AN13151

RT500 Out of the Box (OOB) Demo

Rev. 0 — 19 February 2021

Application Note

Contents

1 Introduction

The i.MX RT500 is a cross-over processor that combines a high-performance Cadence® Tensilica® Fusion F1 DSP with a next-generation Arm® Cortex® -M33 along with a 2D Vector GPU with LCD Interface and MIPI DSI PHY.

The board comes pre-programmed with a “blinky” demo (RGB LED D19 blinking). The demo also uses the DSP, GPU, and Cortex®-M33, executing various math functions, making a simple performance comparator for both cores, and showing the GPU's graphic performance using drawing on a display screen.

This application note explains how to program and run the Out of the Box (OOB) demo and steps to run it on each core.

2 Prepare the demo

To run the OOB demo, ensure that the necessary tools and configurations are installed. For details, see the MIMXRT595-EVK Getting Started Guide (in Section 2. Get Software).

The projects for Cortex®-M33 and the Fusion F1 DSP are available in the software folder. Import each demo using MCUXpresso IDE for CM33 and Xtensa Xplorer IDE for Fusion F1.

Download and save the entire project in the SDK dsp_examples directory at the following location.`<SDK path>/boards/evkmimxrt595/dsp_examples/`.

2.1 Import project in MCUXpresso IDE

1. Open MCUXpresso IDE.
2. In the Workspace, select **File > Import > Existing Projects**.
3. Navigate to the CM33 project folder.
4. Check the **Copy projects into workspace** checkbox.
5. Click the **Finish** button.

2.2 Import project in Xtensa Xplorer IDE

1. Open Xtensa Xplorer IDE.
2. In the Workspace, select **File > Import > General > Existing Projects into Workspace**.
3. Navigate to the DSP project folder.
4. In the **Options** section, do not select any of the listed options.
5. Click the **Finish** button.

3 Running the application in the IMXRT595 EVK

1. Open a serial terminal on your computer and configure it with the following settings.

- 115200 Baud rate
- 8-bit data
- No parity
- One-stop bit
- No flow control

2. Open MCUXpresso IDE and open the `oob_demo` project.
3. To use another supported display, change the macro ‘**DEMO_PANEL**’ in the ‘display_support.h’ file. The default is rectangular RK055AHD091 display.

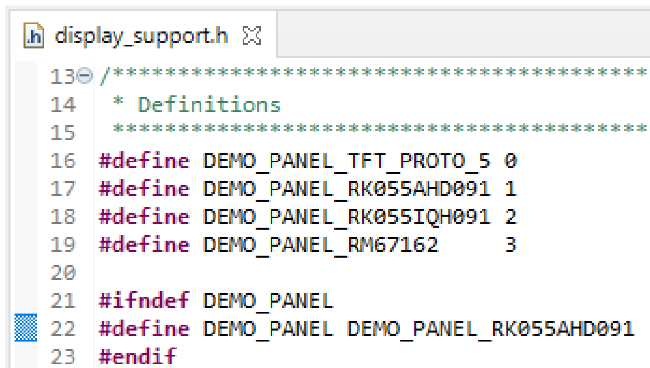


Figure 1. Demo_panel macro in display_support.h

4. Select the OOB project in MCUXpresso, and build it.

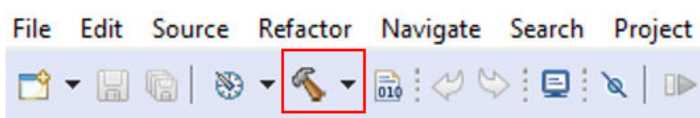


Figure 2. Build icon

5. Connect MIMXRT595-EVK to your computer using a micro USB to J40 (LINK USB) port.
6. Download the CM33 application to your MIMXRT595-EVK.

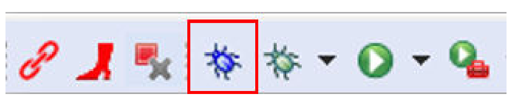


Figure 3. Download icon

7. Run the application in MCUXpresso.

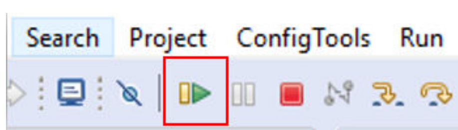


Figure 4. Run icon

8. Open Xtensa Xplorer IDE and configure the following options as shown in Figure 5.

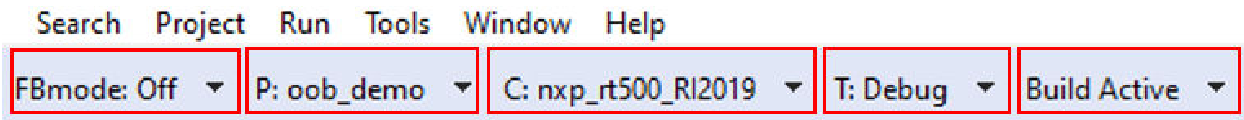
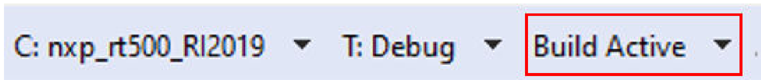


Figure 5. Configure the options

9. Click the **Build Active** button to build the project.



10. Open a command prompt on the following location "*C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.01*" and execute the following command: *"xt-ocd.exe -c topology.xml".* The command prompt appears as in Figure 6.

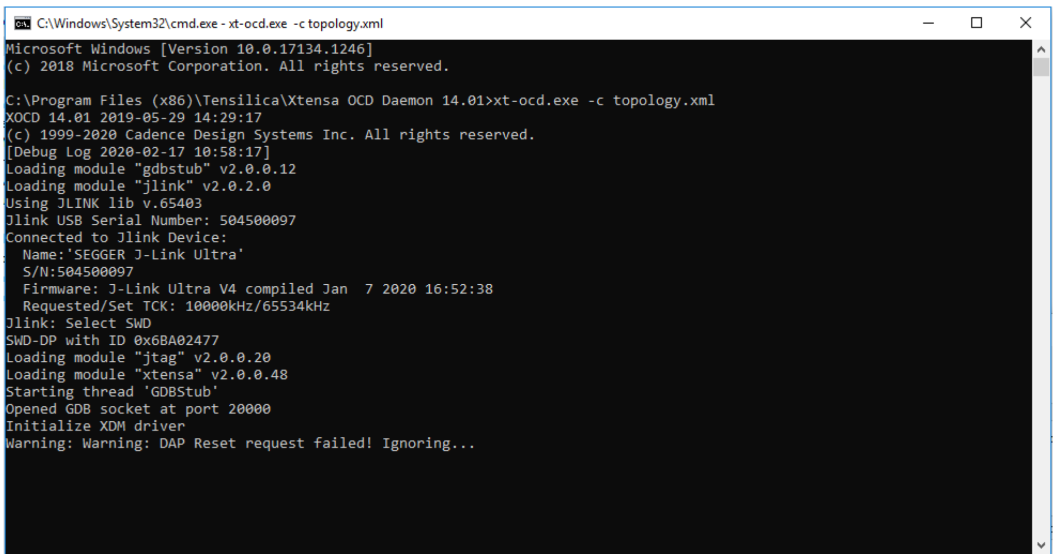


Figure 6. Command prompt

11. Return to Xtensa Xplorer and select **Debug > Debug Configurations**.

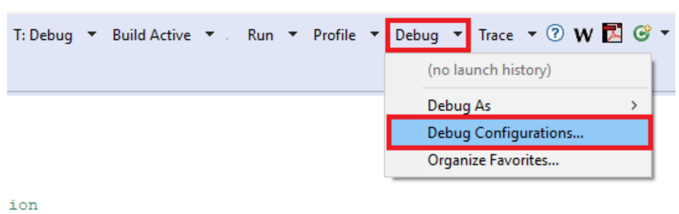
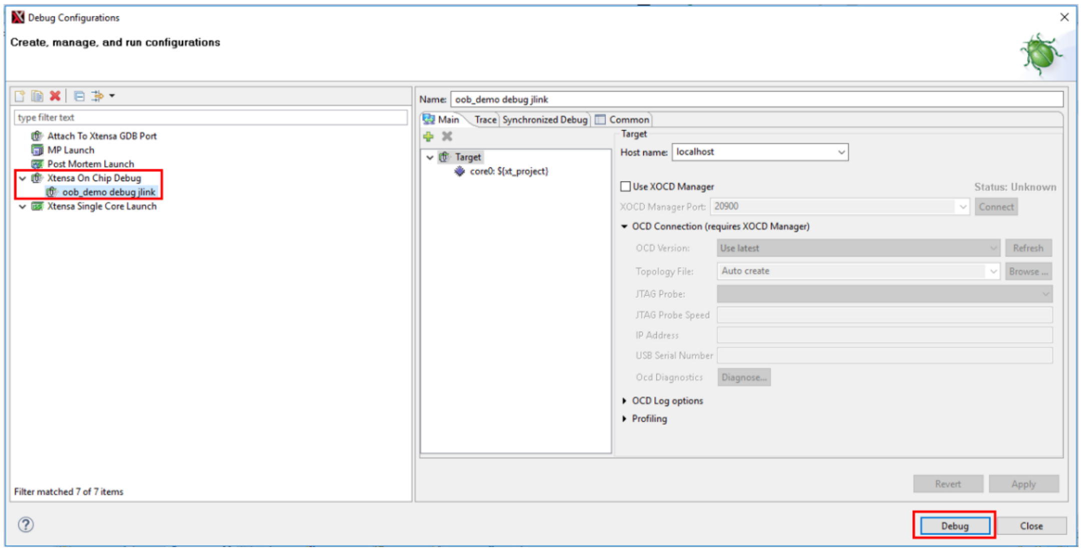


Figure 7.

12. In the **Debug Configurations** window, select **Xtensa On-Chip Debug > oob_demo debug jlink**.

13. Click the **Debug** button.



14. The **Download binary** dialog box prompts to download the application to the core 0.

15. Click **Yes**. The project starts downloading the code to the RT595.

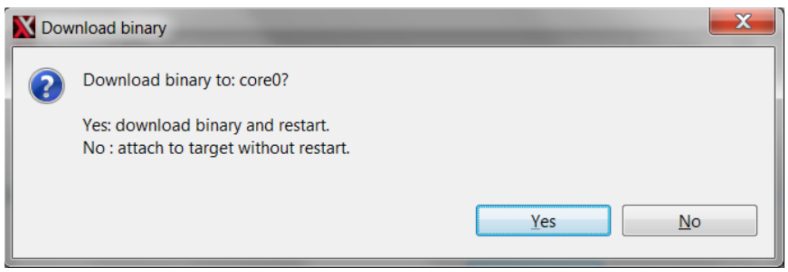


Figure 8. Download library

16. Run the program on Xtensa Xplorer IDE.

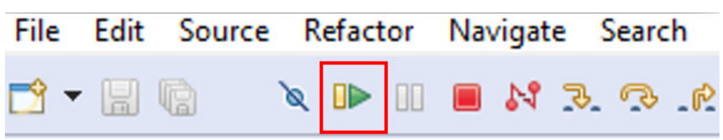


Figure 9. Run icon

17. The project starts running on MIMXRT595-EVK. The red LED blinks and the following information appears on the terminal:

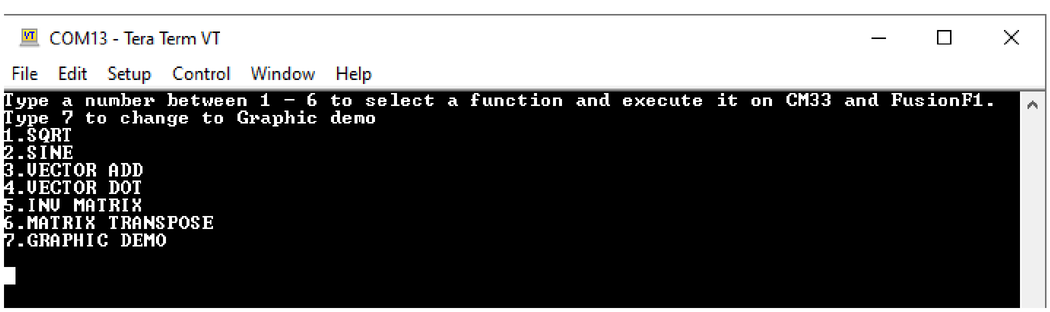


Figure 10. Terminal window

18. To select one of the options, type a number on the serial terminal. For details on the demo project, see the chapter Project overview.

4 Project overview

This demo uses the Cortex®-M33, Fusion F1 DSP, and the GPU. The first six options execute a specific math function and show a simple performance comparator between the Cortex®-M33 and the Fusion F1. The last option draws on the display connected through the MIPI port and shows the GPU performance on the serial terminal.

To execute a math function from option 1-6, type the function number. The cycle count result of each core appears in the terminal. Both the Cortex®-M33 and Fusion F1 DSP cores can execute the functions listed in Table 1.

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 a length of 200 each.
Vector Dot Product	Executes the vector dot product of two float vectors with a 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 communicate the DSP the math function that has to execute. forFor details on Message Unit, see Chapter 42: Messaging Unit on the RT500 User Manual.

4.1 Cortex®-M33

This section shows the code and libraries needed for the Cortex® -M33 application.

The Cortex® -M33 uses some files to execute the math functions. The source files are available at the following path: '*<SDK path>/CMSIS/DSP/Source*'.

The source files required for this demo are:

- arm_common_tables.c
- arm_const_structs.c
- arm_dot_prodf_32.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, all required initializations including clocks, debug console, and the message unit.

The program starts with pin initialization functions. Each function initializes different components used in the demo.

Table 2. Pin configurations

Pin initialization function	Description
BOARD_InitPins()	Initialize the RGB LED and PMIC pins.
BOARD_InitUARTPins()	Initialize the UART pins used for the serial communication with the PC.
BOARD_InitPsRamPins()	Initialize the pSRAM pins used for the GPU demo section.
BOARD_InitFlexIOPanelPins()	Initialize the FlexIO pins if the TFT Proto display is selected on the macro 'DEMO_PANEL'. If set, connect the display through J43 at the back of the board.
BOARD_InitMipiPanelPins()	Initialize MIPI pins if a mipi display is selected on the macro 'DEMO_PANEL'. This is the default configuration. If set, connect the display through J44 at the back of the board.

main_cm.c

```c
int main(void)
{
    /* Init board hardware. */
    status_t status;
    usart_config_t config;

    BOARD_InitPins();
    BOARD_InitUARTPins();
    BOARD_InitPsRamPins();

#if (DEMO_PANEL_TFT_PROTO_5 == DEMO_PANEL)
    BOARD_InitFlexIOPanelPins();

    GPIO_PortInit(GPIO, BOARD_SSD1963_RST_PORT);
    GPIO_PortInit(GPIO, BOARD_SSD1963_CS_PORT);
    GPIO_PortInit(GPIO, BOARD_SSD1963_RS_PORT);
#else
    BOARD_InitMipiPanelPins();

    GPIO_PortInit(GPIO, BOARD_MIPI_POWER_PORT);
    GPIO_PortInit(GPIO, BOARD_MIPI_BL_PORT);
    GPIO_PortInit(GPIO, BOARD_MIPI_RST_PORT);

#if (DEMO_PANEL_RM67162 == DEMO_PANEL)
    GPIO_PortInit(GPIO, BOARD_MIPI_TE_PORT);
#endif

#endif

    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();

    status = BOARD_InitPsRam();
    if (status != kStatus_Success)
    {
        assert(false);
    }
```

Figure 11. Board and pins initialization.

The next step is to initialize the peripherals used in the demo.

Table 3. Peripherals initialization functions

Peripheral initialization function	Description
CTIMER_INIT()	A CTIMER is used for time measuring. The CTIMER2 is configured to trigger every 5 microseconds to obtain the cycle count for the CM33.
LED_INIT()	Initializes RGB LED.
MU_Init(APP_MU)	Initializes the Message Unit, used to communicate the CM33 with the Fusion F1 DSP

```
/* Initialize CTIMER */
CTIMER_INIT();
/* Initialize LED */
LED_INIT();
/* Clear MUA reset */
RESET_PeripheralReset(kMU_RST_SHIFT_RSTn);
/* MUA init */
MU_Init(APP_MU);
```

Figure 12. Peripherals initialization functions

Now, initialize the DSP core with BOARD_DSP_Init(). This function initializes the PMIC, configure the DSP clock, and sets the DSP image address to copy the Fusion F1 application into RAM.

The Cortex® -M33 can copy the entire DSP application into RAM. You can enable or disable this action by changing the macro 'DSP_IMAGE_COPY_TO_RAM' value. In this demo, this action is disabled (DSP_IMAGE_COPY_TO_RAM = 0) by default for debugging purposes.

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

Figure 13. DSP initialization

It is important to remark that the Cortex-M33 has to start the DSP operation setting SYSCTL0_DSPSTALL register inside DSP_Start() function. For further information about DSP initialization and configuration, see the Getting Started with Xplorer for EVK-MIMXRT595. You can find this document inside the SDK folder: '*<SDK path>/docs/*'.

Once the DSP is configured and running, the CM33 waits for the Fusion F1 boot flag. This is to ensure 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 14. DSP boot flag

Now, the UART interrupt is enabled to receive the information typed on the serial terminal.

```
/* 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;

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 15. UART interrupt

Both cores are ready to start the application! The CM33 is waiting for input on the serial terminal and toggling the red LED. The Fusion F1 is waiting for a message from CM33 with the MU. When a number is typed on the terminal, the UART interrupt is triggered, and the application executes the basic performance comparator between the CM33 and Fusion F1 DSP. You can find this inside the cpu_test() function

```
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 16. UART interrupt handler

If the input data corresponds to a number between 1 to 6, then the CM33 executes the math function (showed on the list below) and sends a message to Fusion F1, indicating which function has to run.

- 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()

```
switch(dataTyped)
{
case '1':
  /* Execute square root */
  arm_mat_sqrt_Test();        Execute math function
  /* Communicate with FusionF1 to execute math function */
  MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 1);        Send message to Fusion F1
  dataTyped = 0;        Clear input data variable
  break;
case '2':
  /* Execute sine */
  arm_mat_sine_Test();
  /* Communicate with FusionF1 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 FusionF1 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 FusionF1 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 FusionF1 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 FusionF1 to execute math function */
  MU_SendMsg(APP_MU, CHN_MU_REG_NUM, 6);
  dataTyped = 0;
  break;
case '7':
  /* Change to graphic test */
  graphicTest = 1;        Set flag to execute graphic test
  dataTyped = 0;
  break;

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

Figure 17. Main loop

All math functions are executed LOOP_COUNT times to obtain an average on the cycle count result; by default, this macro has a value of 10000. All functions used by the CM33 have a similar structure, as shown in the following image.

```c
static void arm_mat_sqrt_Test()
{
  uint32_t i;
  uint32_t cycles;
  PRINTF("---------------------------------\r\n");
  PRINTF("        SQRT FUNCTION\r\n");

  q31_t input      = FLOAT_2_Q31(0.25f);

  /* Reset count variable */
  countUseconds = 0;
  /* Start ctimer */
  CTIMER_StartTimer(CTIMER);

  /* Execute math function */
  for(i = 0; i < LOOP_COUNT; i++)
  {
    arm_sqrt_q31(input,&sqrtResult);
  }

  /* Stop ctimer */
  CTIMER_StopTimer(CTIMER);

  /* Verify the result */
  if(abs(sqrtRef - sqrtResult) > 2)
    PRINTF("ERROR on SQRT\r\n");

  /* Convert ctimer counts to cycles */
  cycles = (uint32_t)((countUseconds*CYCLES_PER_COUNT)/LOOP_COUNT);

  PRINTF("CM33 SQRT takes %d cycles\r\n\r\n", cycles);
}
```

Input variable

Restart the counter

Execute math function

Stop Ctimer count

Verify result

Obtain cycle counts

Figure 18. Math function structure

4.2 Vector Graphics Processing Unit (GPU)

If the input data corresponds to a 7, a flag is set indicating that it is time to execute the GPU test. The demo ends with the cpu test, creates vglite_task, and starts the scheduler.

This task manages all the GPU support, clocks, memory registers, and display initialization through VGLite API. The application draws a tiger on the screen and shows you the GPU performance on the serial terminal.

```
static void vglite_task(void *pvParameters)
{
    status_t status;
    vg_lite_error_t error;
    uint32_t startTime, time, n = 0, fps_x_1000;

    status = BOARD_PrepareVGLiteController();
    if (status != kStatus_Success)
    {
        PRINTF("Prepare VGlite contolor error\r\n");
        while (1)
            ;
    }

    error = init_vg_lite();
    if (error)
    {
        PRINTF("init_vg_lite failed: init_vg_lite() returned error %d\n", error);
        while (1)
            ;
    }

    startTime = getTime();
    while (1)
    {
        redraw();
        n++;
        if (n >= 60)
        {
            time       = getTime() - startTime;
            fps_x_1000 = (n * 1000 * 1000) / time;
            PRINTF("%d frames in %d mSec: %d.%d FPS\r\n", n, time, fps_x_1000 / 1000, fps_x_1000 % 1000);
            n          = 0;
            startTime = getTime();
        }
    }
}
```

Configure GPU clocks, memory, and interrupt.

Configure display clocks, the window, and tessellation to start drawing

Update the draw in the display

Obtain FPS and print it on terminal

Figure 19. GPU task

For this demo, the RT595 uses the VG-Lite rendering API, which supports 2D vector/raster rendering operations for interactive user interface and keeps memory footprint to the minimum. This helps implement applications for mobile or IoT devices.

The graphic demo used in this application at '*<SDK path>/boards/evkmimxrt595/vglite_examples/tiger_freertos/'*. You can also find more graphic-focused demos within the RT500 SDK.

4.3 Fusion F1 DSP

This section shows the code and libraries needed for the Fusion F1 DSP application.

NatureDSP is a library provided used by the Fusion F1. This library contains math API's for use. The source files are found in the following path: '`<SDK path>/middleware/dsp/naturedsp/fusionf1`'. The documentation about the library is available inside the '*doc*' folder.

The Fusion F1 uses some files of the Nature DSP library. The files required for this demo are:

• mtx_inv2x2f_fusion.c	• NatureDSP_Signal.h
• mtx_tran4x4f_fusion.c	• NatureDSP_types.h

Table continues on the next page...

- vec_add32x32_fusion_fast.c
- vec_dotf_fusion.c
- vec_recip_table.c
- vec_recip_table.h
- naturedsp_input.h
- scl_sine_table32.h
- scl_sqrt_table.h
- scl_sine32x32_fusion.c
- scl_sine_table32.c
- scl_sqrt32x32_fusion.c
- scl_sqrt_table.c

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.

main_dsp.c

Initialize the debug console and the message unit. The Fusion F1 uses the function MU_SetFlags() to send the boot flag to CM33 and indicates DSP start.

main_dsp.c

```c
int main(void)
{
    usart_config_t config;
    uint8_t g_msgRecv = 0;

    /* 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 20. UART and MU initialization

Finally, the Fusion F1 will wait for CM33 messages using MU_ReceiveMsg(). Depending on the received message from the CM33, the Fusion F1 will call the math function.

```c
/* 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 21. Receive a message from CM33 indicating which function to execute.

srtm_naturedsp_test.c

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

1. TEST_SQRT()
2. TEST_SINE()
3. TEST_VEC_ADD()
4. TEST_VEC_DOT()
5. TEST_MATRIX_INV()
6. TEST_MATRIX_TRANSPOSE()

Each function initialize the required variables, executes the math function from the 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 image.

```
void TEST_SQRT()
{
    uint32_t i;
    const int32_t input       = FLOAT_2_Q31(0.25f);
    int32_t sqrtResult = 0;
    const int32_t sqrtRef     = FLOAT_2_Q31(0.5f);

    /* Obtain init cycle count */
    tic = get_ccount();

    /* Execute math function */
    for(i = 0; i < LOOP_COUNT; i++)
    {
        sqrtResult = scl_sqrt32x32(input);
    }

    /* Obtain end cycle count */
    toc = get_ccount();

    /* Verify the result */
    if((sqrtRef - sqrtResult) > 1000)
        PRINTF("ERROR on SQRT\r\n");

    PRINTF("Fusion F1 SQRT takes %d cycles\r\n\r\n", (toc - tic)/LOOP_COUNT);
}
```

Input, result and reference variables

Obtain cycle count

Execute math function

Obtain cycle count

Verify result

Figure 22. Fusion F1 math function structure

5 Revision history

This table summarizes the revisions of this document.

Table 4. Revision history

Revision number	Date	Substantive changes
0	19 Feb 2021	Initial Draft

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 19 February 2021

Document identifier: AN13151