NuMicro® Family
Arm® Cortex®-A35- based Microprocessor

NuMicro® Family
MA35H0 Non-OS BSP
User Manual

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

www.nuvoton.com

Table of Contents

1 OVERVIEW

The NuMicro MA35H0 series is a high-performance microprocessor targeted to industrial HMI applications. It features dual 64/32-bit Arm Cortex-A35 cores. The Arm Cortex-A35 cores offer powerful performance, running at speeds of up to 650 MHz, and each core has a 32/32 KB I/D L1 cache, while there is a shared 512 KB L2 cache.

The MA35H0 is based on Cortex-A35 Armv8-A 64-bit architecture, with CPU clock speed up to 650 MHz, and is equipped with DDR and rich high-speed peripheral device support, making it suitable for development on the Linux operating system. Nuvoton also provides a complete Linux development BSP solution. However, some people may only need high-speed computing and powerful CPU, and have no desire to develop in a relatively complex Linux environment. In this case, this BSP can meet their needs, as it provides a relatively simple development environment and programming requirements, allowing users to easily and quickly accomplish what they want to do.

Chapter 2 introduces the implementation information of this BSP. Chapter 3 describes the software directory structure and sample code. Chapter 4 introduces how to setup the development environment. Chapter 5 introduces different method to load and execute MA35H0 executable binary.

This Non-OS BSP includes following contents:

- MA35H0 Non-OS drivers.
- Sample code for each supported peripherial devices
- Third-party library
- User manuals.

The MA35H0 Non-OS BSP can be downloaded from OpenNuvoton/MA35H0_NonOS_BSP.

2 BAREMENTAL IMPLEMENTATION

This chapter introduces important implementation information about the non-OS BSP, including CPU clock, cache, memory configuration, and peripherals. Reading this chapter will help understand the capabilities and limitations of this BSP, and whether it is sufficient to support the development of the target application. If you find that these features are not sufficient to help you complete the development of your target application, then it is recommended to consider using the MA35H0 Linux BSP.

2.1 System Management

The section explains the core issues related to the MA35H0 platform.

2.1.1 Cortex-A35 Architecture

The MA35H0 is based on the Cortex-A35 ARMv8-A architecture, which supports Non-secure and Secure modes, and is divided into exception levels EL0 to EL3, as shown in Figure 2.1.

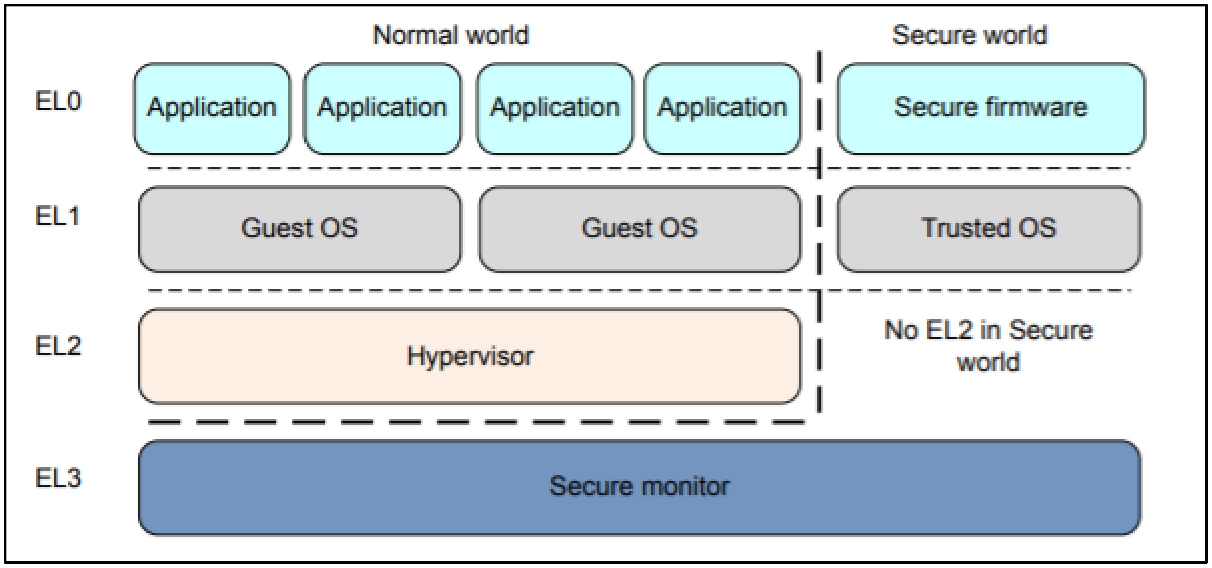

Figure 2-1 ARMv8-A Exception Levels

For ease of use and practicality, the initial code and all sample code of this BSP run in secure-EL3 mode. While there are APIs available to switch modes, doing so can lead to issues with interrupt handling and access permissions. Therefore, it is not recommended to switch modes and this BSP is not designed for that purpose.

Although the MA35H0 supports secure boot, this BSP currently does not provide an implementation e xample. Please refer to the *MA35 Series NuWriter User Manual* for how to implement it.

2.1.2 System Clock and PLL

The MA35H0 features SYS-PLL, CA-PLL, DDR-PLL, EPLL, APLL, and VPLL, which are generated from an external HXT source to produce the output frequency of each PLL.

SYS-PLL is set to a frequency of 180 MHz by the MA35H0's IBR (Internal Boot ROM) firmware. The initial code of the BSP does not modify this setting.

DDR-PLL is set by the IBR to execute the DDR initialization program, which occurs before the BSP program is loaded. Therefore, the BSP typically does not modify the DDR-PLL settings but instead keeps them at the settings of the DDR initialization code, which is usually 266 MHz.

The MA35H0 IBR sets the CA-PLL to 500 MHz as the CPU clock source. The initial code of the BSP adjusts the CPU frequency to 650 MHz.

As for the EPLL, VPLL, and APLL, the initial code of the BSP does not set or enable them. Individual sample codes that require the use of these PLLs will make the necessary settings and enable them. For example, the Display sample code uses EPLL and VPLL, while the I²S audio or SDHC sample code requires the use of APLL.

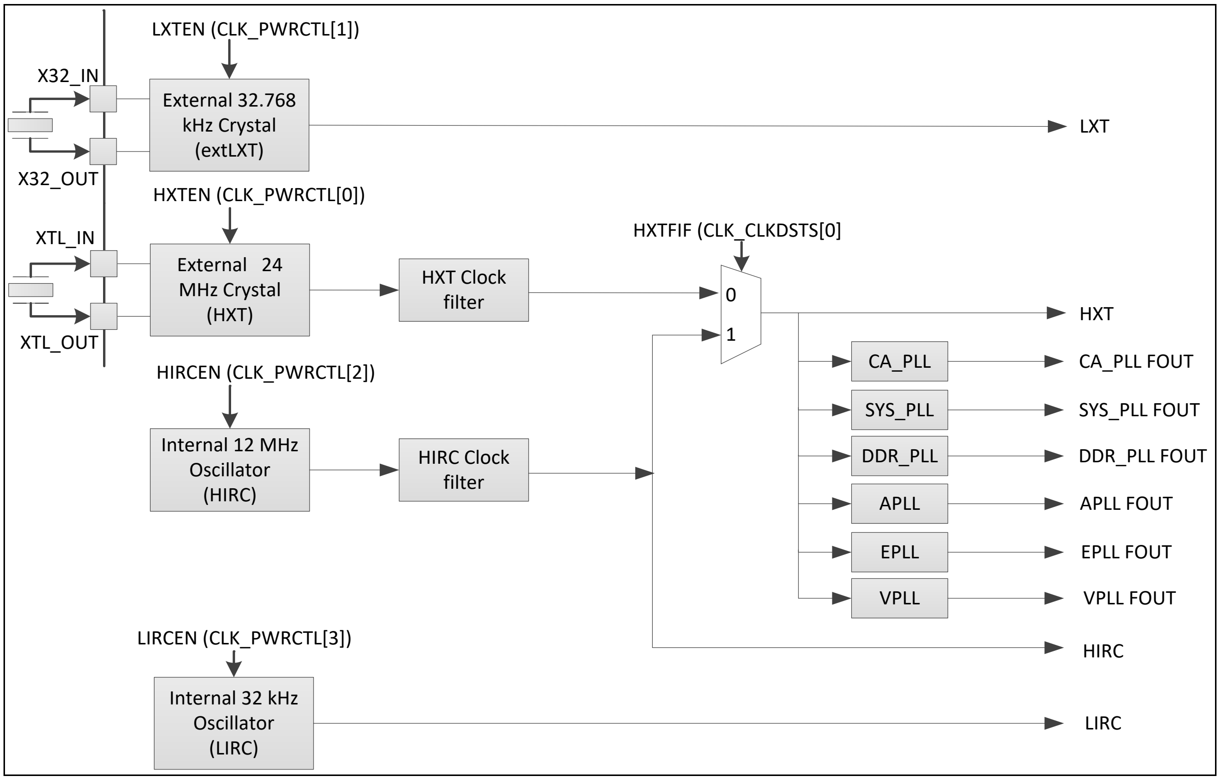

Figure 2-2 MA35H0 Clock Tree

2.1.3 System Memory

The MA35H0 system memory includes DRAM and SRAM1. In order to provide easy system design and manufacture, the MA35H0 series have built-in DRAM with sizes of 128 MB. In the MA35H0 platform, the address space of the DRAM starts from 0x80000000 as a continuous space. All sample codes of the BSP are set to execute at the address of 0x80000000.

There are 256 KB on-chip SRAM located between 0x28000000 and 0x2803FFFF.

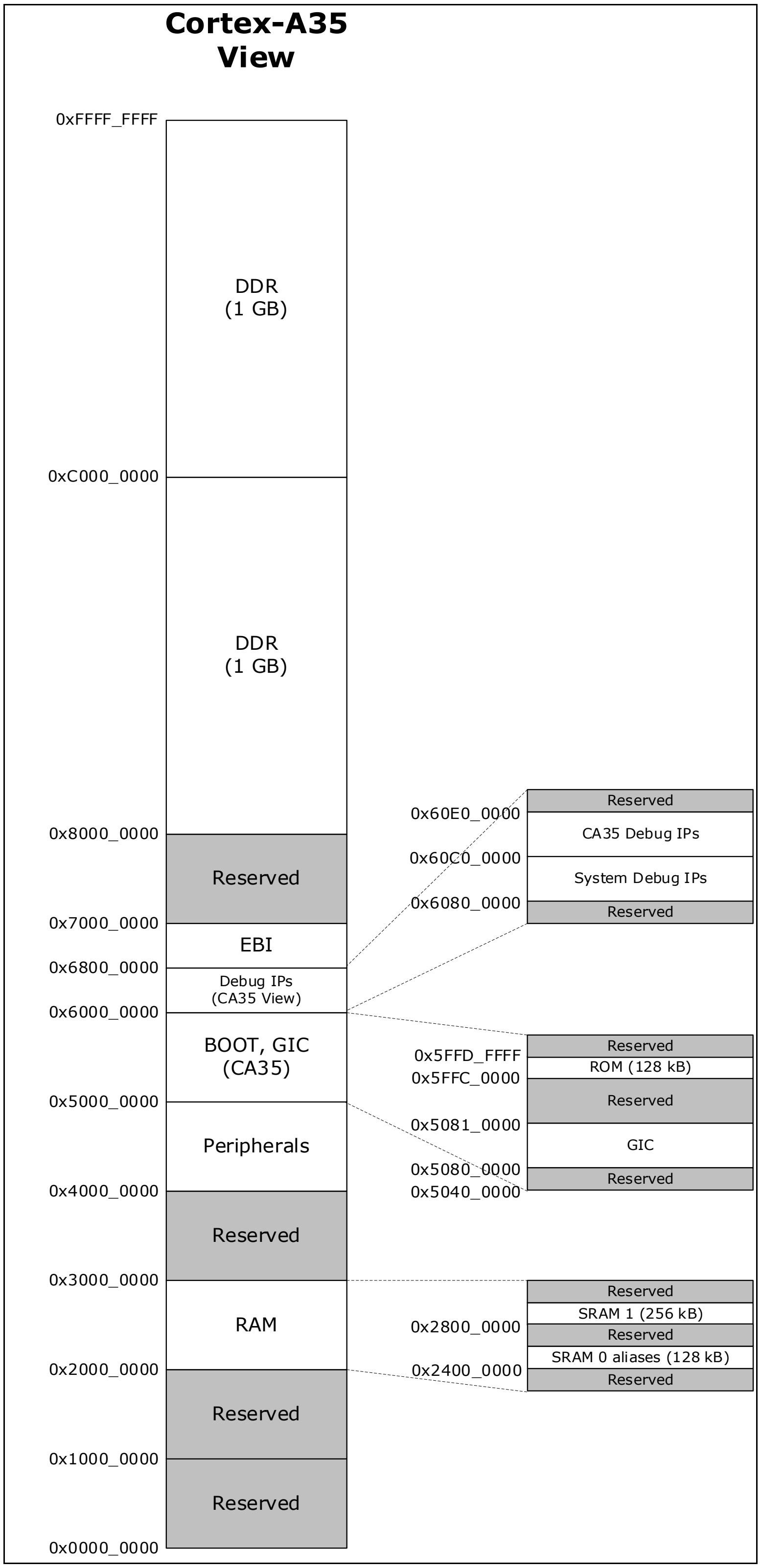

Figure 2-3 MA35H0 System Memory Map

2.1.4 Cache Memory

The MA35H0 has a 512 KB L2 cache. To simplify MMU management and usage, this BSP uses the same approach as the NUC970/NUC980 ARM9 series MPU, using a single bit to distinguish between cache and non-cache memory. In NUC970/NUC980, bit 31 is used, while the MA35H0 uses bit 32.

When bit 32 is set to 1, Cortex-A35 performs non-cache memory access. Conversely, it is cacheable memory access. For example, when accessing the address 0x180900AC0, it is non-cache memory access, and the CPU directly accesses the DRAM memory without involving the cache. However, when accessing the address 0x80900AC0, it goes through the cache memory.

When using controllers with DMA functionality, such as USB Host, USB Device, SDHC, NAND, Display, and VC8000, if Cortex-A35 CPU accesses memory provided for these controllers to use for DMA, it is necessary to avoid the use of cache memory because the cache memory controller cannot know when the peripheral controller reads or writes to memory using DMA, so it cannot synchronize with the actual content of memory.

The following code is taken from the HSUSBH_Mass_Storage sample code.

```
BYTE  Buff_Pool1[BUFF_SIZE] __attribute__((aligned(32)));  /* Working buffer 1 */
BYTE  Buff_Pool2[BUFF_SIZE] __attribute__((aligned(32)));  /* Working buffer 2 */
BYTE  *Buff1;
BYTE  *Buff2;
…
Buff1 = nc_ptr(Buff_Pool1);
Buff2 = nc_ptr(Buff_Pool2);
```

In this example, Buff_Pool1 and Buff_Pool2 are used as DMA buffers for the USB Host Mass Storage driver, which will be accessed by both the Cortex-A35 CPU and the USB Host controller DMA. Therefore, non-cache access must be used to avoid issues where the CPU does not actually read the content of the memory or where data to be written is not properly flushed to memory. These issues can cause errors during USB transmission due to incorrect data content.

In this code, Buff1 and Buff2 are used to obtain the non-cache memory addresses of Buff_Pool1 and Buff_Pool2 through the nc_ptr() macro. As long as the Cortex-A35 CPU reads and writes to these memory addresses, the cache memory will not be involved.

.

2.2 Dual Core Support

The MA35H0 features a dual-core Cortex-A35 CPU and this BSP supports dual-core. When designing dual-core applications, Cortex-A35 Core 0 enters execution from main(), while Cortex-A35 core 1 enters execution from main1(). The BSP sample code *SYS_DualCore* demonstrates simultaneous execution of Cortex-A35 core0 and core1.

Programmers must properly allocate system resources and avoid Core 0 and Core 1 using the same IP and peripheral devices. Memory usage should also be properly separated to prevent the memory contents from being changed by another CPU without detection. If Core 0 and Core 1 must share a system resource, such as console port UART0, mutex protection must be added to avoid various problems that may arise.

The following code snippet is taken from this BSP's *Library\StdDriver\src\retarget.c*, which uses the cpu_spin_lock() API to prevent simultaneous execution of sysprintf by Cortex-A35 core0 and core1. This outputs messages from the hardware UART0, and if not protected, will cause confusion in accessing UART0 registers.

```
#ifdef DEUBG_PORT_ONE_ONLY
static unsigned int mutex_print=0;
```

```
#endif
void sysprintf(char * pcStr,...)
{
       char  *argP;
       va_list va;

#ifdef DEUBG_PORT_ONE_ONLY
       cpu_spin_lock(&mutex_print);
#ifdef DEUBG_SHOW_CORE_INFO
       _PutChar_f((char)(0x30+cpuid()));
       _PutChar_f(':');
#endif
#endif
    va_start(va, pcStr);  /* point at the end of the format string */
    while (*pcStr) {
       /* this works because args are all ints */
        if (*pcStr == '%')
            pcStr = FormatItem(pcStr + 1, va_arg(va, int));
        else
            _PutChar_f(*pcStr++);
    }
    va_end(va);
#ifdef DEUBG_PORT_ONE_ONLY
    cpu_spin_unlock(&mutex_print);
#endif
}
```

Communication between Core0 and Core1 can be achieved through MA35H0's Hardware Semaphore or by designing a shared memory and carefully designing the communication protocol between the different CPU cores.

2.3 SSPCC

The System Security Peripheral Configuration Controller (SSPCC) is utilized to configure the security attributes of SRAM, GPIO, and other peripherals. The SSPCC also collects the security violation responses of the peripherals and generates interrupts when a violation event occurs. Furthermore, the SSPCC includes the Debug Port Manager (DPM), a module that controls the debug access of this chip, as well as the Pre-defined Life-cycle Manager (PLM), which leverages hardware-based controls to manage the ability of certain functions based on predefined life-cycle stages.

This BSP does not modify the SSPCC and retains its reset default settings. Users need to allocate the peripherals properly based on the actual application requirements.

2.4 File System

MA35H0 is equipped with several storage-related controllers, such as SDHC, NAND, SPI/QISP, USB, etc. In order to access the contents of storage devices when using SD cards or USB disks, file system support is required. Currently, this BSP only supports two file systems, FAT32 and YAFFS2.

FatFs is an open-source FAT32 file system, and the USB Host and SDHC drivers in this BSP both support FatFs. SD cards and USB disks must be formatted as FAT32 file systems. If other file systems such as NTFS or ext4 are used, the USB Host library and SDHC driver in this BSP will only be able to access the raw data on the storage device and will not be able to access the file system contents.

YAFFS2 has been widely adopted in embedded systems that require a reliable and efficient file system for NAND flash memory, including automotive, industrial, and medical devices. The NAND driver in this BSP only supports the Yaffs2 file system.

2.4.1 FATFS

FatFs is a generic file system module that supports the FAT file system format, which is commonly used in storage devices such as USB flash drives, SD cards, and hard disk drives. It was developed by ChaN, a Japanese engineer, and is written in ANSI C, making it highly portable and compatible with a wide range of microcontrollers, embedded systems, and operating systems.

The FatFs source code is included in the *ThirdParty* directory of this BSP without any modifications. The USB Host Mass Storage Class driver and SDHC driver in this BSP are designed with interface functions for FatFs, and usage examples can be found in the *SampleCode* directory. For example, the *HSUSBH_Mass_Storage* sample code in *SampleCode\StdDriver* demonstrates various file system operations on a USB disk using a command-line interface.

2.4.2 YAFFS2

YAFFS2 (Yet Another Flash File System 2) is a file system developed specifically for NAND Flash. In addition to normal file system functionality, embedded flash file systems should help devices boot and shut down quickly and handle power loss well without warning, without damaging data, and protect stored data with extremely high reliability. Therefore, embedded flash file systems must take additional steps to achieve these goals, such as using log structures instead of traditional file structures plus flash conversion layers; managing power loss and recovery; including flash wear leveling; handling flash interference-induced unreliability by using error correction codes (ECC), bad block remapping, over-provisioning, and flash block refreshing; preparing for fast writes by using garbage collection and flash block erasure management, while managing writes to avoid write amplification.

Memory in NAND Flash is arranged in pages. Pages are the units that are allocated and programmed. In Yaffs, the unit that is allocated is called a block. Usually, one block will be the same as one NAND page, but blocks can be flexibly mapped to multiple pages. A fixed but configurable number of blocks make up a chunk, and chunks are the units that are erased. NAND flash comes with bad blocks, and other blocks may fail during device operation, so Yaffs detects and marks bad blocks. NAND flash typically also requires some kind of error detection and correction code (ECC). Yaffs can use existing ECC logic or provide its own logic.

A Yaffs2 source code is placed in the ThirdParty directory of this BSP and has been ported to the ma35h0 NAND Control Interface. This BSP provides the NAND_Yaffs2 application example.

2.5 Networking

The MA35H0 provides one ethernet media access control (EMAC) interfaces compatible with RMII PHY interface. The EMAC driver in the non-OS BSP (Board Support Package) is designed to be compatible with lwIP (lightweight IP), an open-source project that provides a small TCP/IP stack for embedded systems. A range of network protocols, including TCP, UDP, IP, ICMP, ARP, DHCP, DNS, and PPP, are supported.

The provided sample codes include two applications: DHCP and iperf. The BSP sample code *EMAC_dhcp* demonstrates how a network device can obtain an IP address from a DHCP server. The sample code *lwIP_iperf_Server* is used to measure network performance. The transfer rate and duplex settings are determined through negotiation between the local and partner Ethernet PHY.

Please note that the transfer rate and duplex settings are determined dynamically based on the negotiation process between the local and partner Ethernet PHY.

2.6 Multimedia Support

2.6.1 Display

The MA35H0 display controller provides two examples. The first example is *DISP_Framebuffer*, which demonstrates how to display input data formats ARGB888 and RGB565 on a synchronous LCD panel. The example initializes the LCD panel, which requires setting the vertical and horizontal timing parameters according to the LCD panel specification. Then, setting the pixel clock of the display controller, selecting the multifunction pin, and configuring the display controller can display the image on the panel.

The MA35H0 display controller comes with an Overlay layer. By opening the Overlay and setting the upper, lower, left, and right coordinates of the Overlay coverage area, the layer can be superimposed on the video framebuffer. The MA35H0 display controller provides two methods, pixel Alpha Blending and Color Key, to control the transparency effect of the overlay layer in the coverage area. Sample code *DISP_Overlay* demonstrates the transparency effect of the coverage area through these two methods.

2.6.2 Touch

The MA35H0 provides an 8-input channel ADC module with 12-bit resolution. It supports two modes: normal ADC mode and 4-wire touchscreen mode. The MA35H0 non-OS BSP includes sample codes for both modes. In the *ADC_Calibrate* sample code, there are calibration and drawing functions that utilize the open-source touchscreen library tslib to integrate the display with the touchscreen. Users can calibrate the touchscreen according to the corresponding display resolution. Once the calibration data is determined, they can freely start drawing on the screen.

2.6.3 Audio

The MA35H0 audio application uses I^2C to configure NAU88C22 audio codec and I^2S for audio data transfer with NAU88C22 audio codec. In order not to spend all CPU resources on moving audio data between I^2S Tx/Rx FIFO and DDR, you can enable PDMA to handle large amount of data moving.

MA35H0 BSP also provides MP3 library, LibMAD, for MP3 playback.

2.6.4 H264 and JPEG Decode

The MA35H0 VC8000 video decoder has H264 and JPEG decode capabilities, and supports post-processing functionality to directly output H264 or JPEG decode results for post-processing and finally output specified image format. For example, a 720P YUV420 format H264 video can be set up with H264 + PP (Post-Processing) to decode H264 bit stream and output 1024 x 600 RGB888 format images.

It should be noted that H264 and JPEG share the same decode engine, so only one can be executed at the same time.

This BSP provides the VC8000 library, and a simplified API designed for users to use. H264 decoding requires a considerable amount of memory space, depending on the user's needs. When calling VC8000_Init() to initialize the VC8000 Library, the user needs to allocate sufficient memory space for the VC8000 Library to use. If supporting H264 decode resolution up to 1280x800, it is recommended to provide 32 MB of memory space. If the provided memory space is insufficient, it may cause video playback failure.

The sample code *VC8000_JpegDecodeFiles* demonstrates reading JPEG files from a USB disk, decoding and displaying them on the LCD screen. The sample code *VC8000_H264DecodeFiles* demonstrates reading H264 bit stream files from a USB disk, decoding them and directly outputting to the LCD for playback.

3 DIRECTORY STRUCTURE

This chapter describes the directory structure of MA35H0 Non-OS BSP software package.

3.1 First Level Directory Information

Directory	Description
Document	Driver reference guide and revision history.
Library	Driver header and source files.
SampleCode	Driver sample code.
ThirdParty	Library from third party.

Table 3-1 First Level Directory

3.2 Document

Directory	Description
UM_EN_MA35H0_Non-OS_BSP.pdf	The user manual of MA35H0 series non-OS BSP.
Revision History.pdf	The revision history of MA35H0 non-OS Series BSP.
NuMicro MA35H0 Driver Reference Guide.chm	This document describes the usage of drivers in MA35H0 software package.

Table 3-2 Document Directory

3.3 Library

Directory	Description
Arch	Cortex-A35 architecture driver and header.
Device	MA35H0 register header files and startup code
DisplayLib	Display library binary and header file.
SmartcardLib	Smartcard library binary and header file.
StdDriver	All peripheral driver header and source files.

UsbHostLib	USB Host library and class drivers source code
VC8000Lib	VC8000 H264 and JPEG decode library

Table 3-3 Library Directory

3.4 SampleCode

Directory	Description
AmpCore1	A project template for Core1 Only.
Crypto	Crypto sample code
FreeRTOS	FreeRTOS sample code ported from FreeRTOS official samples.
Loader	Boot loader sample code for custom DDR.
lwIP	lwIP sample code ported from lwIP samples.
StdDriver	Demonstrate the usage of MA35H0 series MPU peripheral driver APIs.
Template	A project template for MA35H0.

Table 3-4 SampleCode Directory

3.5 ThirdParty

Directory	Description
FatFs	An open source FAT/exFAT file system library. A generic FAT file system module for small embedded systems. Its official website is: http://elm-chan.org/fsw/ff/00index_e.html.
FreeRTOS-Kernel	FreeRTOS the real-time operating system (RTOS) kernel. The FreeRTOS kernel offers features such as task scheduling, inter-task communication, synchronization primitives, and memory management. Its official website is: https://www.freertos.org.
LibMAD	A MPEG audio decoder library that currently supports MPEG-1 and the MPEG-2 extension to lower sampling frequencies, as well as the de facto MPEG 2.5 format. All three audio layers — Layer I, Layer II, and Layer III (i.e. MP3) are fully implemented. This library is distributed under GPL license. Please contact Underbit Technologies for the commercial license.

lwIP	A widely used open source TCP/IP stack designed for embedded systems. Its official website is: http://savannah.nongnu.org/projects/lwip/.
mbedtls-3.1.0	mbed TLS offers an SSL library with an intuitive API and readable source code, so you can actually understand what the code does. Also the mbed TLS modules are as loosely coupled as possible and written in the portable C language. This allows you to use the parts you need, without having to include the total library. Its official website is: https://tls.mbed.org/.
paho.mqtt.embedded-c	The Eclipse Paho project provides open-source client implementations of the MQTT (Message Queuing Telemetry Transport) protocol. One of these implementations is the Paho MQTT Embedded-C client, designed for embedded systems and resource-constrained environments. MQTT is a lightweight and efficient messaging protocol commonly used in IoT (Internet of Things) applications for communication between devices.
tslib	tslib (Touchscreen Library) is an open-source library for handling touchscreen input. The C library for filtering touchscreen events. Visit the tslib website for an overview of the project: http://tslib.org.
yaffs2	YAFFS2 (Yet Another Flash File System 2) is a file system specifically designed for NAND flash memory devices. It is an open-source file system that provides reliable and efficient storage management for embedded systems and devices with limited resources.

Table 3-5 ThirdParty Directory

3.6 SampleCode/AmpCore1

3.6.1 AMP Core1

Sample Name	Description
AmpCore1	A project template for Core1 Only, with message output to UART16 and execution address set to 0x88000000.

Table 3-6 AmpCore1 Samples

3.7 SampleCode/Crypto

3.7.1 Crypto

Sample Name	Description
CRYPTO_AES	This sample program demonstrates the Crypto AES functions by encrypt/decrypt a set of KAT (Known Answer Test) pattern.

CRYPTO_ECC_KeyGen	This sample program demonstrates the Crypto ECC key generation function. Given a private key and generate the expected public key pairs.
CRYPTO_ECC_SigGenVerify	This sample program demonstrates the Crypto ECC ECDSA signature generation and verification.
CRYPTO_PRNG	This sample program demonstrates the Crypto PRNG functions.
CRYPTO_SHA	This sample program demonstrates the Crypto SHA functions by perform SHA operation on a set of known answer test patterns and compare to verify if the result is correct or not.
KS_SRAM	This sample program demonstrates Key Store SRAM operations.
TRNG	This sample program demonstrates how to make TRNG generate random numbers.

Table 3-7 Crypto Samples

3.8 SampleCode/FreeRTOS

3.8.1 FreeRTOS

Sample Name	Description
FreeRTOS	A FreeRTOS project with two demo applications.
FreeRTOS_Dual	An example demonstrates that on the MA35H0, both Cortex-A35 core 0 and core 1 can run their own instances of FreeRTOS independently and simultaneously.
FreeRTOS_Template	This project serves as a simple template for a blinky-style project.

Table 3-8 FreeRTOS Samples

3.9 SampleCode/Loader

3.9.1 Loader

Sample Name	Description
Loader	This loader is designed for custom DDR situation. MA35H0 IBR will load this loader to SRAM and run. This loader will then initialize the custom DDR and then load application images from SPI NAND, NAND, or SD/eMMC to DRAM for execution.
NuWriter	The example Jason files that can be used by Loader.

Table 3-9 Loader Samples

3.10 SampleCode/lwIP

3.10.1 lwIP

Sample Name	Description
lwIP_httpd_netconn	A simple HTTP server demonstrates LwIP under FreeRTOS. The server's IP address could configure statically to 192.168.1.2, or assign by DHCP server.
lwIP_httpd_socket	A simple HTTP server that demonstrates LwIP socket API under FreeRTOS. This HTTP server's IP address can be configured statically to 192.168.0.2, or assigned by DHCP server.
lwIP_iperf_Server	A lwIP iperf which is implemented with LwIP under FreeRTOS. Starting as a server, the sample code listens to iperf port 5001 with IP address 192.168.0.2.
lwIP_MQTT	A MQTT client sample. The lower level MQTT client functions are from eclipse paho.
lwIP_TCP_EchoClient	A TCP echo client which is implemented with LwIP under FreeRTOS. Client sends to 192.168.1.2:80, IP address could configure statically to 192.168.1.3 or assign by DHCP server.
lwIP_TCP_EchoServer	A TCP echo server which is implemented with LwIP under FreeRTOS. The server listen to port 80, IP address could configure statically to 192.168.1.2 or assign by DHCP server. This server replies "Hello World!!" if the received string is "nuvoton", otherwise reply "Wrong Password!!" to its client.
lwIP_TFTP_Client	A TFTP client sample that can receive a file from TFTP server or send a file to TFTP server.
lwIP_TFTP_Server	A TFTP server sample that communicates with TFTP client.
lwIP_UDP_EchoClient	A UDP echo server which is implemented with LwIP under FreeRTOS. This client sends “Hi there…” string to the server.
lwIP_UDP_EchoServer	A UDP echo server which is implemented with LwIP under FreeRTOS. This echo server listens to port 80, and its IP address can be configured statically to 192.168.1.2 or assigned by DHCP server. After receiving any string from its peer, this server echoes that string back.

Table 3-10 lwIP Samples

3.11 SampleCode/StdDriver

3.11.1 Analog-to-Digital Converter (ADC)

Sample Name	Description
ADC_Calibrate	Demonstrate ts_calibration and ts_test from tslib.

ADC_Convert	Demonstrate ADC function by repeatedly convert the input of ADC channel 4 (PB.12) and shows the result on UART console.
ADC_Touch	Demonstrate ADC 4-wire touch panel convert function.

Table 3-11 ADC Samples

3.11.2 M-Controller Area Network with Flexible Data-Rate (CAN FD)

Sample Name	Description
CANFD_CAN_Loopback	Use CAN mode function to do internal loopback test.
CANFD_CAN_TxRx	Transmit and receive CAN message through CAN interface.
CANFD_CAN_TxRxINT	An example of interrupt control using CAN bus communication.
CANFD_CANFD_Loopback	Use CAN FD mode function to do internal loopback test.
CANFD_CANFD_TxRx	Transmit and receive CAN FD message through CAN interface.
CANFD_CANFD_TxRxINT	An example of interrupt control using CAN FD bus communication.

Table 3-12 MCAN Samples

3.11.3 LCD Display Controller (DISP)

Sample Name	Description
DISP_Framebuffer	Demonstrate LCD Framebuffer display.
DISP_Overlay	Demonstrate LCD Overlay display. And demonstrate Alpha blending and ColorKey for Overlay.

Table 3-13 Display Samples

3.11.4 External Bus Interface (EBI)

Sample Name	Description
EBI_NOR	Configure EBI interface to access MX29LV320T (NOR Flash) on EBI interface.
EBI_SRAM	Configure EBI interface to access IS61WV204816BLL(SRAM) on EBI interface.

Table 3-14 EBI Samples

3.11.5 Enhanced PWM Generator and Capture Timer (EPWM)

Sample Name	Description
EPWM_AccumulatorINT_TriggerPDMA	Demonstrate EPWM accumulator interrupt trigger PDMA.
EPWM_AccumulatorStopMode	Demonstrate EPWM accumulator stop mode.
EPWM_Brake	Demonstrate how to use EPWM brake function.
EPWM_Capture	Capture the EPWM1 Channel 0 waveform by EPWM1 Channel 2.
EPWM_DeadTime	Demonstrate how to use EPWM Dead Zone function.
EPWM_DoubleBuffer	Change duty cycle and period of output waveform by EPWM Double Buffer function.
EPWM_OutputWaveform	Demonstrate how to use EPWM counter output waveform.
EPWM_PDMA_Capture	Capture the EPWM1 Channel 0 waveform by EPWM1 Channel 2, and use PDMA to transfer captured data.
EPWM_SwitchDuty	Change duty cycle of output waveform by configured period.
EPWM_SyncStart	Demonstrate how to use EPWM counter synchronous start function.

Table 3-15 EPWM Samples

3.11.6 Ethernet Media Access Controller (EMAC)

Sample Name	Description
EMAC_dhcp	This Ethernet sample tends to get a DHCP lease from DHCP server.

Table 3-16 EMAC Samples

3.11.7 General Purpose I/O (GPIO)

Sample Name	Description
GPIO_EINTAndDebounce	Show the usage of GPIO external interrupt function and de-bounce function.
GPIO_INT	Show the usage of GPIO interrupt function.

GPIO_OutputInput	Show how to set GPIO pin mode and use pin data input/output control.
GPIO_PowerDown	Show how to wake up system from Power-down mode by GPIO interrupt.

Table 3-17 GPIO Samples

3.11.8 High Speed USB 2.0 Device Controller (HSUSBD)

Sample Name	Description
HSUSBD_HID_Mouse	Simulate an USB mouse and draws circle on the screen.
HSUSBD_HID_MouseKeyboard	Simulate an USB mouse and keyboard device.
HSUSBD_HID_Transfer	Simulate an USB HID device and transfer data with PC application.
HSUSBD_Mass_Storage_RAM	Use DDR as back end storage media to simulate a 30 KB USB pen drive.

Table 3-18 HSUSBD Samples

3.11.9 USB 2.0 Host Controller (HSUBSH)

Sample Name	Description
HSUSBH_DEV_CONN	This sample uses connect/disconnect callback to get aware of device connect and disconnect events. It also shows device information represented in UDEV_T.
HSUSBH_HID	Use USB Host core driver and HID driver. This sample demonstrates how to submit HID class request and how to read data from interrupt pipe. This sample supports dynamic device plug/un-plug and multiple HID devices.
HSUSBH_HID_Keyboard	Use USB Host core driver and HID driver. This sample demonstrates how to submit HID class request and how to read data from interrupt pipe. This sample supports dynamic device plug/unplug and multiple HID devices.
HSUSBH_HID_Mouse_Keyboard	Use USB Host core driver and HID driver. This sample demonstrates how to support mouse and keyboard input.
HSUSBH_Mass_Storage	This sample uses a command-shell-like interface to demonstrate how to use USBH mass storage driver and make it working as a disk driver under FATFS file system.
HSUSBH_UAC	Demonstrates how to use USBH Audio Class driver. It shows the mute, volume, auto-gain, channel, and sampling rate control.
HSUSBH_UAC_LoopBack	The sample receives audio data from UAC device, and immediately send back to that UAC device.

HSUSBH_VCOM	Use USB Host core driver and CDC driver. This sample demonstrates how to connect a CDC class VCOM device.

Table 3-19 HSUSBH Samples

3.11.10 Hardware Semaphore (HWSEM)

Sample Name	Description
HWSEM_LockUnlock	Demonstrate hardware semaphore (HWSEM) lock/unlock function to protect a critical section in dual core system which each core may enter it at a same time.

Table 3-20 HWSEM Samples

3.11.11 I²C Serial Interface Controller (I²C)

Sample Name	Description
I2C_EEPROM	Read/write EEPROM via I²C interface.
I2C_Loopback	Demonstrate how to set I²C Master mode and Slave mode. And show how a master access a slave on a chip.
I2C_Master	An I²C master mode demo code. This is a I²C master mode demo and need to be tested with a slave device.
I2C_MultiBytes_Master	Show how to set I²C to use Multi bytes API Read and Write data to Slave. Needs to work with I2C_Slave sample code.
I2C_PDMA_TRX	Demonstrate I²C PDMA mode and need to connect I2C2 (Master) and I²C1 (Slave).
I2C_SingleByte_Master	Show how to use I²C single byte API Read and write data to slave needs to work with I2C_Slave sample code.
I2C_Slave	I²C Driver Sample Code. This is a I²C slave mode demo and need to be tested with a master device.
I2C_Wakeup_Slave	Show how to wake up from Power-down mode through I²C interface. This sample code needs to work with I2C_Master.

Table 3-21 I²C Samples

3.11.12 I²S Controller (I²S)

Sample Name	Description
I2S_Codec	This is an I²S demo using NAU88C22/88L25 audio codec, and used to playback the input from line-in or MIC interface.
I2S_Codec_PDMA	This is an I²S demo with PDMA function connected with NAU88C22 codec.

I2S_MP3PLAYER	This sample plays MP3 files stored on USB mass storage

Table 3-22 I²S Samples

3.11.13 Keypad Interface (KPI)

Sample Name	Description
KPI	Demonstrate KPI Controller.

Table 3-23 KPI Samples

3.11.14 NAND Flash Interface (NFI)

Sample Name	Description
NAND_Yaffs2	Access a NAND flash formatted in YAFFS2 file system.

Table 3-24 NAND YAFFS2 Samples

3.11.15 PDMA Controller (PDMA)

Sample Name	Description
PDMA_BasicMode	Use PDMA2 channel 2 to demonstrate memory to memory transfer.
PDMA_ScatterGather	Use PDMA2 channel 5 to demonstrate memory to memory transfer by scatter-gather mode.
PDMA_ScatterGather_ PingPongBuffer	Use PDMA to implement Ping-Pong buffer by scatter-gather mode (memory to memory).
PDMA_Stride	Use PDMA2 channel 2 to transfer data from memory to memory with stride.
PDMA_Stride_Repeat	Use PDMA2 channel 0 to transfer data from memory to memory with stride and repeat.
PDMA_TimeOut	Demonstrate PDMA2 channel 1 get/clear timeout flag with UART1.

Table 3-25 PDMA Samples

3.11.16 Quad Serial Peripheral Interface (QSPI)

Sample Name	Description
QSPI_DualMode_Flash	Access SPI Flash using QSPI dual mode.

QSPI_QuadMode_Flash	Access SPI Flash using QSPI quad mode.
QSPI_Slave3Wire	Configure QSPI1 as Slave 3 wire mode and demonstrate how to communicate with an off-chip SPI Master device with FIFO mode. This sample code needs to work with SPI_MasterFifoMode sample code.

Table 3-26 QSPI Samples

3.11.17 Real Time Clock (RTC)

Sample Name	Description
RTC_Alarm_Test	Demonstrate the RTC alarm function. It sets an alarm 10 seconds after execution.
RTC_Alarm_Wakeup	Use RTC alarm interrupt event to wake up system.
RTC_Time_Display	Demonstrate the RTC function and displays current time to the UART console.

Table 3-27 RTC Samples

3.11.18 Smartcard Host Interface (SC)

Sample Name	Description
SC_ReadATR	Read the smartcard ATR from smartcard 0 interface.
SC_ReadSIM_PhoneBook	Demonstrate how to read phone book information in the SIM card.
SC_Timer	Demonstrate how to use SC embedded timer
SCUART_TxRx	Demonstrate smartcard UART mode by connecting PK.12 and PK.13 pins.

Table 3-28 SC Samples

3.11.19 Secure Digital Host Controller (SDH)

Sample Name	Description
SDH_FATFS	Access a SD card formatted in FAT file system.

Table 3-29 SDH Samples

3.11.20 Serial Peripheral Interface (SPI)

Sample Name	Description

SPI_Flash	Access SPI Flash through a SPI interface.
SPI_HalfDuplex	Demonstrate SPI half-duplex mode. SPI0 will be configured as Master mode and SPI1 will be configured as Slave mode. Both SPI0 and SPI1 will be configured as half-duplex mode.
SPI_LoopBack	Implement SPI Master loop back transfer. This sample code needs to connect MISO_0 pin and MOSI_0 pin together. It will compare the received data with transmitted data.
SPI_MasterFIFOMode	Configure SPI0 as Master mode and demonstrate how to communicate with an off-chip SPI Slave device with FIFO mode. This sample code needs to work with SPI_SlaveFIFOMode sample code.
SPI_PDMA_LoopTest	Demonstrate SPI data transfer with PDMA. SPI3 will be configured as Master mode and SPI0 will be configured as Slave mode. Both TX PDMA function and RX PDMA function will be enabled.
SPI_SlaveFIFOMode	Configure SPI0 as Slave mode and demonstrate how to communicate with an off-chip SPI Master device with FIFO mode. This sample code needs to work with SPI_MasterFIFOMode sample code.
SPII2S_Master	Configure SPI1 as I^2S Master mode and demonstrate how I^2S works in Master mode. This sample code needs to work with SPII2S_Slave.
SPII2S_Slave	Configure SPI1 as I^2S Slave mode and demonstrate how I^2S works in Slave mode. This sample code needs to work with SPII2S_Master.

Table 3-30 SPI Samples

3.11.21 System Manager (SYS)

Sample Name	Description
SYS_DualCore	Dual Core demo.
SYS_DPDMode_Wakeup	Let CA35 enter DPD mode only, DDR keep working. Use Timer to warm boot system from DPD mode.

Table 3-31 SYS Samples

3.11.22 Timer Controller (TIMER)

Sample Name	Description
TIMER_CaptureCounter	Show how to use the timer0 capture function to capture timer0 counter value.

TIMER_Delay	Demonstrate the usage of TIMER_Delay() API to generate a 1 second delay.
TIMER_EventCounter	Use pin PI.0 to demonstrate timer event counter function.
TIMER_FreeCountingMode	Use the timer pin PI.1 to demonstrate timer free counting mode function. And displays the measured input frequency to UART console.
TIMER_InterTimerTrigger Mode	Use the timer pin PI.0 to demonstrate inter-timer trigger mode function. Also display the measured input frequency to UART console.
TIMER_Periodic	Use the timer periodic mode to generate timer interrupt every 1 second.
TIMER_PeriodicINT	Implement timer counting in periodic mode.
TIMER_PWM_Brake	Demonstrate how to use Timer PWM brake function.
TIMER_PWM_ChangeDuty	Change duty cycle and period of output waveform in PWM down count type.
TIMER_PWM_DeadTime	Demonstrate how to use Timer PWM Dead Time function.
TIMER_PWM_Output Waveform	Enable 4 Timer PWM output channels with different frequency and duty ratio.
TIMER_TimeoutWakeup	Use timer to wake up system from Power-down mode periodically.
TIMER_ToggleOut	Demonstrate the timer 0 toggle out function on pin PI.0.

Table 3-32 TIMER Samples

3.11.23 TS (Temperature Sensor)

Sample Name	Description
TS_TemperatureMeasure	Measure the current temperature by Temperature Sensor.

Table 3-33 TS Samples

3.11.24 UART Interface Controller (UART)

Sample Name	Description
UART_AutoBaudRate	Show how to use auto baud rate detection function.
UART_AutoFlow	Transmit and receive data using auto flow control.

UART_IrDA	Transmit and receive UART data in UART IrDA mode.
UART_PDMA	Demonstrate UART transmit and receive function with PDMA.
UART_RS485	Transmit and receive data in UART RS485 mode.
UART_TxRxFunction	Transmit and receive data from PC terminal through a RS232 interface.
UART_Wakeup	Show how to wake up system from Power-down mode by UART interrupt.

Table 3-34 UART Samples

3.11.25 VC8000 (H.264/JPEG Decoder)

Sample Name	Description
VC8000_H264DecodeClip	This sample code decodes a short H264 bit stream clip with MA35H0 VC8000 H264 decoder. It enables VC8000 PP which directly output images to display memory.
VC8000_H264DecodeFiles	This sample code searches the root directory of the USB disk to play back all h264 bit stream files that are found.
VC8000_JpegDecodeFiles	This sample code decodes a JPEG image with MA35H0 VC8000 JPEG decoder. It enables VC8000 PP which directly output images to display memory.
VC8000_JpegDecodeImage	Implement WDT time-out interrupt event to wake up system and generate time-out reset system event while WDT time-out reset delay period expired.

Table 3-35 VC8000 Samples

3.11.26 Watchdog Timer (WDT)

Sample Name	Description
WDT_TimeoutWakeupAndReset	Implement WDT time-out interrupt event to wake up system and generate time-out reset system event while WDT time-out reset delay period expired.
WDT_TimeoutReset	Implement WDT time-out interrupt and generate time-out reset system event while WDT time-out reset delay period expired.
WDT_TimeoutWakeup	Implement WDT time-out interrupt event to wake up system.

Table 3-36 WDT Samples

3.11.27 Window Watchdog Timer (WWDT)

Sample Name	Description
WWDT_CompareINT	Show how to reload the WWDT counter value.

Table 3-37 WWDT Samples

3.12 SampleCode/Template

3.12.1 Template

Sample Name	Description
Template	A project template for MA35H0 MPU.

Table 3-38 Template Samples

4 DEVELOPMENT ENVIRONMENT

This chapter describes the MA35H0 Non-OS BSP development environment, and how to debug Cortex-A core within the NuEclipse framework.

4.1 Toolchains Supporting

To make the NuEclipse ready for work based on your operation system. The NuEclipse could be downloaded from Nuvoton's official website https://www.nuvoton.com/tool-and-software/ide-and-compiler/. Only NuEclipse version 1.02.025 or higher supports compilation of the MA35H0 Non-OS BSP. For more information about how to use NuEclipse, please refer to *NuEclipse User Manual*.

4.2 Non-OS BSP Download

The MA35H0 Non-OS BSP utilizes NuEclipse as the firmware development environment. Each sample code includes a directory that contains the project file. You can download the BSP from the following link: OpenNuvoton/MA35H0_NonOS_BSP.

4.3 Preliminary Preparation

The Nu-Link2-Pro is a powerful Debugger and Programmer for Nuvoton NuMicro Family microcontrollers. With the Nu-Link2-Pro, users can debug directly on NuEclipse.

As shown in Figure 4-1, the Nu-Link2-Pro is a bridge between an USB and the SWD interface. Table 4-1 shows the pin corresponding to the target board, using NuMaker-HMI-MA35H04F70 as an Example.

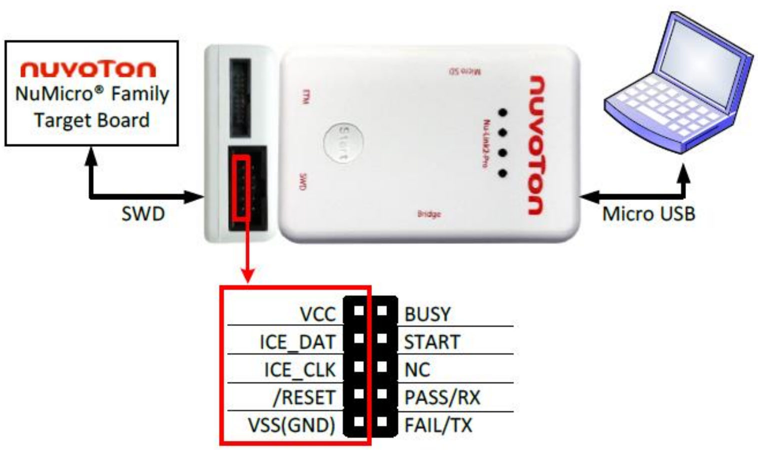

Figure 4-1 SWD Interface Connection Diagram for ICE

SWD Pin Number	Pin Corresponding to the Target Board	Pin Description
CON4.1	CON12.1	VCC
CON4.3	CON12.7	ICE_DAT : Serial Wired Debugger Data pin
CON4.5	CON12.9	ICE_CLK : Serial Wired Debugger Clock pin
CON4.7	CON12.3	/RESET : IC reset pin

CON4.9	CON12.4	VSS : Ground

Table 4-2 SWD Interface Corresponding Pin for NuMaker-HMI-MA35H04F70

Before debugging, we have to use a Nu-Link2-Pro and enable the CMSIS-DAP feature by the following steps:

1. Upgrade the Nu-Link2-Pro firmware whose version is higher than **v7513**.
2. Open NU_CFG.txt file located in the **NuMicro MCU disk folder**.
3. Set **BRIDGE-MODE=0** and re-plug the Nu-Link2-Pro.

```
NU_CFG.TXT - Notepad
File  Edit  Format  View  Help
[Build]
Version=7513r

[Power Control]
POWER-MODE=2
; Default I/O voltage
; 0 = 1.8V
; 1 = 2.5V
; 2 = 3.3V
; 3 = 5V

[Interface configuration]
BRIDGE-MODE=0
; 0 = CMSIS-DAP mode
```

Figure 4-2 Enable CMSIS-DAP Feature

In addition, when the chip is shipped for development, it is at OEM stage, where external debug ability is opened to user (developer) for convenience. In this condition, user has to use the NuWriter tool to put MA35H0 in a state of waiting for the Nu-Link2-Pro attach. Otherwise, the debug function cannot be achieved. The MA35 Series NuWriter provides two methods to achieve the previous purpose, as follows.

The first method is an easy way, by using the NuWriter_MA35_UI.

1. Click the **DDR File Browse** button to select the DDR initialize code.
2. Then click **Attach** button to trigger attach command.

The second method is to boot from a booting storage and executes any image in Non-OS BSP successfully. For more detailed information, please refer to *MA35 Series NuWriter User Manual*.

For more details about how to launch the application into the debug mode, please refer to the following chapter.

4.4 Execute the Project under NuEclipse

This section provides steps on how to run a project by using NuEclipse. When Non-OS BSP projects are available, we can import them into the workspace using the following steps:

1. Double-click **eclipse.exe** to open the toolchain.
2. From the main menu bar, select **File > Import**. The Import wizard shows up.
3. Select General > Existing Project into Workspace and click Next.
4. Choose **Select root directory** and click the associated **Browse** to locate the GCC folder in the Non-OS BSP.

5. Select the project which you would like to import and click **Finish**.

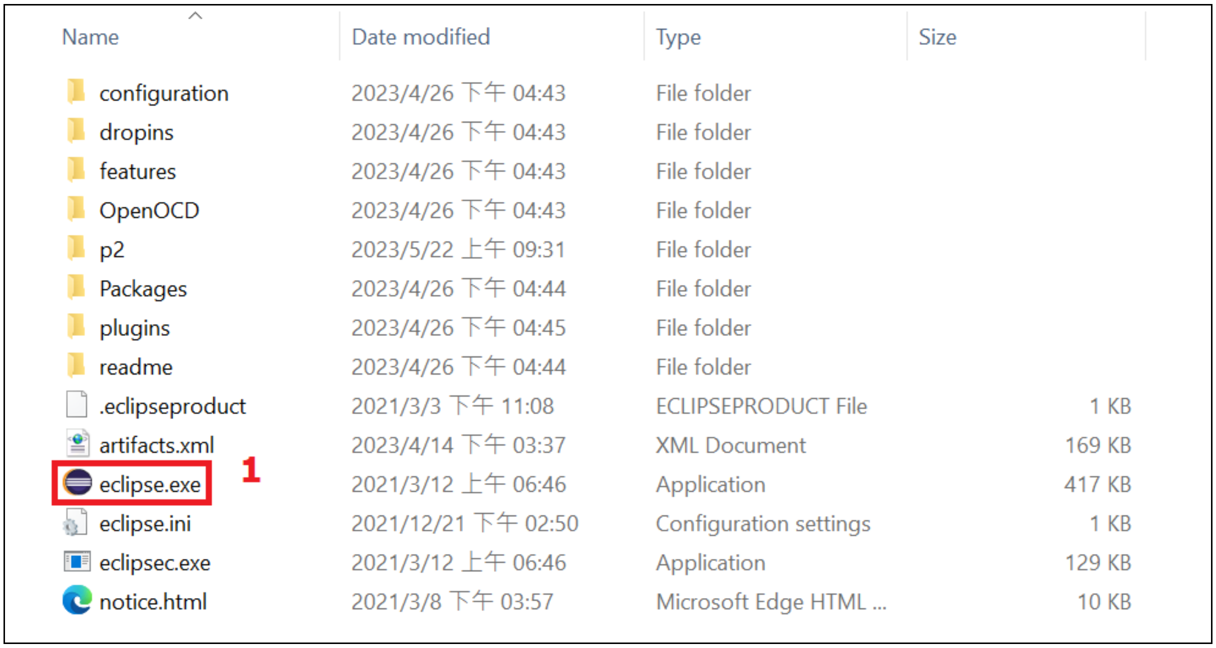

Figure 4-3 Eclipse.exe and Related Folders

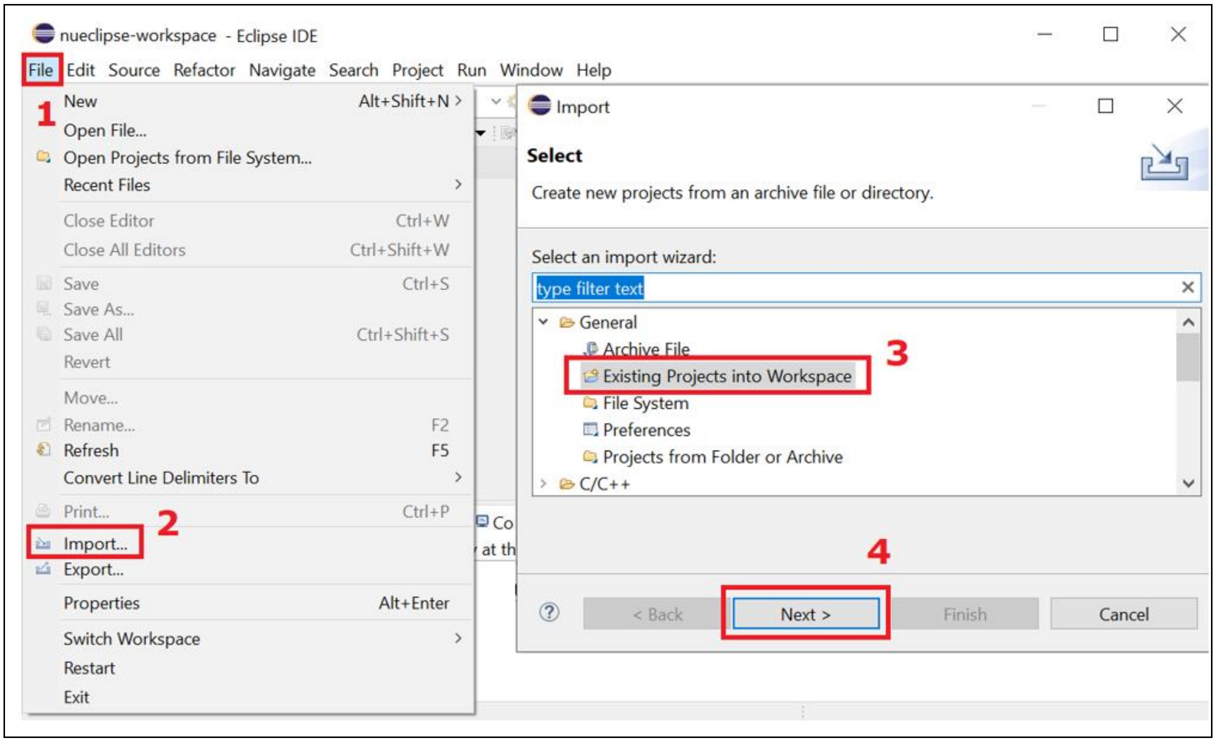

Figure 4-4 Import the Project in NuEclipse

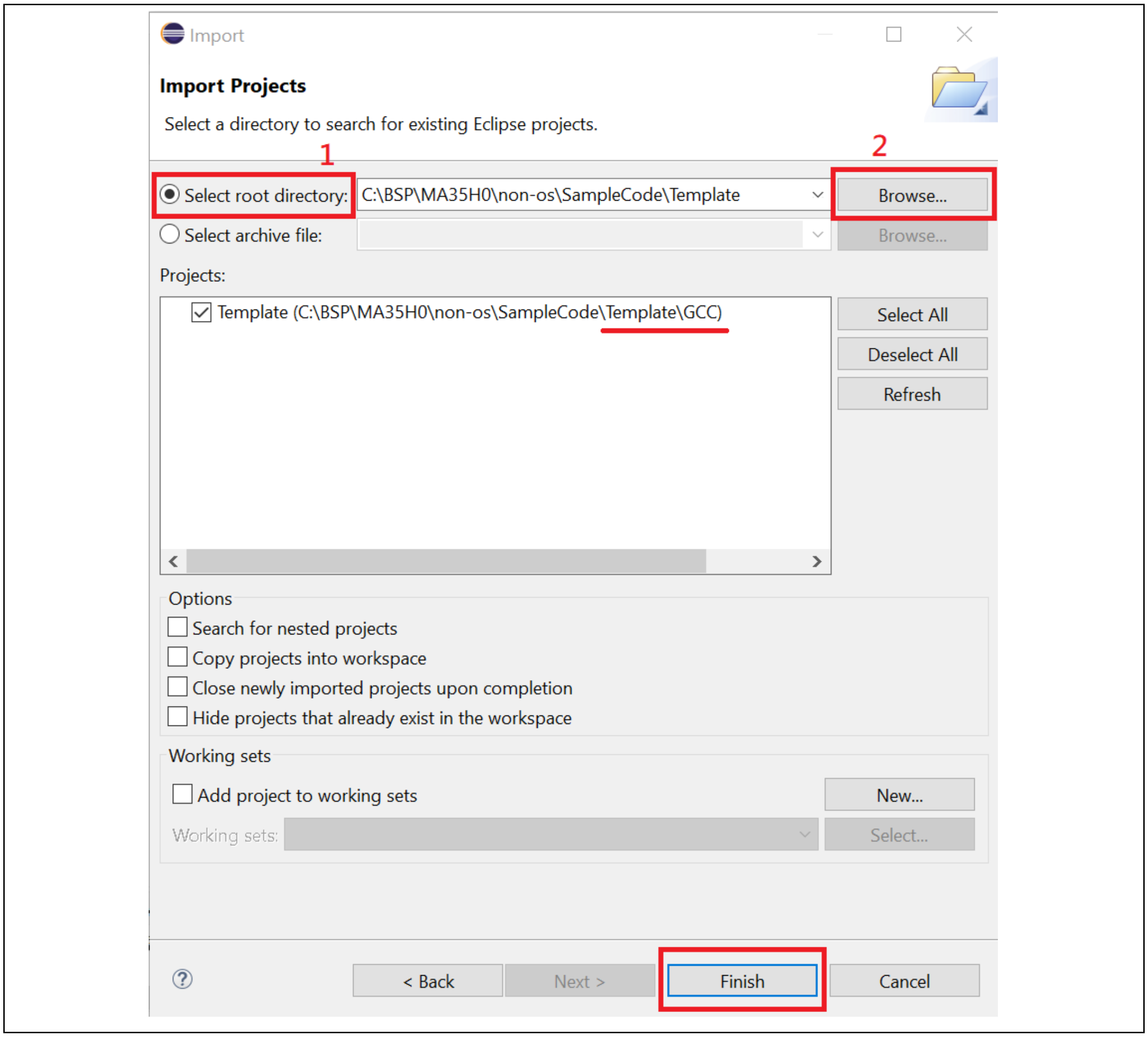

Figure 4-5 Import Projects Windows

6. The next step is to build the project. Select the project and click **Project > Build Project**.

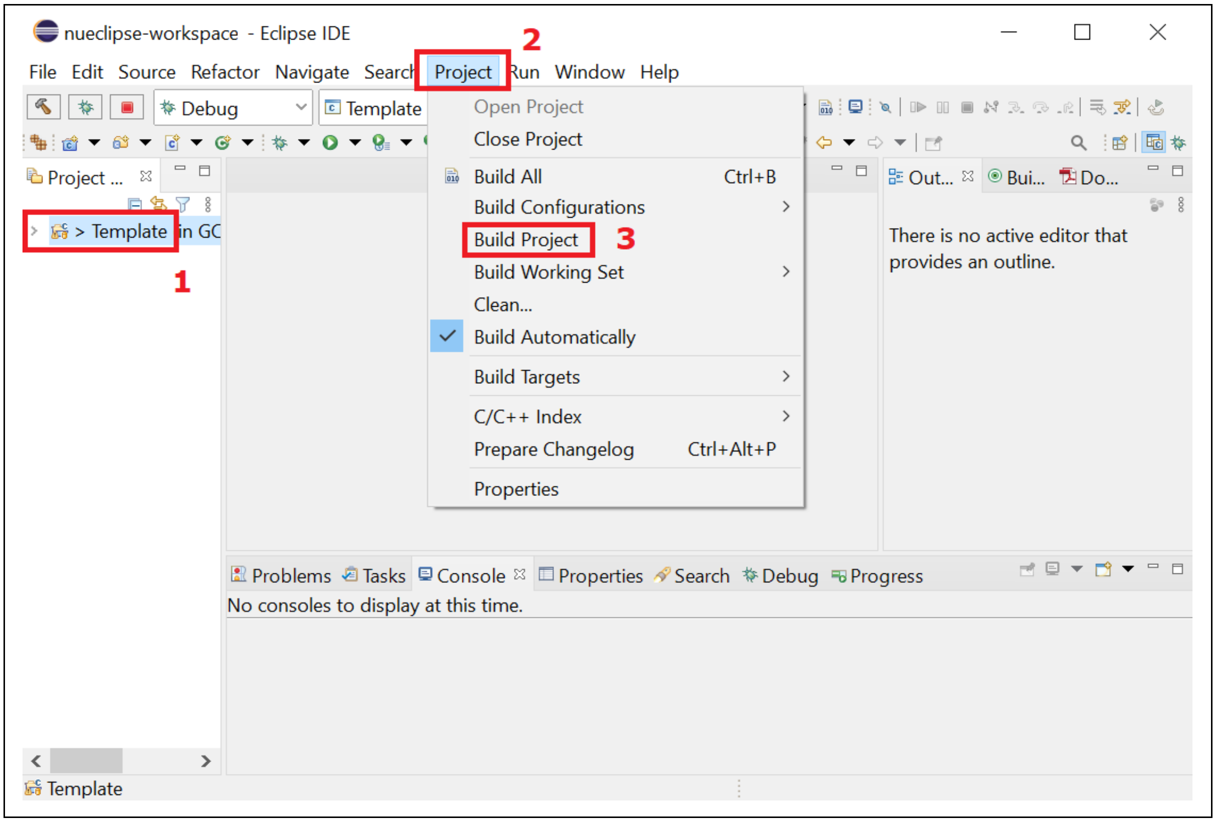

Figure 4-6 Build Project

Before launching the project into the debug mode, we have to prepare a debug configuration. For more information on how to debug MA35H0, please refer to *NuEclipse User Manual*.

1. Click Run > Debug Configurations… to open the debug configuration dialog.
2. Double click on the GDB Nuvoton Nu-Link Debugging group to create the sub item. The Nuvoton Nu-Link debug configuration appears on the right-hand side. After the project is built, the *.elf file will be shown in C/C++ Appliction field.
3. The Debugger tab is used to provide the OpenOCD and GDB Client setup. The configuration files are specified in the Config options field. The interface configuration file named cmsis-dap.cfg and the target configuration file is numicroMA35H0.cfg.
4. In the Startup tab, check the settings highlighted in green and click Apply button to take effect. Click Debug button to launch the project into debug mode. (To load script, the command would be: monitor script filename. E.g. monitor script ../scripts/board/numicroMA35H0_DDR2_128MB_1066MBPS_WINBOND.cfg)

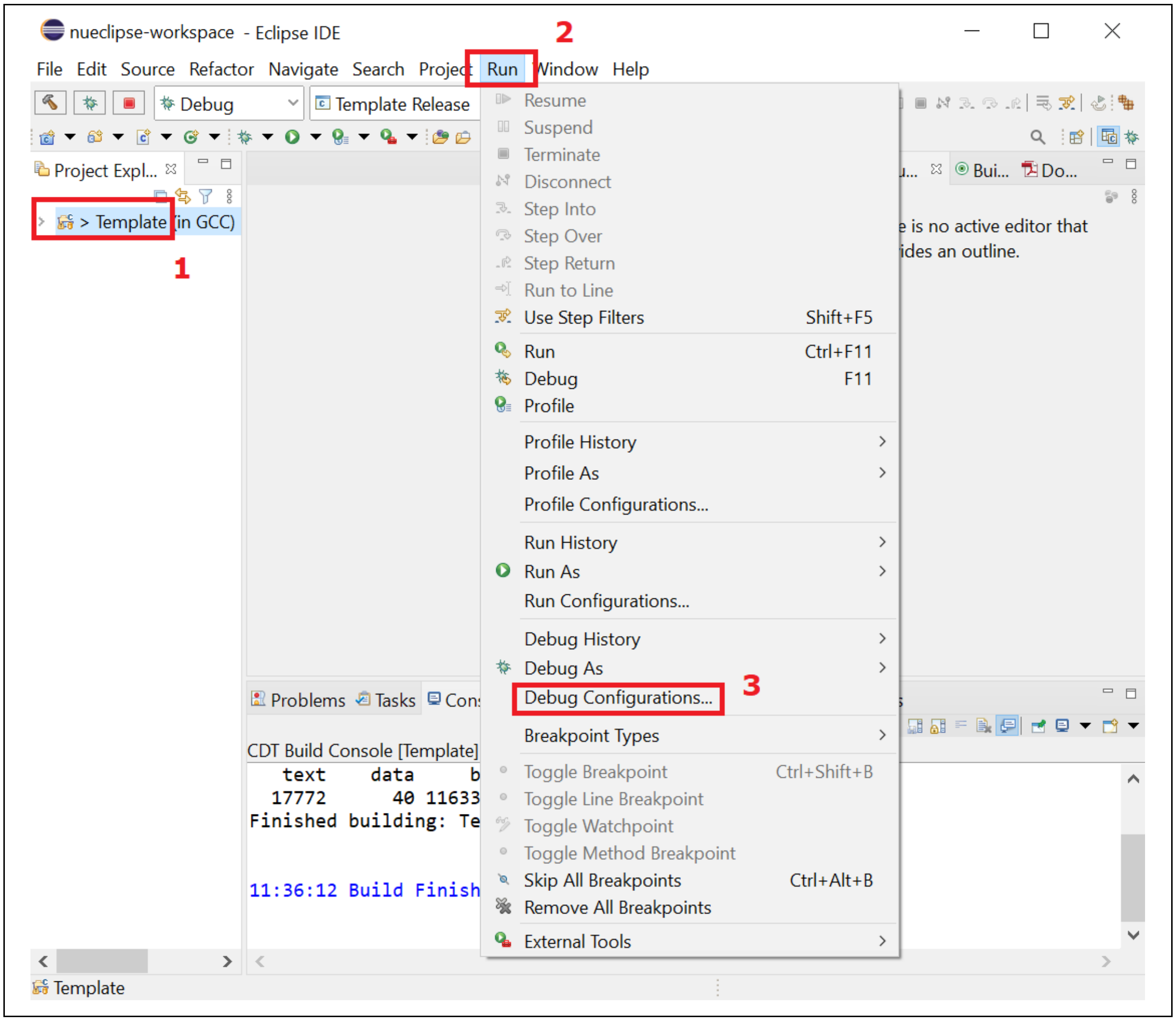

Figure 4-7 Open Debug Configuration

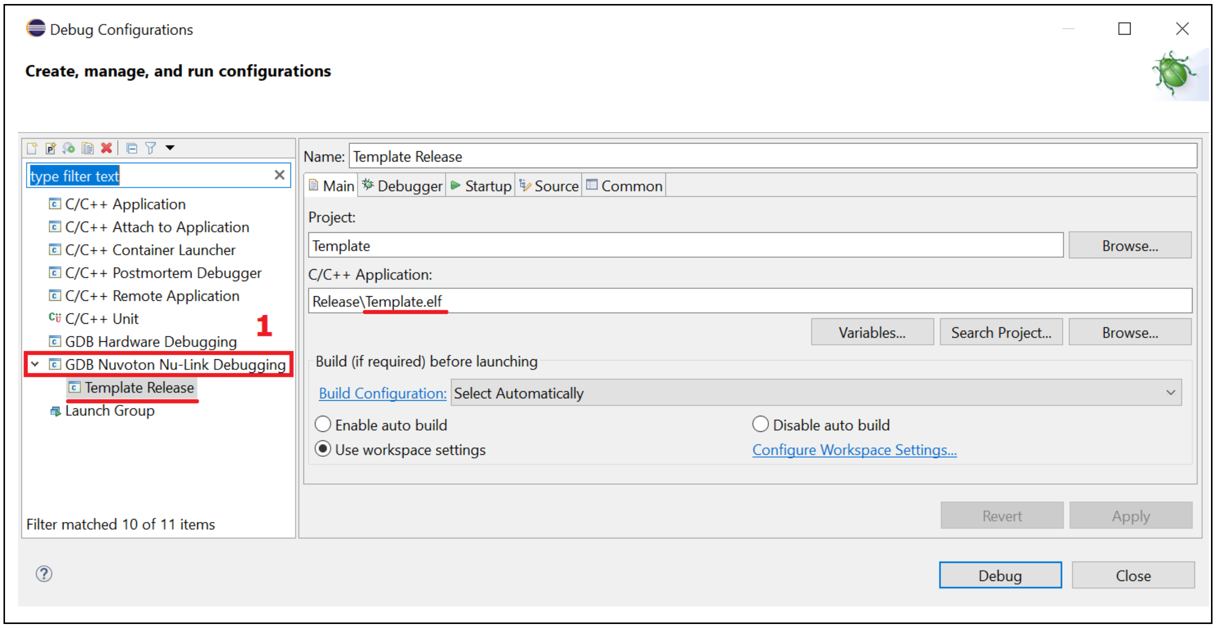

Figure 4-8 Main Tab Configuration

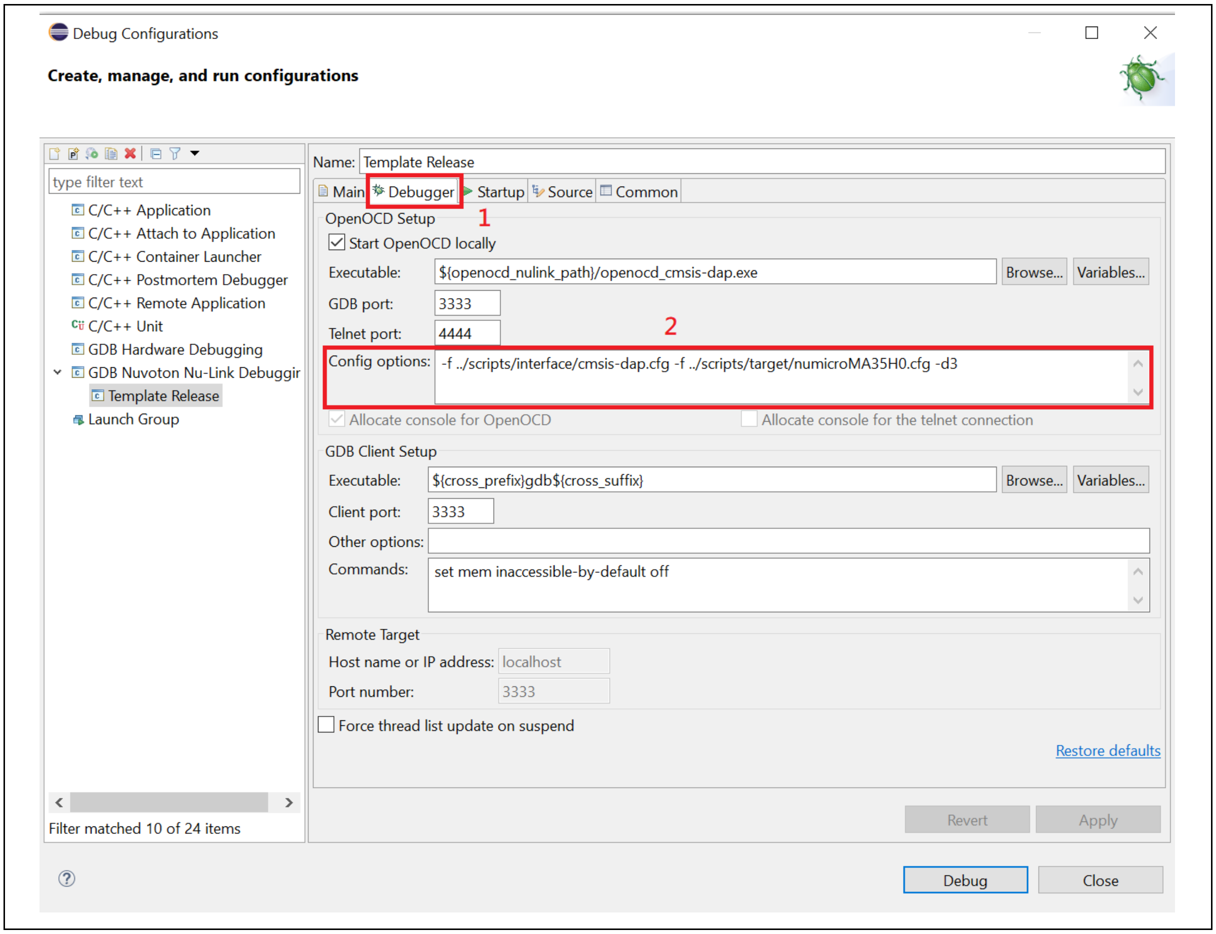

Figure 4-9 Debugger Tab Configuration

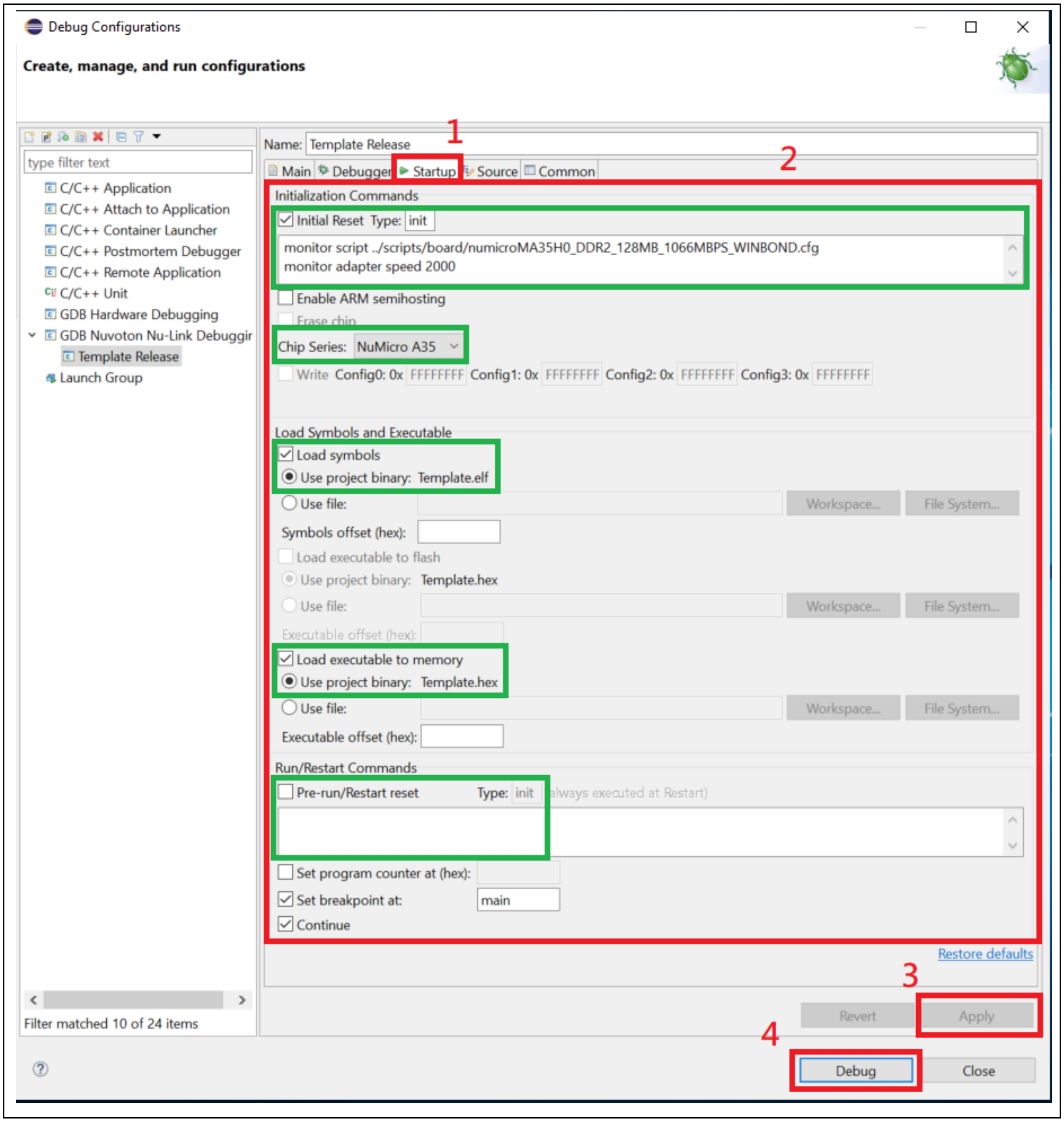

Figure 4-10 Startup Tab Configuration

5 LOAD AND RUN

After the project compilation is complete, in addition to using ICE to load it onto the MA35H0 evaluation board, there is another method to load-and-run images, using the NuWriter_MA35 tool.

NuWriter_MA35 is a programming tool developed by Nuvoton. It runs on a PC and connects to the MA35H0 evaluation board via USB to perform programming operations on the SD/eMMC, NAND, or SPI NAND of the MA35H0. It also supports loading executable binaries into the MA35H0 SRAM/DRAM and executing them on the spot.

This chapter will explain how to use NuWriter_MA35 in both command line and UI modes. For more inf ormation about the NuWriter_MA35 tool, please refer to *MA35 Series NuWriter User Manual*.

5.1 Load and Run with NuWriter_MA35

After a project is successfully compiled, you can find an executable binary file with the .bin extension in its GCC/Release directory. This binary file is the one to be loaded and executed on the MA35H0. It is quite simple to load and execute the executable binary on the MA35H0 using the NuWriter_MA35 tool. It only requires two commands, for example:

```
> NuWriter_MA35.exe -a ddrimg\MA35H04F767C.bin
> NuWriter_MA35.exe -o execute -w ddr 0x80000000 C:\BSP\MA35H0\non-os\SampleCode\StdDriver\SYS_DualCore\GCC\Release\SYS_DualCore.bin
```

The execution of the first command includes connecting via USB to the MA35H0 and loading the specified DDR configuration for DRAM initialization. The second command loads the specified executable binary into the DRAM at address 0x80000000 and starts its execution. If you want to load and execute again, you need to reset the MA35H0 and repeat these two commands.

Before proceeding with the loading process, please make sure that the power-on setting of the MA35H0 is configured to boot from USB. If it is set correctly, after resetting the MA35H0, you should see the following message on the UART debug console:

```
MA35D1 IBR 20211029
RTC power on
USBD Boot
```

After connecting the USB cable, execute the first command. In the Windows command shell, you should see messages similar to the following:

```
Successfully attached 1 device(s)

==== NAND ====
Page per block: 64
Page size: 2048
Block per flash: 8192
Bad block count: 0
Spare size: 64
Is uer config: 0
==== SPI NOR ====
ID: 0
Is uer config: 0
Quad read cmd: 0
Read sts cmd: 0
```

```
Write sts cmd: 0
Sts value: 0
Dummy byte: 0
==== eMMC ====
Block: 7862272
Reserved: 0
==== SPI NAND ====
Is uer config: 0
ID: 15710755
Page size: 2048
Spare size: 64
Quad read cmd: 107
Read sts cmd: 5
Write sts cmd: 1
Sts value: 2
Dummy byte: 1
Block per flash: 4096
Page per block: 64

Successfully get info from 1 device(s)
```

And, the MA35H0 UART debug console will display more messages as follows:

```
ddr ok
xusb ok
finish

NuWriter... version Dec5 2023 09:49:23
connect
Early suspend interrupt
Suspend interrupt :(DSTS):0x400003

Reset interrupt - (SYS_MISCISR):0x40004
Speed Detection interrupt
High Speed Detection : 0x0
*** USB_REQ_SET_ADDRESS (59)
<0x5 / 0x5>

Receive INFO(80) flash Image ...
Get INFO flash Image ...

<0xd00>

```

```
SPI NAND: QSPI->CTL = 0x805    QSPI->SSCTL = 0x5   QSPI->CLKDIV = 0x5
Auto Detect
ID=[0xefba23]
BlockPerFlash = 4096, PagePerBlock = 64
PageSize = 2048, SpareArea = 64
QuadReadCmd    = 0x6b
ReadStatusCmd  = 0x5
WriteStatusCmd = 0x1
StatusValue    = 0x2
Dummybyte      = 0x1
page 2048, spare area 64, block 64 (8192), 1
page size 2048, page/block 64
finish nand: page 2048, oob 64, ecc 0x100000, 1
finish ibr-nand: page 2048, oob 64, ecc 0x100000, 1

BlockPerFlash=8192, PagePerBlock=64, PageSize=2048, oob=64

SD0

sdhci_setup_cfg, caps: 0x276ec898
sdhci_setup_cfg, caps_1: 0x8008077
clock is disabled (0Hz)
selecting mode MMC legacy (freq : 0 MHz)
clock is enabled (400000Hz)
selecting mode SD High Speed (50MHz) (freq : 50 MHz)
sd card: widths [4, 1] modes [SD Legacy, SD High Speed (50MHz)]
host: widths [8, 4, 1] modes [MMC legacy, SD Legacy, MMC High Speed (26MHz), SD
High Speed (50MHz), MMC High Speed (52MHz), UHS SDR12 (25MHz), UHS SDR25 (50MHz),
UHS SDR50 (100MHz), UHS DDR50 (50MHz), UHS SDR104 (208MHz), HS200 (200MHz)]
==========>MMC bus width=4
trying mode SD High Speed (50MHz) width 4 (at 50 MHz)
==========>MMC bus width=4
===========================>SD_BUS_WIDTH_4
==========>MMC bus width=4
selecting mode SD High Speed (50MHz) (freq : 50 MHz)
clock is enabled (50000000Hz)
sd size 14832 MBytes



Finish get INFO!!


debug LED is Port 9 bit 15
```

Once the USB connection has been established successfully, you can proceed to issue the second command to download the executable image and execute it.

NuWriter_MA35 will continuously display the download progress in the Windows command shell until it reaches 100% completion, as shown below:

```
Programming 0: 100%|##########
Successfully programmed 1 device(s)
```

On the other side, the MA35H0 console should display the following messages:

```
download image to DDR...
write address 0x80000000, len 18104
execute image ...
run ... 80000000



This is core 1 0ms
This is core 0 0ms
This is core 1 11ms
This is core 0 12ms
```

5.2 Load and Run with NuWriter_MA35_UI

The NuWriter_MA35_UI is, in fact, a wrapper for NuWriter_MA35. It offers a user-friendly interface to perform the same functions as NuWriter_MA35. Simply run NuWriter_MA35_UI, select the DDR configuration file for the MA35H0 board, and click the '**Attach**' button to establish a USB connection with the MA35H0 target board, as illustrated below:

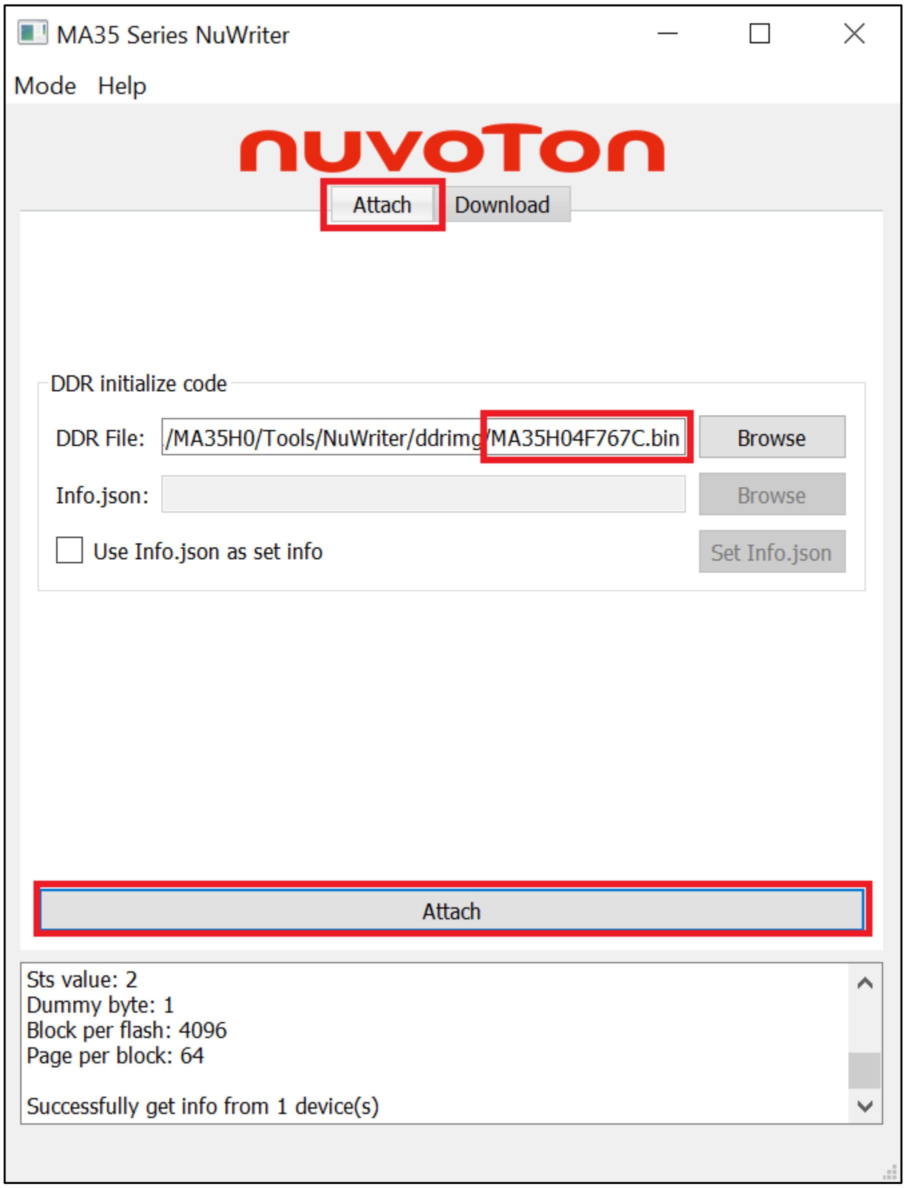

Figure 5-1 NuWriter_MA35_UI Attach USB

Once the USB connection has been successfully established, you can follow the steps below to download and execute the executable binary:

1. From the main menu, select 'Mode' -> 'Develop mode'.
2. Choose the 'Download' tab and then select the 'DDR/SRAM' tab.
3. In the 'Image file' field, click the 'browse' button to select the binary file you want to load.
4. Enter 0x8000000, which is the execution address, in the 'Image addr' field.
5. Check the ‘Execute after download’ checkbox.
6. Click on 'Write' to download the executable binary and initiate the execution.

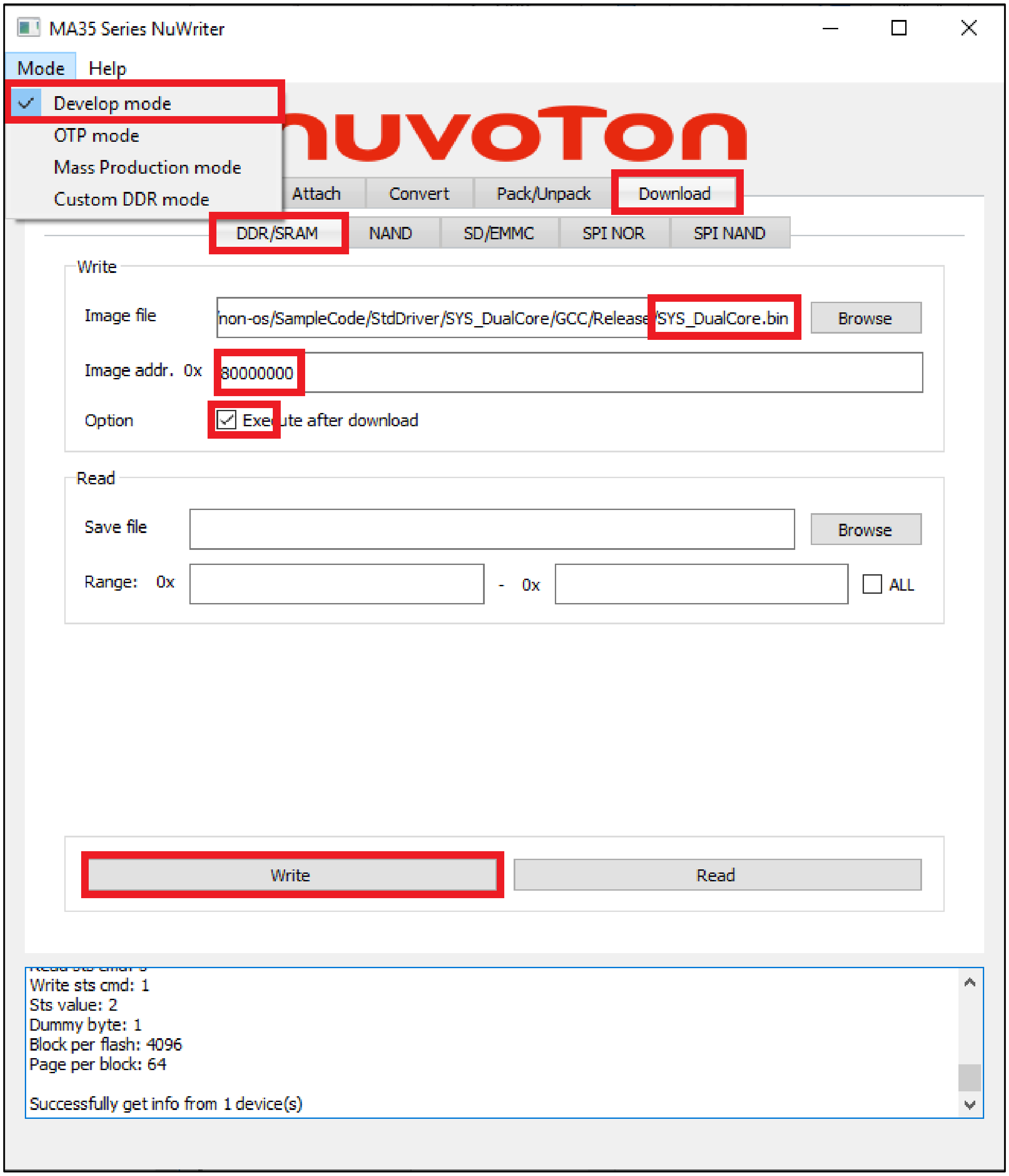

Figure 5-2 NuWriter_MA35_UI Download Binary File

6 MADE BOOTABLE IMAGES

When the development of the test program is completed, in addition to using tools to directly download and execute, it can also be burned to a storage device, such as SPI-NAND, NAND, or eMMC, and then directly downloaded and executed by IBR when booting. This chapter will introduce how to make a bootable program.

6.1 Known DDR Models

The DDR model of MA35H0 is MA35H04F767C. Nuvoton provides the initialization files of the DDR model, which are placed in the ddrimg directory under the NuWriter tool.

The steps to create a bootable program are as follows:

- Prepare header.json and pack.json. Use SPI-NAND as example.

header-spinand.json

```
{
    "header":
    {
        "version": "0x20230505",
        "spiinfo":
        {
            "pagesize": "2048",
            "sparearea": "64",
            "pageperblk": "64",
            "quadread": "0x6B",
            "readsts": "0x05",
            "writests": "0x01",
            "stsvalue": "0x02",
            "dummy1": "0",
            "dummy2": "1",
            "suspintvl": "1"
        },
        "secureboot": "no",
        "entrypoint": "0x80000000",
        "image":
        [
            {
              "offset": "0xC0000",
              "loadaddr": "0x28030000",
              "type": "2",
              "file": "ddrimg/ MA35H04F767C.bin"
            },
            {
              "offset": "0xE0000",
              "loadaddr": "0x80000000",
```

```
                "type": "4",
                "file": "Template.bin"
            }
        ]
    }
}
```

pack-spinand.json

```
{
    "image":
    [
        {
            "offset": "0x0",
            "file": "conv/header.bin",
            "type": 0
        },
        {
            "offset": "0xC0000",
            "file": "ddrimg/ MA35H04F767C.bin",
            "type": 0
        },
        {
            "offset": "0xE0000",
            "file": "Template.bin",
            "type": 0
        }
    ]
}
```

- Use NuWriter to convert and pack.

Convert header.json

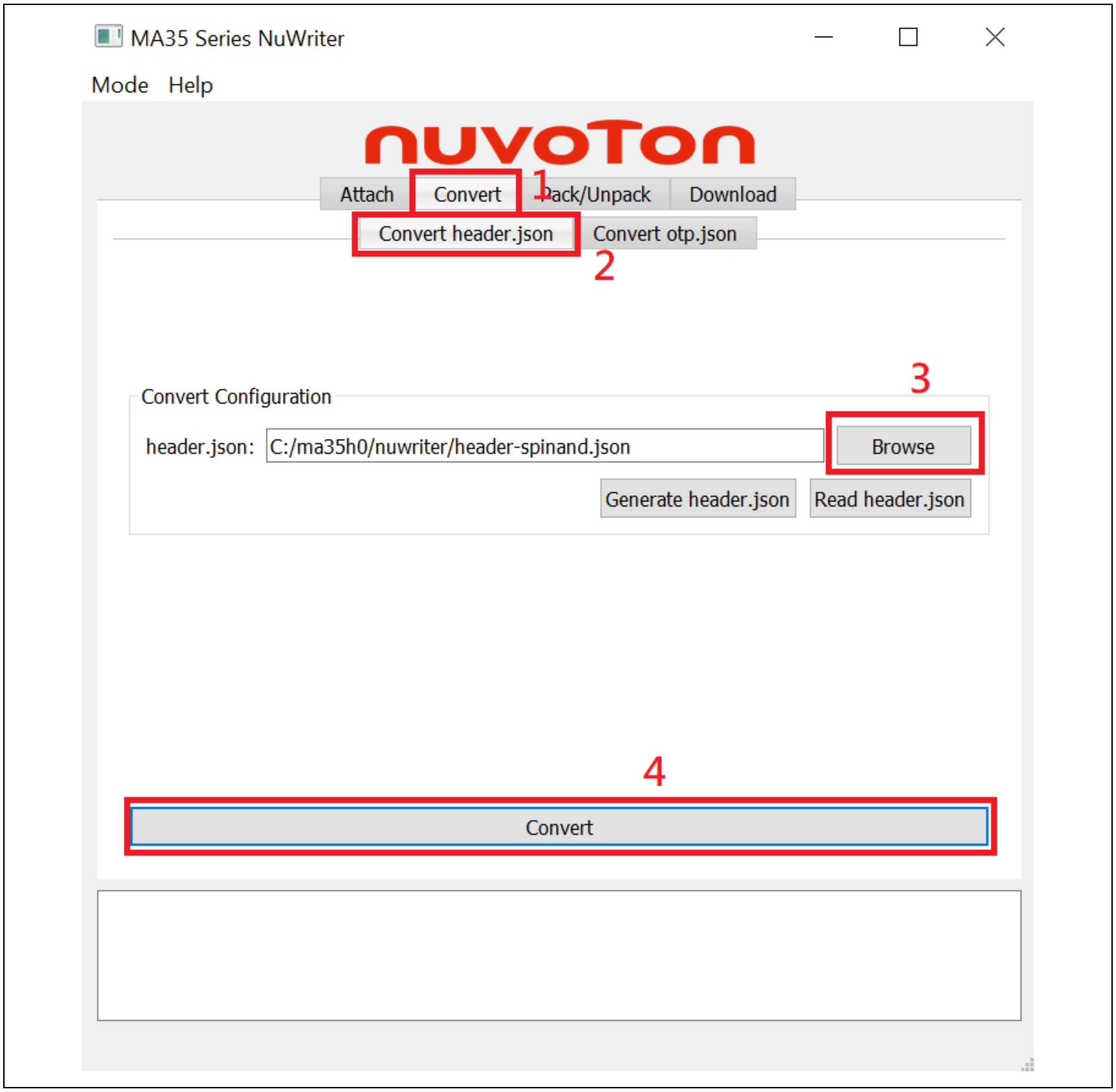

Figure 6-1 NuWriter convert header

Pack pack.json

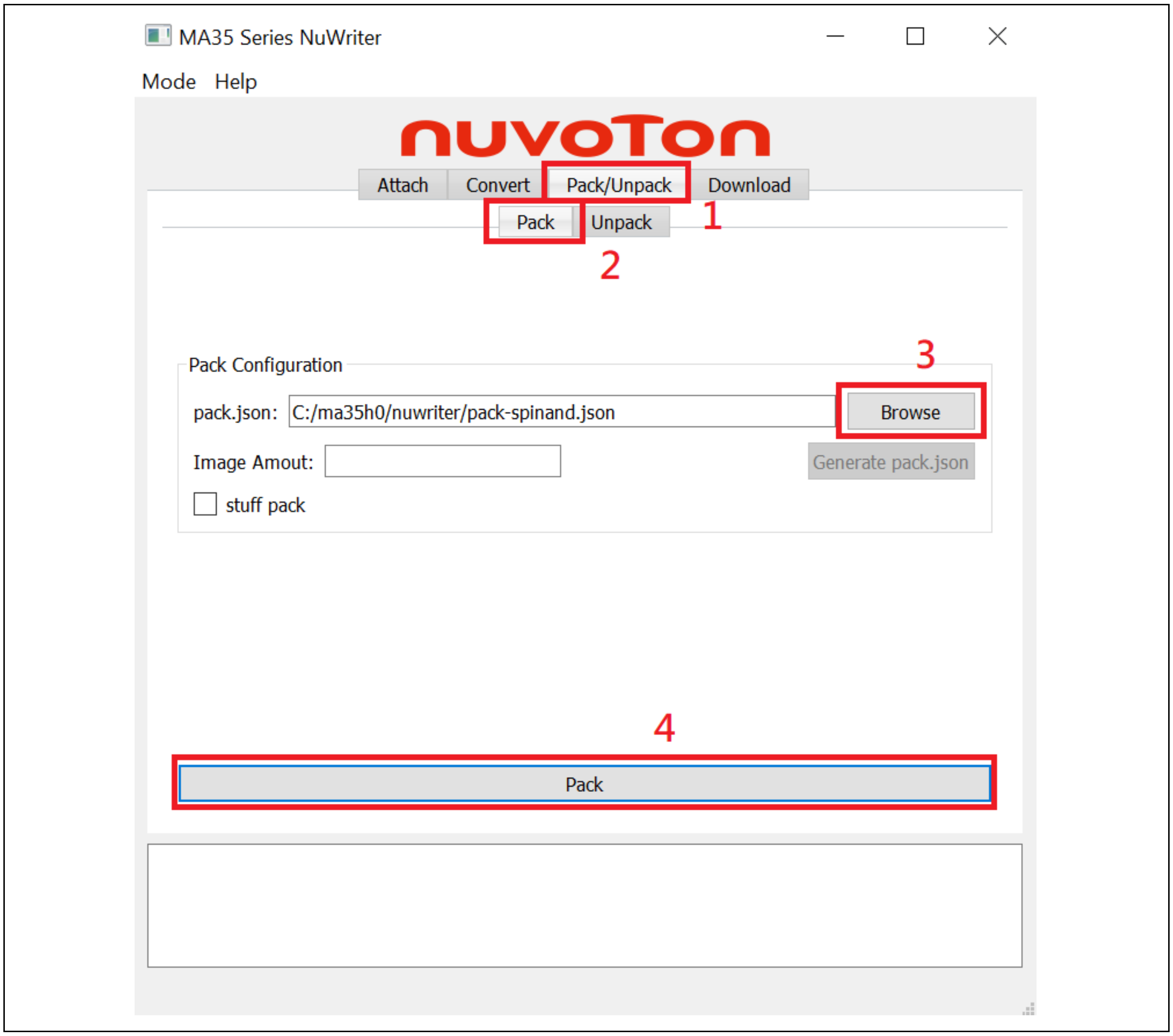

Figure 6-2 NuWriter pack file

- Use NuWriter to program storage. Please refer to *MA35 Series NuWriter User Manual*.

7 REVISION HISTORY

Date	Revision	Description
2024.01.15	1.00	• Initial version.

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.