AN12896

FSCI Blackbox with MCU Host Bluetooth LE Temperature Sensor

Rev. 0 — June 2020

Application Note

Contents

1 Introduction

This document describes the usage of an Framework Serial Communication Interface (FSCI) wrapper developed to control the FSCI Blackbox available on the KW36 platform. The application example emulates the behavior of the temperature sensor from the KW36 SDK using a K64F MCU and a KW36 running the FSCI Blackbox application. The application supports UART or SPI for FSCI serial communication between the FRDM-KW36 and FRDM-K64F. FRDM-K64F measures the temperature and sends it to the KW36 Blackbox using FSCI protocol. FRDM-KW36 transmits temperature data over the air to remote Bluetooth Low Energy (LE) temperature collector device.

2 Development environment

The K64F FSCI temperature sensor application consists of the following setup:

- Hardware
  - 2 x FRDM-KW36 board (version B)
  - 1 x FRDM-K64F board (version D)
  - 3 x Micro USB cables
  - Jumper wires to interface FRDM-KW36 and FRDM-K64F using SPI or UART
- Software
  - SDK packages for KW36 (version 2.2.2) and K64F (version 2.7.0)

    Get it from here.
  - MCUXpresso IDE v11.0.1 or Higher

    Get it from here.
  - K64F FSCI temperature sensor application
  - frdmkw36_wireless_examples_bluetooth_ble_fscibb_freertos application
  - frdmkw36_wireless_examples_bluetooth_temp_coll_freertos application

NOTE

- K64F FSCI temperature sensor application is not part of the FRDM-K64F SDK demo but the software package is available with this application note.
- frdmkw36_wireless_examples_bluetooth_ble_fscibb_freertos application and frdmkw36_wireless_examples_bluetooth_temp_coll_freertos application are part of demo examples available in the FRDM-KW36 SDK.
- Initially, the user may keep low-power mode disabled in BLE collector device application. For further details of low-power mode behavior, see Disconnection.

3 Bluetooth LE API wrapper architecture

This section describes the wrapper implementation details. In the K64F temperature sensor application, the Bluetooth LE Abstraction layer implements various Bluetooth LE APIs wrappers. When the application layer calls any Bluetooth LE host stack API, that API specific wrapper function is being called which is defined at the Bluetooth LE Abstraction layer. The details of the wrapper definition are as follows:

- Wrapper definition uses the `msgBLEabsToFSCI_t` structure to prepare data that is required by the FSCI layer to form FSCI packet.
- `msgBLEabsToFSCI_t` has fields for opGroup and opCode, populated based on the API for which a wrapper is defined. For API mapping with opGroup and opCode, see Bluetooth Low Energy Host Stack FSCI Reference Manual. Document is available inside FRDM-KW36 SDK at location: SDK_2.2.2_FRDM-KW36\docs\wireless\Common.
- Length field must be defines based on the payload value, as it denotes payload length.
- Payload field must be defined based on the data provided by the application layer. Payload is provided to the Bluetooth LE Abstraction layer by the application layer in the form of a wrapper argument.
- `msgBLEabsToFSCI_t` has union `bleRequestType` which consists of payload structures for all the defined API wrappers.
- If the user wants to implement a wrapper for any Bluetooth LE Host stack API, then the user must update `bleRequestType` union and enumerations for request response opCode and opGroup. See Enumeration and structures for details of enumerations and structures.
- When the data is defined in the `msgBLEabsToFSCI_t` structure, the wrapper function inserts the data into the queue and raises the `gFSCIEvtMsgFromAbs_c` event.
- Wrapper function waits for the result that must be provided to the application layer as a return value of the API using `xTaskNotifyWait()` FreeRTOS API. When the FSCI layer gets a response from the KW36 FSCI Blackbox application, it forwards the status using `xTaskNotify()` FreeRTOS API.

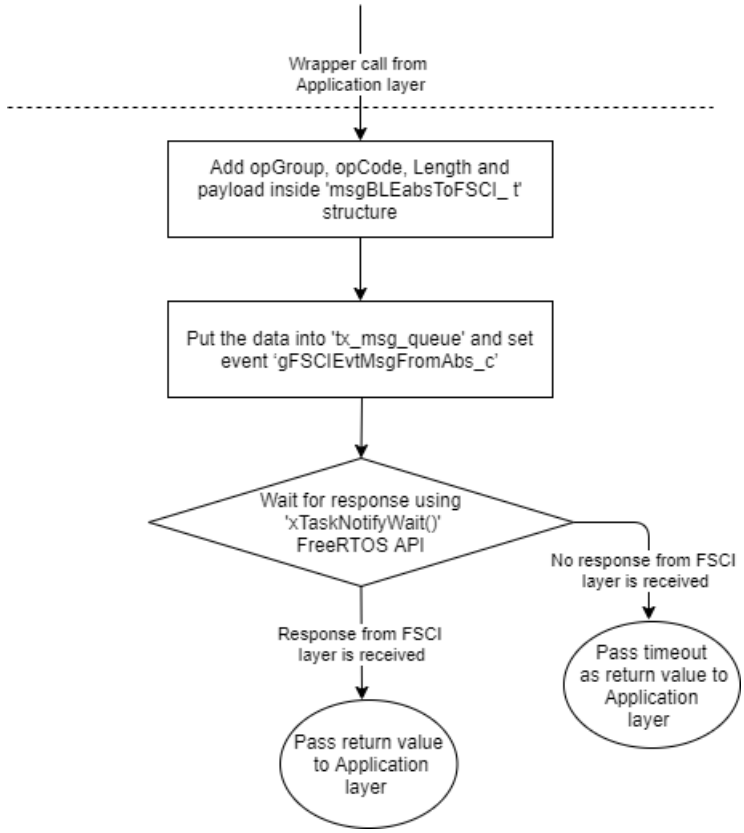

Figure 1. Wrapper architecture

4 Implementation of K64F FSCI application

This section describes how to use the Bluetooth LE stack APIs wrapper to implement Bluetooth LE peripheral functionality.

4.1 Application initialization

K64F FSCI temperature sensor application is divided into three different layers. The following figure shows the layered architecture used in the K64F Application.

1. Application layer
2. Bluetooth LE Abstraction layer
3. FSCI layer

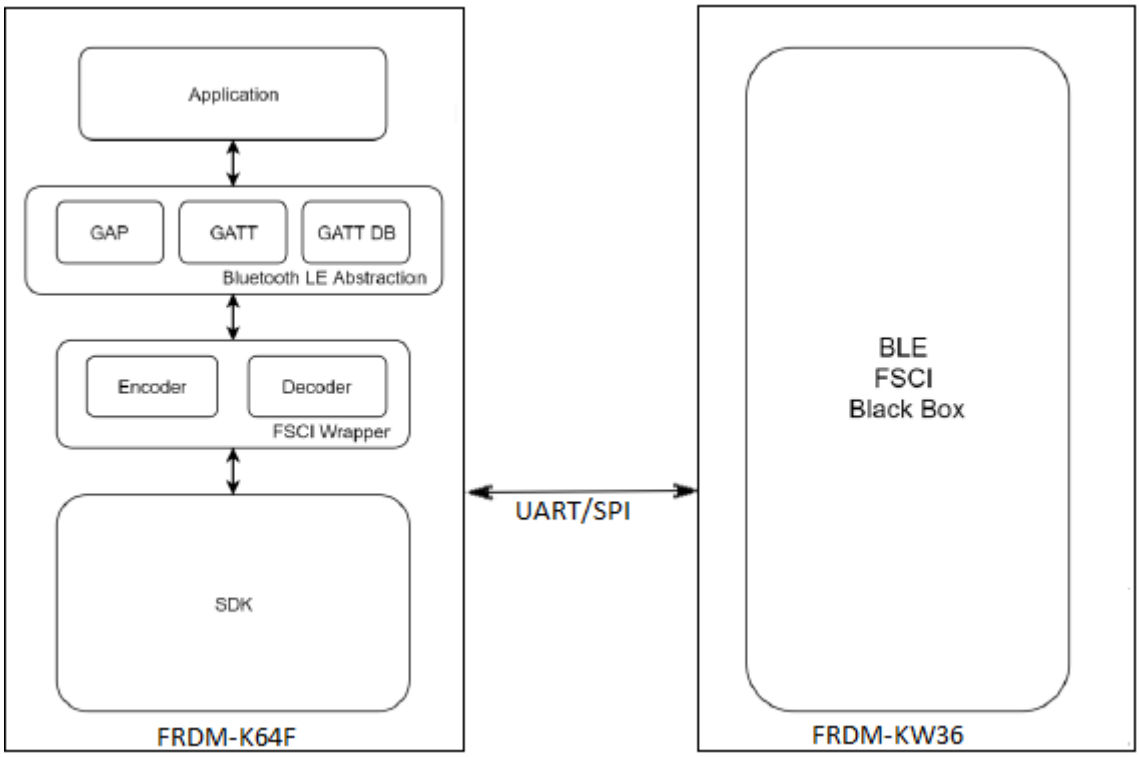

Figure 2. System overview

Application initialization includes FRDM-K64F initialization and Bluetooth LE specific initialization. Bluetooth LE specific initialization messages are sent to the FRDM-KW36 FSCI Blackbox application over the serial interface. The application initializes a Bluetooth LE custom profile and adds different services and characteristics to GATT-DB maintained by the FRDM-KW36 device. K64F application creates various tasks and queues which are used to send and receive FSCI messages. This section describes the details of the initialization process.

- Three queues are created:
  - `tx_msg_queue`: Handle communication from Bluetooth LE Abstraction layer to FSCI layer.
  - `rx_msg_queue`: Handle communication from FSCI layer to Bluetooth LE Abstraction layer.
  - `rsp_msg_queue`: Handle communication from Bluetooth LE Abstraction layer to Application layer.
- Creation of custom profiles, adding services, and characteristics in GATT-DB. The application adds GATT, GAP, temperature, battery and device information services, and characteristics for these services. The GATT-DB is created on FRDM-KW36.

  The APIs used for adding services, characteristics, and CCCD are:
  - `GattDb_AddPrimaryService()`
  - `GattDb_AddCharacteristicDeclarationAndValue()`
  - `GattDb_AddCharacteristicDescriptor()`
  - `GattDb_AddCccd()`
- Application callback registration: GAP generic callback and GATT server callback are registered using using the following API:
  - `App_RegisterGenericCallback(gapGenericCallback_t)`
  - `App_RegisterGattServerCallback(gattServerCallback_t)`

Details of the callbacks are described in Application callbacks.

All the above mentioned APIs in this section are defined at the Bluetooth LE Abstraction layer.

4.2 FSCI packet formation

This section describes the FSCI frame formation process and the FSCI frame structure.

- `FSCI_transmitPayload()` API is implemented to prepare FSCI messages for all the commands using the information provided by the Bluetooth LE Abstraction layer. Bluetooth LE Abstraction layer passes opGroup, opCode, Length, and payload.
- API has four arguments: opGroup, opCode, length, and payload. The API prototype is as follows:

```
void FSCI_transmitPayload(uint8_t opGroup, uint8_t opCode, uint8_t *payload, uint16_t Length);
```

- The above mentioned API prepares the FSCI frame header. The header field includes startMarker, opGroup, opCode, and length. Computes checksum for the header and payload field.

The following figure shows the FSCI frame format. For more details on this, see Connectivity Framework Reference Manual section Framework Serial Communication Interface. Document is available inside FRDM-KW36 SDK at location: SDK_2.2.2_FRDM-KW36\docs\wireless\Common.

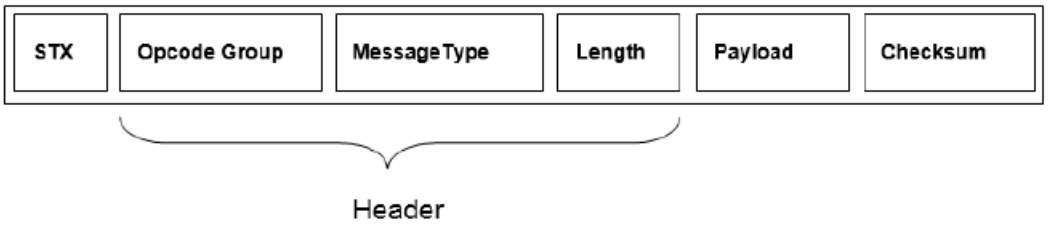

Figure 3. FSCI packet structure

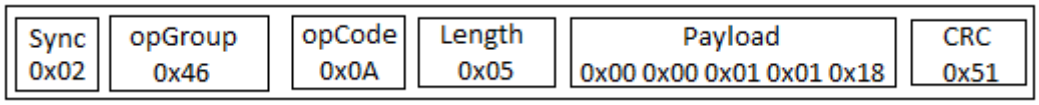

Figure 4. FSCI packet for add primary service command

The following table describes the details of FSCI packet fields.

Table 1. FSCI packet field description

Field name	Length (byte)	Description
Sync/STX	1	Used for synchronization over the serial interface. The value is always 0x02.
opGroup	1	Distinguishes between different Bluetooth LE host stack layer that is GATT, GATT-DB, and GAP.
Message Type/opCode	1	Specifies the exact message opCode that is contained in the packet.
Length	1	The length of the packet payload, excluding the header and FCS. The length field content must be provided in little-endian format.

Table continues on the next page...

Table 1. FSCI packet field description (continued)

Field name	Length (byte)	Description
Payload	Variable	Payload of the actual message.
Checksum	1	Checksum field used to check the data integrity of the packet.

4.3 Sending FSCI message from K64F to KW36 FSCI Blackbox

When an API from the Bluetooth LE Abstraction layer is called from the application layer, the following process takes place.

- Bluetooth LE Abstraction layer processes the data provided by the application layer, append opCode, opGroup, and calculates the length of the payload based on the size of the payload which application layer provides.
- Bluetooth LE Abstraction layer uses msgBLEabsToFSCI_t structure to form messages for various APIs and passes the created messages to the FSCI layer.
- `msgBLEabsToFSCI_t` structure contains the following parameters:
  - opGroup: This field value can be assigned from `bleOpGroupType_t` enumeration.
  - opCode: This field value can be assigned from `bleGATTDBOpCodeType_t`, `bleGATTOpCodeType_t`, or `bleGAPOpCodeType_t` enumerations.
  - Length: This field shows the size of the payload, which is populated in the `bleRequestType` union.
  - bleRequestType: This is a union, that contains various request payload structures. This parameter is defines based on API called by the application layer, opGroup, opCode, bleRequestType, and length.

    Any command which does not have a payload, for that length field is set to 0 and no need to define the bleRequestType field.
- When the message is formed, the Bluetooth LE Abstraction layer inserts the message in `tx_msg_queue` message queue, sets the `gFSCIEvtMsgFromAbs_c` event, and wait for the response of the same request.
- At the Bluetooth LE Abstraction layer, every command waits for the response. This response waiting mechanism is developed by FreeRTOS task notification with a timeout. FSCI layer receives the response from the KW36 FSCI Blackbox application over SPI/UART interface and passes to the Bluetooth LE Abstraction layer.
- FSCI layer uses the `FreeRTOS FSCI_SendPacketTask` task to receive data from the Bluetooth LE Abstraction layer via a message queue, when the data is available in the queue, FSCI layer implements a mechanism to form the FSCI packets as described in FSCI packet formation.
- When the message is formed as per the FSCI frame format, the FSCI layer uses UART/SPI data transfer API to send the FSCI messages over UART/SPI to KW36 FSCI Blackbox device.

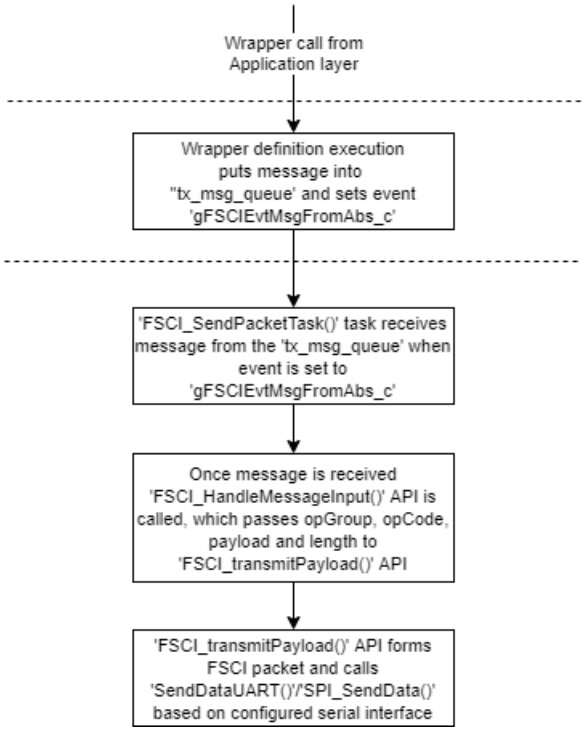

Figure 5. FSCI message send flow diagram

4.4 Receiving an FSCI message from KW36 FSCI Blackbox to K64F

This section describes how the response received from the FRDM-KW36 FSCI Blackbox device is processed by the FSCI layer and forward to the application layer through the Bluetooth LE Abstraction layer.

- Data received over UART/SPI interface is provided to `Prepare_FSCI_Response()` API. This API inserts the received data in the message queue which is established between the FSCI layer and Bluetooth LE Abstraction layer.
- Bluetooth LE Abstraction layer has the receive task (`FSCI_ReceivePacketTask`), which receives data from the receive queue. When the data is received from the queue, the Bluetooth LE Abstraction layer process the data using the following API.

```
void FSCI_HandleResponse(FSCIResponseMsg_t *pMsg, uint8_t *payload, uint8_t crc);
```

- The above mentioned API process the received data and decodes it in `FSCIResponseMsg_t` structure. Following is the list of structure members:
  - sync: This field indicates the start of a frame. The value of the field is always set to 0x02.
  - opGroup: This field contains opGroup of the received response.
  - opCode: This field contains opCode of the received response.
  - length: Length of the payload inside the response.
  - ResponseType: This is a union, which has various structures for different response payloads.
  - crc: Checksum of the response.
- After decoding the data, the Bluetooth LE Abstraction layer takes the action based on the response received, that is triggering callback, notifying the status of a request, and so on.

- Callback functions running in context of `FSCI_ReceivePacketTask()` uses `rsp_msg_queue` message queue and `gAppEvtMsgFromAbs_c` event for communication between the two tasks `FSCI_ReceivePacketTask()` and `App_thread()`. In this way the `App_thread()` receives events.

  See Application callbacks for more details regarding application callback functionality.

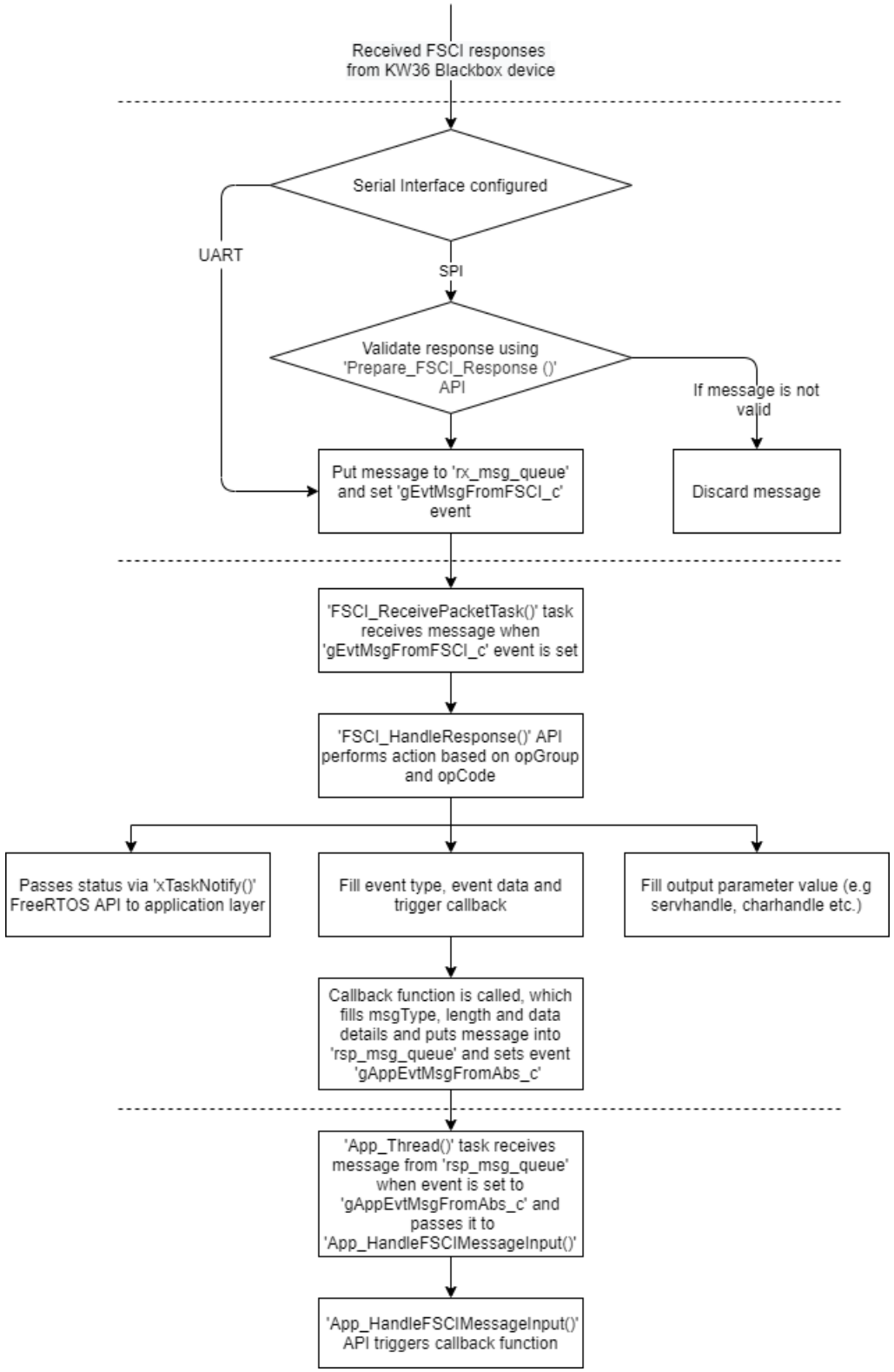

Figure 6. FSCI receive message flow diagram

4.5 Application callbacks

When the initialization is performed at the application layer, the application registers various callbacks as follows:

App_GenericCallback: This callback is registered at application initialization and executed when `gapAdvParamSetupComp` and `gapAdvSetupFailed` events are received from the KW36 FSCI Blackbox application.

App_GattServerCallback: This callback is registered at application initialization and executed on receiving the `gattCharCccdWritten` event from the KW36 FSCI Blackbox application.

APP_AdvertisingCallback: This callback is registered at advertisement starting and executed on receiving `gAppGapAdvertisementMsg_c` event from the KW36 FSCI Blackbox application.

App_ConnectionCallback: This callback is registered at advertisement starting and executed on receiving `gAppGapConnectionMsg_c` and `gAppGapDisConnectionMsg_c` events from the KW36 FSCI Blackbox.

- `App_Thread()` task receive messages from the above callbacks via FreeRTOS message queue `rsp_msg_queue`. `App_Thread()` task further passes these messages to `App_HandleFSCIMessageInput()` API.
- `App_HandleFSCIMessageInput()` API action based on the receiving events. For example, on receiving a connection event, start an LED indication, and stop advertisement timer.

4.6 Developing with application note

4.6.1 Bluetooth LE abstraction code

Folder location: `<Project_Directory>\source\BLE_Abstraction`

Table 2. Bluetooth LE abstraction

File name	Description
BLE_Abstraction_main.c/.h[1]	Receive and handle all FSCI responses.
ble_FSCI.h	Defined GAP, GATT, GATTDB opcodes, and FSCI response structure.
ble_general.h	Defined Bluetooth LE status, error codes, and event types.
ble_sig_defines.h[1]	Defined Bluetooth LE SIG UUID constants.
EmbeddedTypes.h[1]	Defined type constants.
gap_interface.c/.h	Defined Bluetooth LE Abstraction layer APIs for GAP commands.
gap_types.h	Defined GAP layer-specific procedure constants.
gatt_database.h[1]	Defined GATTDB constants.
gatt_server.c/.h	Defined Bluetooth LE Abstraction layer API for GATT server commands.
gatt_types.h	Defined structures for GATT services, characteristics, and attribute constants.
gatt_uuid128.h	Defined temperature service UUID.
gattdb_interface.c/.h	Defined Bluetooth LE Abstraction layer APIs for GATTDB commands.

1. Do not modify the header file.

4.6.2 FSCI code

Folder location: `<Project_Directory>\source\FSCI`

Table 3. FSCI

File name	Description
FSCI_main.c/.h[1]	Preparing FSCI packets for every GAP, GATT, and GATTDB commands.
SPI.c/.h[2]	Defined APIs for SPI initialization, configuration, and send FSCI packet to FRDM-KW36 via SPI interface.
UART_main.c/.h[2]	Defined APIs for UART initialization, configuration, and send FSCI packet to FRDM-KW36 via UART interface.

1. Do not modify the header file.
2. Source and header files are developed specific to K64F device.

4.6.3 Bluetooth LE device profile code

Folder location: `<Project_Directory>\source\profiles`

Table 4. Bluetooth LE profile

File name	Description
battery_interface.h	Interface for battery service. APIs to start, stop, subscribe, and unsubscribe battery service are defined in this file.
battery_service.c	Includes definition of the APIs used in battery interface file.
device_info_interface.h	Interface for device information service. APIs to start and stop the service are declared in this file.
device_info_service.c	Includes definitions of the APIs used in device information interface file.
temperature_interface.h	Interface for temperature service. APIs to start, stop, subscribe, and unsubscribe the temperature service are declared in this file.
temperature_service.c	Includes definition of the APIs used in temperature interface file.

4.6.4 Temperature sensor code

Folder location: `<Project_Directory>\source`

Table 5. Temperature sensor

File name	Description
app_config.c	Defined configurations for advertising data and parameters.
AppMain.c/.h	Done button and LED Initialization. Register callback functions for advertising, connection, and so on.
FreeRTOSConfig.h[1]	FreeRTOS specific configuration file.

Table continues on the next page...

Table 5. Temperature sensor (continued)

temperature_sensor.c/.h	Initiates set advertisement data and parameter procedures. Added service and characteristics for temperature sensor application.
semihost_hardfault.c[1]	Defined hard fault handler functionality.

1. Source and header files are developed specific to K64F device.

NOTE

Based on the application requirements the user can change the files. Consider the following points, when changing the source code files.

- Source and header files are developed specific to K64F device.
- Do not modify the header file.

4.7 Summary

The following figure describes the summary of Implementation of K64F FSCI application.

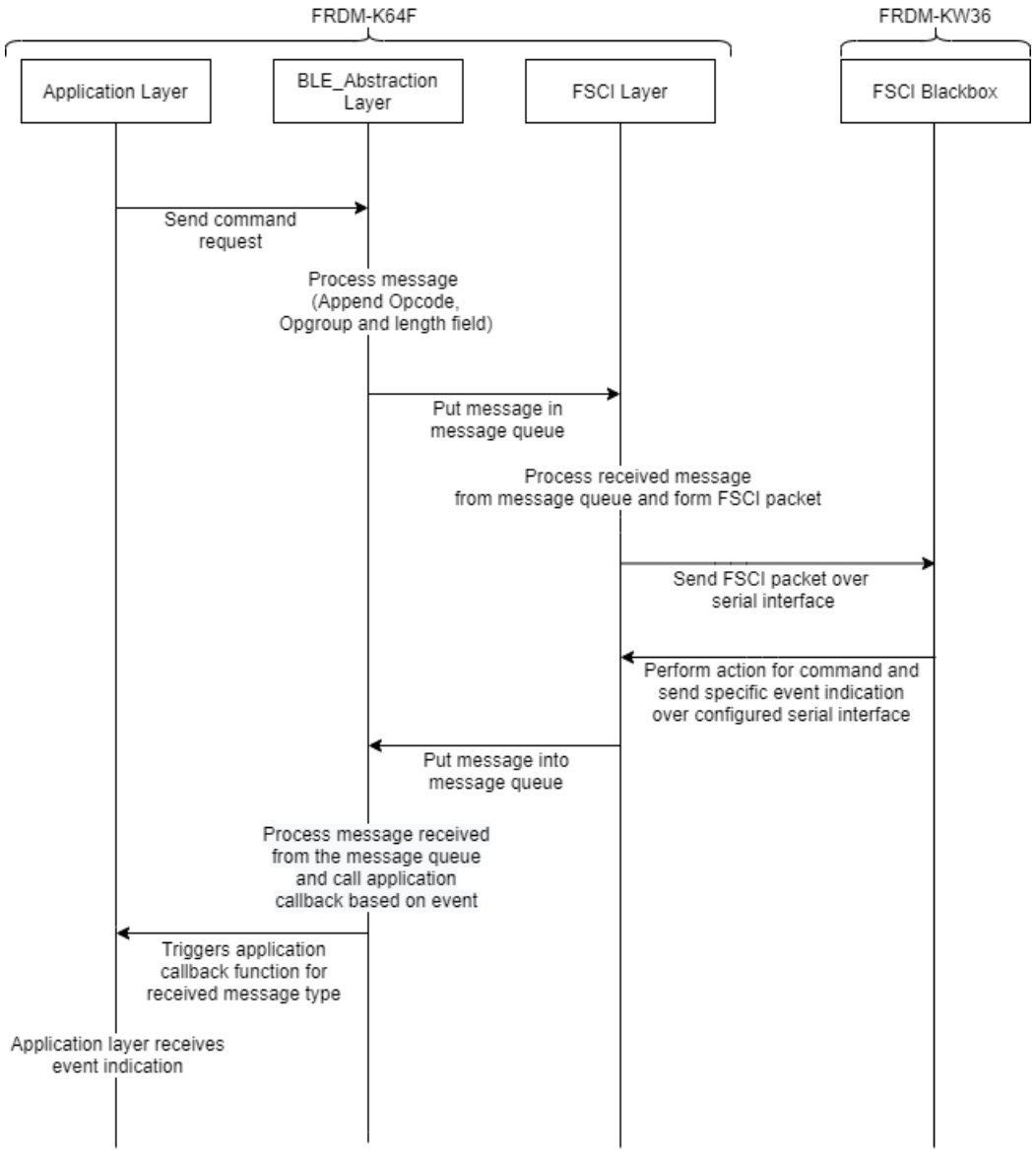

Figure 7. Communication between K64F (FSCI layers) and KW36

- FRDM-K64F application has three layers: Application layer, Bluetooth LE Abstraction layer, and FSCI layer.
- In this design, the application layer calls the APIs defined at the Bluetooth LE Abstraction layer. For example, when the advertising procedure is initiated, the application layer calls start advertising API (GAP_StartAdvertising) with the required parameters.
- Bluetooth LE Abstraction layer adds specific information (OpCode, OpGroup, and length field) and passes this information to the FSCI layer via a message queue.
- FSCI layer receives this information and forms the FSCI packet for a specific command. FSCI layer sends the FSCI packet to the FRDM-KW36 Blackbox via a configured serial interface (UART/SPI).
- FRDM-KW36 receives the FSCI packet, takes appropriate action, and sends a response to FRDM-K64F via serial interface.
- FRDM-K64F receives the response at the FSCI layer and passes the response to the Bluetooth LE Abstraction layer via a message queue.

- Bluetooth LE Abstraction layer triggers the callbacks which are defined at the application layer.

NOTE

This K64F FSCI temperature sensor application does not include security (Pairing and Bonding) feature.

5 FRDM-KW36 DCDC mode configuration

By default, the FRDM-KW36 Blackbox application operates with Buck mode at 1.8 V and FRDM-K64F application operates at 3.3 V, therefore there is a voltage level difference between the boards.

There are two methods to reconfigure FRDM-KW36 operating voltage at 3.0 V:

1. The main method is to increase the Buck mode output voltage to 3.0 V (only software change required). See Modification for Buck mode configuration.
2. The other method is to operate the FRDM-KW36 Blackbox application at 3.0 V by configuring the Bypass mode (software and hardware changes required). See Modification for Bypass mode configuration.

NOTE

Multiple ground connections are required as mentioned in Table 8 to get the correct data over the SPI bus. See Modifications for SPI interface to enable the SPI interface in the FRDM-KW36 FSCI Blackbox application.

Operating the FRDM-KW36 board at 3.0 V meets the required threshold for communication between the platforms, but if user uses the DCDC at 3.3 V, it is limited by the D8 diode.

If the user requires 3.3 V on FRDM-KW36, then provide 3.6 V external power supply and modify the DCDC configuration.

For more details on power management circuit in the FRDM-KW36, see FRDM-KW36 Freedom Development Board User's Guide section *Power management*.

5.1 Modification for Buck mode configuration

To operate FRDM-KW36 Blackbox in Buck mode at 3.0 V, perform the following changes in the software.

Software changes:

- In board/board.c file, modify DCDC output value from 1.8 V to 3.0 V as:

```
.dcdc1P8OutputTargetVal = gDCDC_1P8OutputTargetVal_3_000_c
```

- If `gDCDC_Enabled_d` is not defined in source/app_preinclude.h file, then add `gDCDC_Enabled_d` in the same file as:

```
#define gDCDC_Enabled_d 1
```

- In board/board.h file, make sure `APP_DCDC_MODE` macro is set to `gDCDC_Mode_Buck_c`.

5.2 Modification for Bypass mode configuration

To operate FRDM-KW36 Blackbox in Bypass mode at 3.0 V, perform the following changes in hardware and software.

Hardware changes:

- Short pins as shown in the following figure.
- Cut the traces between the pins as shown in the following figure.

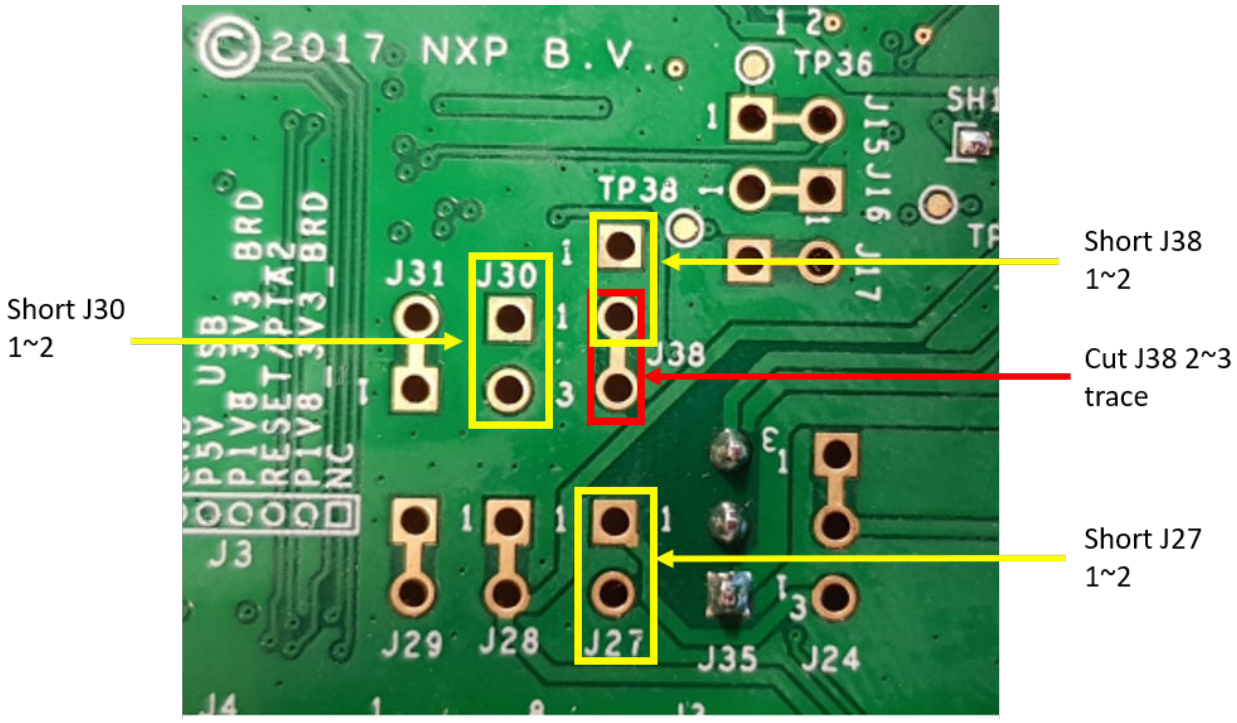

Figure 8. Bypass mode hardware changes

Software changes:

- In board/board.h file, update APP_DCDC_MODE as:

```
#define APP_DCDC_MODE (gDCDC_Mode_Bypass_c)
```

6 FRDM-KW36 FSCI Blackbox application

This section describes the software changes required for enabling UART and SPI interface on the FSCI Blackbox application.

6.1 Modifications for UART interface

- Update the UART instance inside source/app_preinclude.h file.

```
#define APP_SERIAL_INTERFACE_INSTANCE 1
```

- Add the pin muxing for UART inside BOARD_InitLPUART () function.

File: board/pin_mux.c

```
void BOARD_InitLPUART (void) {
CLOCK_EnableClock(kCLOCK_PortA);
PORT_SetPinMux(PORTA, PIN17_IDX, kPORT_MuxAlt3);
PORT_SetPinMux(PORTA, PIN18_IDX, kPORT_MuxAlt3);
SIM->SOPT5 = ((SIM->SOPT5 &
(~(SIM_SOPT5_LPUART1TXSRC_MASK | SIM_SOPT5_LPUART1RXSRC_MASK)))
| SIM_SOPT5_LPUART1TXSRC(SOPT5_LPUART0TXSRC_LPUART_TX)
| SIM_SOPT5_LPUART1RXSRC(SOPT5_LPUART0RXSRC_LPUART_RX));
}
```

Also, add the following defines in the same file.

```
#define PIN17_IDX 17u
#define PIN18_IDX 18u
```

6.2 Modifications for SPI interface

- By default in KW36 FSCI Blackbox application `gSerialMgrUseUART` macro is enabled, the user must disable it by the following macro in source/app_preinclude.h file.

  ```
  #define gSerialMgrUseUART_c 0
  ```

- To enable the SPI interface, user must update the following macro in source/app_preinclude.h file

  ```
  #define gSerialMgrUseSPI_c 1
  #define APP_SERIAL_INTERFACE_INSTANCE 1
  ```

- Add pin muxing for SPI interface inside BOARD_InitSPI() function.

File: board/pin_mux.c

```
void BOARD_InitSPI(void) {
CLOCK_EnableClock(kCLOCK_PortC);
PORT_SetPinMux(PORTA, 16u, kPORT_MuxAlt2);
PORT_SetPinMux(PORTA, 17u, kPORT_MuxAlt2);
PORT_SetPinMux(PORTA, 18u, kPORT_MuxAlt2);
PORT_SetPinMux(PORTA, 19u, kPORT_MuxAlt2);
}
```

7 Running the demonstration application

This section describes how to use the FRDM-K64F FSCI temperature sensor application with the FRDM-KW36 FSCI Blackbox application to communicate with FRDM-KW36 Bluetooth LE collector device.

For details regarding how FRDM-KW36 wireless application works, see Bluetooth Low Energy Demo Applications User Guide. Document is available inside FRDM-KW36 SDK at location: SDK_2.2.2_FRDM-KW36\docs\wireless\Bluetooth.

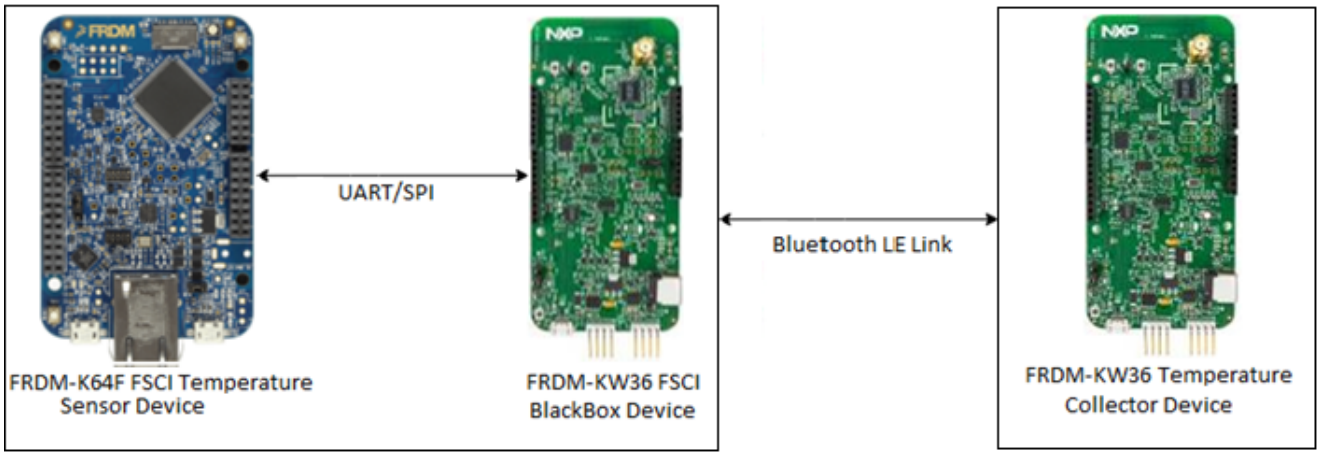

Figure 9. System setup

7.1 FRDM-K64F build configuration

This section describes how to edit the build configuration. The K64F application supports UART and SPI interfaces to communicate with FSCI Blackbox on the KW platform. Select one of the interfaces using the following steps.

1. Right click on the K64F FSCI temperature sensor project and select **Properties** option.
2. From Properties, navigate to **C/C++ Build** → **Settings** →**Tool Settings** →**Preprocessor**.
3. In the Preprocessor folder, add the macro `gSPISupported` to enable the SPI interface or the macro `gUARTSupported` to enable the UART interface. One interface is enabled at a time. By default, the SPI interface is enabled. The following figure shows the build configuration for the SPI interface.

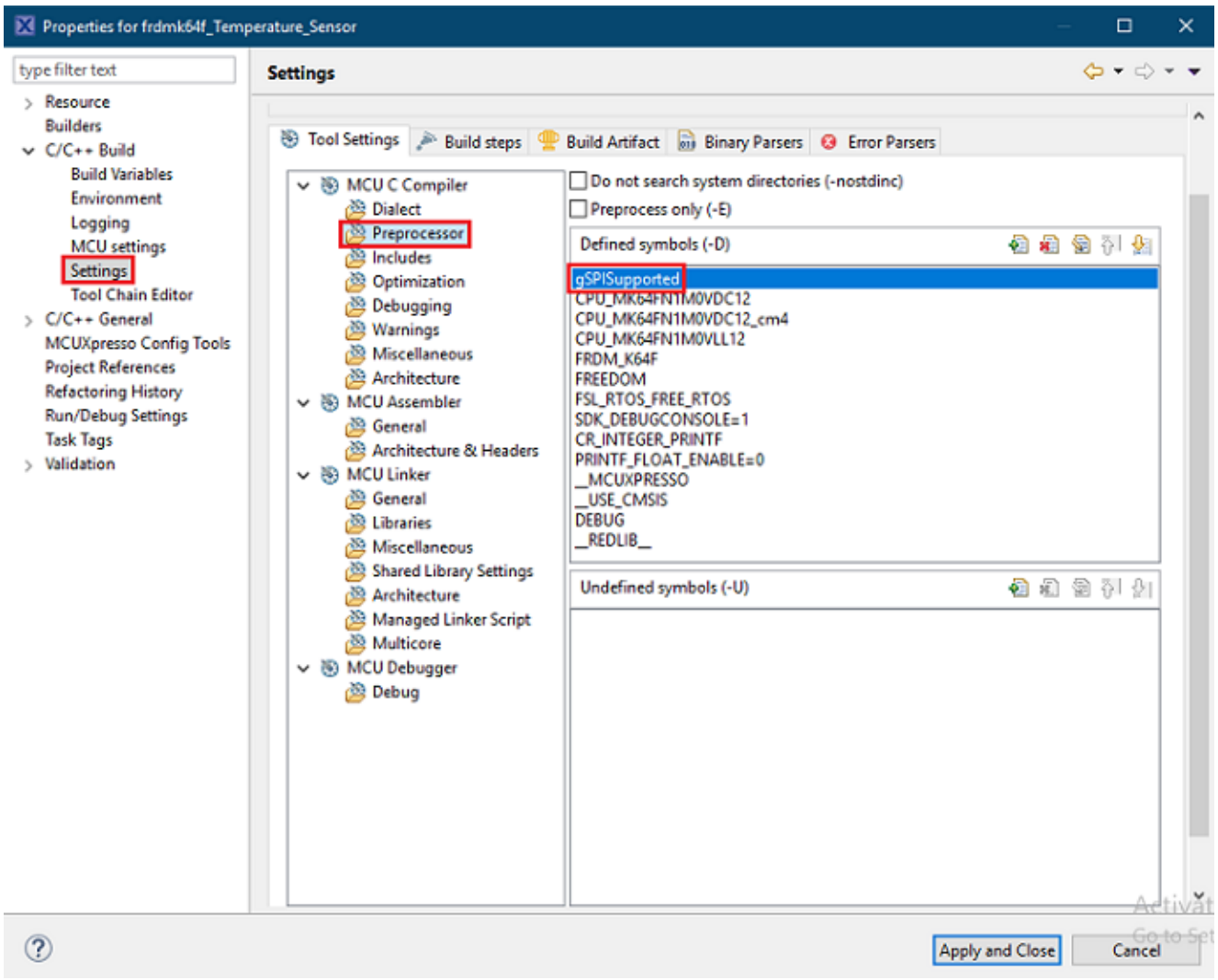

Figure 10. Update the build configuration

NOTE

The user can also enable the debug log by adding macro `DEBUG_PRINT_ENABLE` in the above preprocessor build configuration.

7.2 Loading the application

This section provides the detail for building the application and download it to the evaluation boards. The steps are as follows:

1. Open MCUXpresso by using an existing or a new workspace.
2. Import the SDKs for FRDM-K64F and FRDM-KW36, drag and drop SDK zip files to the Installed SDKs tab in MCUXpresso. Reference for downloading the SDK is provided in Development environment.
3. Import the project: navigate to **File →Import →General →Existing projects into Workspace**. Browse the project location and select the project to import as shown in the following figure.

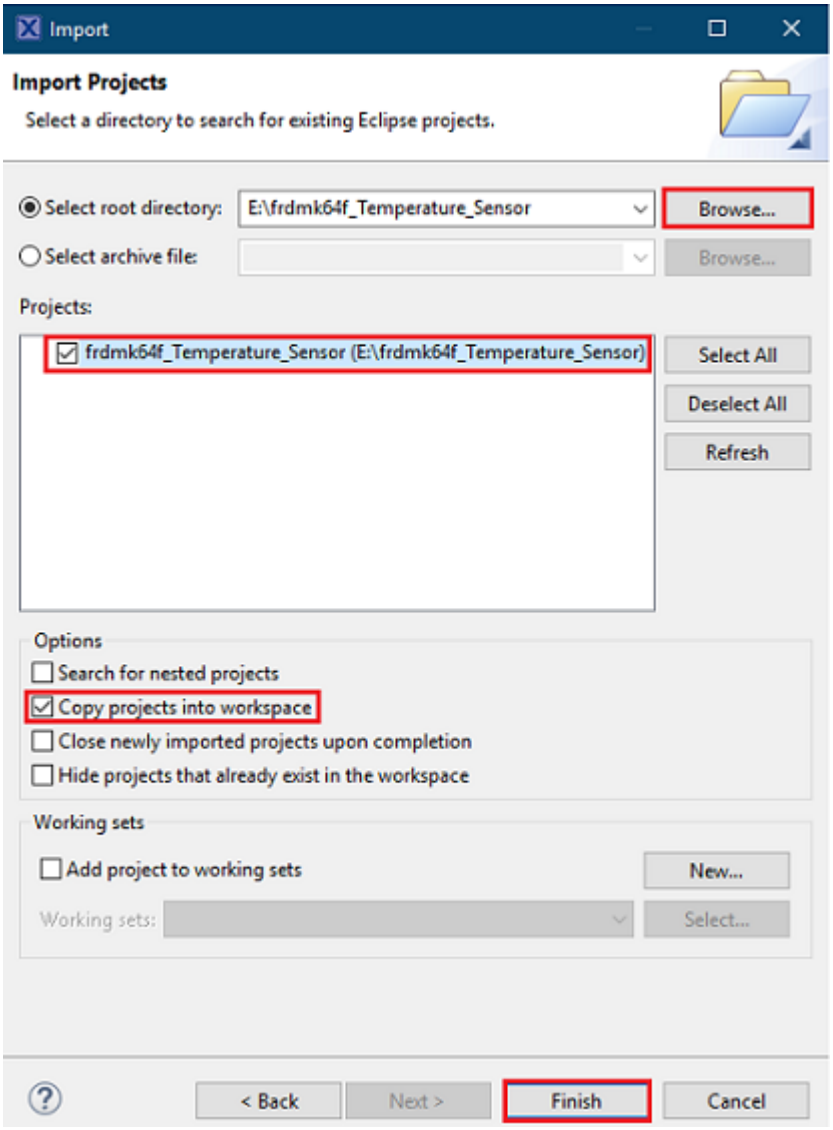

Figure 11. Import project to workspace

4. When the project is imported successfully, the project explorer tab in MCUXpresso displays the status.
5. Right click on the project and perform clean build by Clean Project option.
6. See FRDM-K64F build configuration to update the build configuration for the K64F project only.
7. Right click on the project and select the option Build Project.
8. In MCUXpresso console tab user can see the progress of the build. When the build is completed, the user can drag and drop the generated binary from the location `<MCUXpresso Workspace>/<Project>/Debug/` or use MCUXpresso to download and debug the application.

9. To download the binary, power up the hardware using a micro USB cable. When the device is connected, it shows as a device-specific drive on PC. Drag and drop the application binary to the same drive.
10. When the application is programmed to the board, then power-on reset the board to start and run the application.
11. User can download and debug the application using the Debug option available in the Quickstart Panel tab inside the MCUXpresso, as shown in the following figure.

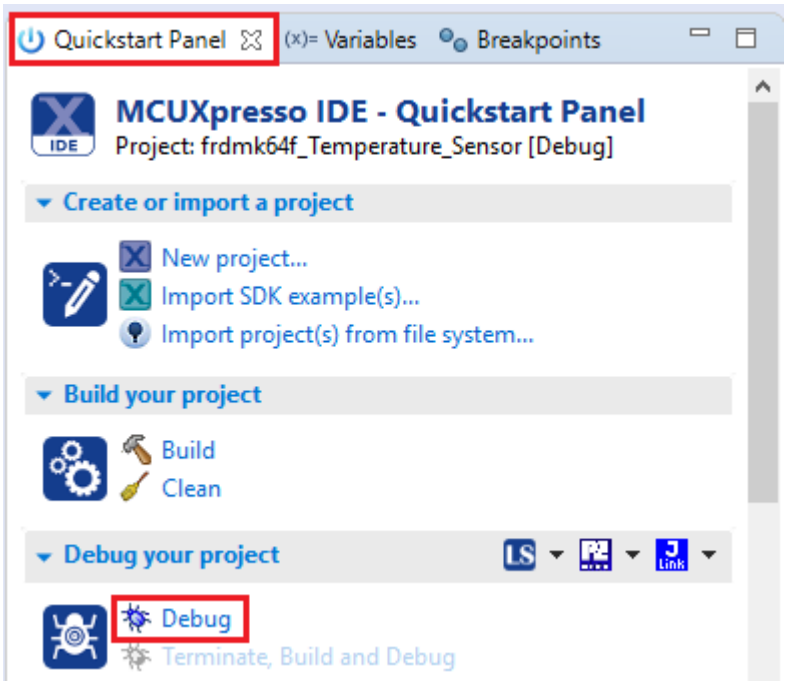

Figure 12. Download and debug option

12. After selecting the Debug option, MCUXpresso shows all the supported probes that are attached to your computer. Select the probe through which you want to debug and click OK button. See the following figure.

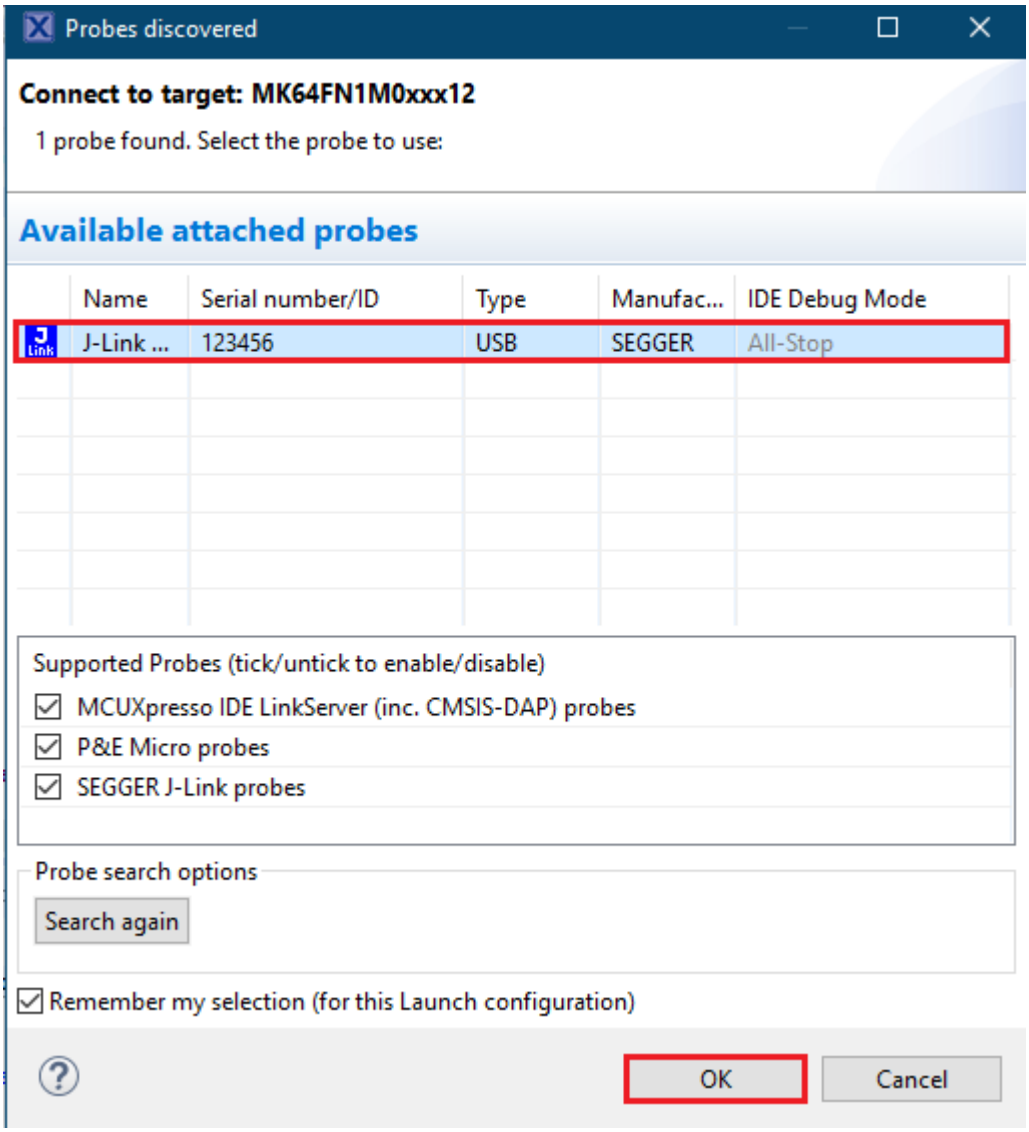

Figure 13. Attached probe list

13. When the probe is selected, the application is downloaded to the target and automatically runs to main() as shown in the following figure. To run the application, further use the Resume button.

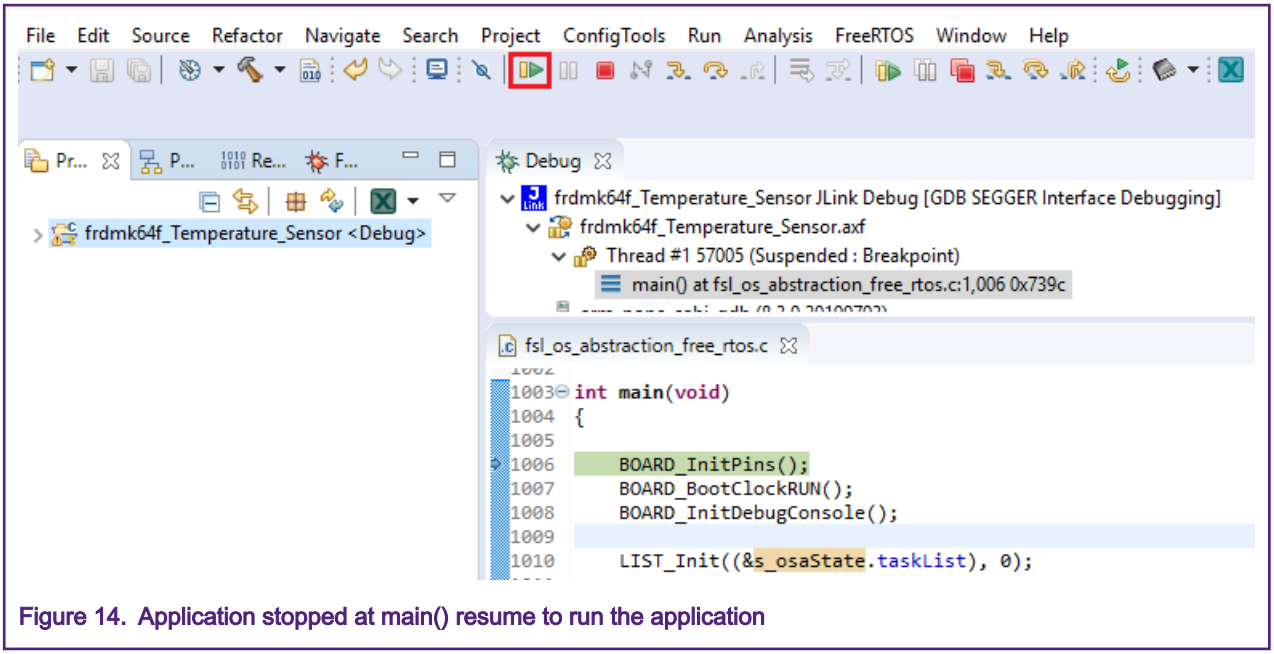

Figure 14. Application stopped at main() resume to run the application

For more details on using MCUXpresso IDE see Getting Started with MCUXpresso SDK for MKW36 Derivatives. Document is available inside FRDM-KW36 SDK at location: SDK_2.2.2_FRDM-KW36\docs.

The following table lists the applications that must be used with specified FRDM boards.

Table 6. Application and hardware module

Application	Project name	Hardware
Temperature sensor	frdmk64f_Temperature_Sensor	FRDM-K64F
	frdmkw36_wireless_examples_bluetooth_ble_fscibb_freertos	FRDM-KW36
Temperature collector	frdmkw36_wireless_examples_bluetooth_temp_coll_freertos	FRDM-KW36

7.3 Initialization of the temperature sensor application

FRDM-KW36 Bluetooth LE FSCI Blackbox application and FRDM-K64F FSCI temperature sensor application work together as a Bluetooth LE temperature sensor application. For communication between two boards of the required interface type, the user must do pin connections using jumper wires. Below are the details of the pin connections for both UART and SPI interfaces. User can configure any interface.

NOTE

For DCDC mode related changes, first see FRDM-KW36 DCDC mode configuration.

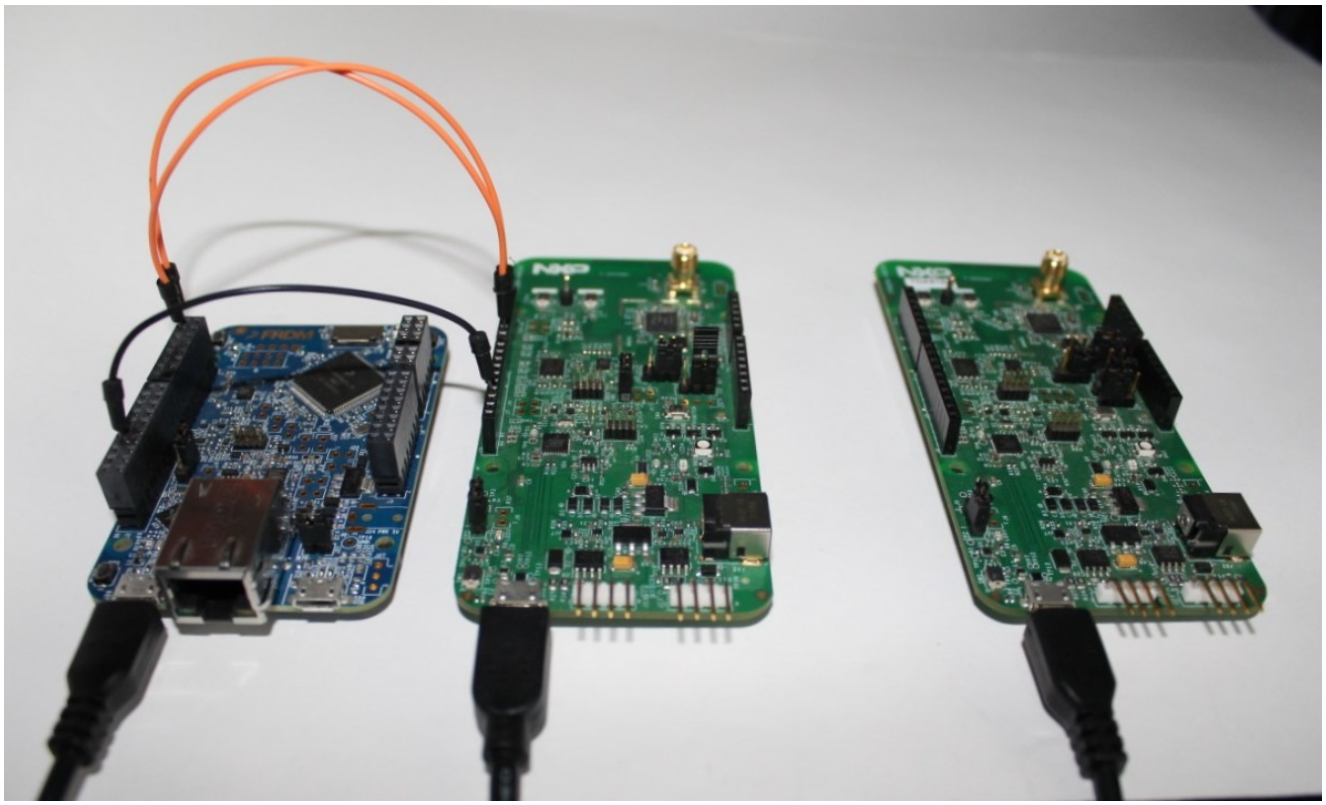

Figure 15. UART connections

Table 7. UART connection

Pin name	K64F jumper and pin	KW36 jumper and pin
UART-RX	J1-2	J1-5
UART-TX	J1-4	J1-7
GND	J2-14	J2-7

NOTE

Ensure that RX and TX pins are crossed between the FRDM-K64F FSCI and FRDM-KW36 FSCI Blackbox board.

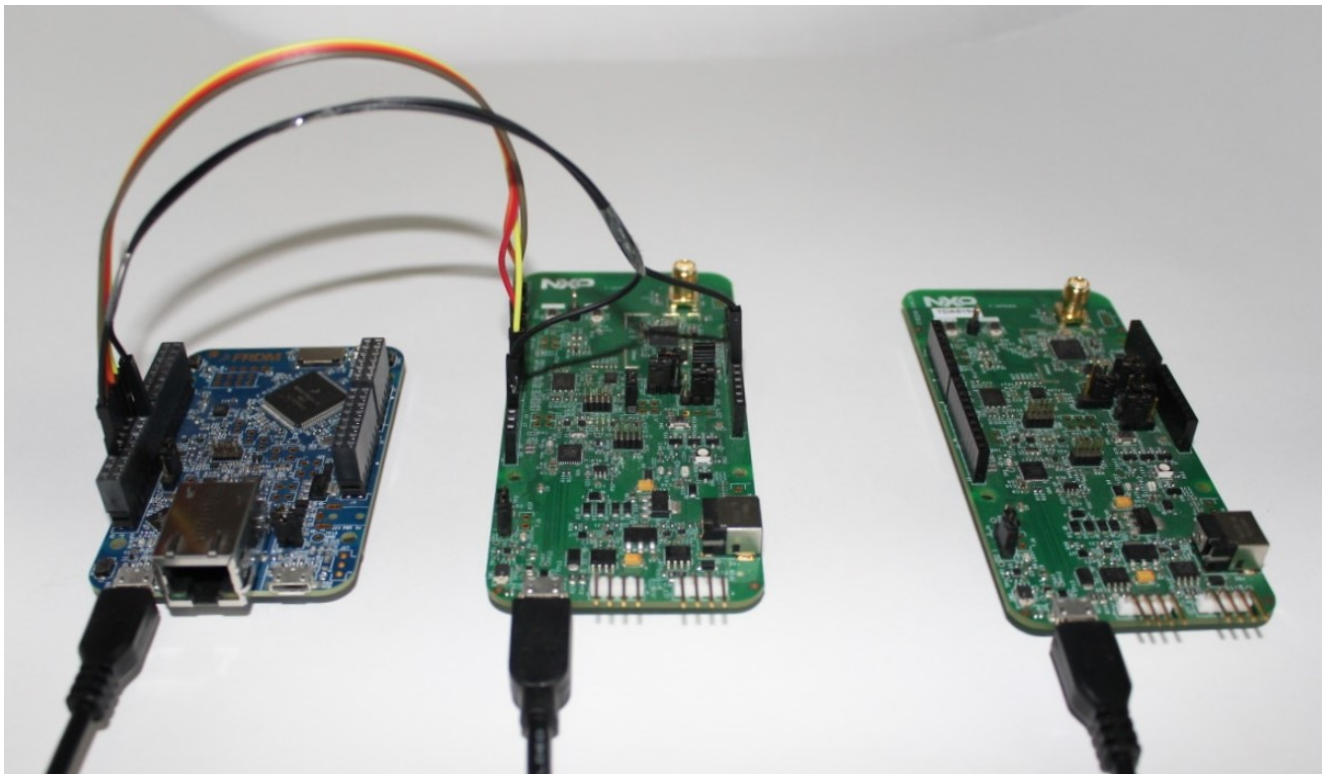

Figure 16. SPI connections

Table 8. SPI connection

Pin name	K64F jumper and pin	KW36 jumper and pin
CLK	J2-12	J1-7
CS	J2-6	J2-3
MOSI	J2-8	J1-5
MISO	J2-10	J2-1
GND	J2-3	J2-7
GND	J2-14	J3-7

When the above-mentioned pin connection is established between the FRDM-K64F and FRDM-KW36 boards, press the SW1 switch to reset the FRDM-KW36 device and FRDM-K64F device in sequence. Now, the FRDM-K64F temperature sensor application starts sending FSCI commands to initialize the Bluetooth LE custom profile using GATT-DB APIs. It also registers callback functions for GAP, advertisement, connection procedures, and generic events. When the initialization is completed on FRDM-K64F, RGB LED starts flashing white color. This RGB LED signals that the user can start the connection procedure as mentioned in Initiate Bluetooth LE connection between temperature sensor (K64F+KW36) and KW36 temperature collector device.

NOTE

To observe the output logs on the FRDM-K64F temperature sensor and FRDM-KW36 temperature collector devices, it requires board console-setup (or a terminal): 115200 Baud Rate, 8 bits, no parity, 1 stop bit, and no flow control.

7.4 Initiate Bluetooth LE connection between temperature sensor (K64F+KW36) and KW36 temperature collector device

Follow these steps to start connection between the K64+KW36 Bluetooth LE temperature sensor and the KW36 Bluetooth LE temperature collector.

1. To start scanning:
    a. When low-power mode is disabled, press SW2 from FRDM-KW36 temperature collector device.
    b. When low-power mode is enabled, press SW3 from FRDM-KW36 temperature collector device.

    NOTE

    User can enable or disable the Low-power mode by setting or clearing the `cPWR_UsePowerDownMode` macro defined in app_preinclude.h file (inside Source folder) of the temperature collector project.

2. When the scanning is started, '`scanning…`' message displayed on the KW36 temperature collector console as shown in Figure 18. It remains active for 10 seconds.
3. User must start advertising from the K64F temperature sensor application within 10 seconds duration to make successful connection with KW36 temperature collector device.
4. To start the advertisement from K64F temperature sensor application board press SW3.
5. When the advertisement starts '`---------- ADVERTISEMENT STARTED ----------`' message displayed on K64F FSCI temperature sensor console as shown in Figure 19. Advertisement remains active for 30 seconds.
6. RGB LED on K64F board flashes with RED color to indicate advertisement in progress.
7. When the connection is successfully established between the two Bluetooth LE devices, then the RGB LED on K64F board remains steady with RED color to indicate connected.
8. On the console of Bluetooth LE collector device and also on K64F device, the message '`Connected!`' and '`---------- CONNECTED ----------`' is displayed respectively.

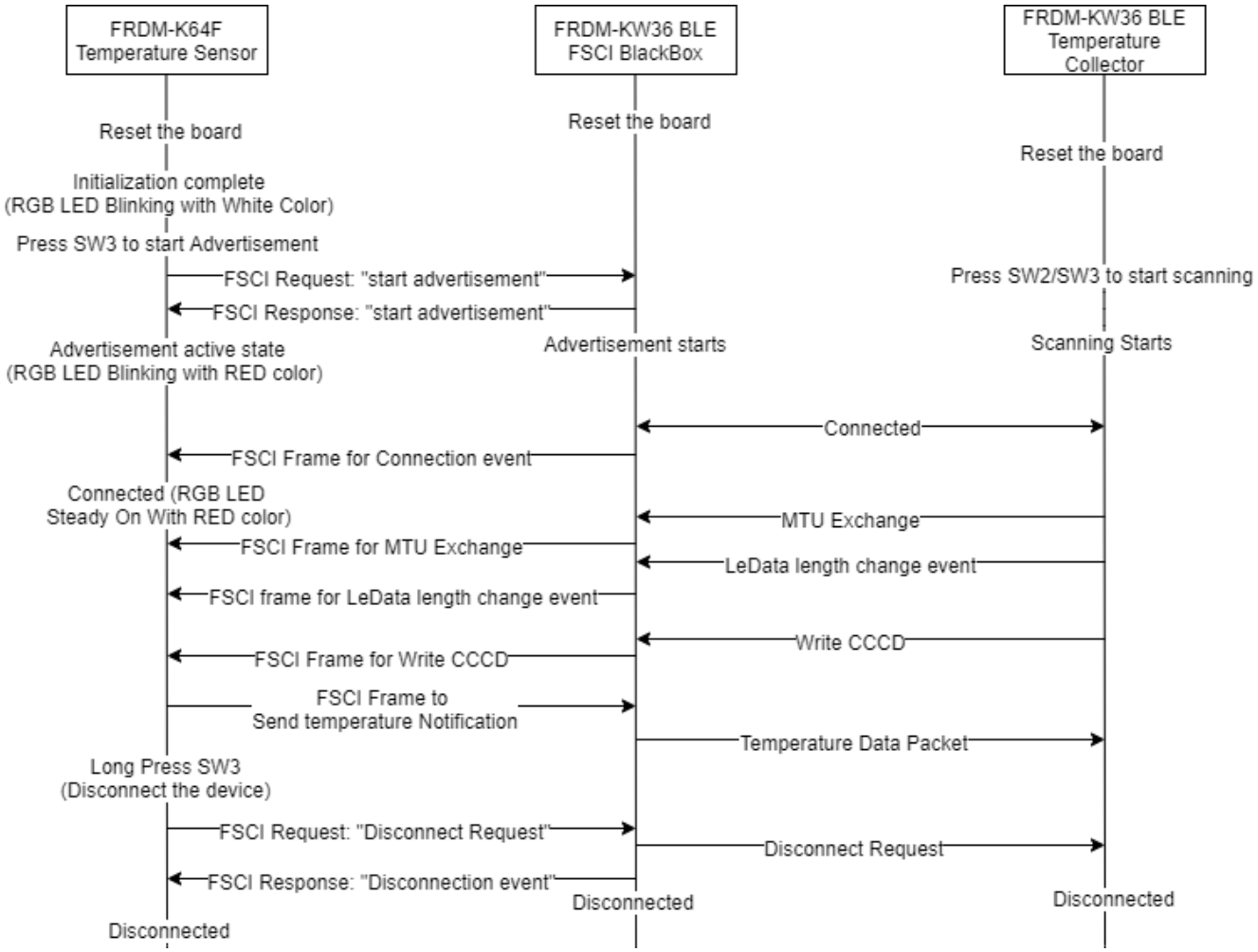

Figure 17. FSCI message flow diagram

When both devices are connected, the temperature sensor application system is ready to send temperature data as described in Temperature value sending mechanism. The above figure shows the message flow between the boards.

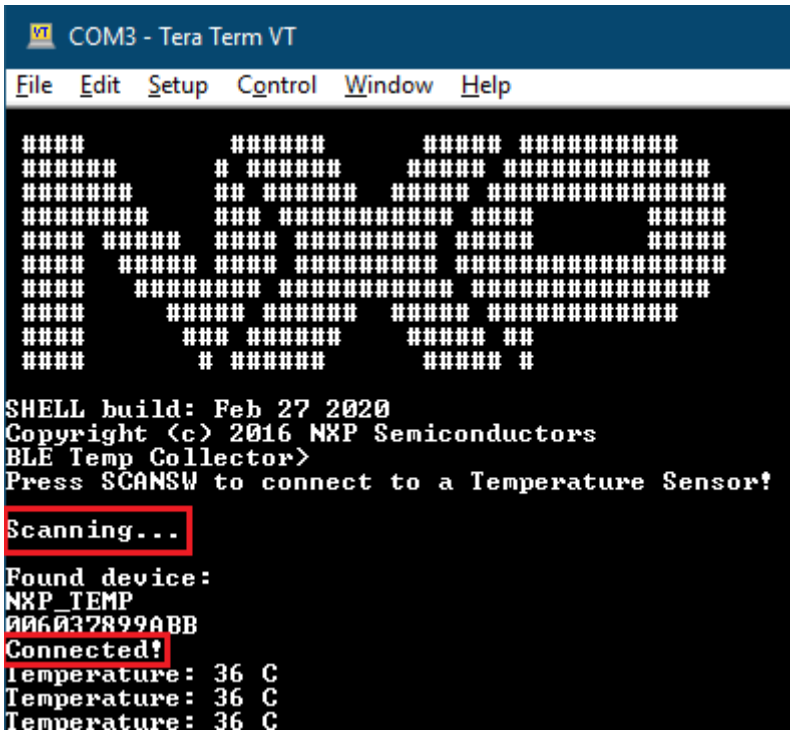

Figure 18. FRDM-KW36 temperature collector device console

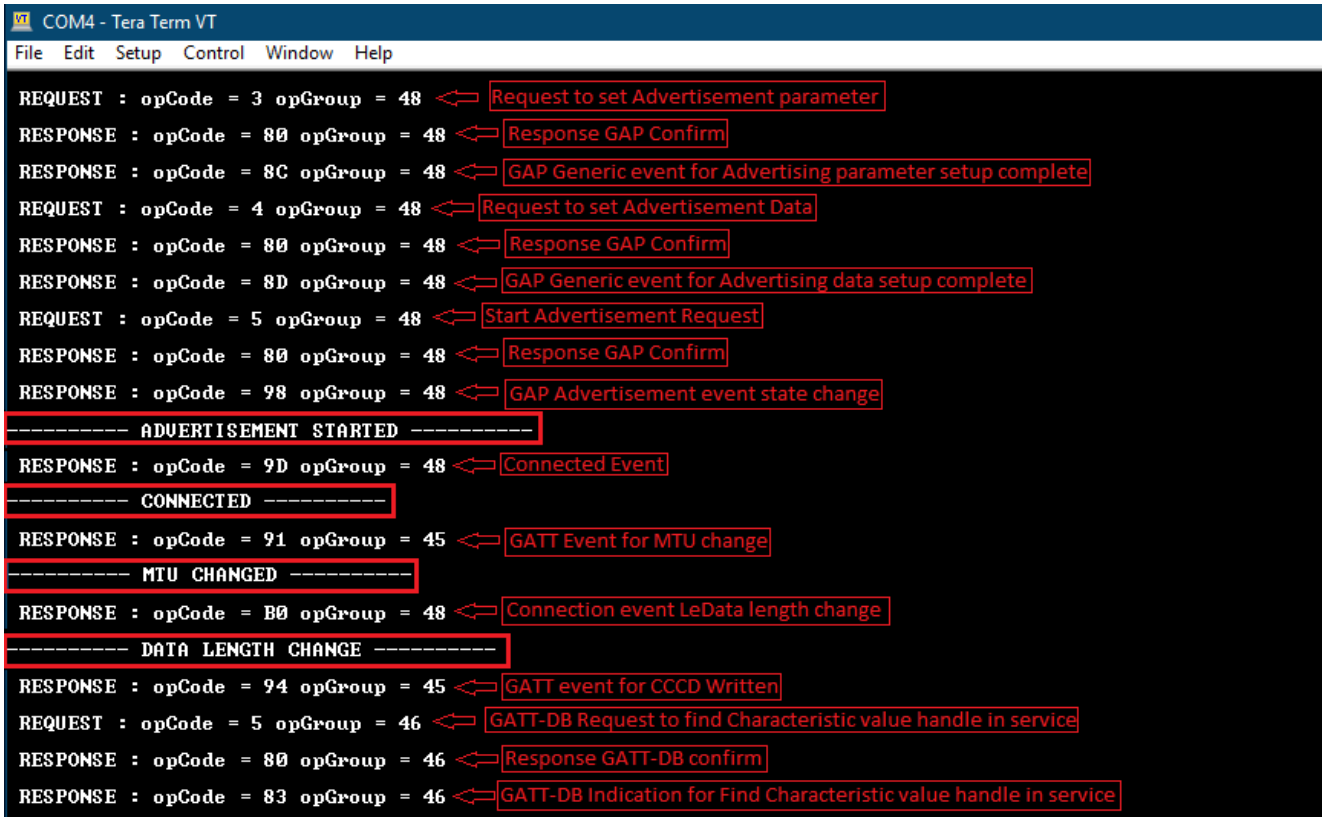

Figure 19. FRDM-K64F temperature sensor device console

NOTE

For more detailed information about OpCode & OpGroup, see Bluetooth Low Energy Host Stack FSCI Reference Manual. Document is available inside FRDM-KW36 SDK at location: SDK_2.2.2_FRDM-KW36\docs\wireless \Common.

Enumeration and structures describes the enumerations used for opCode and opGroup in this application.

7.5 Temperature value sending mechanism

FRDM-K64F FSCI temperature sensor application reads temperature from the sensor available on the FRDM-K64F board using ADC. The temperature value is written over FSCI into GATT-DB maintained by the FRDM-KW36 Blackbox application. So, whenever both devices get connected and the CCCD event is received from the FRDM-KW36 board (Blackbox application), the K64F FSCI temperature sensor application uses `SendNotification()` method to send temperature data to FRDM-KW36 temperature collector device. There are two methods for sending temperature data to the collector device.

1. **Manual:** Press the SW3 button on the FRDM-K64F board.
2. **Periodic:** At every 3 seconds interval, the temperature value is sent.

Both methods are enabled by default in the K64F FSCI temperature sensor application.

User can disable the Periodic method by commenting `gTempTimerSupported` macro defined in ApplMain.c.

When the temperature sensor sends temperature value to the temperature collector device user can see the temperature value print on the collector console.

7.6 Disconnection

To disconnect both the connected devices use one of the following methods:

1. Disconnect using temperature data receive time out

   If KW36 temperature collector does not receive any temperature data for more than 5 seconds, it sends a disconnection request to the temperature sensor device. This method is available only when the KW36 temperature collector application has low-power mode enabled. To Enable or Disable low-power mode see Initiate Bluetooth LE connection between temperature sensor (K64F+KW36) and KW36 temperature collector device. If low-power mode is disabled, then the KW36 temperature collector device does not send a disconnect request.

2. Manual disconnect

   K64F FSCI temperature sensor application can send a disconnect request on long press more than 1 second of the SW3 button at the FRDM-K64F board. By default, this method is enabled in the K64F FSCI temperature sensor application.

When a disconnection is completed, user can see a '`Disconnected!`' and a '`---------- DISCONNECTED ----------`' messages on the terminal consoles of FRDM-KW36 temperature collector device and FRDM-K64F Temperature sensor device respectively.

See Bluetooth Low Energy Demo Applications User Guide for more details about Temperature Sensor and Temperature Collector application example. Document is available inside FRDM-KW36 SDK at location: SDK_2.2.2_FRDM-KW36\docs\wireless \Bluetooth.

8 Enumeration and structures

The list of various request response enumerations are described in this section. `Source\ BLE_Abstraction \ble_FSCI.h` file located inside BluetoothLE_Abstraction folder contains details of numerical values of the mentioned enumerations.

- typedef enum bleOpGroupType_t

  Enumeration for opGroup field.

Table 9. opGroup

Enumeration	opGroup
bleGATT	GATT messages
bleGATTDB	GATT-DB messages
bleGAP	GAP messages

- typedef enum bleGATTDBOpCodeType_t

  Enumeration for opCode used for GATT-DB layer messages.

Table 10. GATT-DB request opCode

Enumeration	GATT-DB opCode
gattDBWriteAttribute	GATT-DB write attribute request
gattDBReadAttribute	GATT-DB read attribute request
gattDBFindServiceHandle	GATT-DB find service handle request
gattDBFindCharValueHandleInService	GATT-DB request to find characteristic value handle in service
gattDBFindCccdHandleForCharValHandle	GATT-DB request to find CCCD handle for characteristic value
gattDBFindDescHandleForCharvalHandle	GATT_DB request to find Descriptor handle for the characteristic value
gattDBInit	GATT-DB initialization request
gattDBReleaseDB	GATT-DB release request
gattDBAddPrimaryService	GATT-DB request to add primary service into GATT database
gattDBAddSecondaryService	GATT-DB request to add secondary service into GATT database
gattDBAddCharDescAndValue	GATT-DB request to add characteristic descriptor and value into GATT database
gattDBAddCharDescriptor	GATT-DB request to add characteristic descriptor into GATT database
gattDBAddCccd	GATT-DB request to add CCCD into GATT database

- typedef enum bleGATTOpCodeType_t

  Enumeration for opCode used for GATT messages.

Table 11. GATT request opCode

Enumeration	GATT opCode
gattServerCallbackRegister	GATT server callback registration request
gattSendNotification	Send notification request

- typedef enum bleGAPOpCodeType_t

  Enumeration for opCode used for GAP messages.

Table 12. GAP request opCode

Enumeration	GAP opCode
gapSetAdvParameter	GAP request to set advertisement parameter
gapSetAdvData	GAP request to set advertisement data
gapStartAdvertisement	GAP request to start advertisement
gapStopAdvertisement	GAP request to stop the advertisement
gapCheckNotificationStatus	GAP request to check the notification status
gapDisconnect	GAP to send disconnect request

- typedef enum bleGapRespOpCodeType_t

  Enumeration for opCode used for GAP response messages.

Table 13. GAP response opCode

Enumeration	GAP response opCode
gapConfirm	GAP confirmation message
gapCheckNotfStatusResp	GAP check notification status response
gapAdvDataSetupComp	GAP Advertisement data set up complete event
gapAdvParamSetupComp	GAP advertisement parameter setup complete
gapAdvSetupFailed	GAP advertisement setup failed event
gapAdvEventStateChanged	GAP advertisement event state change event
gapAdvEventFailed	GAP advertisement event fail event
gapConnectionEventConnected	GAP connection event connected
gapConnectionEventDisConnected	GAP connection event disconnected
gapConnEventLEDataLenChanged	GAP connection event LE data length change event

- typedef enum bleGATTRespOpCodeType_t

  Enumeration for opCode used for GATT response messages.

Table 14. GATT response opCode

Enumeration	GATT response opCode
gattConfirm	GATT confirmation message
gattMTUChanged	GATT response for the MTU change
gattCharCccdWritten	GATT event for the CCCD written

- typedef enum bleGATTDBRespOpCodeType_t

Enumeration for opCode used for GATT-DB response messages.

Table 15. GATT-DB response opCode

Enumeration	GATT-DB response opCode
gattDBConfirm	GATT-DB confirmation message
gattDBAddPrimServResp	GATT-DB add primary service response
gattDBAddSecServResp	GATT-DB add secondary service response
gattDBAddCharDeclandValResp	GATT-DB add characteristic declaration and value response
gattDBAddCharDescResp	GATT-DB add characteristic descriptor response
gattDBAddCccdResp	GATT-DB add CCCD response
gattDBReadAttrResp	GATT-DB read attribute response
gattDBFindSerHandleResp	GATT-DB find service handle response
gattDBFindCharValHandleResp	GATT-DB find characteristic value handle response
gattDBFindCccdHandleResp	GATT-DB find CCCD handle response
gattDBFindDescHandleResp	GATT-DB find descriptor handle response

- typedef struct msgBLEabsToFSCI_tag

  This structure is used to send data from the Bluetooth LE Abstraction layer to FSCI layer.

  uint8_t opGroup: opGroup for the request to send.

  uint8_t opCode: opCode for the request to send.

  uint8_t length: Length of bleRequestType data.

  Union bleRequestType:

Table 16. FSCI request structure

Payload structure	Description
gattDiscoverAllCharacteristicDescriptorReq_t	Discover all the characteristic descriptor
gattServerSendNotificationReq_t	Send notification request
gapAdvertisingParameters_t	Set advertisement parameter request
gapSetAdvertisingData_t	Set advertisement data request
gapScanResponseData_	Scan response data
gapCheckNotificationStatus_t	Check notification status request
gapDisconnect_t	Disconnect request
gattDBAddPrimaryServiceReq_t	Add primary service request

Table continues on the next page...

Table 16. FSCI request structure (continued)

Payload structure	Description
gattDBAddSecondaryServiceReq_t	Add secondary service request
gattDBAddCharDecandValueReq_t	Add characteristic declaration and value request
gattDBAddCharacDescriptorReq_t	Add characteristic descriptor request
gattDBWriteAttributeReq_t	Write attribute request
gattDBReadAttributeReq_t	Read attribute request
gattDBFindSeriveHandleReq_t	Find service handle request
gattDBFindCharValueHandleInServiceReq_t	Find characteristic value handle in service
gattDBFindCccdHandleForCharValueHandleReq_t	Find CCCD handle for the characteristic value handle request
gattDBFindDescHandleForCharValueHandleReq_t	Find descriptor handle for the characteristic value handle request

The union shows already implemented requests, if user wants to add any other request then edit the above structure and relative enumeration.

- typedef struct FSCIResponseMsg_tag

  This structure is used by the Bluetooth LE Abstraction layer to decode and process the response received.

  uint8_t sync: Frame start markup.

  uint8_t opGroup: opGroup of the received response.

  uint8_t opCode: opCode for the received response.

  uint16_t length: Length of the received response payload.

  union ResponseType:

Table 17. FSCI response structure

Payload structure	Description
gapCheckNotfStatusResp_t	Check notification status response
gapAdvEventFailed_t	Advertisement event failed response
GapConfirm_t	GAP confirmation message
GattConfirm_t	GATT confirmation message
gattDBConfirm_t	GATT-DB confirmation message
gattDBAddPrimServResp_t	GATT-DB add primary service response
gattDBAddSecServResp_t	GATT-DB add secondary service response

Table continues on the next page...

Table 17. FSCI response structure (continued)

Payload structure	Description
gattDBAddCharDeclandValResp_t	GATT-DB add characteristic declaration and value response
gattDBAddCharDescResp_t	GATT-DB add characteristic descriptor response
gattDBAddCccdResp_t	GATT-DB add CCCD response
gattDBReadAttrResp_t	GATT-DB read attribute response
gattDBFindServHandleResp_t	GATT-DB find service handle response
gattDBFindCharValHandleResp_t	GATT-DB find characteristic value handle response
gattDBFindCccdHandleResp_t	GATT-DB find CCCD handle response
gattDBFindDescHandleResp_t	GATT-DB find descriptor handle response

uint8_t crc: checksum of the response data.

When a new request is added, the user must update the above structure and relative enumeration for that request related response handling.

9 Related documents

This section provides the list of the documents which are referred in the development of K64F FSCI temperature sensor application. Documents are available inside FRDM-KW36 SDK at locations: SDK_2.2.2_FRDM-KW36\docs\wireless\Bluetooth and SDK_2.2.2_FRDM-KW36\docs\wireless\Common.

1. Bluetooth Low Energy Host Stack FSCI Reference Manual
2. Bluetooth Low Energy Application Developer Guide
3. Bluetooth Low Energy Demo Applications User Guide
4. Bluetooth Low Energy Host Stack API Reference Manual

10 Revision history

Table 18. Revision history

Revision number	Date	Substantive changes
0	06/2020	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.

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

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

Date of release: June 2020

Document identifier: AN12896