0		X
41	Model Zoo Examples	Simple examples for pre-trained models			

The following components are contained in Kneron PLUS:

- KP API
- PLUS examples code
- KDP2 firmware code (KL520 only)
- Pre-build firmware binary files
- Some demonstrative NEF files



Advanced Features for Enterprise Version

Besides the basic features, there are few advanced features provided in Kneron PLUS Enterprise:

Note: Most of the advanced features and examples are C code only. Only **Update Kdp2 to Kdp2 Flash Boot** has the python version example.

Index	Category	Supported Item	Minimum Version	KL520	KL720
1	System	Runtime Upload Firmware via UART			X
2		Hico Mode (MIPI image input, Companion Result Output)		X	
2	Examples	Update Kdp to Kdp2 Usb Boot			X
3		Update Kdp2 to Kdp2 Usb Boot			X
4		Update Kdp to Kdp2 Flash Boot			
5		Update Kdp2 to Kdp2 Flash Boot			
6		Update Model to Flash			
7		Upload Firmware via UART			X
8		Read / Write Device Memory			

Index	Category	Supported Item	Minimum Version	KL520	KL720
9		Access Firmware Log via USB		X	
10		Hico Cam Inference (Kneron LW 3D module is required)		X	
11		Hico ToF Inference (Kneron ToF module is required)	v1.3.0	X	

Installation

Verified platforms, OS and Python Version to run Kneron PLUS API:

OS	Platform	Python Version
Windows 10	x86_64 64-bit	3.5-3.9 (x86_64 64-bit)
Ubuntu 18.04	x86_64 64-bit	3.5-3.9 (x86_64 64-bit)
Raspberry Pi OS - Buster	armv7l 32-bit	3.5-3.9 (armv7l 32-bit)

And the following sections in this chapter will provide the instructions for installing the tools and dependency python packages to the corresponding platform.

1. Install Python Package

- Upgrade pip (pip version >= 21.X.X):

```
$ python -m pip install --upgrade pip
```

- Install the package with pip:

```
$ # Please make sure your pip version >= 21.X.X before installing python packages.
$ cd ./package/{platform}/
$ pip install KneronPLUS-{version}-py3-none-any.whl
```

- Install the examples requirement package with pip:

```
$ pip install opencv-python
```

- Common problem:

  If pip install/run application fails, it may cause by using python 2.X as python interpreter. Please make sure the interpreter and pip is `Python 3` on the host:

```
# check pip version
$ pip -V
$ pip3 -V


# check python interpreter version
$ python -V
$ python3 -V
```

You also can install package by specify python interpreter by following scripts:

```
$ python -m pip install {package_path}
# or
$ python3 -m pip install {package_path}
```

2. Install Kneron AI Device Driver on Windows 10

There are three ways to install device driver to Windows:

- Kneron DFUT (Recommended)
- PLUS Example
- Zadig

2.1 Using Kneron DFUT to Install Driver

Note: This feature is only provided in Kneron DFUT v1.3.0 and above.

- Please refer Upgrade AI Device To KDP2 for the usage.

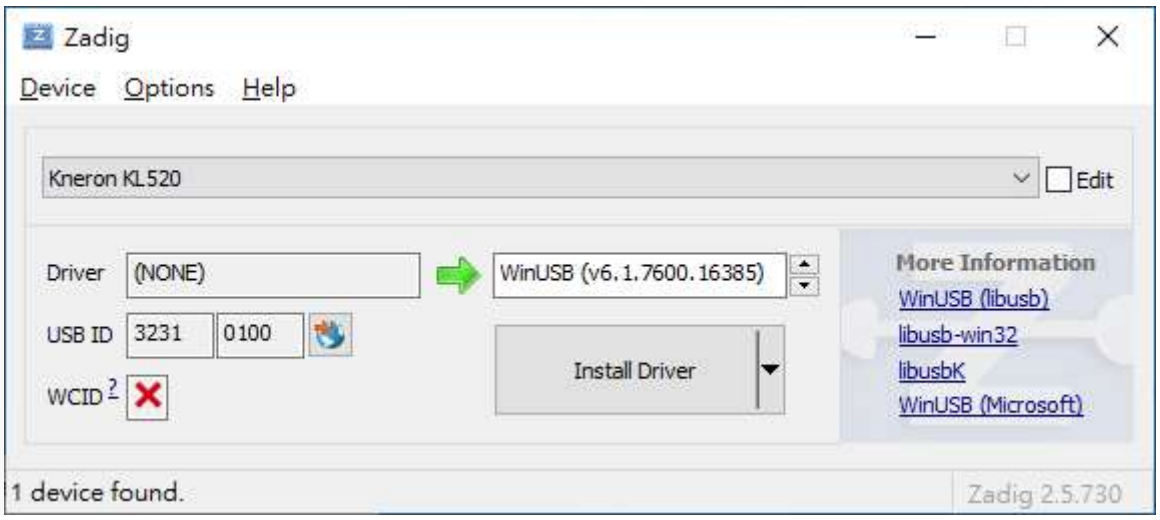
2.2 Using PLUS Example to Install Driver

Note: This feature is only provided in Kneron PLUS v1.3.0 and above.

- Please refer Run Example for the usage.

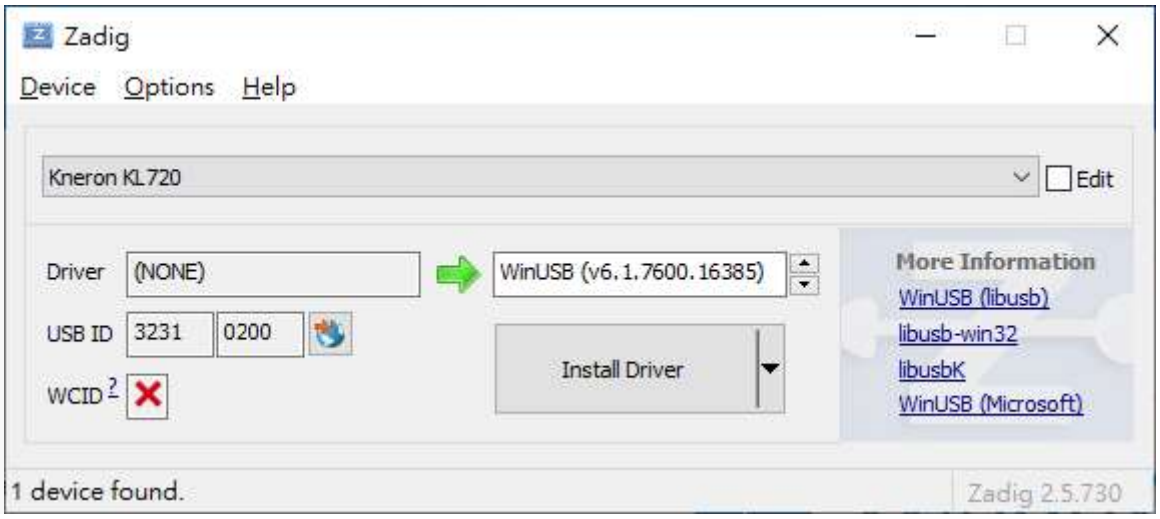
2.3 Using Zadig to Install Driver

- Download Zadig application from zadig.akeo.ie appropriate for Windows 10.
- Connect Kneron KL520/KL720 device to your PC.
- Run the Zadig application.
- KL520
    - The application should detect Kneron KL520 device as "Kneron KL520" with USB ID "3231/0100" as shown below:



  - Make sure that the Driver field, has WinUSB option selected.
  - Click "Install Driver" button.

- KL720
  - The application should detect Kneron KL720 device as "Kneron KL720" with USB ID "3231/0200" as shown below:



  - Make sure that the Driver field, has WinUSB option selected.
  - Click "Install Driver" button.

Note: After Upgrade Kneron KL720 to KDP2 (ex. via Kneron DFUT), you may need to re-install the driver of KL720, since the USB ID will be changed to "3231/0720".

3. Update Kneron AI Device USB Permission on Ubuntu and Raspberry Pi

- Config USB permission on Ubuntu/Raspberry Pi

- `$ sudo bash install_libusb.sh`

- Or add following lines in `/etc/udev/rules.d/10-local.rules` manually

- `KERNEL=="ttyUSB*",ATTRS{idVendor}=="067b",ATTRS{idProduct}=="2303",MODE="0777",SYMLINK+="kneron_uart"`
- `KERNEL=="ttyUSB*",ATTRS{idVendor}=="1a86",ATTRS{idProduct}=="7523",MODE="0777", SYMLINK+="kneron_pwr"`
- `SUBSYSTEM=="usb",ATTRS{idVendor}=="3231",ATTRS{idProduct}=="0100",MODE="0666"`

- `SUBSYSTEM=="usb",ATTRS{idVendor}=="3231",ATTRS{idProduct}=="0200",MODE="0666"`
- `SUBSYSTEM=="usb",ATTRS{idVendor}=="3231",ATTRS{idProduct}=="0720",MODE="0666"`

and restart service by following command (You may need to restart the service after rebooting the host PC)

```
$ sudo udevadm trigger
```

Upgrade AI Device to KDP2 Firmware

Note: KneronDFUT supports 3 platforms - Windows 10 (x86_64 64-bit), Ubuntu 18.04 (x86_64 64-bit), and Raspberry Pi OS - Buster (armv7l 32-bit)

Note: If you are not using the 3 platforms, you may use the DFUT_console provided in Kneron PLUS. Please refer Build with DFUT_console

Note: Please use the latest version of KneronDFUT to avoid problems caused by incompatibility.

Note: Downgrading Kneron AI device to previous KDP firmware is not allowed.

Note: If the Kneron AI device you wish to upgrade is running HICO firmware, please manually reset the device first before the update process.

1. Introduction

KDP2 Firmware is the firmware designed for KP APIs in PLUS. Using KDP2 Firmware allows Kneron AI device performing corresponding operation requested by PLUS.

There are two modes to activate KDP2 firmware in Kneron AI device:

- **Runtime Upload Firmware (USB Boot)**
  - USB boot mode is only available on **KL520**.
  - USB boot mode is using usb to upload KDP2 firmware before the inference process.
  - Uploading firmware requires the assistance from the loader firmware("**KDP2 loader**") in flash memory.
  - The GUI or command line of **KneronDFUT** can be used for writing the loader firmware to flash memory and switch AI devices to USB boot mode.
  - After writing the loader firmware and switching device to USB boot mode. The KDP2 firmware can be uploaded via following KP API, before inference:
    - C user: `kp_load_firmware_from_file()`
    - Python user: `kp.core.load_firmware_from_file()`
- **Firmware in Flash Memory (Flash Boot)**
  - This mode is using the **KDP2 firmware** stored in the flash memory of AI devices.
  - Once the AI device is electrified, the firmware will be automatically activated.
  - The GUI or command line of **KneronDFUT** can be used for writing KDP2 firmware to flash memory and switch AI devices to Flash boot mode.

Note: Some 96 boards may run a customized firmware which does not accept commands from the usb interface. Therefore, these 96 boards are not able to be upgraded to kdp2 firmware through Kneron

DFUT. For the demands of upgrading these 96 boards, please refer Program Flash via JTAG/SWD Interface for KDP2 for more information.

2. Download Kneron DFUT

Download the KneronDFUT_ubuntu.zip into Ubuntu
from https://www.kneron.com/tw/support/developers/. It is located at **Kneron PLUS** section.

```
$ unzip KneronDFUT_ubuntu.zip
$ cd Kneron_DFUT/bin/
```

Command line usage

```
$ sudo ./KneronDFUT --help
[Display help message]
    --help              : [no argument]         help message

[Scan and list all information]
    --list              : [no argument]         list all dongles information

[Update dongles to usb boot] (Only works for KL520)
    --kl520-usb-boot     : [no argument]         choose update to Usb Boot
    --port              : [argument required]   port id set ("all" or specified multiple por
t ids "13,537")

[Update dongles to flash boot] (Only works for KL520)
    --kl520-flash-boot    : [no argument]         choose update to Flash Boot
    --port              : [argument required]   port id set ("all" or specified multiple por
t ids "13,537")
    --scpu              : [argument required]   self pointed scpu firmware file path (.bin)
    --ncpu              : [argument required]   self pointed ncpu firmware file path (.bin)

[Update firmware file to flash memory in dongles (Only works for KL720)
    --kl720-update       : [no argument]         choose write firmware to flash memory
    --port              : [argument required]   port id set ("all" or specified multiple por
t ids "13,537")
    --scpu              : [argument required]   self pointed scpu firmware file path (.bin)
    --ncpu              : [argument required]   self pointed ncpu firmware file path (.bin)

[Update model file to flash memory in dongles
    --model-to-flash     : [argument required]   self pointed model file path (.nef)
    --type              : [argument required]   type of device ("KL520" or "KL720")
    --port              : [argument required]   port id set ("all" or specified multiple por
t ids "13,537")

[Enable Graphic User Interface]
    --gui               : [no argument]         display GUI
```

```
[Get Current Kneron DFUT Version]
   --version            : [no argument]        display the version of Kneron DFUT
```

3. Install Driver for Windows

When you execute any kind of update on Kneron DFUT, it will check whether the driver of KL520 or KL720 has been installed on Windows. If the driver has not been installed, Kneron DFUT will install the driver before any update.

Note: Kneron DFUT only check and install driver when it was executed on Windows.

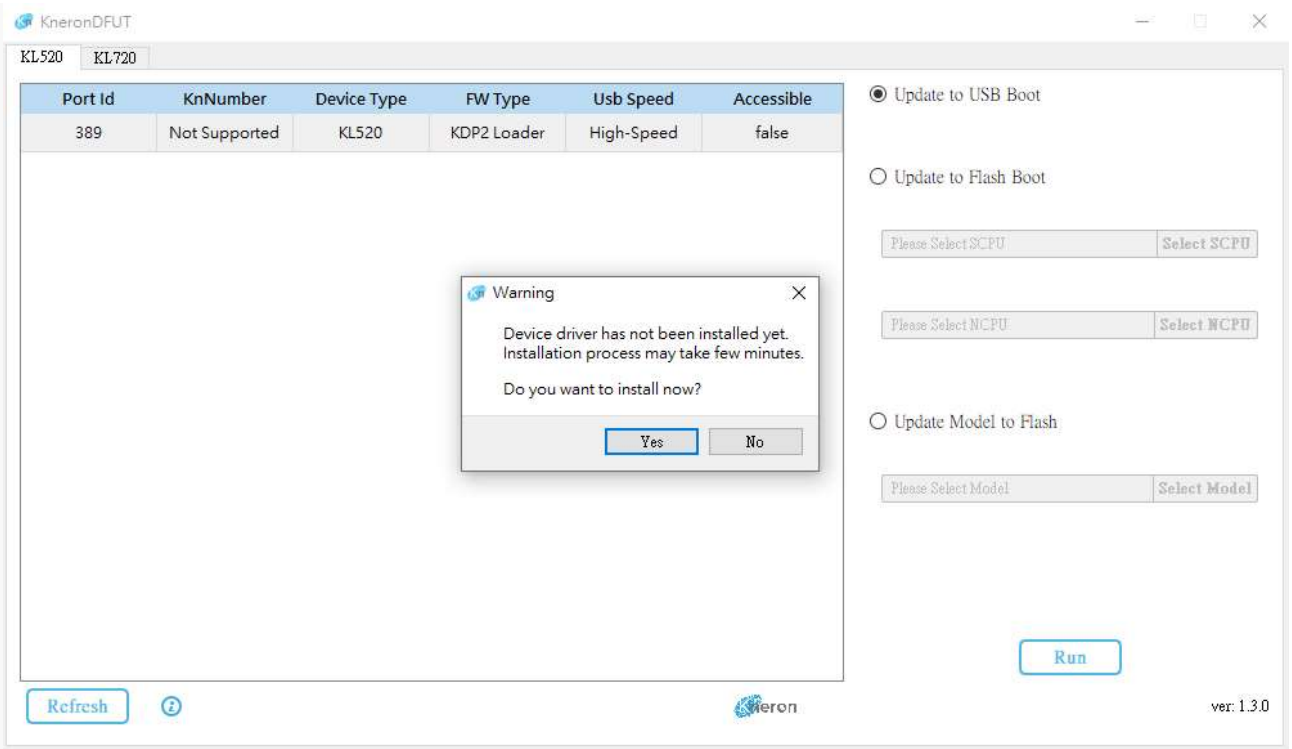
Note: In order to install driver, Kneron DFUT must be run as Administrator.

Note: This feature and example are only provided in Kneron DFUT v1.3.0 and above.

1. Use Command line

```
$ KneronDFUT.exe --kl520-usb-boot --port all
Installing driver for KL520 ... Success(0)

Start Update Device with Port Id 389 to USB Boot

==== Update of Device with Port Id: 389 Succeeded ====
```

8. Use GUI

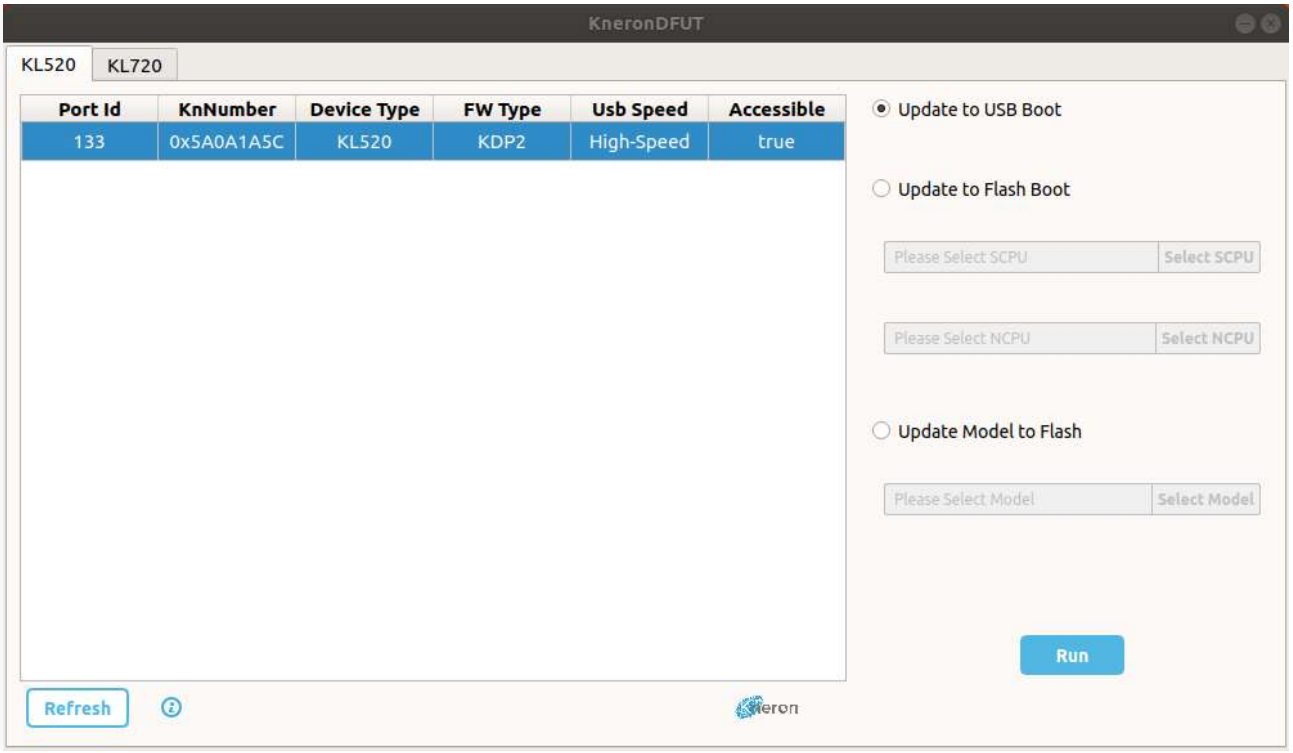


3. [KL520] Update to USB Boot Mode

3.1 Use GUI to Update AI Device

```
$ sudo ./KneronDFUT
```

1. Select **KL520** Tab.
2. Select the KL520 devices to be update to USB Boot Mode.
3. Select **Update to USB Boot**
4. Push **Run** button.



3.2 Use Command Line to Update AI Device

1. List all devices

```
2. $ sudo ./KneronDFUT --list
3. ========================================
4. Index:         1
5. Port Id:       133
6. Kn Number:     0x270A265C
7. Device Type:   KL520
8. FW Type:       KDP
9. Usb Speed:     High-Speed
10. Connectable:   true
11. ========================================
```

12. Upgrade the selected KL520 devices using the port id

```
13. $ sudo ./KneronDFUT --kl520-usb-boot --port 133
14. Start Update Device with Port Id 133 to USB Boot
15.
16. ==== Update of Device with Port Id: 133 Succeeded ====
```

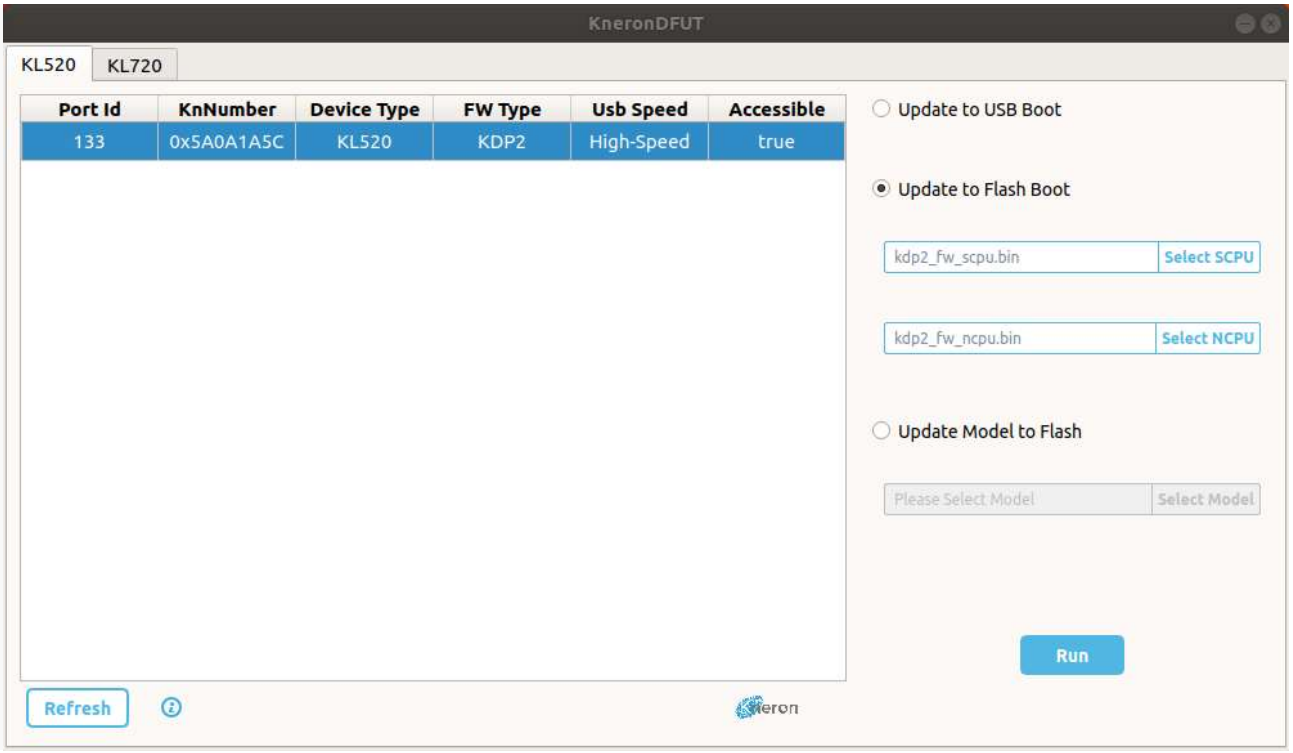
4. [KL520] Update to Flash Boot Mode

4.1 Use GUI to Update AI Device

```
$ sudo ./KneronDFUT
```

1. Select **KL520** Tab.
2. Select the KL520 devices to be **Update to Flash Boot** Mode.
3. Select **Update to Flash Boot**
4. Manually choose **SCPU firmware file** and **NCPU firmware file**.

   SCPU and NCPU firmware file for KL520 can be found in **${PLUS_FOLDER}/res/firmware/KL520/**
5. Push **Run** button.



4.2 Use Command Line to Update AI Device

1. List all devices

```
$ sudo ./KneronDFUT --list
==========================================
Index:          1
Port Id:        133
Kn Number:      0x270A265C
Device Type:    KL520
FW Type:        KDP
Usb Speed:      High-Speed
Connectable:    true
==========================================
```

12. Upgrade the selected KL520 devices using the port id

```
$ sudo ./KneronDFUT --kl520-flash-boot --port 133 --scpu ${SCPU_FILE_PATH} --ncpu ${NCPU_FILE_PATH}
Start Update Device with Port Id 133 to Flash Boot

==== Update of Device with Port Id: 133 Succeeded ====
```

SCPU and NCPU firmware file for KL520 can be found in **${PLUS_FOLDER}/res/firmware/KL520/**

5. [KL720] Update Firmware to Flash Memory

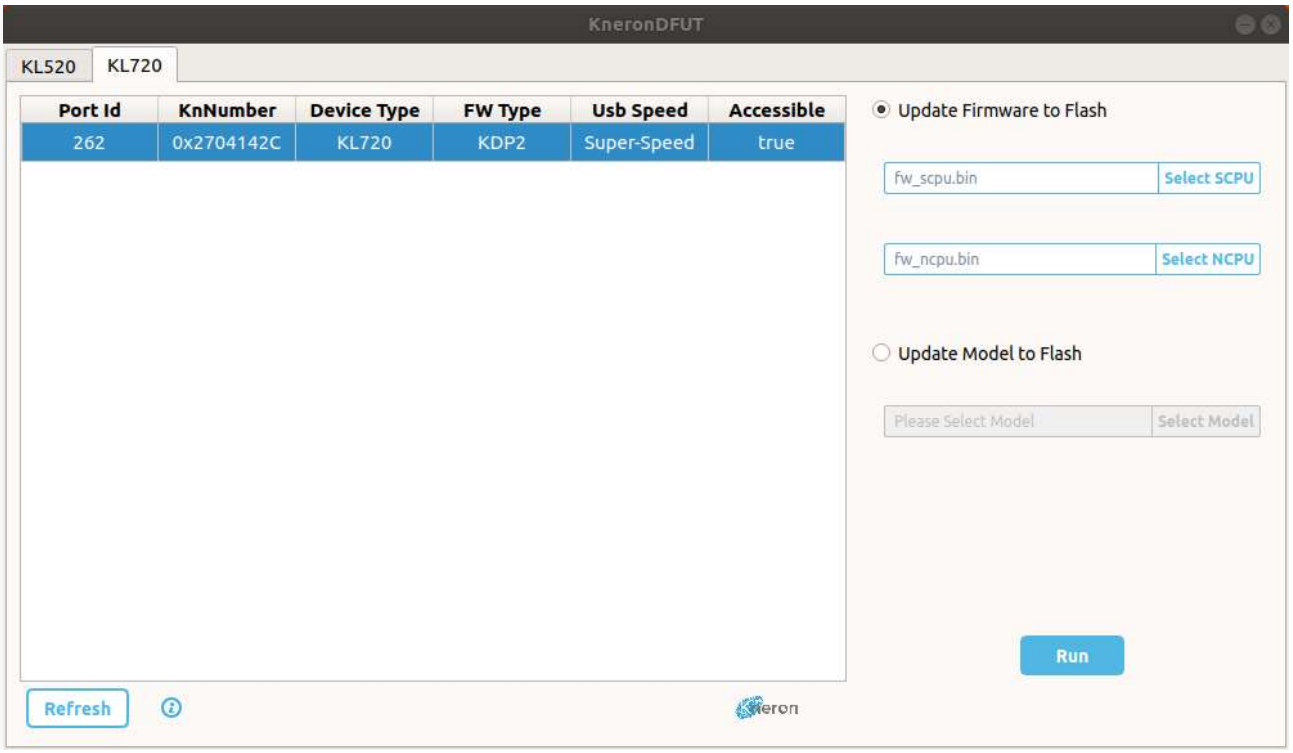
Note: Update flash for KL720 is required under USB3.0(Super-Speed) model

5.1 Use GUI to Update AI Device

```
$ sudo ./KneronDFUT
```

1. Select **KL720** Tab.
2. Select the KL720 devices to be update to KDP2 firmware.
3. Select **Update Firmware to Flash**
4. Manually choose **SCPU firmware file** and **NCPU firmware file**.

   The firmware files can be found in **${PLUS_FOLDER}/res/firmware/KL720/**
5. Push **Run** button.



5.2 Use Command Line to Update AI Device

1. List all devices

```
$ sudo ./KneronDFUT --list
==========================================
```

```
4.  Index:         1
5.  Port Id:       262
6.  Kn Number:     0x2004142C
7.  Device Type:   KL720
8.  FW Type:       KDP
9.  Usb Speed:     Super-Speed
10. Connectable:   true
11. ========================================
```

12. Upgrade the selected KL720 devices using the port id

```
13. $ sudo ./KneronDFUT --kl720-update --port 262 --scpu ${SCPU_FILE_PATH} --ncpu ${NCPU
    _FILE_PATH}
14. Start Update Firmware to Device with Port Id 262
15.
16. ==== Update Firmware to Device with Port Id: 262 Succeeded ====
```

SCPU and NCPU firmware file for KL720 can be found in **${PLUS_FOLDER}/res/firmware/KL720/**

Write Model To Flash

Note: KneronDFUT supports 3 platforms - Windows 10 (x86_64 64-bit), Ubuntu 18.04 (x86_64 64-bit), and Raspberry Pi OS - Buster (armv7l 32-bit)

Note: If you are not using the 3 platforms, you may use the DFUT_console provided in Kneron PLUS. Please refer Build with DFUT_console

Note: Please use the latest version of KneronDFUT to avoid problems caused by incompatibility.

1. Introduction

The inference model must be loaded into Kneron AI device before the inference process.

There are two ways to load models:

- **Upload Model via USB**
  - The model file (.nef) can be uploaded to Kneron AI device using `kp_load_model_from_file()`, a KP API, before inference.
  - For Python users, the model file (.nef) can be uploaded to Kneron AI device using `kp.core.load_model_from_file()`.
  - For the usage, please refer examples related to inference.
  - In this method, the size of the model in device DDR memory (larger than NEF file size) must be below **35 MB** for KL520, and **75 MB** for KL720.
- **Load Model from Flash**
  - The model file (.nef) is written in flash, and it can be loaded using `kp_load_model_from_flash()`, a KP API, before inference.
  - For Python users, the model file (.nef) is written in flash, and it can be loaded using `kp.core.load_model_from_flash()`.

- The GUI or command line of **KneronDFUT** can be used for writing the model file into flash.
- For the usage, please refer the example **kl520_demo_generic_inference_flash_model** or **kl720_demo_generic_inference_flash_model**.
- For Python users, please refer the example **KL520DemoGenericInferenceFlashModel.py** or **KL720DemoGenericInferenceFlashModel.py**.
- In this method, the size of the model file must be below **32 MB** for KL520, and **70 MB** for KL720.

Note: Only one model file (.nef) can be loaded, no matter it was uploaded via USB or loaded from flash. If you want to change the model, please reboot the Kneron AI device.

Note: Upload model via USB can be directly used without writing model into flash.

2. Download Kneron DFUT

Download the KneronDFUT_ubuntu.zip into Ubuntu from https://www.kneron.com/tw/support/developers/. It is located at **Kneron PLUS** section.

```
$ unzip KneronDFUT_ubuntu.zip
$ cd Kneron_DFUT/bin/
```

Show help message

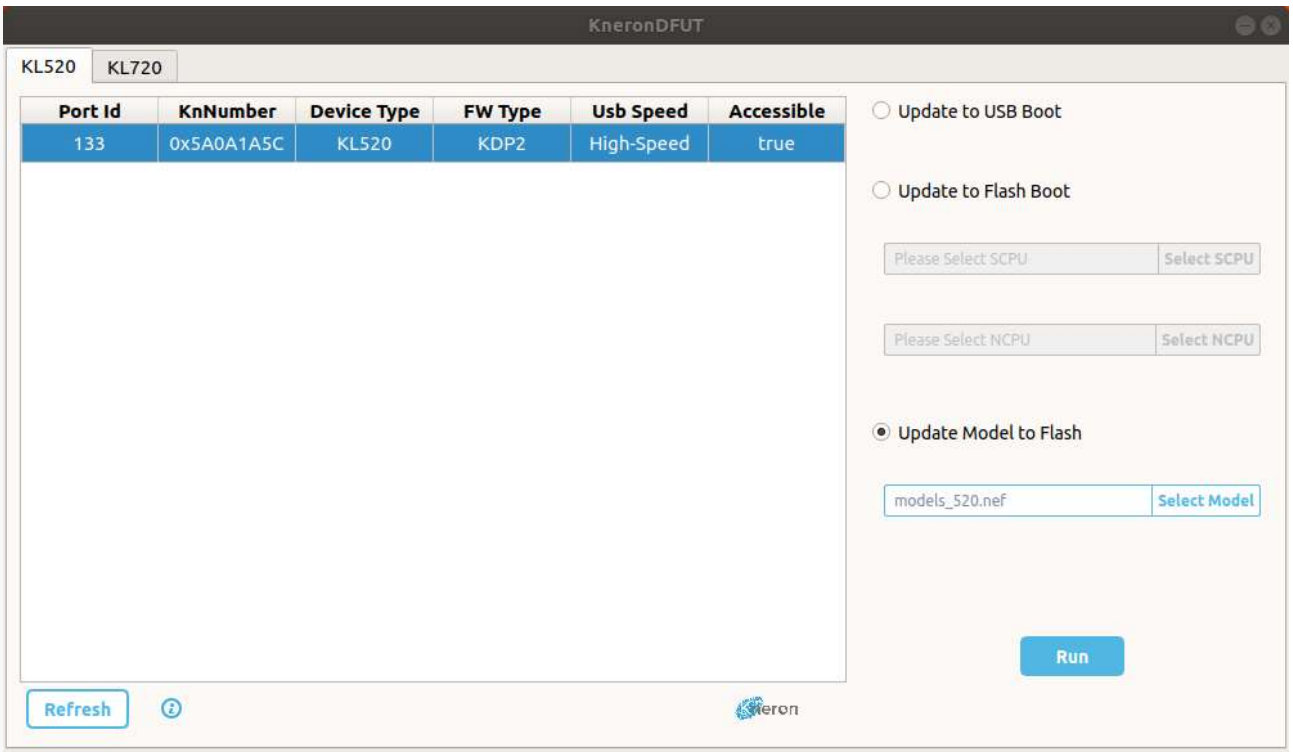
```
$ sudo ./KneronDFUT --help
```

3. Write Model Into KL520

3.1 Use GUI to Write Model into AI Device

```
$ sudo ./KneronDFUT
```

1. Select **KL520** Tab.
2. Select the KL520 devices to write model into.
3. Select **Update Model to Flash**
4. Manually choose **Model file**.
5. Push **Run** button.



3.2 Use Command Line to Write Model into AI Device

1. List all devices

```
$ sudo ./KneronDFUT --list
==========================================
Index:          1
Port Id:        133
Kn Number:      0x270A265C
Device Type:    KL520
FW Type:        KDP
Usb Speed:      High-Speed
Connectable:    true
==========================================
```

12. Write model into the selected KL520 devices using the port id

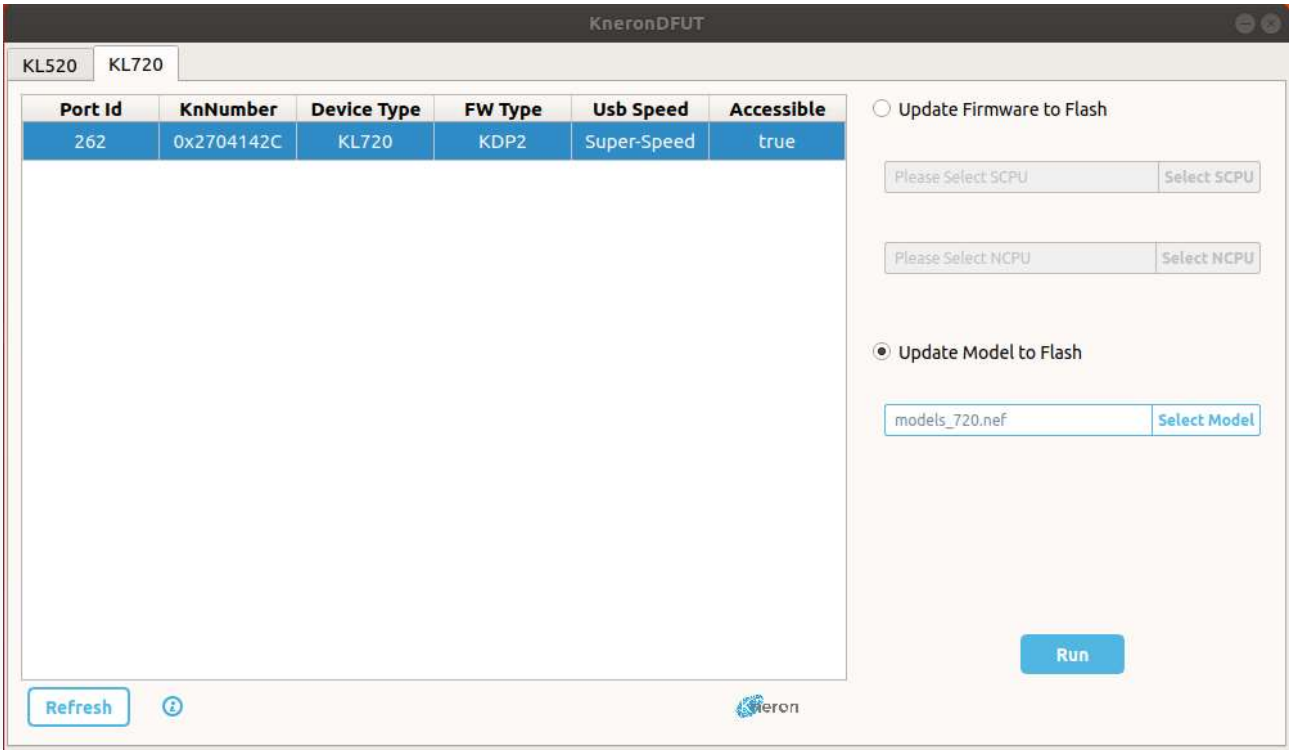
```
$ sudo ./KneronDFUT --model-to-flash ${MODEL_FILE_PATH} --port 133 -- type KL520
Start Update Model to Device with Port Id 133

==== Update Model to Device with Port Id: 133 Succeeded ====
```

4. Write Model Into KL720

4.1 Use GUI to Write Model into AI Device

```
$ sudo ./KneronDFUT
```

1. Select **KL720** Tab.
2. Select the KL720 devices to write model into.
3. Select **Update Model to Flash**
4. Manually choose **Model file**.
5. Push **Run** button.



4.2 Use Command Line to Write Model into AI Device

1. List all devices

```
$ sudo ./KneronDFUT --list
==========================================
Index:          1
Port Id:        262
Kn Number:      0x2004142C
Device Type:    KL720
FW Type:        KDP
Usb Speed:      Super-Speed
Connectable:    true
==========================================
```

2. Write model into the selected KL720 devices using the port id

```
$ sudo ./KneronDFUT --model-to-flash ${MODEL_FILE_PATH} --port 262 --type KL720
Start Update Model to Device with Port Id 262

==== Update Model to Device with Port Id: 262 Succeeded ====
```

Run Examples

The provided examples are designed to show how to use KP APIs and present Kneron Device features. Error handling, wording and application layer features are not covered. They are open for more creatives.

Note 1: In the inference related examples, we are using KL520 for most demo. If you wish to use KL720, just change the prefix of the example name from kl520 to kl720.

Note 2: **[Ubuntu]** Please update Kneron device USB permission before following steps on Ubuntu. See the Installation for details.

Note 3: Few examples will auto connect multiple devices to run inference. If you put hybrid types of devices on host, the inference may fail.

Note 4: If you modify code to change different test image file. Input image aspect ratio is suggested to be aligned to model input aspect ratio.

1. Scan Device Example

Note: This example is to show the usage of `kp.core.scan_devices()`.

While one or multiple AI devices are plugged into the host, they can be scanned to get some basic device information.

```
$ python3 ScanDevices.py


scanning kneron devices ...
number of Kneron devices found: 2
listing devices information as follows:


[0] USB scan index: '0'
[0] USB port ID: '25'
[0] Product ID: '0x100 (KL520)'
[0] USB link speed: '3'
[0] USB port path: '1-6'
[0] KN number: '0xC8062D2C'
[0] Connectable: 'True'
[0] Firmware: 'KDP2 Loader'


[1] USB scan index: '1'
[1] USB port ID: '61'
[1] Product ID: '0x720 (KL720)'
[1] USB link speed: '4'
[1] USB port path: '1-15'
```

```
[1] KN number: '0x1B04132C'
[1] Connectable: 'True'
[1] Firmware: 'KDP2'
```

Above shows that it founds two Kneron devices, a brief description listed below.

- **USB scan index** : An index number represents the device in the scanned order, can be used by KP API to establish USB connection.
- **USB port ID** : An unique number represents the device on the certain usb port, can be used by KP API to establish USB connection.
- **Product ID** : The product ID.
- **USB link speed** : USB link speed, High-Speed is fastest speed for USB2.0.
- **USB port path** : This means the physical USB port path on the host.
- **KN number** : Kneron's serial number for the device.
- **Connectable** : It tells if this device is connectable; one device can only be connected by one program at the same time.
- **Firmware** : This shows which firmware the AI device is using, KDP, KDP2 or KDP2 Loader.

2. Install Driver for Windows Example

Note: This example is to show the usage of `kp.core.install_driver_for_windows()` and help users to install driver to Windows directly.

Note: This example is only available on Windows 10, and it must be run as Administrator (Run CMD/PowerShell as administrator).

1. For installing the driver for KL520:

```
$ python3 InstallDriverWindows.py --target KL520
[Note]
 - You must run this app as administrator on Windows
[Installing Driver]
 - [KP_DEVICE_KL520]
    - Success
```

2. For installing the driver for KL720:

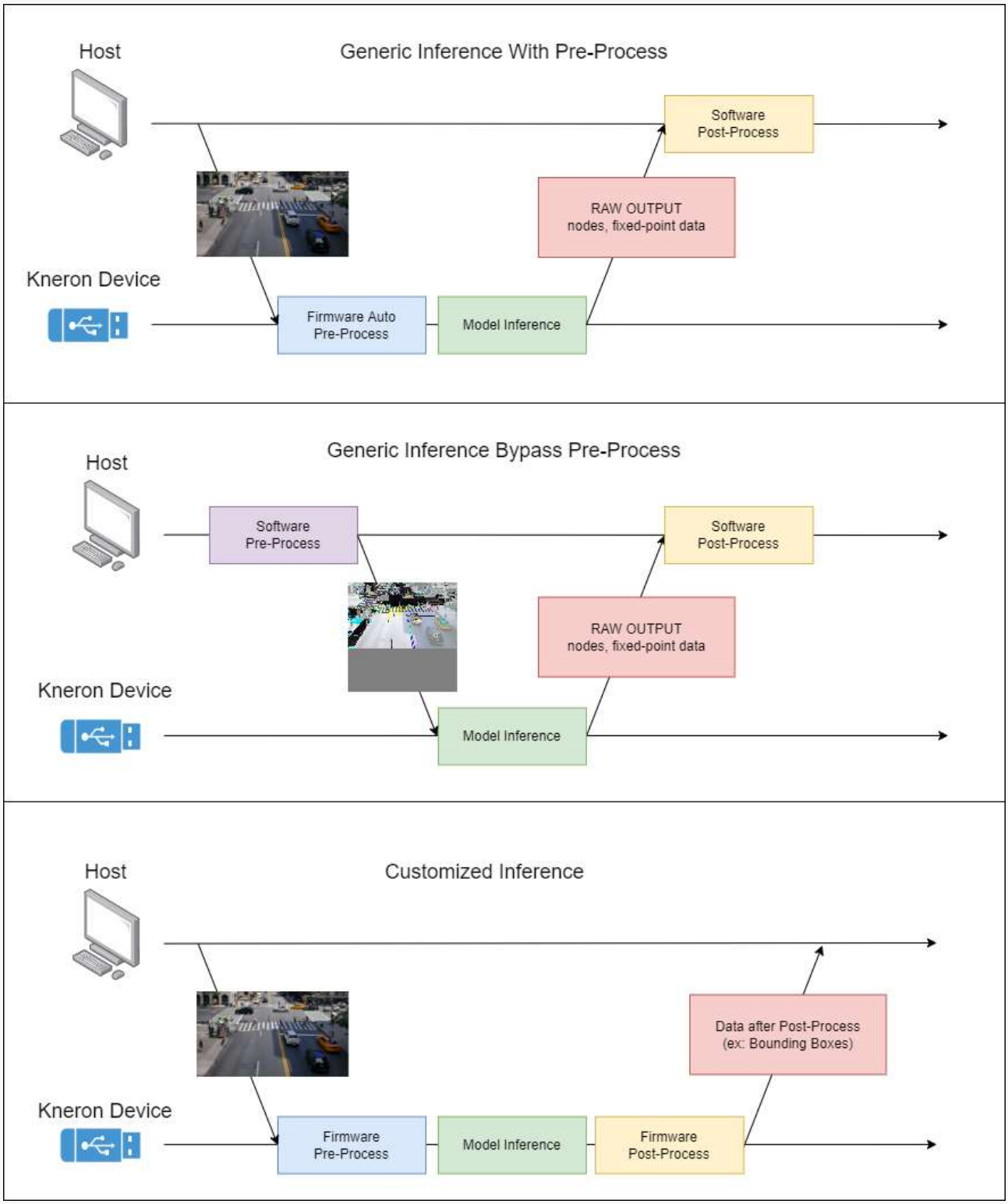
```
$ python3 InstallDriverWindows.py --target KL720
[Note]
 - You must run this app as administrator on Windows
[Installing Driver]
 - [KP_DEVICE_KL720_LEGACY]
    - Success
 - [KP_DEVICE_KL720]
    - Success
```

3. Run Inference Examples

The PLUS provides two categories of API set for model inference.

1. **Generic inference** category which is intended for advanced users who are interested in developing their models and implement corresponding post-processing code.
2. **Customized inference (C Language Only)** category which is intended for advanced users who are interested in developing their models and implement corresponding post-processing code **on Kneron AI devices** (or implement different pre-processing on devices)

The main difference between **Generic Inference** and **Customized Inference** is shown below:



Below will demonstrate only usage in two examples for **Generic inference**. For **Customized inference (C Language Only)**, please refer the **C language documents**.

3.2 Generic Inference Example

Following examples show the usage of `kp.inference.generic_raw_inference_send()` and `kp.inference.generic_raw_inference_receive()`.

Generic inference examples are using the **Generic Inference API**, which is intended for advanced users who are interested in developing their models and implement corresponding pre/post-processing code.

Generic Inference API allows users to directly run a model with or without Kneron pre-processing and obtain the raw output from the model, without any developments of Kneron AI device's firmware. Please refer following sections for the demonstration of the usage:

- 3.2.1 Generic Inference With Raw Output
- 3.2.2 Generic Inference Without Kneron Pre-Processing On Device

However, **Generic Inference API** can only provide the raw output from the model without post-processing. If you wish to get the result with post-processing, you may implement the corresponding post-processing in Software. Please refer following sections for the demonstration of the usage:

- 3.2.3 Generic Inference With Post-Processing

In **Generic Inference API**, you may customized what to do in the pre-processing. There are few items are provided:

1. Image Resize
   - You can choose to do or not to do the image resize by setting `resize_mode` in `kp.GenericRawImageHeader`.
2. Image Padding
   - You can choose to do *Symmetric Padding* (Top, Bottom, Left, Right), *Corner Padding* (Right, Bottom), and not to do the image padding by setting `padding_mode` in `kp.GenericRawImageHeader`.
3. Image Cropping
   - You can choose to do or not to do the image cropping by setting `inference_crop_box_list` in `kp.GenericRawImageHeader`.
   - Please refer 3.2.4 Generic Inference With Cropping Image in Pre-Process for the demonstration.
4. Image Format
   - You have to provide the format of the input image correctly by setting `image_format` in `kp.GenericRawImageHeader`.
   - In the pre-process, the image will be convert to the format *RGBA8888*.
5. Data Normalization
   - You can choose to do *Kneron Normalization*, *Tensor Flow Normalization*, *Yolo Normalization*, or other *Customized Normalization* by setting `normalize_mode` in `kp.GenericRawImageHeader`.

Generic Inference API provide following functions to retrieve specific output node data (More information please reference kp API Document - kp.inference):

Retrieve Node Function	Description
kp.inference.generic_inference_retrieve_fixed_node()	Retrieves and converts RAW format data to **fixed-point** data on the per-node basis.
kp.inference.generic_inference_retrieve_float_node()	Retrieves and converts RAW format data to **floating-point** data on the per-node basis.

3.2.1 Generic Inference With Raw Output

The example **'KL520DemoGenericInference'** not do any post-processing and prints feature map raw output for each output node.

```
$ python3 KL520DemoGenericInference.py


[Connect Device]
 - Success
[Set Device Timeout]
 - Success
[Upload Firmware]
 - Success
[Upload Model]
 - Success
[Read Image]
 - Success
[Starting Inference Work]
 - Starting inference loop 50 times
 - ..................................................
[Retrieve Inference Node Output ]
 - Success
[Result]
[{
    "width": 7,
    "height": 7,
    "channel": 255,
    "channels_ordering": "ChannelOrdering.KP_CHANNEL_ORDERING_CHW",
    "num_data": 12495,
    "ndarray": [
        "[[[[  1.3589103    0.33972758   0.50959134 ...   0.16986379",
        "     0.33972758  -0.849319   ]",
        "   [  1.698638    -0.50959134   0.50959134 ...  -0.16986379",
        "     -0.16986379  -0.849319   ]",
        "   [  1.5287741    0.50959134   0.16986379 ...   0.",
        "     -0.33972758  -0.50959134]",
        "   ...",
        "   [  1.3589103    1.5287741   -1.0191827  ...   2.547957",
        "     -1.1890465   -0.67945516]",
        "   [  1.3589103    1.8685017   -1.0191827  ...   1.0191827",
        "     -0.67945516  -0.33972758]",
        "   [  0.849319     0.849319    -0.16986379 ...   0.16986379",
        "      0.16986379  -0.33972758]]",
        "",
```

```
        "  ...",
        "",
        "  [[ -8.49319    -9.342508   -10.021964   ... -10.191828",
        "    -9.002781   -7.6438704 ]",
        "   [ -7.983598   -10.021964   -9.852099   ...  -9.172645",
        "    -8.323326   -6.6246877 ]",
        "   [ -7.983598   -10.191828   -10.021964   ...  -8.153461",
        "    -7.983598   -7.304143  ]",
        "   ...",
        "   [ -7.6438704   -9.682236   -10.361691   ...  -9.002781",
        "    -10.021964   -9.172645  ]",
        "   [ -7.304143   -10.701419   -12.400057   ...  -9.682236",
        "    -9.682236   -9.002781  ]",
        "   [ -6.1150966   -8.153461   -9.342508   ...  -7.983598",
        "    -8.153461   -8.49319   ]]]]"
    ]
}, {
    "width": 14,
    "height": 14,
    "channel": 255,
    "channels_ordering": "ChannelOrdering.KP_CHANNEL_ORDERING_CHW",
    "num_data": 49980,
    "ndarray": [
        "[[[[  0.8736945  -0.3494778  -0.1747389 ...   0.        -0.1747389",
        "    -0.6989556]",
        "   [  0.6989556  -0.8736945  -0.6989556 ...  -0.5242167  -0.1747389",
        "    -0.5242167]",
        "   [  0.5242167  -0.8736945  -0.6989556 ...  -0.1747389   0.1747389",
        "    -0.8736945]",
        "   ...",
        "   [ -1.0484334   0.         0.        ...   0.3494778   0.3494778",
        "     0.1747389]",
        "   [ -0.6989556  -0.5242167  -0.1747389 ...   0.5242167   0.3494778",
        "     0.1747389]",
        "   [  0.1747389   0.         0.1747389 ...   0.5242167   0.3494778",
        "     0.      ]]",
        "",
        "  ...",
        "",
        "  [[ -7.5137725  -9.261162   -9.785378  ... -10.659073  -10.484334",
        "    -8.562206 ]",
        "   [-10.484334  -13.280156  -14.678067  ... -15.202284  -15.202284",
        "    -11.008551 ]",
        "   [-11.18329   -14.678067  -15.202284  ... -16.07598   -15.027545",
```

```
        "    -12.231723 ]",
        "   ...",
        "   [-11.18329   -15.377023  -18.172846  ... -13.105417  -12.581201",
        "    -10.833812 ]",
        "   [-10.134856  -14.153851  -16.774935  ... -10.659073   -9.61064",
        "     -8.387467 ]",
        "   [ -9.086423  -12.231723  -12.930678  ... -10.134856   -9.261162",
        "     -7.3390336]]]]"
    ]
}]
```

3.2.2 Generic Inference Without Kneron Pre-Processing On Device

The **'KL520DemoGenericInferenceBypassPreProc.py'** is an example for showing how it gets raw output from device, running a Tiny Yolo v3 model with a non pre-processing required image (normalized, same size as model input required, and in format RGBA8888). This example shows the usage of `kp.inference.generic_raw_inference_bypass_pre_proc_send()` and `kp.inference.generic_raw_inference_bypass_pre_proc_receive()`.

```
$ python3 KL520DemoGenericInferenceBypassPreProc.py


[Connect Device]
 - Success
[Set Device Timeout]
 - Success
[Upload Firmware]
 - Success
[Upload Model]
 - Success
[Read Image]
 - Success
[Starting Inference Work]
 - Starting inference loop 50 times
 - ..................................................
[Retrieve Inference Node Output ]
 - Success
[Tiny Yolo V3 Post-Processing]
 - Success
[Result]
{
    "class_count": 80,
    "box_count": 6,
    "box_list": {
        "0": {
```

```
        "x1": 46,
        "y1": 62,
        "x2": 91,
        "y2": 191,
        "score": 0.9704,
        "class_num": 0
    },
    "1": {
        "x1": 44,
        "y1": 96,
        "x2": 99,
        "y2": 209,
        "score": 0.5356,
        "class_num": 1
    },
    "2": {
        "x1": 122,
        "y1": 70,
        "x2": 217,
        "y2": 183,
        "score": 0.9976,
        "class_num": 2
    },
    "3": {
        "x1": 87,
        "y1": 85,
        "x2": 131,
        "y2": 117,
        "score": 0.4992,
        "class_num": 2
    },
    "4": {
        "x1": 28,
        "y1": 77,
        "x2": 56,
        "y2": 99,
        "score": 0.4109,
        "class_num": 2
    },
    "5": {
        "x1": 3,
        "y1": 84,
        "x2": 48,
        "y2": 181,
```

```
            "score": 0.2346,
            "class_num": 2
        }
    }
}
```

3.2.3 Generic Inference With Post-Processing

Note: Reference to Yolo Object Name Mapping for the detection result classes of YOLO examples.

The **'KL520DemoGenericInferencePostYolo.py'** is an example for showing how it gets raw output from device, running a Tiny Yolo v3 model, and does post-processing in the software (host side).

```
$ python3 KL520DemoGenericInferencePostYolo.py


[Connect Device]
 - Success
[Set Device Timeout]
 - Success
[Upload Firmware]
 - Success
[Upload Model]
 - Success
[Read Image]
 - Success
[Starting Inference Work]
 - Starting inference loop 50 times
 - ..................................................
[Retrieve Inference Node Output ]
 - Success
[Tiny Yolo V3 Post-Processing]
 - Success
[Result]
{
    "class_count": 80,
    "box_count": 6,
    "box_list": {
        "0": {
            "x1": 46,
            "y1": 62,
            "x2": 91,
            "y2": 191,
            "score": 0.965,
            "class_num": 0
        },
```

```
        "1": {
            "x1": 44,
            "y1": 96,
            "x2": 99,
            "y2": 209,
            "score": 0.4651,
            "class_num": 1
        },
        "2": {
            "x1": 122,
            "y1": 70,
            "x2": 218,
            "y2": 183,
            "score": 0.998,
            "class_num": 2
        },
        "3": {
            "x1": 87,
            "y1": 85,
            "x2": 131,
            "y2": 117,
            "score": 0.4991,
            "class_num": 2
        },
        "4": {
            "x1": 28,
            "y1": 77,
            "x2": 55,
            "y2": 100,
            "score": 0.368,
            "class_num": 2
        },
        "5": {
            "x1": 3,
            "y1": 84,
            "x2": 48,
            "y2": 181,
            "score": 0.2297,
            "class_num": 2
        }
    }
}
```


[Output Result Image]

- Output bounding boxes on 'output_bike_cars_street_224x224.bmp'

It draws detected objects in a new-created **output_one_bike_many_cars_224x224.bmp**.

3.2.4 Generic Inference With Cropping Image in Pre-Process

Note: Reference to Yolo Object Name Mapping for the detection result classes of YOLO examples.

The **'KL520DemoGenericInferenceCrop.py'** is an example for showing how to do cropping image on device, execute inference only on the cropped areas of image, get the raw output from device, and does post-processing in the software.

The flow in concept:

1. Setting crop information in `kp.GenericRawImageHeader`
2. Send an image to inference
3. Receive result *N* times (*N* specify for number of crop bounding boxes)

```
$ python3 KL520DemoGenericInferenceCrop.py


[Connect Device]
 - Success
[Set Device Timeout]
 - Success
[Upload Firmware]
 - Success
[Upload Model]
 - Success
[Read Image]
 - Success
[Starting Inference Work]
 - Starting inference loop 50 times
 - ..................................................
[Retrieve Inference Node Output ]
 - Success
[Tiny Yolo V3 Post-Processing]
 - Success
[Result]
[Connect Device]
 - Success
```

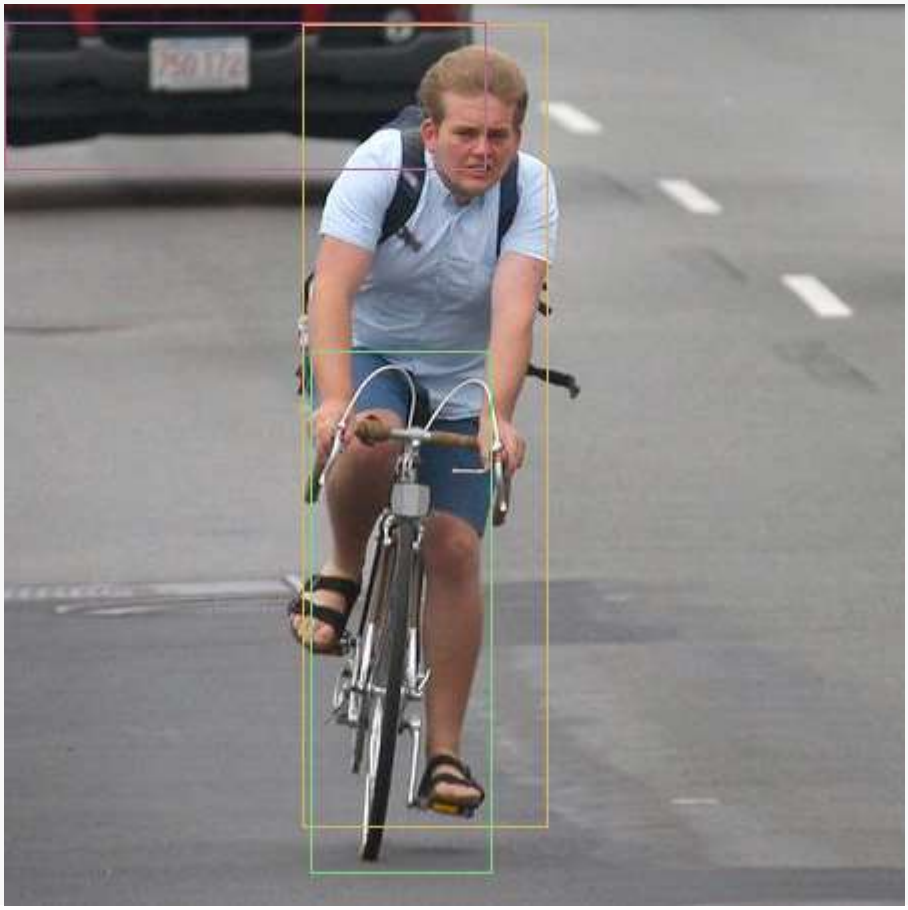
```
[Set Device Timeout]
 - Success
[Upload Firmware]
 - Success
[Upload Model]
 - Success
[Read Image]
 - Success
[Starting Inference Work]
 - Starting inference loop 50 times
 - ..................................................
 - Retrieve 2 Nodes Success
[Post-Processing]
 - Success
[Result]
 - total inference 50 images

[Crop Box 0]
 - [Crop Box Information]
{
    "crop_box_index": 0,
    "x": 0,
    "y": 0,
    "width": 400,
    "height": 400
}
 - [Crop Box Result]
{
    "class_count": 80,
    "box_count": 5,
    "box_list": {
        "0": {
            "x1": 120,
            "y1": 144,
            "x2": 399,
            "y2": 397,
            "score": 0.941,
            "class_num": 2
        },
        "1": {
            "x1": 248,
            "y1": 54,
            "x2": 392,
            "y2": 154,
```

```
            "score": 0.8298,
            "class_num": 2
        },
        "2": {
            "x1": 0,
            "y1": 96,
            "x2": 198,
            "y2": 218,
            "score": 0.6638,
            "class_num": 2
        },
        "3": {
            "x1": 159,
            "y1": 25,
            "x2": 330,
            "y2": 106,
            "score": 0.2677,
            "class_num": 2
        },
        "4": {
            "x1": 17,
            "y1": 81,
            "x2": 62,
            "y2": 224,
            "score": 0.224,
            "class_num": 2
        }
    }
}

[Crop Box 1]
 - [Crop Box Information]
{
    "crop_box_index": 1,
    "x": 230,
    "y": 335,
    "width": 450,
    "height": 450
}
 - [Crop Box Result]
{
    "class_count": 80,
    "box_count": 3,
    "box_list": {
```

```
        "0": {
            "x1": 149,
            "y1": 10,
            "x2": 271,
            "y2": 410,
            "score": 0.9547,
            "class_num": 0
        },
        "1": {
            "x1": 153,
            "y1": 173,
            "x2": 243,
            "y2": 433,
            "score": 0.7877,
            "class_num": 1
        },
        "2": {
            "x1": 0,
            "y1": 9,
            "x2": 240,
            "y2": 82,
            "score": 0.2248,
            "class_num": 2
        }
    }
}
```


```
[Output Result Image]
 - Output bounding boxes on 'output_one_bike_many_cars_800x800_crop0.bmp'
 - Output bounding boxes on 'output_one_bike_many_cars_800x800_crop1.bmp'
```

And it draws detected objects in a new-created **output_one_bike_many_cars_800x800_crop0.bmp** and **output_one_bike_many_cars_800x800_crop1.bmp**.



4. Multiple Threads Usage Example

In the previous inference related examples, sending images to device and receiving results from device are running sequentially.

However, sending images and receiving results can be done in different threads to maximum the processing speed.

The **'KL520DemoGenericInferenceMultiThread.py'** is an example for showing how to put sending image to device and receiving results from device into two different threads.

```
$ python3 KL520DemoGenericInferenceMultiThread.py


[Connect Device]
 - Success
[Set Device Timeout]
 - Success
[Upload Firmware]
 - Success
[Upload Model]
 - Success
[Read Image]
 - Success
[Starting Inference Work]
 - Starting inference loop 100 times
 -
 ..........................................................................................
............
[Result]
 - Total inference 100 images
 - Time spent: 1.99 secs, FPS = 50.2
[Retrieve Inference Node Output ]
 - Success
[Result]
[{
    "width": 7,
    "height": 7,
    "channel": 255,
    "channels_ordering": "ChannelOrdering.KP_CHANNEL_ORDERING_CHW",
    "num_data": 12495,
    "ndarray": [
        "[[[[  1.3589103    0.33972758   0.50959134 ...   0.16986379",
        "      0.33972758  -0.849319   ]",
        "   [  1.698638    -0.50959134   0.50959134 ...  -0.16986379",
        "     -0.16986379  -0.849319   ]",
        "   [  1.5287741    0.50959134   0.16986379 ...   0.",
        "     -0.33972758  -0.50959134]]",
        "  ...",
        "   [ -7.6438704   -9.682236   -10.361691   ...  -9.002781",
        "    -10.021964    -9.172645   ]",
        "   [ -7.304143   -10.701419   -12.400057   ...  -9.682236",
        "     -9.682236    -9.002781   ]",
        "   [ -6.1150966   -8.153461    -9.342508   ...  -7.983598",
        "     -8.153461    -8.49319    ]]]]"
```

```
    ]
}, {
    "width": 14,
    "height": 14,
    "channel": 255,
    "channels_ordering": "ChannelOrdering.KP_CHANNEL_ORDERING_CHW",
    "num_data": 49980,
    "ndarray": [
        "[[[[  0.8736945  -0.3494778  -0.1747389 ...   0.        -0.1747389",
        "     -0.6989556]",
        "    [  0.6989556  -0.8736945  -0.6989556 ...  -0.5242167  -0.1747389",
        "     -0.5242167]",
        "    [  0.5242167  -0.8736945  -0.6989556 ...  -0.1747389   0.1747389",
        "     -0.8736945]",
        "   ...",
        "    [-11.18329   -15.377023  -18.172846  ... -13.105417  -12.581201",
        "     -10.833812 ]",
        "    [-10.134856  -14.153851  -16.774935  ... -10.659073   -9.61064",
        "     -8.387467 ]",
        "    [ -9.086423  -12.231723  -12.930678  ... -10.134856   -9.261162",
        "     -7.3390336]]]]"
    ]
}]
```

5. Drop Frame Usage Example

If the camera produces frames faster than device inference, displaying frames from camera may be delayed by the inference speed since sending image to device may be blocked when buffer of device is full.

'KL520DemoCamGenericInferenceDropFrame.py' is an example for showing how to config device to drop frame if the buffer is full.

The configure function shows in the following code block:

```
"""
configure inference settings (make it frame-droppable for real-time purpose)
"""
try:
    print('[Configure Inference Settings]')
    kp.inference.set_inference_configuration(device_group=device_group,
                                             inference_configuration=kp.InferenceConfiguration(enable_frame_drop=True))
    print(' - Success')
except kp.ApiKPException as exception:
    print('Error: configure inference settings failed, error = \'{}\''.format(str(exception)))
```

```
    exit(0)
```

6. Model Zoo Examples

Model Zoo examples simply show one image inference via different pre-trained models. The model backbones are available and could be retrained for specific need. Please refer to Model Zoo section for more information.

```
$ python3 KL720KnModelZooGenericInferenceClassification.py


[Connect Device]
 - Success
[Set Device Timeout]
 - Success
[Upload Model]
 - Success
[Read Image]
 - Success
[Starting Inference Work]
 - Starting inference loop 1 times
 - .
[Retrieve Inference Node Output ]
 - Success
[Result]
Top1 class: 0, score: 0.94401616
- Success
```

Run Examples for Enterprise

Other than the examples briefed in Run Examples, Kneron PLUS Enterprise also provides few examples for demonstrating the usage of advanced features.

Note 1: In the inference related examples, we are using KL520 for most demo. If you wish to use KL720, just change the prefix of the example name from kl520 to kl720. (There might be no KL520 version or KL720 version on certain examples.)

Note 2: Few examples will auto connect multiple devices to run inference. If you put hybrid types of devices on host, the inference may fail.

Note 3: If you modify code to change different test image file. Input image aspect ratio is suggested to be aligned to model input aspect ratio.

1. Update KDP2 Firmware to KDP2 Flash Boot

This example is to show the sequence of `kp.core.load_firmware_from_file()` and `kp.core._update_kdp2_firmware_from_file()` to update Kneron AI device from KDP2 firmware (both Usb boot and Flash boot are acceptable) to KDP2 Flash boot.

```
$ python3 KL520UpdateKdp2ToKdp2FlashBoot.py
```

```
[Connect Device]
 - Success
[Upload Firmware]
 - Success
[Update Firmware]
 - Success
```

- **Note**: Firmware Update Support Table

Origin Firmware	Target Firmware	Kneron DFUT	C API	Python API
KDP	KDP2 USB-Boot(KL520 Only)	Yes	Yes	
KDP	KDP2 Flash-Boot	Yes	Yes	
KDP2 USB-Boot	KDP2 Flash-Boot	Yes	Yes	Yes
KDP2 Flash-Boot	KDP2 USB-Boot(KL520 Only)	Yes	Yes	
KDP2 Flash-Boot	KDP2 Flash-Boot	Yes	Yes	Yes

Quickstart for beginners

This short introduction use `kp` to build simple edge AI application by following steps:

1. Connect Kneron device
2. Upload SCPU/NCPU firmware
3. Upload NEF model
4. Inference image

Note 1: Please using pip to install `kp` API and `opencv-python` before the following tutorial. See the Installation for details.

Note 2: Please using `kneronDFUT` to upgrade firmware from **KDP** to **KDP2** before the following tutorial. See the Upgrade AI Device to KDP2 for details.

Import `kp` and `cv2` into your program:

```
import kp
import cv2
```

Get one Kneron device USB port ID for connecting:

```
device_descriptors = kp.core.scan_devices()


if 0 < device_descriptors.device_descriptor_number:
    usb_port_id = device_descriptors.device_descriptor_list[0].usb_port_id
else:
    print('Error: no Kneron device connect.')
    exit(0)
```

Connect the device and get Kneron device handler (Device Group):

```
device_group = kp.core.connect_devices(usb_port_ids=[usb_port_id])
```

Set timeout of the USB communication (Default: infinity wait):

```
kp.core.set_timeout(device_group=device_group,
                    milliseconds=5000)
```

Upload firmware to Kneron device:

Please replace `SCPU_FW_PATH`, `NCPU_FW_PATH` by kdp2_fw_scpu.bin and kdp2_fw_ncpu.bin path (Please find target device firmware under `res/firmware` folder)

```
SCPU_FW_PATH = 'res/firmware/KL520/kdp2_fw_scpu.bin'
NCPU_FW_PATH = 'res/firmware/KL520/kdp2_fw_ncpu.bin'
kp.core.load_firmware_from_file(device_group=device_group,
                                scpu_fw_path=SCPU_FW_PATH,
                                ncpu_fw_path=NCPU_FW_PATH)
```

Upload NEF model to Kneron device:

Please replace `MODEL_FILE_PATH` by models_520.nef path (Please find target device NEF model under `res/models` folder)

```
MODEL_FILE_PATH = 'res/models/KL520/tiny_yolo_v3/models_520.nef'
model_nef_descriptor = kp.core.load_model_from_file(device_group=device_group,
                                                    file_path=MODEL_FILE_PATH)
```

Inference image:

- Read image from disk > Please replace `IMAGE_FILE_PATH` by image path (Example image can be found under `res/images` folder)

```
IMAGE_FILE_PATH = 'res/images/bike_cars_street_224x224.bmp'

img = cv2.imread(filename=IMAGE_FILE_PATH)
img_bgr565 = cv2.cvtColor(src=img, code=cv2.COLOR_BGR2BGR565)
```

- Prepare generic inference configuration

```
generic_raw_image_header = kp.GenericRawImageHeader(
    model_id=model_nef_descriptor.models[0].id,
    resize_mode=kp.ResizeMode.KP_RESIZE_ENABLE,
    padding_mode=kp.PaddingMode.KP_PADDING_CORNER,
    normalize_mode=kp.NormalizeMode.KP_NORMALIZE_KNERON,
    inference_number=0
)
```

- Start inference work

```
kp.inference.generic_raw_inference_send(device_group=device_group,
                                        generic_raw_image_header=generic_raw_image_header,
                                        image=img_bgr565,
                                        image_format=kp.ImageFormat.KP_IMAGE_FORMAT_RGB565)

generic_raw_result = kp.inference.generic_raw_inference_receive(device_group=device_group,
                                                                generic_raw_image_header=generic_raw_image_header,
                                                                model_nef_descriptor=model_nef_descriptor)
```

- Retrieve inference node output

```
inf_node_output_list = []

for node_idx in range(generic_raw_result.header.num_output_node):
```

```
    inference_float_node_output = kp.inference.generic_inference_retrieve_float_node(node_idx=node_idx,
                                                                                     generic_raw_result=generic_raw_result,
                                                                                     channels_ordering=kp.ChannelOrdering.KP_CHANNEL_ORDERING_CHW)
    inf_node_output_list.append(inference_float_node_output)

print(inf_node_output_list)

'''
[{
    "width": 7,
    "height": 7,
    "channel": 255,
    "channels_ordering": "ChannelOrdering.KP_CHANNEL_ORDERING_CHW",
    "num_data": 12495,
    "ndarray": [
        "[[[[  1.3589103   0.33972758  0.50959134 ...  0.16986379",
        "     0.33972758 -0.849319  ]",
        "   [  1.698638   -0.50959134  0.50959134 ... -0.16986379",
        "    -0.16986379 -0.849319  ]",
        "   [  1.5287741   0.50959134  0.16986379 ...  0.",
        "    -0.33972758 -0.50959134]",
        "   ...",
        "   [  1.3589103   1.5287741  -1.0191827  ...  2.547957",
        "    -1.1890465  -0.67945516]",
        "   [  1.3589103   1.8685017  -1.0191827  ...  1.0191827",
        "    -0.67945516 -0.33972758]",
        "   [  0.849319    0.849319   -0.16986379 ...  0.16986379",
        "     0.16986379 -0.33972758]]",
        "",
        "  ...",
        "",
        "  [[ -8.49319    -9.342508  -10.021964   ... -10.191828",
        "    -9.002781   -7.6438704 ]",
        "   [ -7.983598  -10.021964   -9.852099   ...  -9.172645",
        "    -8.323326   -6.6246877 ]",
        "   [ -7.983598  -10.191828  -10.021964   ...  -8.153461",
        "    -7.983598   -7.304143  ]",
```

```
        "   ...",
        "   [ -7.6438704   -9.682236   -10.361691   ...  -9.002781",
        "    -10.021964    -9.172645  ]",
        "   [ -7.304143   -10.701419   -12.400057   ...  -9.682236",
        "     -9.682236    -9.002781  ]",
        "   [ -6.1150966   -8.153461    -9.342508   ...  -7.983598",
        "     -8.153461    -8.49319   ]]]]"
    ]
}, {
    "width": 14,
    "height": 14,
    "channel": 255,
    "channels_ordering": "ChannelOrdering.KP_CHANNEL_ORDERING_CHW",
    "num_data": 49980,
    "ndarray": [
        "[[[[  0.8736945  -0.3494778  -0.1747389 ...   0.        -0.1747389",
        "     -0.6989556]",
        "   [  0.6989556  -0.8736945  -0.6989556 ...  -0.5242167  -0.1747389",
        "     -0.5242167]",
        "   [  0.5242167  -0.8736945  -0.6989556 ...  -0.1747389   0.1747389",
        "     -0.8736945]",
        "   ...",
        "   [ -1.0484334   0.         0.        ...   0.3494778   0.3494778",
        "      0.1747389]",
        "   [ -0.6989556  -0.5242167  -0.1747389 ...   0.5242167   0.3494778",
        "      0.1747389]",
        "   [  0.1747389   0.         0.1747389 ...   0.5242167   0.3494778",
        "      0.       ]]",
        "",
        "  ...",
        "",
        "  [[ -7.5137725  -9.261162   -9.785378  ... -10.659073  -10.484334",
        "     -8.562206 ]",
        "   [-10.484334  -13.280156  -14.678067  ... -15.202284  -15.202284",
        "     -11.008551 ]",
        "   [-11.18329   -14.678067  -15.202284  ... -16.07598   -15.027545",
        "     -12.231723 ]",
        "   ...",
        "   [-11.18329   -15.377023  -18.172846  ... -13.105417  -12.581201",
```

```
        "    -10.833812 ]",
        "   [-10.134856  -14.153851  -16.774935  ... -10.659073   -9.61064",
        "     -8.387467 ]",
        "   [ -9.086423  -12.231723  -12.930678  ... -10.134856   -9.261162",
        "     -7.3390336]]]]"
    ]
}]
```

```
'''
```

Get Kneron PLUS Version

This tutorial shows how to get Kneron PLUS version.

Get Kneron PLUS version by *kp.core.get_version()*:

```
import kp


print(kp.core.get_version())
```

Scane All Kneron Device

This tutorial shows how to get USB information of Kneron devices.

Import `kp` into your program:

```
import kp
```

Get USB information of Kneron devices by `kp.core.scan_devices()`:

```
device_descriptors = kp.core.scan_devices()
```

Simply show all information:

```
print(device_descriptors)


'''
{
    "0": {
        "usb_port_id": 13,
```

```
        "vendor_id": "0x3231",
        "product_id": "0x100",
        "link_speed": "UsbSpeed.KP_USB_SPEED_HIGH",
        "kn_number": "0xC8062D2C",
        "is_connectable": true,
        "usb_port_path": "1-3",
        "firmware": "KDP2 Loader"
    }
}
'''
```

Connect to Kneron device

This tutorial shows how to connect to Kneron devices and get Kneron device handler (Device Group).

Import `kp` into your program:

```
import kp
```

Get Kneron device USB port IDs for connecting:

Device Group is Kneron device handler, which supports multiple `same product ID` Kneron devices connection ability.

1. Get `USB port ID` of KL520 devices by `kp.core.scan_devices()`

```
2. target_device_type = kp.ProductId.KP_DEVICE_KL520
3. usb_port_ids = []
4.
5. device_descriptors = kp.core.scan_devices()
6.
7. for device_descriptor in device_descriptors.device_descriptor_list:
8.     if target_device_type == device_descriptor.product_id and \
9.        device_descriptor.is_connectable:
10.        usb_port_ids.append(device_descriptor.usb_port_id)
```

11. Connect to Kneron device by `kp.core.connect_devices(usb_port_ids=List[int])`

    - Connect one Kneron device

    ```
    device_group = kp.core.connect_devices(usb_port_ids=[usb_port_ids[0]])
    ```

    - Connect multiple Kneron devices
      Note: Multiple Kneron devices connection ability only supports the **same product ID Kneron devices**. Please check your USB port IDs have the same product ID before `kp.core.connect_devices(usb_port_ids=List[int])`.

    ```
    device_group = kp.core.connect_devices(usb_port_ids=usb_port_ids)
    ```

Upload SCPU/NCPU Firmware

This tutorial shows how to upload SCPU nad NCPU Frimware to Kneron devices in following two ways:

1. Upload firmware by file path
2. Upload firmware by binary data

Note 1: Please connect Kneron device and get `Device Group` before the following tutorial. See the Connect to Kneron device for details.

Note 2: `load_firmware_from_file` and `load_firmware` only support Kneron KL520 USB-Boot firmware. If you want to update Kneron KL520/KL720 Flash-Boot firmware, please see the Upgrade AI Device to KDP2 for details.

Upload firmware by file path

Please replace `SCPU_FW_PATH`, `NCPU_FW_PATH` by kdp2_fw_scpu.bin and kdp2_fw_ncpu.bin path (Please find target device firmware under `res/firmware` folder)

```
SCPU_FW_PATH = 'res/firmware/KL520/kdp2_fw_scpu.bin'
NCPU_FW_PATH = 'res/firmware/KL520/kdp2_fw_ncpu.bin'

kp.core.load_firmware_from_file(device_group=device_group,
                                scpu_fw_path=SCPU_FW_PATH,
                                ncpu_fw_path=NCPU_FW_PATH)
```

Upload firmware by binary data

Please replace `SCPU_FW_PATH`, `NCPU_FW_PATH` by kdp2_fw_scpu.bin and kdp2_fw_ncpu.bin path (Please find target device firmware under `res/firmware` folder)

```
SCPU_FW_PATH = 'res/firmware/KL520/kdp2_fw_scpu.bin'
NCPU_FW_PATH = 'res/firmware/KL520/kdp2_fw_ncpu.bin'

with open(SCPU_FW_PATH, 'rb') as file:
    scpu_fw_buffer = file.read()

with open(NCPU_FW_PATH, 'rb') as file:
    ncpu_fw_buffer = file.read()

kp.core.load_firmware(device_group=device_group,
                      scpu_fw_buffer=scpu_fw_buffer,
                      ncpu_fw_buffer=ncpu_fw_buffer)
```

Load NEF Model

This tutorial shows how to load NEF model to Kneron devices in the following three ways:
1. Upload NEF model by file path
2. Upload NEF model by binary data
3. Load NEF model from device flash

Note: Please upload firmware on Kneron device before the following tutorial. See the Upload SCPU/NCPU Firmware for details.

Upload NEF model by file path

Please replace `MODEL_FILE_PATH` by models_520.nef path (Please find target device NEF model under `res/models` folder)

```
MODEL_FILE_PATH = 'res/models/KL520/tiny_yolo_v3/models_520.nef'


model_nef_descriptor = kp.core.load_model_from_file(device_group=device_group,
                                                    file_path=MODEL_FILE_PATH)
```

Upload NEF model by binary data

Please replace `MODEL_FILE_PATH` by models_520.nef path (Please find target device NEF model under `res/models` folder)

```
MODEL_FILE_PATH = 'res/models/KL520/tiny_yolo_v3/models_520.nef'


with open(MODEL_FILE_PATH, 'rb') as file:
    nef_buffer = file.read()


model_nef_descriptor = kp.core.load_model(device_group=device_group,
                                          nef_buffer=nef_buffer)
```

Load NEF model from device flash

Please update NEF model in device flash by **`Kneron DFUT`** before `load_model_from_flash`. Reference chapter **`Write Model To Flash`** for more information.

```
model_nef_descriptor = kp.core.load_model_from_flash(device_group=device_group)
```

Simply show **`ModelNefDescriptor`** information:

```
print(model_nef_descriptor)
```

```
'''
{
    "crc": "0x6CBF1FF9",
    "num_models": 1,
    "models": {
        "0": {
            "id": 19,
            "max_raw_out_size": 85752,
            "width": 224,
            "height": 224,
            "channel": 3,
            "img_format": "ImageFormat.KP_IMAGE_FORMAT_RGBA8888"
        }
    }
}
'''
```

Get System Information on Kneron Device

This tutorial shows how to get system information from Kneron devices.

Note: Please upload firmware on Kneron device before the following tutorial. See the Upload SCPU/NCPU Firmware for details.

Get system information of Kneron devices by *kp.core.get_system_info(device_group: DeviceGroup, usb_port_id: int)*:

```
system_infos = []

for usb_port_id in usb_port_ids:
    system_info = kp.core.get_system_info(device_group=device_group,
                                          usb_port_id=usb_port_id)
    system_infos.append(system_info)
```

Simply show all information:

```
print(system_infos)

'''
[{
    "kn_number": "0xC8062D2C",
    "firmware_version": "1.5.0-build.113"
```

```
}]
'''
```

Get Model Information on Kneron Device

This tutorial shows how to get model information from Kneron devices. NEF model is Kneron provided model format, which can combine multiple models in one NEF file. You can use `kp.core.get_model_info(device_group: DeviceGroup, usb_port_id: int)` to check models contain in uploaded NEF file.

Note: Please upload NEF model on Kneron device before the following tutorial. See the Load NEF Model for details.

Get model information of Kneron devices by *kp.core.get_model_info(device_group: DeviceGroup, usb_port_id: int)*:

```
model_nef_descriptors = []


for usb_port_id in usb_port_ids:
    model_nef_descriptor = kp.core.get_model_info(device_group=device_group,
                                                  usb_port_id=usb_port_id)
    model_nef_descriptors.append(model_nef_descriptor)
```

Simply show all information:

```
print(model_nef_descriptors)


'''
[{
    "crc": "0x6CBF1FF9",
    "num_models": 1,
    "models": {
        "0": {
            "id": 19,
            "max_raw_out_size": 85752,
            "width": 224,
            "height": 224,
            "channel": 3,
            "img_format": "ImageFormat.KP_IMAGE_FORMAT_RGBA8888"
        }
    }
}]
```

```
'''
```

KL520 SDK Introduction

note: KL520 SDK v1.7.x is compatible with Kneron PLUS v1.3.x

1. Requirements

Hardware:

Board with KL520 chip, like 520 dongle, 96board, m.2 board

(For applications with camera) Kneron KL520 series AI SoC Development Kit

Software:

licensed software: ARM Keil MDK

ARM Keil/MDK docs

2. File Structure

The whole SDK package is composed of device firmware, the folder design is described below

```
.
├── common                          # common interface between SCPU/NCPU
├── example_projects
|   ├── kdp2_companion_user_ex         # Keil project for Kneron PLUS user example
|   ├── tiny_yolo_v3_host              # Keil project for host mode(Standalone) firmware exa
mple
|   └── tiny_yolo_v3_host_usbout       # Keil project for host mode with outputing result v
is usb
├── platform
|   └── kl520
|       ├── common                   # part of common interface between SCPU/NCPU
|       ├── ncpu
|       |   ├── drv                  # drivers
|       |   ├── rtos/rtx              # cloned rtos/rtx code
|       |   └── startup               # startup assembly code, and FW init code
|       └── scpu
|           ├── drv                  # peripheral drivers
|           ├── rtos/rtx              # cloned rtos/rtx code
|           └── startup               # startup assembly code, and FW init code
├── ncpu
|   ├── device                       # device configurations
|   ├── lib                          # folder for libraries
|   └── project
|       └── tiny_yolo_v3              # ncpu project
├── scpu
|   ├── board                        # for device board configurations
|   ├── config                       # for device board configurations
```

```
|   ├── device                      # device memory address configurations
|   ├── drivers                     # system drivers
|   ├── framework                    # framework layer code
|   ├── kdev                        # device driver code
|   ├── kmdw                        # middleware
|   ├── lib                         # folder for libraries
|   └── project
|       └── tiny_yolo_v3
|           ├── host                # Keil project for host mode firmware example
|           └── host_usbout          # Keil project for host mode with outputing result vis
 usb
├── ncpu_kdp2
|   ├── lib_app                     # folder for kdp2-ncpu-app.lib
|   ├── lib_sdk                     # folder for kdp2-ncpu-sdk.lib
|   └── project
|       └── ncpu_companion_user_ex     # Keil project for Kneron PLUS user example
├── scpu_kdp2
|   ├── app                         # application layer code for Kneron PLUS firmware examp
le
|   ├── lib_sdk                     # kdp2_scpu_sdk.lib folder
|   └── project
|       └── scpu_companion_user_ex     # Keil project for Kneron PLUS example
├── models
|   └── tiny_yolo_v3                 # model file for demo
├── sdkexamples                      # driver examples
└── utils                           # firmware/model utilities
```

3. SoC Peripheral Drivers

KL520 also provides some simple examples to show how to use basic peripherals such as, I2C, PWM, DMA, GPIO... User can find them from `sdkexamples` folder.

There is also a PDF file to briefly describe the peripheral APIs. Please download it from the following link: KL520_Peripheral_Driver_APIs.pdf

Supported/Unsupported Peripheral Table

Image Input

Peripherals	Companion	Host Mode
MIPI CSI RX	x	O
DVP	x	driver/example
UVC Host	x	specified cameras

Peripherals	Companion	Host Mode
USB(proprietary)	O	x
SPI Master, non-DMA	x	driver/example
SPI Slave, non-DMA	x	driver/example
SPI Master, DMA	x	x
SPI Slave, DMA	x	x
UART	x	x

Image/Result Output

Peripherals	Companion	Host Mode
MIPI DSI TX	x	x
MIPI CSI TX	x	x
DVP	x	O
UVC device	x	x
USB bulk	x	O
USB(proprietary)	O	x
SPI Master, non-DMA	x	driver/example
SPI Slave, non-DMA	x	driver/example
SPI Master, DMA	x	x
SPI Slave, DMA	x	x
UART	x	O
I2C	x	driver/example
I2S	x	x

Peripherals	Companion	Host Mode
INTEL 8080	x	x

Appendix

host mode example and **host mode with USB output example** are shown

KL720 SDK Introduction

note: KL720 SDK v1.5.x is compatible with Kneron PLUS v1.3.x

1. Requirements

Hardware:

Board with KL720 chip, like 720 dongle, 96board, m.2 board.

(for MIPI application example) Kneron LW 3D module

(for ToF application example) ToF_ISR module

Software:

licensed software: ARM Keil MDK ARM Keil/MDK docs

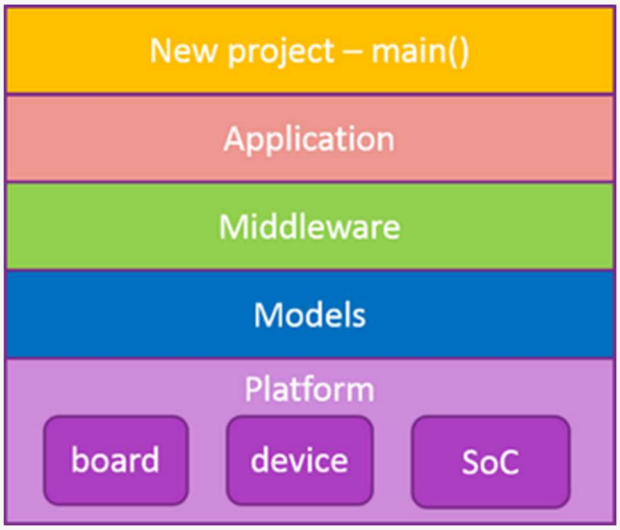
licensed software: Cadence Tensilica Xtensa SDK

2. File Structure

The whole SDK package is composed of device firmware, the folder design is described below.

2.1 Basic Concept

The basic concept of FW folder structure is modularize and stratification for all source code. FW code belonged to same feature will be put to one dedicated folder and easy to include/exclude it. Refer to basic FW architecture shown below, the listed items will have corresponding folders.



Here is the example folder design for Kneron SDK.

```
└─────firmware
    ├────app
    ├────build
    │    └─────solution_**
    ├────include
    ├────mdw
    ├────platform
    │    ├────board
    │    ├────dev
    │    └─────kl720
```

```
│       ├──common
│       └──scpu
│           ├──drv
│           ├──rtos/rtx
│           └──startup
└──utils
    ├──bin_gen
    ├──dfu
    ├──flash_programmer
    ├──JLink_programmer
    └──spl_aes
```

2.2 Detailed explanation

firmware: Contains all device FW source/lib code, utilities, build environment

```
└──firmware
    ├──app
    ├──build
    ├──include
    ├──mdw
    ├──platform
    └──utils
```

Basically, the firmware source_code=app+mdw+platform(+include). We hope all firmware source code will be put under these 3 folders and will not be influenced by any projects, that is, it's a source code data base. All project related code will be put under build folder.

app: All application firmware code. Every module or C file have prefix `kapp_`.

build: Build environment. Include (Keil) project files, workspace, main.c, makefiles. C source files will be pulled in a project and then engineer generates a new project.

include: C header files for all source code

mdw: Middleware. It's kind of "service", "manager". We can put some useful and special purpose pure software feature here, such as file system, software timer, DFU function, memory management.

platform: It means a HW platform or a SoC for AI development. Platform consists of an SoC, a PCB, and some onboard devices(flash, eeprom...).

utils: some useful utilities, such as flash programming, calculate checksum...

```
└──firmware
    └──build
        ├──example_**
        ├──lib
        └──solution_**
```

There are two major components in build folder, example projects and solution projects. As the name implies, small app demo, simple peripheral drivers demo, or any features demonstration belong to example projects. The purpose is to show how to use our SDK. Solution projects is a solution for customer. It contains more features, or complex functions in a single project. Example projects will have example_ prefix and solution projects will have solution_ prefix. If you need to build a library and share with other projects, create lib projects in lib folder.

example_ : a prefix for example project. ex. `example_i2c` , `example_tiny_yolo`

solution_ : a prefix for solution project. ex. `solution_kdp2`

lib: Some source files need to be hidden or need to generate library. Put the library project here

```
└────firmware
    └────mdw
        ├──console
        ├──dfu
        ├──errand
        ├──flash
... ...
```

Collect independent modules to become middle ware here. It can be generic flash driver, firmware upgrade manager, file system, etc.

```
└────firmware
    └────platform
        ├──board
        ├──dev
        │   ├──eeprom
        │   ├──nand
        │   ├──nor
        │   └──wifi
        └────kl720
            ├──common
            └────scpu
                ├──drv
                ├──rtos
                └──startup
```

Platform = board + dev + ASIC

board: PCB information, flash size, IO mapping,

dev: device drivers, such as flash driver, eeprom driver, wifi module driver, panel driver, sensor driver

kl720: contain all peripheral drivers, real time OS, startup assembly code, and FW init code.

The whole SDK package is composed of device firmware, the folder design is described in this section.

3. Create New SDK Application

Step by step to create new SDK application, please refer to the section **Kneron PLUS / Customized API**.

4. Secure Boot

Kneron KL720 provide secure protect with AES and SHA.

5. SoC Peripheral Drivers

The peripheral definitions and prototypes for the application programming reference.

6.1 Supported/Unsupported Peripheral Table

Image Input

Peripherals	Companion	HICO
MIPI CSI RX	x	driver/example
DVP	x	driver/example
UVC Host	x	specified cameras
USB(proprietary)	O	x
SPI Master, non-DMA, DMA	x	driver/example
SPI Slave, non-DMA, DMA	x	driver/example
UART	x	x

Image/Result Output

Peripherals	Companion	HICO
MIPI DSI TX	x	x
MIPI CSI TX	x	x
DVP	x	O
UVC device	x	x
USB bulk	x	O
USB(proprietary)	O	x
SPI Master, non-DMA, DMA	x	driver/example
SPI Slave, non-DMA, DMA	x	driver/example
UART	x	O
I2C	x	driver/example

Peripherals	Companion	HICO
I2S	x	x
INTEL 8080	x	x

Peripheral Driver APIs

6. Power Management

Provide functions to allow developers control the power states switching.