Kinetis W MCUs: Enabling two BLE connections on a peripheral device Lab

Lab Hand Out

Contents

1 Lab Overview

This lab is intended to create simultaneous connections on a BLE peripheral device using a MCUXpresso example for the FRDM-KW41.

2 Prerequisites

The following items are needed to complete this hands-on lab:

- Boards
  - 1 FRDM-KW41Z
- Software
  - KW41Z SDK package: https://mcuxpresso.nxp.com
  - MCUXpresso IDE v10.0.0: http://www.nxp.com/mcuxpresso/ide
  - Two smartphones with Kinetis BLE Toolbox app installed
    - Available in iOS and Android App stores.
    - IOS 9 and Android 4.4 as minimum versions.

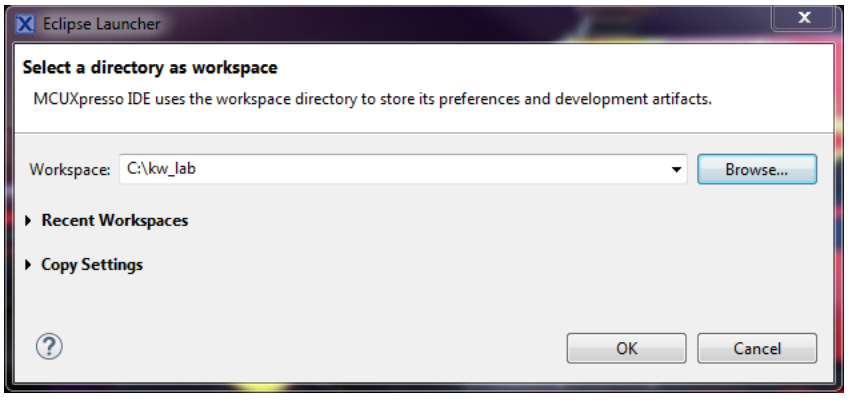
3 Heart rate sensor application

In this section you will download and import the Heart Rate Sensor application using MCUXpresso webpage and IDE.

3.1 Importing the demo

1. Open MCUXpresso IDE

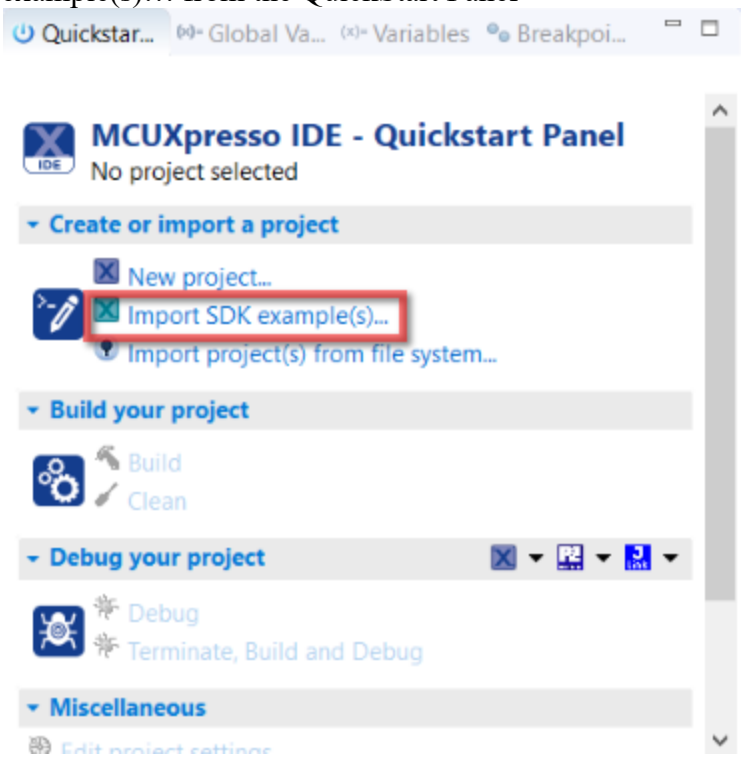
2. Set the workspace directory to **C:\kw_lab** (or your choice of any empty new directory) and click on OK.



3. Close the welcome screen

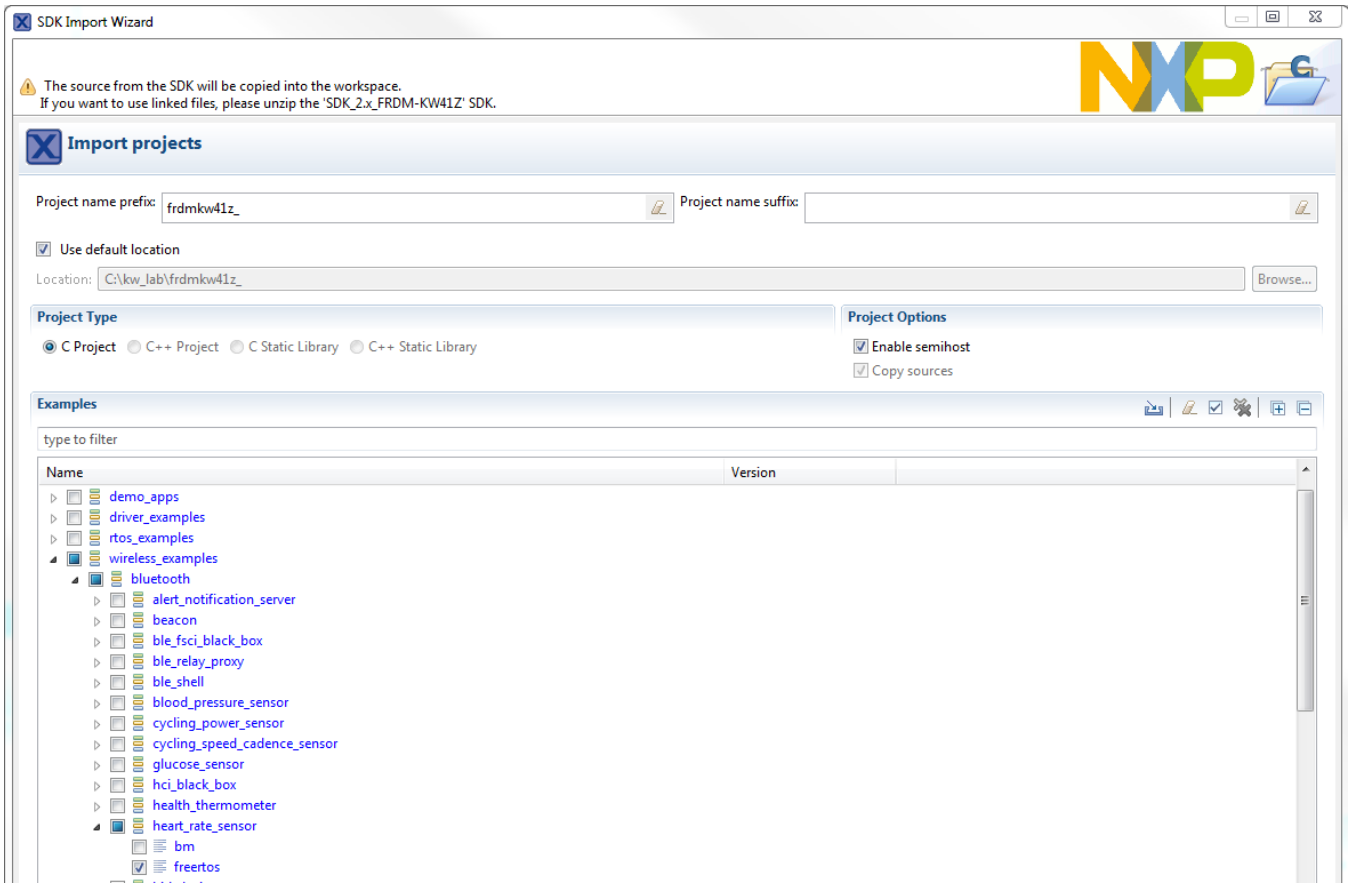
4. Next we will import the "heart rate sensor" demo for the FRDM-KW41Z.

5. Select Import SDK example(s)… from the QuickStart Panel

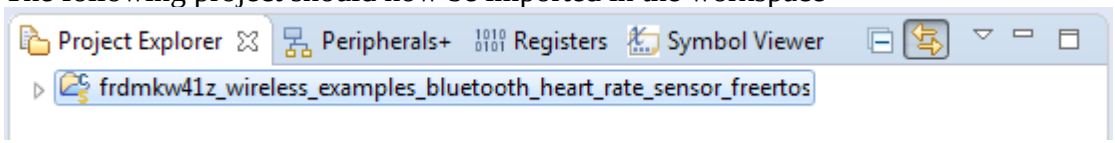


6. Select frdmkw41z from the Available boards view and click Next

7. Expand wireless_examples, then Bluetooth and heart_rate_sensor. Select FreeRTOS and click Finish.



8. The following project should now be imported in the workspace



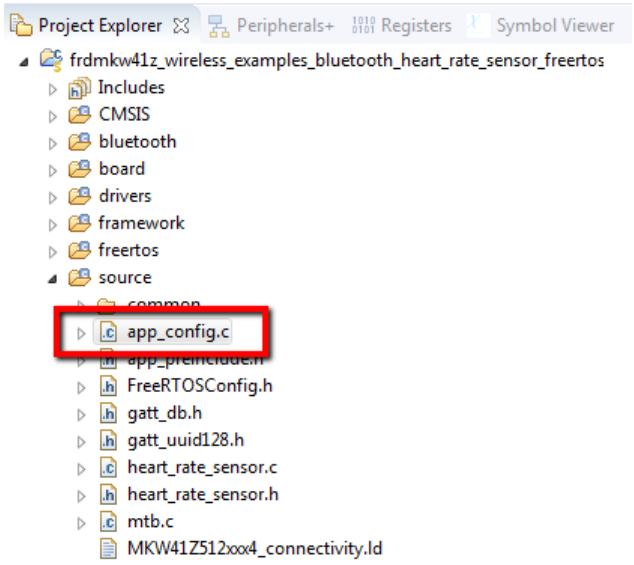
4 Adding a second connection

In this section, you will modify the Heart Rate Sensor application to allow a second connection.

4.1 Customizing the application

1. First, we will change some configuration options to make your board unique for this lab, avoid wireless interference, and ensure you'll identify your board.

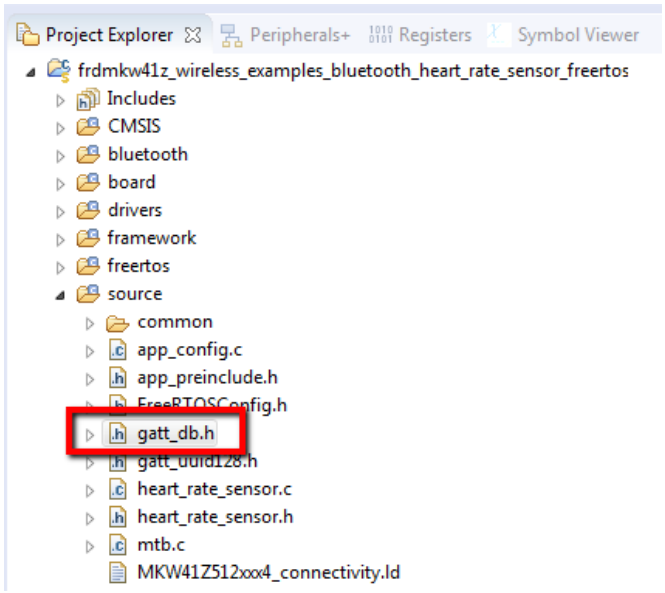
2. Change the `gAdShortenedLocalName_c` "FSL_HSR" string on **app_config.c**, line 92 and add the number assigned on the whiteboard. Make sure to update the length accordingly. This length is the amount of characters + 1
    a. File is located here:



b. After changing it, it would look similar to:
.length = 9,
.aData = (uint8_t*)"FSL_HRS1"

3. Change the Device Name on the GATT database by adding the numbers on the whiteboard and modify the size to match the new string. The GATT database resides on **gatt_db.h** and the device name is located at line 8

a. File is located here

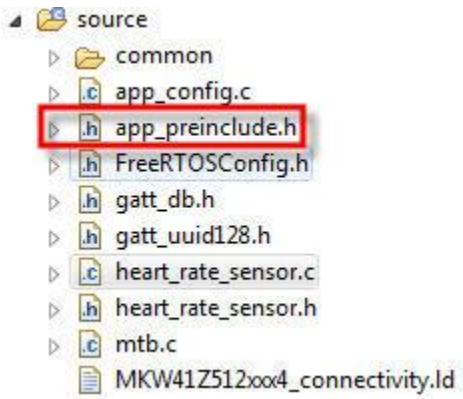


b. After changing it, it would look similar to:

VALUE(value_device_name, gBleSig_GapDeviceName_d, (gPermissionFlagReadable_c), **12, "FSL_BLE_HRS1")**

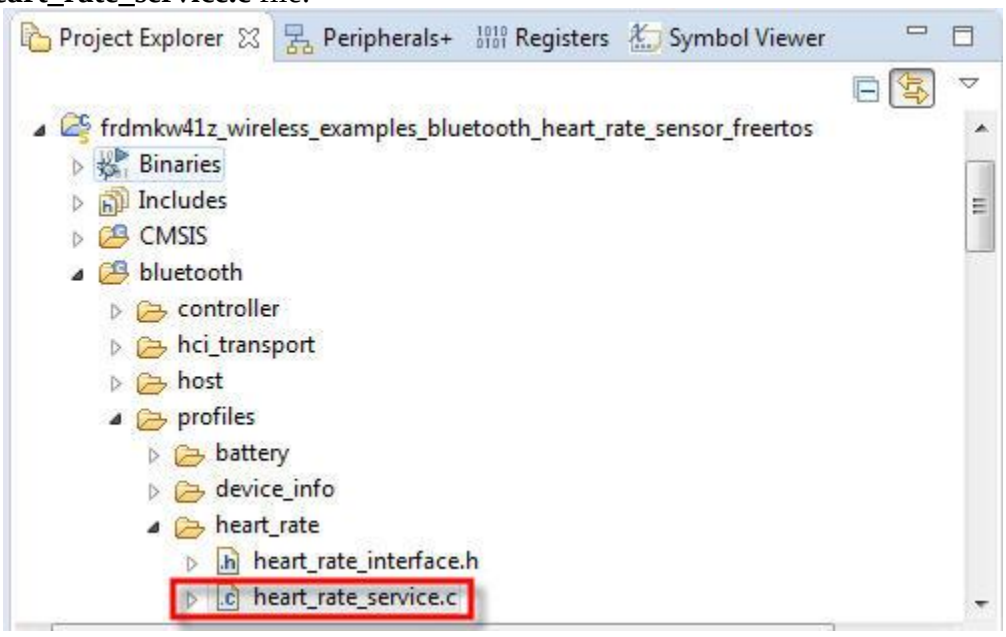
4.2 Modify heart rate service's functions

1. First, make sure low power is disabled with `cPWR_UsePowerDownMode` define in **app_preinclude.h** file around line 110.




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

2. Services' functions must be modified. Heart Rate Sensor requires a heart rate service and a battery service. Let's start modifications with the heart rate service description located in the **heart_rate_service.c** file.



3. First, modify `mHrs_SubscribedClientId` variable to store 2 devices IDs, around line 80

```
/*! Heart Rate Service - Subscribed Client*/
deviceId_t mHrs_SubscribedClientId[2] =
{
        gInvalidDeviceId_c,
        gInvalidDeviceId_c
};
```

4. Add a variable for a connections counter on line 86.

```
uint8_t HrsConnections = 0;
```

5. Initialize the created variables in **Hrs_Start** function, line 128. Make sure the previous **mHrs_SubscribedClientId** initialization is commented:

```
//mHrs_SubscribedClientId = gInvalidDeviceId_c;
  mHrs_SubscribedClientId[0] = gInvalidDeviceId_c;
  mHrs_SubscribedClientId[1] = gInvalidDeviceId_c;

   HrsConnections = 0;
```

6. Modify **Hrs_Subscribe** function to allow 2 connections on line 189.

```
bleResult_t Hrs_Subscribe(deviceId_t deviceId)
{
    //mHrs_SubscribedClientId = deviceId;
    if(HrsConnections < 2)
  {
        mHrs_SubscribedClientId[HrsConnections] = deviceId;
        HrsConnections++;
        return gBleSuccess_c;
  }
  else
  {
        return gBleInvalidState_c;
  }

    return gBleSuccess_c;
}
```

7. Modify **Hrs_Unsubscribe** function according the next code.

```
bleResult_t Hrs_Unsubscribe(deviceId_t clientDeviceId)
{
        if(HrsConnections)
        {
              if(clientDeviceId == mHrs_SubscribedClientId[0])
              {
                    HrsConnections--;
                    mHrs_SubscribedClientId[0] = gInvalidDeviceId_c;
              }
              else if(mHrs_SubscribedClientId[1] == clientDeviceId)
              {
                    HrsConnections--;
                    mHrs_SubscribedClientId[1] = gInvalidDeviceId_c;
              }
        }

        return gBleSuccess_c;
}
```

8. Then **Hrs_Stop** function, line 183.

```
bleResult_t Hrs_Stop (hrsConfig_t *pServiceConfig)
{
    Hrs_Unsubscribe(mHrs_SubscribedClientId[0]);
    Hrs_Unsubscribe(mHrs_SubscribedClientId[1]);
    return gBleSuccess_c;
}
```

9. Modify **Hrs_SendNotifications** function to send the data to both centrals.

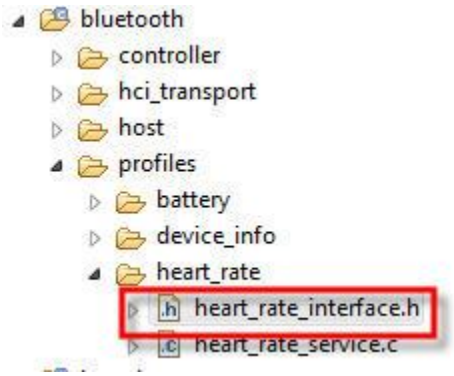
```
static void Hrs_SendNotifications(uint16_t handle)
{
    uint16_t hCccdHrMeasurement;
    bool_t isActive;

    /* Get handle of Heart Rate Measurement CCCD */
    if (GattDb_FindCccdHandleForCharValueHandle(handle, &hCccdHrMeasurement) !=
gBleSuccess_c)
        return;

  if (mHrs_SubscribedClientId[0] != gInvalidDeviceId_c)
  {
        if (gBleSuccess_c == Gap_CheckNotificationStatus \
                    (mHrs_SubscribedClientId[0], hCccdHrMeasurement, &isActive)
&& \
                          TRUE == isActive)
              {
                    GattServer_SendNotification(mHrs_SubscribedClientId[0],
handle);
              }
  }

  if (mHrs_SubscribedClientId[1] != gInvalidDeviceId_c)
  {
        if (gBleSuccess_c == Gap_CheckNotificationStatus \
                    (mHrs_SubscribedClientId[1], hCccdHrMeasurement, &isActive)
&& \
                          TRUE == isActive)
        {
              GattServer_SendNotification(mHrs_SubscribedClientId[1], handle);
        }
  }
}
```

10. Now open **heart_rate_interface.h** file under Bluetooth folder.

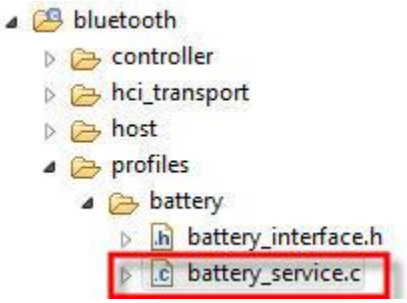


11. Add an `deviceId_t` input parameter to **`Hrs_Unsubscribe`** function on line 222:

```
bleResult_t Hrs_Unsubscribe(deviceId_t clientDeviceId);
```

4.3 Modify battery service's functions

1. Next step is to do same modifications with battery service. Let's first open **battery_service.c** file stored in Bluetooth folder.



2. Modify the ID's variable to support 2 connections. This variable is located around line 73.

```
/*! Battery Service - Subscribed Client*/
deviceId_t mBas_SubscribedClientId[2] =
{
        gInvalidDeviceId_c,
        gInvalidDeviceId_c
};
```

3. Add a variable to count connections at line 79.

```
uint8_t BasConnections = 0;
```

4. Modify **`Bas_Start`** function.

```
bleResult_t Bas_Start (basConfig_t *pServiceConfig)
{
    /* Record initial battery level measurement */
    Bas_RecordBatteryMeasurement(pServiceConfig->serviceHandle, pServiceConfig-
>batteryLevel);
```

```
    mBas_SubscribedClientId[0] = gInvalidDeviceId_c;

    mBas_SubscribedClientId[1] = gInvalidDeviceId_c;

    return gBleSuccess_c;
}
```

5. Now do the next modifications in **Bas_Stop** function.

```
bleResult_t Bas_Stop (basConfig_t *pServiceConfig)
{
    mBas_SubscribedClientId[0] = gInvalidDeviceId_c;

    mBas_SubscribedClientId[1] = gInvalidDeviceId_c;

    BasConnections = 0;

    return gBleSuccess_c;
}
```

6. Modify **Bas_Subscribe** function.

```
bleResult_t Bas_Subscribe(deviceId_t clientDeviceId)
{
    if(BasConnections < 2)
   {
         mBas_SubscribedClientId[BasConnections] = clientDeviceId;
         BasConnections++;
         return gBleSuccess_c;
   }
   else
   {
         return gBleInvalidState_c;
   }
}
```

7. Do the same with **Bas_Unsubscribe** function.

```
bleResult_t Bas_Unsubscribe(deviceId_t clientDeviceId)
{
   if(BasConnections)
   {
         if(clientDeviceId == mBas_SubscribedClientId[0])
         {
               BasConnections--;
               mBas_SubscribedClientId[0] = gInvalidDeviceId_c;
         }
         else if(mBas_SubscribedClientId[1] == clientDeviceId)
         {
               BasConnections--;
               mBas_SubscribedClientId[1] = gInvalidDeviceId_c;
         }
   }
```

```
    return gBleSuccess_c;
}
```

8. Finally, for this file do the next changes in **Bas_SendNotifications** functions.

```
static void Bas_SendNotifications(uint16_t handle)
{
    uint16_t  handleCccd;
    bool_t isActive;

    /* Get handle of CCCD */
    if (GattDb_FindCccdHandleForCharValueHandle(handle, &handleCccd) !=
gBleSuccess_c)
        return;

    if(mBas_SubscribedClientId[0] != gInvalidDeviceId_c)
    {
        if (gBleSuccess_c == Gap_CheckNotificationStatus
            (mBas_SubscribedClientId[0], handleCccd, &isActive) &&
            TRUE == isActive)
        {
            GattServer_SendNotification(mBas_SubscribedClientId[0], handle);
        }
    }

    if(mBas_SubscribedClientId[1] != gInvalidDeviceId_c)
    {
        if (gBleSuccess_c == Gap_CheckNotificationStatus
            (mBas_SubscribedClientId[1], handleCccd, &isActive) &&
            TRUE == isActive)
        {
            GattServer_SendNotification(mBas_SubscribedClientId[1], handle);
        }
     }

}
```

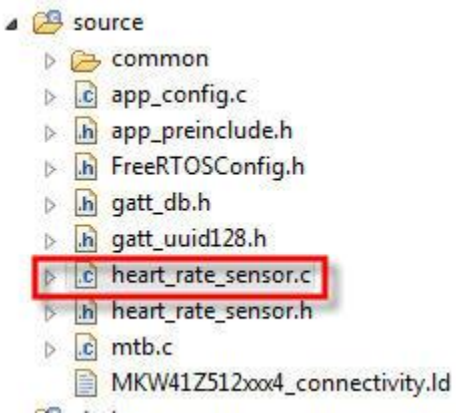
9. Now open **battery_interface.h** file stored in Bluetooth folder.

10. Update **Bas_Unsubscribe** function declaration to receive an input deviceId_t parameter.

```
bleResult_t Bas_Unsubscribe(deviceId_t clientDeviceId);
```

4.4 Modify heart rate sensor source

1. Open **heart_rate_sensor.c** file stored in source folder.



2. Modify ID variable to store 2 devices. You'll find this variable around line 122.

```
static deviceId_t  mPeerDeviceId[2] =
{

        gInvalidDeviceId_c,
        gInvalidDeviceId_c
};
```

3. Declare a timer to restart advertising on line 142.

```
static tmrTimerID_t RestartAdvTimerId;
```

4. Declare the timer callback that will be used to re-start advertising after the first device is connected on line 160.

```
static void RestartAdvTimerCallback (void *);
```

5. Definition of the timer callback. Note the timer is disabled. This definition should be at the end of the file, around line 622.

```
static void RestartAdvTimerCallback (void * pParam)
{
  BleApp_Start();

  TMR_StopTimer(RestartAdvTimerId);
}
```

6. In **BleApp_HandleKeys** (line 215) function do the following changes. These allow the application to start advertising when SW4 is pressed if there are connections available:

a. In *gKBD_EventPressPB1_c* change the comparison to handle both connections

```
//if (mPeerDeviceId == gInvalidDeviceId_c)
if ((mPeerDeviceId[0] == gInvalidDeviceId_c) || (mPeerDeviceId[1] ==
gInvalidDeviceId_c))
```

b. In *gKBD_EventLongPB1_c*. When SW4 is pressed for more than 10 seconds, both devices will be disconnected

```
//if (mPeerDeviceId != gInvalidDeviceId_c
//{
//    Gap_Disconnect(mPeerDeviceId);
if (mPeerDeviceId[0] != gInvalidDeviceId_c)
{
  Gap_Disconnect(mPeerDeviceId[0]);
}
if(mPeerDeviceId[1] != gInvalidDeviceId_c)
{
  Gap_Disconnect(mPeerDeviceId[1]);
}
//}
```

7. On **BleApp_Config** function allocate the Restart advertising at line 325.

```
RestartAdvTimerId = TMR_AllocateTimer();
```

8. In **BleApp_ConnectionCallback** do the next changes.

a. At ***gConnEvtConnected_c***

i. Comment `mPeerDeviceId = peerDeviceId` and assign the connected device ID to an empty slot on the device list on line 455.

```
 //mPeerDeviceId = peerDeviceId;
if(mPeerDeviceId[0] == gInvalidDeviceId_c)
{
      mPeerDeviceId[0] = peerDeviceId;
}
else if (mPeerDeviceId[1] == gInvalidDeviceId_c)
{
      mPeerDeviceId[1] = peerDeviceId;
}
```

ii. To start advertising after the first device is connected, add the following code on line 466. Also, comment the led flashing code lines.

```
#if (!cPWR_UsePowerDownMode)
            /* UI */
            //LED_StopFlashingAllLeds();
            //Led1On();
#endif
            if((mPeerDeviceId[0] != gInvalidDeviceId_c) && (mPeerDeviceId[1] !=
gInvalidDeviceId_c))
            {
```

```
            LED_StopFlashingAllLeds();
            Led1On();
            }
            else
            {
            TMR_StartLowPowerTimer(RestartAdvTimerId, gTmrLowPowerSingleShotMillisTimer_c,
TmrSeconds(5), RestartAdvTimerCallback, NULL);
            }
```

b. At ***gConnEvtDisconnected_c***

i. Once a disconnection happens, the services must be unsubscribed. Chane the line 511 and 512 to take the peer device as parameter to unsubscribe the correct one.

```
Bas_Unsubscribe(peerDeviceId);
Hrs_Unsubscribe(peerDeviceId);
```

ii. At line 514, add the following code to disable the current device. Don't forget to comment `mPeerDeviceId = gInvalidDeviceId_c;`

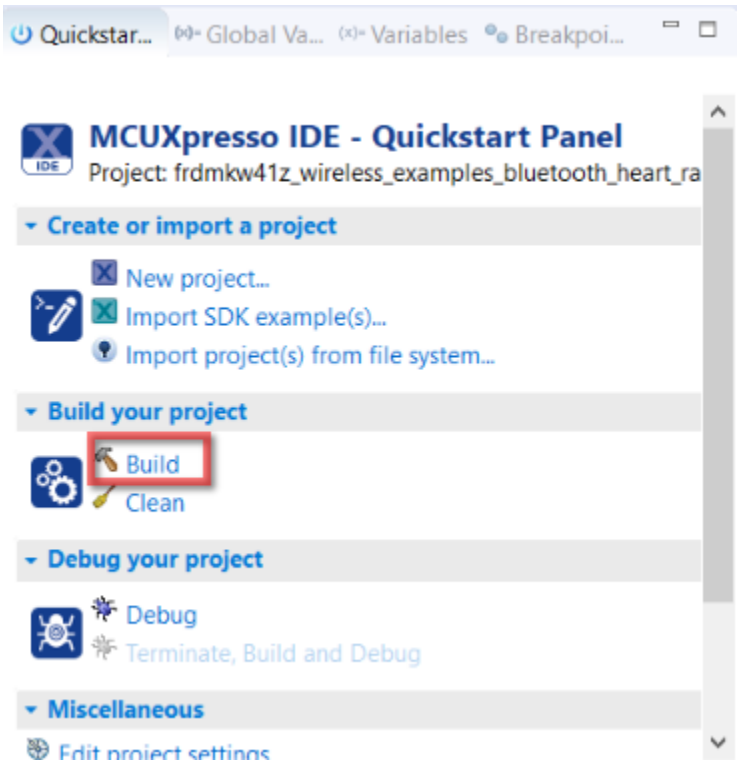
```
//mPeerDeviceId = gInvalidDeviceId_c;
if(mPeerDeviceId[0] == peerDeviceId)
{
      mPeerDeviceId[0] = gInvalidDeviceId_c;
}
else if(mPeerDeviceId[1] == peerDeviceId)
{
      mPeerDeviceId[1] = gInvalidDeviceId_c;
}
```

i. On line 524 add the code to stop timers used to report data when there's no device connected. Comment old TMR_StopTimer functions.

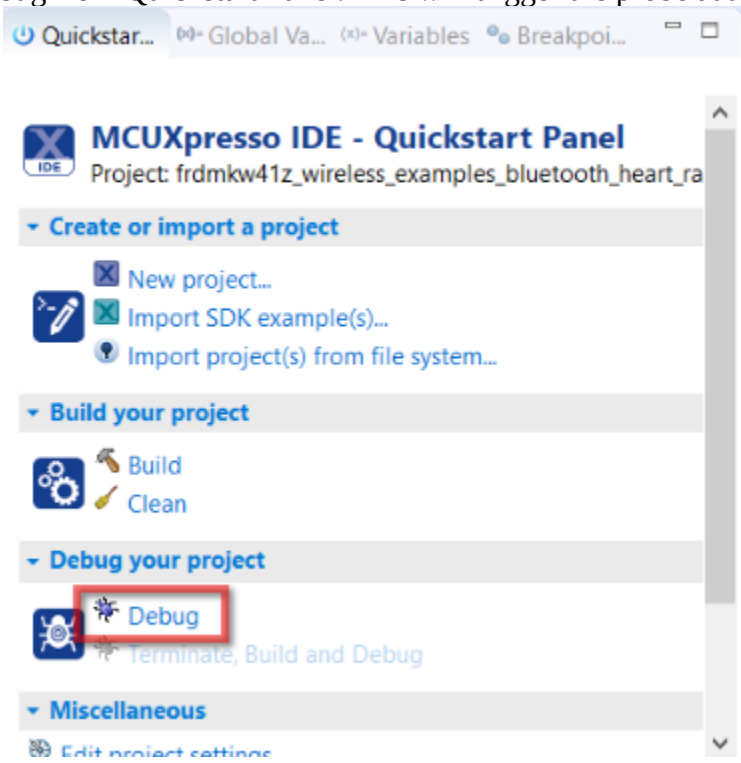
```
//TMR_StopTimer(mMeasurementTimerId);
//TMR_StopTimer(mBatteryMeasurementTimerId);
if((mPeerDeviceId[0] == gInvalidDeviceId_c)||(mPeerDeviceId[1] ==
gInvalidDeviceId_c))
{
TMR_StopTimer(mMeasurementTimerId);
TMR_StopTimer(mBatteryMeasurementTimerId);
}
```

4.5 Build and run the project

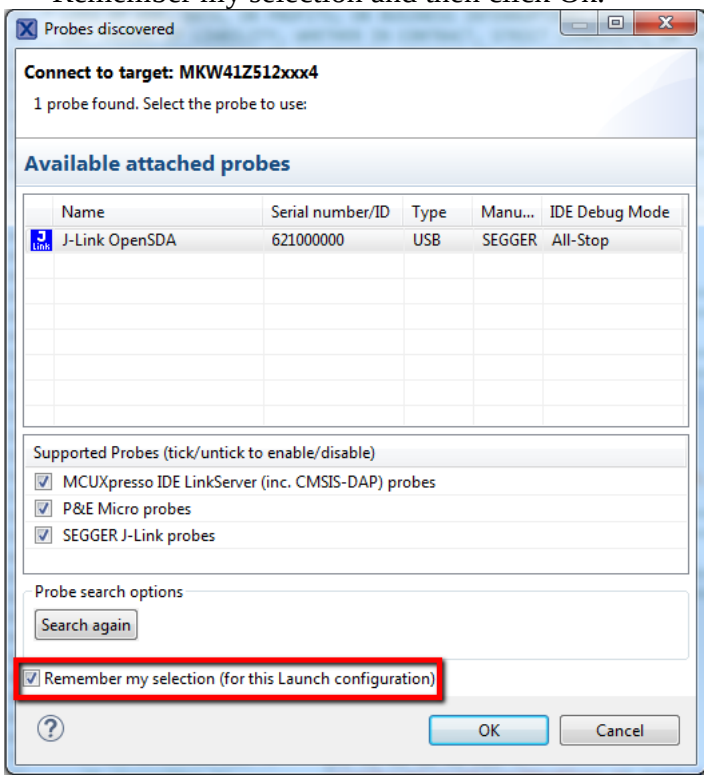
1. Now we will build this project. Click on the "heart rate sensor" project and then hit build from Quickstart Panel to compile the example. Details of the compilation process and any errors and warnings can be found by looking at the Console tab in the bottom part of the IDE.



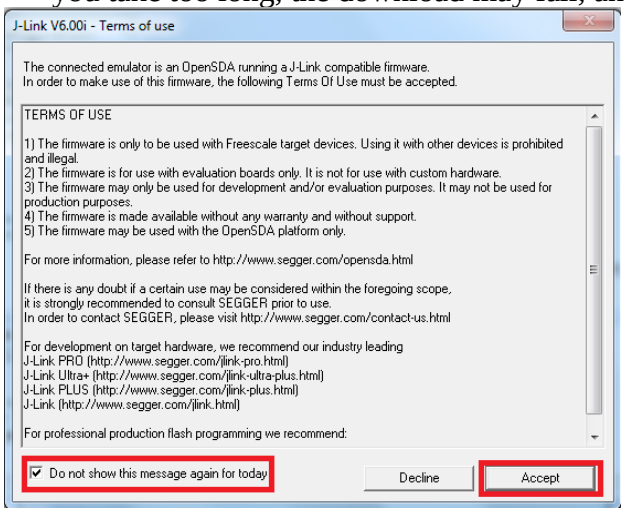
2. Download the firmware you just compiled to FRDM-KW41Z
    a. Connect the micro-USB cable to the USB connector (J6) on the board.
    b. Windows may begin to install some drivers, this is normal and should be allowed to complete before downloading the application.
    c. Click on Debug from Quickstart Panel. This will trigger the probe autodetect



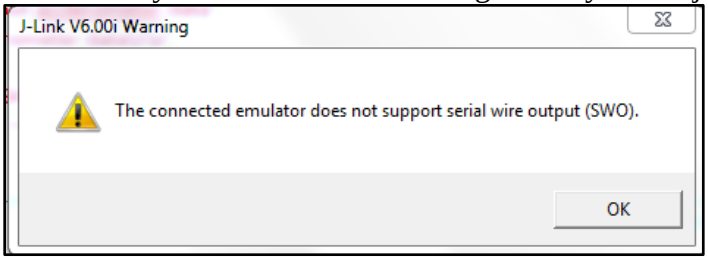
d. On Probes discovered window, make sure J-Link OpenSDA is shown. Select Remember my selection and then click Ok.



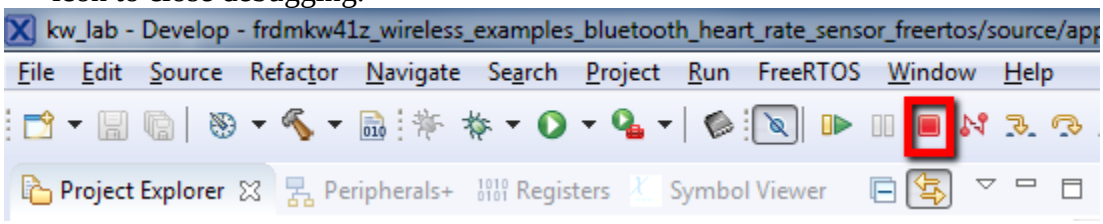
e. You may see a message about accepting the J-Link terms of use. Check the checkbox to disable this message for the rest of the day, and then click on Accept. Note that if you take too long, the download may fail, and you will need to try again.



f. You may also see a SWO warning which you can just click OK to dismiss:



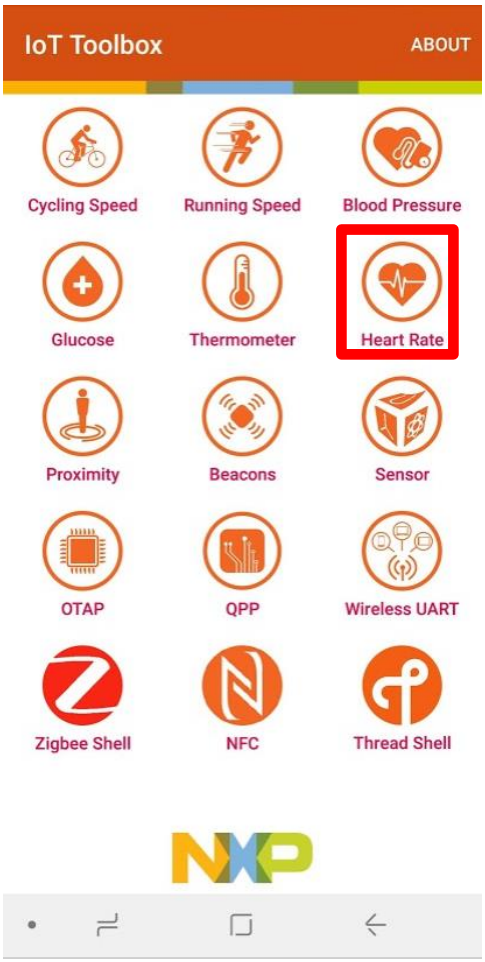
g. MCUXpresso will now flash the board. Once it is finished, click on the Terminate icon to close debugging.



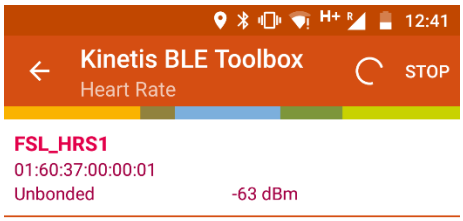
h. Power cycle the board.

4.6 Run heart rate sensor with Kinetis BLE Toolbox

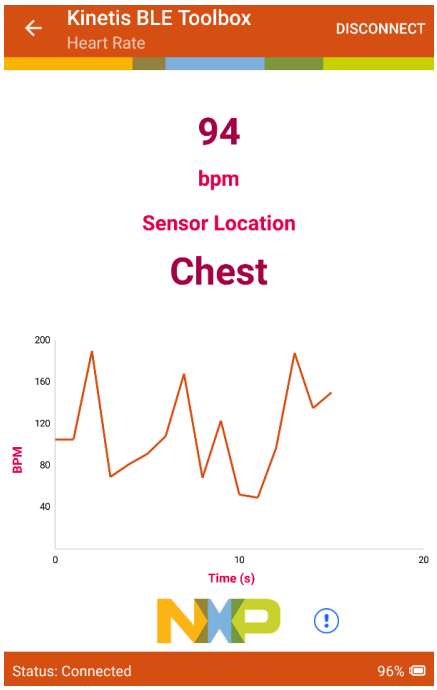
1. Open the IoT Toolbox in two smartphones and select Heart Rate option.



2. Press SW4 to start advertising and select connect the first device to FRDM-KW41.



3. When the device is connected, the application shows the sensor location, the current beats per minute and a graph with the latest measurements. These values are randomly generated on the MCU.



4. Repeat steps 2 and 3 to connect the second device.