Arm® Cortex®-A35
64/32-bit Microcontroller

NuMicro® Family
MA35D1 NuWriter
User Manual

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 OVERVIEW

NuWriter is a programing tool for the MA35D1 series microprocessor. It communicates with MA35D1 through USB interface, and allowing user to issue commands from computer to program storage devices connected with MA35D1 and the OTP within MA35D1. For external storage device, it supports NAND flash, SPI NAND flash, SPI NOR flash, eMMC, and SD cards.

NuWriter tool for MA35D1 came with two different modes. One is the GUI mode, this is a user friendly mode, with an easily to use front end allowing user to control with simple mouse click. The other one is command line mode, this mode is suitable for combine with shell script for automation control.

NuWriter is a cross platform tool developed with Python language. So it can be use on both Windows and Linux platform. User can use this tool on whichever operating system they are familiar with.

2 INSTALLATION

NuWriter is released in both source format and executable form. The executable file can execute directly without installation. But to execute from the python script, some required packages are described below.

2.1 Install Python3

NuWriter is developed with Python3. Theirs is no guarantee to work with Python2. Please install python 3 in order to execute NuWriter source code. In an Ubuntu system, use the command below to install Python3.

```
$ sudo apt install python3 python3-pip
```

For Windows user, the Python3 package could be download from Python's official web site https://www.python.org/downloads/windows/.

2.2 Python Modules

After install Python3, please use following command to install required modules for NuWriter command line tool.

```
$ pip3 install pyusb usb crypto ecdsa crcmod tqdm pycryptodome
```

In addition to the modules above, to use GUI mode, the PyQt5 module is required.

```
$ pip3 install PyQt5
```

2.3 libusb

In Windows platform, it is required to install libusb manually. Download the library from http://sourceforge.net/projects/libusb/files/libusb-1.0/libusb-1.0.20/libusb-1.0.20.7z/download, and extract the download file. Copy MS64\dll\libusb-1.0.dll to C:\Windows\System32 and Copy MS64\dll\libusb-1.0.lib to C:\Users\<user name>\AppData\Local\Programs\Python\<python ver>\Lib.

2.4 Install USB Driver

NuWriter must install the USB VCOM driver on the computer to use the NuWriter tool. To install the USB VCOM driver, please follow the steps below:

Step 1: After connecting the computer and the MA35D1 through USB cable, after turning on the power of the MA35D1, Windows will find a new USB device, and select the executable file *WinUSB4NuVCOM.exe* on the computer to start installing the driver. (Figure 2-1)

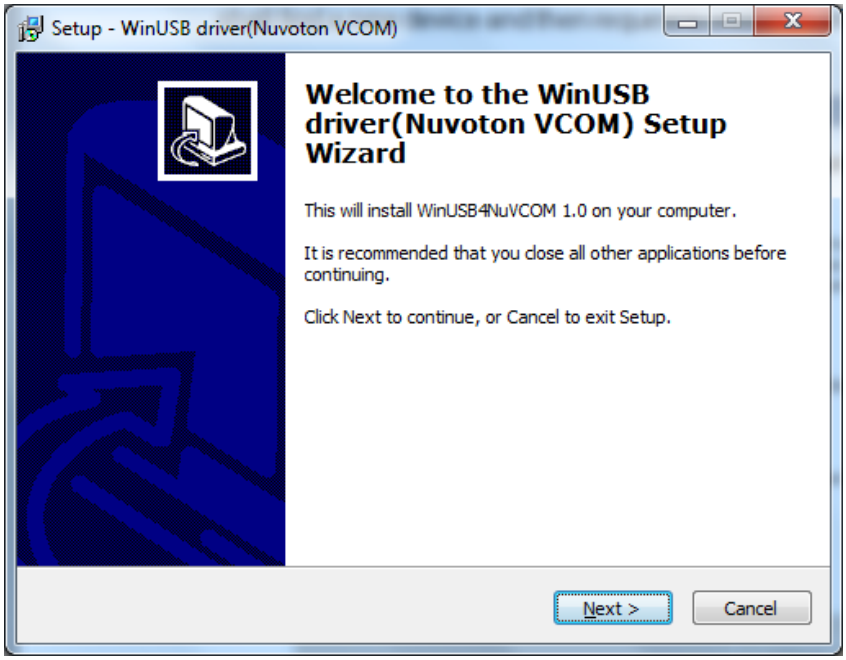

Figure 2-1 WinUSB4NuVCOM Driver Setup (1)

Step 2: Press "Next". This screen tells you that the WinUSB4NuVCOM 1.0 driver is about to be installed (Figure 2-2).

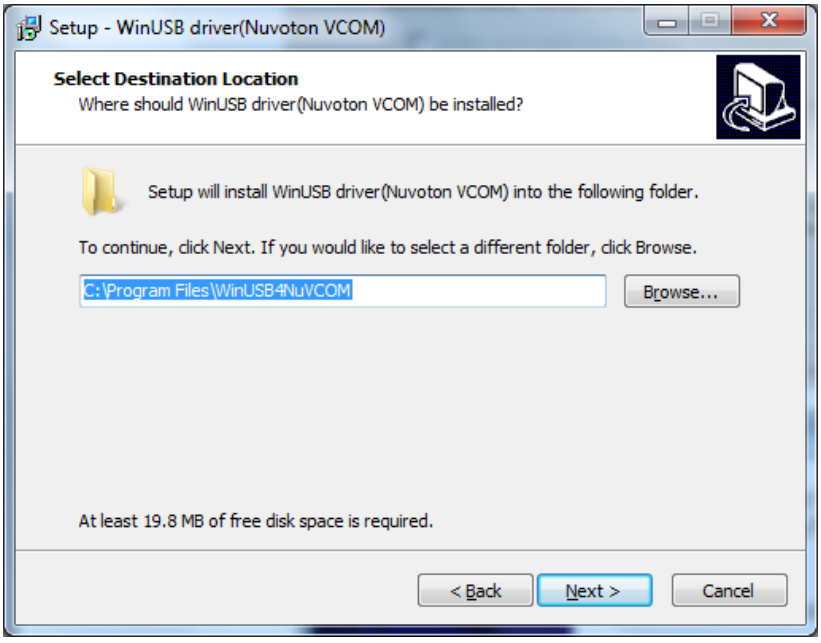

Figure 2-2 WinUSB4NuVCOM Driver Setup (2)

Step 3: Select the path the user wants to install or use the default path, and press “Next” after confirming (Figure 2-3).

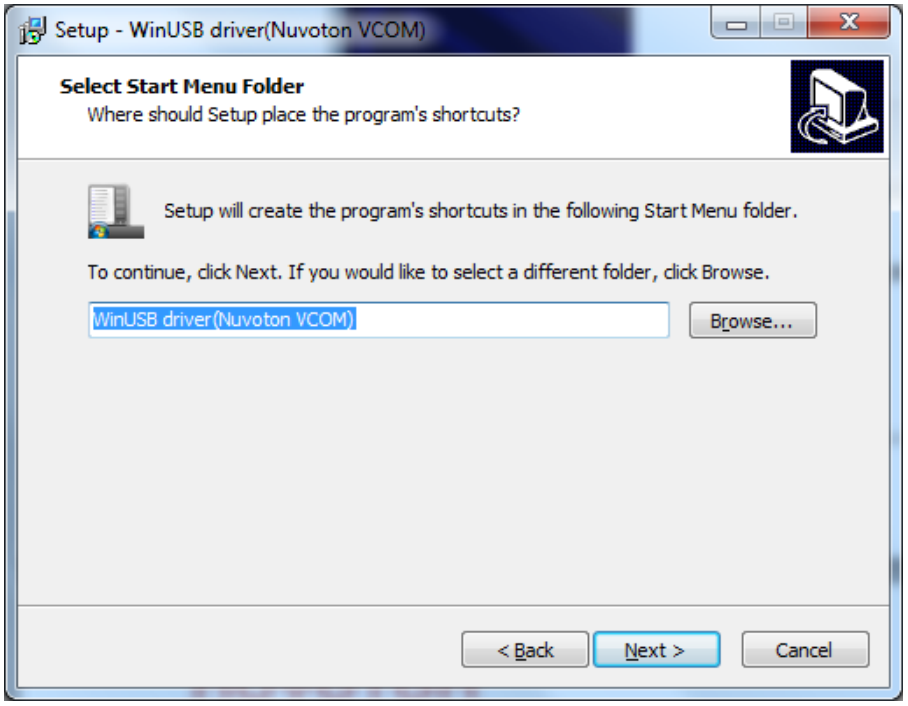

Figure 2-3 WinUSB4NuVCOM Driver Setup (3)

Step 4: Press “Next” (Figure 2-4).

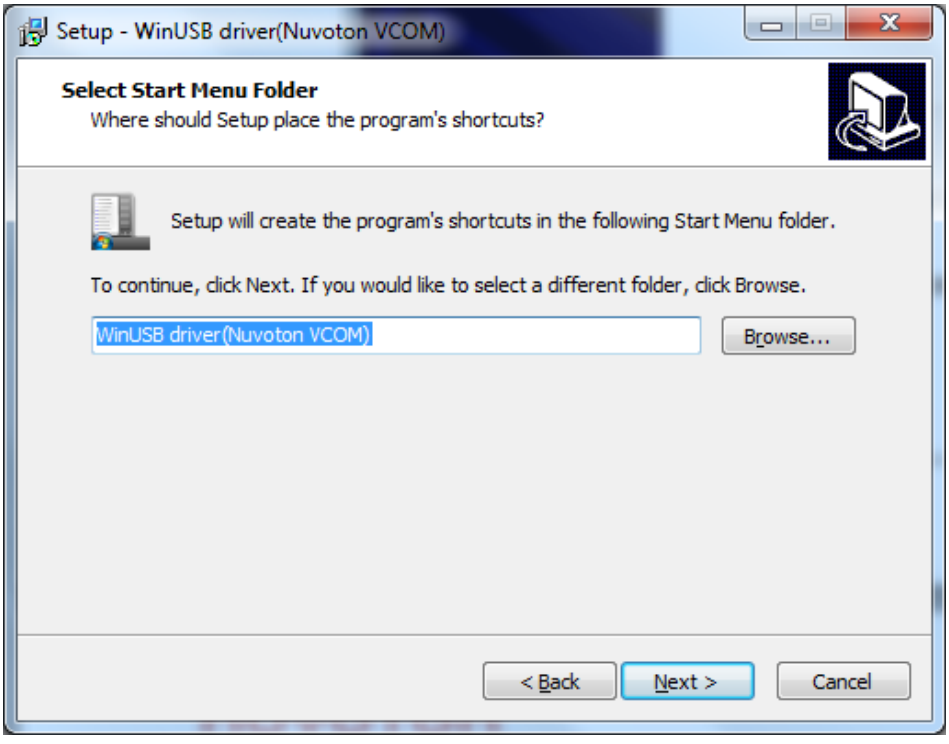

Figure 2-4 WinUSB4NuVCOM Driver Setup (4)

Step 5: Press “Install” (Figure 2-5Figure 2-3).

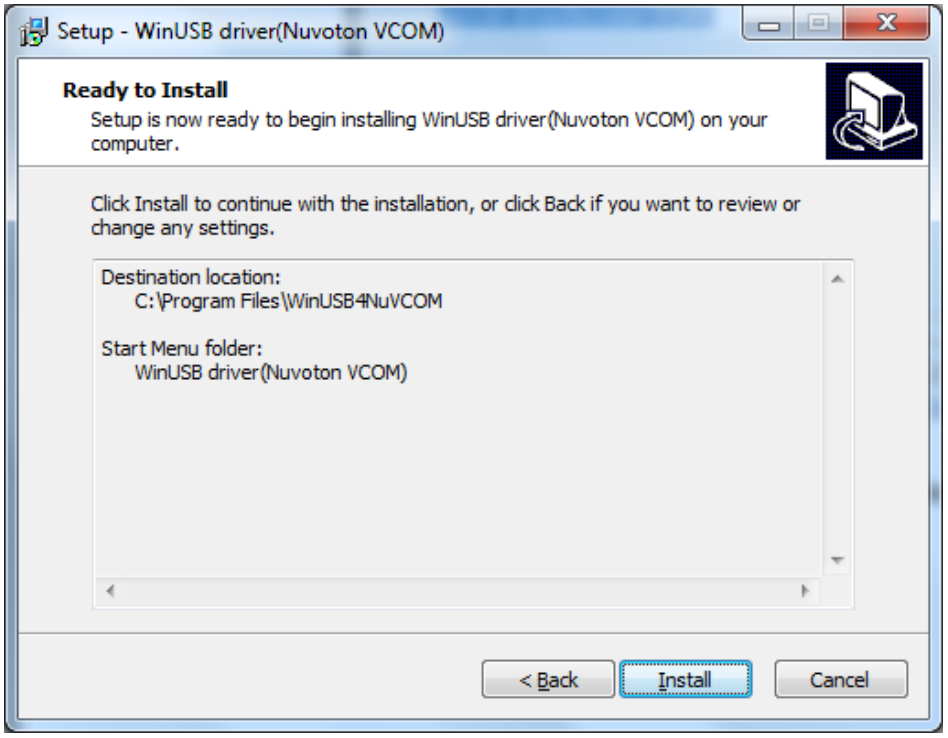

Figure 2-5 WinUSB4NuVCOM Driver Setup (5)

Step 6: Press “Finish” to complete the installation of the USB VCOM driver (Figure 2-6Figure 2-3).

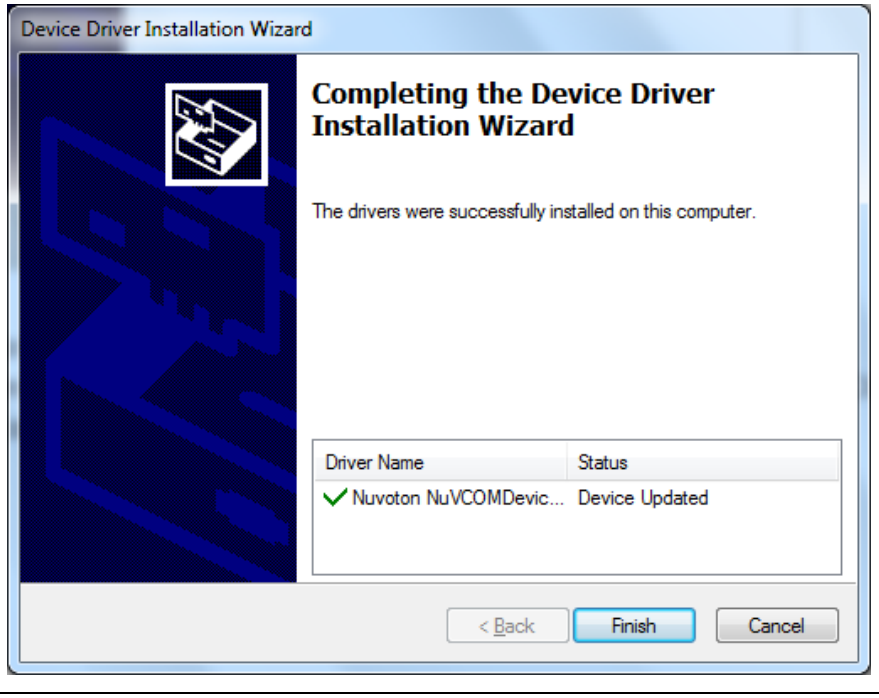

Figure 2-6 WinUSB4NuVCOM Driver Setup (6)

After the installation of the USB VCOM driver is successfully completed, the user can confirm that the Windows operating system should detect the new device and automatically load its corresponding USB settings. Users can see “WinUSB driver (Nuvoton VCOM)” in Device Manager, which means the

driver is installed successfully (Figure 2-7).

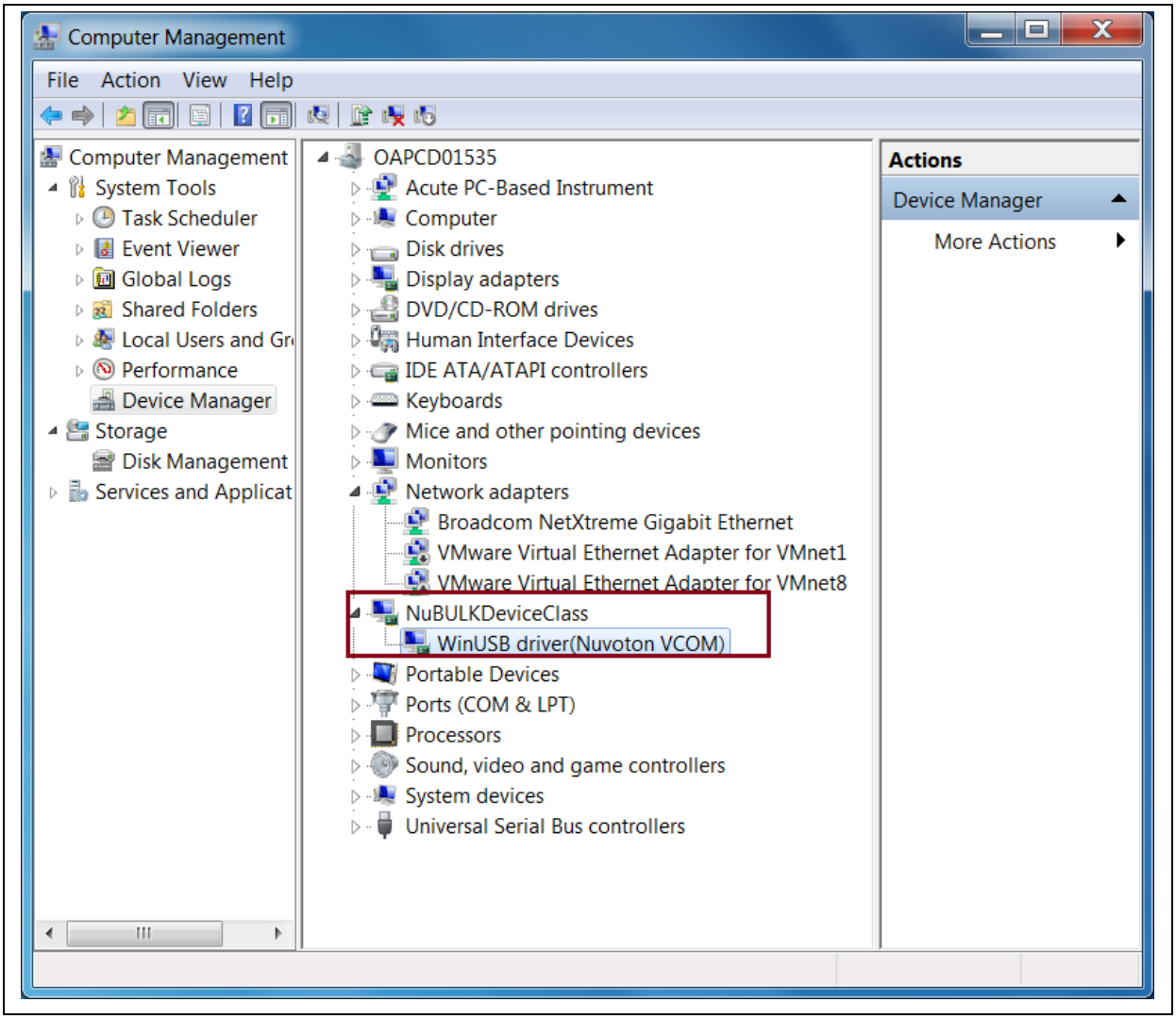

Figure 2-7 NuWriter VCOM Device

3 HARDWARE CONFIGURATION

3.1 Hardware Connection

The MA35D1 IBR is communicate with NuWriter through USB port 0. A USB cable should connect between MA35D1 and computer allowing the NuWriter tool to control MA35D1. The UART cable connects between PC and MA35D1 UART0 is used to shown debug message printed by NuWriter firmware it takes no role in the data transfer between MA35D1 and PC. Figure 3-1 shows the USB device and UART interface on NuMaker-HMI-MA35D1 board.

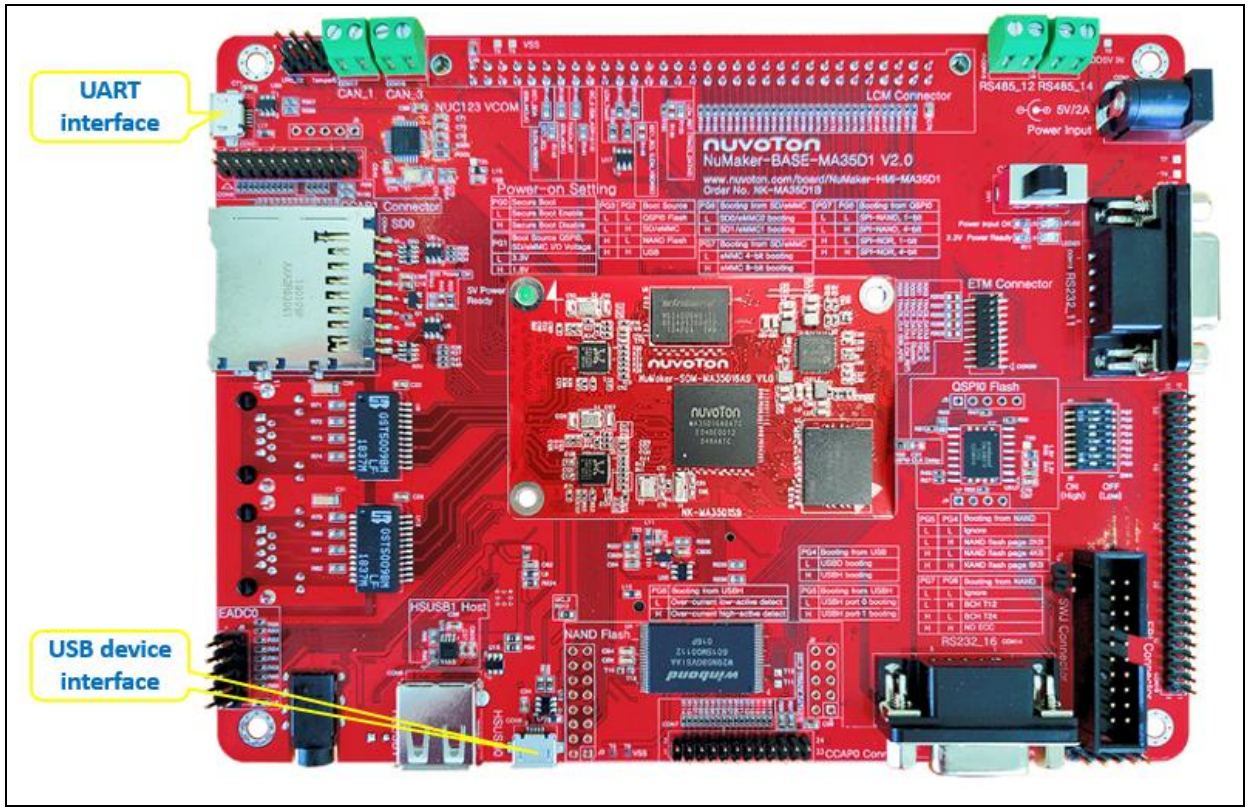

Figure 3-1 Board Connection

3.2 Power-On-Setting

NuWriter uses Power-On-Setting PG[7:4] to determine the attribute of storage medias. For example PG[7:6] is used to select SD interface to access and also decide ECC while accessing NAND flash. So the Power-On-Setting on PG[7:4] must configured according the storage media to be accessed. For example, to program SD1, PG[7:6] should set 01 regardless of the ECC setting to the NAND flash on board if any.

Pin	Description
PG[0]	Secure Boot Disable Bit 0 = Secure Boot Enabled. 1 = Secure Boot Disabled.
PG[3:2]	Boot Source Selection 00 = Boot from SPI Flash. 01 = Boot from SD/eMMC. 10 = Boot from NAND Flash. 11 = Boot from USB.

PG[5:4]	NAND Flash Page Size Selection 00 = Ignore. 01 = NAND Flash page size is 2 KB. 10 = NAND Flash page size is 4 KB. 11 = NAND Flash page size is 8 KB.
PG[7:6]	Miscellaneous Configuration If BTSRCSEL = 01, Boot from SD/eMMC. 00 = SD0/eMMC0 4-bit mode booting. 01 = SD1/eMMC1 4-bit mode booting. 10 = eMMC0 8-bit mode booting. 11 = eMMC1 8-bit mode booting. If BTSRCSEL = 10, the Boot from NAND Flash. 00 = Ignore. 01 = ECC is BCH T12. 10 = ECC is BCH T24. 11 = No ECC. If BTSRCSEL = 00, the Boot from SPI Flash. 00 = SPI-NAND Flash with 1-bit mode booting. 10 = SPI-NOR Flash with 1-bit mode booting. Others reserved.

Table 3-1 Power on Setting

4 BOOT INFORMATION FORMAT

IBR identify the boot image through the boot information programmed in the storage media. This chapter introduces the boot information structure.

4.1 Boot Information

Below is the boot info structure format and explanation of each elements MA35D1's IBR uses.

```
typedef struct boot_info
{
```

A marker which should be string value "NVT".

```
    unsigned int    bootmarker;
```

The CRC32 checksum of boot_info structure except marker.

```
    unsigned int    checksum;
```

Total header length.

```
    unsigned int    length;
```

Boot information version number.

```
    unsigned int    version;
```

A structure holds the SPI flash information. Please check 4.2 for more information.

```
    SPI_INFO_T      spiinfo;
```

The address IBR branch to after images are loaded and verified correctly

```
    unsigned int    entrypoint;
```

Total image count. Maximum value is 4.

```
    unsigned int    count;
```

A structure array holds the attribute of the images. Please check 4.3 for more information.

```
    IMAGE_INFO_T    image[4];
} BOOT_INFO_T;
```

4.2 SPI Information

This structure is use to hold the SPI NOR and SPI NAND flash attribute. And is ignored if booting from other storage device. Please refer to the flash's datasheet for the attributes to fill in this structure.

```
typedef struct spi_info
{
```

The SPI NAND flash size.

```
    unsigned short  PageSize;
```

The SPI NAND spare area size of each page.

```
    unsigned short  SpareArea;
```

The SPI NAND page count per block.

```
    unsigned short  PagePerBlock;
```

This is the quad read command of SPI flash.

```
        unsigned char   QuadReadCmd;
```

The command used to read flash status.

```
        unsigned char   ReadStatusCmd;
```

The command used to write flash status.

```
        unsigned char   WriteStatusCmd;
```

This value defines the status register bit to control SPI NOR flash enter and exit quad mode.

```
        unsigned char   StatusValue;
```

This attribute is used to store the dummy byte count between command and address.

```
        unsigned char   dummybyte1;
```

This attribute is used to store the dummy byte count between and data.

```
        unsigned char   dummybyte2;
```

This attribute defines the suspend interval between two successive transmit/receive transaction in a transfer. Valid values are between 0~15.

```
        unsigned char   SuspendInterval;
```

These three bytes are used to make following entry in boot_info word aligned.

```
        unsigned char   reserved[3];
} SPI_INFO_T;
```

4.3 Image Information

This structure is used to hold the attribute of images in boot_info.

```
typedef struct image_info
{
```

The offset this image is stored in storage device.

```
        unsigned int   offset;
```

The image load address in DDR or SRAM.

```
        unsigned int   loadaddress;
```

The image size in byte.

```
        unsigned int   size;
```

Image type. Valid values are 1: stands for TSI image, 2: System setting image, 3: Data image, and 4: Loader image.

```
        unsigned int   type;
```

These fields store the ECDSA signature of the image.

```
        unsigned char  signatureR[32];
        unsigned char  signatureS[32];
} IMAGE_INFO_T;
```

5 PACKED IMAGE FORMAT

NuWriter tool support packed image, which is a combination of several independent images. A packed image could be used for production. NuWriter support pack and unpack function. Table 5-1 shows the packed image format, all starts at 16-byte boundary and reserved fields should filled with 0xFF.

Address	0x0	0x4	0x8	0xC
0x0000	Marker (“NVT”)	CRC32	Total image count	Reserved
0x0010	Image 0 length		Image 0 offset	
0x0020	Image type	Reserved	Data	
0x0030	Data			
…	Data			
…	Data			Reserved
…	Image 1 length		Image 1 offset	
…	Image type	Reserved	Data	
…	…			

Table 5-1 Pack Image Format

Note: CRC32 calculates the checksum starting from offset 8.

Note: Image type should leave as 0.

6 NUWRITER COMMAND LINE INTERFACE

6.1 Command Format

The command format of NuWriter command format is shown as follows:

```
$ nuwriter.py [-h] [-a] [-o OPTION [OPTION ...]]
              [-c | -p | -r READ [READ ...] | -w WRITE [WRITE ...] | -e
              ERASE [ERASE ...] | -s STORAGE [STORAGE ...]]
              [CONFIG]
```

Where -a is to attach the USB interface. -c is to convert the boot header and encrypt/sign the images is necessary. -p is to pack images for mass production. -r, -w, and -e commands are for read, write and erase storage device.

6.2 Attach

A connection must be setup for NuWriter tool to communicate with MA35D1 before access the storage device. And this command is used to set up the connection. Except convert and pack commands, other commands can only execute after the connection is setup. Here is the attach command format where DDR_INIT is the DDR initialize code match the MA35D1 part.

The DDR type of MA35D1 MCP package supports Winbond DDR3 256MB, Winbond DDR3 512MB, and Winbond DDR2 128MB. The external DDR type of MA35D1 only supports Micron DDR3 1GB, ISSI DDR3 1GB, and Zentel DDR3 1GB.

MA35D1 supports booting from two SD/eMMC interface depending on the power on setting. In order for NuWriter to control the correct interface, the power on setting should set properly before issue the attach command.

```
$ nuwriter.py –a DDR_INIT
```

For some storage devices NuWriter cannot program due to attribute detect error, user can use an setinfo option to specify the storage device's attribute. If this option is set, NuWriter will parse attribute from a file named info.json. The command format is:

```
$ nuwriter.py –o setinfo –a DDR_INIT
```

6.2.1 JSON File Format for SETINFO option

info.json contains three parts, represent the attribute of SPI NOR, SPI NAND, and NAND respectively. The top level keys "spinor", "spinand", and "nand" are optional, they don't need to present if setting its attribute is not required. But once an object presents, all its keys must be set according to the flash to be access.

For SPI NOR flash these options are for quad mode operations. This information can be found in SPI Flash datasheet.

```
{
    "spinor":
    {
```

Quad mode read command.

```
        "quadread": "0x6B",
```

Read status command.

```
        "readsts": "0x05",
```

Write status command.

```
        "writests": "0x01",
```

Write enable bit in SPI Flash status register.

```
        "stsvalue": "0x02",
```

Dummy byte length in Quad mode read command

```
        "dummy": "0"
    }
```

The some attributes are the same as the settings in JSON file for convert command, please refer to 6.3.1 for the definition of these attributes.

```
    "spinand":
    {
        "pagesize": "2048",
        "sparearea": "64",
        "quadread": "0x6B",
        "readsts": "0x05",
        "writests": "0x01",
        "stsvalue": "0x02",
        "pageperblk": "64",
```

Following two attributes does not present in 6.3.1, “dummy” is not used and should keep as 0, “blkcnt” is the total block count of SPI NAND flash.

```
        "dummy": "0",
        "blkcnt": "2048"
    },
```

Only two NAND flash’s attribute can be set, total block count and page per block. Other attributes (page size and ECC) should be set by power-on-setting.

```
    "nand":
    {
        "blkcnt": "4096",
        "pageperblk": "128"
    }
}
```

6.3 Convert

The main purpose of convert command is to generate a boot header that can be used program into storage media allowing MA35D1 IBR to identify the images for booting up the system. This command can also help to encrypt and sign the image and convert a plan text U-Boot environment variable file to an environment binary image.

This command can generate random AES key or ECDSA key for encryption and sign the image if the keys are empty in configuration file and secure boot is enabled. Here listed the convert command format where the CONFIG is the JSON format configuration file.

```
$ nuwriter.py -c CONFIG
```

NuWriter tool will then generate a directory name by the date and time of the script execution time and put the generated files under this directory. And create a symbolic link named “conv” link to the latest generated directory. Below list the generated files and their meaning.

- header.bin

  Header file to be programmed into storage allowing IBR to identify the boot images.

- header_key.txt (optional)

  Stores the AES key and ECDSA public and private key for boot images (TSI image and BL2 image) if secure boot is enabled. If the keys are not specified in the configuration file, NuWriter can help to generate random keys and put in this file.

- enc_[original boot image name] (optional)

  NuWriter will generate encrypted and signed image if secure boot is enabled.

- uboot-env.bin (optional)

  U-Boot environment binary image if the plan text file is specified in configuration file.

- data_key.txt (optional)

  Stores the AES key and ECDSA public and private key for data images.

- enc_[original data image name] (optional)

  Encrypted and signed data image.

- sig_[original data image name] (optional)

  Signature (R and S) of signed data image.

6.3.1 JSON File Format for Convert Command

This section describes the meaning of each configuration file format.

“header” object is used to describe the boot header parsing by IBR.

```
{
    "header":
    {
```

Version number is a 32-bit user defined version number.

```
        "version": "0x20200622",
```

“spiinfo” is to store the SPI flash attribute. These fields could leave with 0s for the system boots from storage device other than SPI NOR/NAND flash. The values to fill in this object could be found from the SPI NOR/NAND flash’s datasheet.

```
        "spiinfo":
        {
```

Page size of SPI NAND flash.

```
            "pagesize": "2048",
```

Spare area size of SPI NAND flash.

```
            "sparearea": "64",
```

Page per block of SPI NAND flash.

```
            "pageperblk": "64",
```

Quad read command for SPI NAND flash. Could leave 0 if the system is booting with 1-bit mode.

```
            "quadread": "0x6B",
```

Read status register command of SPI flash.

```
            "readsts": "0x05",
```

Write status register command of SPI flash.

```
            "writests": "0x01",
```

This value defines the status register bit to control SPI flash enter and exit quad mode.

```
            "stsvalue": "0x02",
```

“dummy1” specifies the dummy byte count between command and address in the control sequence. And “dummy2” specifies the dummy byte count between address and data in the control sequence.

```
            "dummy1": "0",
            "dummy2": "1",
```

This attribute defines the suspend interval between two successive transmit/receive transaction in a transfer. Valid values are between 0~15.

```
            "suspintvl": "1"
        },
```

Set “yes” to generate header file for secure boot. Otherwise NuWriter will generate header file without secure boot support.

```
        "secureboot": "yes",
```

“entrypoint” specifies the location IBR branch to after images are loaded and verified correctly.

```
        "entrypoint": "0x90000000",
```

An optional entry used to hold AES 256 encrypt key for image encryption. If secure boot is enabled and this key is missing in configuration file, NuWriter will generate a random key.

```
        "aeskey":
"1234891273239865093047419837109232109381208489741 29ABBACD879FFFF",
```

An optional entry used to hold ECDSA private key for image signing. If secure boot is enabled and this key is missing in configuration file, NuWriter will generate a random key.

```
        "ecdsakey":
"2321093812084897412 9ABBACD879FFFFABF000345E0EE2B2B1B3123B12B3B10",
```

“image” array is used to define up to four images recorded in boot header. Each element should contain an “offset” value stores the offset in storage device where the image is programmed. A “loadaddr” value stores the DDR or SRAM address where the image is to be load to. A “type” value specifies the image type where “1” is TSI image, “2” is system setting image, “3” is data image such as device tree blob, and “4” is loader image. And a “file” value indicates the image name and directory in the local disk.

```
        "image":
        [
            {
              "offset": "0x500000",
              "loadaddr": "0x28030000",
```

```
            "type": "2",
            "file": "loader.bin"
        },
        {
            "offset": "0x50A000",
            "loadaddr": "0x88000000",
            "type": "3",
            "file": "ma35d1.dtb"
        }
    ]
},
```

“env” object is optional. If exist, this object should contain the plan text environment variable file, and specify the storage device block size. NuWriter will generate a binary image with padding 0xFF to the size of blksize bytes.

```
    "env":
    {
        "file": "uboot-env.txt",
        "blksize": "0x10000"
    },
```

The “data” object in configuration file is also optional. The images specified in this object will not be included in the boot header information. If AES key or ECDSA private key is missing, NuWriter can randomly generate the keys for encryption and sign. The main purpose for “data” object is to use NuWriter tool for encrypt data images not interpret by IBR. For example, the Linux kernel image.

```
    "data":
    {
        "aeskey":
"7A9C34B12321313293047419837109232109381 20FAADDEFE1BF2EB1F2EFFE12",
        "ecdsakey":
"817239EF1BBB7A8EF25B7995157844747FABE669ABF000345E0EE21B63624D59",
        "image":
        [
            {"file": "image1.bin"},
            {"file": "image2.bin"}
        ]
    }
}
```

6.4 Pack

The pack command can pack several images into a single one. Allowing NuWriter to programming these images in a single write command. So the packed image can be used in production stage.

The command format for generate a packet image is listed below where CONFIG is the JSON configuration file describes the images to be packed.

```
$ nuwriter.py –p CONFIG
```

NuWriter tool will then generate a file named pack.bin under a directory name by the date and time of the script execution time. And create a symbolic link named “pack” link to the latest generated directory. For example, the image file will be 1224-235959000/pack.bin if user execute this command at the last second of Christmas eve.

Pack command supports a “stuff” option that can generate a single image with 0xFF stuffs between the image gaps. Some writer can program this image to SPI NOR flash. Please note packet image generated by stuff option should not be used to program NAND or SPI NAND flash because the image base may be shifted due to bad blocks.

This command also supports the reverse direction to unpack an image, below is the command to unpack a packed image where PACK_IMAGE is the packed image name. After unpacked, the files will be stored under a directory name by the date and time of the script execution time. And create a symbolic link named “unpack” link to the latest generated directory.

```
$ nuwriter.py –o unpack –p PACK_IMAGE
```

6.4.1 JSON File Format for Pack Command

Below is the JSON configuration file format for pack command. It should contain the offset in storage media, image type, file name (including directory path if necessary) of the images to be packed. The image type is reserved for future use and should keep as 0.

```
{
    "image":
    [
        {
          "offset": "0x500000",
          "file": "image1.bin",
          "type": 0
        },
        {
          "offset": "0x50A000",
          "file": "image2.bin",
          "type": 0
        }
    ]
}
```

6.5 Read

Read command can be used the read back the content in storage device for verification or debug. The command format is:

```
$ nuwriter.py –o OPTION –r DEVICE RANGE FILE
```

The range could be “all” to read whole device back, or specified the starting block and block count to read back. FILE specifies the output file to store the data read back form device.

For example, to read whole SPI NOR back to a file named output.bin, the command is:

```
$ nuwriter.py –r spinor all output.bin
```

To read OTP block1 ~ block7 back to a file named output.bin, the command is:

```
$ nuwriter.py –r otp all output.bin
```

To read OTP block 3 length is 8 bytes back to a file named output.bin, the command is:

```
$ nuwriter.py –r otp otp3 8 output.bin
```

To read three blocks starting from NAND flash block five and store to a file name output.bin, the command is:

```
$ nuwriter.py –r nand 5 3 output.bin
```

The default behavior of read command is to skip bad block on NAND flash and SPI NAND flash. So if the block five on a NAND flash is a bad block, previous command can skip block five on NAND flash and read back block six, seven and eight. Sometimes it is necessary to read the block back even it is a bad block for debugging purpose. NuWriter offers an option “withbad” that can force to read the data back regardless the bad block mark in NAND flash. Please note that with this option, the OOB content will also be read back.

```
$ nuwriter.py –o withbad –r nand 5 3 output.bin
```

6.6 Write

The write command is used to program the external storage device as well as internal SRAM and OTP. The command format is:

```
$ nuwriter.py –o OPTION –w DEVICE OFFSET FILE
```

The DEVICE specify where to program to, could be “ddr”, “sram”, “sd”, “spinand”, “spinor”, “nand” and “otp”. The OFFSET specified the offset in storage device to program to and should be omitted while programming OTP or packed image. And OFFSET must be block aligned for NAND and SPI NAND flash, 4KB aligned for SPI NOR flash. FILE is could be the single image, packed image or JSON file for program the OTP. Below are some samples of write command.

This command programs image.bin to offset 0x1000 of SPI NOR flash.

```
$ nuwriter.py –w spinor 0x1000 image.bin
```

Following command programs OTP according to the otp.json configuration file.

```
$ nuwriter.py –w otp otp.json
```

This command programs a packed image into NAND flash. Since the packed image contains the offset information already, there is no need to specify the offset in this command.

```
$ nuwriter.py –w nand pack.img
```

NuWriter tool supports “execute” option if the storage set as DDR or SRAM. If the “execute” option is set, NuWriter MA35D1 will branch to the offset address and execute after transfer complete.

```
$ nuwriter.py –o execute –w ddr 0x80000 test.bin
```

Another option support by write command is “verify”. This option is valid if the storage specified in the command is other than “ddr”, “sram”, and “opt”. NuWriter will read back the image and compare the content after program complete and report the result to console.

```
$ nuwriter.py –o verify –w sd 0x100000 test.bin
```

6.6.1 JSON File Format for Programming OTP

NuWriter required a JSON configuration file to specify the value to program into OTP. Here describes the meaning of each objects. All objects and elements in objects are optional and can be omitted if the value does not need to be changed.

```
{
```

“boot_cfg” object contains the booting related setting.

```
    "boot_cfg":
    {
```

If “postop” set to “enable”, the setting in OTP will overwrite the setting of Power-On-Setting pins. So the pull up resistors can be removed on PCBA for production.

```
        "posotp": "enable",
```

“qspiclk” specifies the SPI clock for SPI boot, the valid settings are “50mhz” and “30mhz”. IBR use this setting for loading the binary images stores in SPI NOR and SPI NAND flash. Firmware can set to other desired clock frequency.

```
        "qspiclk": "50mhz",
```

“wdt1en” specified whether the WDT1 is “enabled” or “disabled” after power on.

```
        "wdt1en": "disable",
```

“uart0en” specified whether the UART0 is “enabled” or “disabled” after power on. If disabled, IBR will not print out any message while booting. Firmware can enable UART0 later.

```
        "uart0en": "enable",
```

“sd0bken” is used to “enable” or “disable” SD0 back up feature. If this feature is enabled, IBR will try to load a backup image from “SD0” is booting from the storage assigned by “bootsrc” failed.

```
        "sd0bken": "disable",
```

“tsiimg” is used to “enable” or “disable” IBR load TSI image during boot stage.

```
        "tsiimg": "enable",
```

“tsidbg” is used to “enable” or “disable” TSI debug interface.

```
        "tsidbg": "disable",
```

“bootsrc” specifies the boot source for MA35D1. The valid settings are “nand”, “spi” (for boot from SPI NAND or SPI NOR flash), and “sd” (for boot from SD card or eMMC). Only set to “usb” for debugging purpose, because once set, the system can only load image from NuWriter.

```
        "bootsrc": "nand",
```

“page” is used to configure the NAND flash page size. Valid options are “2k”, “4k”, “8k”, and “ignore”. If this value is set to “ignore”, IBR will try to determine the correct page size by ONFI and ID of the flash.

```
        "page": "2k",
```

Like Power-On-Setting pins PG[7:6], “option” controls different feature while booting with different “bootsrc”. While booting from NAND flash, valid settings are “ignore”, “t12”, “t24”, and “noecc”. IBR will determine the ECC bit to use by ONFI and ID of the flash. “t12” and “t24” will force the IBR to use T12 and T24 to detect and correct data. The last option “noecc” is set for those NAND flash controllers contain its own ECC controller unit.

While booting from SD card or eMMC, valid values are “sd0” and “sd1”. IBR will use Port C for booting while set to “sd0” and use Port J while set to “sd1”.

```
        "option": "ignore",
```

“secboot” can ether set to “disable” or “enable” to disable/enable secure boot feature.

```
        "secboot": "disable"
    },
```

“mac0” and “mac1” specifies the MAC address of MA35D1’s two GMAC interfaces. The MAC address range should be purchased from IEEE.

```
    "mac0": "000000112233",
    "mac1": "000000445566",
```

The “dplypwd” element is a 32-bit value that reserved for future used. And should fill with 0s.

```
    "dplypwd": "00000000",
```

“sec” is a hexadecimal number up to 88-byte that can only be access from secure state once programmed into OTP. The usage is user defined. Like its counterpart “sec”, “nonsec” is the hexadecimal number up to 88-byte that can be access from non-secure state once programmed into OTP.

```
    "sec": "00",
    "nonsec": "00",
```

“huk0”, “huk1”, and “huk2” are hardware unique keys that stores in OTP. The usage is application specified. The maximum length of these keys are 128 bits. And meta fields only maps to the key owner – “cpu” and cpu-readable.

```
    "huk0":
    {
        "key": "002091283FAD81329304741983710923"
    },
```

“key3”, “key4”, and “key5” are key 3, 4, and 5 stored in OTP. The purpose is for cryptographic. It is for the application to decide when or how to use these three keys. The maximum length of these keys are 256 bits. And meta fields maps to the key owner. They are “aes256-unreadable”, “aes256-cpu-readable”, “sha256-unreadable”, “sha256-cpu-readable”, “eccp256-unreadable” and “eccp256-cpu-readable”.

```
    "key5":
    {
        "key": "002091283FAD883293047419837109232109381 20FBDD4EFE1BF2EB1F2EFFE12",
        "meta": "aes256-cpu-readable"
    },
```

“publicx” and “publicy” are the public key for verify the signature of signed images for secure boot. And “aeskey” are used to encrypt the image before programmed to storage device.

```
    "publicx": "886C323AC321313293046DDF83710923210938120F54345AAEFF2EB1F2226E12",
    "publicy": "97272B1232ADDEFE19304834183960ADC99AA9023FAA123FE1BF2EB1F2EFFE10",
    "aeskey" : "7A9C34B12321313793047466837189232109381 20F0993E2E1BF0989898AE39A"
}
```

6.7 Erase

Erase command is used to erase all or partial of the storage device. The command format is:

```
$ nuwriter.py –o OPTION –e DEVICE RANGE
```

The DEVICE could be “otp”, “nand”, “spinand”, and “spinor” where the case is ignored. The RANGE could be “all” to erase whole device, or specified the starting address and size to erase. Please note that the starting address for erase be block aligned for NAND and SPI NAND flash, 4KB aligned for SPI NOR flash.

For example, to erase whole SPI NOR flash, the command would be:

```
$ nuwriter.py –e spinor all
```

Or “all” can be omitted in previous command. Without specifying the erase range, whole flash will be erased.

```
$ nuwriter.py –e spinor
```

OTP only block1, 3, 4 can be erased. To erase OTP block4, the command is:

```
$ nuwriter.py –e otp otp4
```

To erase three blocks on a 128KB block sized NAND flash starting from offset 1MB, the command is:

```
$ nuwriter.py –e nand 0x10000 0x60000
```

The default behavior of erase command is to skip bad block on NAND flash and SPI NAND flash. So if the block five on a NAND flash is a bad block, previous command will skip block five on NAND flash and erase only back block six and seven. On a rare occasion, it is necessary to erase the block even it is marked as bad. NuWriter offers an option “scrub” that can force to erase the block regardless the bad block mark in NAND flash. This option will also erase the bad block mark stored in the OOB area. Please use this option with caution.

```
$ nuwriter.py –o scrub –e nand 0x10000 0x60000
```

7 NUWRITER GUI INTRODUCTION

NuWriter GUI mode provides more friendly way for using NuWriter. Currently it supports all function described in previous chapter. There are two option modes can be enabled on NuWriter GUI – Develop mode and OTP mode. If Develop mode is enabled, then NuWriter GUI will show the offline (Convert and Pack/Unpack) options at the main window. If OTP mode is enabled, then NuWriter GUI will show OTP option at the "Download" page.

The main window is divided into two or four parts – two online (Attach and Download) and two optional offline (Convert and Pack/Unpack) depend on the Develop mode setting. The main window is show as Figure 7-1 and will be descripted in the following chapter.

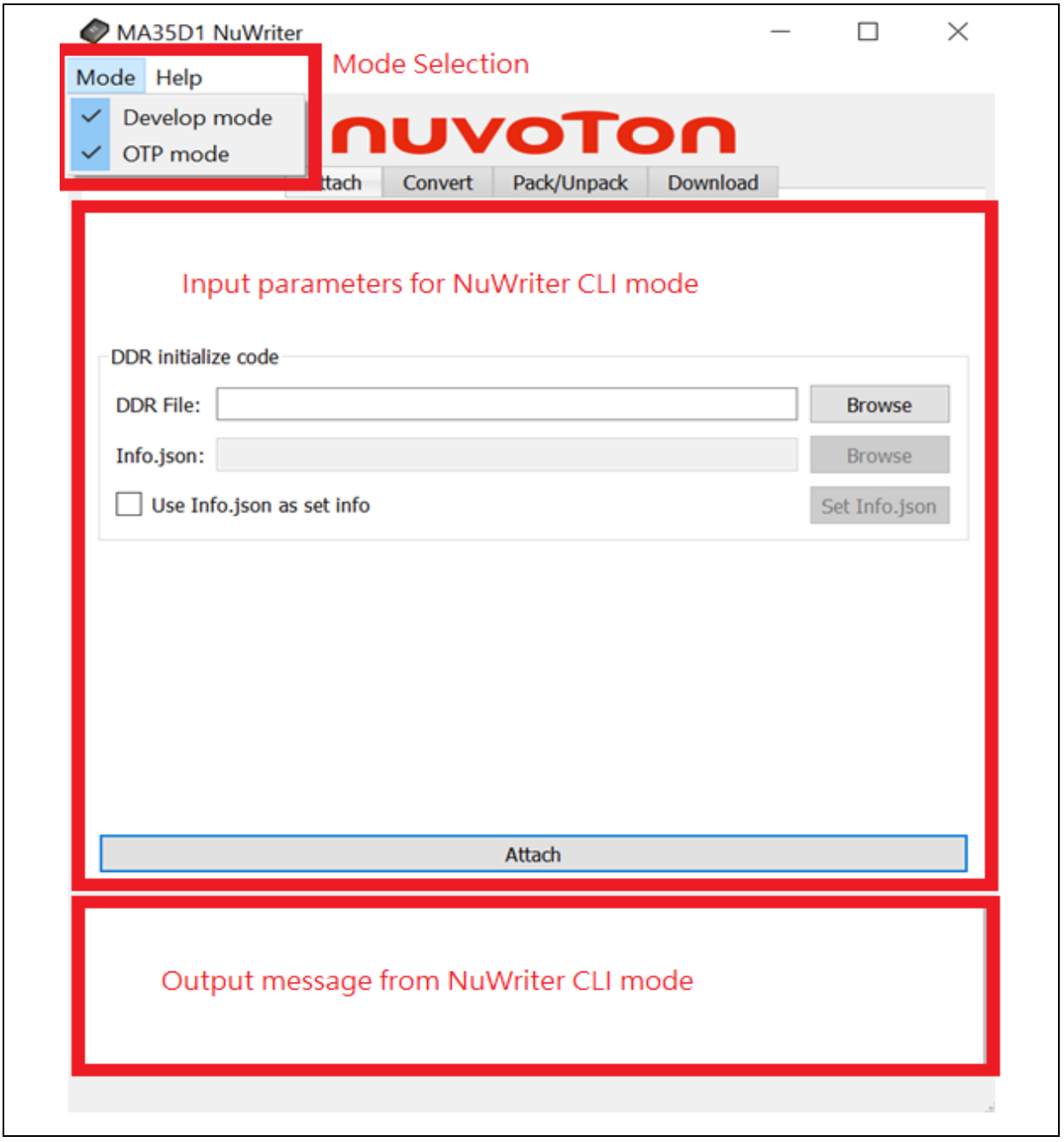

Figure 7-1 Main Windows

7.1 Attach

A connection must set up for NuWriter tool to communicate with MA35D1 to access the storage device. Before attaching the device, user can set platform information to device by info.json. Select “Use Info.json as set info”. As shown in Figure 7-2. Click the “Info.json Browse” button to select existent file. As shown in Figure 7-3. Click the “Set Info.json” button to show and set the information. User can click the Export button to export the new info.json. As shown in Figure 7-4

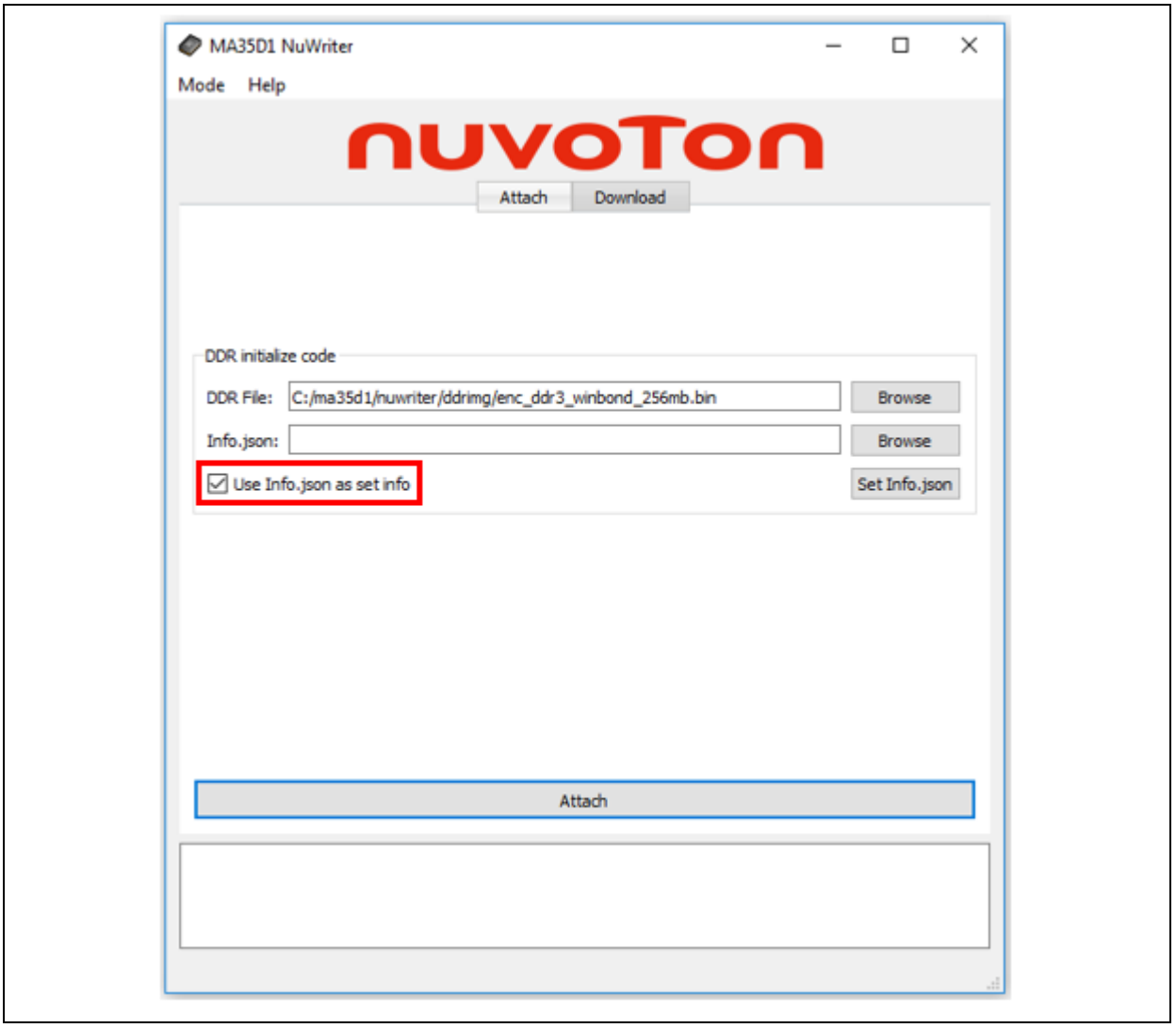

Figure 7-2 Use Information file

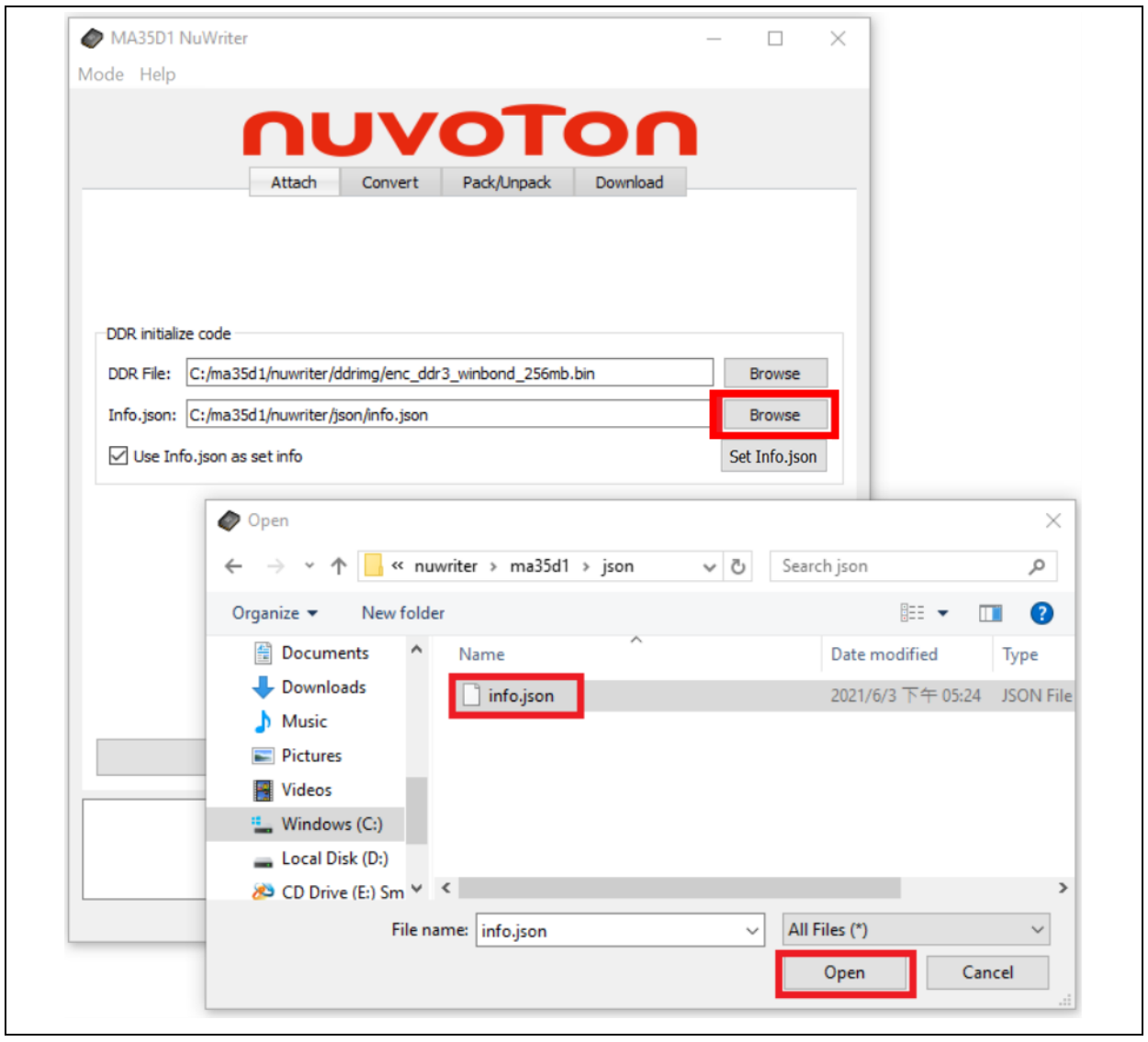

Figure 7-3 Browse Information file

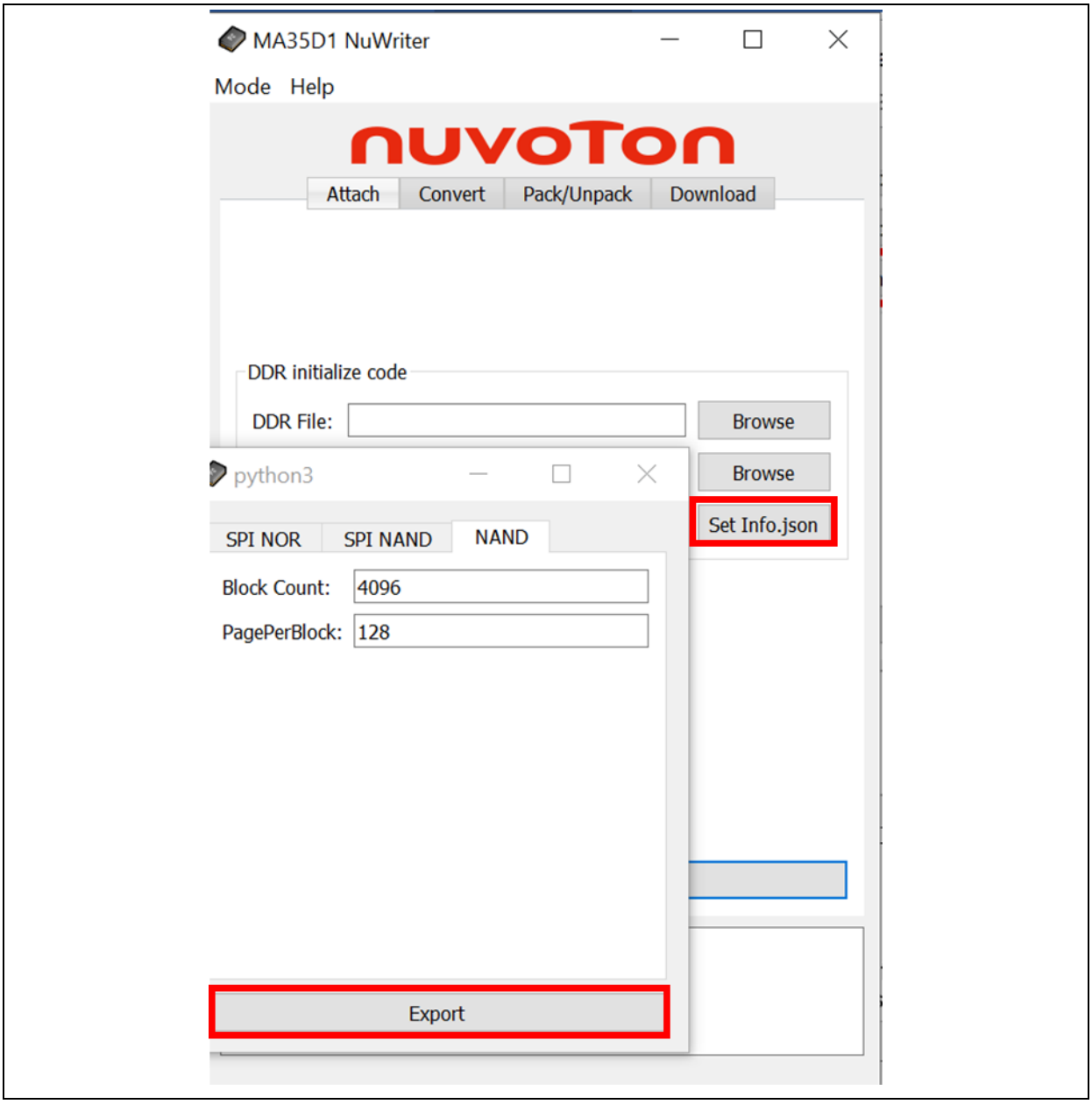

Figure 7-4 Information Content

Click the “DDR File Browse” button to select the DDR initialize code from the pop-up window as shown in Figure 7-5. Please make sure the DDR initialize code must be consistent with user’s hardware setting. Nuvoton provides some DDR initialize code in the “ddrimg” folder. For NuMaker-HMI-MA35D1 board, please select “enc_ddr3_winbond_256mb.bin”.

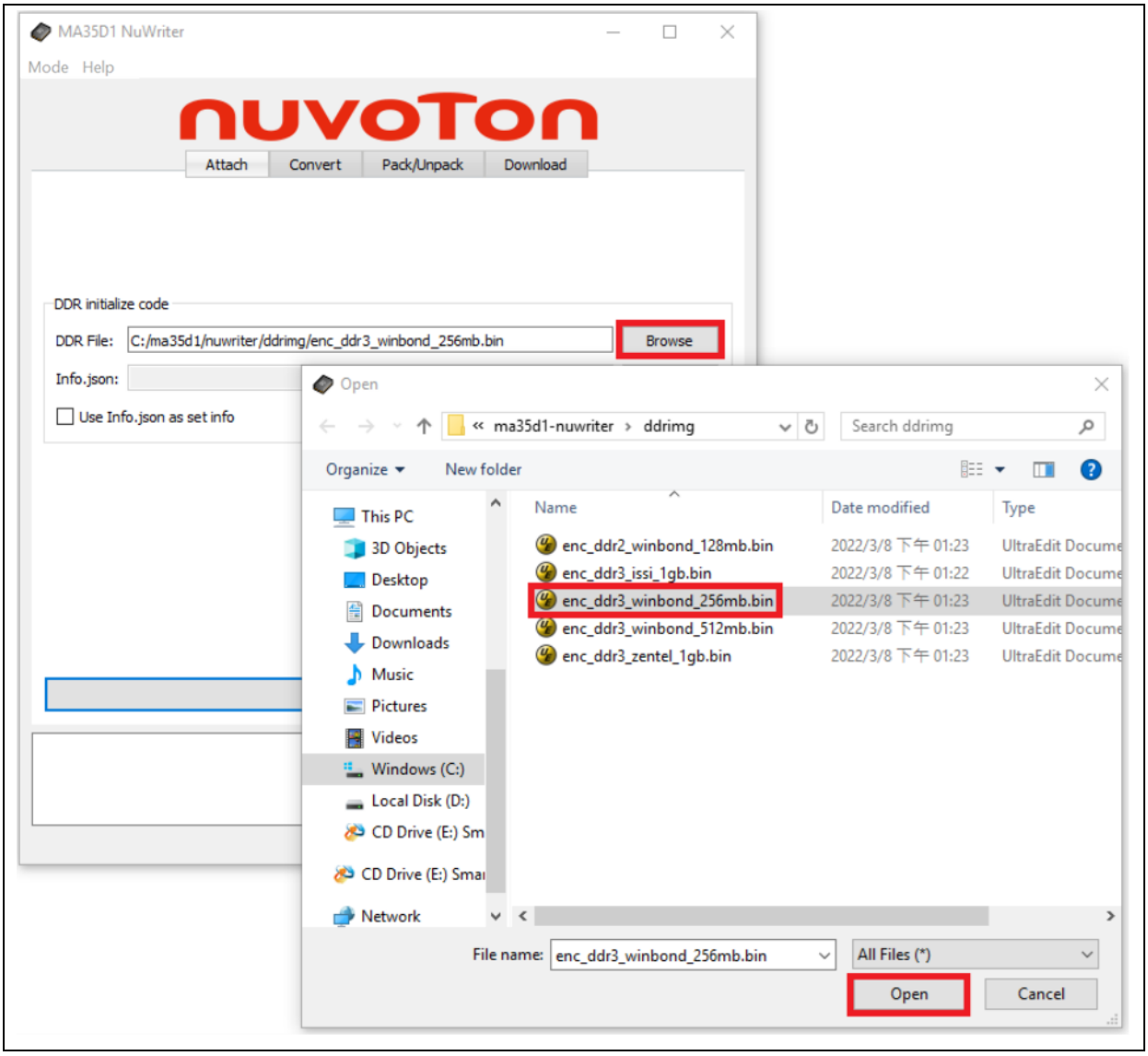

Figure 7-5 Browse DDR initialize code

Then click “Attach” button to trigger attach command. Figure 7-6 shows the attach result from the target board.

Figure 7-6 Attach Result

The GUI mode will collect all required inputs and trigger CLI mode to work. So the output message is consistent. The following figure shows the attach result from CLI mode and GUI mode. If any error occurred, please refer to the CLI chapter. The output message won’t be discussed in the following chapter.

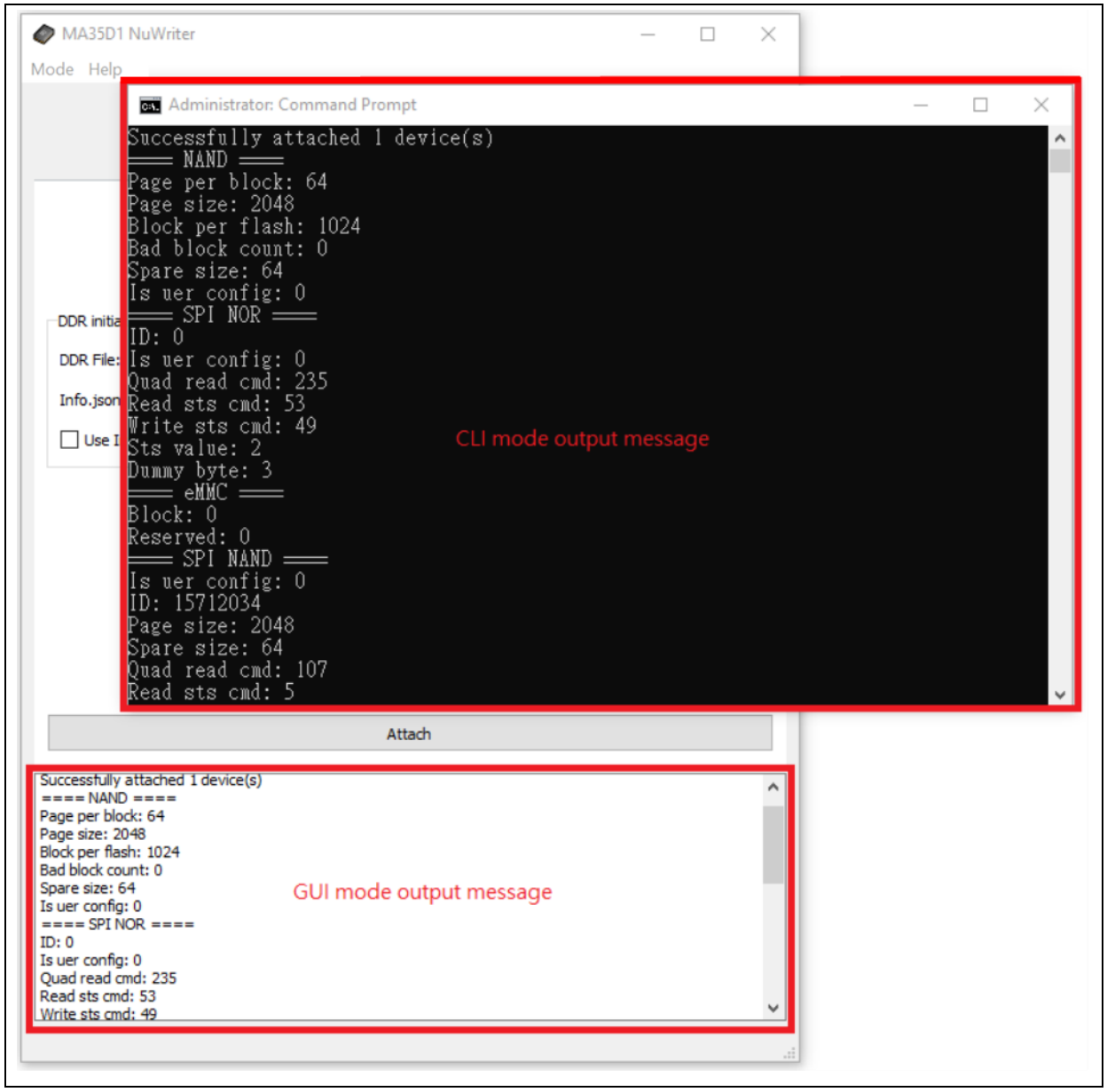

Figure 7-7 CLI and GUI mode output message

7.2 Download

In the “Download” page, user can find many storage media pages. We will take NAND to demo “Write”, “Read”, and “Erase” operations, since these operations are all similar for different storage media.

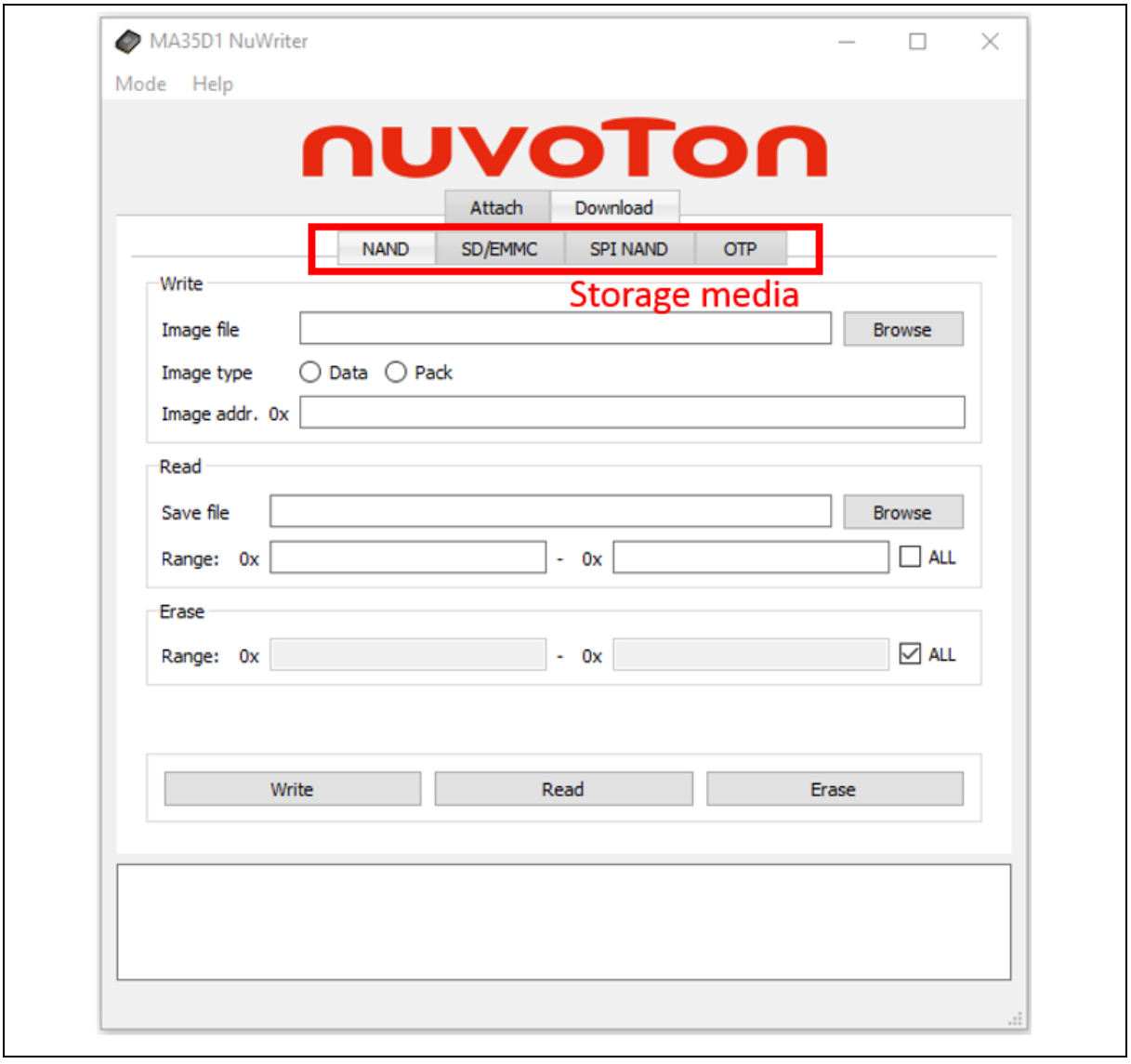

Figure 7-8 Storage Media

7.2.1 Write Operation Steps

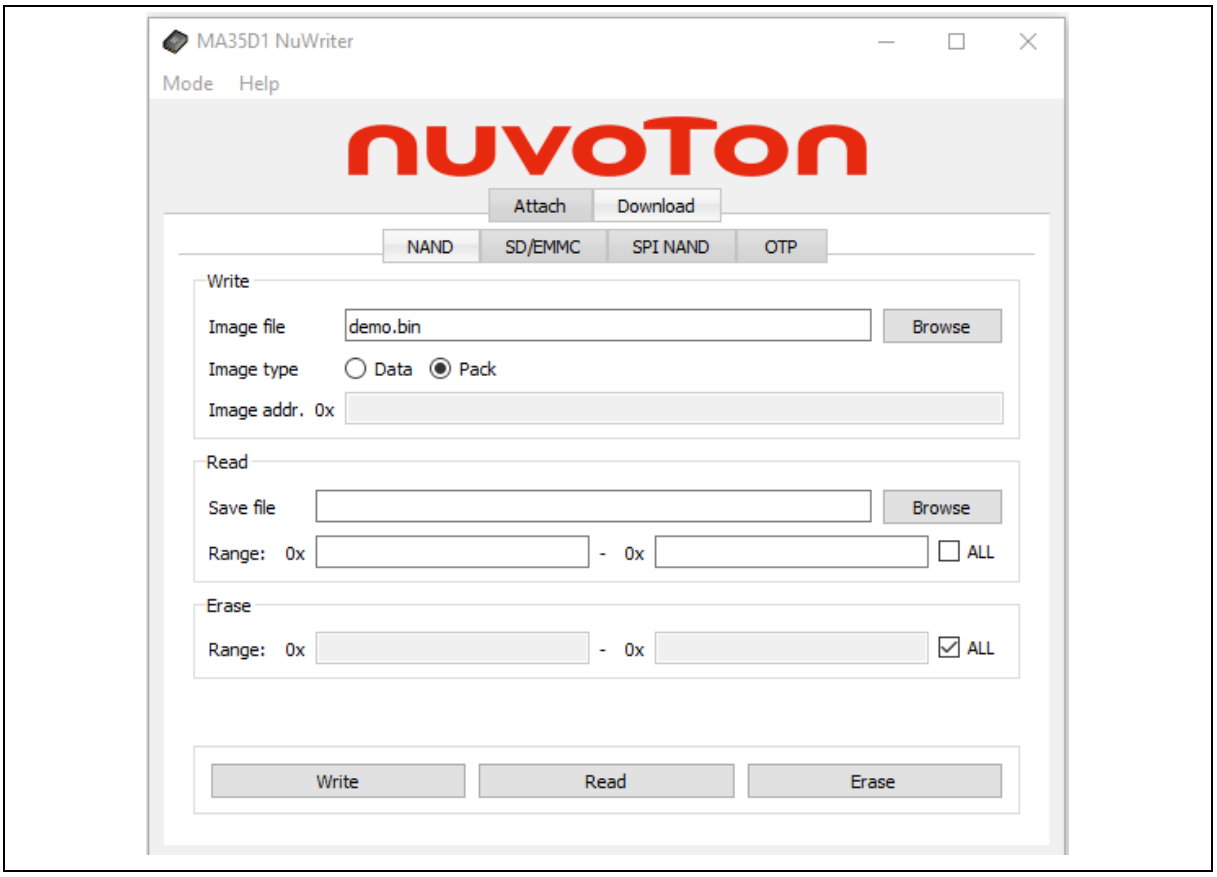

Figure 7-9 Write Example

According to the Figure 7-9 above, The NAND mode is used to download “demo.bin”. Follow the steps listed below:

1. Select “NAND”.
2. Browse the Image file by click the “Browse” button.
3. Select Image type according to the Image file. Demo.bin is a Pack Image.
4. Enter the image address. This is only required for “Data” Image type.
5. Click “Write”

7.2.2 Read Operation Steps

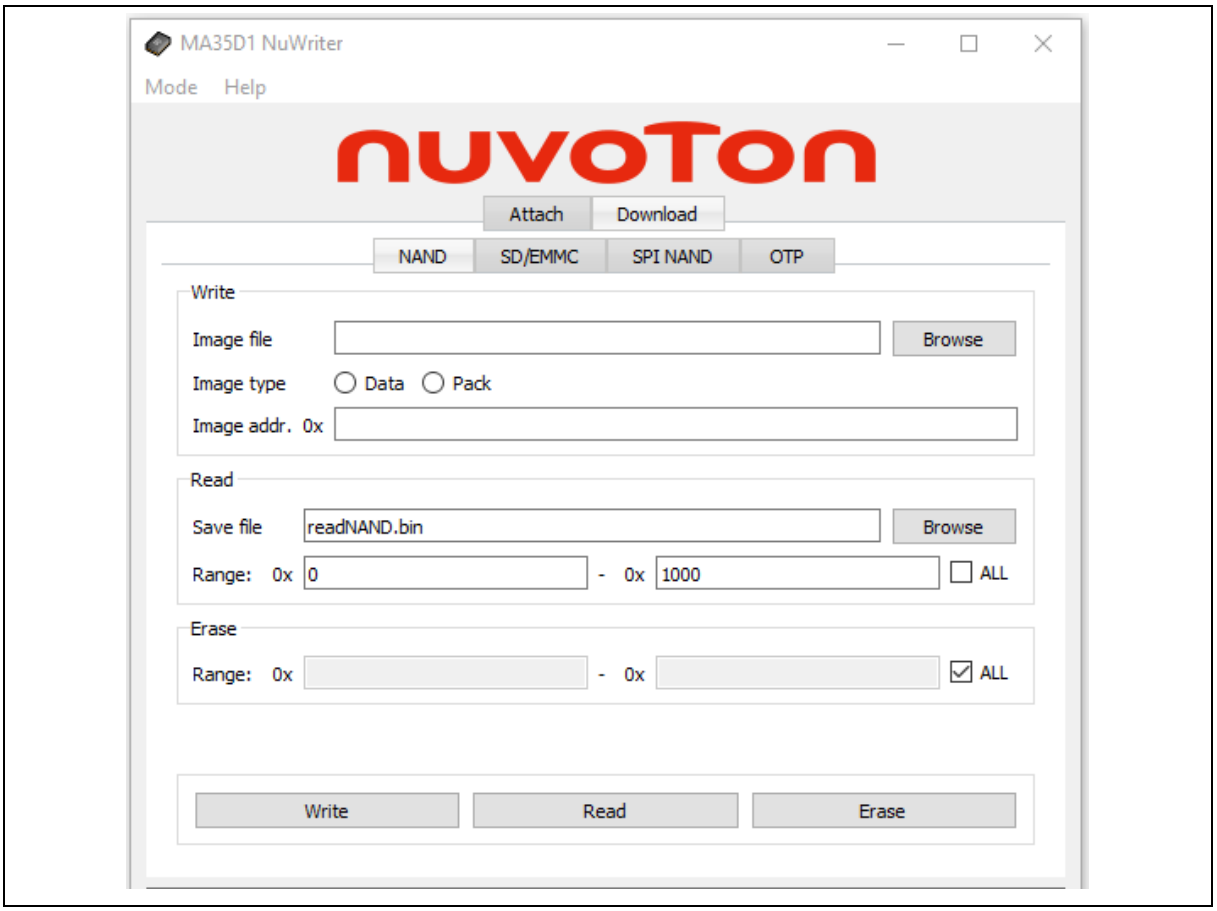

Figure 7-10 Read Example

According to the Figure 7-10 above, follow the steps below to read out the content from the NAND flash with the specific range.

1. Select “NAND”.
2. Browse the Save file by click the “Browse” button.
3. Enter the image Range.
4. Click “Read”

7.2.3 Erase Operation Steps

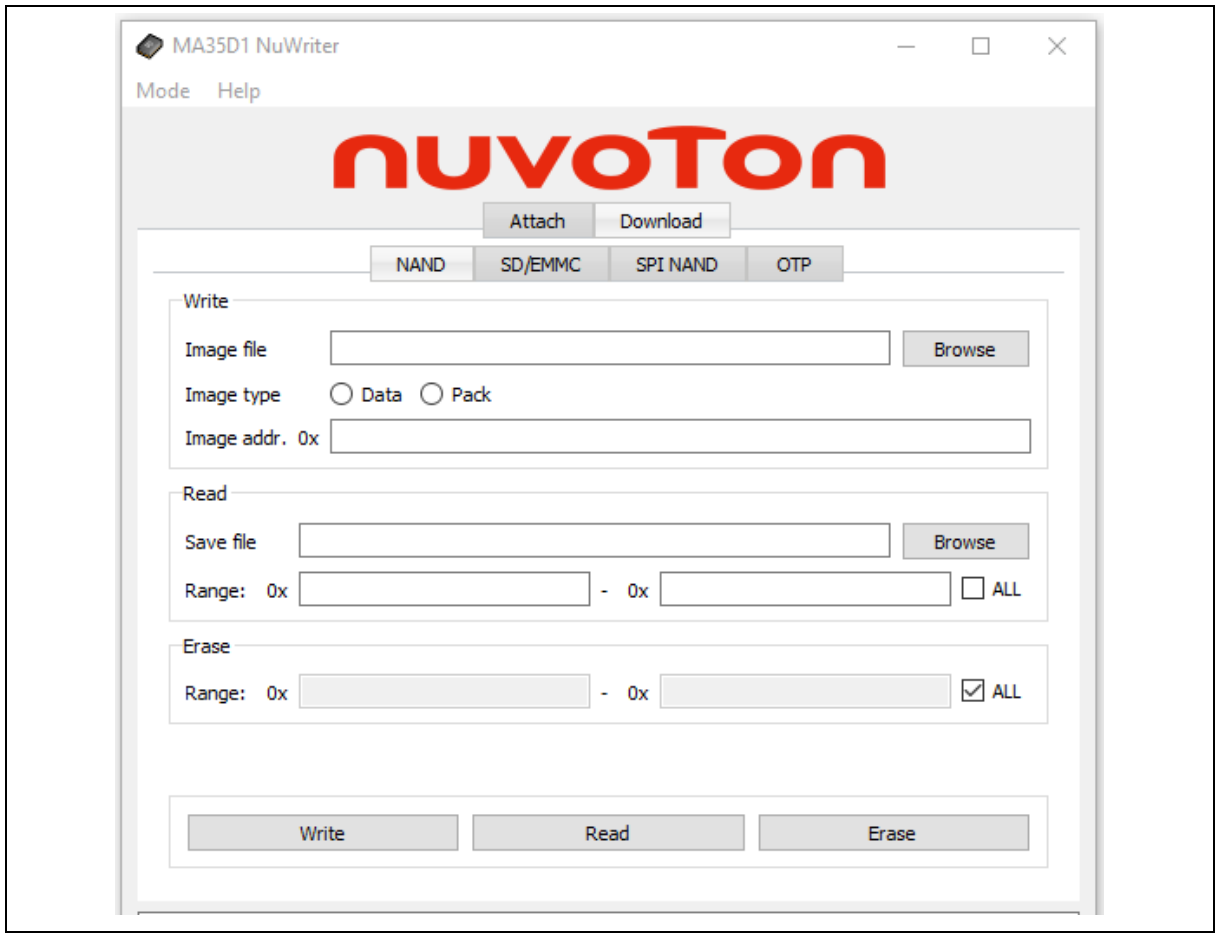

Figure 7-11 Erase Example

According to the Figure 7-11Figure 7-11 above, follow the steps below to erase the NAND flash with the specific range.

1. Select “NAND”.
2. Enter the Range or select “ALL” to erase whole storage media.
3. Click “Erase”

7.2.4 OTP Operation Steps

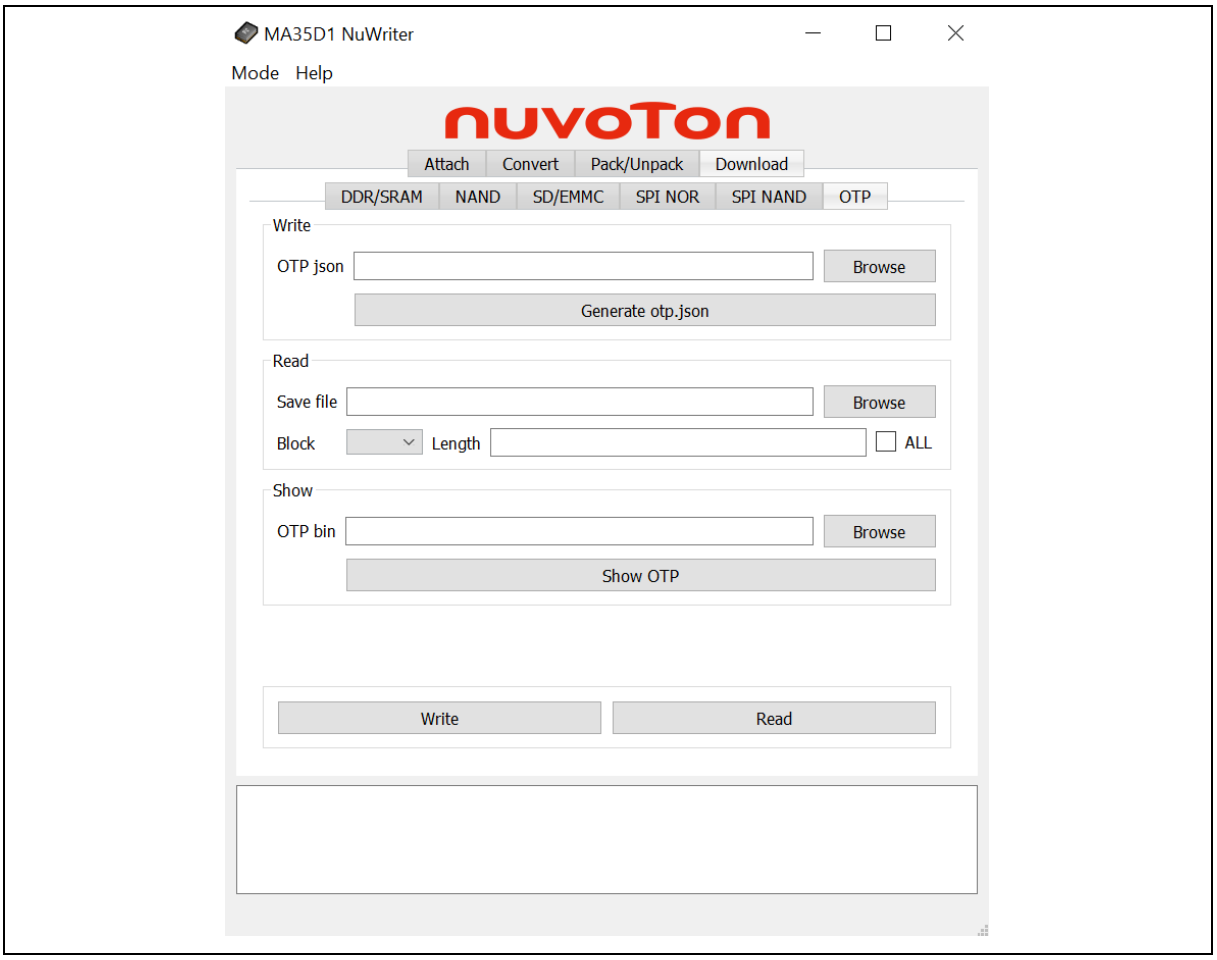

Figure 7-12 OTP Example

In the “OTP” page, NuWriter provides how to access OTP user field. Such as power-on-setting, MAC address, secure region, non-secure region, ...etc. User can program the exist OTP setting file directly, or create a new otp.json. NuWriter also can read OTP block1 ~ block7 back. The OTP block describes as Table 7-1.

Block No.	OTP
1	Power-on-Setting
3	MAC0 Address
4	MAC1 Address
5	Deployed Password
6	Secure Region
7	Non-Secure Region

Table 7-1 OTP Blocks

Create a new otp.json:

1. Click “Generate otp.json”. Figure 7-13
2. Enable / Disable OTP setting. Figure 7-14 and Figure 7-15
3. Select “Export OTP settings” to generate otp.json. Figure 7-16

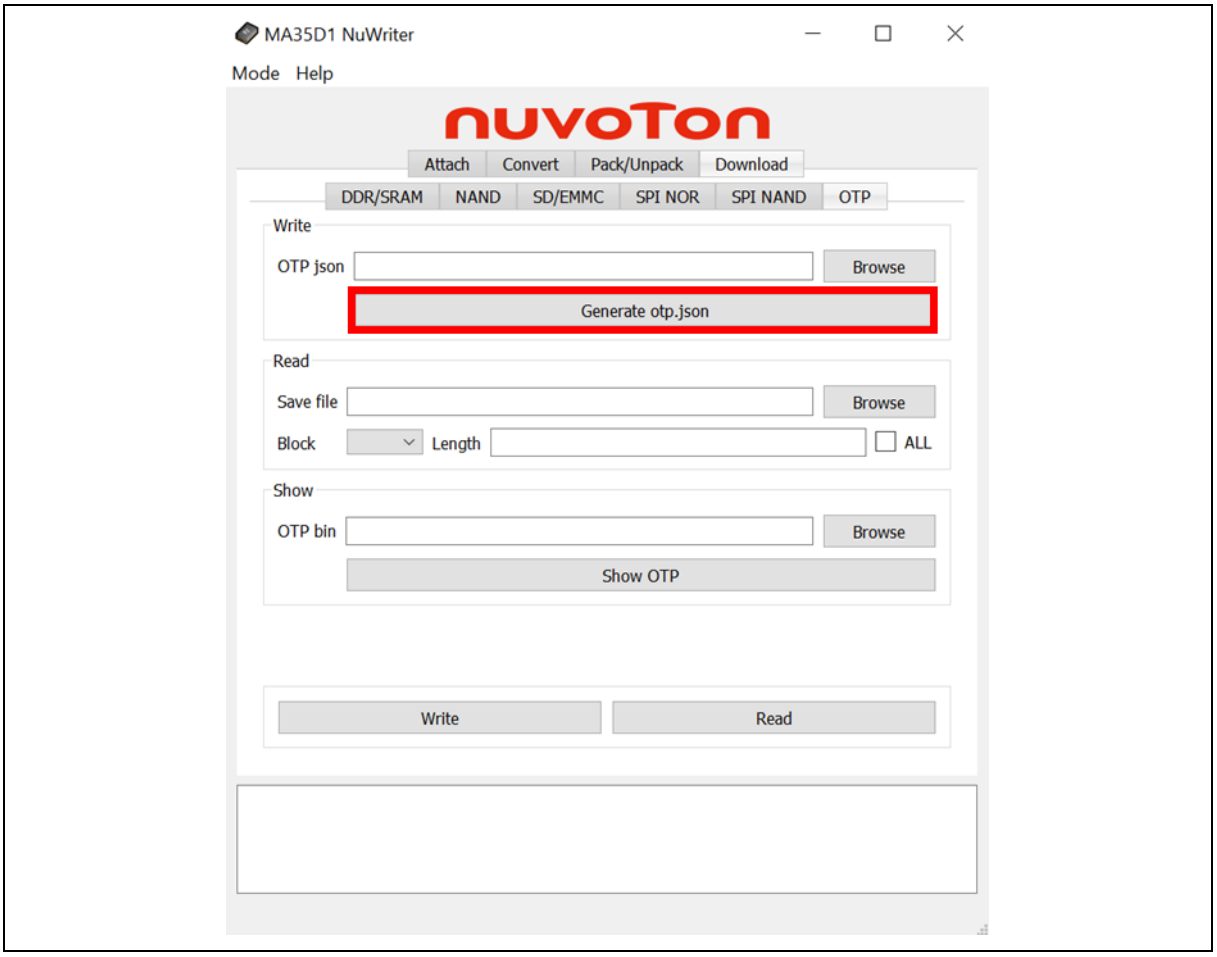

Figure 7-13 Generate otp.json

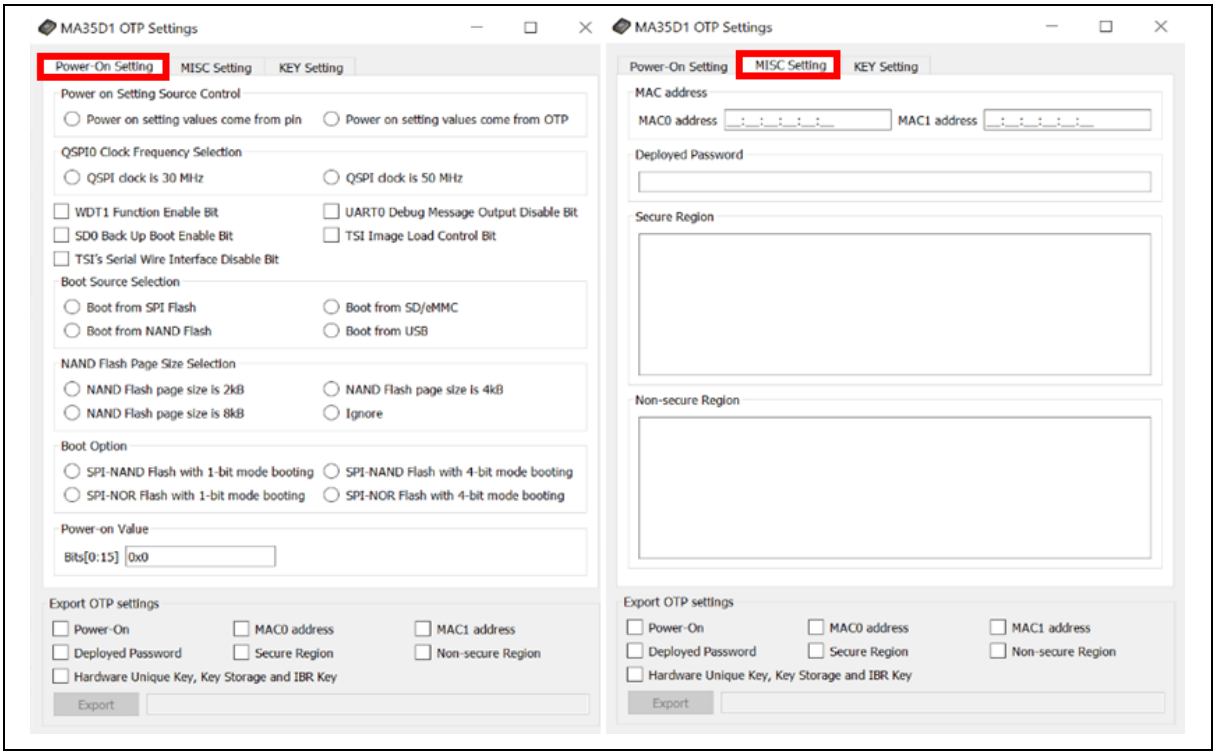

Figure 7-14 OTP Setting Page -1

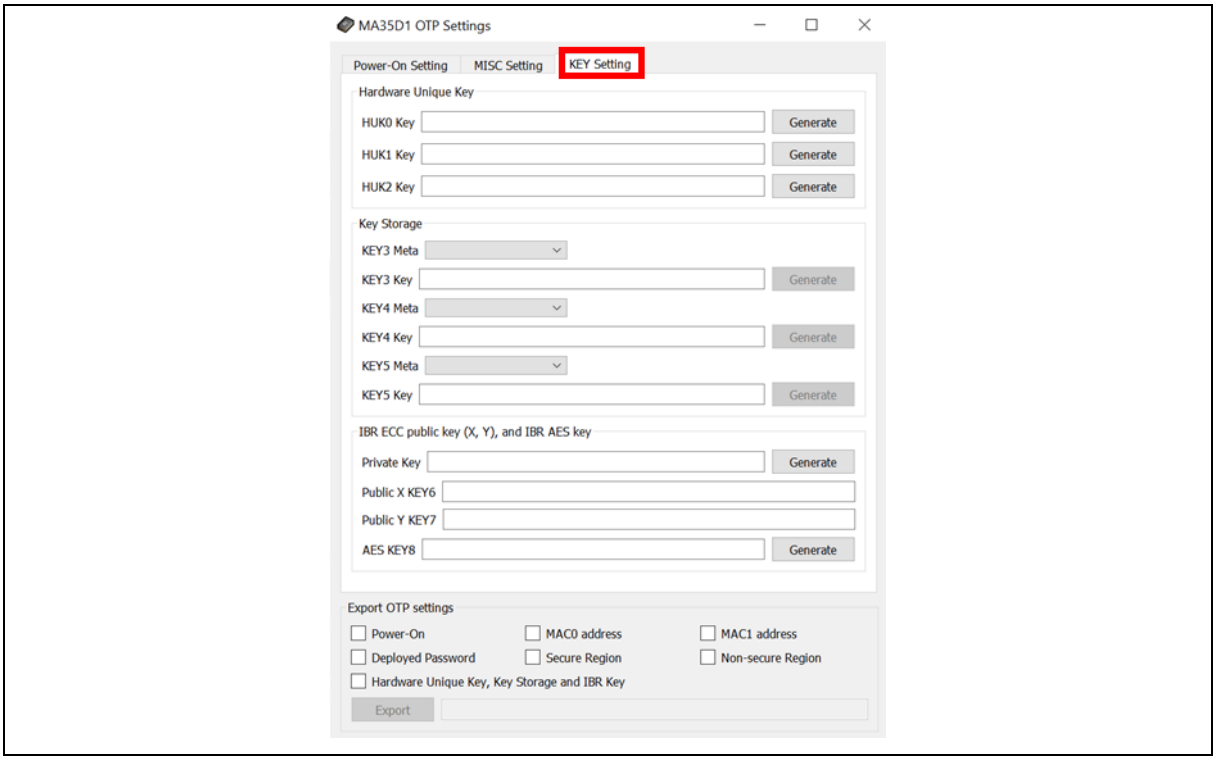

Figure 7-15 OTP Setting Page - 2

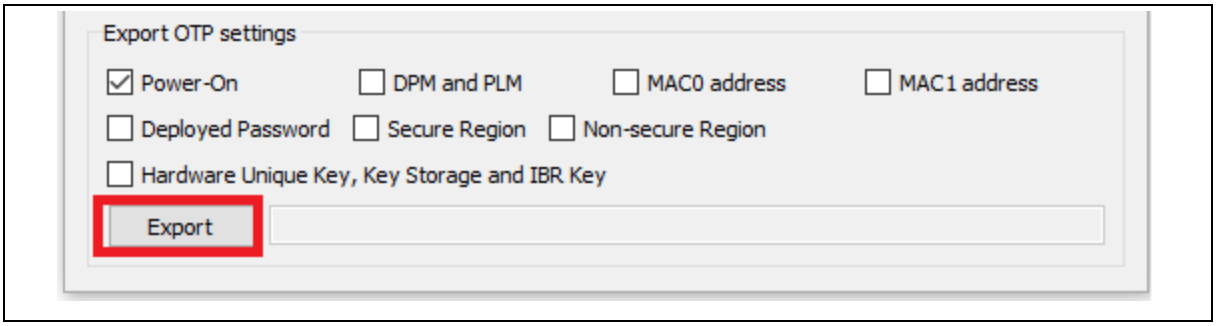

Figure 7-16 OTP Export

In “Power-On Setting” page, select own setting and it will show in “Power-On-Value”.

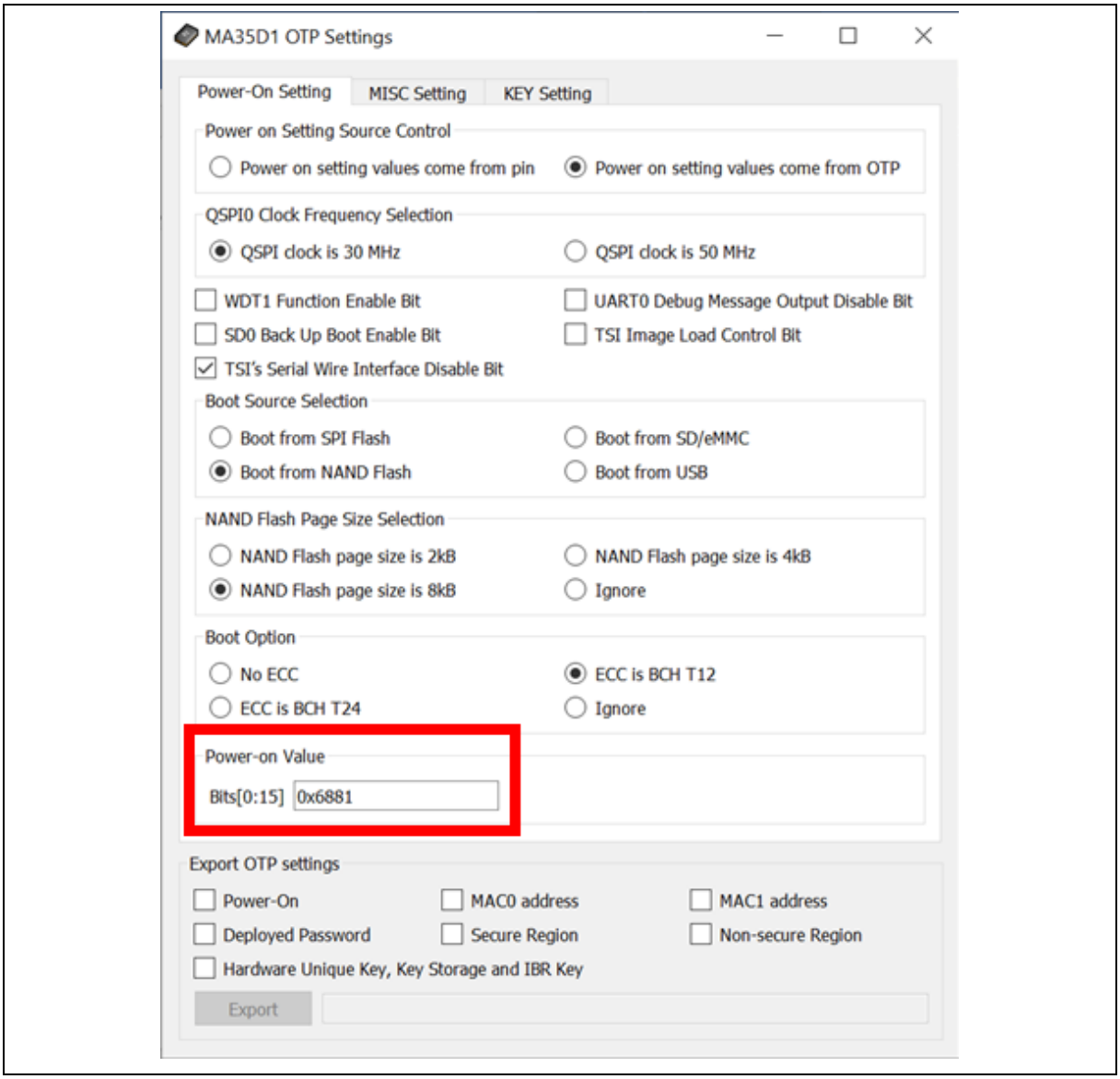

Figure 7-17 OTP Power-On Setting

According to the Figure 7-19 and Figure 7-20, user can choose the OTP setting read from the device (by read all) and show the content by clicking "show OTP" button.

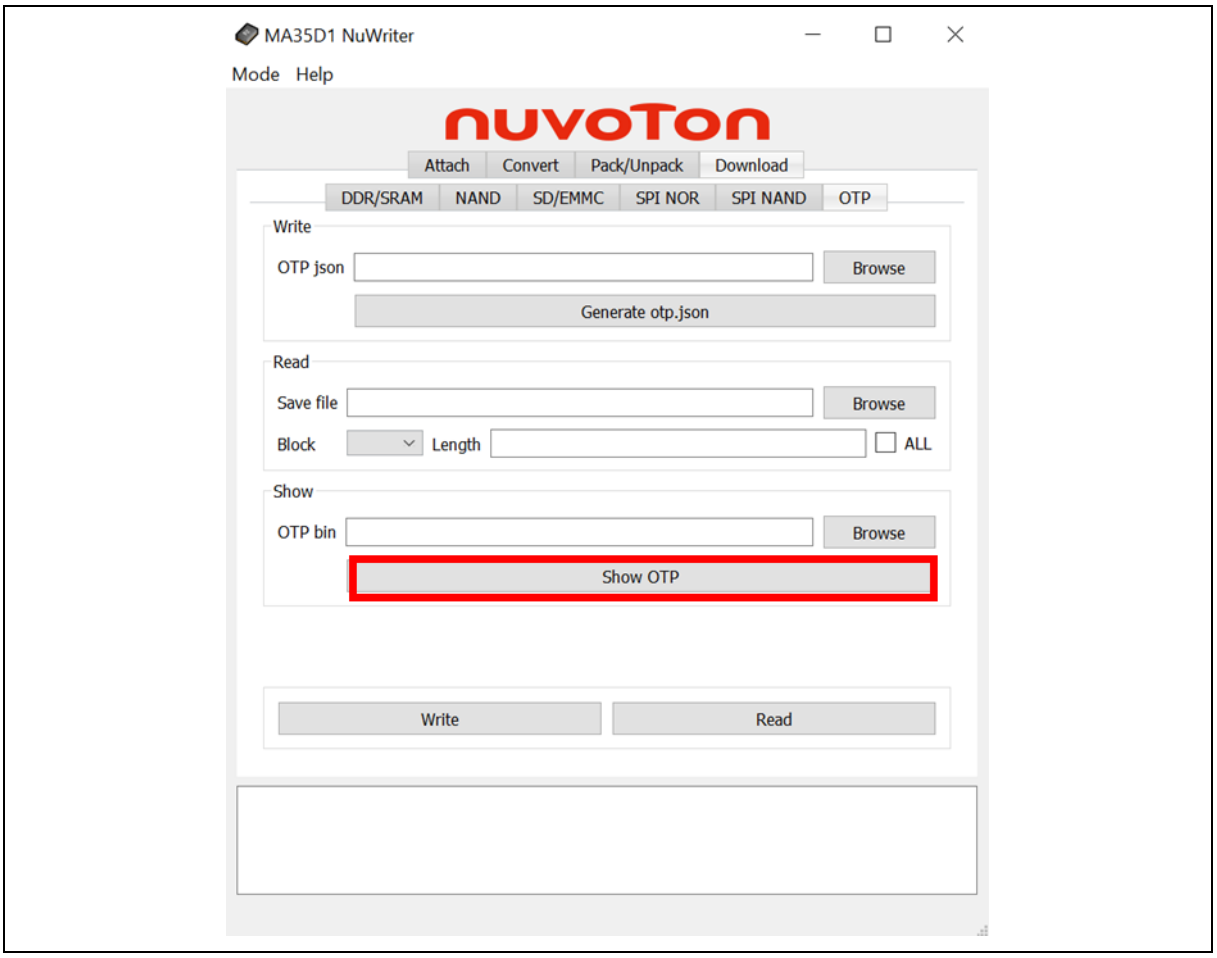

Figure 7-18 Show OTP Example -1

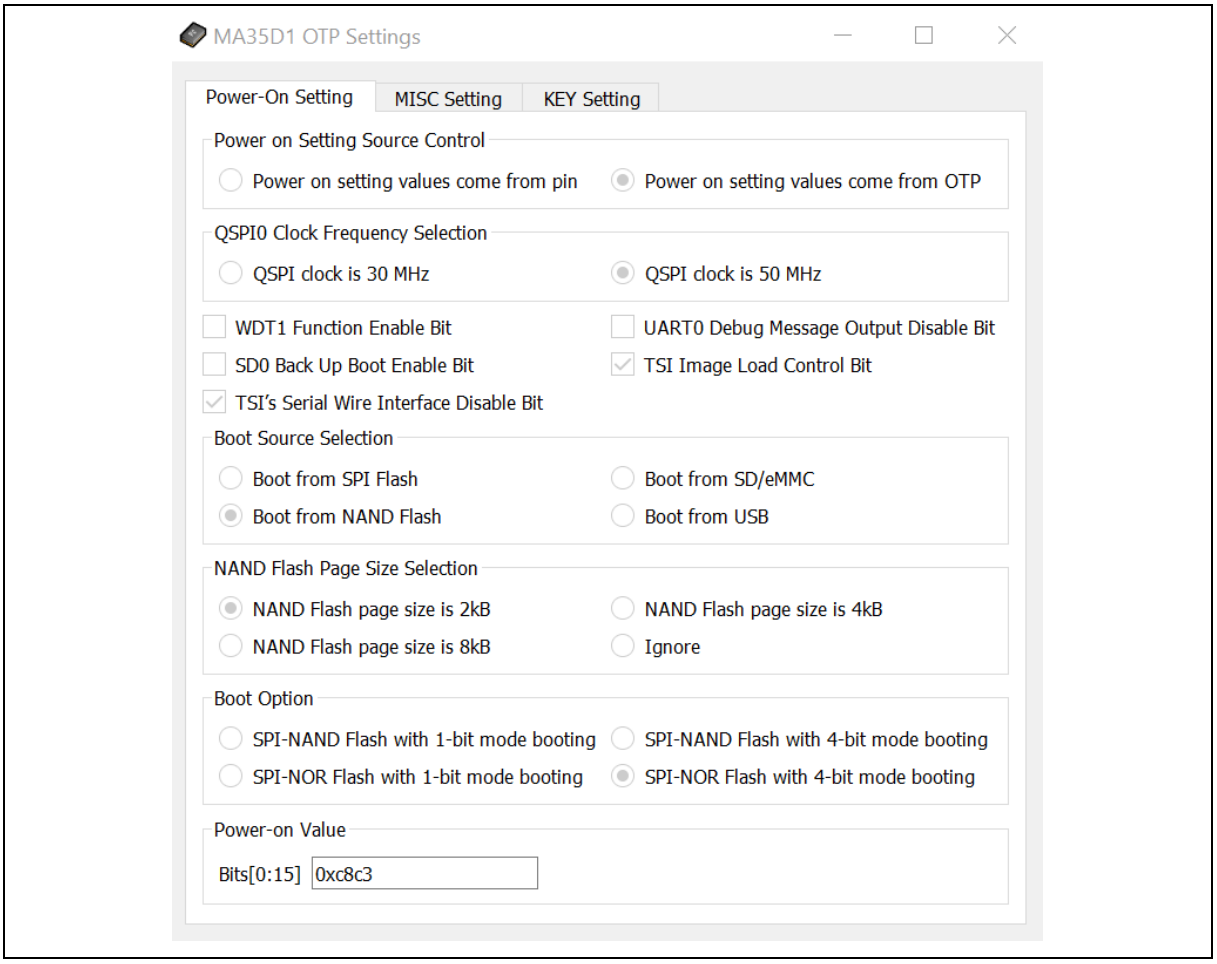

Figure 7-19 Show OTP Example -2

7.3 Convert

In the “Convert” page, this is an offline function, no needs to connect with MA35D1 platform. User can convert the MA35D1 platform image header file by exist file or create new one by NuWriter. For more detail, please reference previous chapter.

Click “Browse” button, select the existent header.json file. Click “Convert” button, the header.json file will convert to binary for MA35D1 IBR used.

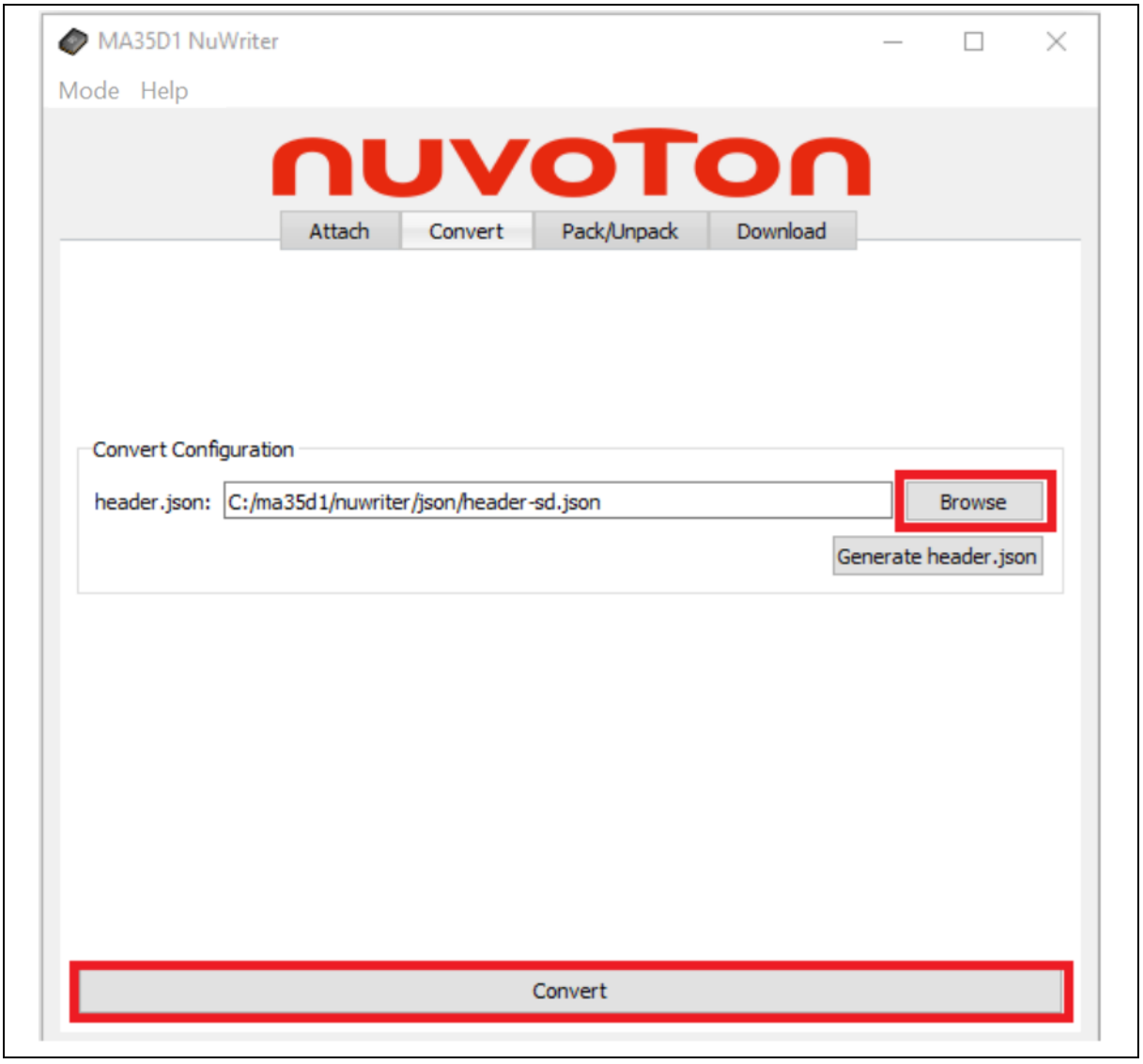

Figure 7-21 Convert Window

Create a new header.json file. Click “Generate header.json” button, set the header information – header version, SPI information, entry point, images, … etc. After all setting are complete, user can save the header format to json file by “Export” button.

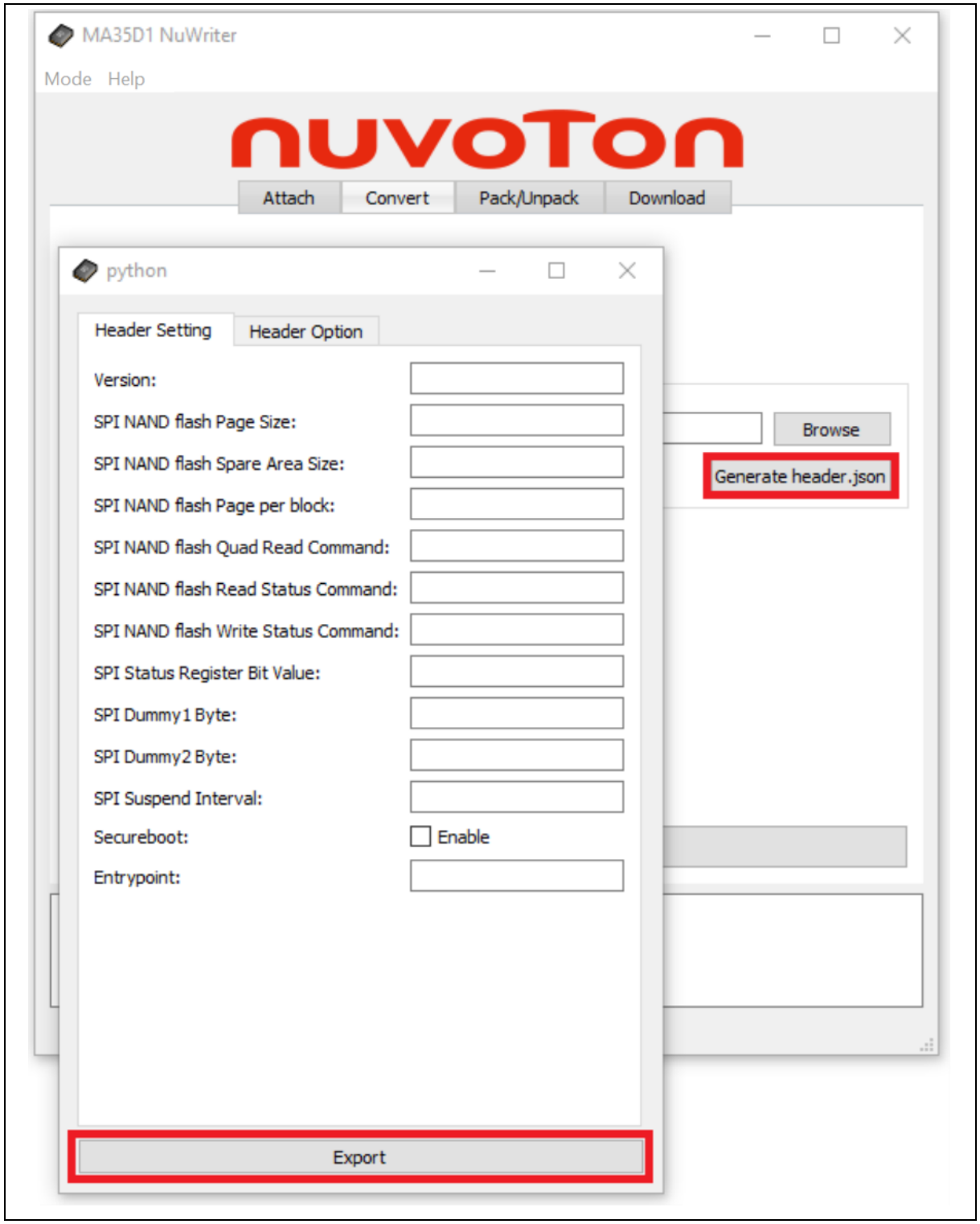

Figure 7-20 Generate header file

7.4 Pack/Unpack

The “Pack/Unpack” page is an offline function, no needs to connect with MA35D1 platform. Please reference the 6.4 Pack chapter for more detail. Click “Browse” button, select existent pack.json or input the image count to create a new pack.json. Click “Generate pack.json” button, fill all image information. Click “Pack” button to pack all images in one binary file.

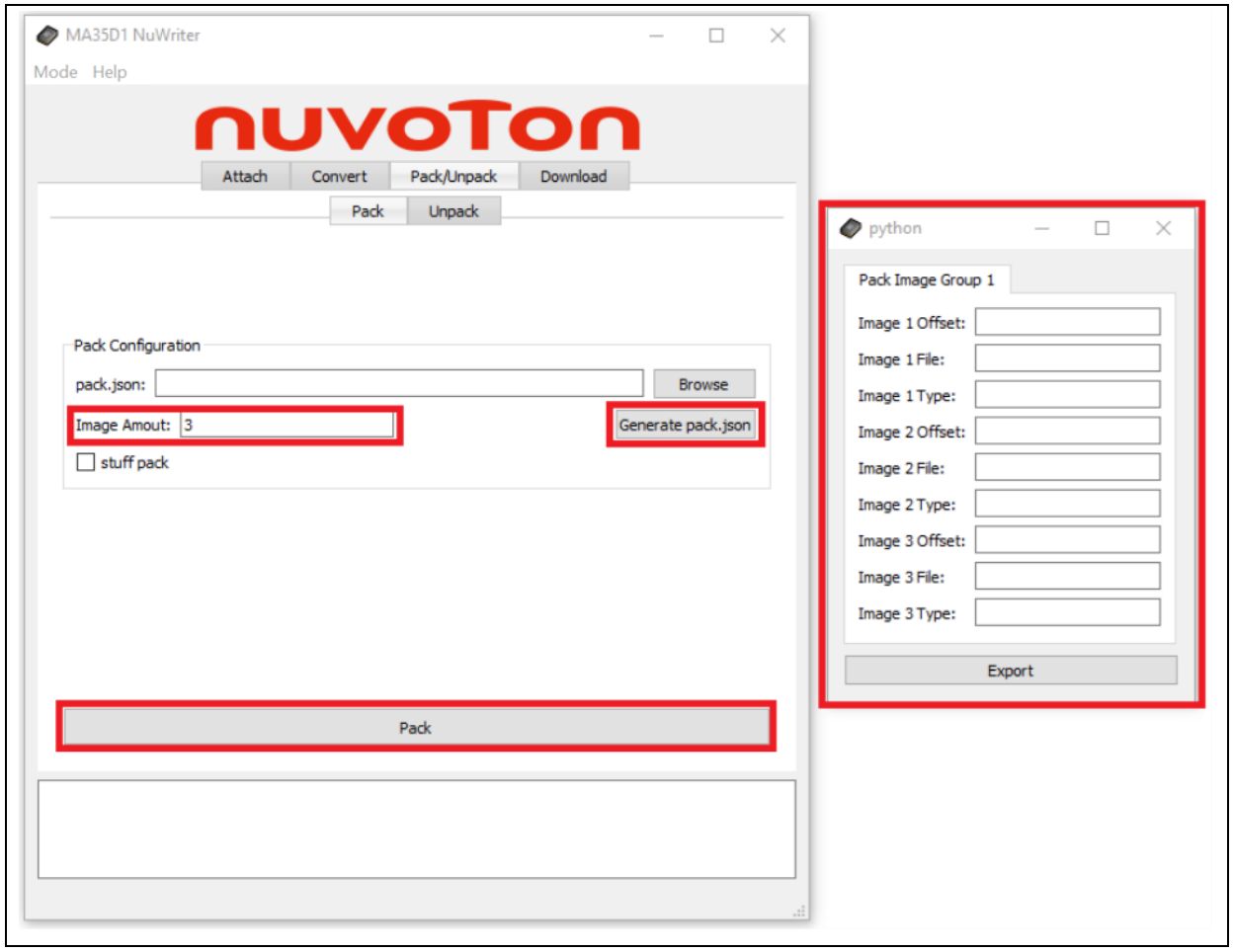

Figure 7-21 Pack Window

Click “Browse” button, select existent pack.json. Click “Unpack” button to extract all images.

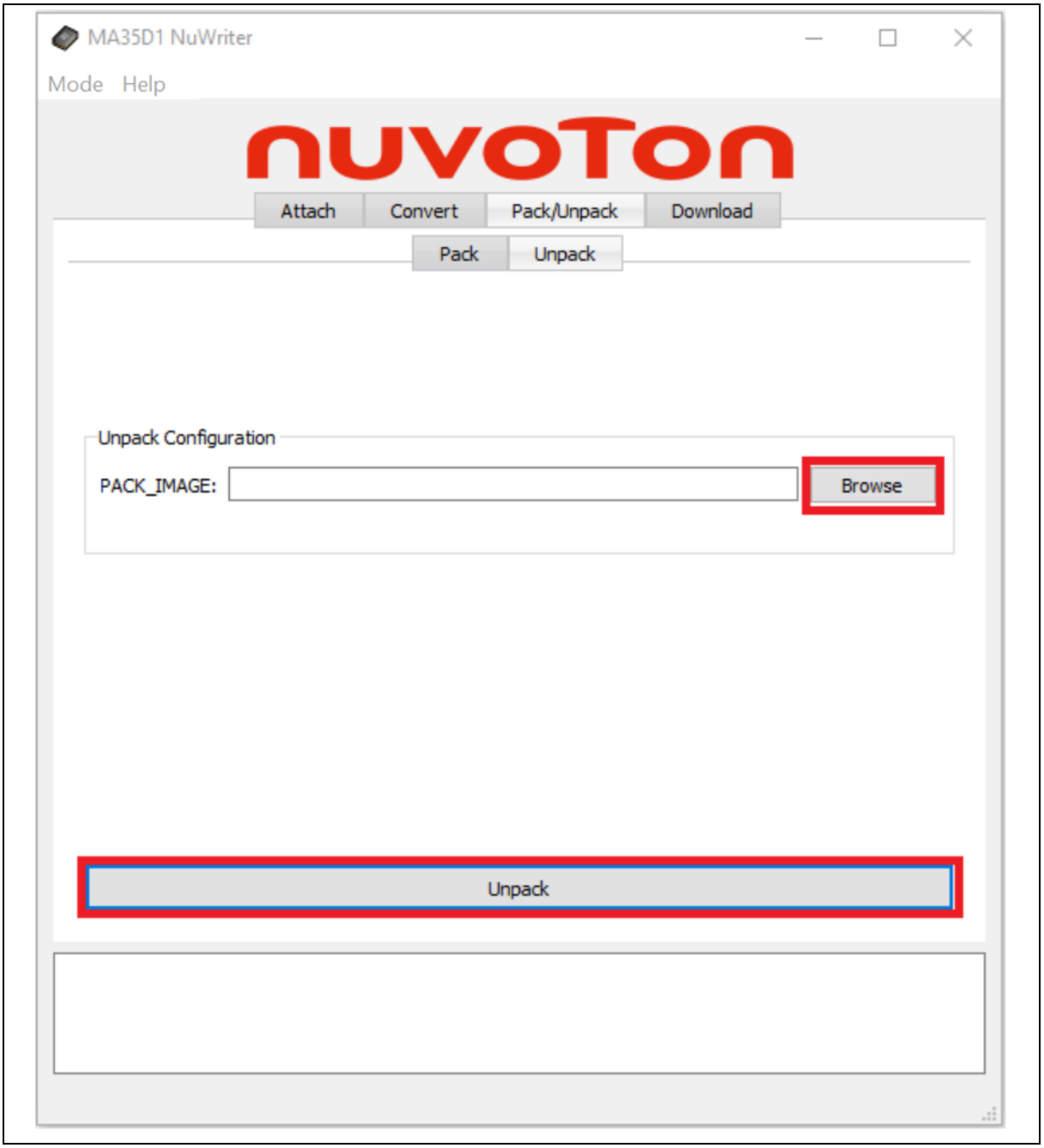

Figure 7-22 Unpack Window

7.5 Program One File

MA35D1 BSP supports two environments – Buildroot and Yocto. When build the image, it will generate one pack file. It includes all needed files. Such as TFA BL2, SMC BL31, secure OS BL32, uboot BL33 and linux kernel. NuWriter supports to program the pack file directly. If customer needs to wrote only one file to storage. He can get the image offset by refer to the pack.json.

In MA35D1 BSP architecture, the BL31, BL32 and BL33 will combine to one file, named FIP (Firmware Image Package). If customer only rebuild the uboot, he also needs to rebuild the TFA to get the new FIP. And then write the FIP to storage. For more detail, please refer to 'NuMicro® Family MA35D1 TF-A User Manual'.

Use NAND for example. Please refer pack-nand.json. In yocto, the file path is (*build/tmp-glibc/deploy/images/numaker-som-ma35d16a81/nuwriter/pack-nand.json*). In buildroot, the file path is (*buildroot/board/nuvoton/ma35d1/nuwriter/pack-nand.json*). The content of pack-nand.json is as follow.

```
{
    "image":
    [
        {
          "offset": "0x000000",
          "file": "conv/header.bin",
          "type": 1
        },
        {
          "offset": "0xC0000",
          "file": "bl2-ma35d1.dtb",
          "type": 0
        },
        {
          "offset": "0xE0000",
          "file": "bl2-ma35d1.bin",
          "type": 0
        },
        {
          "offset": "0x100000",
          "file": "fip.bin-nand",
          "type": 1
        },
        {
          "offset": "0x300000",
          "file": "u-boot-initial-env.bin-nand",
          "type": 1
        },
        {
          "offset": "0x3C0000",
          "file": "Image.dtb",
          "type": 1
        },
        {
          "offset": "0x400000",
          "file": "Image",
          "type": 1
    },
        {
          "offset": "0x1C00000",
          "file": "rootfs.ubi-nand",
          "type": 1
        }
    ]
}
```

```
{
    "image":
    [
        {
          "offset": "0x000000",
          "file": "header.bin",
          "type": 1
        },
        {
          "offset": "0xC0000",
          "file": "bl2.dtb",
          "type": 0
        },
        {
          "offset": "0xE0000",
          "file": "bl2.bin",
          "type": 0
        },
        {
          "offset": "0x100000",
          "file": "fip.bin-nand",
          "type": 1
        },
        {
          "offset": "0x300000",
          "file": "uboot-env.bin-nand",
          "type": 1
        },
        {
          "offset": "0x3C0000",
          "file": "Image.dtb",
          "type": 1
        },
        {
          "offset": "0x400000",
          "file": "Image",
          "type": 1
    },
        {
          "offset": "0x1C00000",
          "file": "rootfs.ubi",
          "type": 1
        }
    ]
}
```

Figure 7-23 Pack-nand.json of yocto and buildroot

Update "fip.bin-nand" (SMC BL31, Secure OS BL32 or Uboot BL33). Follow the steps listed below:

1. Select "NAND".

2. Browse the Image file "fip.bin-nand" by click the "Browse" button.

3. Select Image type “Data”.
4. Enter the image address – 0x100000. The offset in the storage.
5. Click “Write”.

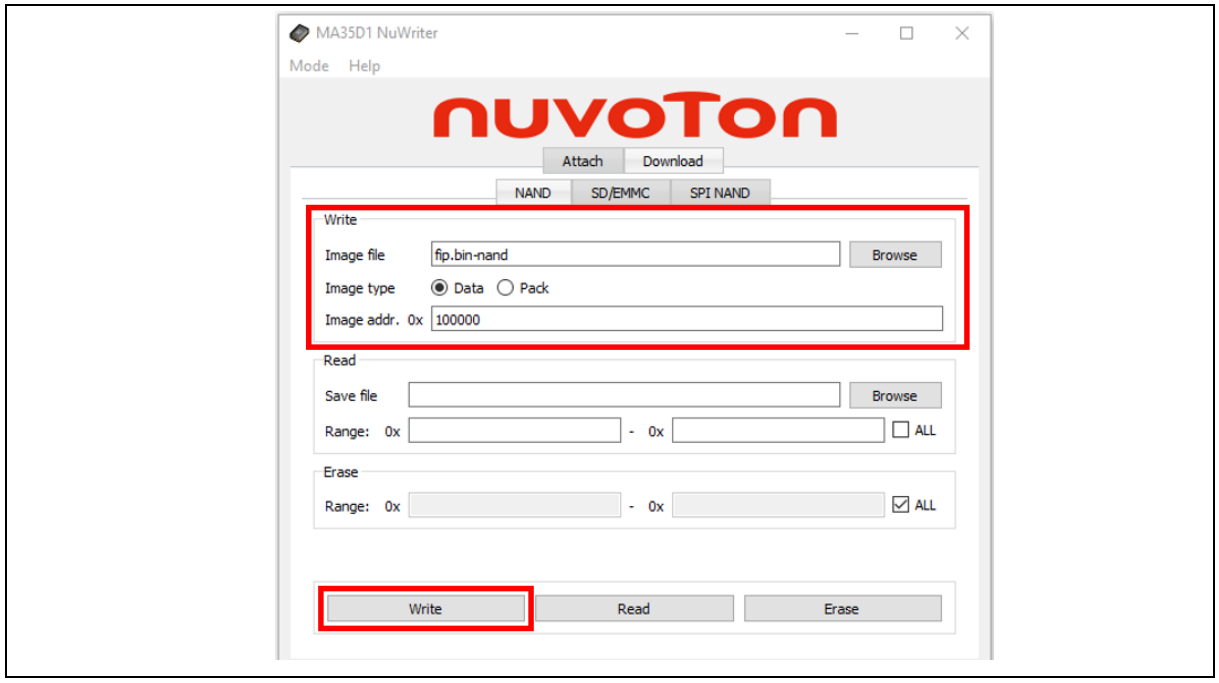

Figure 7-24 Write fip.bin-nand

Update “Image” (linux kernel). Follow the steps listed below:

1. Select “NAND”.
2. Browse the Image file “Image” by click the “Browse” button.
3. Select Image type “Data”.
4. Enter the image address – 0x400000. The offset in the storage.
5. Click “Write”.

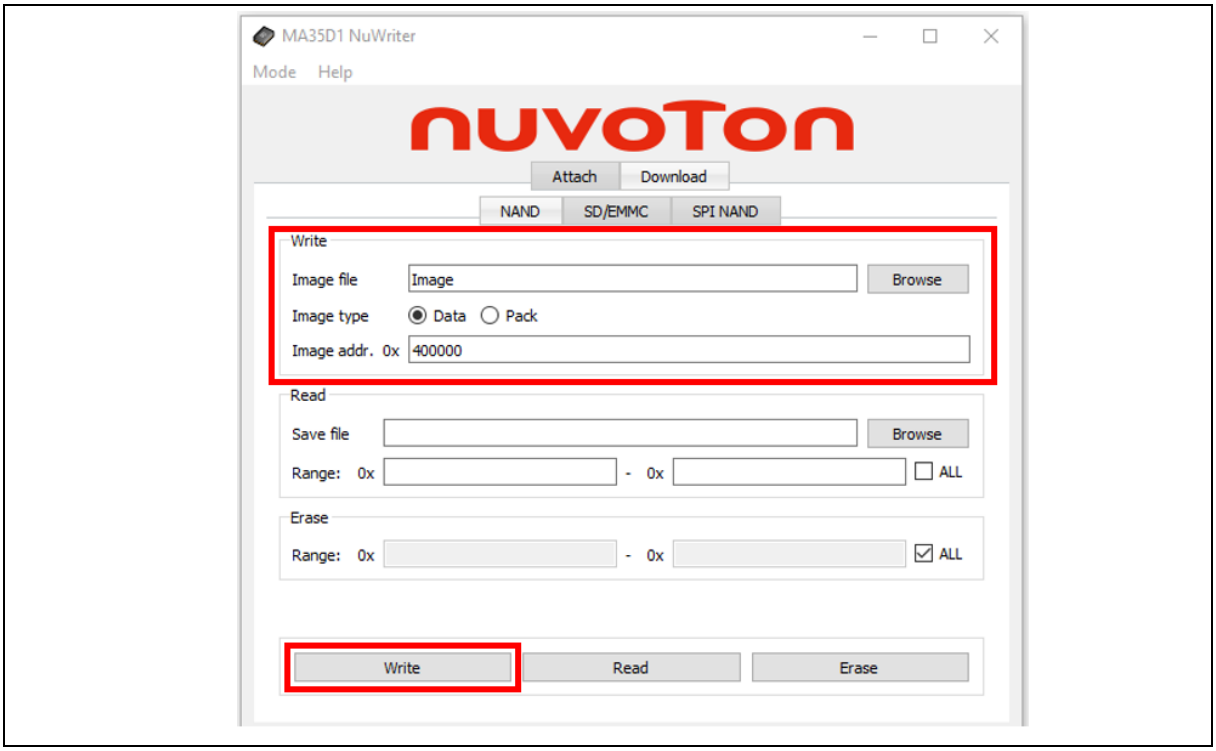

Figure 7-25 Write fip.bin-nand

Update “bl2.bin” (TFA BL2). Follow the steps listed below:

1. Select “NAND”.
2. Browse the Image file “bl2.bin” by click the “Browse” button.
3. Select Image type “Data”.
4. Enter the image address – 0xE0000. The offset in the storage.
5. Click “Write”.

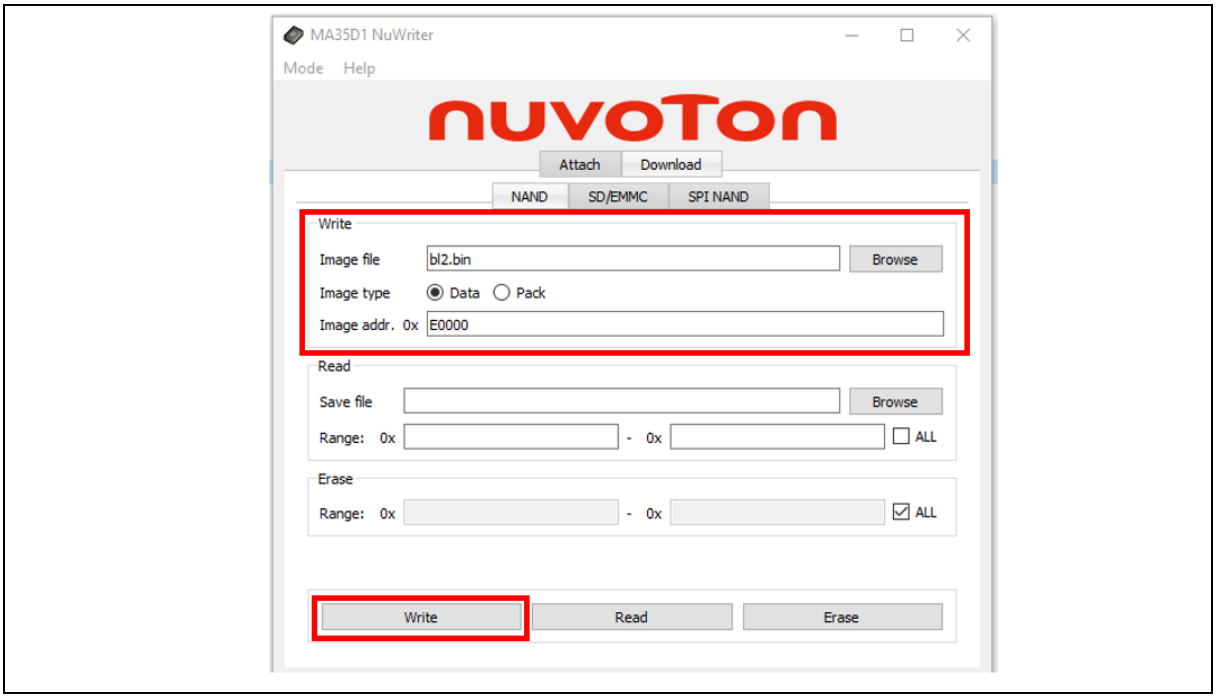

Figure 7-26 Write fip.bin-nand

8 REVISION HISTORY

Date	Revision	Description
2022.07.15	1.02	1. Add driver install 2. Add Program one file
2022.05.30	1.01	1. Update NuWriter UI 2. Fixed typo and made minor modifications
2022.05.18	1.00	1. Preliminary issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.