NXP Semiconductors
Application Notes

Document Number: AN13063
Rev. 0, 12/2020

MPC5777C MCAL4.3 Dual Core Project Implementation

by: NXP Semiconductors

Contents

1. Introduction

This application note introduce the dual core project implementation in MCAL4.3.

Additionally, this sample application is for MPC5777C microcontroller with AUTOSAR MCAL4.3 version RTM1.0.0.

This application note provides an introduction about startup and link file modification to implement dual core project.

This document also provides the sample code in the software package accompanying this document.

The following table shows the abbreviations used throughout the document.

Table 1. **Acronym and definition**

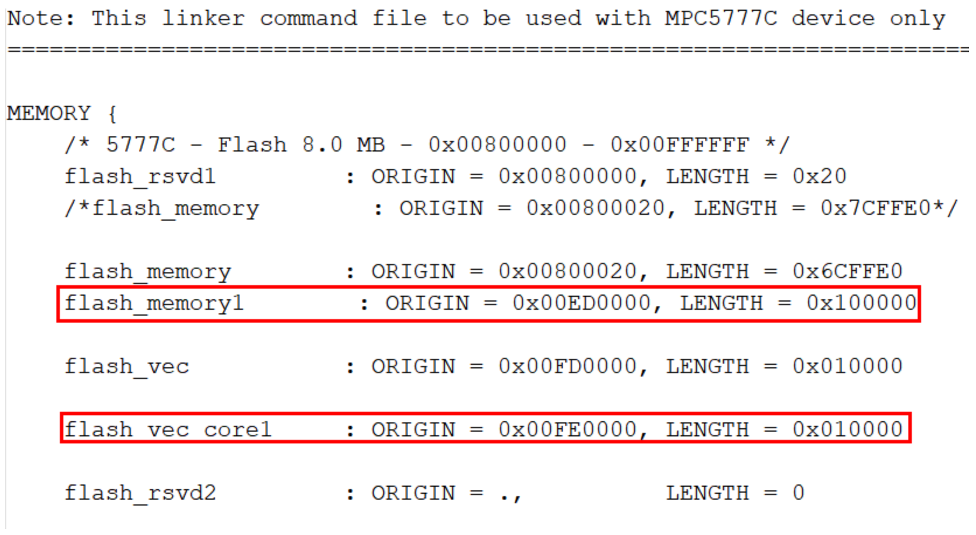
Term/Acronym	Definition
BAM	Boot Assist Module
IVPR	Interrupt vector prefix register

2. MCAL4.3 Startup and Link file

MCAL4.3 startup and link file is only for one core, so the startup and link file needs to be modified for two cores.

2.1. Dual core memory section

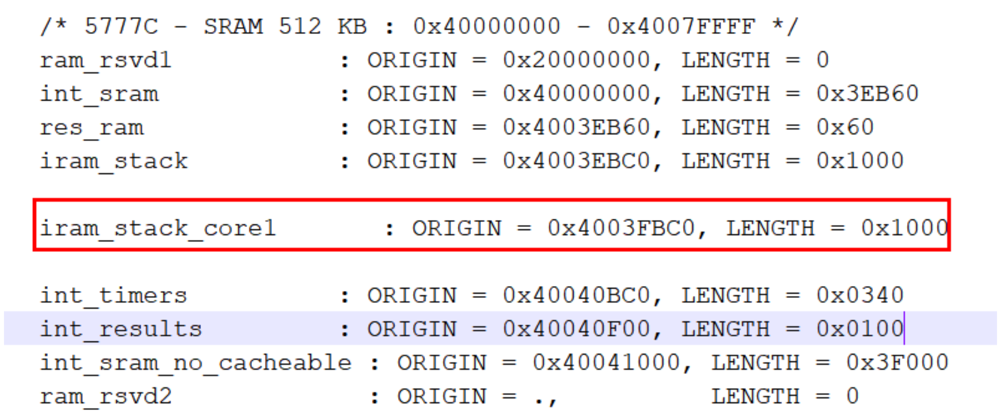
Flash_rsvd1 is used to store RCW, flash_memory is used by core 0. You should modify the link file which allocates the flash_memory 1 to store the core 1 startup code and fix the vector reset address as 0x00ED0000. You also need to configure this address in EB tresos.



```
Note: This linker command file to be used with MPC5777C device only
=====================================================================

MEMORY {
    /* 5777C - Flash 8.0 MB - 0x00800000 - 0x00FFFFFF */
    flash_rsvd1          : ORIGIN = 0x00800000, LENGTH = 0x20
    /*flash_memory         : ORIGIN = 0x00800020, LENGTH = 0x7CFFE0*/

    flash_memory         : ORIGIN = 0x00800020, LENGTH = 0x6CFFE0
    flash_memory1         : ORIGIN = 0x00ED0000, LENGTH = 0x100000

    flash_vec            : ORIGIN = 0x00FD0000, LENGTH = 0x010000

    flash_vec_core1      : ORIGIN = 0x00FE0000, LENGTH = 0x010000

    flash_rsvd2          : ORIGIN = .,          LENGTH = 0
```

Figure 1. **Memory for two core**

The following figure shows the allocated stack for core 0, the address starts from 0x4003EBC0. The core1 stack address startx from 0x4003FBC0 and the length is 4k.

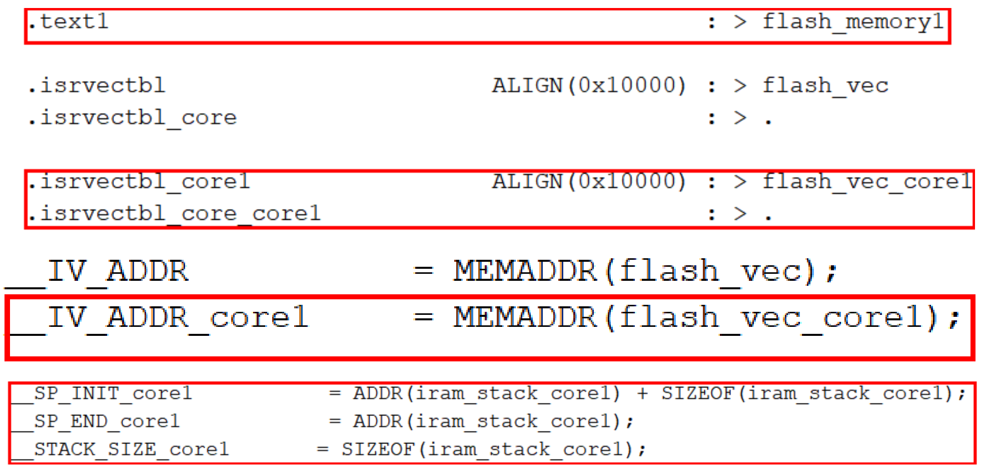


```
/* 5777C - SRAM 512 KB : 0x40000000 - 0x4007FFFF */
ram_rsvd1             : ORIGIN = 0x20000000, LENGTH = 0
int_sram              : ORIGIN = 0x40000000, LENGTH = 0x3EB60
res_ram               : ORIGIN = 0x4003EB60, LENGTH = 0x60
iram_stack            : ORIGIN = 0x4003EBC0, LENGTH = 0x1000

iram_stack_core1         : ORIGIN = 0x4003FBC0, LENGTH = 0x1000

int_timers            : ORIGIN = 0x40040BC0, LENGTH = 0x0340
int_results           : ORIGIN = 0x40040F00, LENGTH = 0x0100
int_sram_no_cacheable : ORIGIN = 0x40041000, LENGTH = 0x3F000
ram_rsvd2               : ORIGIN = .,          LENGTH = 0
```

Figure 2. **Stack for core1**

The following figure shows the allocated new section for core 1.

- .text1: section for core 1 startup code
- .isrvectbl_core1: section for core 1 interrupt vector table
- .isrvectbl_core_core1: section for core1 core vector table(IVOR)
- __IV_ADDR_core1: get the flash_vec_core1 address
- __SP_INIT_core1: get iram_stack_core1 address

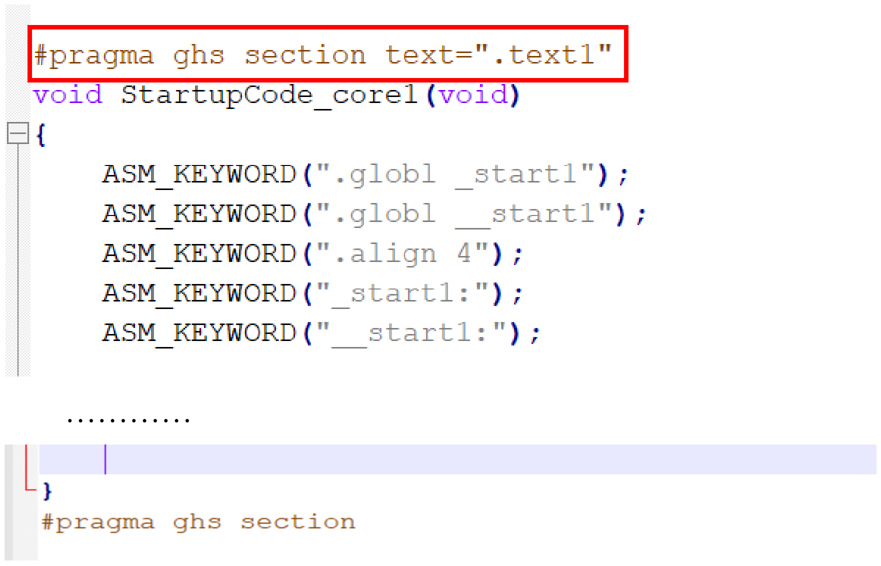


```
.text1                                          : > flash_memory1

.isrvectbl                       ALIGN(0x10000) : > flash_vec
.isrvectbl_core                                 : > .

.isrvectbl_core1                 ALIGN(0x10000) : > flash_vec_core1
.isrvectbl_core_core1                           : > .

__IV_ADDR           = MEMADDR(flash_vec);
__IV_ADDR_core1     = MEMADDR(flash_vec_core1);

__SP_INIT_core1           = ADDR(iram_stack_core1) + SIZEOF(iram_stack_core1);
__SP_END_core1            = ADDR(iram_stack_core1);
__STACK_SIZE_core1       = SIZEOF(iram_stack_core1);
```

Figure 3. **Allocated new section for core 1**

2.2. Dual core startup file

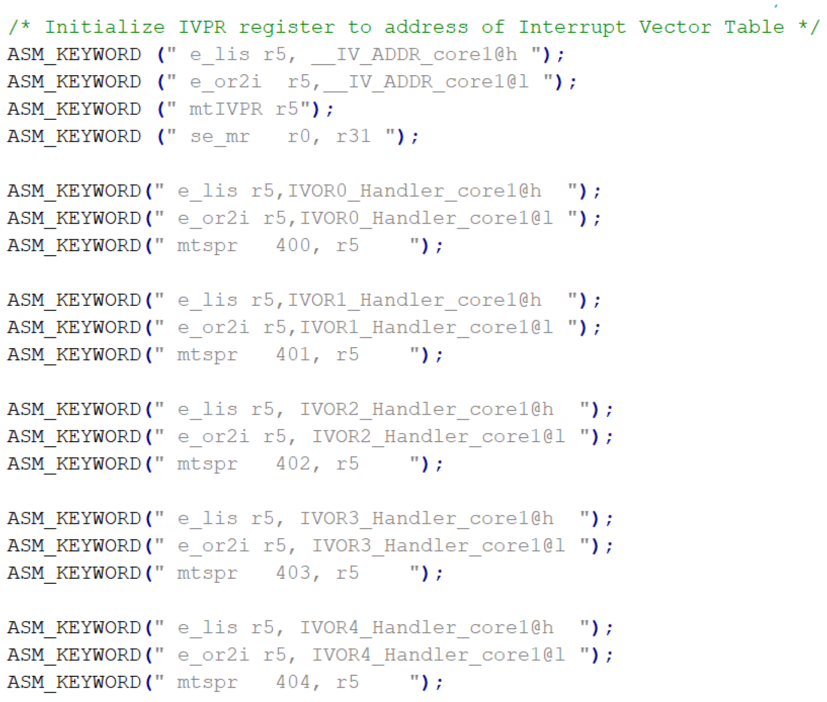
As shown in the following figure, in order to put the startup code into their own section, use the GHS compiler key words to set the core 1 startup code to .text1 section.



```
#pragma ghs section text=".text1"
void StartupCode_core1(void)
{
    ASM_KEYWORD(".globl _start1");
    ASM_KEYWORD(".globl __start1");
    ASM_KEYWORD(".align 4");
    ASM_KEYWORD("_start1:");
    ASM_KEYWORD("__start1:");

  …………

}
#pragma ghs section
```

Figure 4. **Startup code for core 1**

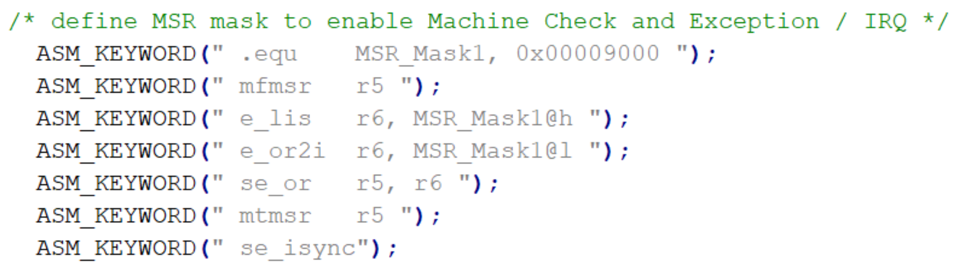
The following figure shows the IVPR register that needs to be initialized to the address of the interrupt vector table and exception vector handler. These handler addresses are provided as configuration parameters in link setting.



```
/* Initialize IVPR register to address of Interrupt Vector Table */
ASM_KEYWORD (" e_lis r5, __IV_ADDR_core1@h ");
ASM_KEYWORD (" e_or2i  r5,__IV_ADDR_core1@l ");
ASM_KEYWORD (" mtIVPR r5");
ASM_KEYWORD (" se_mr   r0, r31 ");

ASM_KEYWORD(" e_lis r5,IVOR0_Handler_core1@h  ");
ASM_KEYWORD(" e_or2i r5,IVOR0_Handler_core1@l ");
ASM_KEYWORD(" mtspr   400, r5    ");

ASM_KEYWORD(" e_lis r5,IVOR1_Handler_core1@h  ");
ASM_KEYWORD(" e_or2i r5,IVOR1_Handler_core1@l ");
ASM_KEYWORD(" mtspr   401, r5    ");

ASM_KEYWORD(" e_lis r5, IVOR2_Handler_core1@h  ");
ASM_KEYWORD(" e_or2i r5, IVOR2_Handler_core1@l ");
ASM_KEYWORD(" mtspr   402, r5    ");

ASM_KEYWORD(" e_lis r5, IVOR3_Handler_core1@h  ");
ASM_KEYWORD(" e_or2i r5, IVOR3_Handler_core1@l ");
ASM_KEYWORD(" mtspr   403, r5    ");

ASM_KEYWORD(" e_lis r5, IVOR4_Handler_core1@h  ");
ASM_KEYWORD(" e_or2i r5, IVOR4_Handler_core1@l ");
ASM_KEYWORD(" mtspr   404, r5    ");
```

Figure 5. **Interrupt vector and exception handler initialization**

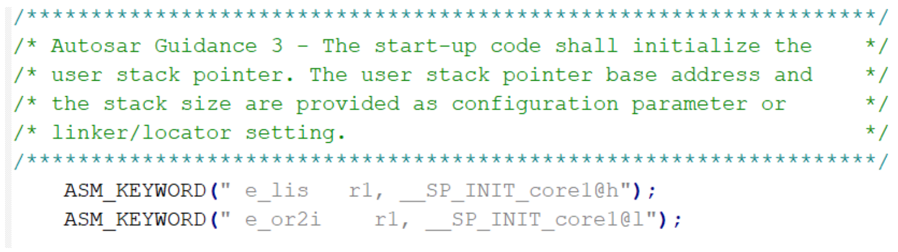
The machine check and exception for core 1 should be enabled as shown in the following figure.



```
/* define MSR mask to enable Machine Check and Exception / IRQ */
  ASM_KEYWORD(" .equ    MSR_Mask1, 0x00009000 ");
  ASM_KEYWORD(" mfmsr   r5 ");
  ASM_KEYWORD(" e_lis   r6, MSR_Mask1@h ");
  ASM_KEYWORD(" e_or2i  r6, MSR_Mask1@l ");
  ASM_KEYWORD(" se_or   r5, r6 ");
  ASM_KEYWORD(" mtmsr   r5 ");
  ASM_KEYWORD(" se_isync");
```

Figure 6. **Machine check and exception for core 1**

Start-up code should initialize user stack pointer for core 1, as shown in the following figure.



```
/******************************************************************/
/* Autosar Guidance 3 - The start-up code shall initialize the    */
/* user stack pointer. The user stack pointer base address and    */
/* the stack size are provided as configuration parameter or      */
/* linker/locator setting.                                        */
/******************************************************************/
    ASM_KEYWORD(" e_lis   r1, __SP_INIT_core1@h");
    ASM_KEYWORD(" e_or2i   r1, __SP_INIT_core1@l");
```

Figure 7. **Stack initialization for core 1**

3. Core 0 and core 1 reset vector configuration

On the MPC577C device, the BAM occupies 16 KB of memory space, 0xFFFF_C000 to 0xFFFF_FFFF. The actual code size of the BAM program is less than 4 KB and starts at 0xFFFF_F000, repeating itself after every 4 KB in the BAM address space. BAM program execution as it reset vector from address 0xFFFF_FFFC.

The BAM exits to user code at 0xFFFF_FFF8. The last instruction executed by BAM is a BLR. The link register is pre-loaded with the user application start address. The value of the start address depends on the boot mode:

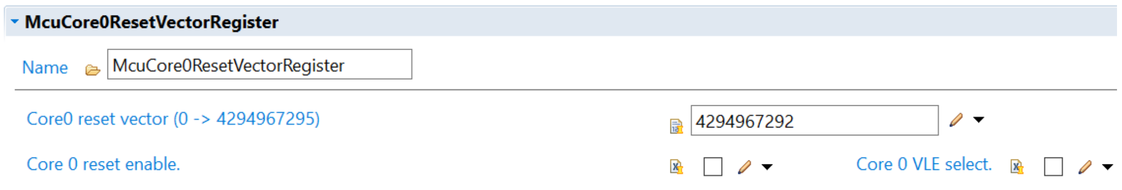
- Booting from internal or external flash memory: 32-bit word following a valid RCHW holds the start address value
- Serial boot is set according to the serial boot protocol

The following table shows the BAM address map.

Table 2. **BAM memory map**

Address	Description
0xFFFF_C000 – 0xFFFF_EFFF	BAM program mirrored
0xFFFF_F000 – 0xFFFF_FFFF	BAM program
0xFFFF_FFFC	MCU reset vector
0xFFFF_FFF8	BAM last executed instruction

In order to make the two cores have a fixed vector initial address which is consistent with the EB configuration, the core 0 reset vector address and core 1 reset vector address should be configured with 0xFFFF_FFFC(4294967292) and 0x00ED0000(15532032) in EB tresos.



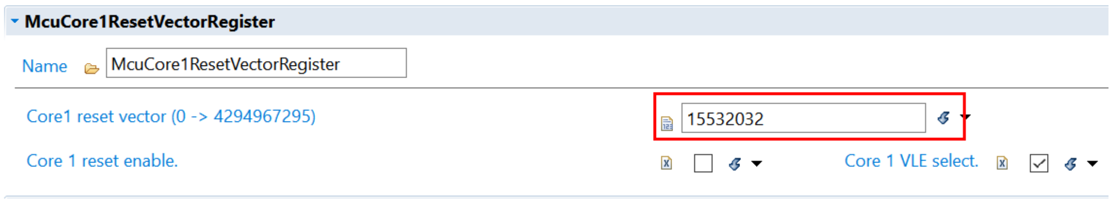
McuCore0ResetVectorRegister

Name: McuCore0ResetVectorRegister

Core0 reset vector (0 -> 4294967295): 4294967292

Core 0 reset enable. ☐ Core 0 VLE select. ☐

Figure 8. **Configure the reset vector for core0 in EB**



McuCore1ResetVectorRegister

Name: McuCore1ResetVectorRegister

Core1 reset vector (0 -> 4294967295): 15532032

Core 1 reset enable. ☐ Core 1 VLE select. ☑

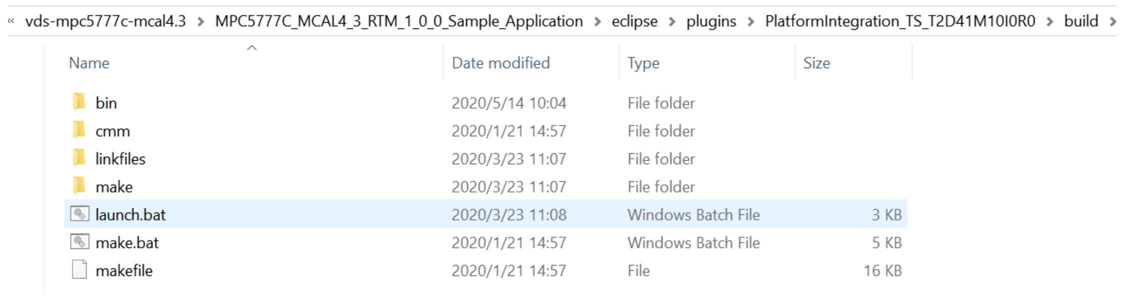
Figure 9. **Configure the reset vector for core1 in EB**

4. Compile and testing

4.1. Compiling GHS

The test hardware is the MPC5777C EVB. Compiler GHS version is compiler 2019.5.4(it is recommended to update to the latest version), EB tresos version is 25.1 and the MCAL package version is MPC5777C MCAL4.3_1.0.0.

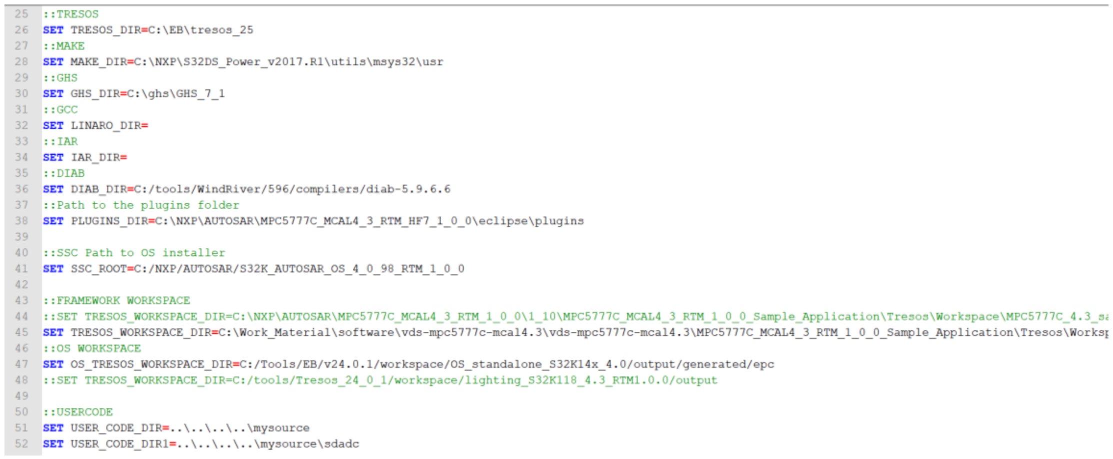
Open the build folder as shown in the following figure.



« vds-mpc5777c-mcal4.3 › MPC5777C_MCAL4_3_RTM_1_0_0_Sample_Application › eclipse › plugins › PlatformIntegration_TS_T2D41M10I0R0 › build ›

Name	Date modified	Type	Size
bin	2020/5/14 10:04	File folder	
cmm	2020/1/21 14:57	File folder	
linkfiles	2020/3/23 11:07	File folder	
make	2020/3/23 11:07	File folder	
launch.bat	2020/3/23 11:08	Windows Batch File	3 KB
make.bat	2020/1/21 14:57	Windows Batch File	5 KB
makefile	2020/1/21 14:57	File	16 KB

Figure 10. **Sample code build folder**



```
::TRESOS
SET TRESOS_DIR=C:\EB\tresos_25
::MAKE
SET MAKE_DIR=C:\NXP\S32DS_Power_v2017.R1\utils\msys32\usr
::GHS
SET GHS_DIR=C:\ghs\GHS_7_1
::GCC
SET LINARO_DIR=
::IAR
SET IAR_DIR=
::DIAB
SET DIAB_DIR=C:/tools/WindRiver/596/compilers/diab-5.9.6.6
::Path to the plugins folder
SET PLUGINS_DIR=C:\NXP\AUTOSAR\MPC5777C_MCAL4_3_RTM_HF7_1_0_0\eclipse\plugins

::SSC Path to OS installer
SET SSC_ROOT=C:/NXP/AUTOSAR/S32K_AUTOSAR_OS_4_0_98_RTM_1_0_0

::FRAMEWORK WORKSPACE
::SET TRESOS_WORKSPACE_DIR=C:\NXP\AUTOSAR\MPC5777C_MCAL4_3_RTM_1_0_0\1_10\MPC5777C_MCAL4_3_RTM_1_0_0_Sample_Application\Tresos\Workspace\MPC5777C_4.3_sa
SET TRESOS_WORKSPACE_DIR=C:\Work_Material\software\vds-mpc5777c-mcal4.3\vds-mpc5777c-mcal4.3\MPC5777C_MCAL4_3_RTM_1_0_0_Sample_Application\Tresos\Worksp
::OS WORKSPACE
SET OS_TRESOS_WORKSPACE_DIR=C:/Tools/EB/v24.0.1/workspace/OS_standalone_S32K14x_4.0/output/generated/epc
::SET TRESOS_WORKSPACE_DIR=C:/tools/Tresos_24_0_1/workspace/lighting_S32K118_4.3_RTM1.0.0/output

::USERCODE
SET USER_CODE_DIR=..\..\..\..\mysource
SET USER_CODE_DIR1=..\..\..\..\mysource\sdadc
```

Figure 11. **Launch.bat file**

TRESOS_DIR: EB tresos installation path

MAKE_DIR : make.exe file path

GHS_DIR: GHS compiler installation path

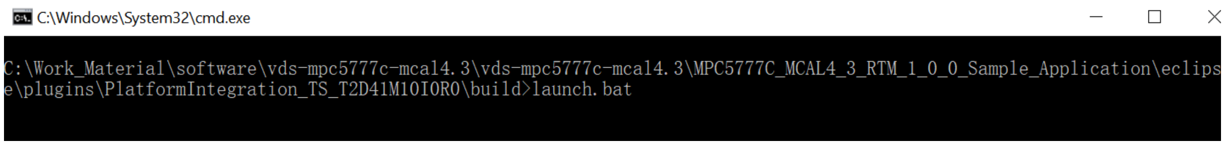
PLUGINS_DIR : The MCAL4.3 plugins installation path

TRESOS_WORKSPACE_DIR: EB project generated code path, which is

C:\vds-mpc5777c-mcal4.3\ MPC5777C_MCAL4_3_RTM_1_0_0_Sample_Application\ Tresos\

Workspace\MPC5777C_4.3_sample_applications\output

To build the sample, execute the following command to run launch.bat: launch.bat



```
C:\Work_Material\software\vds-mpc5777c-mcal4.3\vds-mpc5777c-mcal4.3\MPC5777C_MCAL4_3_RTM_1_0_0_Sample_Application\eclips
e\plugins\PlatformIntegration_TS_T2D41M10I0R0\build>launch.bat
```

Figure 12. **Build command**

The object files and linker output file (PlatformIntegration.elf) will be generated in the /bin subdirectory.

Open the cmm folder and run the smp_demo.cmm script using **LAUTERBACH** debugger and debug the sample application code.

4.2. Test result

This sample application was tested on the MPC5777C EVB as showing in the following figure. Core 0 is running the MCAN transmission and core 1 is running the CAN transmission.

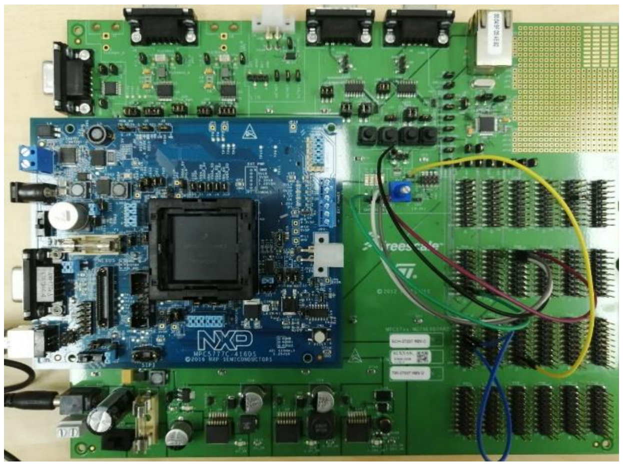
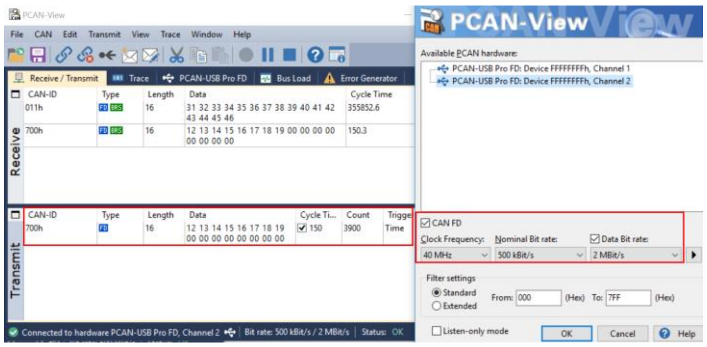


Figure 13. **MPC5777C EVB board**



Receive

CAN-ID	Type	Length	Data	Cycle Time
011h	FD BRS	16	31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46	355852.6
700h	FD BRS	16	12 13 14 15 16 17 18 19 00 00 00 00 00 00 00 00	150.3

Transmit

CAN-ID	Type	Length	Data	Cycle Ti...	Count	Trigge
700h	FD	16	12 13 14 15 16 17 18 19 00 00 00 00 00 00 00 00	✓ 150	3900	Time

Available PCAN hardware:
- PCAN-USB Pro FD: Device FFFFFFFFh, Channel 1
- PCAN-USB Pro FD: Device FFFFFFFFh, Channel 2

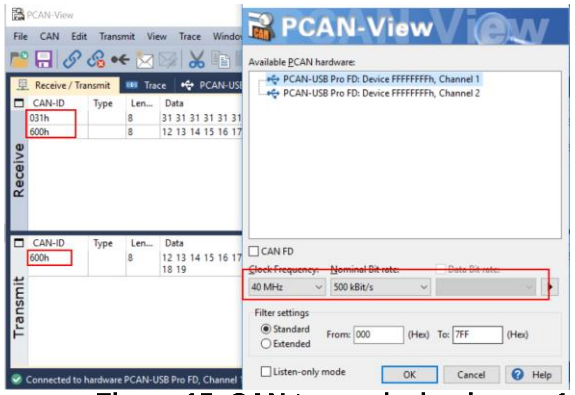
☑ CAN FD
Clock Frequency: 40 MHz Nominal Bit rate: 500 kBit/s ☑ Data Bit rate: 2 MBit/s

Filter settings: ● Standard ○ Extended From: 000 (Hex) To: 7FF (Hex)

☐ Listen-only mode

Connected to hardware PCAN-USB Pro FD, Channel 2 | Bit rate: 500 kBit/s / 2 MBit/s | Status: OK

Figure 14. **MCAN sample transmission in core0**



Receive

CAN-ID	Type	Len...	Data
031h		8	31 31 31 31 31 31
600h		8	12 13 14 15 16 17

Transmit

CAN-ID	Type	Len...	Data
600h		8	12 13 14 15 16 17 18 19

Available PCAN hardware:
- PCAN-USB Pro FD: Device FFFFFFFFh, Channel 1
- PCAN-USB Pro FD: Device FFFFFFFFh, Channel 2

☐ CAN FD
Clock Frequency: 40 MHz Nominal Bit rate: 500 kBit/s ☐ Data Bit rate:

Filter settings: ● Standard ○ Extended From: 000 (Hex) To: 7FF (Hex)

☐ Listen-only mode

Connected to hardware PCAN-USB Pro FD, Channel 1

Figure 15. CAN transmission in core1

5. References

- MPC5777C Reference Manual
- MPC5777C EVB User Guide

Document Number: AN13063
Rev. 0
12/2020