Application Software Pack: Dynamic Voltage Scaling Using PVT on i.MX RT500

Lab Hand Out - Revision 1

Contents

1 Lab Overview

The application in this lab showcases the Dynamic Voltage Scaling (DVS) capabilities of i.MX RT500 using the internal PVT sensor. This lab covers how to run, customize, and profile this application. Lastly, it explains how to enable the PVT sensor in your own application.

1.1 Application Description

The application first initializes the main_clk to use FRO 192 MHz and sets the CPU divider to 1, meaning that the core frequency is set to 192 MHz. Next, it initializes the PVT sensor by doing the following:

1. Sets VDDCORE to 0.9V.
2. Reads the PVT delay from OTP or the PVT ring oscillator.
3. Enables the PVT interrupt.
4. Enables the PVT alert counter. This tells you the amount of times the PVT interrupt has triggered.
5. Sets the PVT delay.
6. Starts the PVT sensing.

After initializing the necessary hardware, it launches the following tasks:

- **Workload Task**: Runs Coremark for ~10 seconds, prints the results, and then delays for 5 seconds to allow the FreeRTOS idle task to enable deep sleep mode.
- **PVT Task**: The PVT task is in charge of handling DVS in the application. The general flow of the task is shown below

2   Equipment

The following equipment is needed for this lab:
- [MIMXRT595-EVK](#)
- Micro-B USB cable
- [MCUXpresso IDE V11.6.0+](#)
- TeraTerm (or any other terminal emulator)
- Oscilloscope (optional)

3   Resources

The following are helpful resources for this lab:
- [Software Pack Repository](#)
- [Application Note AN13695](#)

4   Import the App Software Pack Into MCUXpresso IDE

There are two methods to get the application software pack. This section will cover both options, but only one needs to be done.

4.1 Option #1: Get the App Software Pack with MCUXpresso IDE

1. Open MCUXpresso IDE and select a workspace location in an empty directory.
2. Right click in the blank area of the **Installed SDKs** panel at the bottom and select **Import remote SDK Git repository…**
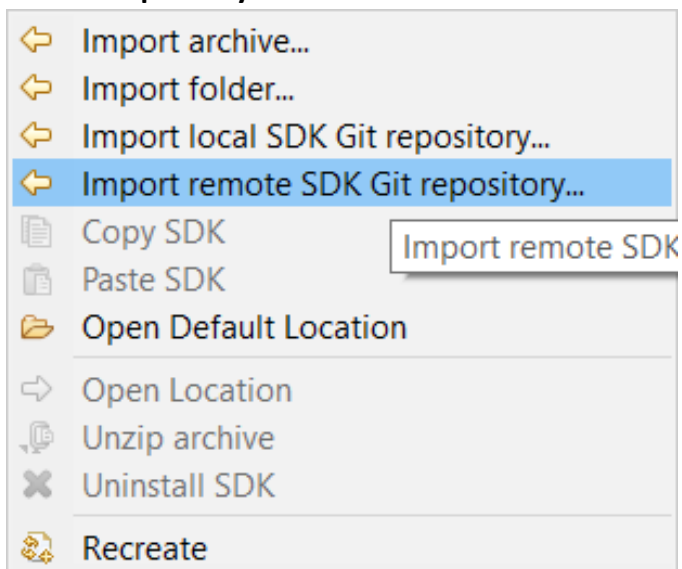
3. In the dialog box that comes up:
   a) In the **Location** field, click on the Browse button and create an empty directory named **appswpacks-dvs-pvt-sensor** for the application software pack to be downloaded to. Make note of this location as it'll be used throughout this lab.
   b) In the **Repository** field put: **https://github.com/NXPmicro/appswpacks-dvs-pvt-sensor**
   c) In the **Revision** field put: **mcux_release_github**

Then hit **OK** to download the application software pack.
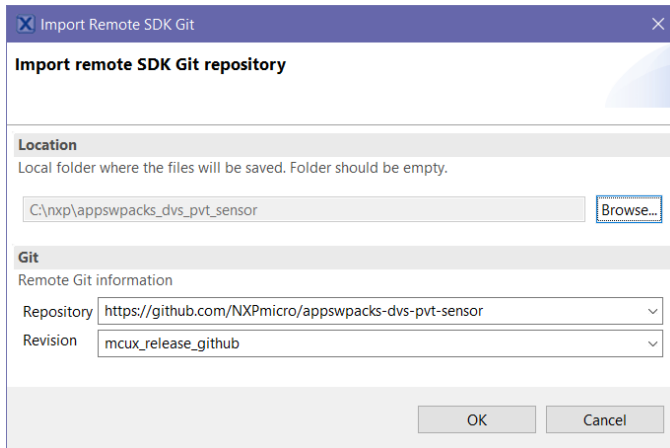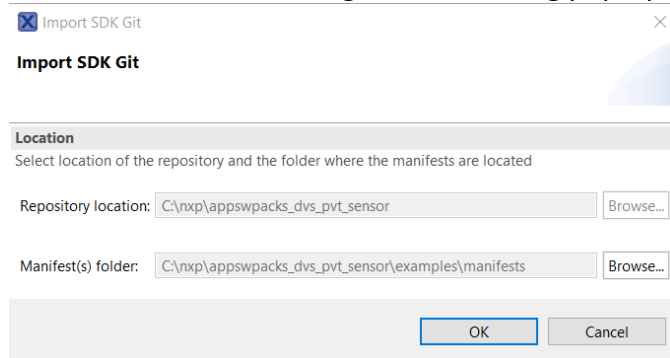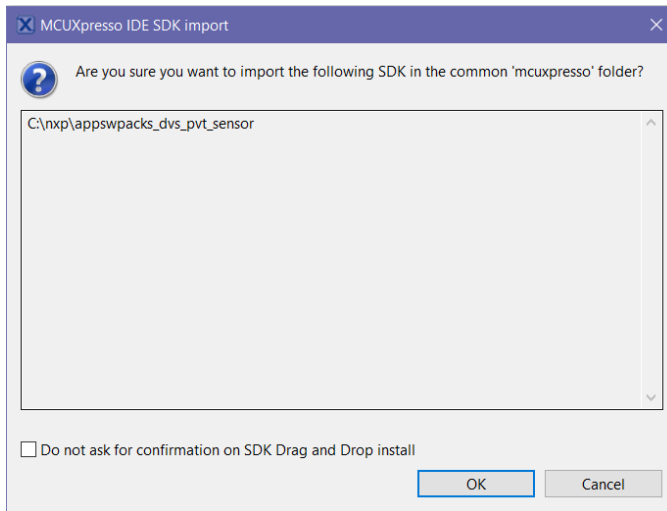

4. Once imported the Installed SDKs tab will look like this:

4.2 Option #2: Use the Command Line

1. Open up a Windows command line and execute the following:
   west init -m https://github.com/NXPmicro/appswpacks-dvs-pvt-sensor --mr mcux_release_github appswpacks_dvs_pvt_sensor
   cd appswpacks_dvs_pvt_sensor
   west update
2. Open MCUXpresso IDE and select a workspace location in an empty directory.
3. Drag-and-drop the **appswpacks_dvs_pvt_sensor** directory that was created in the previous step into the **Installed SDKs** window, located on a tab at the bottom of the screen named "Installed SDKs". You will get the following pop-ups, so hit **OK.**
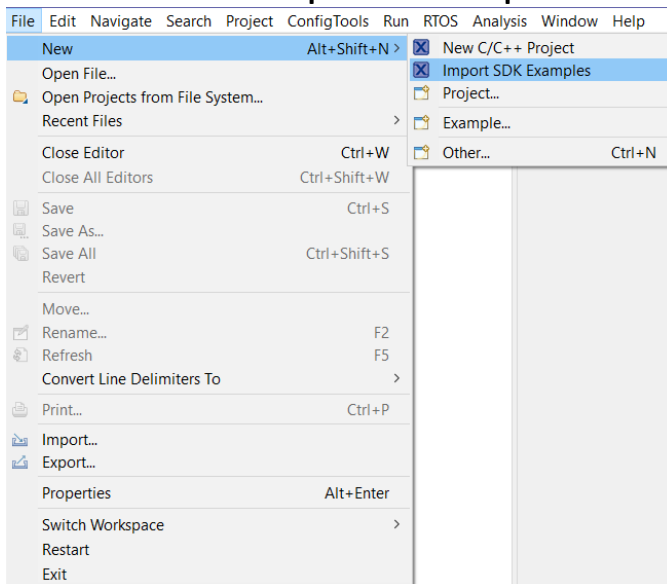
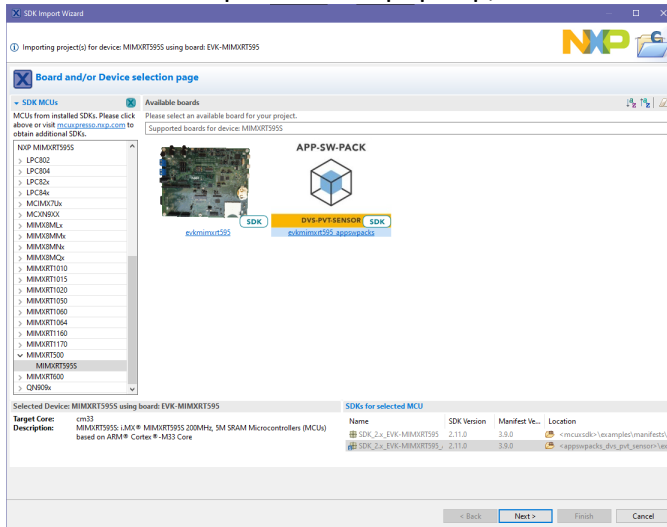4. Once imported, the Installed SDK panel will look something like this:

4.3 Importing the Project Into Your Workspace

Once you finish importing the Software Pack into the Installed SDKs, it is time to import the project into your workspace.
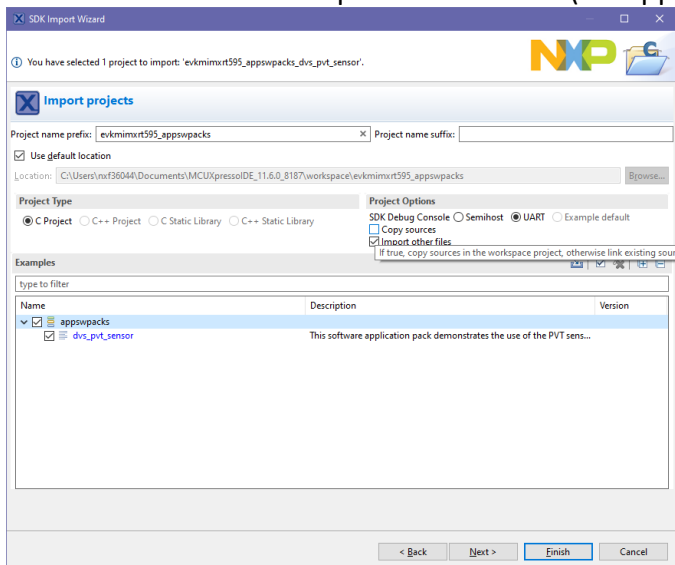
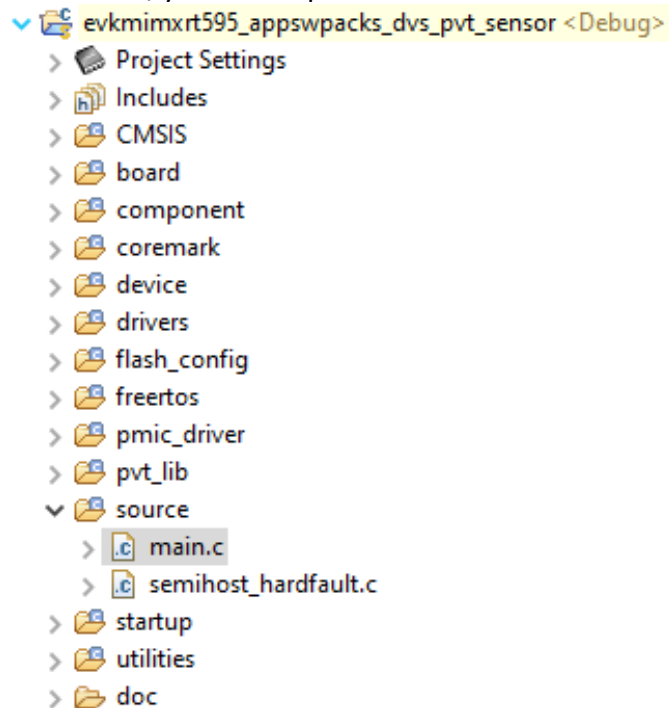1. Go to **File -> New -> Import SDK Examples.**

2. Once the SDK Import Wizard pops up, select **evkmimxrt595_appswpacks** and click **Next.**

3. Check the **appswpacks** checkbox.
4. Check or Uncheck the **Copy sources** checkbox and click **Finish**.
    a. If checked, MCUXpresso will make copies of all project files and place them in your selected workspace directory.
    b. If unchecked, MCUXpresso will link the project files to the files in the repository that was cloned in the previous sections (i.e. appswpacks_dvs_pvt_sensor)

5. Once done, your workspace should look like below.

- ∨ evkmimxrt595_appswpacks_dvs_pvt_sensor <Debug>
  - › Project Settings
  - › Includes
  - › CMSIS
  - › board
  - › component
  - › coremark
  - › device
  - › drivers
  - › flash_config
  - › freertos
  - › pmic_driver
  - › pvt_lib
  - ∨ source
    - › main.c
    - › semihost_hardfault.c
 - › startup
  - › utilities
  - › doc

5 Running the Application

This section describes how to configure your EVK jumpers for the application and then walks you through the steps to build, flash, and run the application.

5.1 EVK Configuration

This application leverages the on-board PMIC, so make sure a jumper is on JS28 pins [2:3] and another jumper is on JS25. The picture below highlights where these, along with other key pins and ports, are located on the EVK.

- **SW3 : Reset**
- **J40** : **Link USB** – Debug Port
- **JP27[2] : CLKOUT** – Core Frequency / 200
- **JS25[1] : PMIC_SW1_OUT** – VDDCORE
- **J28[2] : PIO4_28** – PVT Task Profiling
- **JS28[2:3] : LDO_ENABLE** – Setting it to [2:3] disables the internal LDO
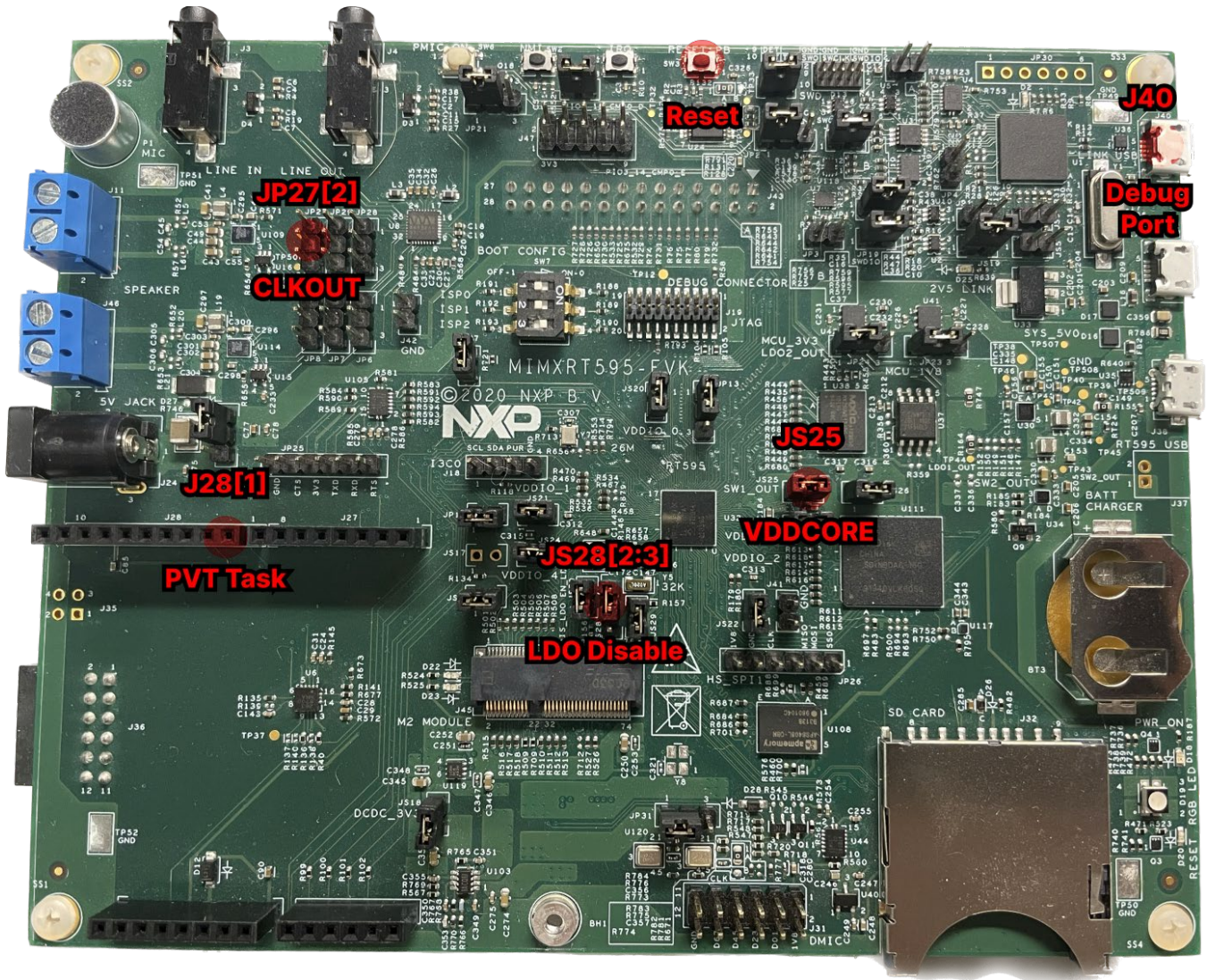
5.2 Building, Flashing, and Running

Once you finish configuring the EVK jumpers, you can build, flash, and run the application.

1. Build the project by clicking **Build** in the Quickstart Panel.
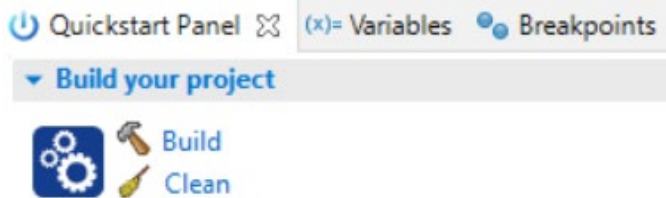
2. Once the build finishes, the output console should look like below.

```
Building target: evkmimxrt595_appswpacks_dvs_pvt_sensor.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\Users\nxf36044\Documents\MCUXpressoIDE_11.6.0_81
Memory region        Used Size   Region Size  %age Used
     QSPI_FLASH:      70724 B       64 MB       0.11%
           SRAM:      24748 B      2560 KB       0.94%
        USB_RAM:          0 GB       16 KB       0.00%
Finished building target: evkmimxrt595_appswpacks_dvs_pvt_sensor.axf

Performing post-build steps
arm-none-eabi-size "evkmimxrt595_appswpacks_dvs_pvt_sensor.axf"; # arm-none-eabi-o
   text    data     bss     dec     hex filename
  70724       0   19356   90080    15fe0 evkmimxrt595_appswpacks_dvs_pvt_sensor.axf


13:10:10 Build Finished. 0 errors, 0 warnings. (took 22s.291ms)
```
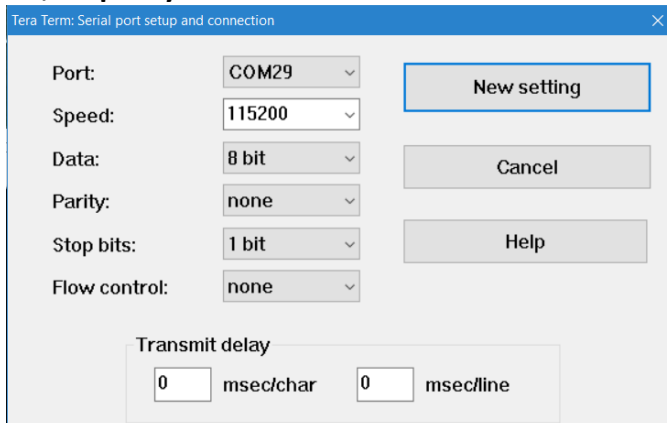
3. Connect the micro-B USB cable to **J40** on the EVK.
4. Open TeraTerm or a different terminal emulator, and connect to the COM port belonging to the EVK.
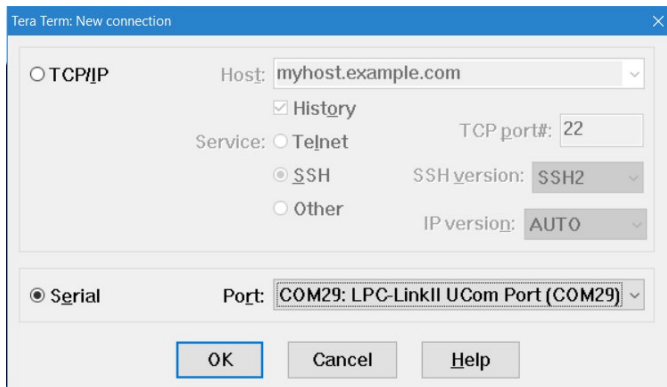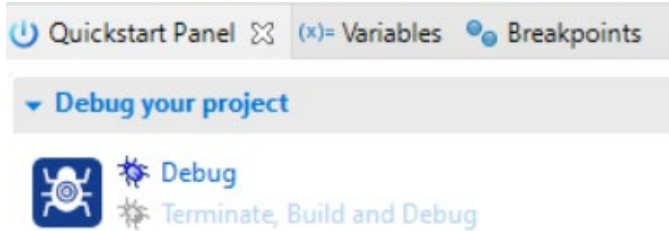
5. In TeraTerm, go to **Setup -> Serial Port** and configure the settings to use **115200 baud, 1 stop bit, no parity**.

6. In TeraTerm, go to **Setup -> Terminal** and change the **New-line Receive** to **Auto** or **LF**. This is necessary for the Coremark results to print correctly.
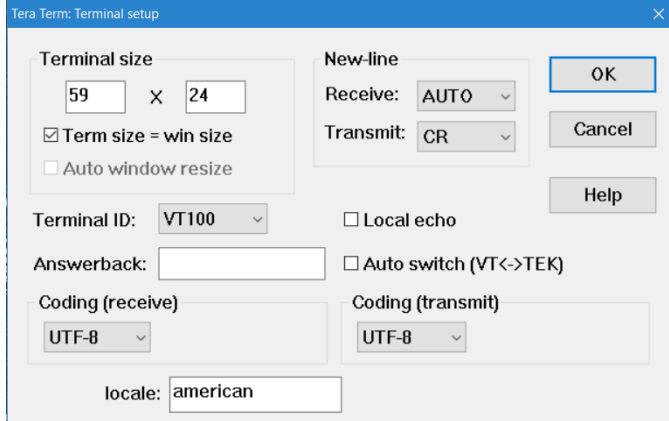
7. Debug the project by clicking on **Debug** in the Quickstart Panel.

8. It will ask what interface to use. Select the detected debug probe. The screenshot below shows the default CMSIS-DAP probe.
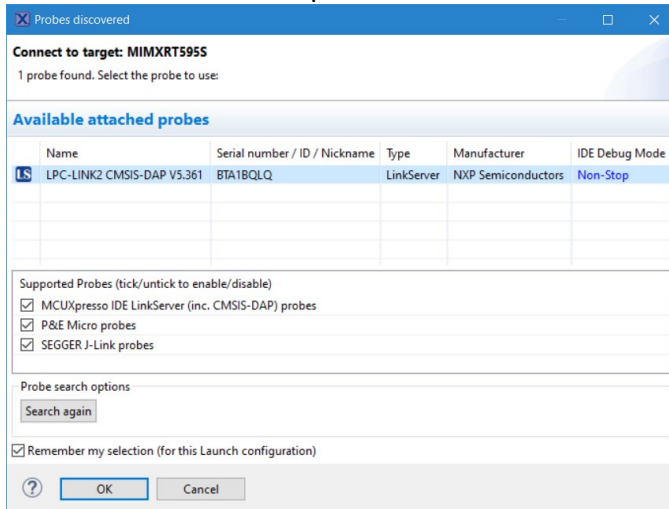
9. The debugger will download the firmware and open up the debug view. Click on the Resume button to start running.

10. The following text should appear in the terminal program. If the warning appears, make sure the conditions stated are met and press **ENTER** to continue.
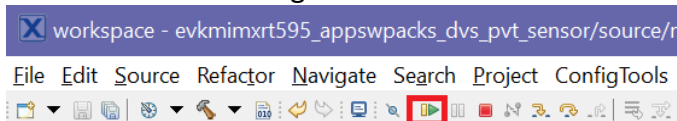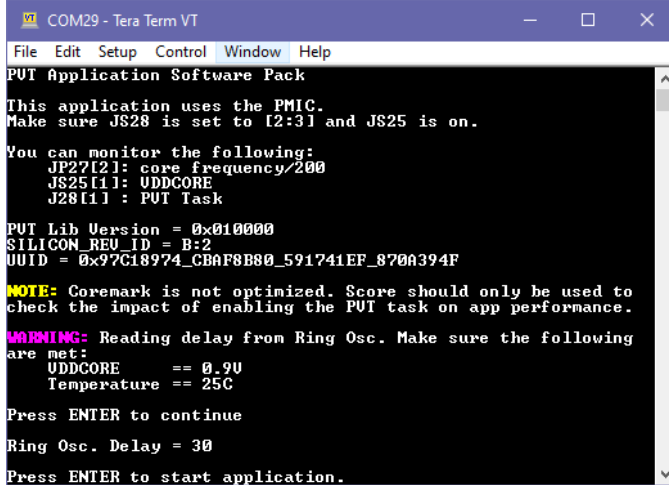
```
COM29 - Tera Term VT
File  Edit  Setup  Control  Window  Help
PVT Application Software Pack

This application uses the PMIC.
Make sure JS28 is set to [2:3] and JS25 is on.

You can monitor the following:
    JP27[2]: core frequency/200
    JS25[1]: UDDCORE
    J28[1] : PVT Task

PVT Lib Version = 0x010000
SILICON_REV_ID = B:2
UUID = 0x97C18974_CBAF8B80_591741EF_870A394F

NOTE: Coremark is not optimized. Score should only be used to
check the impact of enabling the PVT task on app performance.

WARNING: Reading delay from Ring Osc. Make sure the following
are met:
    UDDCORE     == 0.9V
    Temperature == 25C

Press ENTER to continue

Ring Osc. Delay = 30

Press ENTER to start application.
```
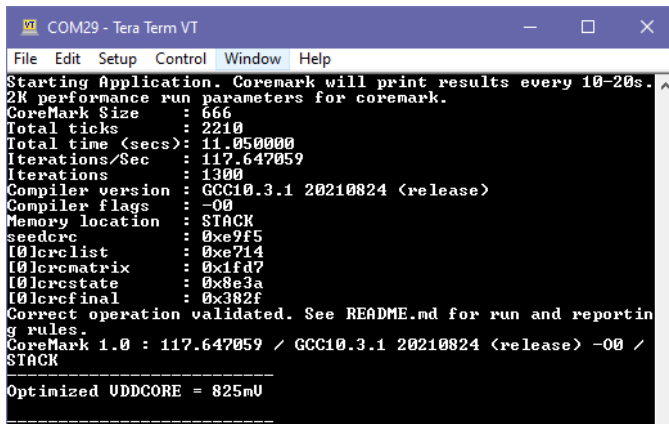
11. Press **ENTER** again to start the application. The Coremark results should print every 10-20s as shown below.
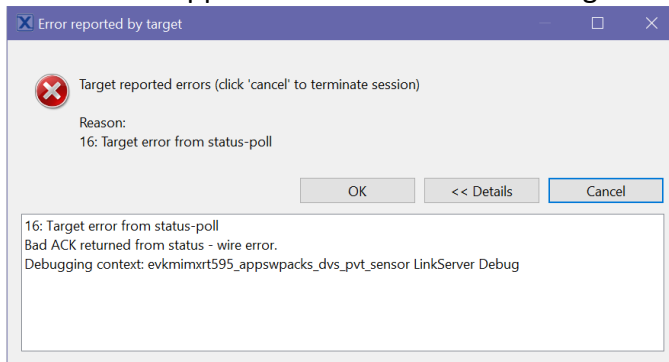
```
COM29 - Tera Term VT
File  Edit  Setup  Control  Window  Help
Starting Application. Coremark will print results every 10-20s.
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 2210
Total time (secs): 11.050000
Iterations/Sec   : 117.647059
Iterations       : 1300
Compiler version : GCC10.3.1 20210824 (release)
Compiler flags   : -O0
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x382f
Correct operation validated. See README.md for run and reportin
g rules.
CoreMark 1.0 : 117.647059 / GCC10.3.1 20210824 (release) -O0 /
STACK
-------------------------------
Optimized UDDCORE = 825mV
-------------------------------
```

NOTE: The application is not configured to maximize the Coremark score. The Coremark score should only be used to profile the impact the PVT task has on application performance.

12. Since the application goes into deep sleep mode after running Coremark, it will cause the debug session to disconnect and you will see the message below. Click **Cancel** to exit the debug session. The application will continue running on the EVK.

```
Error reported by target

    ⊗   Target reported errors (click 'cancel' to terminate session)

        Reason:
        16: Target error from status-poll

                          OK      << Details    Cancel

16: Target error from status-poll
Bad ACK returned from status - wire error.
Debugging context: evkmimxrt595_appswpacks_dvs_pvt_sensor LinkServer Debug
```

6 Customizing the Application

There are several ways to customize the application to help you profile power and runtime performance in different scenarios and help you determine how to best integrate the PVT sensor into your own application. All of the following defines are located in **main.c**.

6.1 Enable/Disable the PVT Task

You can disable the PVT task by setting ENABLE_PVT to 0.

```
/**
 * - 0: Disables PVT task. Can be used to profile impact of PVT task on performance.
 * - 1: Enables the PVT task.
 */
#define ENABLE_PVT              1
```

6.2 PVT Task Wait Time

This value determines how often the PVT task checks if it can set VDDCORE lower. You have the liberty to experiment with different values to determine which value works best for your application.

```
/**
 * How often for PVT task to check if VDDCORE can go lower.
 */
#define PVT_TASK_WAIT_MS        10000U
```

6.3 PMIC Settling Time

This value represents the amount of time it takes for the PMIC to change the voltage one unit and for the voltage to settle to the new value. This value is dependent on the board design and PMIC you are using. Change it accordingly. Refer to AN13695 for more information.

```
/**
 * Amount if time it takes the PMIC SW1_OUT to stabilize after decreasing it 1 step.
 */
#define PMIC_SETTLING_TIME_MS   5U
```

6.4 Ring Oscillator Wait Time

This value is used to calculate the ring oscillator frequency, which in turn is used to calculate the ideal PVT delay for the device. If your device needs to use the ring oscillator to read the PVT delay, then it is recommended to keep this value at or near 500ms.

```
/**
 * Time to wait between enabling the PVT ring oscillator and reading the delay.
 */
#define PVT_RING_OSC_WAIT_TIME_MS  500U
```

6.5 Workload Task Delay and Disabling Deep Sleep Mode

This allows you to control how long the application stays in deep sleep mode. You can:
   a. Increase the amount of time it spends in deep sleep mode by increasing the delay.
   b. Decrease the amount it spends in deep sleep mode by decreasing the delay.
   c. Prevent it from going into deep sleep mode by setting it to 0.

```
/*
 * Amount of time to wait in between Coremark runs.
 * Allows idle task to activate low power mode (deep sleep).
 */
#define WORKLOAD_DELAY_MS           5000U
```

6.6 Task Priorities

It is recommended to keep the PVT task as the highest priority so it can react to PVT interrupts in a timely manner.

```
/**
 * Task priorities.
 * PVT Should be highest priority so it can increase VDDCORE ASAP if the PVT
 * interrupt triggers.
 */
#define workload_task_PRIORITY     (configMAX_PRIORITIES - 2)
#define pvt_task_PRIORITY          (configMAX_PRIORITIES - 1)
```

6.7 Max and Min VDDCORE Voltages

These values can vary depending on your application. Please refer to section "Requirements for depending on PVT Sensor" in AN13695 for more details.

```
/**
 * PMIC defines.
 */
#define MIN_VDDCORE                kPCA9420_Sw1OutVolt0V800
#define MAX_VDDCORE                kPCA9420_Sw1OutVolt0V900
```

6.8 PVT Wait Timer

The UTICK timer was chosen for this application because of its simplicity, but you are free to use a different timer.

```
/*
 * Timer to signal the PVT task to run.
 */
#define PVT_WAIT_TIMER             UTICK0
```

7   Profiling the Application

The application by default configures the following pins for profiling purposes:
- **JP27[2] : CLKOUT** – Core Frequency / 200
- **JS25[1] : PMIC_SW1_OUT** – VDDCORE
- **J28[2]: PIO4_28** – PVT Task Profiling

Refer to EVK Configuration to see where they're located on the EVK.

7.1 VDDCORE Sample Capture

Below is a sample capture of VDDCORE over temperature using the default application settings. The temperature is first set to 25C to read the PVT delay from the ring oscillator. It is then increased to 70C

before finally decreasing it to 20C. A temperature chamber was used to control the temperature.


7.2 PVT Task Enabled vs. Disabled

This can help determine the effect the PVT task has on the following:

1. **Power consumption:** Compare core power consumption at JS25 with the PVT task enabled and disabled.
2. **App performance:** Compare Coremark scores of the application with the PVT task enabled and disabled.

7.3 PVT Task CPU Load

Since the voltage scaling is done in software, there is some overhead associated with it, but it is minimal. To profile this, we can attach a probe on J28[2] and measure the amount of time the application spends in the PVT Task. The below image was captured using the Release build configuration (-O3 optimization) and low power mode disabled (WORKLOAD_DELAY_MS == 0).
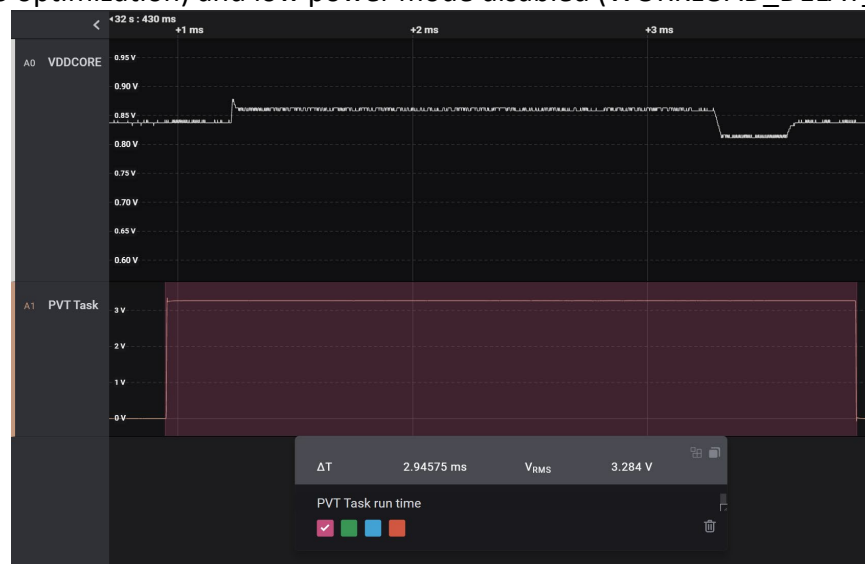
As you can see, it takes around 2.94ms for the application to test and set a lower voltage. Since we chose the PVT Task to run every 10s, the estimated load on the CPU is:

$$CPU\ LOAD = \frac{0.00294}{(10 + 0.00294)} * 100 = 0.0293\%$$

It is important to note that this is a simplified scenario in which we assume that the PVT task is only able to adjust the voltage by one step every time and runs periodically every 10s. Additionally, the PVT Task run time is highly dependent on the PMIC settling time.

8   Enabling the PVT Sensor in Your Application

Enabling the PVT sensor in your own application is fast and simple because the PVT library is a standalone static library that does not depend on the NXP SDK.
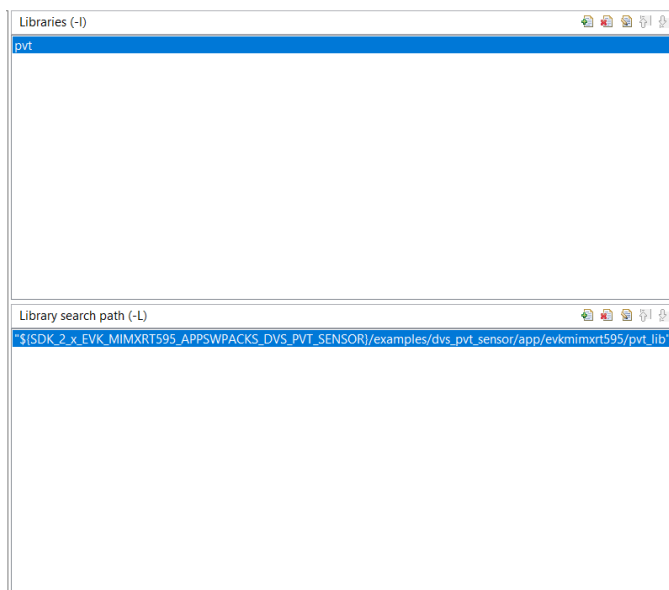
1. Once you create your project, open the project settings by going to **Project -> Properties**.
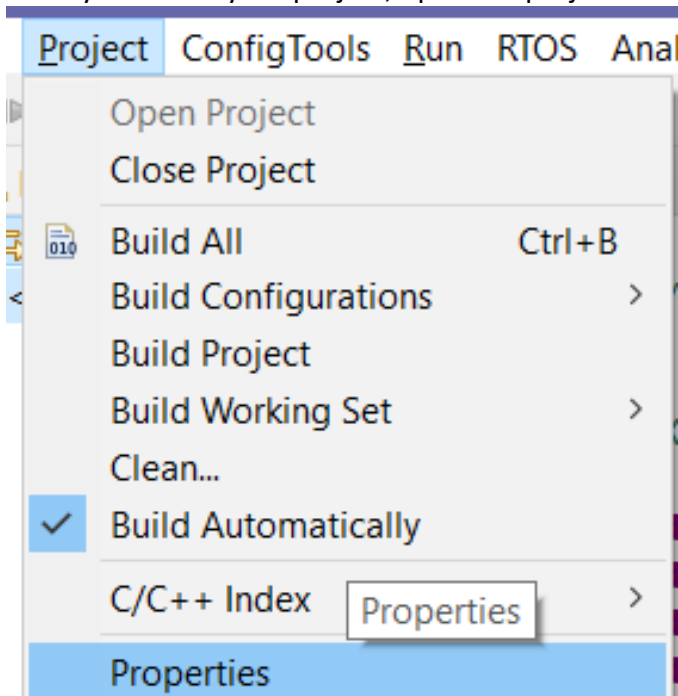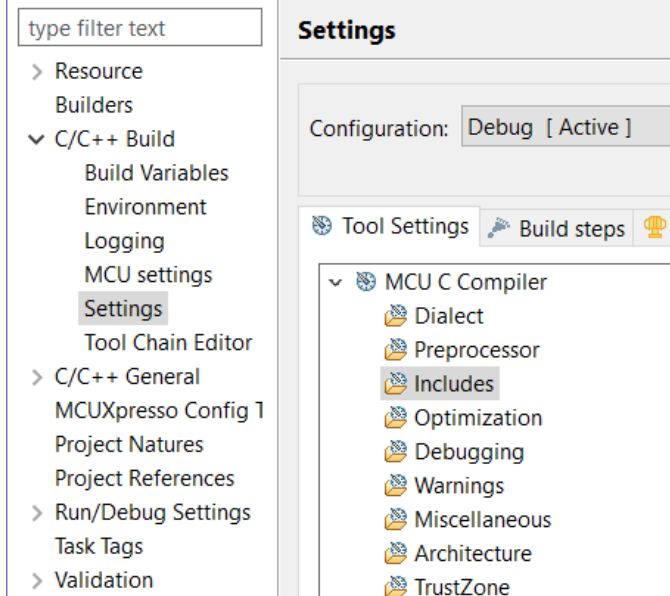
2. In the project settings go to **C/C++ Build -> Tool Settings -> MCU Linker -> Libraries.**
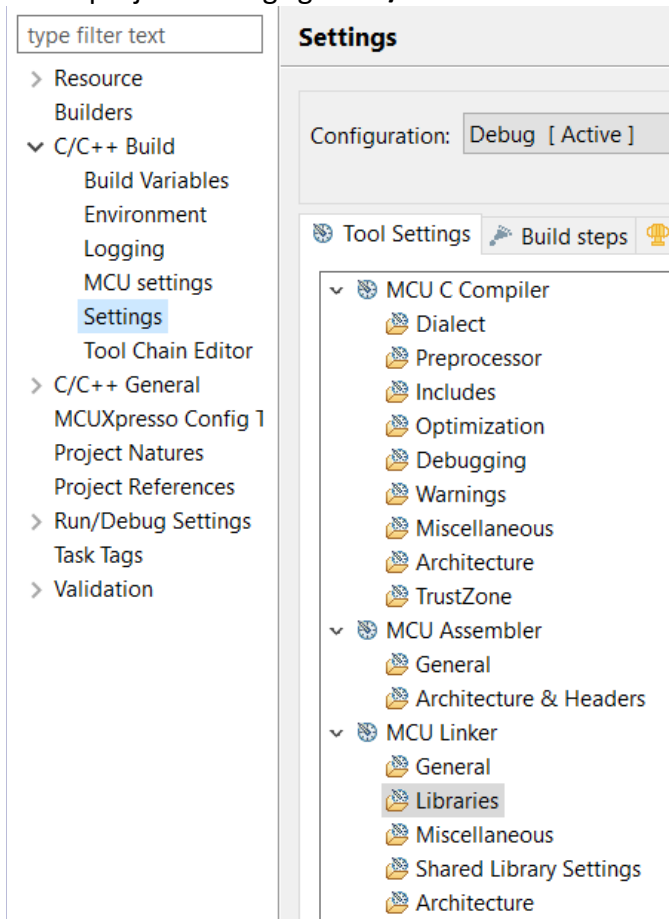
3. Under **Libraries** add **pvt** and under **Library search path** add
    "${SDK_2_x_EVK_MIMXRT595_APPSWPACKS_DVS_PVT_SENSOR}/examples/dvs_pvt_sensor/app/evkmimxrt595/pvt_lib"

4. With the project settings still open, go to **C/C++ Build -> Tool Settings -> MCU C Compiler -> Includes.**

5. Under **Include paths** add:

   "${SDK_2_x_EVK_MIMXRT595_APPSWPACKS_DVS_PVT_SENSOR}/examples/dvs_pvt_sensor/app/common/pvt_lib"

   "${SDK_2_x_EVK_MIMXRT595_APPSWPACKS_DVS_PVT_SENSOR}/examples/dvs_pvt_sensor/app/evkmimxrt595/pvt_lib"

6. Click on **Apply and Close** on the bottom right of the settings window to apply your changes and close the project settings window.
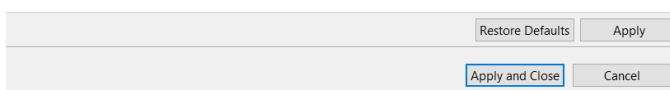
7. Finally, include pvt.h in your source:

```
#include "pvt.h"
```

8.1 Important Notes to Remember

There are a couple of notes to keep in mind when integrating the PVT sensor into your application. You can find detailed explanations for each one in AN13695.

1. You must use FRO trimmed at 192 MHz for main_clk since this is the only one that has been characterized and tested. Other frequencies for main_clk are not supported with the PVT sensor at this time.
2. The absolute minimum VDDCORE voltage is 0.8V as explained in section "Minimum voltage required" in AN13695.
3. VDDCORE must be set to 0.9V when reading the PVT delay using the ring oscillator.

4. Keep the delay time between starting the ring oscillator and reading the delay as close to 500ms as possible. We recommend using a timer for this as shown below.

```c
volatile bool ring_osc_ready;

static void ringo_wait_timer_cb(void) {
    ring_osc_ready = true;
}

void read_pvt_delay_from_ring_osc() {
    pvt_delay_t delay;
    PVT_EnableRingOsc();
    ring_osc_ready = false;
    /* Use a timer to wait 500ms because it is more accurate */
    START_RINGO_WAIT_TIMER(PVT_RING_OSC_WAIT_TIME_MS);
    while (!ring_osc_ready);
    PVT_ReadDelayFromRingOsc(PVT_RING_OSC_WAIT_TIME_MS, &delay);
    PVT_DisableRingOsc();
}
```

5. Don't forget to disable the ring oscillator after reading the delay to prevent unnecessary power consumption as shown above.
6. It is recommended to disable PVT sensing in the PVT interrupt to give the application time to adjust VDCCORE.

Please refer to section "Using the PVT Sensor in application" in AN13695 for further details.

9 PVT Library API

Refer to pvt.h and pvt_int.h in the application software pack's *pvt_lib* directory.

10 Conclusion

This lab demonstrated how to use the PVT Library to enable the PVT sensor and implement DVS on i.MX RT500.