NXP Semiconductors
Application Note

Document Number: AN12405
Rev. 0, 05/2019

Multiple Connections in Bluetooth LE Peripheral Device

Contents

1. Introduction

NXP provides a complete Bluetooth® LE solution that enables you to create applications that support up to eight simultaneous connections using the KW36/35 SoC, which can be configured as either a central device or a peripheral device. This application note describes the procedure to enable multiple connections on a Bluetooth LE peripheral device using the Temperature Sensor demo application.

2. Prerequisites

These items are required to complete the implementation of multiple connections on a peripheral device:

- At least 3 FRDM-KW36 modules
- FRDM-KW36 SDK package
- MCUXpresso IDE
- Temperature Collector demo application
- Temperature Sensor demo application
- TeraTerm or any other serial terminal software

3. Enabling multiple connections on Bluetooth LE peripheral device

This section shows how to enable multiple connections using the Temperature Sensor application and MCUXpresso IDE.

3.1. Creating workspace and importing SDK to MCUXpresso IDE

1. Download the FRDM-KW36 SDK at https://mcuxpresso.nxp.com/en/select?device=FRDM-KW36.
2. Open the MCUXpresso IDE.
3. Create or select the workspace directory and click the **OK** button.

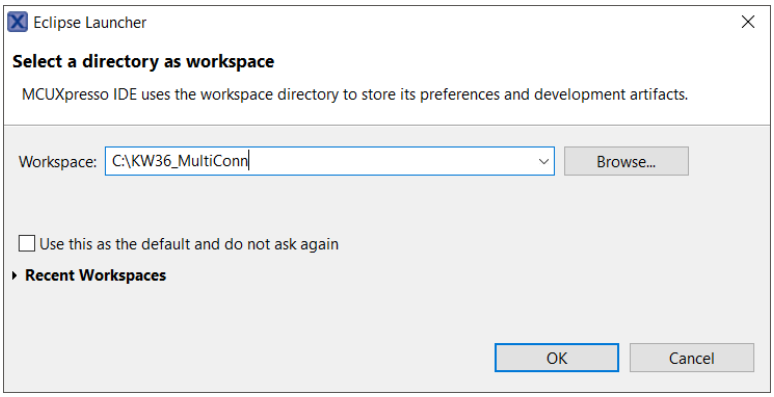

Figure 1. Selecting workspace

4. If there is no previous SDK installed, import the FRDM-KW36 SDK. To install a new SDK in the MCUXpresso IDE, drag and drop the SDK *.zip* file into the **Installed SDKs** view.

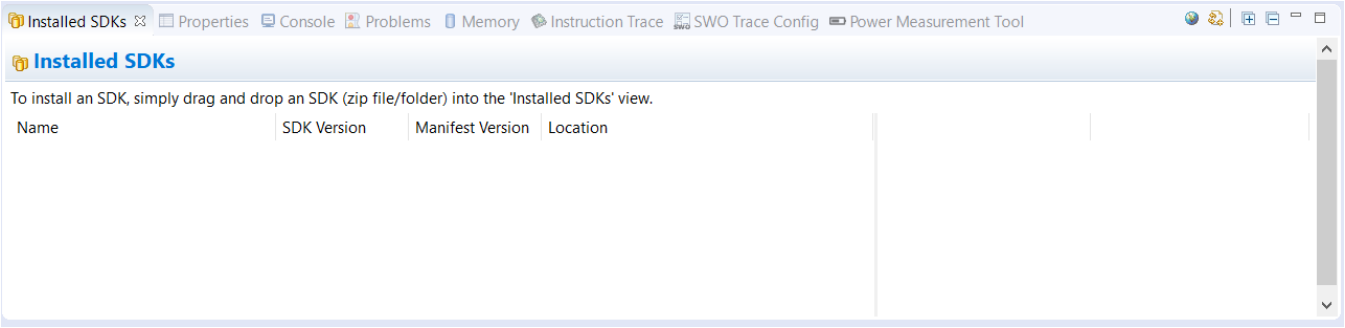

Figure 2. MCUXpresso “Installed SDKs” view

5. When installed, the MCUXpresso IDE looks as Figure 3.

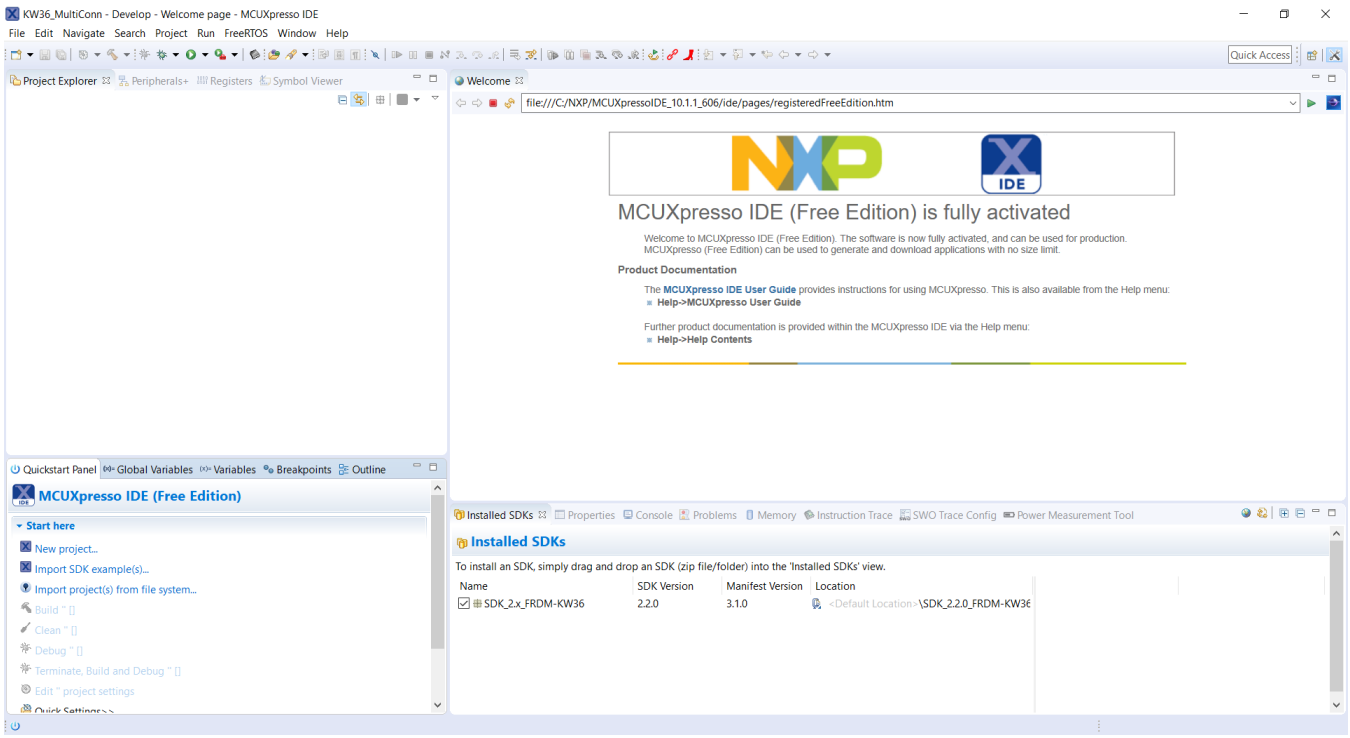

Figure 3. MCUXpresso IDE main screen

3.2. Importing SDK example

1. In the **Quickstart Panel** tab, click the **Import SDK example(s)…** option.

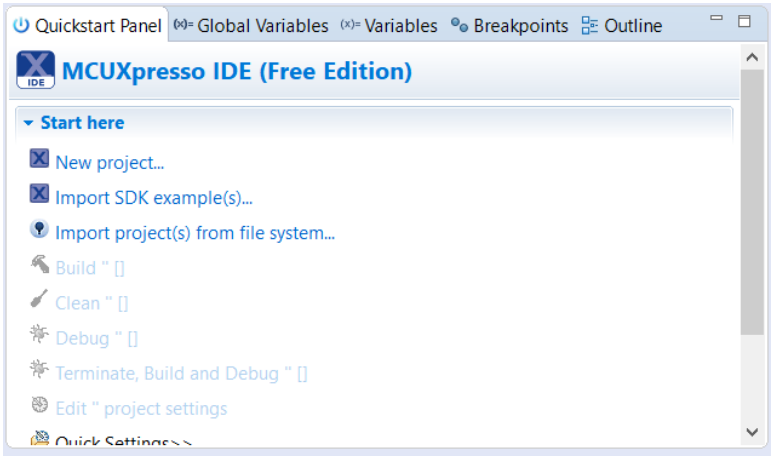

Figure 4. "Quickstart Panel" tab

2. Select the **frdmkw36 SDK** in the **Available boards** screen and click the **Next >** button.

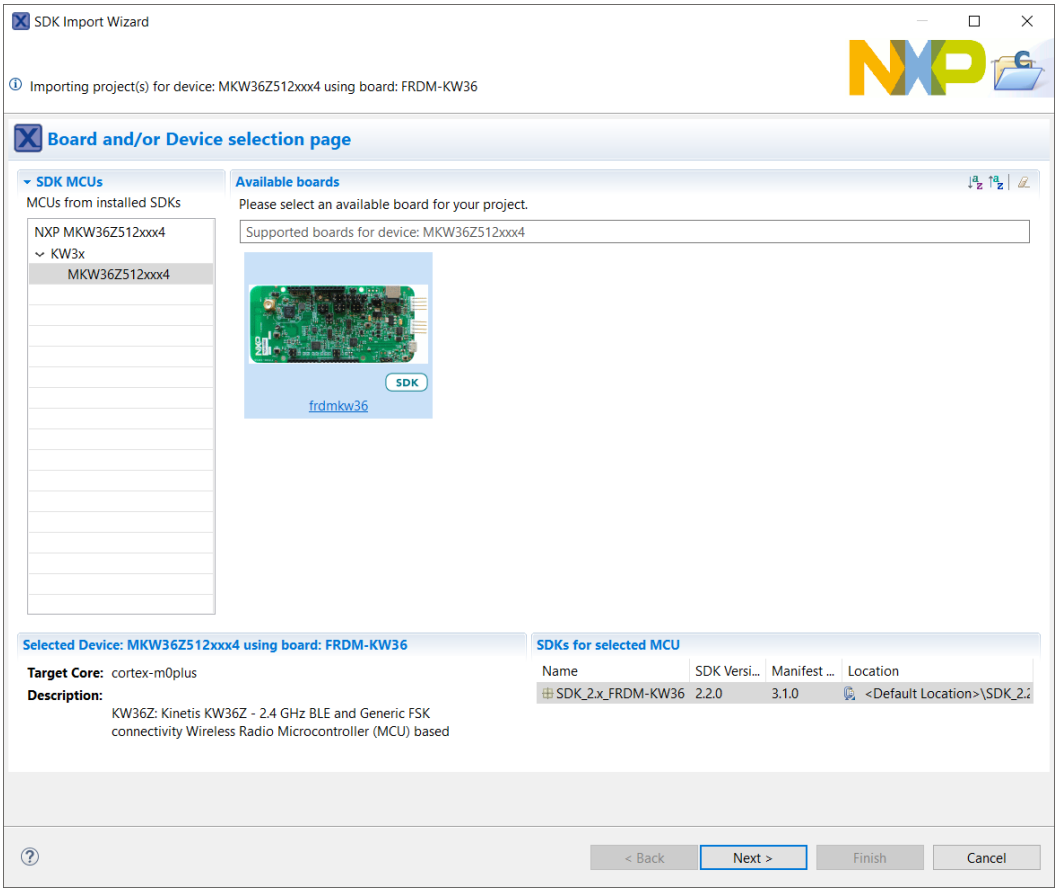

Figure 5. SDK import wizard

3. In the **Examples view**, expand the *wireless_examples* folder, expand the *bluetooth* subfolder, and then the *temp_sens* subfolder. Tick the **freertos** option and click the **Finish** button.

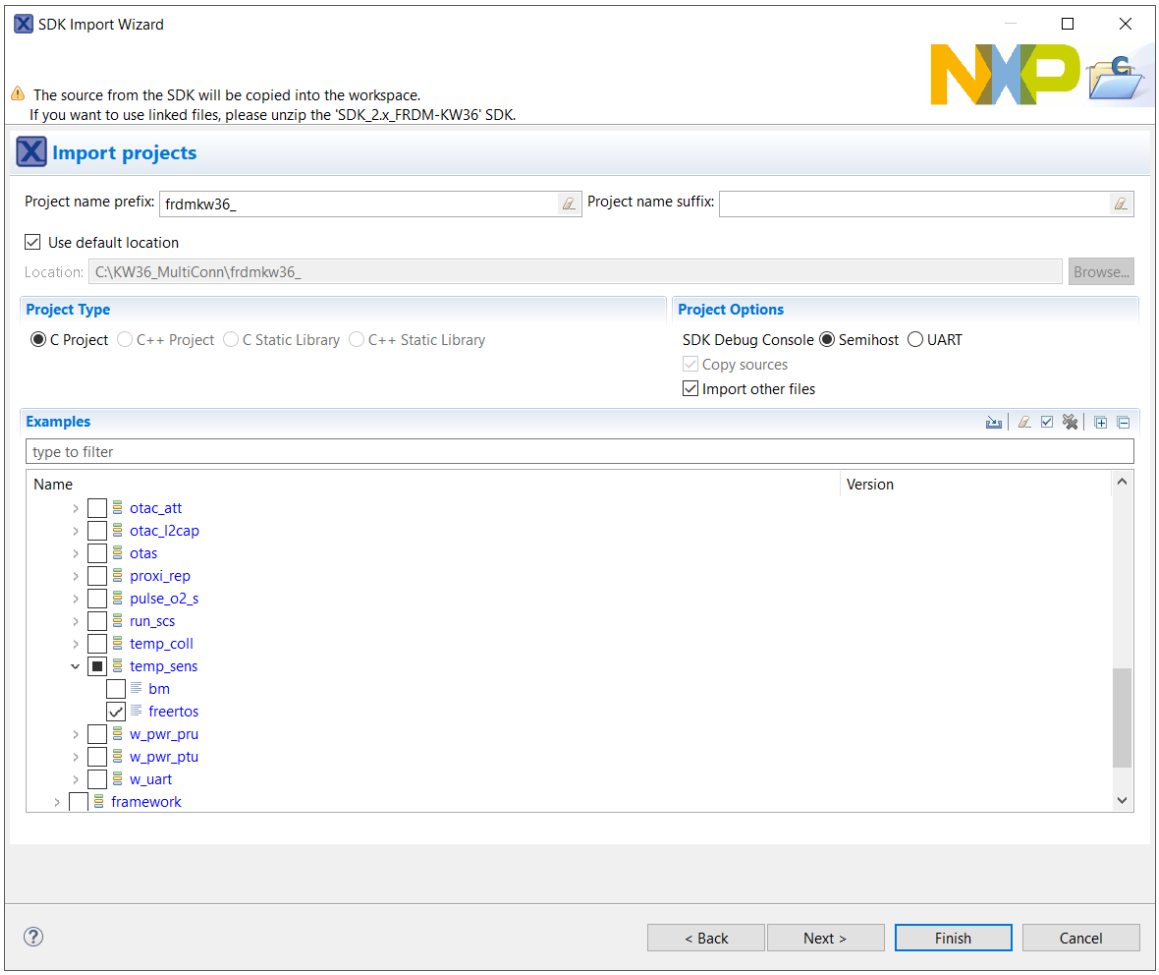

Figure 6. Importing Temperature Sensor project to workspace

4. Adding multiple connection support

When the Temperature Sensor application is imported to the MCUXpresso IDE, the following files must be modified to enable multiple connections*: app_preinclude.h*, *temperature_service.c*, *temperature_interface.h*, and *temperature_sensor.c*.

4.1. Modifying app_preinclude.h file

1. In the **Project Explorer** view, expand the Temperature Sensor project and locate the *app_preinclude.h* file in the source folder.

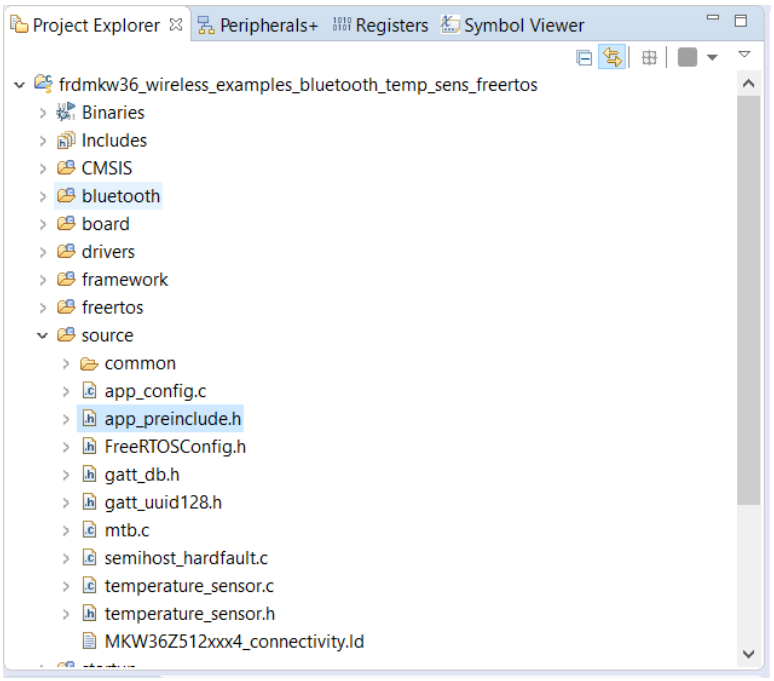

Figure 7. app_preinclude.h file

2. Add the following define. This define determines the maximum number of simultaneous connections. The maximum number of connections is eight.

```
/*! *********************************************************************************
 *     App Configuration
 ********************************************************************************** */
/*! Number of connections supported by the application */
#define gAppMaxConnections_c    8
```

3. Locate the **gTmrStackTimers_c** define and modify it as shown below. This define must be increased by one for each device to be connected with pairing.

```
#define gTmrStackTimers_c                (6 + gAppMaxConnections_c)
```

4. If debug is required, modify the following macro to disable the usage of the low-power mode.

```
/* Enable/Disable PowerDown functionality in PwrLib */
#define cPWR_UsePowerDownMode            0
```

4.2. Modifying temperature_service.c file

1. In the **Project Explorer** view, expand the Temperature Sensor project and locate the *temperature_sensor.c* file in the *bluetooth/profiles/temperature* folder.

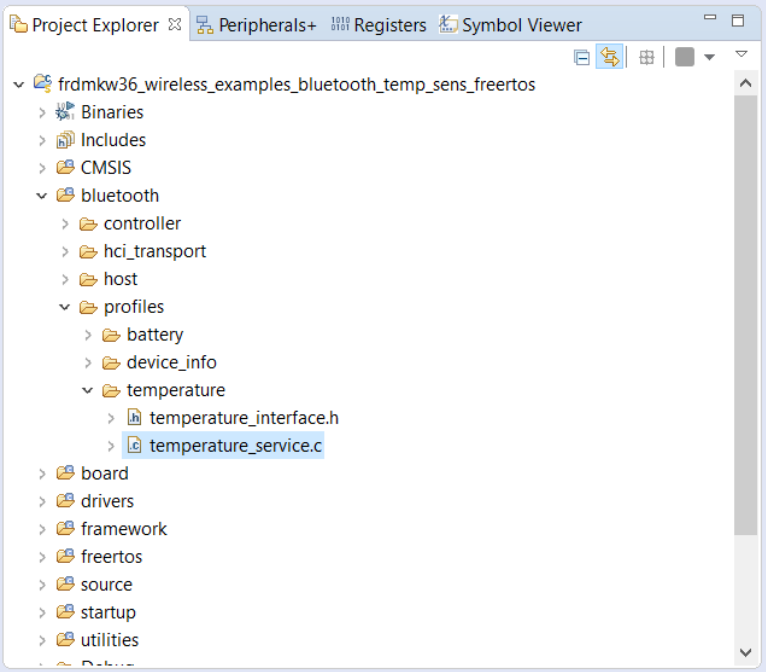

Figure 8. temperature_service.c file

2. Locate the `static deviceId_t mTms_SubscribedClientId` declaration and comment the line.

```
/*! Temperature Service - Subscribed Client*/
//static deviceId_t mTms_SubscribedClientId;
```

3. Locate the `Hts_SendTemperatureMeasurementNotification` function declaration and modify it as follows:

```
static void Hts_SendTemperatureMeasurementNotification(tmsConfig_t *pServiceConfig,
uint16_t handle);
```

4. Go to the `Tms_Start` function and modify it as follows:

```
bleResult_t Tms_Start (tmsConfig_t *pServiceConfig)
{
       uint8_t mClientId = 0;

    /* reset all slots for valid subscribers */
    for(mClientId = 0; mClientId < pServiceConfig->validSubscriberListSize;
mClientId++)
    {
        pServiceConfig->aValidSubscriberList[mClientId] = FALSE;
    }

    return Tms_RecordTemperatureMeasurement(pServiceConfig);
}
```

5. Go to the **Tms_Stop** function and modify it as follows:

```
bleResult_t Tms_Stop (tmsConfig_t *pServiceConfig)
{
    uint8_t mClientId = 0;

    /* reset all slots for valid subscribers */
    for(mClientId = 0; mClientId < pServiceConfig->validSubscriberListSize;
mClientId++)
    {
        pServiceConfig->aValidSubscriberList[mClientId] = FALSE;
    }

    return gBleSuccess_c;

}
```

6. Go to the **Tms_Subscribe** function and modify it as follows:

```
bleResult_t Tms_Subscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId)
{
    if(deviceId >= pServiceConfig->validSubscriberListSize)
    {
        return gBleInvalidParameter_c;
    }

    pServiceConfig->aValidSubscriberList[deviceId] = TRUE;

    return gBleSuccess_c;
}
```

7. Go to the **Tms_Unsubscribe** function and modify it as follows:

```
bleResult_t Tms_Unsubscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId)
{
    if(deviceId >= pServiceConfig->validSubscriberListSize)
    {
        return gBleInvalidParameter_c;
    }

    pServiceConfig->aValidSubscriberList[deviceId] = FALSE;
    return gBleSuccess_c;
}
```

8. Go to the **Tms_RecordTemperatureMeasurement** function and modify it as follows:

```
bleResult_t Tms_RecordTemperatureMeasurement (tmsConfig_t *pServiceConfig)
{
    uint16_t  handle;
    bleResult_t result;
    bleUuid_t uuid = Uuid16(gBleSig_Temperature_d);

    /* Get handle of Temperature characteristic */
    result = GattDb_FindCharValueHandleInService(pServiceConfig->serviceHandle,
gBleUuidType16_c, &uuid, &handle);

    if (result != gBleSuccess_c)
        return result;

    /* Update characteristic value */
    result = GattDb_WriteAttribute(handle, sizeof(uint16_t), (uint8_t*)&pServiceConfig-
>temperature);

    if (result != gBleSuccess_c)
        return result;
```

```
    Hts_SendTemperatureMeasurementNotification(pServiceConfig, handle);

    return gBleSuccess_c;
}
```

9. Go to the **Hts_SendTemperatureMeasurementNotification** function and modify it as follows:

```
static void Hts_SendTemperatureMeasurementNotification(tmsConfig_t *pServiceConfig,
uint16_t handle)
{
    uint16_t  hCccd;
    bool_t isNotificationActive;
    uint8_t   mClientId = 0;

    /* Get handle of CCCD */
    if (GattDb_FindCccdHandleForCharValueHandle(handle, &hCccd) != gBleSuccess_c)
        return;
    for(mClientId = 0; mClientId < pServiceConfig->validSubscriberListSize;
mClientId++)
    {
       if(pServiceConfig->aValidSubscriberList[mClientId])
       {
              if (gBleSuccess_c == Gap_CheckNotificationStatus
                           (mClientId, hCccd, &isNotificationActive) &&
                                  TRUE == isNotificationActive)
              {
                     GattServer_SendNotification(mClientId, handle);
              }
       }
    }
}
```

4.3. Modifying temperature_interface.h file

1. In the **Project Explorer** view, expand the Temperature Sensor project and locate the *temperature_interface.h* file in *the bluetooth/profiles/temperature* folder.

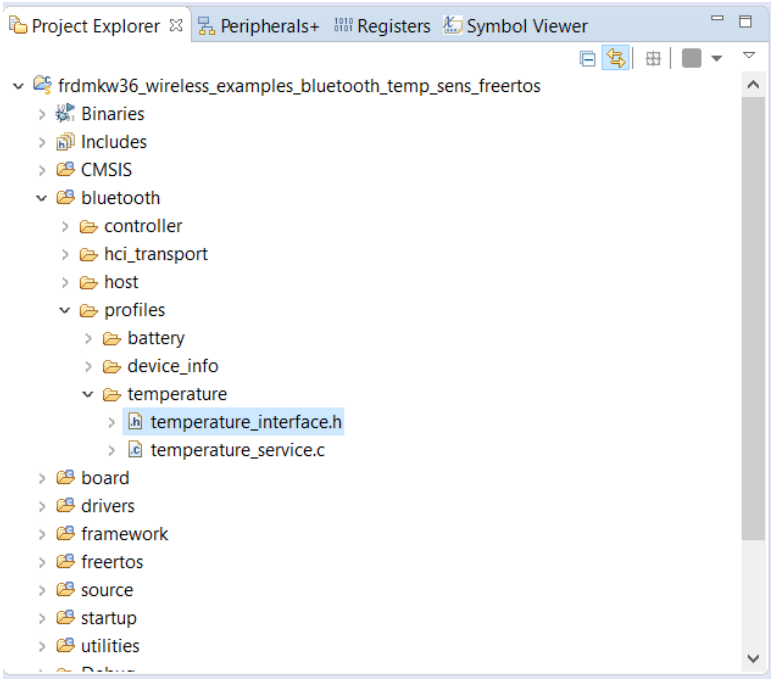

Figure 9. temperature_interface.h file

2. Locate the structure and modify it as follows:

```
/*! Temperature Service - Configuration */
typedef struct tmsConfig_tag
{
    uint16_t    serviceHandle;
    int16_t     temperature;
    bool_t*     aValidSubscriberList;
    uint8_t     validSubscriberListSize;

} tmsConfig_t;
```

3. Locate the **Tms_Subscribe** function declaration and modify it as follows:

```
bleResult_t Tms_Subscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId);
```

4. Locate the **Tms_Unsubscribe** function and modify it as follows:

```
bleResult_t Tms_Unsubscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId);
```

5. Locate the **Tms_RecordTemperatureMeasurement** function and modify it as follows:

```
bleResult_t Tms_RecordTemperatureMeasurement (tmsConfig_t *pServiceConfig);
```

4.4. Modifying temperature_sensor.c file

1. In the **Project Explorer** view, expand the Temperature Sensor project and locate the *temperature_sensor.c* file in the *source* folder.

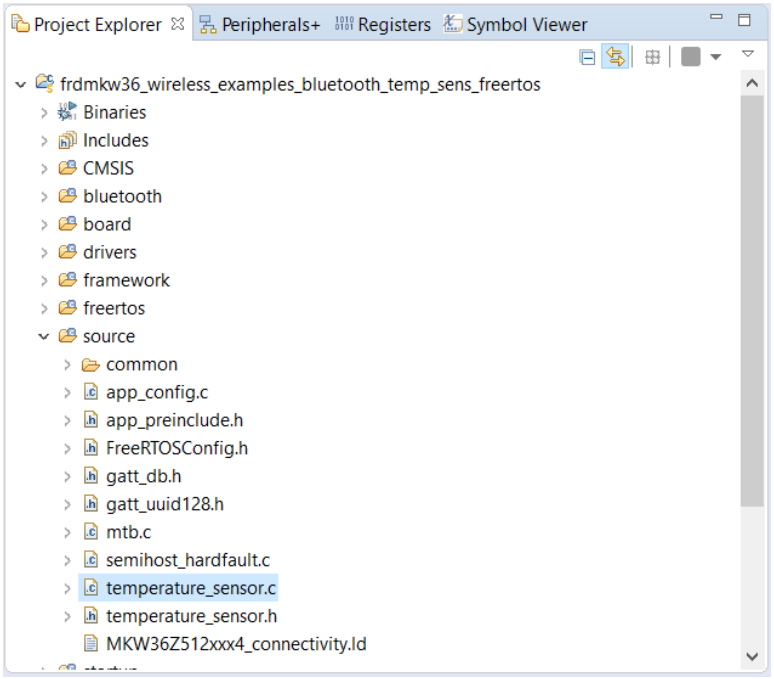

Figure 10. temperature_sensor.c file

2. Locate the **mPeerDeviceId** variable declaration and modify it as follows:

```
static deviceId_t  mPeerDeviceId[gAppMaxConnections_c] = {gInvalidDeviceId_c};
```

3. Create a global variable to be used to track the number of active devices. This variable can be placed below the **mPeerDeviceId[gAppMaxConnections_c]** declaration.

```
uint8_t mActiveConnections = 0;
```

4. Create a global variable to be used as a valid client list. This variable can be placed below the **basValidClientList[gAppMaxConnections_c]** declaration.

```
static bool_t tmsValidClientList[gAppMaxConnections_c] = {FALSE};
```

5. Locate the **tmsServiceConfig** variable and modify it as follows:

```
static tmsConfig_t tmsServiceConfig = {service_temperature, 0, tmsValidClientList,
gAppMaxConnections_c};
```

6. Locate the declaration of the **DisconnectTimerCallback** function and comment the lines as shown below. Find the declaration within the `#if - #endif` preprocessor directive.

```
/* Timer Callbacks */
#if (cPWR_UsePowerDownMode)
static void AdvertisingTimerCallback (void *);
//static void DisconnectTimerCallback(void* );
#endif
```

7. Go to the **BleApp_Start** function and modify the `if` statement as follows:

```
if (mActiveConnections < gAppMaxConnections_c)
```

8. Go to the **BleApp_Config** function, locate the **tmsServiceConfig.initialTemperature** value assignation, and modify it as follows:

```
tmsServiceConfig.temperature = 100 * BOARD_GetTemperature();
```

9. Go to the **BleApp_AdvertisingCallback** function, locate the **#ifdef MULTICORE_HOST** preprocessor directive, and modify it as follows:

```
#ifdef MULTICORE_HOST
	#if gErpcLowPowerApiServiceIncluded_c
		PWR_ChangeBlackBoxDeepSleepMode(3);
	#endif
#else
	if(mActiveConnections > 0)
	{
		PWR_ChangeDeepSleepMode(1);
	}
	else
	{
		PWR_ChangeDeepSleepMode(3);
	}
#endif
```

10. Go to the **BleApp_ConnectionCallback** function.

    a. At **case gConnEvtConnected_c**, locate and modify the following line as follows:

```
1) mPeerDeviceId[mActiveDevices] = peerDeviceId;
2) Tms_Subscribe(&tmsServiceConfig, peerDeviceId);
```

    b. At **case gConnEvtConnected_c**, add the following line below the **Tms_Subscribe** function call:

```
mActiveConnections++;
```

    c. At **case gConnEvtDisconnected_c**, locate and modify the following line as follows:

```
1) mPeerDeviceId[peerDeviceId] = gInvalidDeviceId_c;
2) Tms_Unsubscribe(&tmsServiceConfig, peerDeviceId);
```

    d. At **case gConnEvtDisconnected_c**, add the following line below the **Tms_Unsubscribe** function call:

```
mActiveConnections--;
```

    e. At **case gConnEvtDisconnected_c**, locate the **#if (cPWR_UsePowerDownMode)** preprocessor directive and modify it as follows:

```
#if (cPWR_UsePowerDownMode)
            /* Go to sleep */
            #ifdef MULTICORE_HOST
                #if gErpcLowPowerApiServiceIncluded_c
                    PWR_ChangeBlackBoxDeepSleepMode(3);
                #endif
            #else
                if(mActiveConnections > 0)
                {
                        PWR_ChangeDeepSleepMode(1);
                }
```

```
                    else
                    {
                        PWR_ChangeDeepSleepMode(3);
                    }
                #endif
                Led1Off();
        #else
                LED_TurnOffAllLeds();
                LED_StartFlash(LED_ALL);
        #endif
```

11. Go to the **DisconnectTimerCallback** function and comment the lines as follows:

```
/*
static void DisconnectTimerCallback(void* pParam)
{
    if (mPeerInformation.deviceId != gInvalidDeviceId_c)
    {
        Gap_Disconnect(mPeerDeviceId);
    }
}
*/
```

12. Go to the **BleApp_SendTemperature** function and modify it as follows:

```
static void BleApp_SendTemperature(void)
{
    TMR_StopTimer(appTimerId);

    /* Update with initial temperature */
    tmsServiceConfig.temperature = BOARD_GetTemperature() * 100;
    Tms_RecordTemperatureMeasurement(&tmsServiceConfig);

/*
#if (cPWR_UsePowerDownMode)
     Start Sleep After Data timer
    TMR_StartLowPowerTimer(appTimerId,
                           gTmrLowPowerSecondTimer_c,
                           TmrSeconds(gGoToSleepAfterDataTime_c),
                           DisconnectTimerCallback, NULL);
#endif
*/

}
```

5. Testing peripheral device with multiple connections

The Temperature Collector demo application is needed together with the Temperature Sensor demo application to demonstrate the functionality of multiple connections. The following steps show how to generate two (or more) central devices enabled with the Temperature Collector to test multiple connections in a peripheral device.

5.1. Importing Temperature Collector example

1. See Section 3.2, "Importing SDK example" and follow the steps described there. In step number 3, select **temp_coll** instead of **temp_sens**. This imports the Temperature Collector demo application into the workspace.

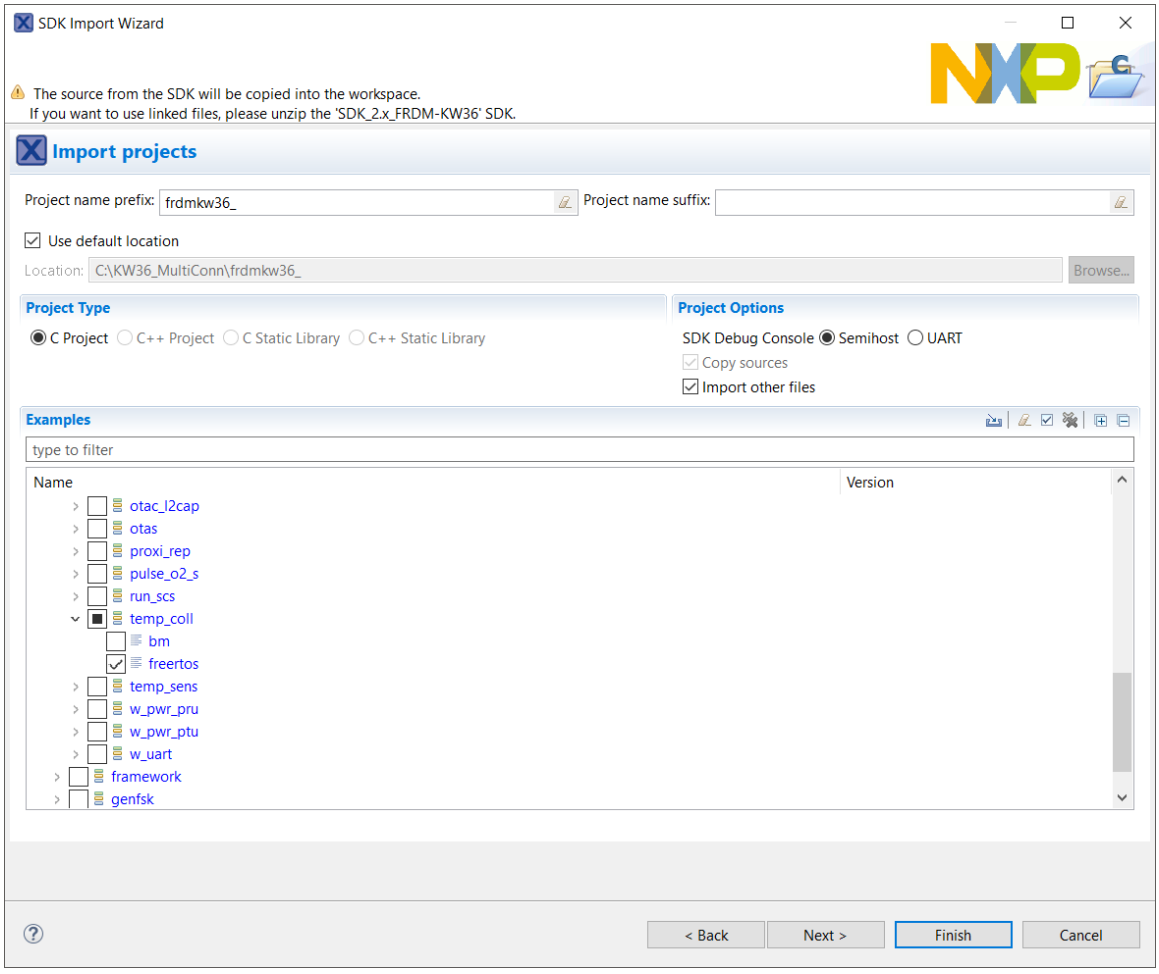

Figure 11. Importing Temperature Collector project to workspace

2. The Temperature Collector application has the low-power mode enabled by default and, like the Temperature Sensor application, it has a timer that disconnects the device when the temperature is reported. To avoid disconnection, perform these steps:

    a) Open the Temperature Collector application's *app_preinclude.h* file and modify the

following macro to disable the low-power mode:

```
/* Enable/Disable PowerDown functionality in PwrLib */
#define cPWR_UsePowerDownMode           0
```

b) Open the Temperature Collector application's *temperature_collector.c* file, navigate to the `BleApp_GattNotificationCallback` function, and modify the following lines:

```
/*
#if (cPWR_UsePowerDownMode)
         Restart Wait For Data timer
        TMR_StartLowPowerTimer(mAppTimerId,
                       gTmrLowPowerSecondTimer_c,
                       TmrSeconds(gWaitForDataTime_c),
                       DisconnectTimerCallback, &serverDeviceId);
#endif
*/
```

3. If debugging is required, open the Temperature Collector application's *app_preinclude.h* file and modify the following macro to disable the low-power mode.

```
/* Enable/Disable PowerDown functionality in PwrLib */
#define cPWR_UsePowerDownMode           0
```

5.2. Building and downloading projects

1. Select the Temperature Sensor project in the **Project Explorer** view and compile it by clicking the **Build** button in the **Quickstart Panel** view.

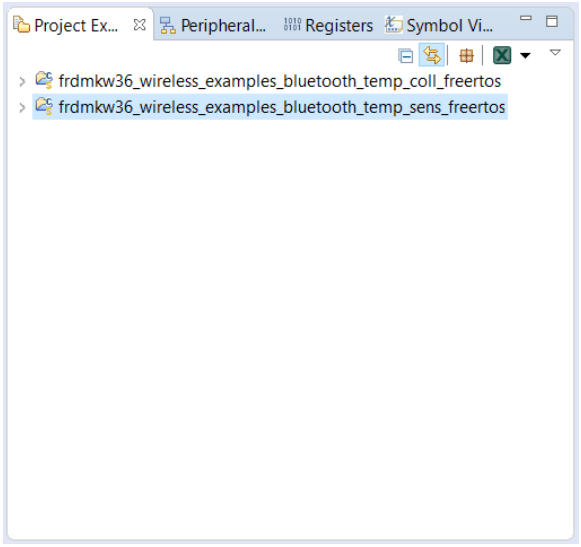

Figure 12. Temperature Sensor project selected

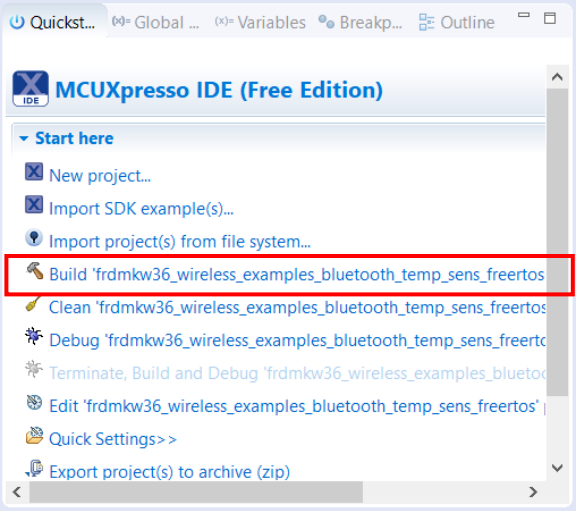

Figure 13. Build project button

2. Connect the FRDM-KW36 module, which acts as the temperature sensor and wait for the drivers to be installed.

3. When the drivers are installed, make sure that the Temperature Sensor project is still selected and download the code to the FRDM-KW36 board by clicking the **Debug** button in the **Quickstart Panel** view.

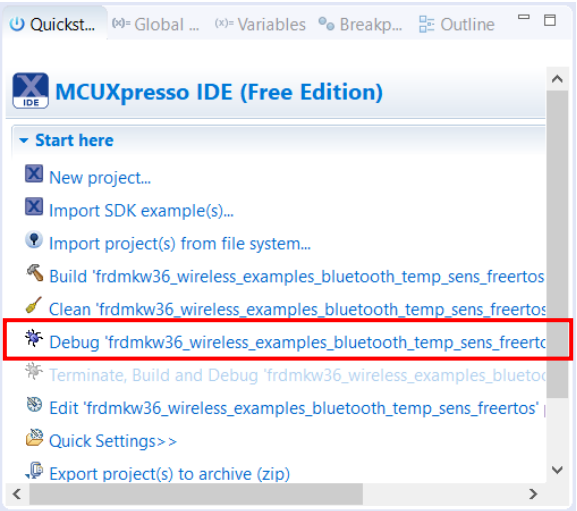

Figure 14. Build project button

4. Stop the debugger by clicking the **Terminate** button and disconnect the board.

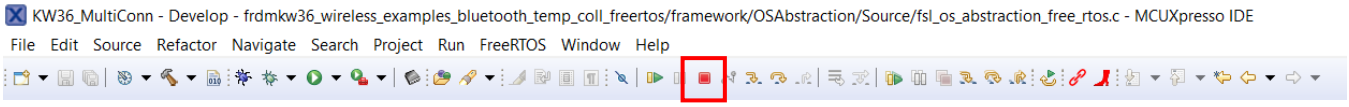

Figure 15. Terminate button

5. Repeat the previous steps for the remaining boards that act as temperature collectors. Make sure that the Temperature Collector project is selected.

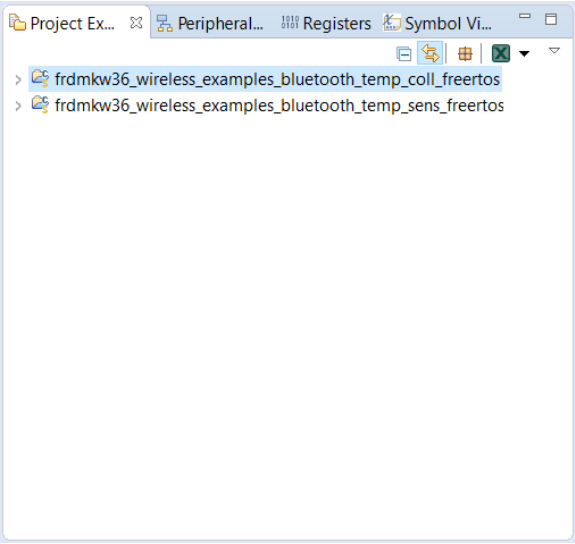

Figure 16. Temperature collector project selected

5.3. Running application

1. Connect the FRDM-KW36 boards flashed with the Temperature Collector application.
2. Launch TeraTerm and open the port assigned to the FRDM-KW36 board.

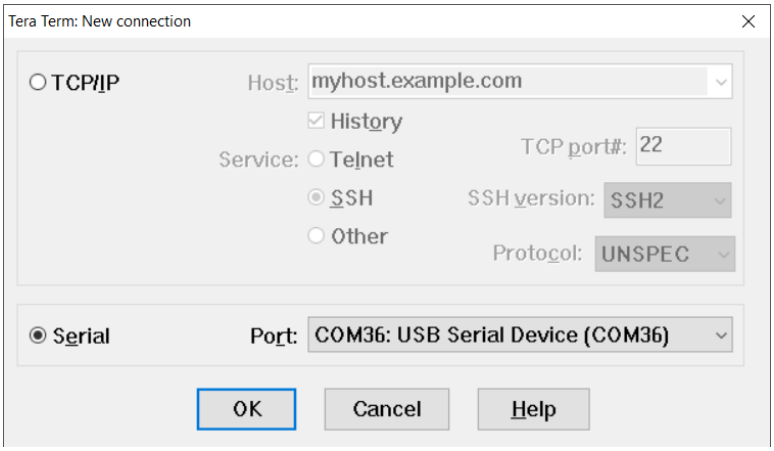

Figure 17. Opening serial port

3. Open the **Setup** menu and select the **Serial port** option. Make sure to configure the settings as in Figure 18.

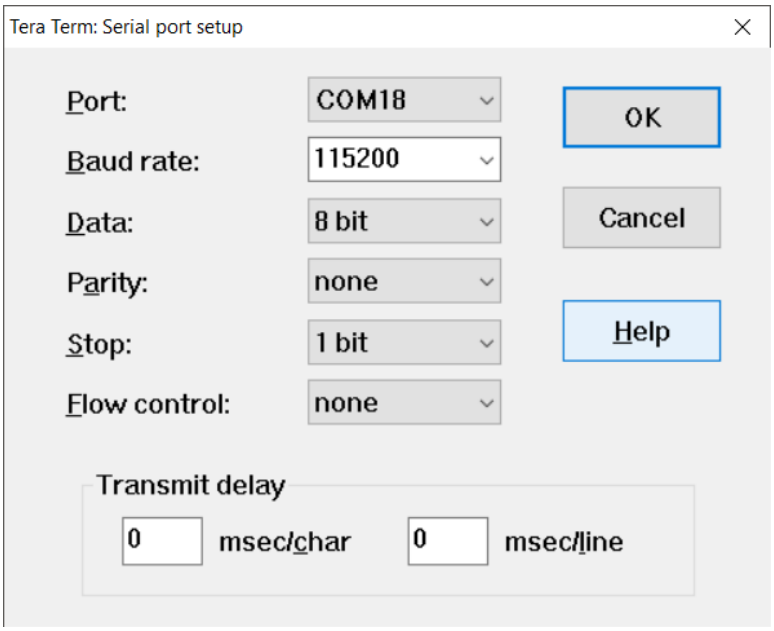

Figure 18. Configuring serial port

4. Press the **Reset** (SW1) button on the FRDM-KW36 board. The Temperature Collector displays the screen shown in Figure 19.

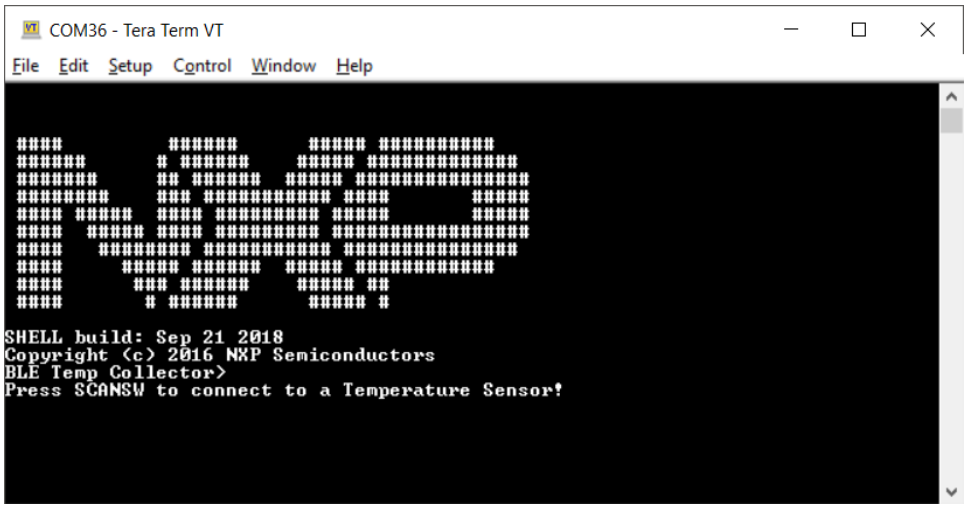

Figure 19. Temperature collector

5. Open the **File** menu and select the **New connection** option. Open the port of the remaining Temperature Collector boards and repeat steps 3 and 4.

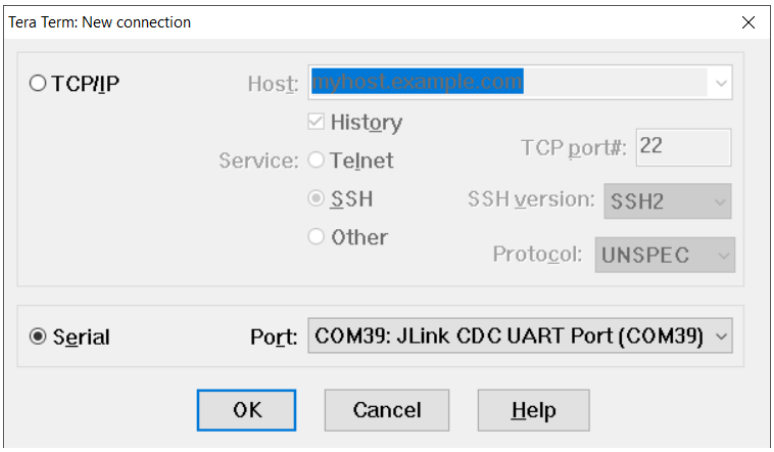

Figure 20. Opening serial port

6. Connect the Temperature Sensor board and press the SW2 button to start advertising. When all the connections are established, the SW2 button can be used to send additional temperature reports.
7. Press the SW2 button on any Temperature Collector board to start scanning. When the Temperature Sensor is connected, the Temperature Collector shows the temperature reported by the sensor.

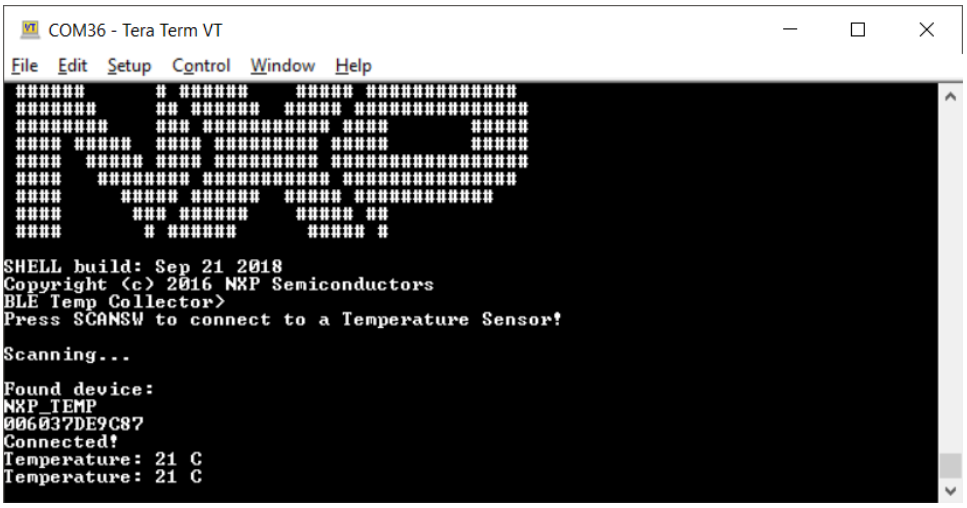

Figure 21. First Temperature Collector connected

8. Repeat steps 6 and 7 with the remaining boards. Each time the Temperature Sensor sends a temperature report, it is shown simultaneously on each Temperature Collector.

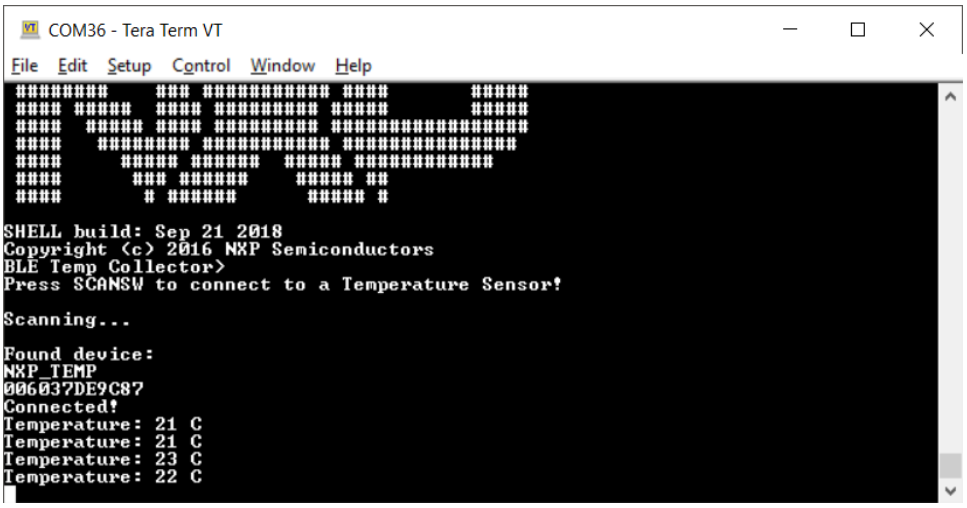

Figure 22. Second Temperature Collector connected

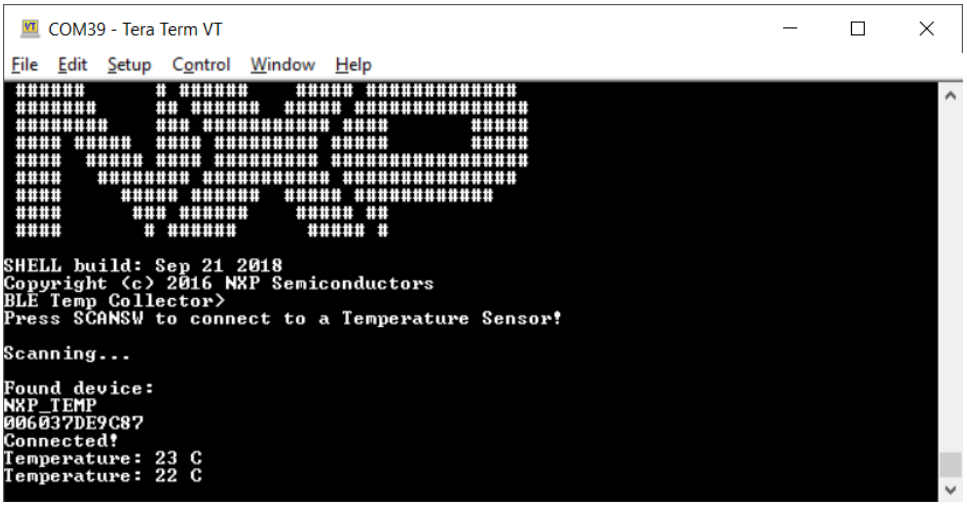

Figure 23. Last two measurements repeated on both collectors

9. If the connection drops, each Temperature Collector shows a disconnection message.

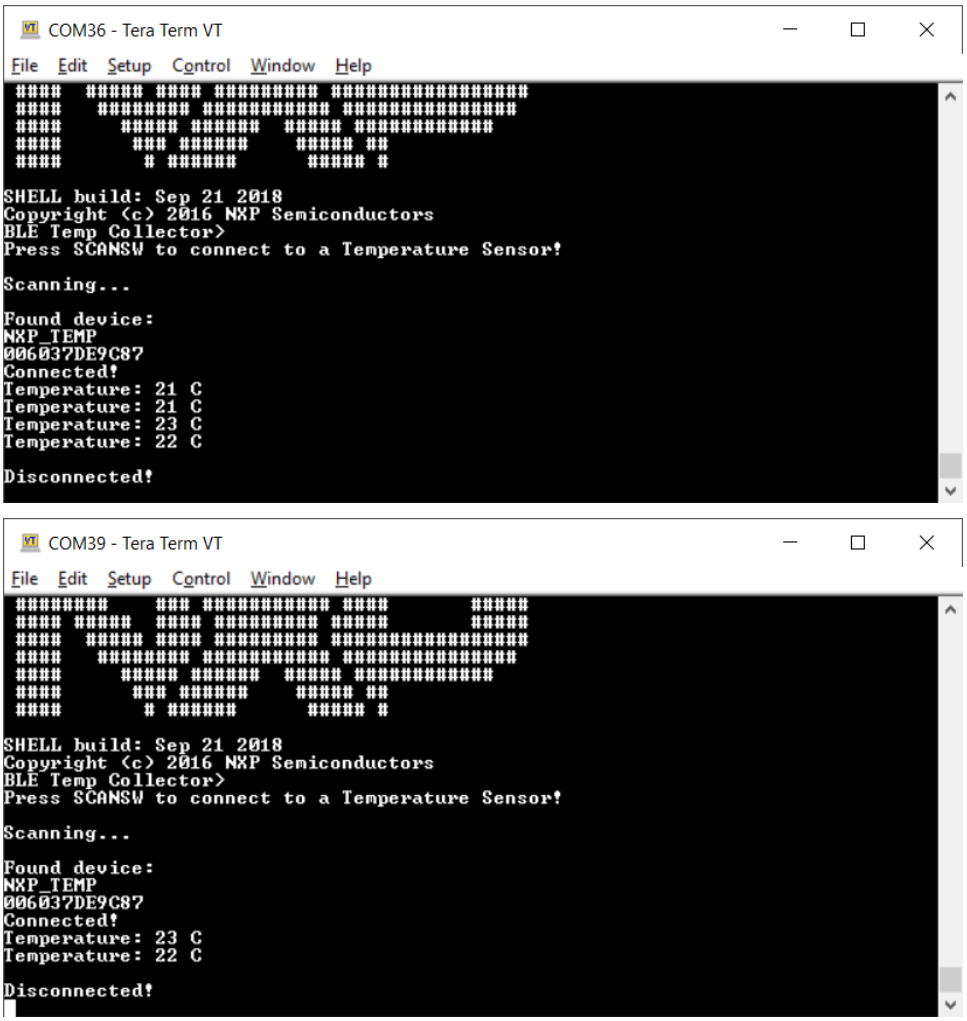

Figure 24. Temperature Sensor disconnected from both Temperature Collectors

How to Reach Us:

Home Page:
www.nxp.com

Web Support:
www.nxp.com/support

Document Number: AN12405
Rev. 0
05/2019