**NXP Semiconductor.**
Application Notes

Document Number: AN12414
Rev. 0, 04/2019

# Multiple Connections in Bluetooth LE Central Device

## 1. Introduction

NXP provides a complete Bluetooth LE solution, that allows to create applications that supports up to 8 simultaneous connections using the KW36/35 SoC. The connections can be configured to be a Central or Peripheral device.

Temperature Collector demo application is used to describes the procedure to enable multiple connections on a Bluetooth LE Central device.

## 2. Prerequisites

The following items are required to complete the implementation of multiple connections on a Central device:

- At least 3 FRDM-KW36
- FRDM-KW36 SDK Package
- MCUXpresso IDE
- Temperature Collector Demo Application
- Temperature Sensor Demo Application
- Tera Term or any serial terminal software

### Contents

# 3. Enabling multiple connections on Bluetooth LE Central device

This section describe the procedure to enable multiple connections using the Temperature Collector application and MCUXpresso IDE.

## 3.1. Creating a workspace and importing the SDK to MCUXpresso IDE

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

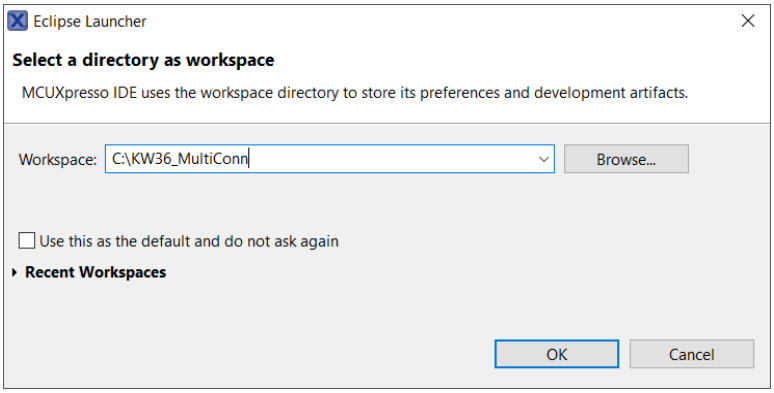


**Figure 1. Select a workspace**

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

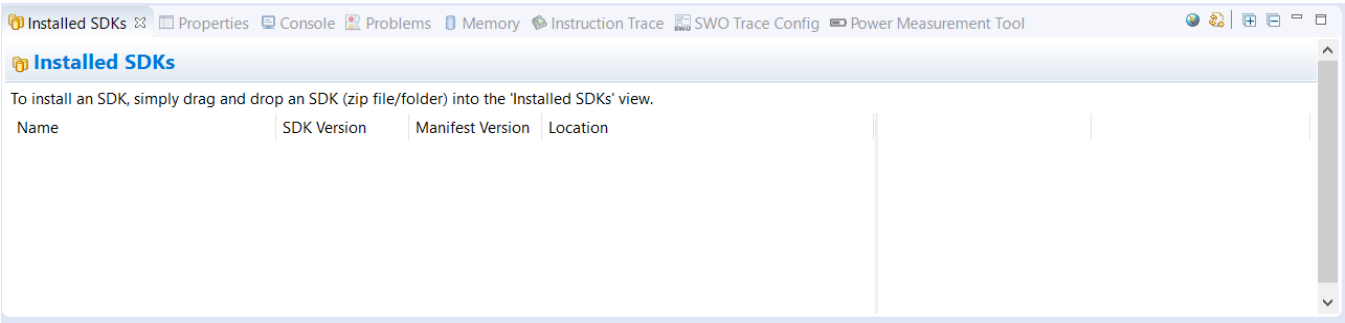


**Figure 2. MCUXpresso Installed SDKs view**

5. Once installed, MCUXpresso IDE looks as Figure 3:

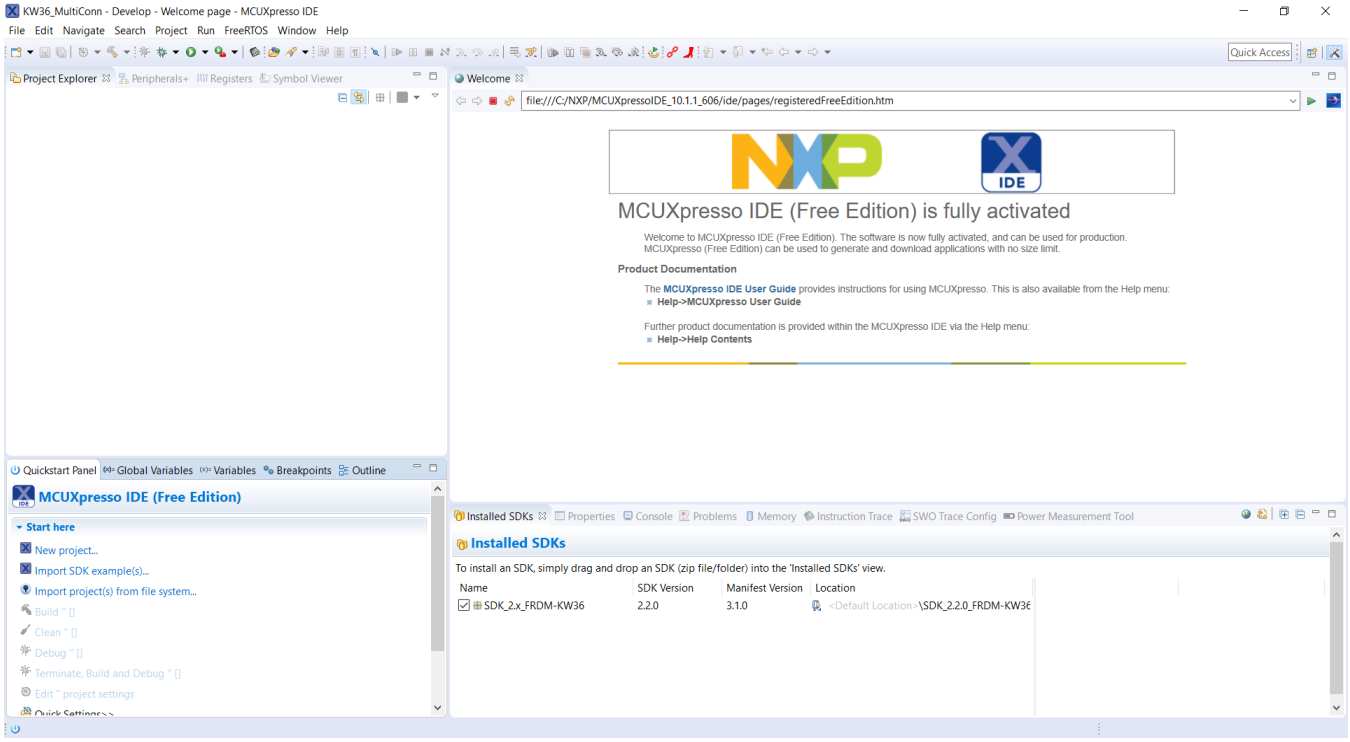


**Figure 3. MCUXpresso IDE main screen**

## 3.2. Importing an SDK example.

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

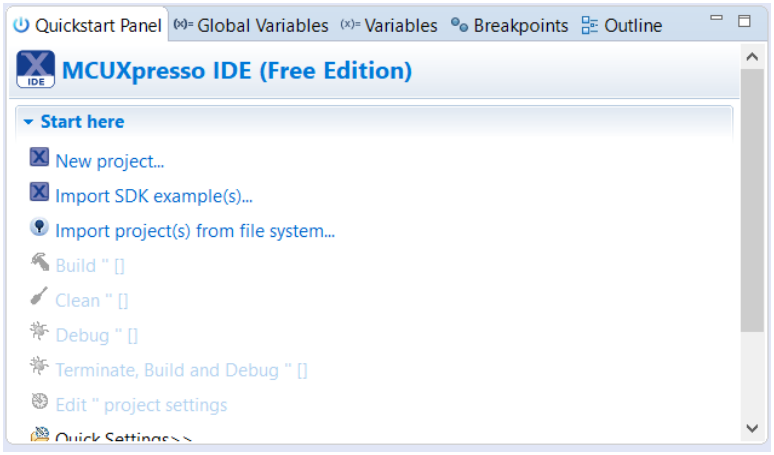


**Figure 4. Quickstart Panel**

2. Select the **frdmkw36 SDK** from available boards and click **Next >** button.

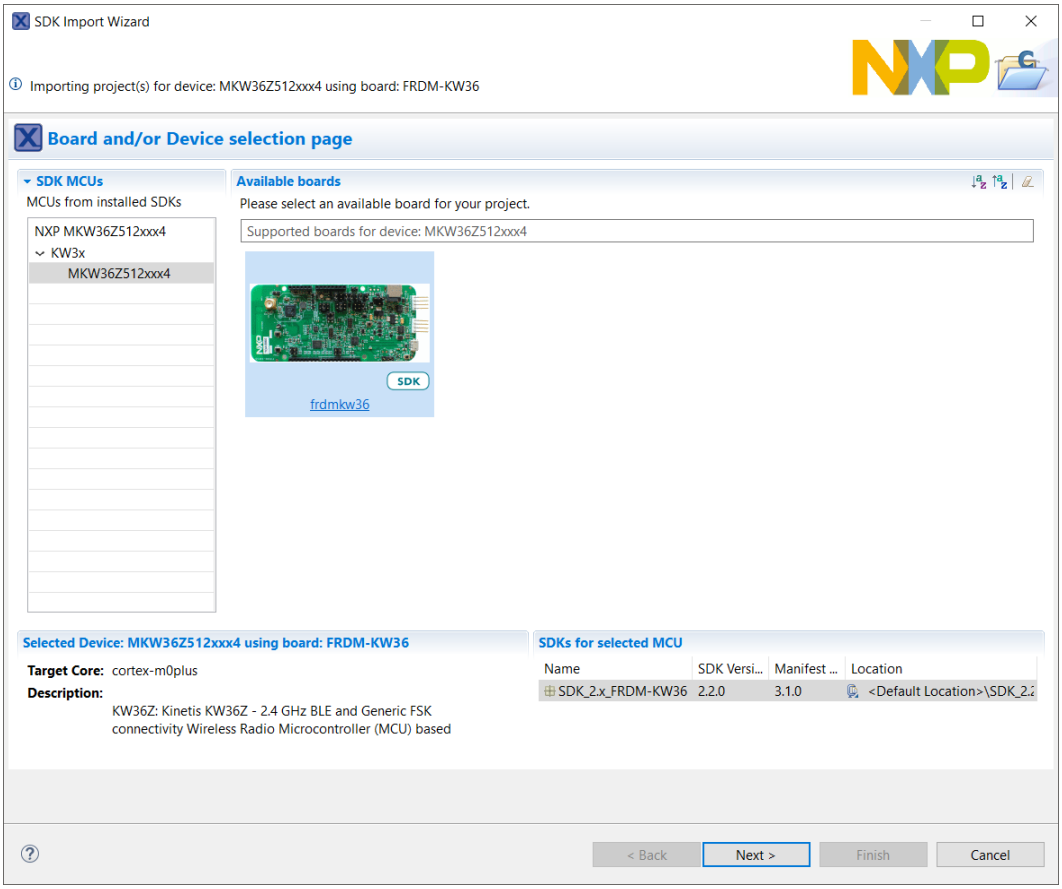


**Figure 5. SDK import wizard**

3. In **Examples view**, expand *wireless_examples* folder, then *bluetooth* subfolder and *temp_coll* subfolder. Select **freertos** and click **Finish.**

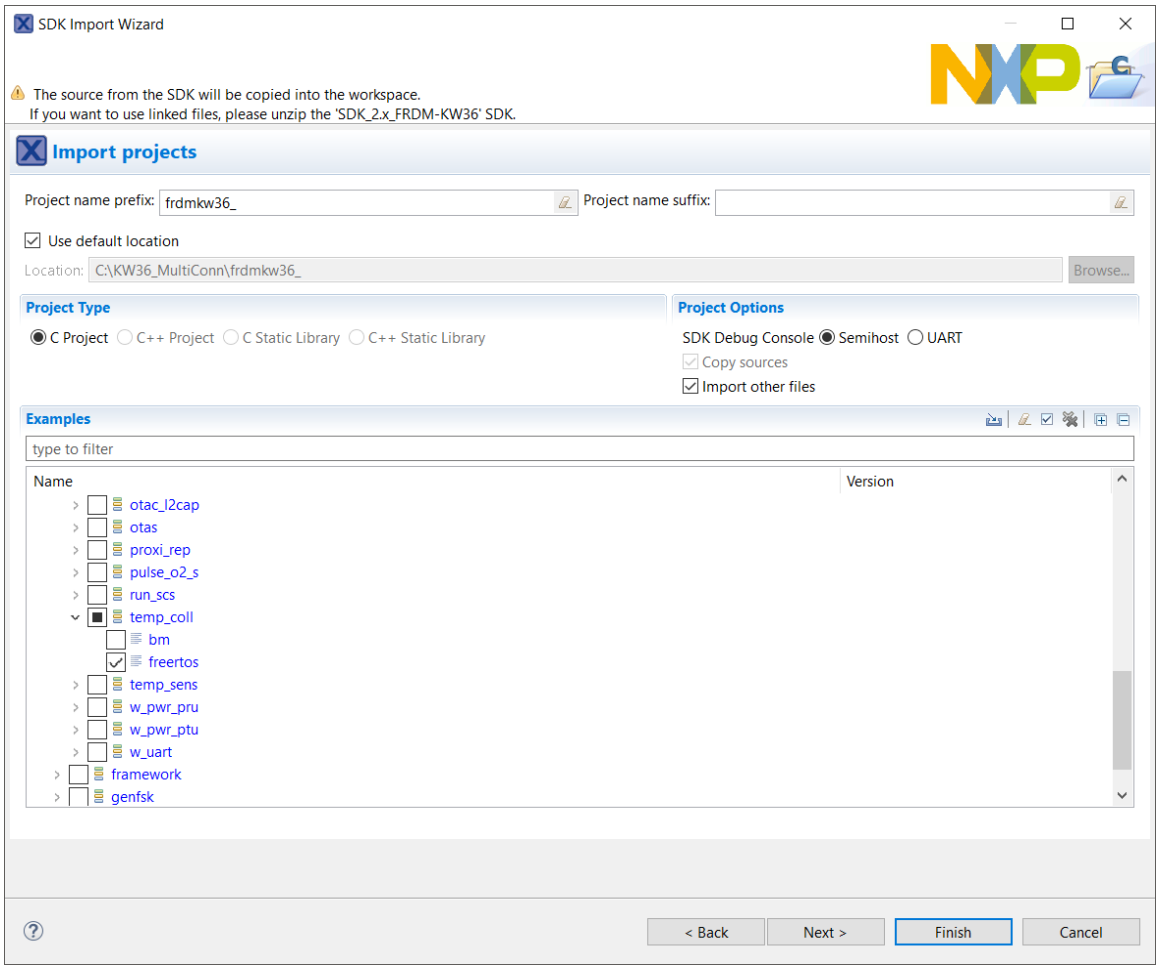


**Figure 6. Importing Temperature Collector project to workspace**

# 4. Adding multiple connections support to Temperature Collector

Once the Temperature Collector application is imported to MCUXpresso IDE, the following files need to be modified to enable multiple connections: *app_preinclude.h* and *temperature_collector.c*.

### 4.1.1. Modifying app_preinclude.h file.

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

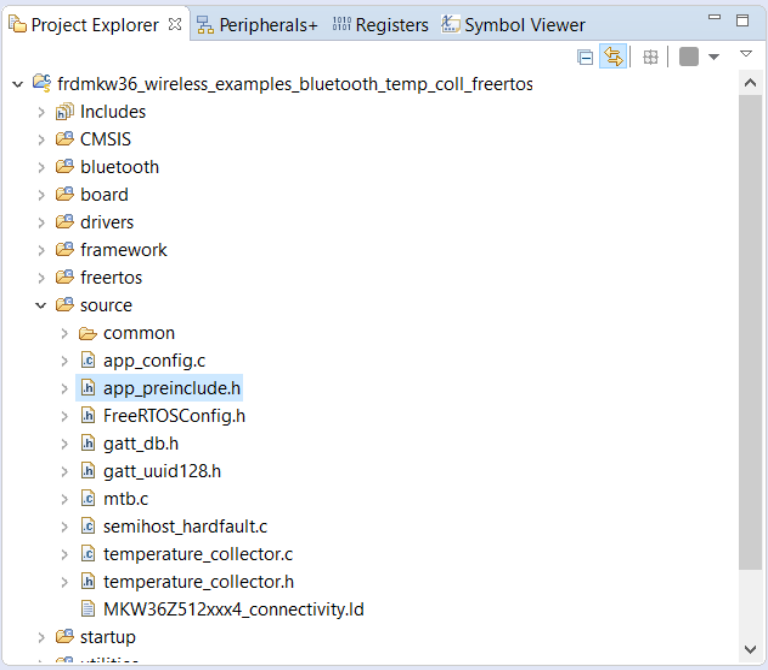


**Figure 7. app_preinclude.h file**

2. Add the following define. This define determines the maximum number of simultaneous connections. The maximum number of connections permitted is 8.

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

3. Locate the `gTmrStackTimers_c` define and modify as below. This define needs to be increased by 1 for each device you want to connect with pairing.

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

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

```
/* Enable/Disable PowerDown functionality in PwrLib */

#define cPWR_UsePowerDownMode           0
```

### 4.1.2. Modifying temperature_collector.c file.

1. In **Project Explorer** view, expand the Temperature Collector project and locate *temperature_collector.c* file at *source* folder.

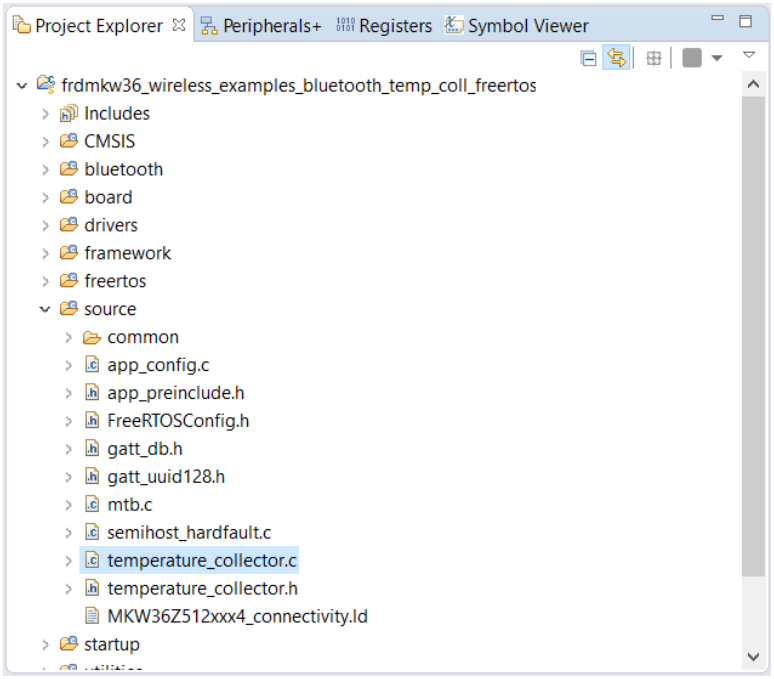


**Figure 8. temperature_collector.c file**

2. Locate the `static appPeerInfo_t mPeerInformation` declaration and modify it as below:

```
static appPeerInfo_t mPeerInformation[gAppMaxConnections_c];
```

3. Create a global variable which will be used as active connections counter. This variable can be placed below `mPeerInformation[gAppMaxConnections_c]` declaration.

```
uint8_t mActiveConnections = 0;
```

4. Locate the following functions declaration and add the `deviceId_t peerDeviceId` parameter as below.

```
1)  static void BleApp_StoreServiceHandles
        (
                        deviceId_t      peerDeviceId,
                        gattService_t   *pService
        );

2)  static void BleApp_StoreDescValues
        (
                        deviceId_t      peerDeviceId,
                        gattAttribute_t     *pDesc
        );

3)  static void BleApp_PrintTemperature
        (
```

```
                        deviceId_t      peerDeviceId,
                        uint16_t temperature
        );
```

4) `static bleResult_t BleApp_ConfigureNotifications(deviceId_t peerDeviceId);`

5. Locate the declaration of **`DisconnectTimerCallback`** function and comment the lines as below. You may find the declaration within an **`#if - #endif`** preprocessor directive.

```
/*
#if (cPWR_UsePowerDownMode)
static void DisconnectTimerCallback(void* pParam);
#endif

*/
```

6. Go to **`BleApp_HandleKeys`** function and modify it as below. This will trigger the manual disconnection of all the connected peripheral devices when low power is disabled.

```
case gKBD_EventLongPB1_c:
{
        for(uint8_t i = 0; i < gAppMaxConnections_c; i++)
        {
                if (mPeerInformation[i].deviceId != gInvalidDeviceId_c)
                        Gap_Disconnect(mPeerInformation[i].deviceId);
        }
        break;

}
```

7. Go to **`BleApp_Config`** function and modify the **`mPeerInformation`** variable initialization as below.

```
/* Initialize private variables */
for(uint8_t i = 0; i < gAppMaxConnections_c; i++)
{
        mPeerInformation[i].appState = mAppIdle_c;

}
```

8. Go to **`BleApp_ScanningCallback`** to function

   a. At **`case`** *`gDeviceScanned_c,`* modify as below.

```
        case gDeviceScanned_c:
        {
                /* Check if the scanned device implements the Temperature Custom Profile */
                if( FALSE == mFoundDeviceToConnect )
                {
                        mFoundDeviceToConnect = CheckScanEvent(&pScanningEvent->eventData.scannedDevice);

                        if (mFoundDeviceToConnect)
                        {
                                /* Set connection parameters and stop scanning. Connect on gScanStateChanged_c. */
                                gConnReqParams.peerAddressType = pScanningEvent->eventData.scannedDevice.addressType;
                                FLib_MemCpy(gConnReqParams.peerAddress,
                                                        pScanningEvent->eventData.scannedDevice.aAddress,
                                                        sizeof(bleDeviceAddress_t));

                                (void)Gap_StopScanning();
        #if gAppUsePrivacy_d
                        gConnReqParams.usePeerIdentityAddress = pScanningEvent-
        >eventData.scannedDevice.advertisingAddressResolved;
```

```
#endif

                }
        }
}
```

b. At `case gScanStateChanged_c,` locate and modify the `#if (cPWR_UsePowerDownMode)` preprocessor directive as below.

```
#if (cPWR_UsePowerDownMode)
                    Led1Off();
                    /* 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
#else
                    LED_StopFlashingAllLeds();
                    Led1Flashing();
                    Led2Flashing();
                    Led3Flashing();
                    Led4Flashing();

#endif
```

9. Go to **`BleApp_ConnectionCallback`** function.

a. At `case gConnEvtConnected_c,` locate and modify the following lines as below.

```
1) mPeerInformation[peerDeviceId].deviceId = peerDeviceId;
2) mPeerInformation[peerDeviceId].isBonded = FALSE;
3) Gap_CheckIfBonded(peerDeviceId, &mPeerInformation[peerDeviceId].isBonded);
4) if ((mPeerInformation[peerDeviceId].isBonded) &&
        (gBleSuccess_c == Gap_LoadCustomPeerInformation(peerDeviceId,
        (void*) &mPeerInformation[peerDeviceId].customInfo, 0, sizeof (appCustomInfo_t))))
5) BleApp_StateMachineHandler(mPeerInformation[peerDeviceId].deviceId, mAppEvt_PeerConnected_c);
```

b. At `case gConnEvtConnected_c,` add the following line below

```
mPeerInformation[peerDeviceId].isBonded = FALSE;
    mActiveConnections++;
```

c. At `case gConnEvtDisconnected_c,` locate and modify the following lines as below.

```
1) mPeerInformation[peerDeviceId].deviceId = gInvalidDeviceId_c;
2) mPeerInformation[peerDeviceId].appState = mAppIdle_c;
```

d. At `case gConnEvtDisconnected_c,` add the following line below `mPeerInformation[peerDeviceId].appState = mAppIdle_c;`

```
mActiveConnections--;
```

e. At `case gConnEvtDisconnected_c,` locate the `#if (cPWR_UsePowerDownMode)` preprocessor directive and modify as below.

```
#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
```

f. At `case gConnEvtPairingComplete_c,` locate and modify the following line as below.

```
BleApp_StateMachineHandler(mPeerInformation[peerDeviceId].deviceId, mAppEvt_PairingComplete_c);
```

g. At `case gConnEvtEncryptionChanged_c,` locate and modify the following line as below.

```
if( gBleSuccess_c != BleApp_ConfigureNotifications(peerDeviceId) )
```

10. Go to **`BleApp_ServiceDiscoveryCallback`** function. At `case gServiceDiscovered_c,` modify as below.

```
case gServiceDiscovered_c:
{
        BleApp_StoreServiceHandles(peerDeviceId, pEvent->eventData.pService);
}
break;
```

11. Go to **`BleApp_StoreServiceHandles`** function.

a. Add `deviceId_t peerDeviceId` in the function arguments.

```
static void BleApp_StoreServiceHandles (deviceId_t peerDeviceId, gattService_t *pService)
```

b. Modify **`mPeerInformation.customInfo.tempClientConfig.hService`** as below.

```
/* Found Temperature Service */
mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hService = pService->startHandle;
```

c. Modify the **mPeerInformation.customInfo.tempClientConfig.hTemperature** as below.

```
/* Found Temperature Char */
mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTemperature = pService->
aCharacteristics[i].value.handle;
```

d. At **case** gBleSig_CharPresFormatDescriptor_d, modify as below.

```
case gBleSig_CharPresFormatDescriptor_d:
{
        mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTempDesc = pService->
aCharacteristics[i].aDescriptors[j].handle;
        break;

}
```

e. At **case** *gBleSig_CCCD_d,* modify as below.

```
case gBleSig_CCCD_d:
{
        mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTempCccd = pService->
aCharacteristics[i].aDescriptors[j].handle;
        break;

}
```

12. Go to **BleApp_StoreDescValues** function and modify as below.

```
static void BleApp_StoreDescValues(deviceId_t peerDeviceId, gattAttribute_t *pDesc)
{
    if (pDesc->handle == mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTempDesc)
    {
        /* Store temperature format*/
        FLib_MemCpy(&mPeerInformation[peerDeviceId].customInfo.tempClientConfig.tempFormat,
                    pDesc->paValue,
                    pDesc->valueLength);
    }

}
```

13. Go to **BleApp_PrintTemperature** function and modify as below.

```
static void BleApp_PrintTemperature(deviceId_t peerDeviceId, uint16_t temperature)
{
    shell_write("Temperature: ");
    /*
     * The following lines will print the id of the device reporting the temperature.
     * Then will print a blank space and the temperature. I.E: Temperature: 0 25 C
     */
    shell_writeDec(peerDeviceId);
    shell_write(" ");
```

```
    shell_writeDec(temperature/100);
    if (mPeerInformation[peerDeviceId].customInfo.tempClientConfig.tempFormat.unitUuid16 == 0x272F)
    {
        shell_write(" C\r\n");
    }
    else
    {
        shell_write("\r\n");
    }

}
```

14. Go to **`BleApp_GattClientCallback`** function and modify the **`BleApp_StoreDescValues`** function call as below.

```
BleApp_StoreDescValues(serverDeviceId, mpCharProcBuffer);
```

15. Go to **`BleApp_GattNotificationCallback`** function and modify as below.

```
static void BleApp_GattNotificationCallback
(
    deviceId_t  serverDeviceId,
    uint16_t    characteristicValueHandle,
    uint8_t*    aValue,
    uint16_t    valueLength
)
{
    if (characteristicValueHandle == mPeerInformation[serverDeviceId].customInfo.tempClientConfig.hTemperature)
    {
        BleApp_PrintTemperature(serverDeviceId, *(uint16_t*)aValue);
/*
#if (cPWR_UsePowerDownMode)
         Restart Wait For Data timer
        TMR_StartLowPowerTimer(mAppTimerId,
                       gTmrLowPowerSecondTimer_c,
                       TmrSeconds(gWaitForDataTime_c),
                       DisconnectTimerCallback, NULL);
#endif
*/

    }
}
```

16. Go to **`BleApp_StateMachineHandler`** function.
    a. Modify the **`switch`** `(mPeerInformation.appState)` statement as below.

```
switch (mPeerInformation[peerDeviceId].appState)
```

    b. At **`case`** *`mAppIdle_c,`* modify as below.

```
case mAppIdle_c:
{
        if (event == mAppEvt_PeerConnected_c)
        {
#if gAppUseBonding_d
                if (mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTemperature !=
gGattDbInvalidHandle_d)
                {
                        /* Moving to Running State and wait for Link encryption result */
```

```
                        mPeerInformation[peerDeviceId].appState = mAppRunning_c;
                }
                else
#endif
                {
                        /* Moving to Exchange MTU State */
                        mPeerInformation[peerDeviceId].appState = mAppExchangeMtu_c;
                        GattClient_ExchangeMtu(peerDeviceId);
                }
        }
}
break;
```

c. At `case mAppExchangeMtu_c,` locate and modify the following line as below.

```
mPeerInformation[peerDeviceId].appState = mAppServiceDisc_c;
```

d. At `case mAppServiceDisc_c,` locate and modify the following lines as below.

```
1) mPeerInformation[peerDeviceId].appState = mAppReadDescriptor_c;
2) if (mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTempDesc)
3) mpCharProcBuffer->handle = mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTempDesc;
4) GattClient_ReadCharacteristicDescriptor(mPeerInformation[peerDeviceId].deviceId, mpCharProcBuffer
   ,23);
```

e. At `case mAppReadDescriptor_c,` locate and modify the following lines as below.

```
1) if (mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTempCccd)
2) if( gBleSuccess_c != BleApp_ConfigureNotifications(peerDeviceId) )
3) mPeerInformation[peerDeviceId].appState = mAppRunning_c;
```

f. At `case mAppRunning_c,` locate and modify the following lines as below.

```
1) Gap_SaveCustomPeerInformation(mPeerInformation[peerDeviceId].deviceId,
                          (void*) &mPeerInformation[peerDeviceId].customInfo, 0,
                          sizeof (appCustomInfo_t));
2) /*
   #if (cPWR_UsePowerDownMode)
          //Start Wait For Data timer
          TMR_StartLowPowerTimer(mAppTimerId,
                        gTmrLowPowerSecondTimer_c,
                        gConnReqParams.connIntervalMax * 20,
                        DisconnectTimerCallback, NULL);
   #endif
   */
3) if( gBleSuccess_c != BleApp_ConfigureNotifications(peerDeviceId) )
```

17. Go to **BleApp_ConfigureNotifications** function, locate and modify the following lines as below.

```
1)  static bleResult_t BleApp_ConfigureNotifications(deviceId_t peerDeviceId)
2)  mpCharProcBuffer->handle = mPeerInformation[peerDeviceId].customInfo.tempClientConfig.hTempCccd;
3)  GattClient_WriteCharacteristicDescriptor(mPeerInformation[peerDeviceId].deviceId,
                                             mpCharProcBuffer,
                                                    sizeof(value), (void*)&value);
```

18. Go to **DisconnectTimerCallback** function and comment the lines as below.

```
/*
#if (cPWR_UsePowerDownMode) //@AV
static void DisconnectTimerCallback(void* pParam)
{
    if (mPeerInformation.deviceId != gInvalidDeviceId_c)
    {
        Gap_Disconnect(mPeerInformation.deviceId);
    }
}
#endif

*/
```

# 5. Testing a Central device with multiple connections

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

## 5.1. Importing the Temperature Sensor Example

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

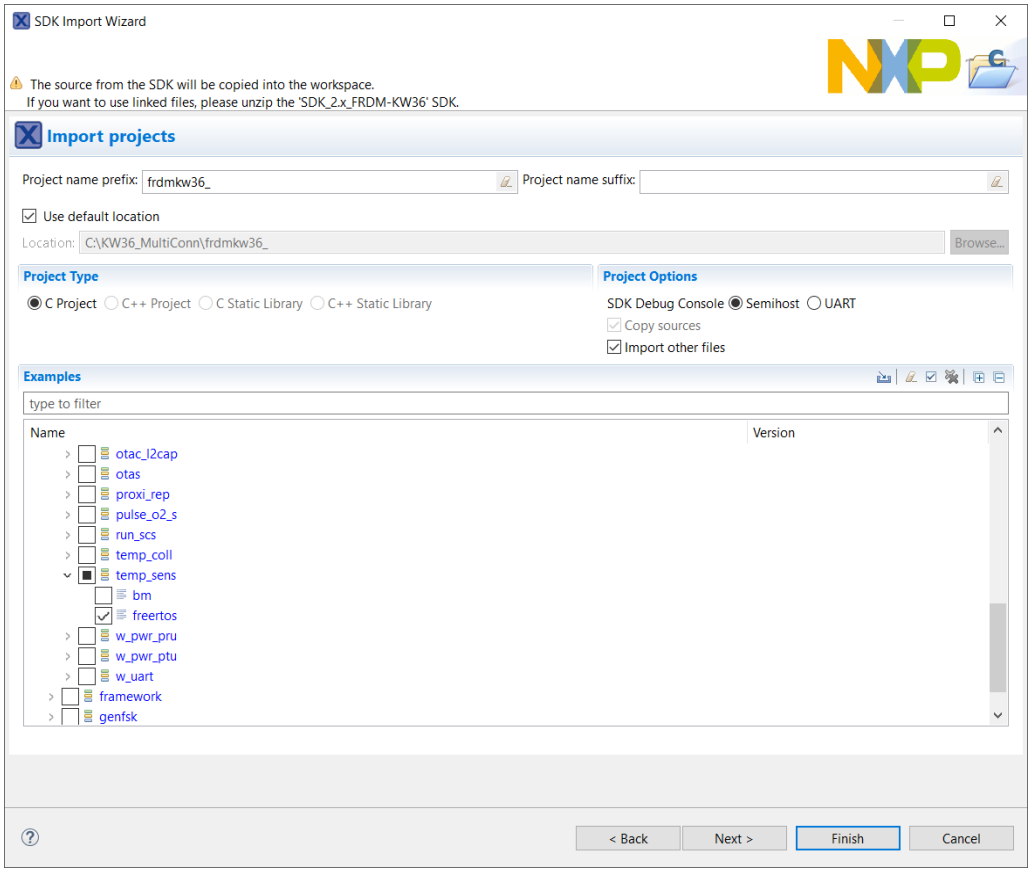


**Figure 9. Importing Temperature Sensor project to workspace**

2. Temperature Sensor application, has low-power mode enabled by default and also, as Temperature Collector, it has a timer that disconnects the device once the temperature is reported. To avoid the disconnection, the following actions can be taken:

    a) Open Temperature Sensor's *app_preinclude.h* file and modify the following macro to disable low power mode:

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

    b) Open Temperature Sensor's *temperature_sensor.c* file, go to **`BleApp_SendTemperature`** function and modify the following lines as below:

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

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

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

## 5.2. Building and downloading the projects

1. Select the **Temperature Collector** project in **Project Explorer** view and compile it by pressing the **Build** button at **Quickstart Panel** view.

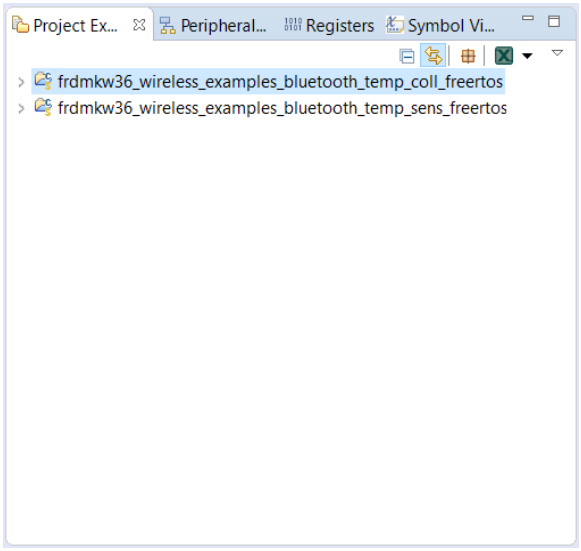


**Figure 10. Temperature Collector project selected**

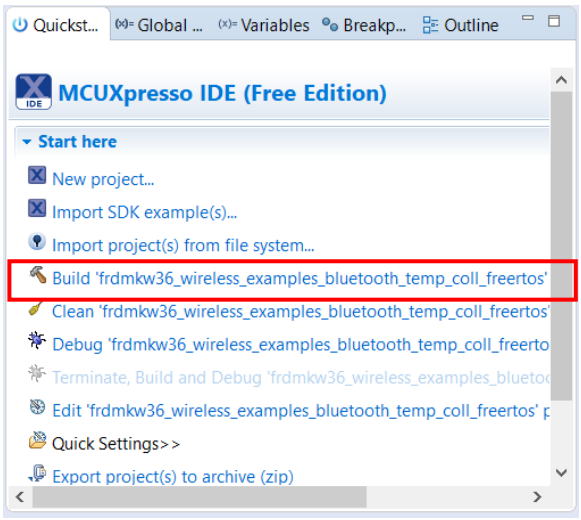


**Figure 11. Build Project button**

2. Connect the FRDM-KW36 which acts as Temperature Collector and wait for the drivers to be installed.

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

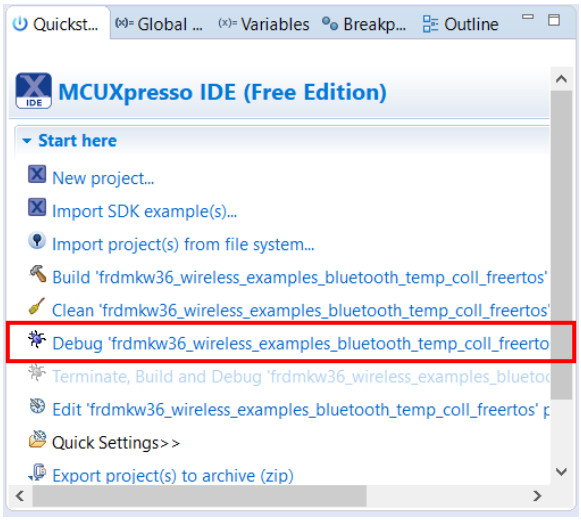


**Figure 12. Build Project button**

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

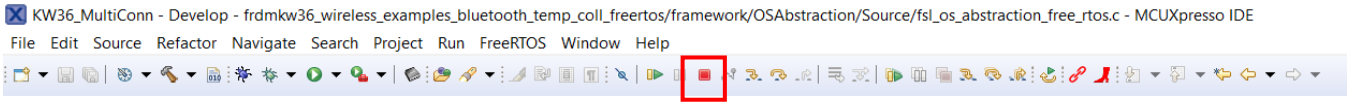


**Figure 13. Terminate button**

5. Repeat the previous steps for the remaining boards that acts as Temperature Sensors. Make sure the project selected now is the Temperature Sensor project.

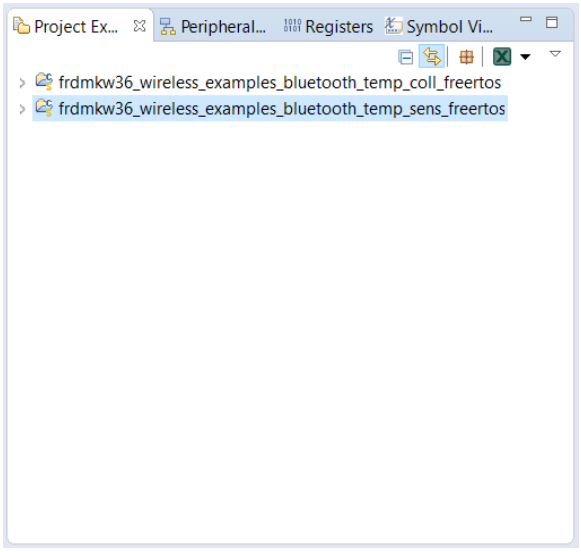

**Figure 14. Temperature Sensor project selected**

## 5.3. Running the application

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

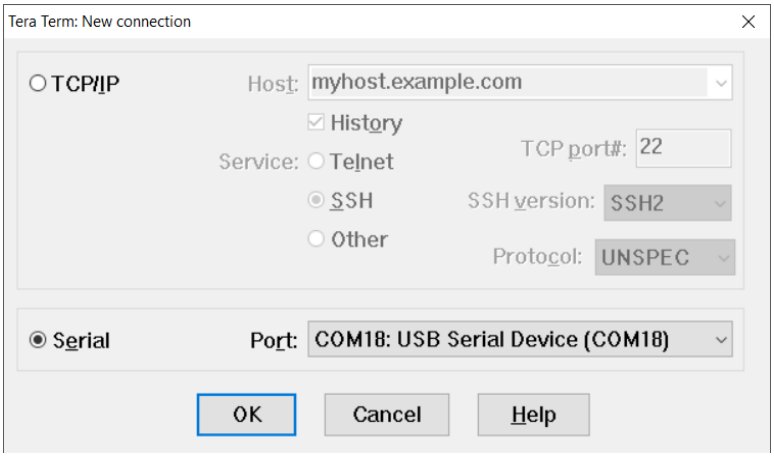


**Figure 15. Open Serial Port**

3. Expand Setup menu and click over Serial Port. Make sure to configure the settings as in Figure 16.

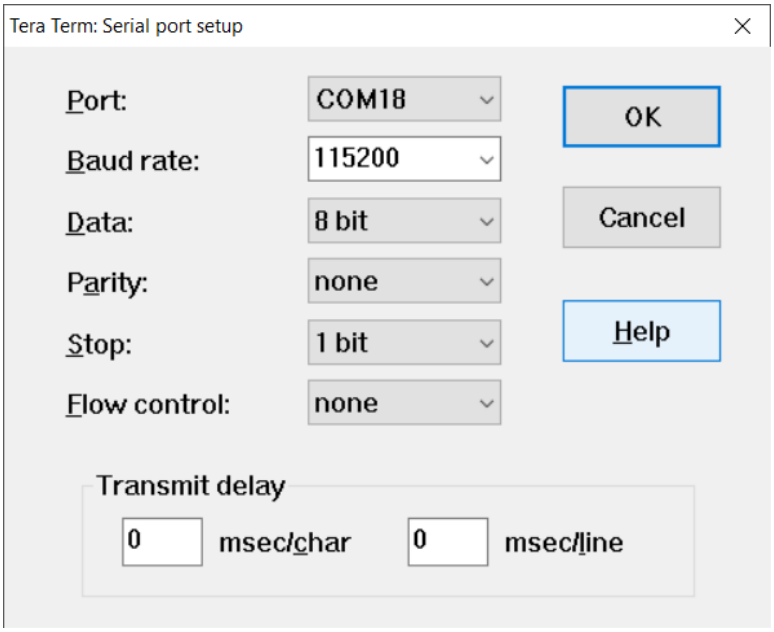


**Figure 16. Configure Serial Port**

4. Press Reset (SW1) button on the FRDM-KW36 board. The Temperature Collector should be displayed as Figure 17.

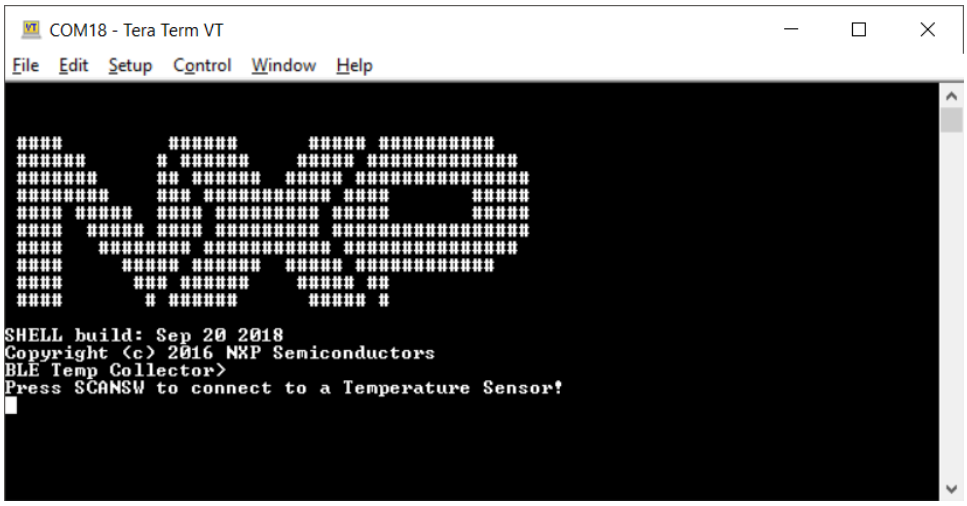


**Figure 17. Temperature Collector**

5. Connect the boards flashed with the Temperature Sensor application.
6. Press SW2 and the Temperature Collector shall start scanning for peripherals.

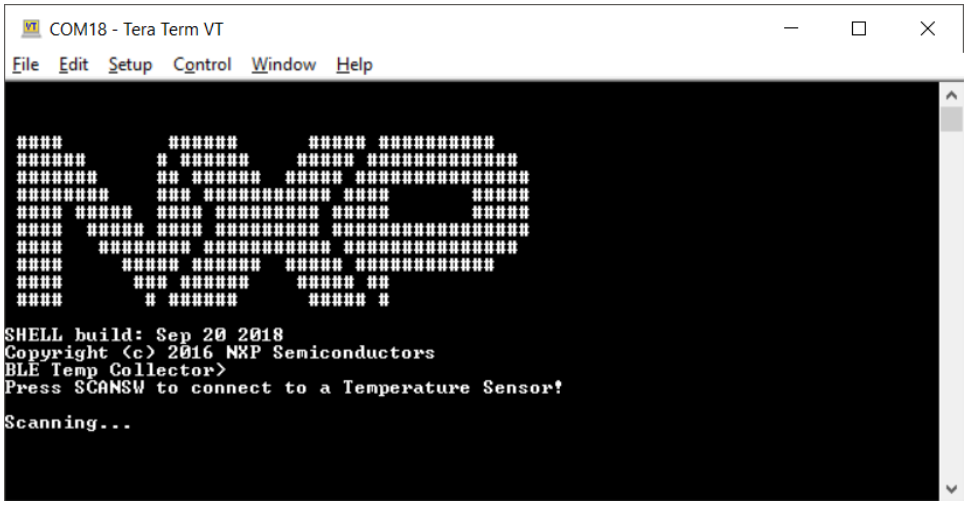


**Figure 18. Temperature Collector searching for Temperature Sensors**

7. Press SW2 on a board flashed with a Temperature Sensor to start advertising. Once the Temperature Sensor is connected, the Temperature Collector shows the temperature reported by the sensor.


**Figure 19. Temperature Collector connected to first Temperature Sensor**

8. Repeat steps 6 and 7 to connect a second Temperature Sensor to the Temperature Collector.

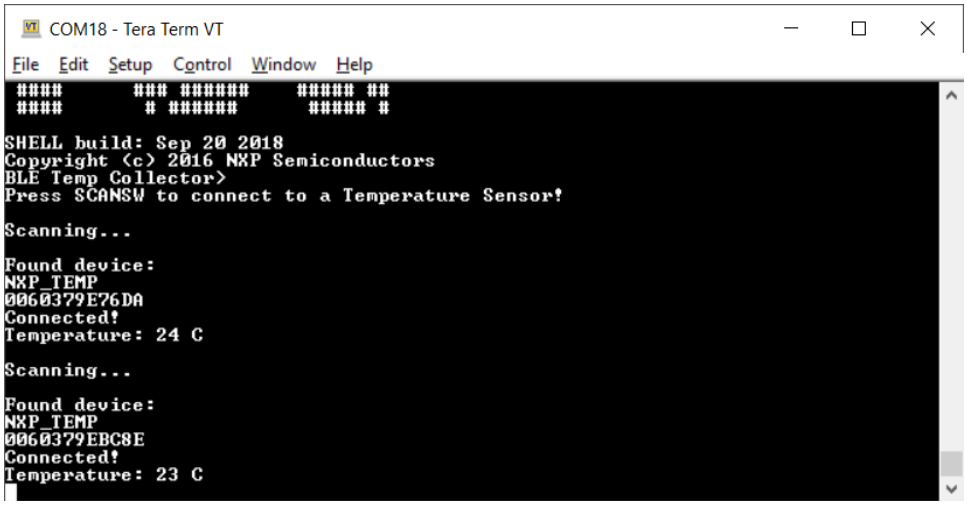


**Figure 20. Temperature Collector connected to second Temperature Sensor**

9. Press SW2 on each Temperature Sensor to trigger a temperature report to the Temperature Collector.
10. If a disconnection is performed, the Temperature Collector shows a disconnection message for each board connected.

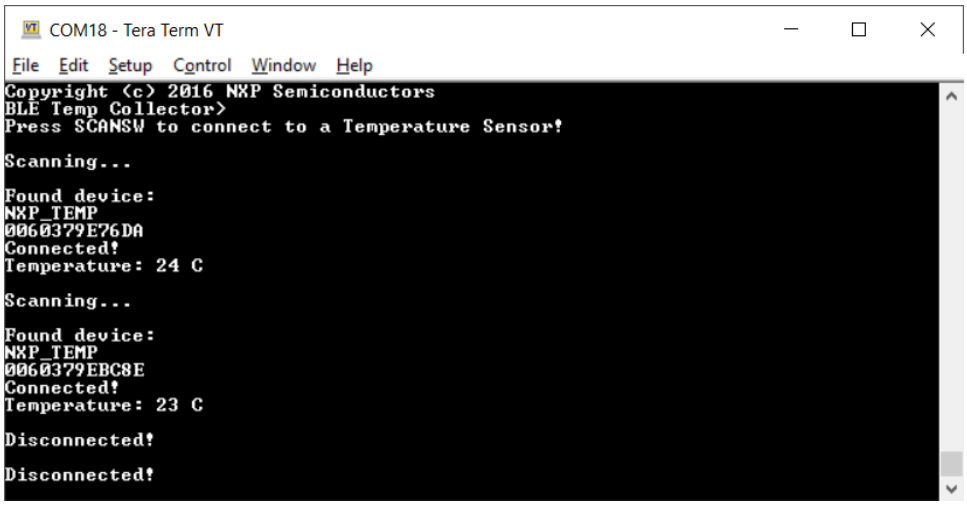


**Figure 21. Temperature Sensors disconnected from Temperature Collector**

# 6. Revision history

| Rev. Number | Date | Substantive Change |
|---|---|---|
| 0 | 05/2019 | Initial release |

***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.



Document Number: AN12414
Rev. 0
04/2019