AN12824

RT600 Out of the Box (OOB) Demo

Rev. 0 — 20-04-2020 Application Note

1 Introduction

The i.MX RT600 is a cross-over processor which combines a high-performance Cadence® Tensilica® Hi-Fi4 audio Digital Signal Processor (DSP) with a next generation Arm® Cortex®-M33 (CM33).

The board comes preprogrammed with a “blinky” demo (LED D9 blinking). The demo exercises the DSP HiFi4 and CM33 communication executing various math functions and making a simple performance comparator showing the number of cycles for both cores.

This application note explains how to program and run the second part of the out of the box demo, that is how to execute the math functions and code to run each core (DSP HiFi4 and CM33).

Contents

1 Introduction..... 1
2 Prepare demo..... 1
3 Project overview..... 3
4 DSP HiFi4 application..... 4
4.1 Code..... 4
5 Cortex®-M33 application..... 8
5.1 Code..... 8
6 Running DSP HiFi4 and CM33 applications in IMXRT685 EVK..... 13

2 Prepare demo

As a prerequisite of running the OOB demo, you need to ensure that you have all the necessary tools and configurations installed as mentioned in the MIMXRT685-EVK Get Started Guide (in Section 2. Get Software).

Once you have all tools installed, follow the steps below:

1. Download “RT600 Out of the Box (OOB) Demo” available in Application Note Software.
2. Navigate to the following path: *<RT600 SDK path>/boards/evkmimxrt685/dsp_examples*, and unzip the project inside the *dsp_examples* folder.

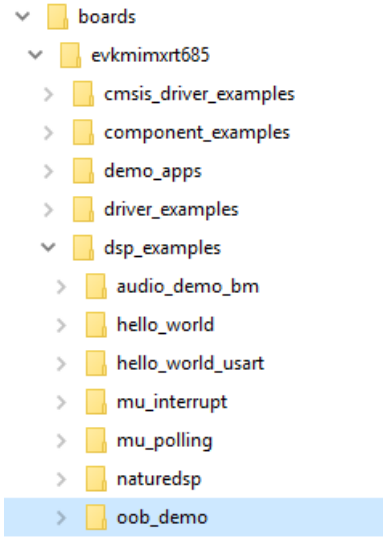

Figure 1. Path to OOB demo project

Inside the *oob_demo* folder, you find folders for cm33 and dsp.

3. Import each project using MCUXpresso IDE for CM33 and Xtensa Xplorer IDE for DSP HiFi4.

MCUXpresso IDE

a. Open MCUXpresso IDE and choose your preferred workspace.

b. Click **File > Import > Existing Projects into Workspace**.

c. Browse for the project you added in step 2, and select the *cm33* folder.

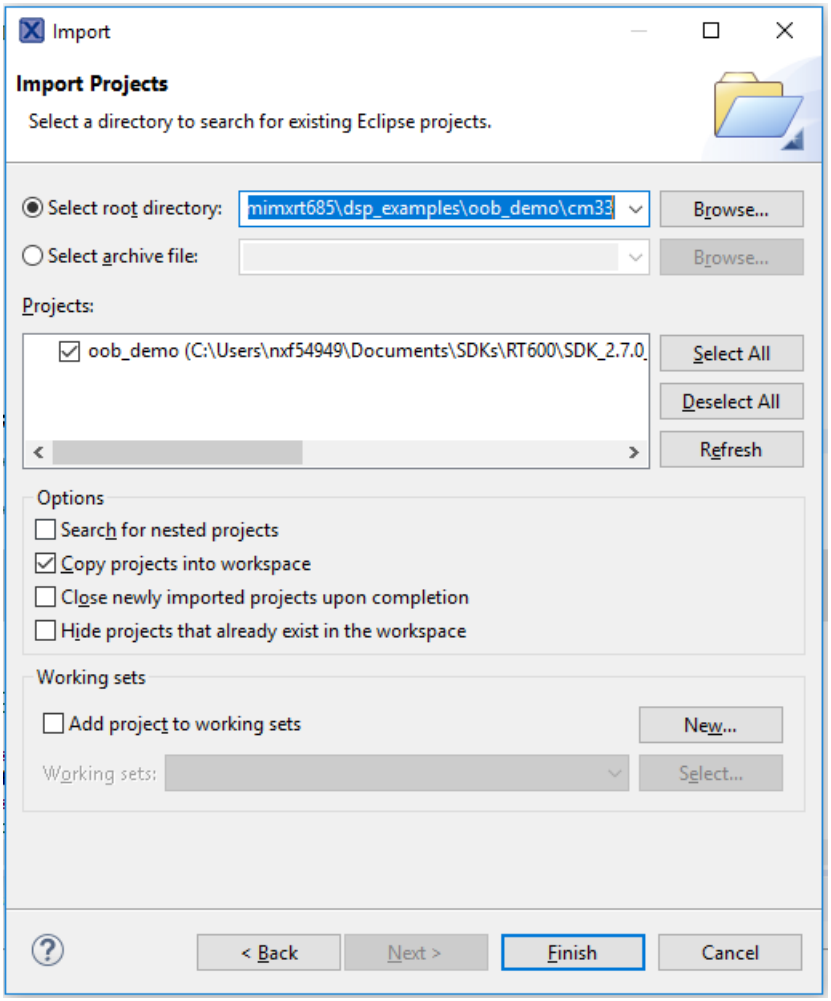

Figure 2. Importing CM33 project

d. Select the **Copy projects into workspace** checkbox.

e. Click **Finish**.

Xtensa Xplorer IDE

a. Open Xtensa Xplorer IDE and choose your preferred workspace.

b. Click **File > Import > Existing Projects into Workspace**.

c. Browse for the project you added in step 2, and select the *dsp* folder.

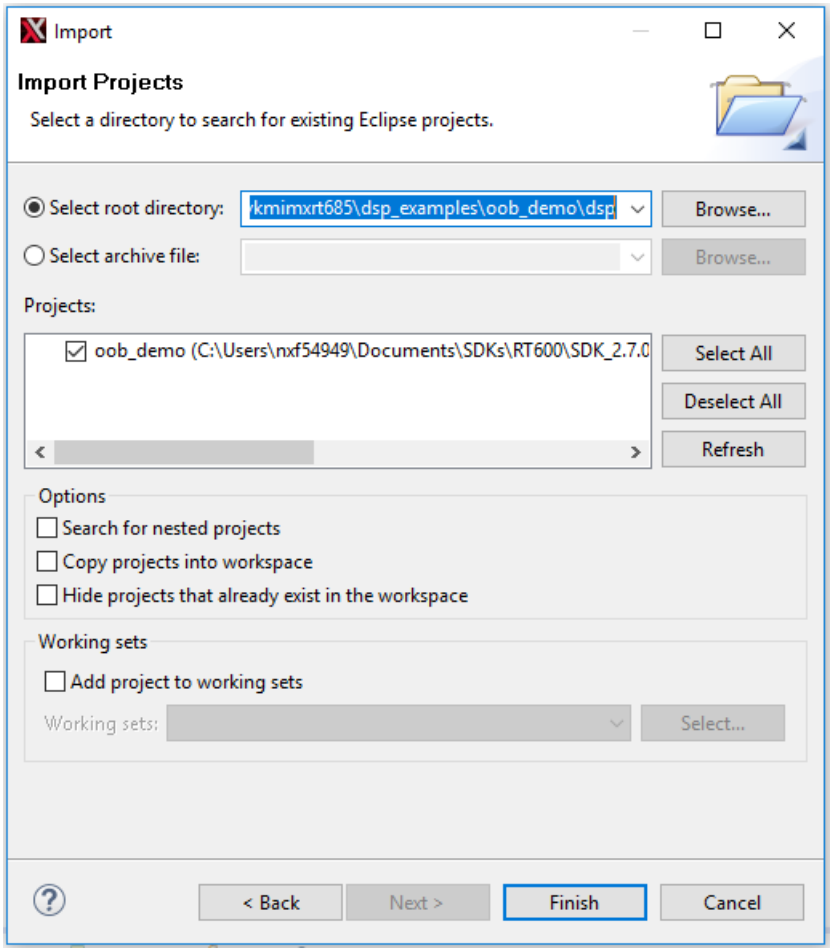

Figure 3. Importing DSP HiFi4 project

d. Do not select any of the checkbox options.

e. Click **Finish**.

3 Project overview

The board comes pre-programmed with this demo; if you have not flashed MIMXRT6xx with a different application, follow the steps below. If you already flashed your board with a different application, see Section 6.

1. Connect the MIMXRT685-EVK board on the J5 “Link USB” connector to your computer using a micro USB, and notice the red LED D9 blinking.
2. Open a serial terminal, identify the COM port, and configure it with:
   - 115200 Baud rate
   - 8-bit data
   - No parity
   - 1 stop bit
   - No flow control
3. Press the reset button, and notice the following message on the screen, as shown below.

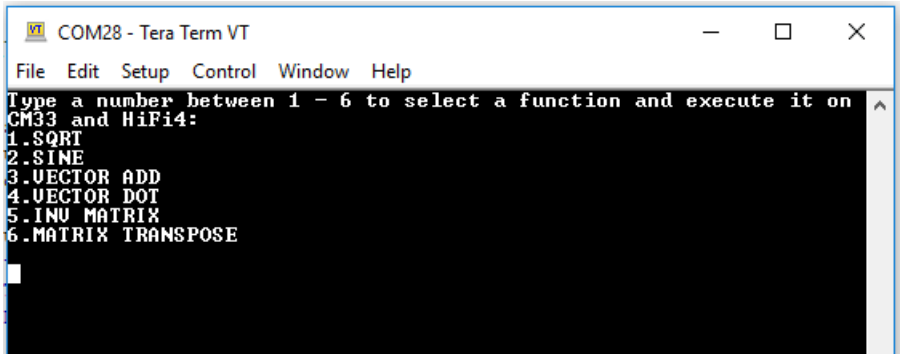

Figure 4. Serial terminal menu

You can select a math function by typing the function number to see its execution. The cycle count result of each core displays on the terminal.

Both cores, CM33 and DSP HiFi4, have the option to execute the following functions:

Table 1. Math functions

Math Function	Description
Square Root	Gets the square root of a decimal number. In this demo, the input value is 0.25.
Sine	Gets the sine of a decimal number. In this demo, the input value is 0.5.
Vector Add	Makes an addition of two integer vectors with length of 200 each.
Vector Dot Product	Executes the vector dot product of two float vectors with length of 16 each.
Inverse Matrix	Executes the inverse of a 2x2 float matrix.
Matrix Transpose	Executes the transpose operation of an 8x8 float matrix.

The demo uses the message unit to coordinate the execution and to communicate to the DSP HiFi4 core which math function has to run.

For further information about the message unit, refer to the *RT600 Dual-Core Communication and Debugging (AN12789)* application note and *Chapter 31: RT6xx Message Unit* in the *RTxx User Manual (UM11147)*.

4 DSP HiFi4 application

This section lists the code and libraries needed to run the DSP HiFi4 application. Open the project using Xtensa Xplorer IDE.

4.1 Code

The DSP HiFi4 application provides the NatureDSP library, which contains various math API's that you can use in the demo. The source files can be found in the following path: *<SDK path>/middleware/dsp/naturedsp_hifi4*. You can also find documentation about the library inside the *doc* folder.

Following files from NatureDSP library are needed for this demo:

- mtx_inv2x2f_hifi4.c
- mtx_transpose32x32_fast_hifi4.c
- mtx_transpose_fast_hifi4.c

- scl_sine_32x32_hifi4.c
- scl_sine_table32.c
- scl_sqrt_32x32_hifi4.c
- scl_sqrt_table32.c
- vec_add32x32_fast_hifi4.c
- vec_dotf_hifi4.c
- vec_recip_table.c
- vec_recip_table.h
- NatureDSP_Signal_complex.h
- NatureDSP_Signal_math.h
- NatureDSP_Signal_matinv.h
- NatureDSP_Signal_matop.h
- NatureDSP_Signal_vector.h
- NatureDSP_Signal.h
- NatureDSP_types.h
- scl_sine_table32.h
- scl_sqrt_table32.h
- sine_table.h
- sqrt_table.h

The main files for this application are *main_dsp.c* and *srtm_naturedsp_test.c*. Both files are located in the project's *source* folder. Each file has the required functions to initialize and execute the DSP HiFi4 application.

main_dsp.c

This file initializes the required resources and executes the math function. The application begins in the main() function. It first initializes the debug console and UART and then the message unit. Then, the DSP HiFi4 application uses the MU_SetFlags() function to send the boot flag and indicate that the DSP has started up.

main_dsp.c

```
int main(void)
{
    usart_config_t config;

    /* Init board hardware. */
    BOARD_InitDebugConsole();

    /* Enable Rx and Tx on UART */
    USART_GetDefaultConfig(&config);
    config.baudRate_Bps = BOARD_DEBUG_UART_BAUDRATE;
    config.enableTx     = true;
    config.enableRx     = true;

    USART_Init(DEMO_USART, &config, DEMO_USART_CLK_FREQ);

    /* MUB init */
    MU_Init(APP_MU);

    /* Send flag to Core 0 to indicate Core 1 has startup */
    MU_SetFlags(APP_MU, BOOT_FLAG);

    while (1)
    {
        /* Wait until a message from CM33 is received */
        g_msgRecv = MU_ReceiveMsg(APP_MU, CHN_MU_REG_NUM);
        /* Execute math function */
        switch(g_msgRecv)
        {
```

Figure 5. Uart and MU initialization

The DSP HiFi4 application now waits for the CM33 messages using MU_ReceiveMsg(). Depending on the received message from CM33, the application calls the math function.

```
    /* Send flag to Core 0 to indicate Core 1 has startup */
    MU_SetFlags(APP_MU, BOOT_FLAG);

    while (1)
    {
        /* Wait until a message from CM33 is received */
        g_msgRecv = MU_ReceiveMsg(APP_MU, CHN_MU_REG_NUM);
        /* Execute math function */
        switch(g_msgRecv)
        {
            case 1:
                /* Execute square root */
                TEST_SQRT();
                break;
            case 2:
                /* Execute sine */
                TEST_SINE();
                break;
            case 3:
                /* Execute vector add */
                TEST_VEC_ADD();
                break;
            case 4:
                /* Execute vector dot product */
                TEST_VEC_DOT();
                break;
            case 5:
                /* Execute inverse matrix */
                TEST_MATRIX_INV();
                break;
            case 6:
                /* Execute matrix transpose */
                TEST_MATRIX_TRANSPOSE();
                break;
            default:
                break;
        }
    }
}
```

Figure 6. Receive message from CM33 indicating which function to execute

srtm_naturedsp_test.c

In this file, you find the declaration of six math functions used in the application.

- TEST_SQRT()
- TEST_SINE()
- TEST_VEC_ADD()
- TEST_VEC_DOT()
- TEST_MATRIX_INV()
- TEST_MATRIX_TRANSPOSE()

Each function initializes the required variables, executes the math function from NatureDSP library, and verifies their results. The math function is executed LOOP_COUNT times to obtain an average on the cycle count result. By default, this macro has a value of 100. All functions have a similar structure as shown in the following figure.

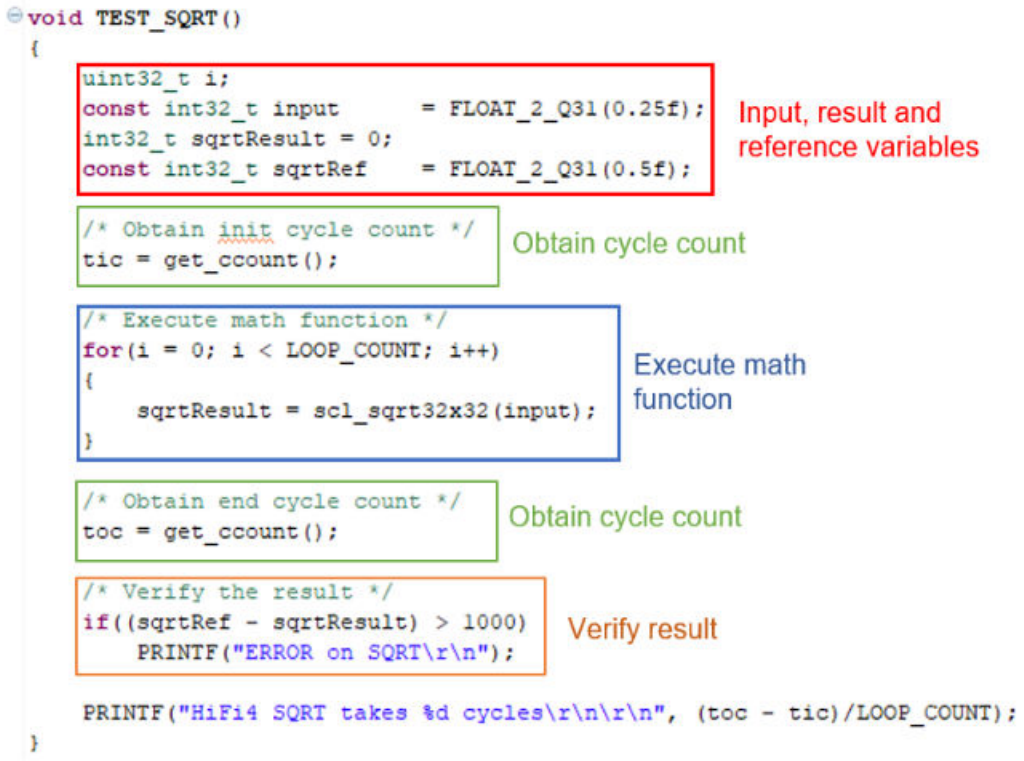

Figure 7. HiFi4 function structure

5 Cortex®-M33 application

This section lists the code and libraries needed to run the Cortex®-M33 (CM33) application. Open the project using MCUXpresso.

5.1 Code

The CM33 application uses the following files to execute the math functions for the demo. You can find the source files at the following location: *<SDK path>/CMSIS/DSP/Source.*

- arm_bitreversal2.c
- arm_common_tables.c
- arm_const_structs.c
- arm_dot_prodf32.c
- arm_sqrt_q31.c
- arm_mat_add_q31.c
- arm_mat_inverse_f32.c
- arm_mat_trans_f32.c
- arm_sin_q31.c

main_cm.c

This file contains all the math functions and all needed initializations such as clocks, debug console, and the message unit.

1. The program starts with the BOARD_InitPins() function. The following pins are initialized:

Table 2. Pin configuration

Pin	Configured as
P0_31	Red LED
P0_14	Green LED
P0_26	Blue LED
fc15_i2c_scl	Used for PMIC configuration
fc15_i2c_sda	Used for PMIC configuration
P0_1	Used for Tx Uart
P0_2	Used for Rx Uart

2. Initialize the Systick using TEST_InitTime(). The Systick is configured to trigger every 500 ns to follow a better approach in obtaining the cycle count for CM33.

main_cm.c

```
109 int main(void)
110 {
111     usart_config_t config;
112
113     /* Init board hardware.*/
114     BOARD_InitPins();
115     BOARD_BootClockRUN();
116     BOARD_InitDebugConsole();
117
118     /* Initialize LED */
119     LED_INIT();
120
121     /* Initialize SysTick */
122     TEST_InitTime();
123
124     /* Clear MUA reset */
125     RESET_PeripheralReset(kMU_RST_SHIFT_RSTn);
126
127     /* MUA init */
128     MU_Init(APP_MU);
129
130     /* Copy DSP image to RAM and start DSP core. */
131     BOARD_DSP_Init();
```

Figure 8. Board and Systick initialization

3. Initialize the message unit. Remember that the message unit builds communication between CM33 and DSP HiFi4 cores and inform the math function to execute.
4. Initialize the DSP HiFi4 core using the BOARD_DSP_Init() function.

```
/* Copy DSP image to RAM and start DSP core. */
BOARD_DSP_Init();
```

Figure 9. DSP HiFi4 initialization

The BOARD_DSP_Init() function initializes the PMIC and the clocks for the DSP HiFi4.

```
67  /* Initialize PMIC PCA9420 */
68  BOARD_InitPmic();
69  /* Configure PMIC Vddcore value according to main/dsp clock. */
70  BOARD_SetPmicVoltageForFreq(CLOCK_GetMainClkFreq(), CLK_600MHZ);
71
72  /* Enable DSP PLL clock 594MHz. */
73  CLOCK_InitSysPfd(kCLOCK_Pfd1, 16);
74  /*Let DSP run on DSP PLL clock with divider 1 (594Mhz). */
75  CLOCK_AttachClk(kDSP_PLL_to_DSP_MAIN_CLK);
76  CLOCK_SetClkDiv(kCLOCK_DivDspCpuClk, 1);
77  CLOCK_SetClkDiv(kCLOCK_DivDspRamClk, 2);
78
79  /* Initializing DSP core */
80  DSP_Init();
```

Figure 10. Initialization of PMIC and clocks

5. The CM33 application starts the DSP HiFi4 operation by setting the SYSCTL0_DSPSTALL register inside the DSP_Start() function.

```
/* Run DSP core */
DSP_Start();
```

Figure 11. Start DSP HiFi4

For further information about DSP HiFi4 initialization and configuration, see Chapter 5.1 in the *Getting Started with Xplorer for EVK-MIMXRT685* document. You can find this document at *<SDK path>/docs.*

6. Once the DSP HiFi4 configures and starts running, the CM33 waits for the HiFi4 boot flag. This ensures that the DSP is up and ready to start the application.

```
/* MUA init */
MU_Init(APP_MU);

/* Copy DSP image to RAM and start DSP core. */
BOARD_DSP_Init();

/* Wait DSP core is Boot Up */
while (BOOT_FLAG != MU_GetFlags(APP_MU));

/* Enable Rx and Tx on UART */
USART_GetDefaultConfig(&config);
config.baudRate_Bps = BOARD_DEBUG_UART_BAUDRATE;
config.enableTx     = true;
config.enableRx     = true;
```

Waits for the DSP boot flag.

Figure 12. Wait for DSP boot flag

7. Enable the UART interrupts to be able to receive the information typed on the serial terminal.

```
/* Copy DSP image to RAM and start DSP core. */
BOARD_DSP_Init();

/* Wait DSP core is Boot Up */
while (BOOT_FLAG != MU_GetFlags(APP_MU));

/* Enable Rx and Tx on UART */
USART_GetDefaultConfig(&config);
config.baudRate_Bps = BOARD_DEBUG_UART_BAUDRATE;
config.enableTx     = true;
config.enableRx     = true;

USART_Init(DEMO_USART, &config, DEMO_USART_CLK_FREQ);
/* Enable RX interrupt. */
USART_EnableInterrupts(DEMO_USART, kUSART_RxLevelInterruptEnable | kUSART_RxErrorInterruptEnable);
EnableIRQ(DEMO_USART_IRQn);
```

Figure 13. UART interrupt

8. Now, both cores are ready to start the application. At this time, the CM33 application waits for an input on the serial terminal and toggles the red LED. The DSP HiFi4 application waits for a message from CM33. When a number is typed on the terminal, the UART interrupt triggers.

```
void DEMO_USART_IRQHandler(void)
{
        /* If new data arrived. */
    if ((kUSART_RxFifoNotEmptyFlag | kUSART_RxError) & USART_GetStatusFlags(DEMO_USART))
    {
        dataTyped = USART_ReadByte(DEMO_USART);
        uartTyped = true;
    }
/* Add for ARM errata 838869, affects Cortex-M4, Cortex-M4F Store immediate overlapping
  exception return operation might vector to incorrect interrupt */
#if defined __CORTEX_M && (__CORTEX_M == 4U)
    __DSB();
#endif
}
```

Figure 14. UART interrupt handler

9. If the data typed corresponds to a valid number (1 to 6), then the CM33 application executes the math function, sends the message to HiFi4 indicating which function has to run, and cleans the dataTyped and uartTyped variables.

```
switch(dataTyped)
{
  case '1':
    /* Execute square root */
    arm_mat_sqrt_Test();                    --> Execute math function
    /* Communicate with HiFi4 to execute math function */
    MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 1);  --> Send message to HiFi4
    dataTyped = 0;                          --> Clear input data variable
    break;
  case '2':
    /* Execute sine */
    arm_mat_sine_Test();
    /* Communicate with HiFi4 to execute math function */
    MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 2);
    dataTyped = 0;
    break;
  case '3':
    /* Execute vector add */
    arm_mat_vec_add_Test();
    /* Communicate with HiFi4 to execute math function */
    MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 3);
    dataTyped = 0;
    break;
  case '4':
    /* Execute vector dot product */
    arm_mat_vec_dot_Test();
    /* Communicate with HiFi4 to execute math function */
    MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 4);
    dataTyped = 0;
    break;
  case '5':
    /* Execute inverse matrix */
    arm_mat_mtx_inv_Test();
    /* Communicate with HiFi4 to execute math function */
    MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 5);
    dataTyped = 0;
    break;
  case '6':
    /* Execute matrix transpose */
    arm_mat_mtx_tnsp_Test();
    /* Communicate with HiFi4 to execute math function */
    MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 6);
    dataTyped = 0;
    break;

  default:
    break;
  }
 uartTyped = false;                         --> Clear uart flag
}
delay();
/* Toggle led */
LED_RED_TOGGLE();                           --> Toggling LED
}
```

Figure 15. Main loop

Below the main function, you can find definition of the 6 math functions:

- arm_mat_sqrt_Test()
- arm_mat_sine_Test()
- arm_mat_vec_add_Test()
- arm_mat_vec_dot_Test()
- arm_mat_mtx_inv_Test()
- arm_mat_mtx_tnsp_Test()

10. The math function is executed LOOP_COUNT times to obtain an average on the cycle count result. By default, this macro has a value of 100. All functions used by the CM33 application have a similar structure as shown in the following figure.

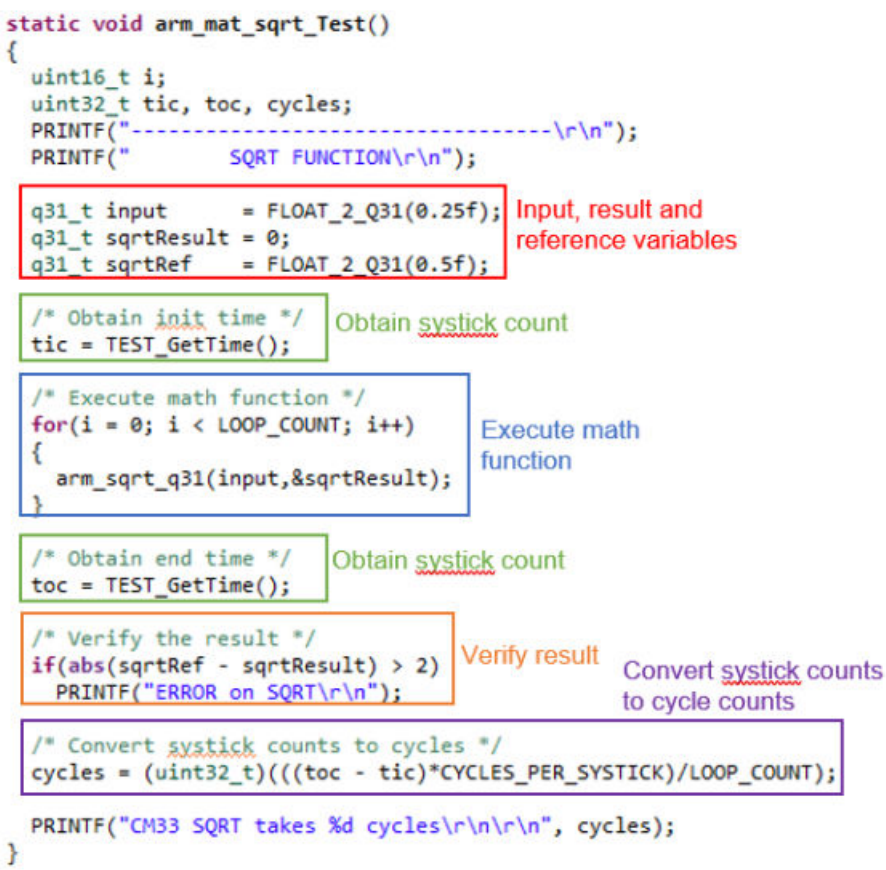

Figure 16. Math function structure

6 Running DSP HiFi4 and CM33 applications in IMXRT685 EVK

Follow the steps below to run code in the IMXRT685 EVK board.

1. Select the project on MCUXpresso and build it.

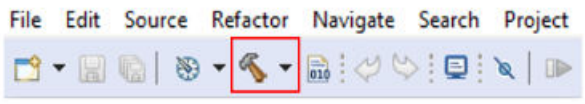

Figure 17. Build project

2. Connect the MIMXRT685-EVK board to your computer using a mirco USB to J5 port.
3. Download the CM33 application to the MIMXRT685-EVK board.

Figure 18. Download application

4. Run the application in MCUXpresso.

Figure 19. Run application

5. Open Xtensa Xplorer IDE, and configure the following options as shown in the following figure.

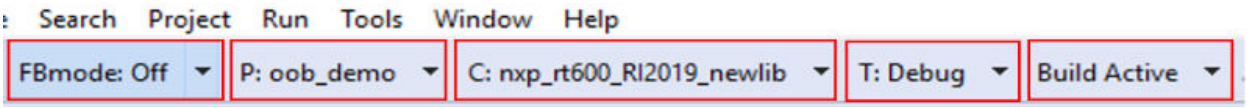

Figure 20. Xtensa Xplorer IDE

6. Build the project by clicking **Build Active**.

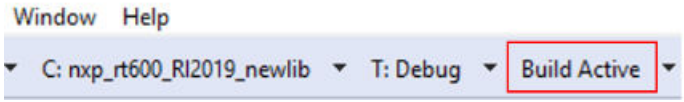

Figure 21. Select Build Active

7. Open a serial terminal on your PC and configure it with the following settings:
   - 115200 Baud rate,
   - 8-bit data,
   - No parity,
   - 1 stop bit,
   - No flow control
8. Open the command prompt, navigate to the following path, *C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.01*, and execute the following command: *xt-ocd.exe -c topology.xml*. You should see the following:

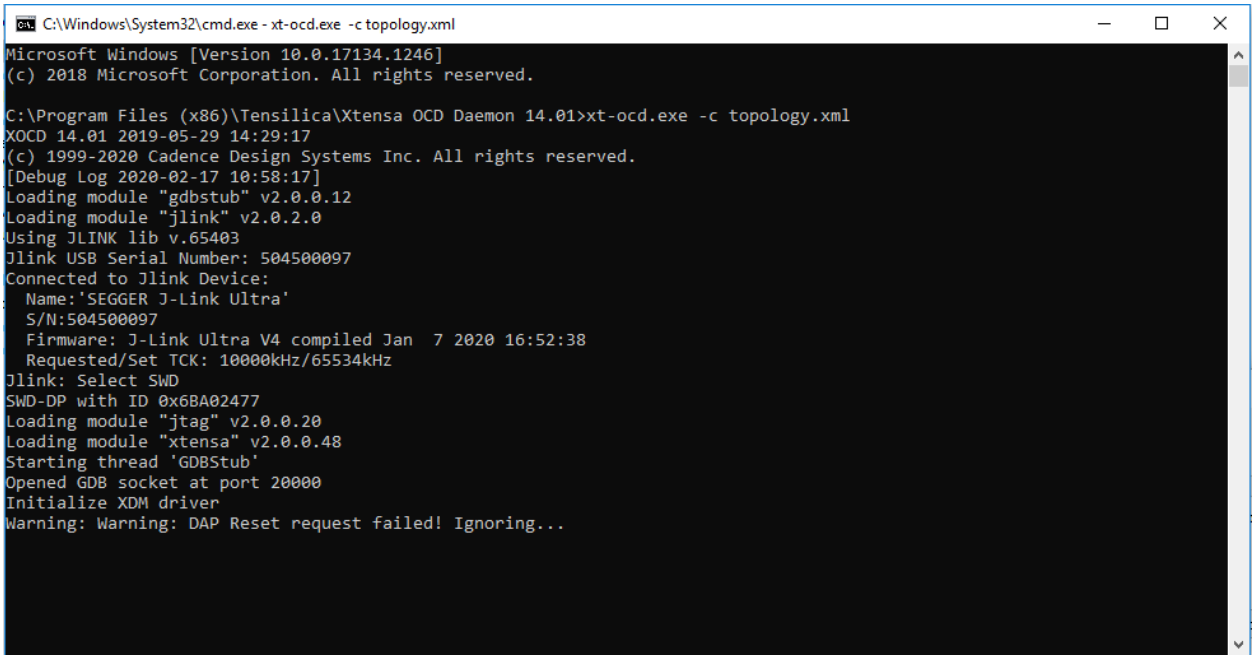

Figure 22. Command prompt - execute xt-ocd.exe

9. Return to Xtensa Xplorer, and click **Debug > Debug Configurations**.

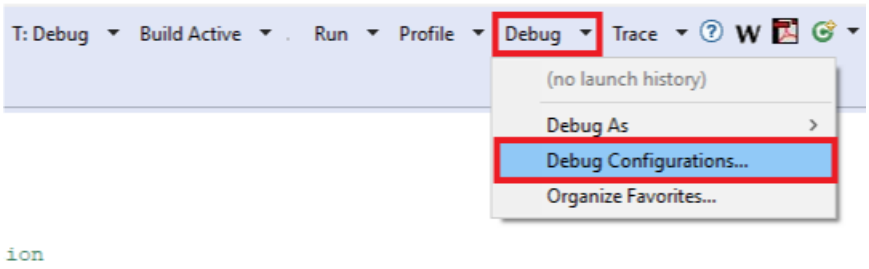

Figure 23. Debug configuration in Xtensa Xplorer

10. Select **oob_demo Xtensa On Chip Debug**, and click **Debug**.

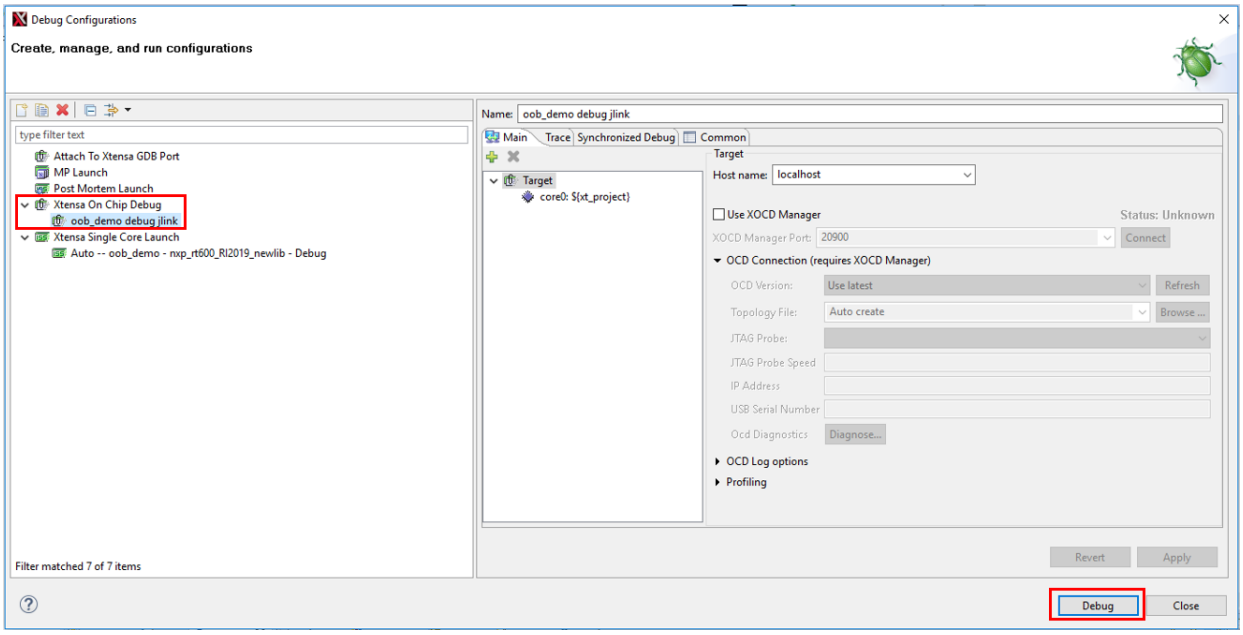

Figure 24. Debugging application

11. Click **Yes** in the window that appears prompting to download the application to the core 0.

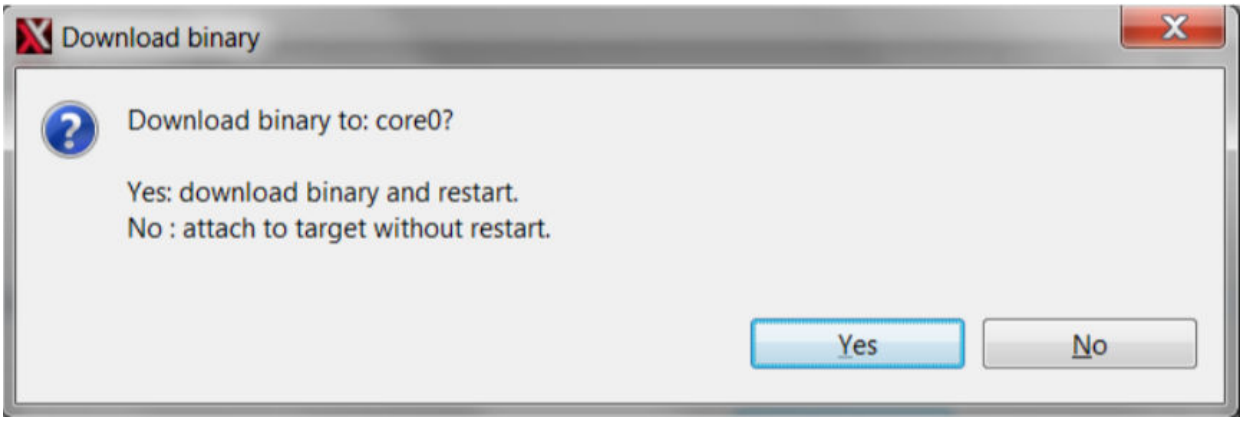

Figure 25. Download binary message box

12. Run the program on Xtensa Xplorer IDE.

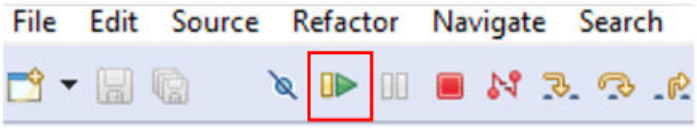

Figure 26. Run program on Xtensa Xplorer IDE

You should see the red LED blinking and the below information on the terminal.

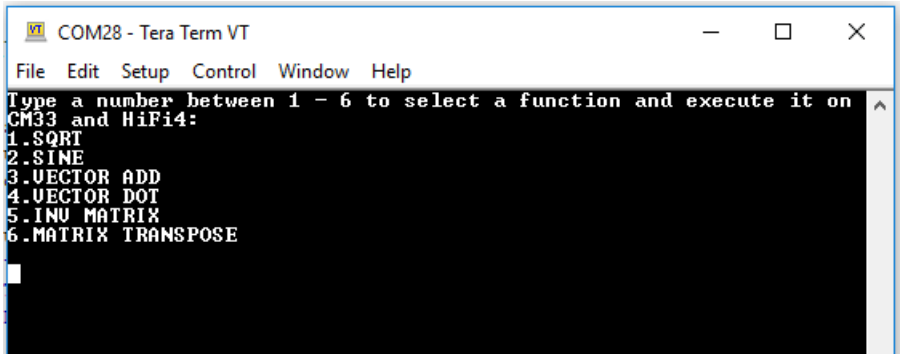

Figure 27. Terminal window

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 20-04-2020

Document identifier: AN12824