Kneron Inc

Document Name: **DFW Boot**

KL520 DFW (UART) BOOT

Kneron Inc

Engineering Design Document

Document Number: D-000x
Document Revision: 0.01
Document Security: Level-2 Confidential
Document Type: Engineering Design
Document Author: Teresa Chen
Last Updated: Dec. 21st, 2020

Author	Teresa Chen
Reviewer	
Circulation	Restricted, for internal use only.
Document Name	DFW Boot
Description	

Detailed History of Changes:

Version	Date	Author	Description of Changes
0.1	12/21/2020	Teresa Chen	Initial Draft

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to descript how to develop/apply the DFW (Device Firmware Write) (via UART) Boot feature for KL520.
You can use UART to download firmware bin file to KL520 internal Memory space and then directly boot up KL520 from internal memory, therefore external flash could be optional.

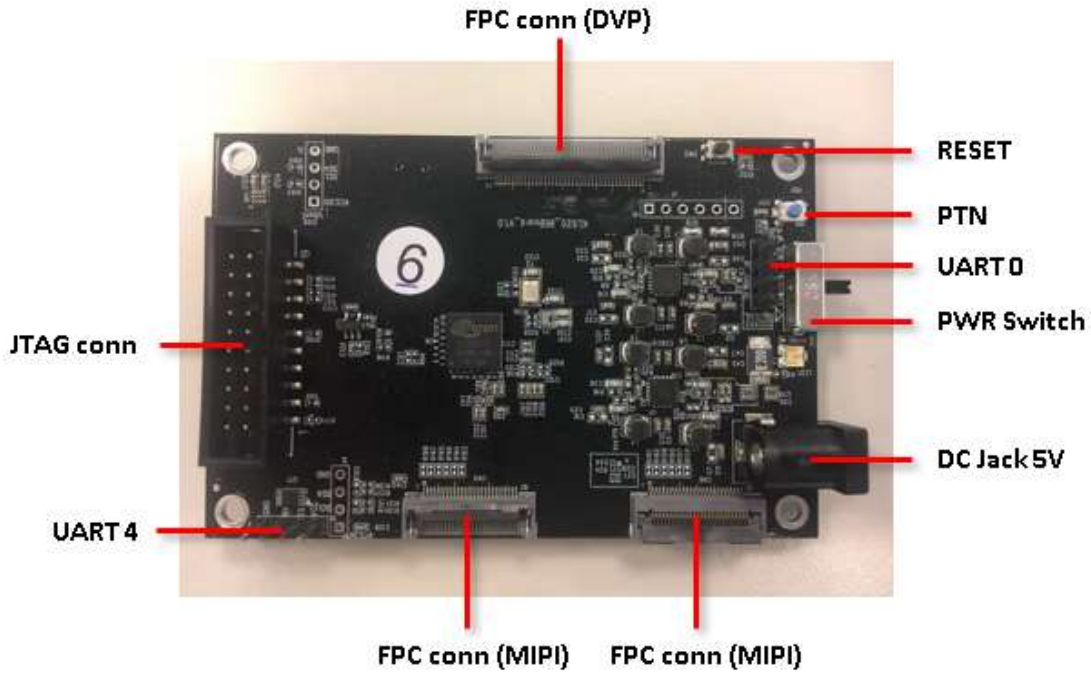
2 Reference

Kneron Mozart Design Specification, Rev. 0.5, Feb. 2019
FTUART010, UART and IrDA Controller, Rev. 1.21, Dec. 2017

3 Acronyms, Abbreviations, Definitions

UART – Universal Asynchronous Receiver-Transmitter

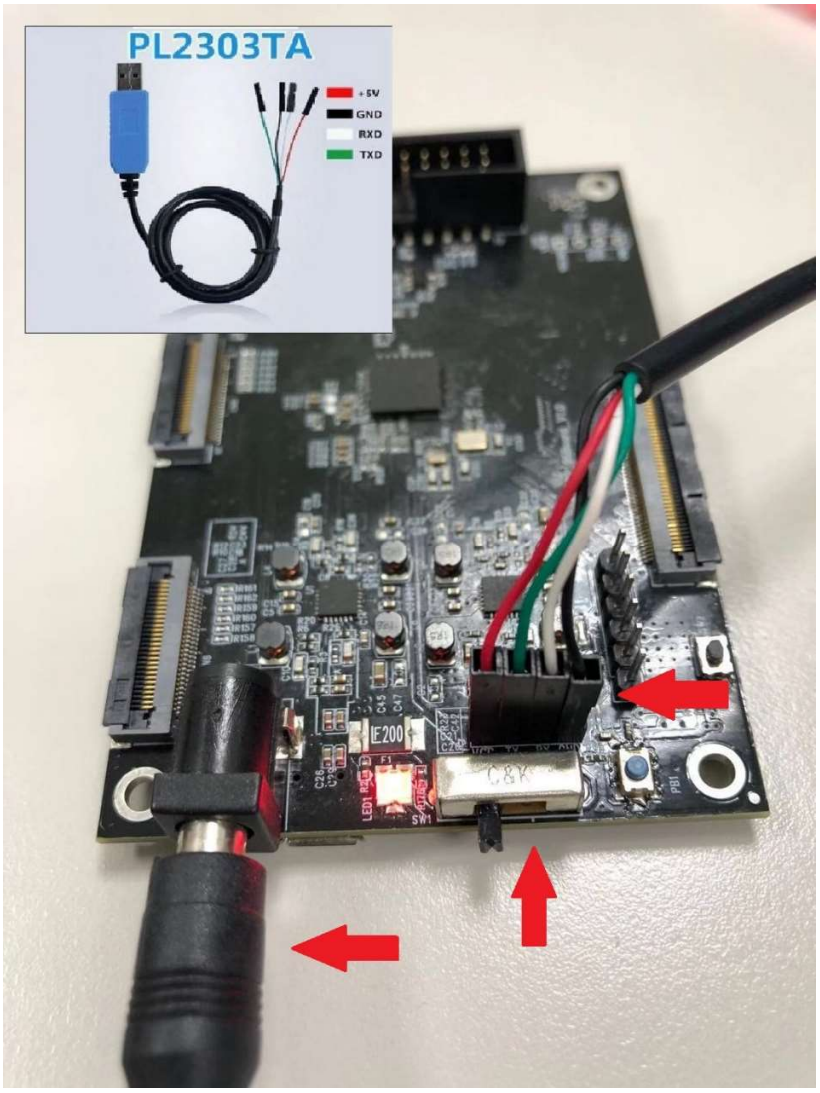
4 Board Overview



5 Hardware Setting

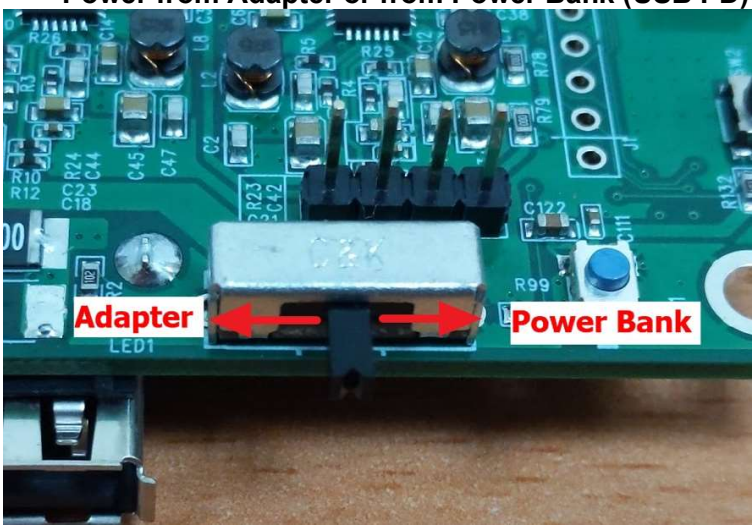
5.1 Connecting UART0

UART0: Command Port



5.2 Connecting 5V Power

5.3 Turn ON power switch.
Power from Adapter or from Power Bank (USB PD)



5.4 Wake up chip from RTC power domain by pressing PTN button (please do it every time after plugging in the power)

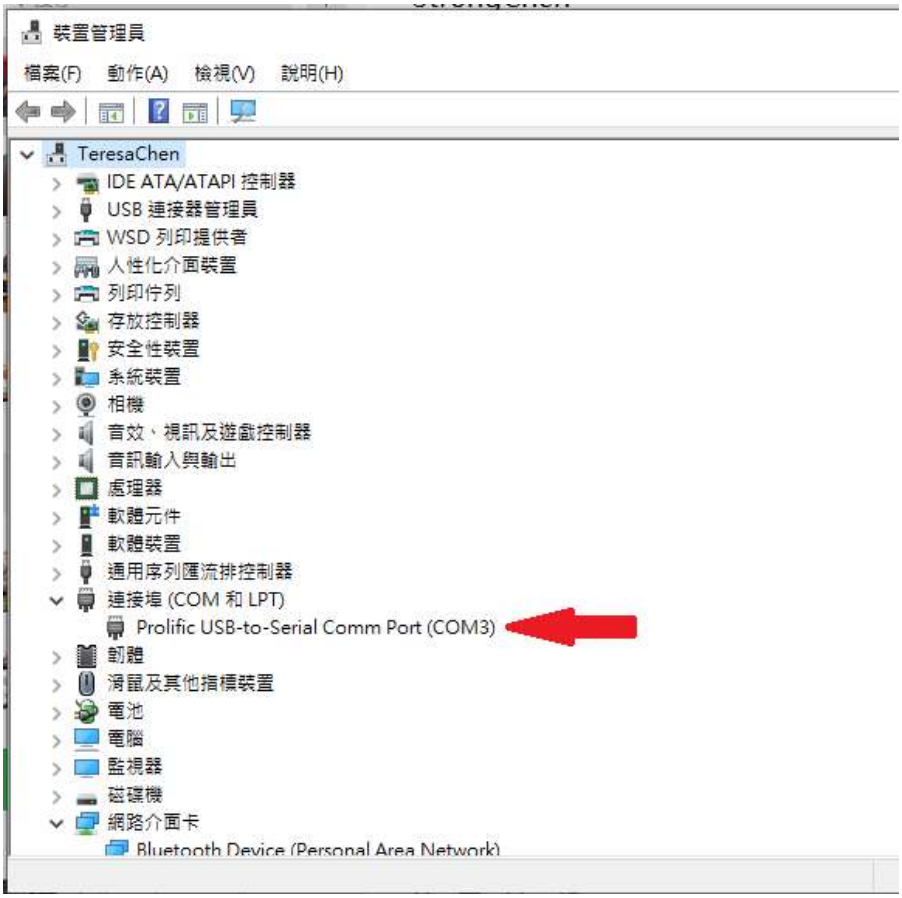
6 DFW via UART0 Interface

6.1 DFW Boot necessaries

1. Open command terminal for DFW Boot execution
   Tool path: kl520_sdk\utils\dfw_boot\uart_dfu_boot.py
2. install Necessary python modules: kl520_sdk\utils\requirements.txt

6.2 Edit python verification setting

1. Check UART port number from device manager
2. Edit setup.py, search “COM_ID” and modify the ID to match your UART port number
   ex: COM_ID = 3 # COM3

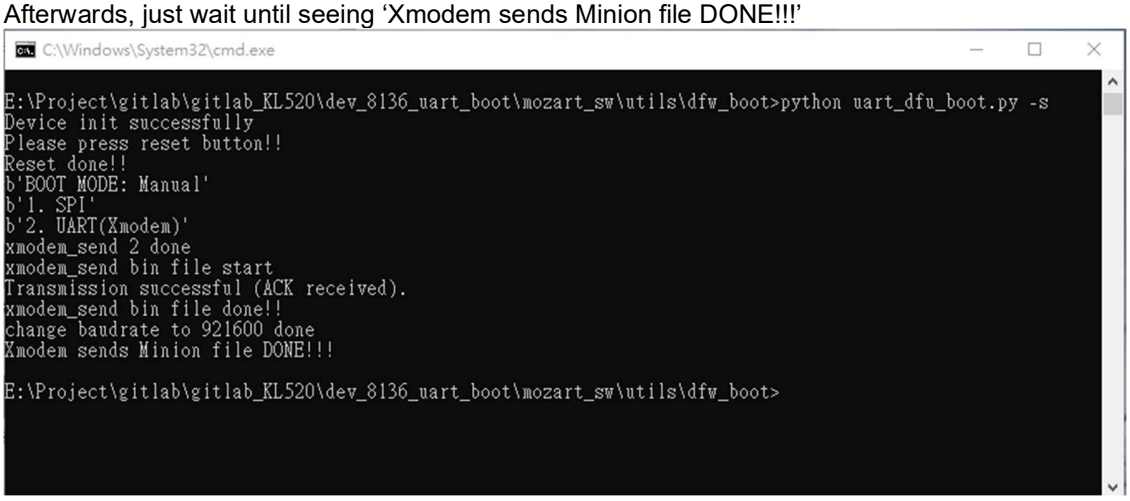


6.3 Chip Initialize and Send/Start Minion FW

```
>> python uart_dfu_boot.py -s
```

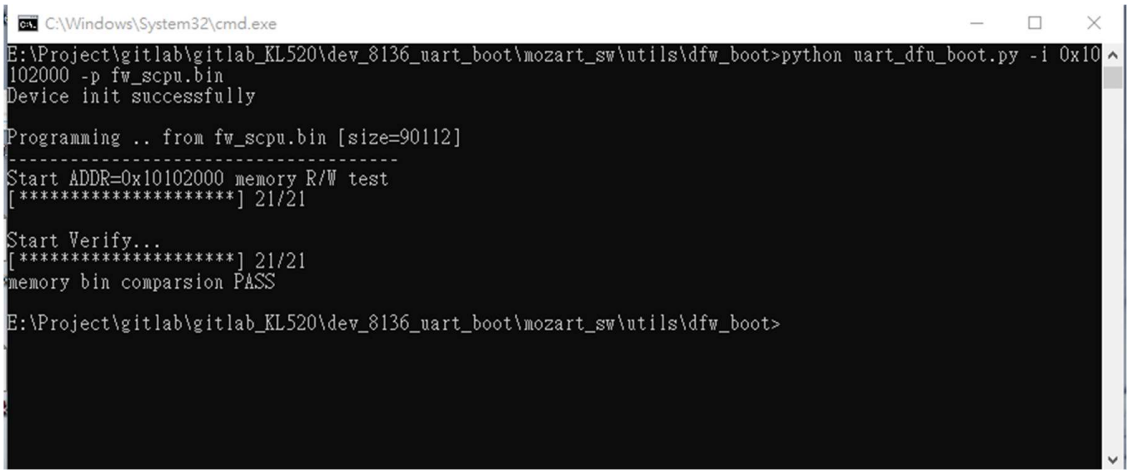
Please press RESET BTN while you are seeing “Please press reset button!!”

Afterwards, just wait until seeing ‘Xmodem sends Minion file DONE!!!’



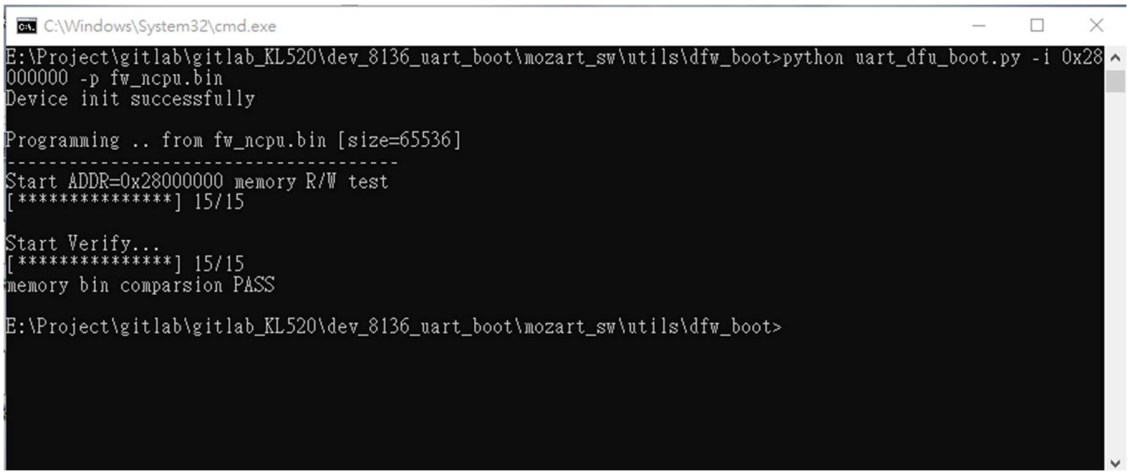
6.4 DFW fw_scpu.bin to memory address 0x10102000

```
>> python uart_dfu_boot.py -i 0x10102000 -p fw_scpu.bin
```



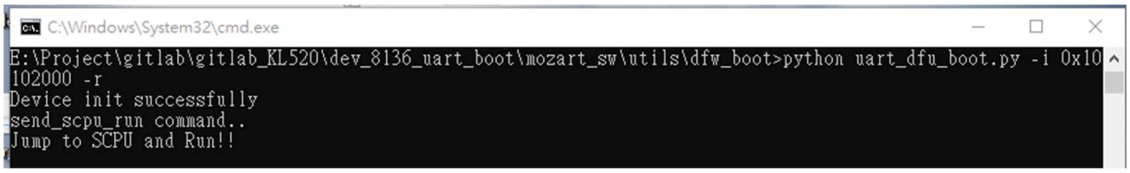
6.5 DFW fw_ncpu.bin to memory address 0x28000000

```
>> python uart_dfu_boot.py -i 0x28000000 -p fw_ncpu.bin
```



6.6 Command to boot up KL520 from memory address 0x10102000

```
>> python uart_dfu_boot.py -i 0x10102000 -r
```

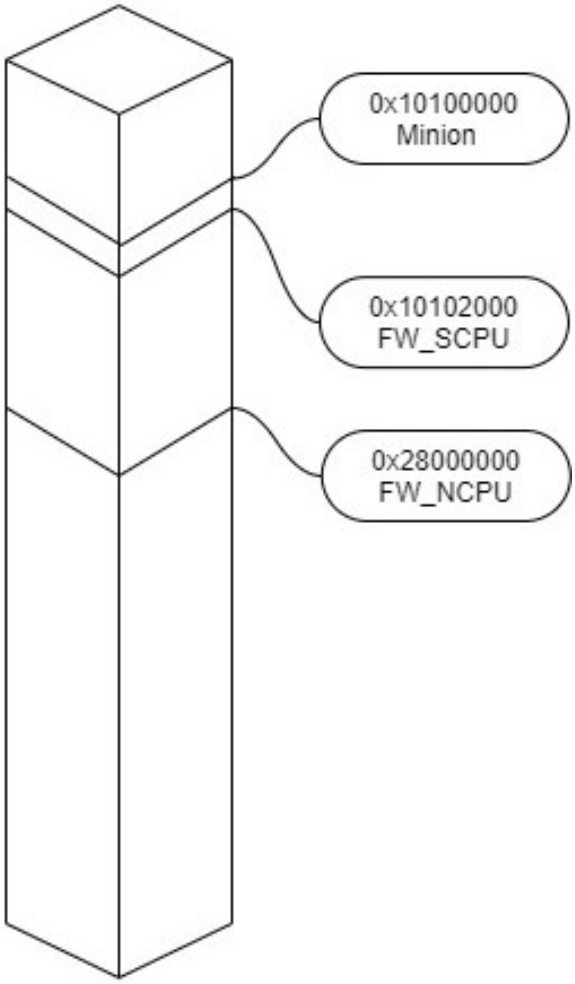


Note: To write specific bin file to specific memory address
“-i" means the memory index/address
“-p" means the FW code you would like to program

7 Design Principle

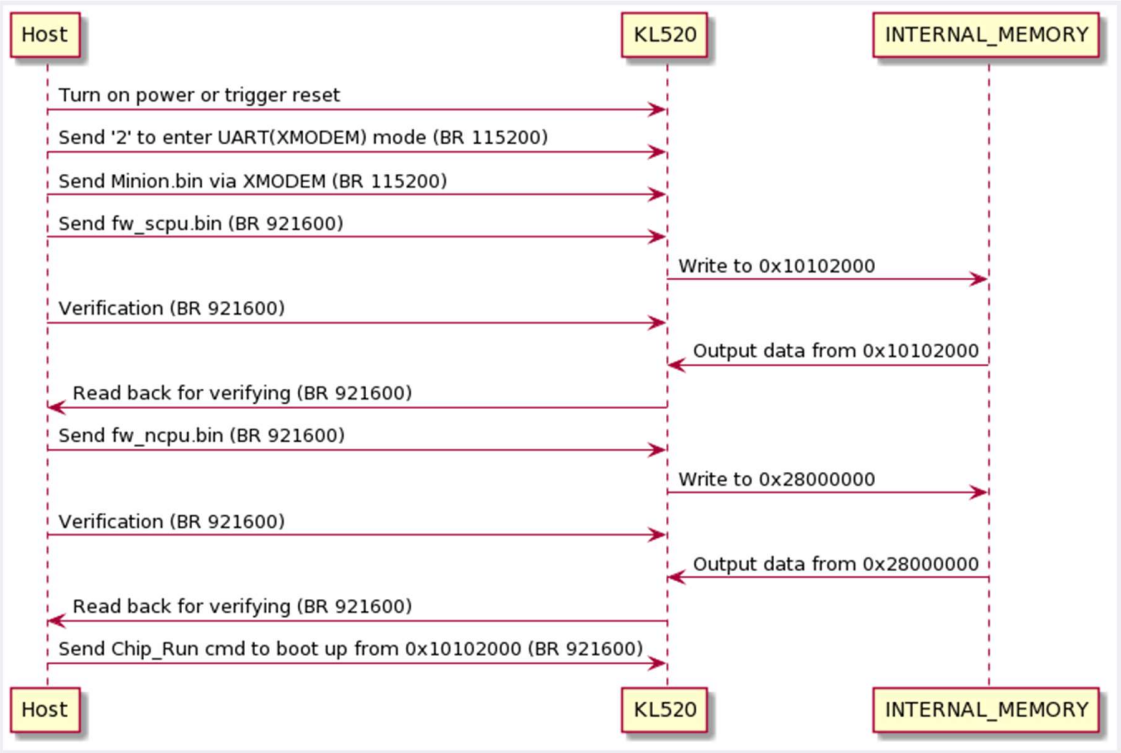
7.1 Memory Space Address

1. Minion FW starts from 0x10100000 and size must < 0x2000 bytes
2. SCPU FW starts from 0x10102000
3. NCPU FW starts from 0x28000000



7.2 Workflow

1. Send Minion FW through UART0/XMODEM and also process the followings sequence through UART0 interface
2. Minion is the bridge to handle host command, collect bin data and then save it into internal memory space



7.3 Host control principle

1. Initial UART COM port with baud rate 115200 and send '2' to tell KL520 to enter 2.UART(XMODEM) mode.
2. Send Minion.bin through UART by XMODEM protocol.
   Reference: https://pythonhosted.org/xmodem/xmodem.html
3. Switch to baud rate 921600.
4. **Send SCPU FW to address 0x10102000**.
   Buf[n] = Msg_header + data_buf[4096]
   - Msg_header:
     typedef struct {
       uint16_t preamble;
       uint16_t crc16;
       uint32_t cmd;
       uint32_t addr;
       uint32_t len;
     } __attribute__((packed)) MsgHdr;
   - data_buf[4096]: transfer bin by unit 4096 bytes(max.) each time
   - Packet TX Preamble: PKTX_PAMB = 0xA583
   - crc16: calculate checksum from buf[4] to buf[n-1] except preamble and crc16
   - cmd for mem_write: 0x1004
   - cmd for mem_read: 0x1003 (read back for verification)
   - addr: 0x10102000
   - len: 4096 or remainder

5. **Send NCPU FW to address 0x28000000**.
   Buf[n] = Msg_header + data_buf[4096]
   - Msg_header:
     typedef struct {

```
    uint16_t preamble;
    uint16_t crc16;
    uint32_t cmd;
    uint32_t addr;
    uint32_t len;
  } __attribute__((packed)) MsgHdr;
```

- data_buf[4096]: transfer bin by unit 4096 bytes(max.) each time
- Packet TX Preamble: PKTX_PAMB = 0xA583
- crc16: calculate checksum from buf[4] to buf[n-1] except preamble and crc16
- cmd for mem_write: 0x1004
- cmd for mem_read: 0x1003 (read back for verification)
- addr: 0x28000000
- len: 4096 or remainder

6. **Send CHIP_RUN command with address set as 0x10102000**.
   Buf[n] = Msg_header
   typedef struct {
   ```
       uint16_t preamble;
       uint16_t crc16;
       uint32_t cmd;
       uint32_t addr;
       uint32_t len;
     } __attribute__((packed)) MsgHdr;
   ```
   - Packet TX Preamble: PKTX_PAMB = 0xA583
   - crc16: calculate checksum from buf[4] to buf[n-1] except preamble and crc16
   - cmd for scup_run: 0x1005
   - addr: 0x10102000
   - len: 0

You can refer to /utils/dfw_boot/uart_dfw_boot.py and setup.py as example to develop code for host controls.