NuMicro® Family

Arm® Cortex®-A35- based Microprocessor

NuMicro® Family

MA35D1 RTP

User Manual

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

www.nuvoton.com

Table of Contents

1 OVERVIEW

MA35D1 is a heterogeneous microprocessor contains a dual-core Corex-A35 and a Cortex-M4. Although MA35D1 is a powerful microprocessor, executing Linux is not that suitable for handling real time event such as motor control. The Cortex-M4 becomes quite handy as a real time processor to process real time events in such scenario.

Another scenario is to use Cortax-A35 for data processing and can stay in power down mode most of the time and wake up to process the data periodically. While Cortex-A35 is in power down mode, Cortex-M4 is still capable for control some peripherals for data collecting, so the system can keep functioning with minimum power consumption.

Chapter 2 describes the development of RTP, chapter 3 describes the software directory structure. Chapter 4 introduces different method to load and execute RTP firmware. Chapter 5 and 6 describes the method Cortex-A35 and Cortex-M4 communicate with each other.

2 DEVELOPMENT ENVIRONMENT

The RTP software package supports using Keil uVision, IAR Embedded Workbench, and NuEclipse as firmware development environment. Every sample code contains three directories holding the project file for these tools.

2.1 Preliminary Preparation

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 v7174.
2. Open NU_CFG.txt file located in the NuMicro MCU disk folder.
3. Set CMSIS-DAP=1 and re-plug the Nu-Link2-Pro.

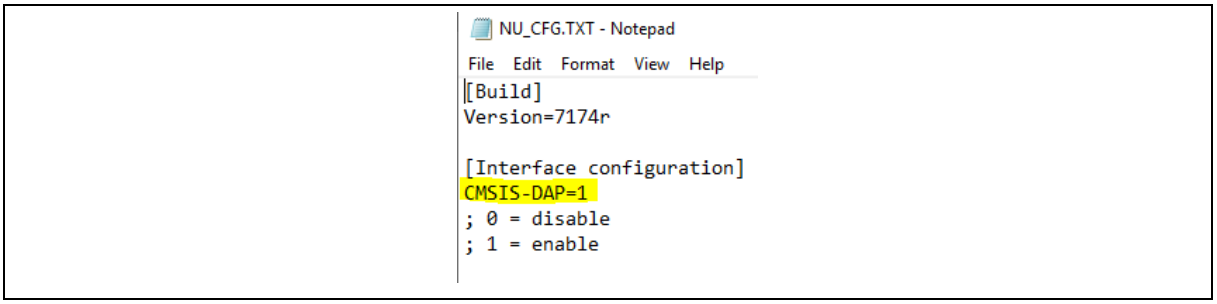


Figure 2-1 Enable CMSIS-DAP feature

2.2 Keil uVision

To open the Keil project, double click the uvproj file under the Keil/ directory, or open the uvproj file from the “Project” pull down menu after launch the Keil uVision as shown in Figure 2-2.

Figure 2-2 Open Keil Project

In “**Options for Target – Linker**” tab, modify “R/O Base” and “R/W Base” value shown in Figure 2-3.

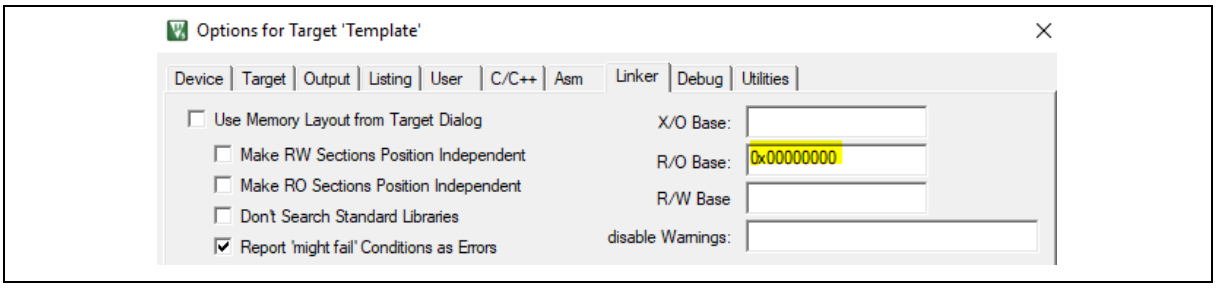


Figure 2-3 Linker Configuration

To build the project click the “Build” icon shown in Figure 2-4.

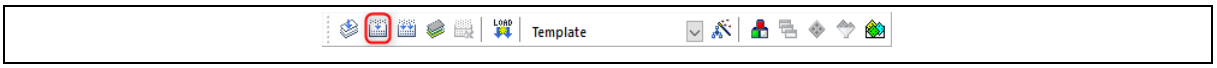


Figure 2-4 Build Keil Project

In “**Options for Target – Debug**” tab, select CMSIS-DAP Debugger Driver shown in Figure 2-5 and click settings.

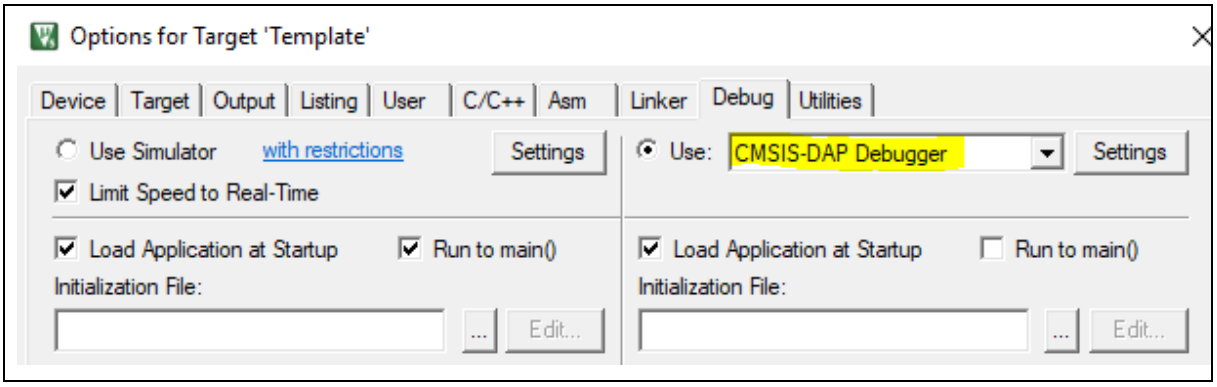


Figure 2-5 Select Debugger Driver

Select SW adapter “Nu-Link 2 CMSIS-DAP” and check IDCODE for device connection and set AP to 0x02 shown in Figure 2-6.

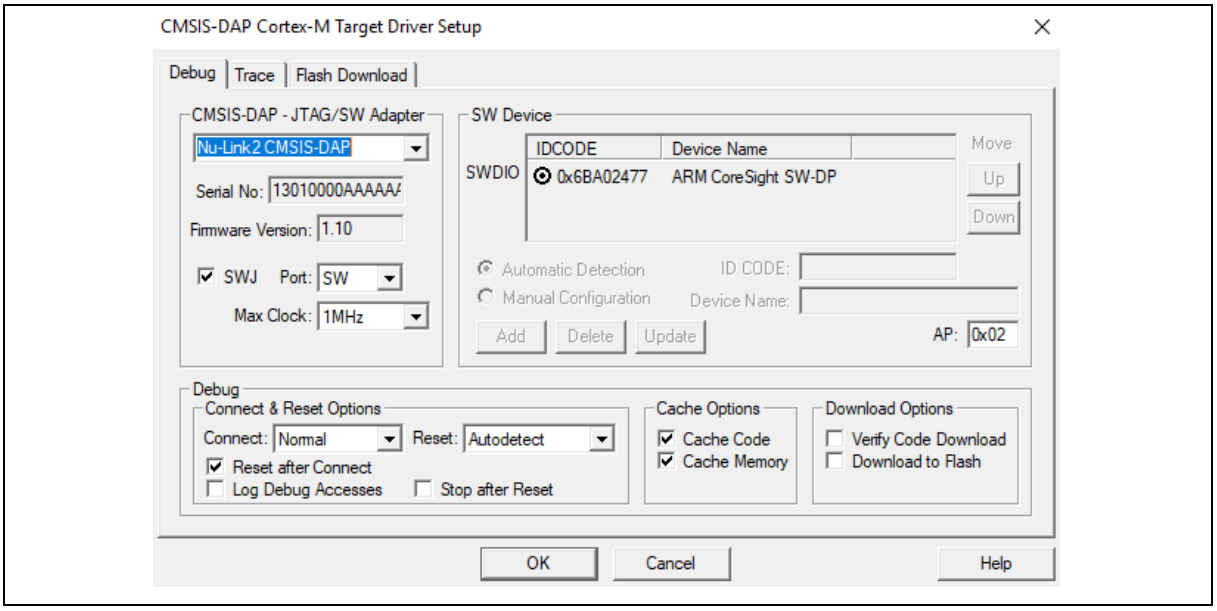


Figure 2-6 CMSIS-DAP Debug Settings

In “**Options for Target – Utilities**” tab, uncheck “Update Target before Debugging” option shown in Figure 2-7.

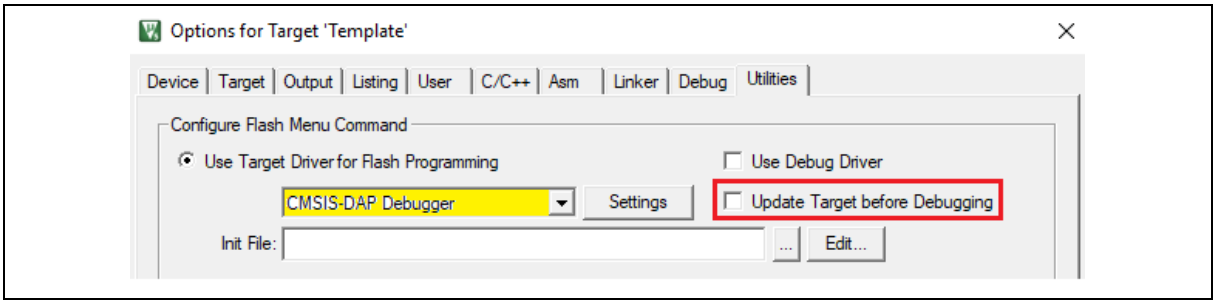


Figure 2-7 CMSIS-DAP Utilities Settings

uVision will load the built image and enter debug mode after click the “Start/Stop Debug Session” icon as shown in Figure 2-8.

Figure 2-8 Debug Keil Project

2.3 IAR Embedded Workbench

To open the IAR workspace, double click the eww file under IAR/ directory or open the eww file from the “File” pull down menu after launch the IAR Embedded Workbench as shown in Figure 2-9.

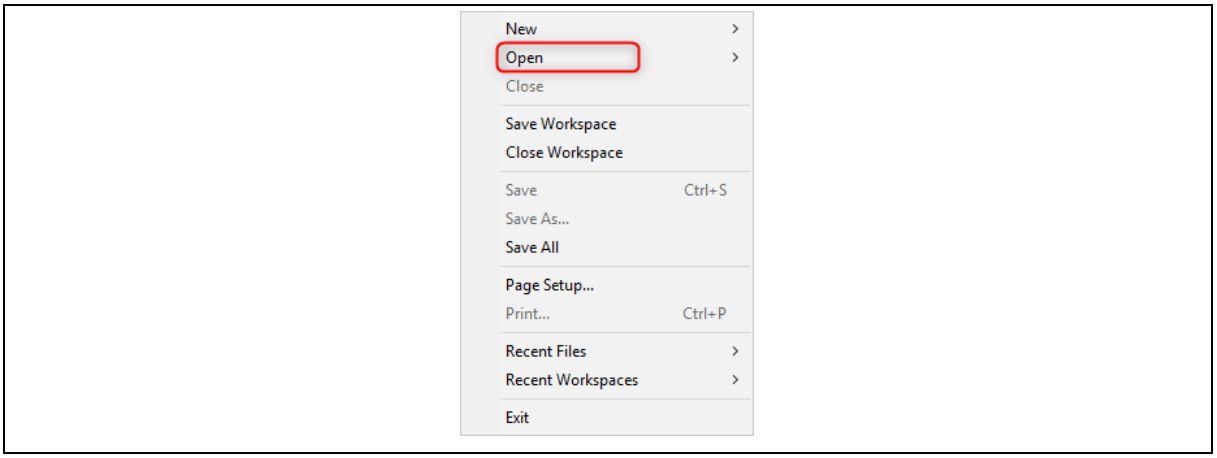


Figure 2-9 Open IAR Workspace

Open Options to modify setting shown in Figure 2-10.

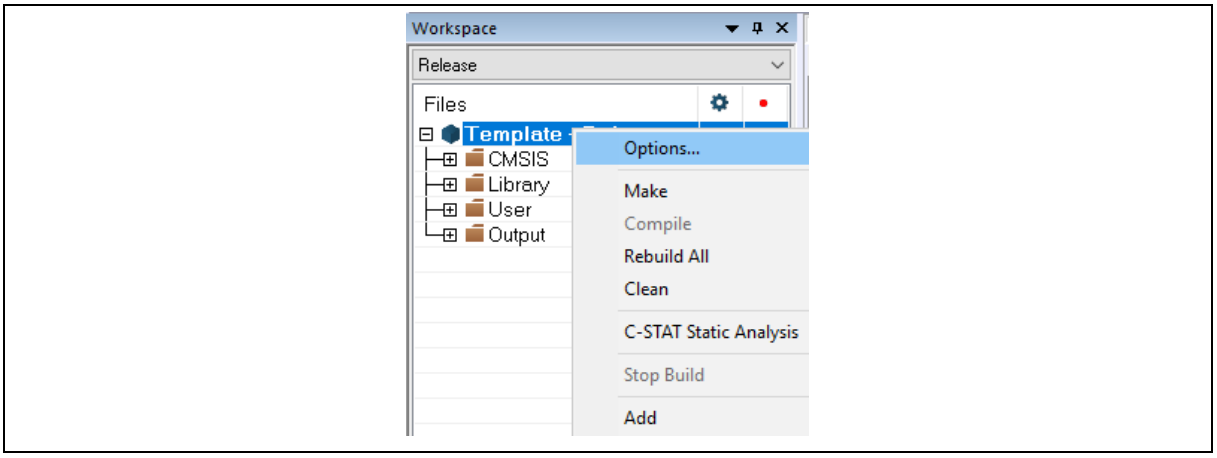


Figure 2-10 Workspace Options

Select Cortex-M4F or Cortex-M4 with floating point option shown in Figure 2-11.

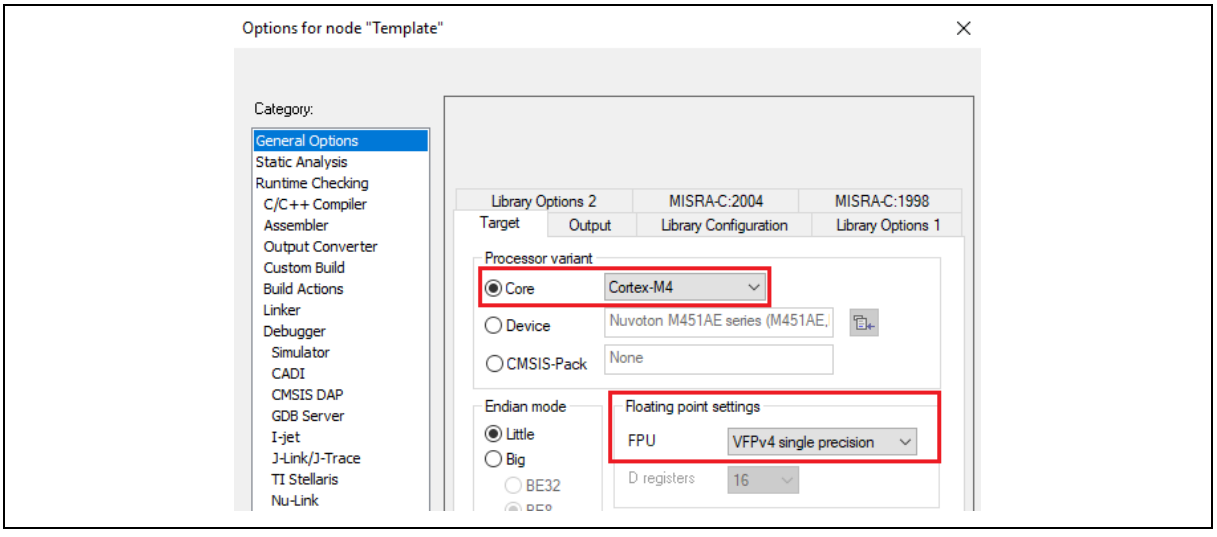


Figure 2-11 General Options

Click “Override default” option to edit linker configuration for debugging on SRAM address shown in Figure 2-12.

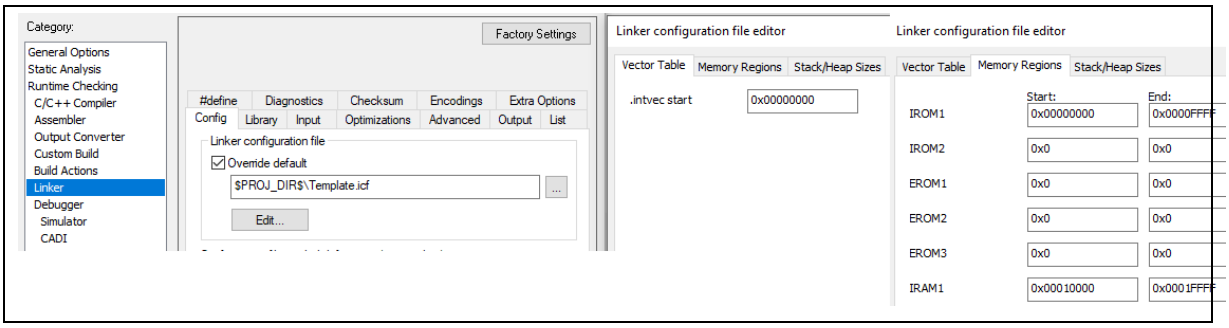


Figure 2-12 Linker Options

Select CMSIS-DAP driver and uncheck “Use flash loader” option shown in Figure 2-13.

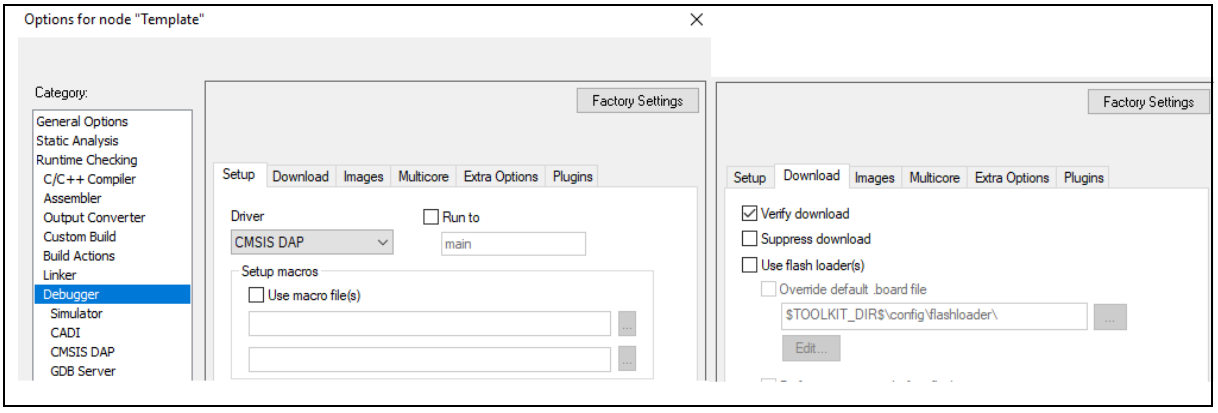


Figure 2-13 Debugger Options

Select SWD interface for CMSIS-DAP setting shown in Figure 2-14.Figure 2-13

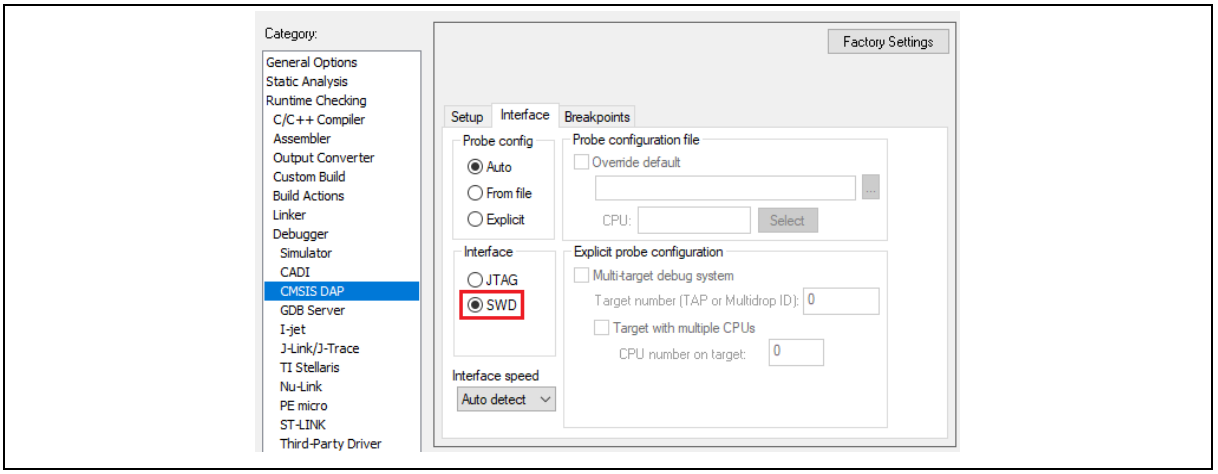


Figure 2-14 CMSIS-DAP Interface Setting

To build the workspace click the “Make” icon on shown in Figure 2-15.

Figure 2-15 Build IAR Workspace

Modify “CMSIS-DAP Memory Configuration” base on SRAM address range shown in

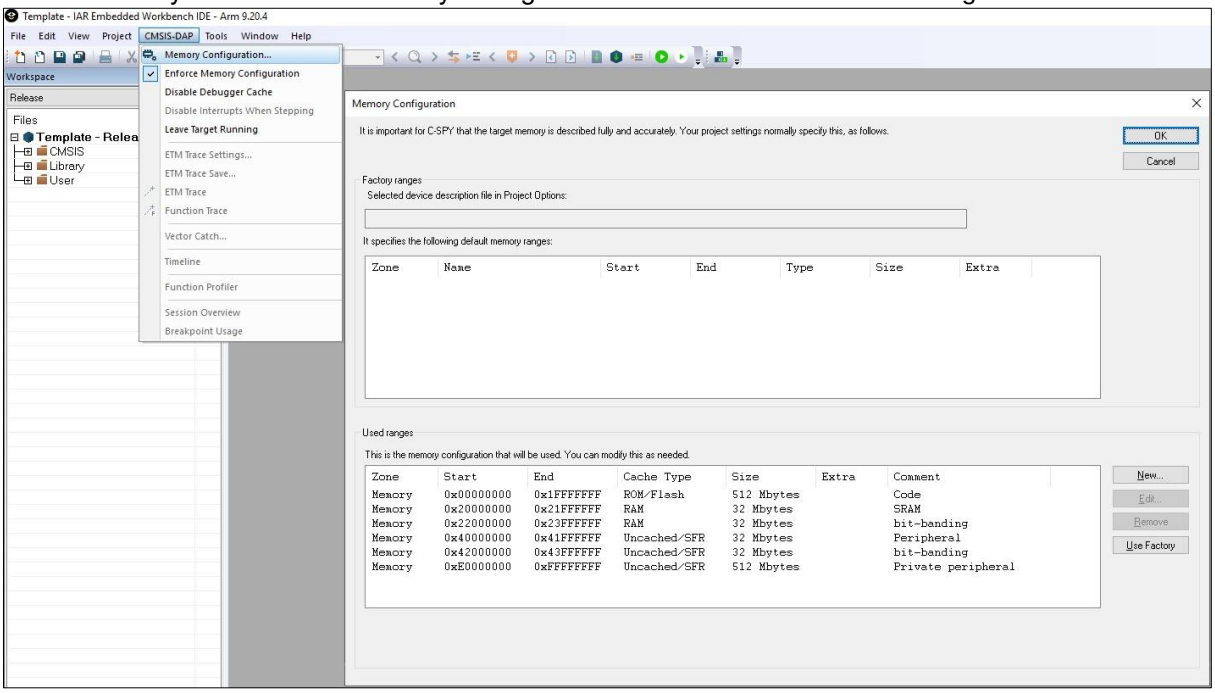


Figure 2-16.

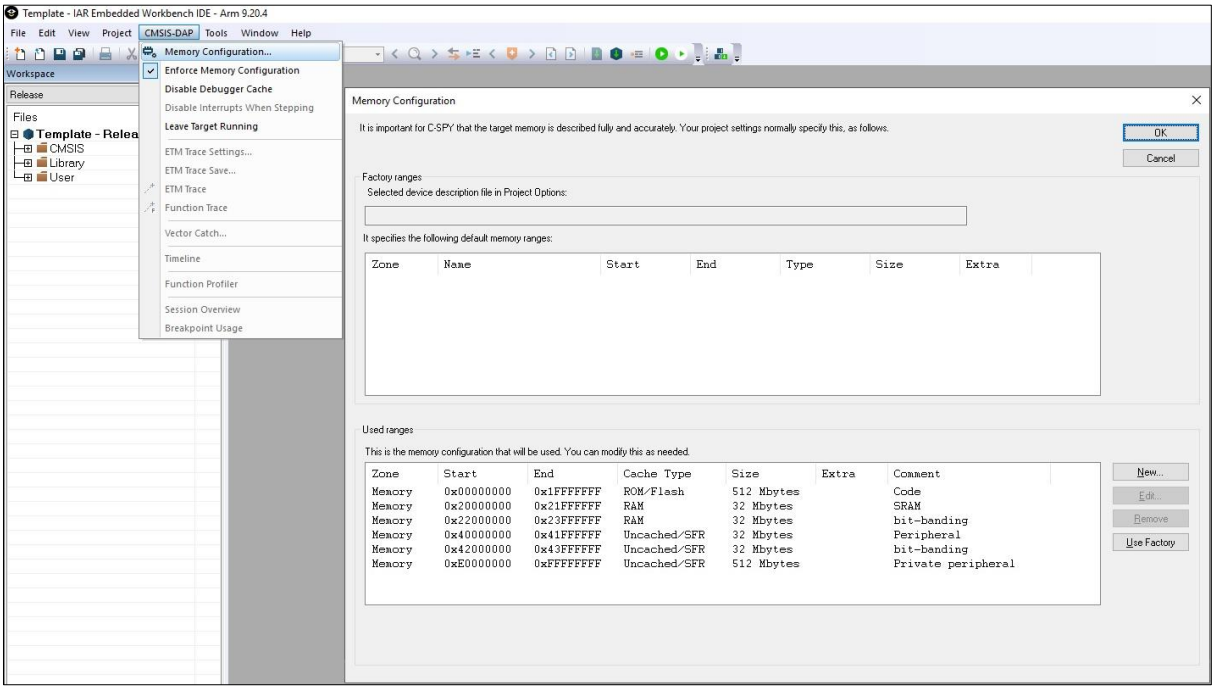


Figure 2-16 CMSIS-DAP Memory Configuration

IAR will load the built image and enter debug mode after click the “Download and Debug” icon as shown in Figure 2-17.

Figure 2-17 Debug IAR Project

2.4 NuEclipse

To use open the NuEclipse project, first launch the NuEclipse tool, select “Import…” from the File menu as shown in Figure 2-18.

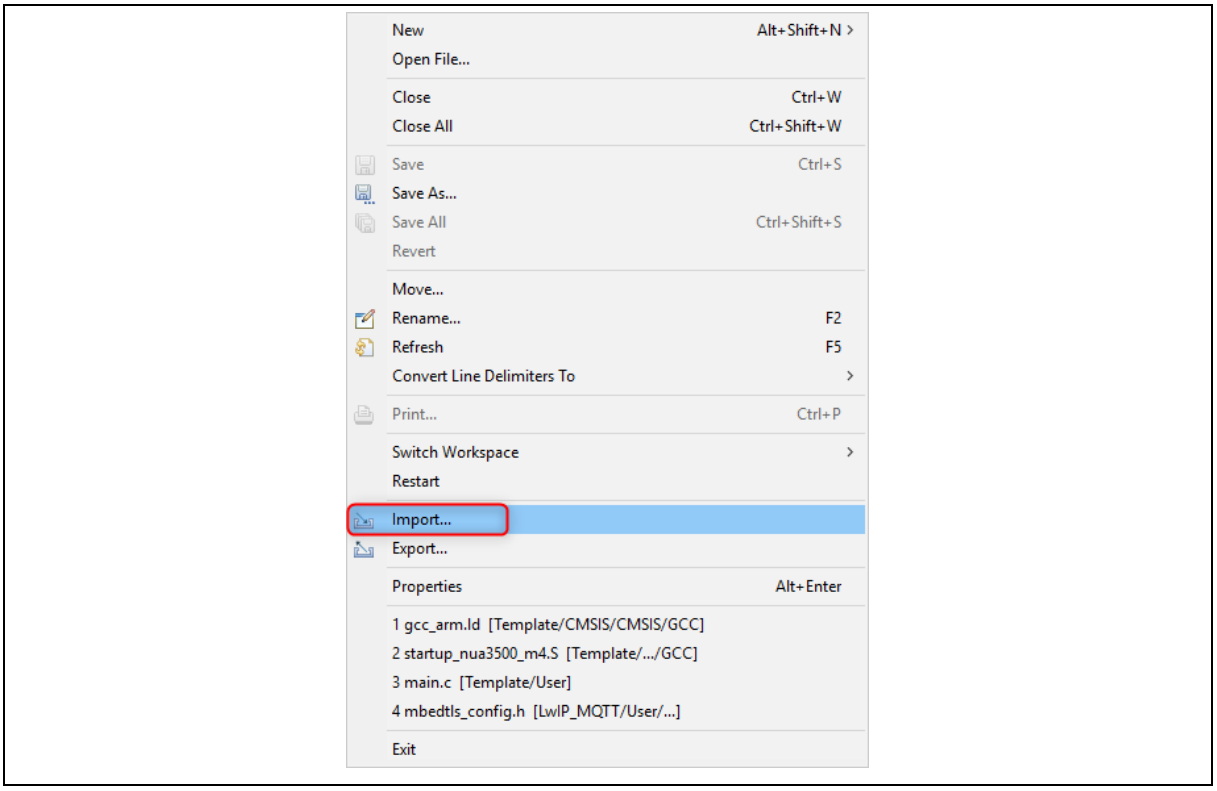


Figure 2-18 NuEclipse Import Project

The next step is to select import existing project to workspace and click “Next >” button as shown in Figure 2-19.

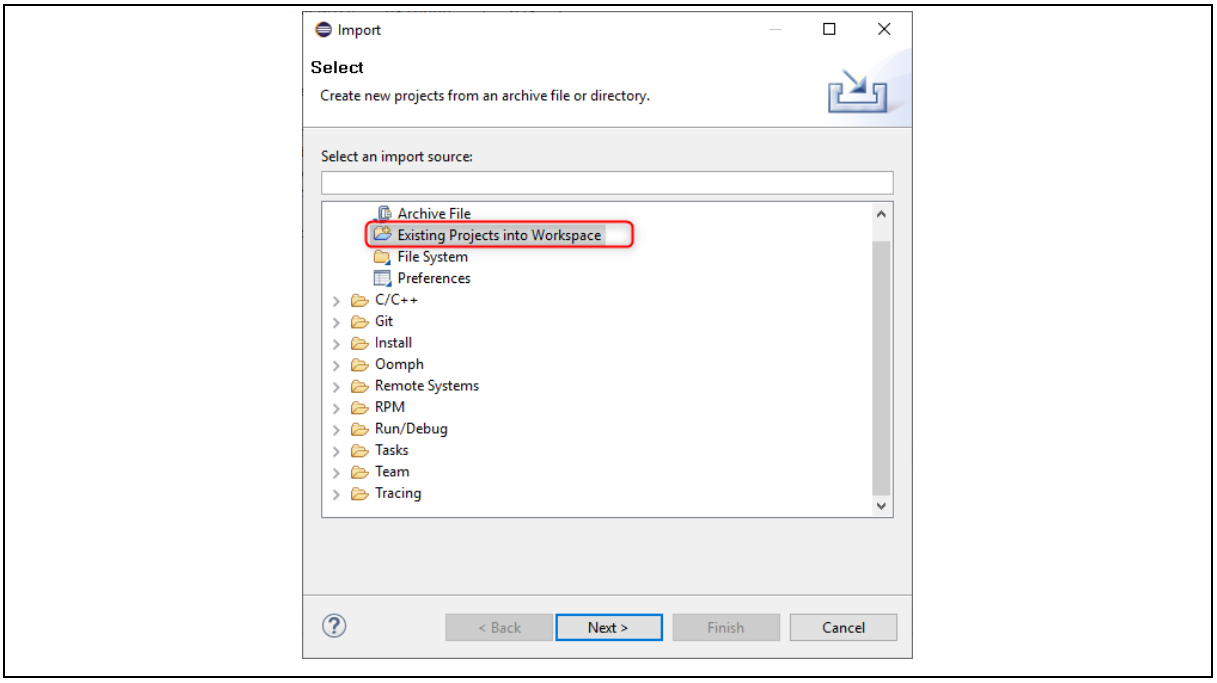


Figure 2-19 NuEclipse Import Existing Project

And the last step is to select the project file, and then click “Finish” button.

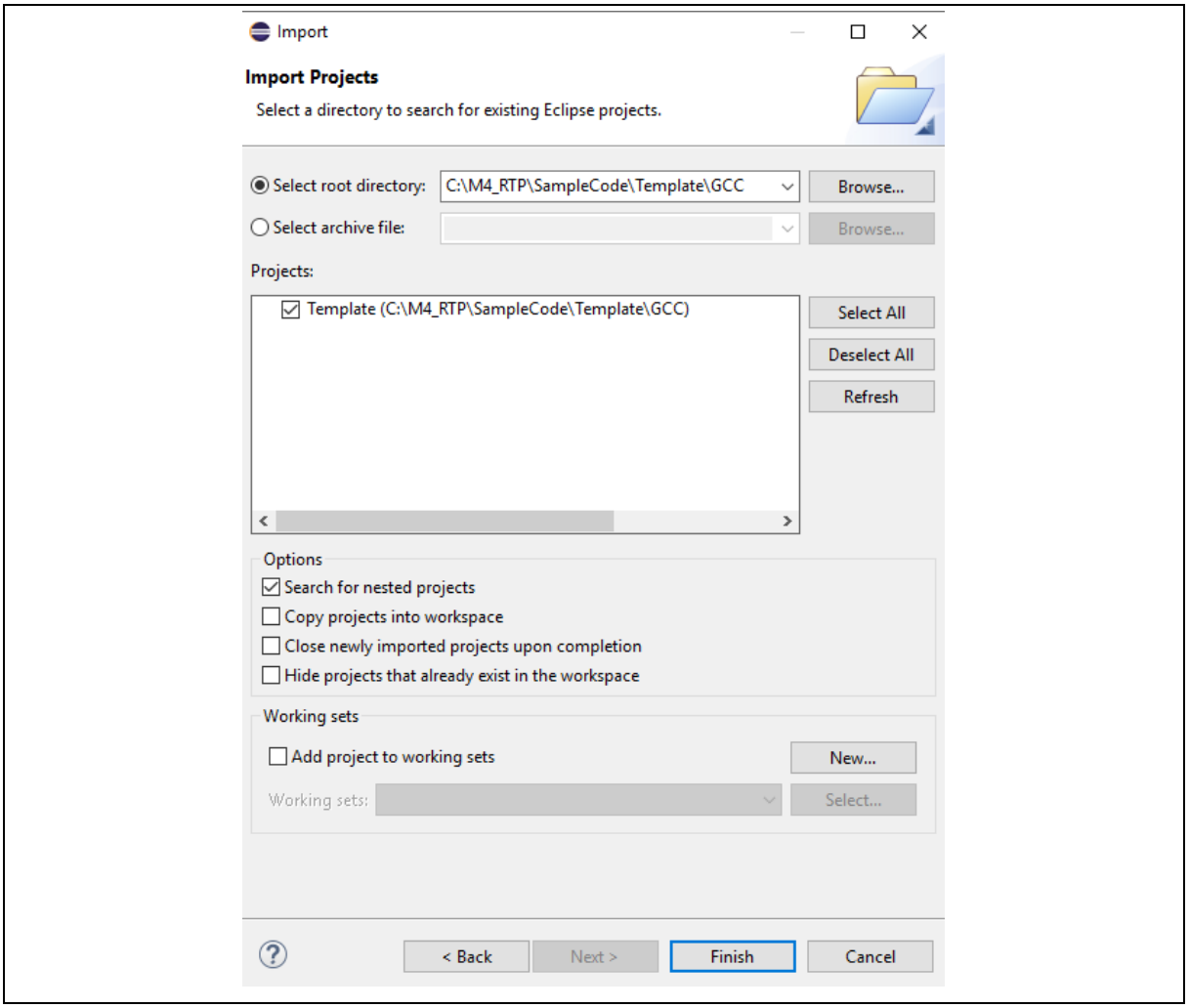


Figure 2-20 NuEclipse Select Project

To Build NuEclipse project, click the build icon as shown in Figure 2-21 or use the Ctrl + B hotkey.

Figure 2-21 Build NuEclipse Project

Click “Debug Configuration” menu item shown in Figure 2-22.

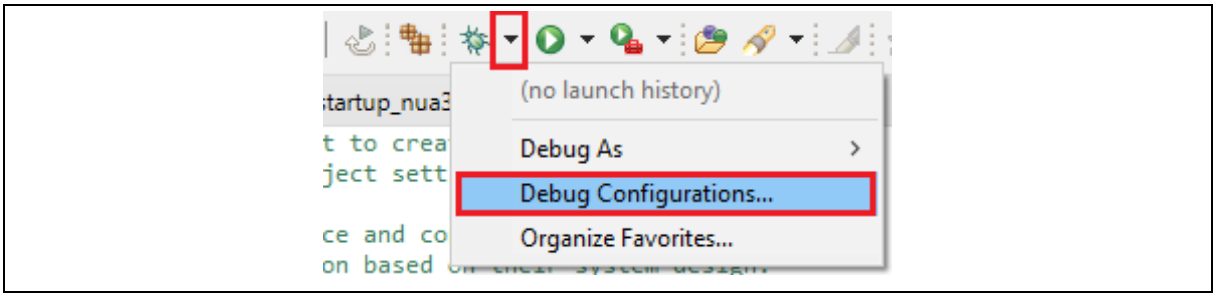


Figure 2-22 Debug Configuration

Double click on the “GDB Nuvoton Nu-Link Debugging” group. The Nuvoton Nu-Link debug configuration appears on the right-hand side.

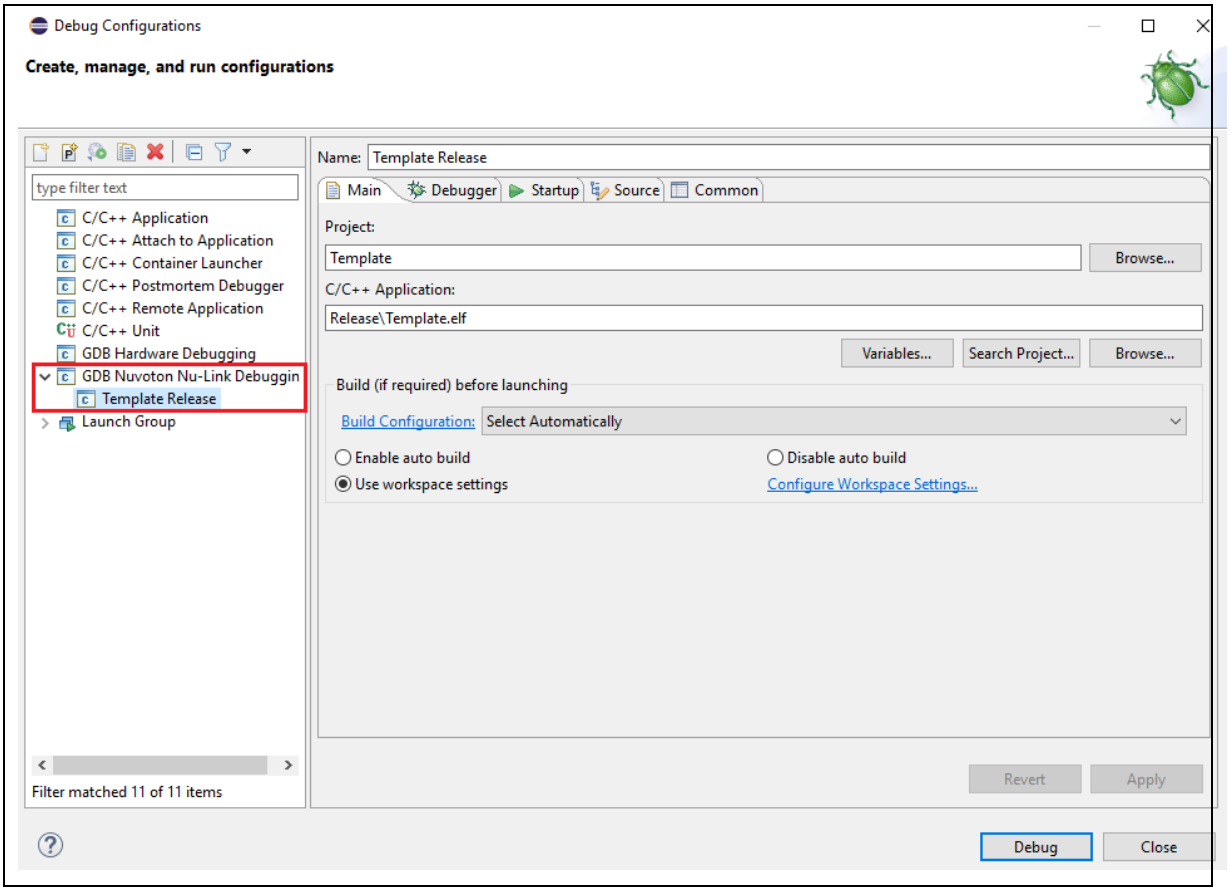


Figure 2-23 GDB Nuvoton Nu-Link Debugging

The value of GDB client port is set to 3334.

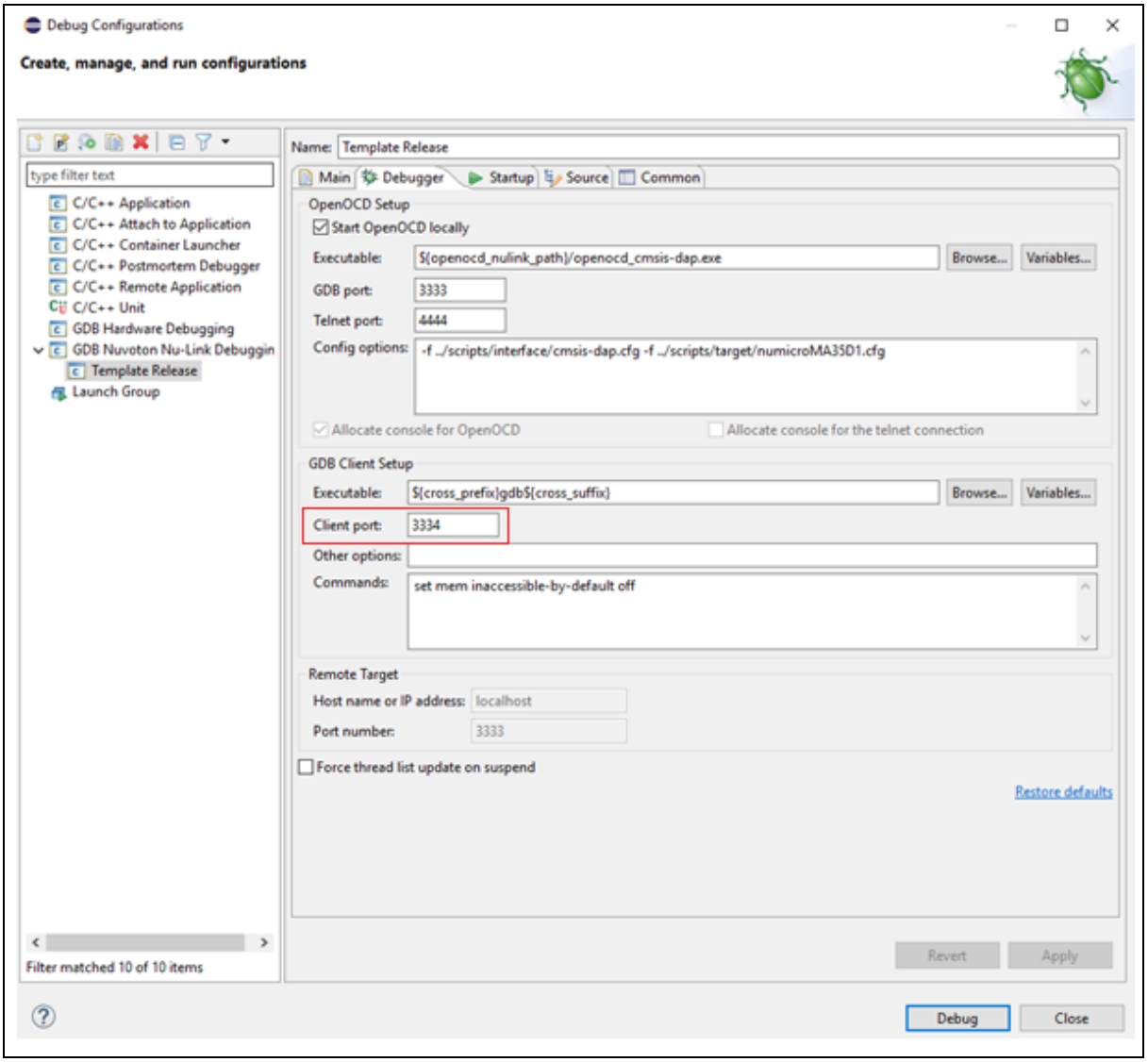


Figure 2-24 GDB Client Port

Check the startup debug setting shown in Figure 2-25, and then click “Debug” button. NuEclipse will load the built image and enter debug mode.

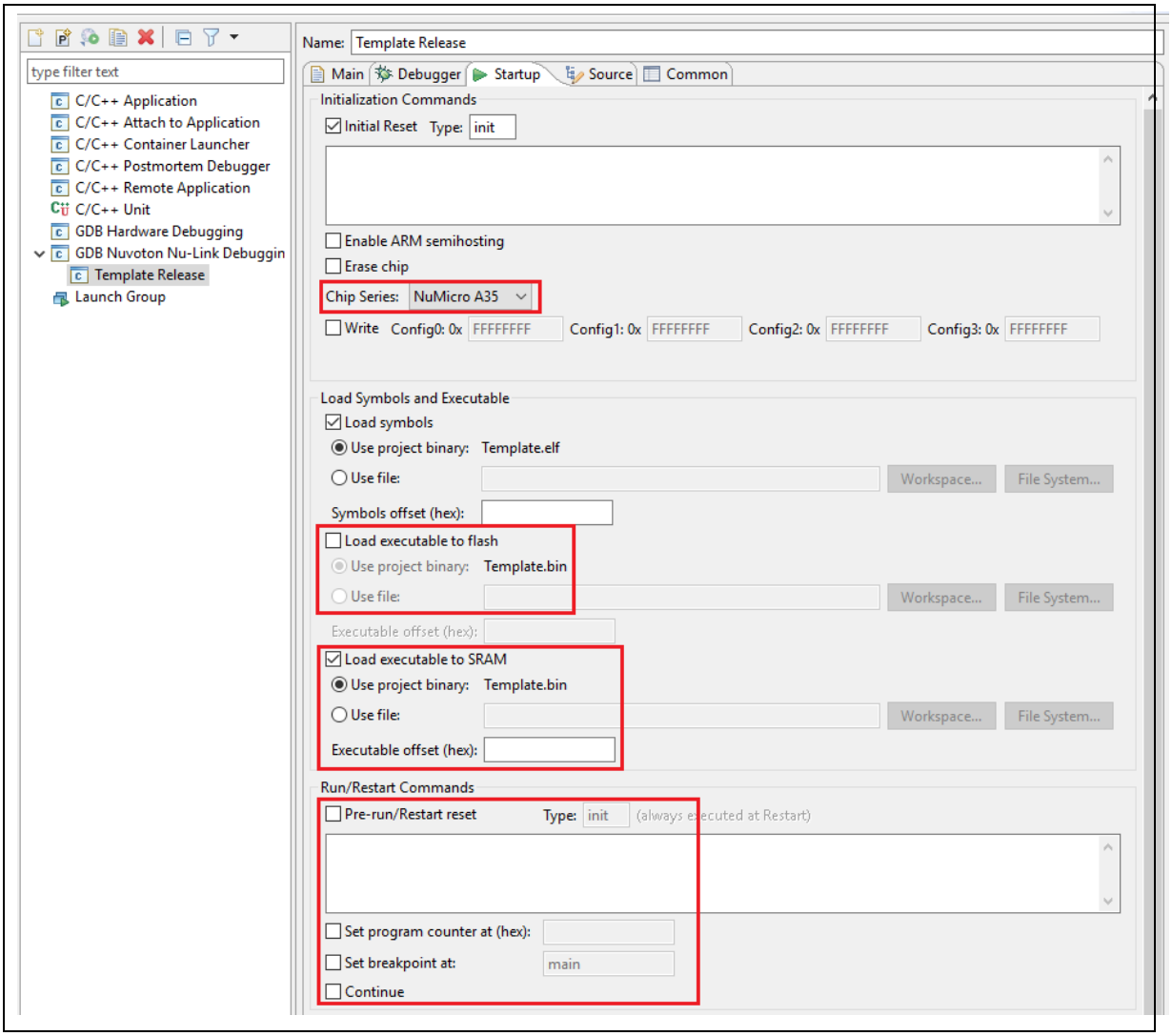


Figure 2-25 Startup Debug Setting

3 DIRECTORY STRUCTURE

This chapter describes the directory structure of RTP 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, including FreeRTOS™, and OpenAMP.

Table 3-1 First Level Directory

3.2 Document

Directory	Description
CMSIS.html	Document of CMSIS version 4.5.0.
NuMicro MA35D1 RTP Driver Reference Guide.html	This document describes the usage of drivers in MA35D1 RTP software package.

Table 3-2 Document Directory

3.3 Library

Directory	Description
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) V4.5.0 definitions by Arm® Corp.
Device	CMSIS compliant device header file.
SmartcardLib	Smartcard library binary and header file.
StdDriver	All peripheral driver header and source files.

Table 3-3 Library Directory

3.4 SampleCode/CortexM4

3.4.1 DSP_FF

Sample Name	Description
DSP_FF	Demonstrate how to call ARM CMSIS DSP library to calculate FFT.

3.4.2 MPU

Sample Name	Description
MPU	Demonstrate the usage of Cortex-M4 MPU.

3.5 SampleCode/FreeRTOS

3.5.1 FreeRTOS

Sample Name	Description
FreeRTOS	A template of FreeRTOS project.

3.6 SampleCode/OpenAMP

3.6.1 OpenAMP

Sample Name	Description
Share_memory_demo	This sample code is designed for non-OS & Linux. The provided sample code is based on rpmsg v1, which utilizes "SRAM" as shared memory. 1. This project involve RTP portion of share memory demo, please also review the sample code in non-OS BSP. https://github.com/OpenNuvoton/MA35D1_NonOS_BSP/tree/master/SampleCode/OpenAMP/Share_Memory_SRAM 2. This project involve RTP portion of share memory demo, please also review the sample code in Linux applications. https://github.com/OpenNuvoton/MA35D1_Linux_Applications/tree/master/examples/rpmsg
Share_Memory_SDRAM	This sample code is designed for non-OS environments only. The provided sample code is based on rpmsg v1, which utilizes "DRAM" as shared memory. This project involves RTP portion of share memory demo, please also review the sample code in non-OS BSP. https://github.com/OpenNuvoton/MA35D1_NonOS_BSP/tree/master/SampleCode/OpenAMP/Share_Memory_SDRAM

rpmsg_rtp	This sample code is designed for Linux only. The provided sample code can be configured for rpmsg v1 or rpmsg v2, which utilizes SRAM or DRAM as shared memory. If recursive read/write is necessary in your application, it is highly recommended to use the v2 architecture. This project involve RTP portion of rpmsg demo, please also review the sample codes in Linux applications. 1. rpmsg-v1 https://github.com/OpenNuvoton/MA35D1_Linux_Applications/tree/master/examples/rpmsg 2. rpmsg-v2 https://github.com/OpenNuvoton/MA35D1_Linux_Applications/tree/master/examples/rpmsg-v2

3.7 SampleCode/StdDriver

Here is the complete standard driver sample code list and the description.

3.7.1 Wormhole Controller (WHC)

Sample Name	Description
WHC_TxRx	Demonstrate send and receive function through wormhole channel.

Table 3-4 WHC Samples

3.7.2 Hardware Semaphore (HWSEM)

Sample Name	Description
HWSEM_LockUnlock	Demonstrate hardware semaphore lock and unlock function.

Table 3-5 HWSEM Samples

3.7.3 External Bus Interface (EBI)

Sample Name	Description
EBI_NOR	Configure EBI interface to access NOR Flash connected to EBI interface.
EBI_SRAM	Configure EBI interface to access SRAM connected to EBI interface.

Table 3-6 EBI Samples

3.7.4 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 3-7 GPIO Samples

3.7.5 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 PDMA timeout feature.

Table 3-8 PDMA Samples

3.7.6 Timer Controller (TIMER)

Sample Name	Description
TIMER_CaptureCounter	Show how to use the timer2 capture function to capture timer2 counter value.
TIMER_Delay	Demonstrate the usage of TIMER_Delay() API to generate a 1 second delay.
TIMER_EventCounter	Use pin PJ.12 to demonstrate timer event counter function.
TIMER_FreeCountingMode	Use the timer pin PJ.13 to demonstrate timer free counting mode function. And displays the measured input frequency to UART console.

TIMER_InterTimerTrigger Mode	Use the timer pin PJ.12 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 by Timer PWM Double Buffer function.
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 2 toggle out function on pin PJ.12.

Table 3-9 TIMER Samples

3.7.7 Watchdog Timer (WDT)

Sample Name	Description
WDT_TimeoutWakeup AndReset	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 3-10 WDT Samples

3.7.8 Window Watchdog Timer (WWDT)

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

Table 3-11 WWDT Samples

3.7.9 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 3-12 RTC Samples

3.7.10 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 Time function.
EPWM_DoubleBuffer	Change duty cycle and period of output waveform by EPWM Double Buffer function.
EPWM_OutputWaveform	Demonstrate how to use PWM 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 PWM counter synchronous start function.

Table 3-13 EPWM Samples

3.7.11 Enhanced Input Capture Timer (ECAP)

Sample Name	Description
ECAP_GetInputFreq	Show how to use ECAP to measure clock frequency.
ECAP_GetQEIFreq	Show how to use ECAP interface to get QEIA frequency.

Table 3-14 ECAP Samples

3.7.12 Quadrature Encoder Interface (QEI)

Sample Name	Description
QEI_CompareMatch	Show the usage of QEI compare function.

Table 3-15 QEI Samples

3.7.13 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 3-16 UART Samples

3.7.14 Smartcard Host Interface (SC)

Sample Name	Description
SC_ReadATR	Read the smartcard ATR from smartcard 1 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 PF.10 and PF.11 pins.

Table 3-17 SC Samples

3.7.15 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	Demonstrate QSPI1 3-wire mode.

Table 3-18 QSPI Samples

3.7.16 Serial Peripheral Interface (SPI)

Sample Name	Description
SPI_Flash	Access SPI Flash through a SPI interface.
SPI_HalfDuplex	Demonstrate SPI half-duplex mode.
SPI_LoopBack	A SPI read/write demo connecting SPI0 MISO and MOSI pins.
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. QSPI1 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 I²S Master mode and demonstrate how I²S works in Master mode.
SPII2S_Slave	Configure SPI1 as I2S Slave mode and demonstrate how I2S works in Slave mode. This sample code needs to work with SPII2S_Master.

Table 3-19 SPI Samples

3.7.17 I²C Serial Interface Controller (I²C)

Sample Name	Description
I2C_EEPROM	Read/write EEPROM via I²C interface.
I2C_Loopback	Demonstrate how a Master accesses Slave.

I2C_Master	An I²C master mode demo code.
I2C_MultiBytes_Master	Demonstrate how to use multi-bytes API to access slave. This sample code needs to work with I2C_Slave.
I2C_PDMA_TRX	Demonstrate I2C PDMA mode that needs to connect I2C0 (Master) and I2C1 (Slave).
I2C_SingleByte_Master	Demonstrate how to use single byte API to access slave. This sample code needs to work with I2C_Slave.
I2C_Slave	An I²C slave mode demo code.
I2C_Wakeup_Slave	Demonstrate how to set I2C to wake up MCU from Power-down mode. This sample code needs to work with I2C_Master.

Table 3-20 I²C Samples

3.7.18 I²S Controller (I²S)

Sample Name	Description
I2S_Codec	An I²S demo used to play back the input from line-in or MIC interface.
I2S_Codec_PDMA	An I²S with PDMA demo used to play back the input from line-in or MIC interface.

Table 3-21 I²S Samples

3.7.19 M-Controller Area Network (MCAN)

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 3-22 MCAN Samples

3.7.20 Enhanced 12-bit Analog-to-Digital Converter (EADC)

Sample Name	Description
EADC_EPWM_Trigger	Demonstrate how to trigger EADC by EPWM.
EADC_PDMA_EPWM_ Trigger	Demonstrate how to trigger EADC by EPWM and transfer conversion data by PDMA.
EADC_Pending_Priority	Demonstrate how to trigger multiple sample modules and got conversion results in order of priority.
EADC_ResultMonitor	Monitor the conversion result of channel 2 by the digital compare function.
EADC_SWTRG_Trigger	Trigger EADC by writing EADC_SWTRG register.
EADC_Timer_Trigger	Show how to trigger EADC by timer.

Table 3-23 EADC Samples

3.7.21 Analog-to-Digital Converter (ADC)

Sample Name	Description
ADC_Convert	Demonstrate normal ADC convert function.
ADC_Touch	Show how to use ADC to detect touch screen event.

Table 3-24 ADC Samples

3.7.22 Keypad Interface (KPI)

Sample Name	Description
KPI	Demonstrate KPI Controller.

Table 3-25 ADC Samples

3.8 ThirdParty

Directory	Description
FreeRTOS	A real time operating system available for free download. Its official website is: http://www.freertos.org/.
AMP	The OpenAMP framework provides software components that enable development of software applications for Asymmetric Multiprocessing (AMP)

	systems. Its official website is: https://www.openampproject.org/.

Table 3-26 ThirdParty Directory

4 LOAD FIRMWARE FOR EXECUTION

RTP firmware could be load by ICE for debugging or load by TF-A or Linux. Chapter 2 covers the steps to load and debug firmware by ICE. Here describes how to load RTP firmware from TF-A or Linux executing on A35 core.

4.1 Load RTP Image From Linux

MA35D1 supports to load executable images into RTP M4 SRAM for from the Cortex-A35 side using the Linux built-in remoteproc framework. Of course, the user needs to copy the compiled firmware to the Linux file system before using the Cortex-A35 to load the RTP M4 images. Currently Linux only supports loading firmware in ELF format. Users can also use the remoteproc driver to start/stop RTP M4 CPU execution.

The default configuration files in MA35D1 Yocto and Buildroot distribution enable the remoteproc support by default. The Linux kernel configuration about remoteproc is shown as the following:

```
Device Drivers  --->
  Remoteproc drivers  --->
       [*] Support for Remote Processor subsystem
       <*>   MA35D1 remoteproc support
```

The device tree node settings of remoteproc is also enabled:

```
rproc {
              compatible = "nuvoton,ma35d1-rproc";
              mboxes = <&wormhole 1>;
              resets = <&reset MA35D1_RESET_CM4>;
              status = "okay";
       };
```

After starting the Linux kernel, user can load and start/stop the remote processor firmware through the sysfs interface (The sysfs filesystem is a pseudo-filesystem which provides an interface to kernel data structures). The firmware components are stored in the file system, by default in the /lib/firmware/ folder. Optionally another location can be set. In this case, the remoteproc framework parses this new path in priority. The below command adds a new path for firmware parsing:

```
$> echo -n <firmware_path> > /sys/module/firmware_class/parameters/path
```

Use the following command to specify the firmware file name. Where the “X” of remoteprocX is the remoteproc instance number, and is 0 by default.

```
$> echo -n <firmware_name.elf> > /sys/class/remoteproc/remoteprocX/firmware
```

Then users can use the following command to load and start the firmware:

```
$> echo start > /sys/class/remoteproc/remoteprocX/state
```

And users can use the following command to stop the firmware:

```
$> echo stop > /sys/class/remoteproc/remoteprocX/state
```

4.2 Load RTP Image From TF-A

MA35D1 supports to load executable images into RTP M4 SRAM from the Cortex-A35 side using the TF-A. The file format for loading RTP images from TF-A is raw binary, unlike loading ELF format RTP images in Linux.

Both MA35D1 buildroot and Yocto Project support configuring to load the RTP M4 binary image from TF-A.

4.2.1 Buildroot

If you are developing a project using MA35D1 buildroot, to configure loading the RTP M4 binary image from TF-A, first, enter the buildroot menuconfig to perform the configuration.

```
$ make menuconfig
```

Navigate to the 'Bootloaders' submenu and check 'Add SCP BL2 image into FIP Image'. Once checked, 'Load Image into FIP Image' will automatically appear. Please select 'RTP M4 Image'. Also, specify the '(RTP-BSP/Template.bin) SCP_BL2 binary file names', which is the default path for the RTP M4 binary image. It is relative to the actual path of buildroot, which is the 'output/image/RTP-BSP' directory.

```
Bootloaders  --->
    [*] ARM Trusted Firmware (ATF)
    [*]   Add SCP BL2 image into FIP Image
            Load Image into FIP Image (RTP M4 Image)  --->
    (RTP-BSP/Template.bin) SCP_BL2 binary file names
```

After the configuration is complete, proceed with compilation.

```
$ make
```

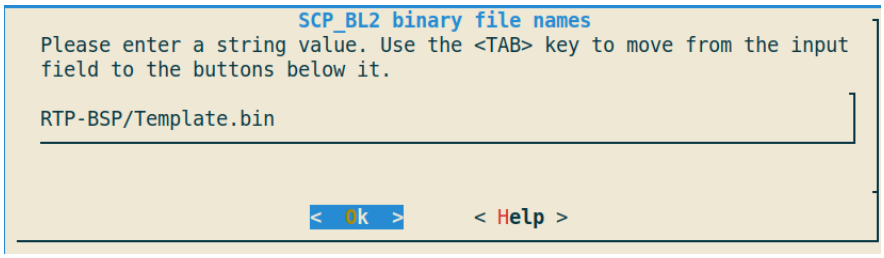
After compilation is complete, you will find an additional RTP-BSP directory in the ‘output/images’ directory. Inside, you will find raw binary and ELF images for all examples of the MA35D1 RTP BSP.

```
$ ls output/images/RTP-BSP/ -l
total 15400
-rwxr-xr-x 1 osboxes osboxes   6780 Apr  2 10:15 ADC_Convert.bin
-rwxr-xr-x 1 osboxes osboxes 199596 Apr  2 10:15 ADC_Convert.elf
-rwxr-xr-x 1 osboxes osboxes   7060 Apr  2 10:15 ADC_Touch.bin
-rwxr-xr-x 1 osboxes osboxes 206892 Apr  2 10:15 ADC_Touch.elf
-rwxr-xr-x 1 osboxes osboxes  31184 Apr  2 10:15 CANFD_CANFD_Loopback.bin
-rwxr-xr-x 1 osboxes osboxes 246172 Apr  2 10:15 CANFD_CANFD_Loopback.elf
-rwxr-xr-x 1 osboxes osboxes  34008 Apr  2 10:15 CANFD_CANFD_TxRx.bin
-rwxr-xr-x 1 osboxes osboxes 246772 Apr  2 10:15 CANFD_CANFD_TxRx.elf
...
```

The raw binary image loaded by default in TF-A is ‘Template.bin’. If you want to load a different binary, please re-enter the buildroot menuconfig menu, navigate to ‘Bootloaders’ --> ‘(RTP-BSP/Template.bin) SCP_BL2 binary file names’, and press 'Enter' to make the modification.

```
Bootloaders  --->
    (RTP-BSP/Template.bin) SCP_BL2 binary file names
```

In the subsequent editing window that pops up, modify Template.bin to the desired raw binary file name you want to load. Please note that only raw binary files are supported here, and ELF files are not supported. Please ensure that the raw binary you want to load exists in the 'output/images/RTP-BSP' directory.



If you have changed the raw binary file, you must recompile TF-A with a clean build.

```
$ make arm-trusted-firmware-dirclean
$ make arm-trusted-firmware-rebuild
$ make
```

The final compiled pack image should include the RTP M4 image. We can determine if TF-A has loaded the RTP M4 image from the boot message of MA35D1. In the TF-A boot message, you should be able to find the following message in blue font, indicating that TF-A is loading the RTP M4 image.

```
NOTICE:  BL2: v2.3(release):tc_v1.00-117-gbd27a7b71c
NOTICE:  BL2: Built : 02:13:07, Apr  3 2024
INFO:    BL2: Loading image id 2
INFO:    Loading image id=2 at address 0x85000000
INFO:    Image id=2 loaded: 0x85000000 - 0x8500181c
INFO:    Load SCP_BL2
INFO:    BL2: Doing platform setup
INFO:    BL2: Loading image id 3
INFO:    Loading image id=3 at address 0x28025000
```

After entering the Linux shell, you can dump the contents of the RTP M4 SRAM, which starts at address 0x24000000. It should match the contents of the RTP binary you specified.

```
# devmem 0x24000000
0x00040000
# devmem 0x24000004
0x00000619
# devmem 0x24000040
0x00000641
# devmem 0x24000200
0xB9337823
```

The BSP defaults to using UART16 for RTP M4. If the RTP M4 image has been loaded by TF-A and is running correctly, you should be able to see its printed messages from UART16.

4.2.2 Yocto

If you are developing a project using MA35D1 yocto, to configure loading the RTP M4 binary image from TF-A, first, open and modify file “source/meta-ma35d1/conf/machine/numaker-som-ma35d16a81.conf”. In the file, locate TFA_LOAD_M4, which is by default set to "no". This setting instructs TF-A not to load the RTP M4 image. Please modify it to "yes". Additionally, in the TFA_M4_BIN item, input the filename of the M4 image to load. The m4proj directory is located at "tmp-glibc/deploy/images/numaker-som-ma35d16a81/m4proj".

```
# Load RTP-M4 into FIP image and run RTP-M4
TFA_LOAD_M4 = "yes"

# Need to set binary file from deploy/images/{machine}
TFA_M4_BIN= "m4proj/Template.bin"
```

If you have changed the raw binary file, you must recompile TF-A and BSP.

```
$ bitbake tf-a-ma35d1 –C compile
$ bitbake nvt-image-qt5
```

The final compiled pack image should include the RTP M4 image. We can determine if TF-A has loaded the RTP M4 image from the boot message of MA35D1. In the TF-A boot message, you should be able to find the following message in blue font, indicating that TF-A is loading the RTP M4 image.

```
NOTICE:  BL2: v2.3(release):tc_v1.00-117-gbd27a7b71c
NOTICE:  BL2: Built : 02:13:07, Apr  3 2024
INFO:    BL2: Loading image id 2
INFO:    Loading image id=2 at address 0x85000000
INFO:    Image id=2 loaded: 0x85000000 - 0x8500181c
INFO:    Load SCP_BL2
INFO:    BL2: Doing platform setup
INFO:    BL2: Loading image id 3
INFO:    Loading image id=3 at address 0x28025000
```

5 RESOURCE MANAGEMENT

In MA35D1, the only resource that is bounded with Cortex-M4 is WDT2 and WWDT2. To prevent contention, MA35D1 use System Security Memory Configuration Controller (SSMCC) to control the memory space that can access by Cortex-M4. And use System Security Peripheral Configuration Controller (SSPCC) to assign the peripherals that are controlled by Cortex-M4. A wormhole controller (WHC) is introduced to allow Cortex-A35 and Cortex-M4 to communicate with each other. To prevent Cortex-A35 and Cortex-M4 access share memory simultaneously causing data corruption, Hardware Semaphore (HWSEM) is added in MA35D1.

This chapter describes the usage of the IPs mentioned above.

5.1 SSMCC

SSMCC is a memory configurator based on Arm® TZC-400. Only Cortex-A35 can configure SSMCC while executing in secure state. So it is necessary to assign the resource and configure with BL2 where all the system resource is getting initialed and assigned or later by BL31/BL32 at run time. SSMCC can configured the DDR ranged that can access by Cortex-M4.

5.1.1 DDR

The 128 KB SRAM0 should be sufficient to hold the code and data used by Cortex-M4 in most of the use scenario. In the case that Coretex-M4 requires more memory space, user can configure SSMCC allowing Cortex-M4 to share the first 4MB of DDR (Note: This is set in TF-A by default). The total memory accessible by Cortex-M4, while 0~128KB maps to SRAM0, and 128KB~4MB maps to DDR.

MA35D1 consists of I-cache and D-cache for Cortex-M4 that can be enabled by setting RTPDCACHEN (SYS_MISCFCR0[1]) and RTPICACHEN (SYS_MISCFCR0[0]) bits to mitigate the performance drop due to the longer access path. These caches have no role in the SRAM access path so should keep disabled to save power consumption unless DDR is shared by Cortex-M4. Please note that application can still suffer 25% CoreMark score drop with cache on while executing on DDR.

5.2 SSPCC

SSPCC, is used to configure the security attribute of SRAM, GPIO and all other peripherals for Cortex®-A35 and Cortex®-M4 and should be configured before the GPIO and peripherals cab be control by Corex-M4. Otherwise read or write to the peripheral control register will be ignored. Individual peripheral attribute setting is control by SSPCC_PSSET0 ~ SSPCC_PSSET11 registers. EBI attribute is controlled by SSPCC_EBISSET register, and GPIO attribute is controlled by SSPCC_IOxSSET (where x= A~N) registers. The attribute of peripheral and GPIO must be consistent. For example, for an I^2C interface assigned to Cortex-M4, the GPIO occupied by its SCL and SDA function must be assigned to Cortex-M4 as well. Otherwise the behavior is undefined. Like SSMCC, only Cortex-A35 execute in secure state can configure SSPCC.

5.2.1 SRAM

The 128KB SRAM0 is designed to be used by Cortex-M4 to store code and data. By default, the whole SRAM0 can be shared between Cortex-M4 and Cortex-A35. To prevent the SRAM0 accidentally corrupt by Cortex-A35 after Cortex-M4 is up and execute, SSPCC provides a feature that can restrict certain range of SRAM that can only access by Cortex-M4.

The SR0BOUND(SSPCC_SRAMSB[4:0]) is used to set a 16 Kbytes-aligned boundary region starting from offset 0 of SRAM0 that can only access by Cortex-M4. The region above the boundary will be the share memory between Cortex-A35 and Cortex-M4.

Another 256KB internal SRAM1 is reserved for Cortex-A35 only and cannot access by Cortex-M4.

5.3 WHC

Except exchanging data through share memory, MA35D1 provides WHC for each processor to deliver and receive data through unidirectional message channels. There are 4 channels each direction, and

every channel can send 4 words of message each time. WHC provides a TX status register WHCx_TXSTS allowing the sender to know if the data has been read by receiver or not, a RX status register WHCx_RXSTS notify if there's new message coming in.

Besides message exchange, there's also a WHCx_CPSTS register indicates the counterpart operating and reset status. And WHCx_GINTTRG register allowing trigger interrupt without message exchange.

5.4 HWSEM

Since there is memory space that are share between different cores, it is necessary to implement a synchronization mechanism to prevent different core access same memory space and results to unpredictable behavior. The Hardware Semaphore block provides 8 hardware semaphores, which can be used to synchronize between different processors using different semaphore keys.

Register HWSEM_SEM0 ~ HWSEM_SEM7 can be access by both Cortex-A35 and Cortex-M4 are used to control the 8 semaphores for synchronization. A write to these register will try to hold the semaphore. ID (HWSEM_SEMx[3:0]) indicates the current owner of the semaphore and a write to these with the same KEY (HWSEM_SEMx[15:8]) unlock the semaphore.

6 INTERCOMMUNICATION BETWEEN A35 AND RTP

In the MA35D1 development environment, the communication between the RTP M4 and Cortex-A35, whether running Non-OS or Linux, is based on the OpenAMP architecture, utilizing rpmsg as the underlying communication protocol, and the handshake protocol is implemented by the MA35D1 Wormhole hardware.

6.1 OpenAMP and rpmsg

OpenAMP is a software framework developed by Linaro that simplifies managing heterogeneous processors (APs) on multi-core processor systems. It provides a set of APIs (Application Programming Interfaces) that allows developers to interact with and control these APs effectively.

The rpmsg is a lightweight messaging protocol specifically designed for communication between processors, OpenAMP utilizes rpmsg as its underlying communication layer. OpenAMP builds upon rpmsg by offering a broader set of functionalities for comprehensive AP management.

User code can call the OPENAMP_create_endpoint() function to create a callback function to receive messages from the ARM core. And call OPENAMP_check_for_message() function to check whether there is a message from ARM Core. Also can call the OPENAMP_send() function to send message to ARM core.

6.2 RTP Samples using OpenAMP

In the MA35D1 RTP BSP, there are several examples available for using rpmsg communication with OpenAMP. The example program directory is located at "RTP\SampleCode\OpenAMP". Within this directory, there is a README file. Please read this file to ensure the correct selection and usage of example programs.

6.2.1 Share_memory_demo

This sample code is designed for communicate with the Cortex-A35 running non-OS or Linux. The provided sample code is based on MA35D1 rpmsg v1, which utilizes "SRAM" as shared memory. The shared memory base and size are defined in “Share_memory_demo\porting\openamp_conf.h”. The default setting for this shared memory is based on 0x2401ff00 with size 128 * 2 bytes.

If the Cortex-A35 is running Non-OS, the corresponding sample code is https://github.com/OpenNuvoton/MA35D1_NonOS_BSP/tree/master/SampleCode/OpenAMP/Share_Memory_SRAM.

If the Cortex-A35 is running Linux, the corresponding sample code is https://github.com/OpenNuvoton/MA35D1_Linux_Applications/tree/master/examples/rpmsg.

Please note that due to limitations in MA35D1 rpmsg v1, there must be a short time gap between the transmission of messages and the next message to avoid potential ACK loss. For Linux user, it is highly recommended to adopt the MA35D1 rpmsg v2 solution.

6.2.2 Share_Memory_SDRAM

This sample code is designed for non-OS environments only. The provided sample code is based on MA35D1 rpmsg v1, which utilizes "DRAM" as shared memory.

The corresponding Non-OS sample code is https://github.com/OpenNuvoton/MA35D1_NonOS_BSP/tree/master/SampleCode/OpenAMP/Share_Memory_SDRAM.

6.2.3 rpmsg_rtp

This sample code is designed for Linux only. The provided sample code can be configured for rpmsg v1 or rpmsg v2, which utilizes SRAM or DRAM as shared memory.

If recursive read/write is necessary in your application, it is highly recommended to use the v2

architecture. This project involves RTP portion of rpmsg demo, please also review the sample codes in Linux applications.

This sample code can be configured to use either the MA35D1 rpmsg v1 or v2 protocol. If configured to use MA35D1 rpmsg v1, its corresponding Linux-side application is: https://github.com/OpenNuvoton/MA35D1_Linux_Applications/tree/master/examples/rpmsg.

If configured to use MA35D1 rpmsg v2, its corresponding Linux-side application is: https://github.com/OpenNuvoton/MA35D1_Linux_Applications/tree/master/examples/rpmsg-v2.

If "RPMSG_DDR_BUF" is defined, please ensure that the symbols "rpmsg-ddr-buf" and "memory-region" in device tree are also enabled for Linux driver.

If "RPMSG_V2_ARCH" is defined, please ensure that the symbol "rpmsg-v2-arch" in device tree is also enabled for Linux driver.

If you would like to modify "SHM_START_ADDRESS" and "Share_Memory_Size", please ensure that they match the definition of "rpmsg_buf" in device tree.

7 REVISION HISTORY

Date	Revision	Description
2024.04.08	1.02	• Update list of SampleCode • Add section 4.2 to introduce how to load RTP image from TF-A in Buildroot or Yocto • Rewrite Chapter 6, and change the title from 'OPENAMP' to 'Intercommunication Between A35 and RTP”
2022.05.30	1.01	• Add KPI sample
2022.02.27	1.00	• 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.