MtM+ Technology

MtConnect08

LoRaWAN SDK User Guide

GUIDELINE

6th September, 2017

Revision HISTORY

Revision	**Date**	**Author**	**Reason**
V0.0.1	2017/02/16	Ray Tsou	New Create (draf)
V0.0.2	2017/03/01	Ray Tsou	Add Chapter 3 “Debugging with LoRaWAN Example”
V0.0.3	2017/03/08	Ray Tsou	Correct Chapter2-3 step6
V0.0.4	2017/4/16	Ray Tsou	Update LoRaSDK version to V02. - LoRaSDK V02 support LoRa SPEC V1.0.2 Add used GPIO definition..
V0.0.5	2017/4/20	Ray Tsou	Add an Example for Custom Channel in section 2-4-2
V0.0.6	2017/5/10	Ray Tsou	Update LoRaSDK QMSI verion from 1.1 to 1.4. Update ISSM version from 2016.1.067 to 2016.2.090. Update 1-1-2 Features. - Add Compliance Test Status. Add 1-2-5 : LoRa Mac Information Base (MIB) list Add 2-5 : LoRaWAN Example flow chart
V0.0.7	2017/08/11	Ray Tsou	Update LoRaSDK Receive windows timing. End Node receive data more accurately.
V0.0.8	2017/8/25	Ray Tsou	Update LoRaSDK to Version 04. EU868 supported LoRaWAN Regional Parameters v1.0.2b Know issue : US915 Receive window DR11~DR13 have problem. Suggest use default datarate. Disable ADR defined.
V0.0.9	2017/09/06	Ray Tsou	Update LoRaSDK to Version 05. Fix issue: US915 Recevie windows problem fixed.

1.Introduction

The aim of this example is to show an project of the endpoint LoRaWAN stack implementation.

This LoRaWAN stack is an EU868 and US915 bands Class A and Class C endpoint implementation
fully compatible with LoRaWAN 1.0.2 specification.
Each LoRaWAN application example includes the LoRaWAN certification protocol implementation.

SX1276 radio drivers are also provided.
In case only point to point links are required a Ping-Pong application is provided as example.

The chapter 1: "LoRaWAN stack API" descript how to use LoRaWAN stack API.
The chapter 2: "LoRaWAN Example" descript how to use this example.
The chapter 3: "Debugging with LoRaWAN Example " descript how to Debug with ISSM.

2. LoRaWAN stack API

2.1. Preface

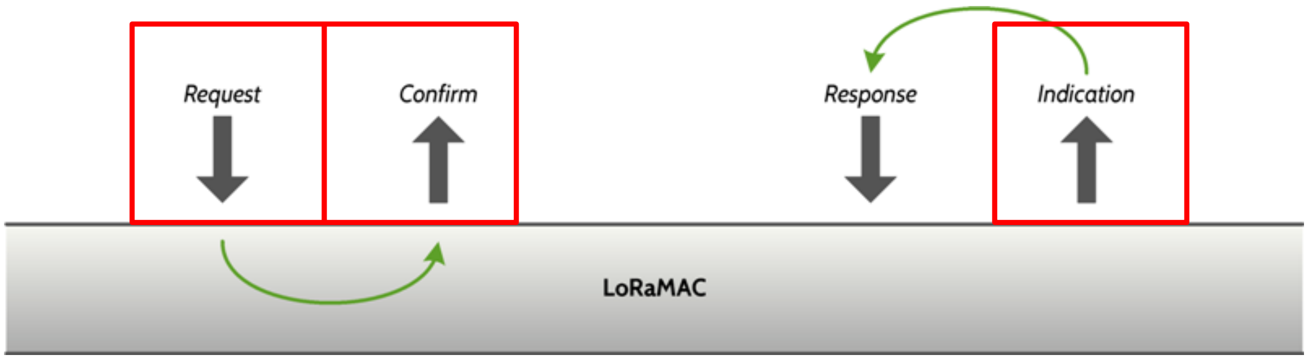
2.1.1. Architecture

The concept of the API follows the concept of primitives of the IEEE Standard for local and metropolitan area networks — part 15.4: Low-Rate Wireless Personal Area Networks (IEEE802.15.4-2011).

The upcoming figure shows the concept of primitives of the LoRaMAC layer. The concept follows the Request-Confirm and Indication-Response architecture. The LoRaMAC layer offers MCPS (MAC Common Part Sublayer) services, MLME (MAC layer management entity) services and a MIB (MAC information base).

In general, the LoRaMAC layer utilizes MCPS services for data transmissions and data receptions, and MLME services to manage the LoRaWAN network. The MIB is responsible to store important runtime information and holds the configuration of the LoRaMAC layer.

An upper layer (e.g. the main application) is able to perform Requests which the LoRaMAC layer confirms with a Confirm primitive. The LoRaMAC layer notifies an upper layer with the Indication primitive in case of events. The upper layer may responses to an Indication with the Response primitive.



2.1.2. Features

The following features are supported according to the LoRaWAN 1.0.2 specification:

LoRaWAN device classes :

Class A/C

Class B (TBD.)

ISM Band support :

EU868 / US915 / CN779 / EU433 / AU915

CN470 / AS923 / KR920 / IN865

Compliance Test Status:

EU868	Pass	
US915	Pass	
CN779	In Progress	-
EU433	In Progress	-
AU915	In Progress	-
CN470	In Progress	-
AS923	In Progress	-
KR920	In Progress	-
IN865	In Progress	

2.2. Quick-Start API

2.2.1. Overview

This section provides a short introduction into important operations of the LoRaMAC layer. Full examples for the different device classes are available in the example section.

2.2.2. Initialization

The initialization function of the LoRaMAC layer is **LoRaMacInitialization**(**LoRaMacPrimitives_t *primitives, LoRaMacCallback_t *callbacks**). The function has two parameters, **LoRaMacPrimitives_t** and **LoRaMacCallback_t**. The parameters define function which have to be defined by upper layers. The first parameter contains function pointers to the Confirm and Indication primitives, and the second parameter contains a function pointer to the function which measures the battery level. This function is available in the board support package.

The following code snippet provides an example for the initialization:

```
static void McpsConfirm( McpsConfirm_t *McpsConfirm )
{
   // Implementation of the MCPS-Confirm primitive
}
static void McpsIndication( McpsIndication_t *McpsIndication )
{
   // Implementation of the MCPS-Indication primitive
}
static void MlmeConfirm( MlmeConfirm_t *MlmeConfirm )
{
   // Implementation of the MLME-Confirm primitive
}
LoRaMacPrimitives_t LoRaMacPrimitives;
LoRaMacCallback_t LoRaMacCallbacks;
LoRaMacStatus_t Status;
int main( void )
{
   LoRaMacPrimitives.MacMcpsConfirm = McpsConfirm;
   LoRaMacPrimitives.MacMcpsIndication = McpsIndication;
   LoRaMacPrimitives.MacMlmeConfirm = MlmeConfirm;
   Status = LoRaMacInitialization( &LoRaMacPrimitives, &LoRaMacCallbacks );
   if( Status == LORAMAC_STATUS_OK )
   {
      // Initialization successful
   }
}
```

2.2.3. Over-the-air actvation

The LoRaMAC layer provides a MLME-Request to perform the over-the-air activation (**LoRaMacMlmeRequest(MlmeReq_t *mlmeRequest)**). The LoRaMAC layer creates the join-request message and reports the status of the operation with the *MacMlmeConfirm* primitive.

The following code snippet provides an example:

```
static uint8_t DevEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
static uint8_t AppEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
static uint8_t AppKey[] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA,
0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x00 };
static void MlmeConfirm( MlmeConfirm_t *MlmeConfirm )
{
        if( MlmeConfirm->Status == LORAMAC_EVENT_INFO_STATUS_OK )
        {
                switch( MlmeConfirm->MlmeRequest )
                {
                        case MLME_JOIN:
                        {
                                // Status is OK, node has joined the network
                                break;
                        }
                        case MLME_LINK_CHECK:
                        {
                                // Check DemodMargin
                                // Check NbGateways
                                break;
                        }
                        default:
                                break;
                }
        }
}
```

```
int main( void )
{
    MlmeReq_t mlmeReq;
    LoRaMacStatus_t status;
    // This comment is a placeholder for the initialization of the LoRaMAC layer
    // Setup the request type
    mlmeReq.Type = MLME_JOIN;
    // Fill the join parameters
    mlmeReq.Req.Join.DevEui = DevEui;
    mlmeReq.Req.Join.AppEui = AppEui;
    mlmeReq.Req.Join.AppKey = AppKey;
    status = LoRaMacMlmeRequest( &mlmeReq );
    if( status == LORAMAC_STATUS_OK )
    {
        // Join request was send successfully
    }
}
```

2.2.4. Activation by personalization

The activation by personalization procedure requires to store important information on the LoRaMAC node:

Network Identifier
Device Address
Network session key
Application session key
The application must configure those parameters to perform a personalized activation. In addition, the application has to enable the LoRaWAN network itself.

The following code snippet provides an example:

```
static uint8_t NwkSKey[] = { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB,
0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };
static uint8_t AppSKey[] = { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB,
0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };
int main( void )
{
  MibRequestConfirm_t mibReq;
  LoRaMacStatus_t status;
  uint32_t DevAddr;
  // This comment is a placeholder for the initialization of the LoRaMAC layer
  // Choose a random device address
  DevAddr = randr( 0, 0x01FFFFFF );
  mibReq.Type = MIB_NET_ID;
  mibReq.Param.NetID = 0x000000;
  if( LoRaMacMibSetRequestConfirm( &mibReq ) != LORAMAC_STATUS_OK )
    return;
  mibReq.Type = MIB_DEV_ADDR;
  mibReq.Param.DevAddr = DevAddr;
  if( LoRaMacMibSetRequestConfirm( &mibReq ) != LORAMAC_STATUS_OK )
    return;
  mibReq.Type = MIB_NWK_SKEY;
  mibReq.Param.NwkSKey = NwkSKey;
  if( LoRaMacMibSetRequestConfirm( &mibReq ) != LORAMAC_STATUS_OK )
    return;
    mibReq.Type = MIB_APP_SKEY;
    mibReq.Param.AppSKey = AppSKey;
    if( LoRaMacMibSetRequestConfirm( &mibReq ) != LORAMAC_STATUS_OK )
        return;
    mibReq.Type = MIB_NETWORK_JOINED;
    mibReq.Param.IsNetworkJoined = true;
    if( LoRaMacMibSetRequestConfirm( &mibReq ) != LORAMAC_STATUS_OK )
        return;
    // LoRaWAN node has joined the network
}
```

2.2.5. LoRa Mac Information Base (MIB) list

The following table lists the MIB parameters and the related attributes:

Attribute	Description	Get	Set
MIB_DEVICE_CLASS	LoRaWAN device class	Yes	Yes
MIB_NETWORK_JOINED	LoRaWAN Network joined attribute	Yes	Yes
MIB_ADR	Adaptive data rate	Yes	Yes
MIB_NET_ID	Network identifier	Yes	Yes
MIB_DEV_ADDR	End-device address	Yes	Yes
MIB_NWK_SKEY	Nework session key	Yes	Yes
MIB_APP_SKEY	Application session key	Yes	Yes
MIB_PUBLICE_NETWORK	The network type to public or not	Yes	Yes
MIB_REPEATER_SUPPORT	Support the operation with repeaters	Yes	Yes
MIB_CHANNELS	Communication channels	Yes	No
MIB_RX2_CHANNEL	Receive window 2 channel	Yes	Yes
MIB_RX2_DEFAULT_CHANNEL	Receive window 2 default channel	Yes	Yes
MIB_CHANNELS_MASK	LoRaWAN channels mask	Yes	Yes
MIB_CHANNELS_DEFAULT_MASK	LoRaWAN default channels mask	Yes	Yes
MIB_CHANNELS_NB_REP	Number of repetitions on a channel	Yes	Yes
MIB_MAX_RX_WINDOW_DURATION	Maximum receive window duration in ms	Yes	Yes
MIB_RECEIVE_DELAY_1	Receive delay 1 in ms	Yes	Yes
MIB_RECEIVE_DELAY_2	Receive delay 2 in ms	Yes	Yes
MIB_JOIN_ACCEPT_DELAY_1	Join accept delay 1 in ms	Yes	Yes
MIB_JOIN_ACCEPT_DELAY_2	Join accept delay 2 in ms	Yes	Yes
MIB_CHANNELS_DATARATE	Default data rate of a channel	Yes	Yes
MIB_CHANNELS_DEFAULT_DATARATE	Data rate of a channel	Yes	Yes
MIB_CHANNELS_TX_POWER	Transmission power of a channel	Yes	Yes
MIB_CHANNELS_DEFAULT_TX_POWER	Default transmission power of a channel	Yes	Yes
MIB_UPLINK_COUNTER	LoRaWAN Up-link counter	Yes	Yes
MIB_DOWNLINK_COUNTER	LoRaWAN Down-link counter	Yes	Yes
MIB_MULTICAST_CHANNEL	Multicast channels	Yes	No

Please reference LoRaMac.h of SDK to get more detail information.

3.LoRaWAN Example

3.1. Example Introduction

This example continuous sends sensor data to LoRaWAN Gateway. It could be use OTAA or ABP method connect to LoRa Network, then send received sensor data.

Node sensors
LoRaWAN Gateway

Figure 1

3.2. Prerequisites

Host machine:

PC running Microsoft Windows 7 or later with an available USB port.

Hardware :

(1). A LoRa Development Kit (LoRa DK).

(2). Some sensor devices. (optional)

(3). An LoRa Gateway. (We use Band868/915 Multitech conduit for testing)

Software :

(1). Intel IDE : Intel System Studio for Microcontrollers. (**Version :ISSM_2016.2.090**)

(2). LoRaWAN stack SDK. (include Radio Driver) } LoRaSDK.rar

(3). LoRaWAN example source code.

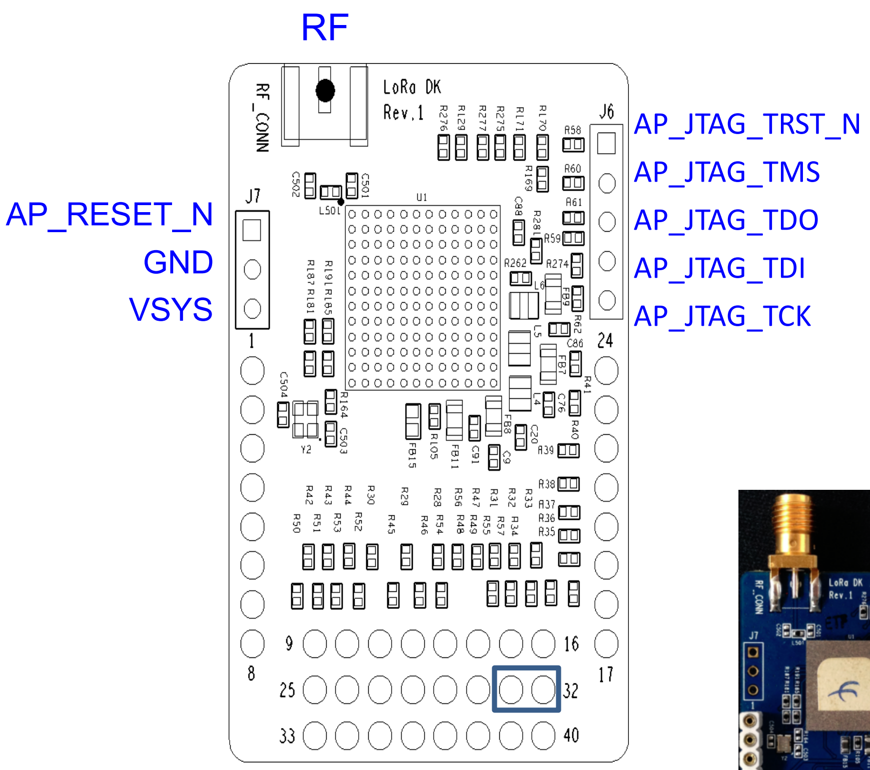
3.3. Setup Example Steps

3.3.1. Step 1: Connect the LoRa DK to host PC.

Intel Quark C1000 support JATG interface for programming and debugging. The JTAG interface of LoRa DK is on the top left and right (block pin headers), list as below figure. Connect JTAG interface to those pins.

Board No.		
Series Pin No.	Functions	Intel LoRa
1	AVDD	VCC_IO
2	DEC2	-
3	GND	GND
4	GND	GND
5	VDD	VSYS
6	SWDIO	-
7	SWDCLK	-
8	GND	GND
9	P0_27	-
10	P0_26	-
11	P0_16	AP_UART[0]_RXD_ADC[18]
12	P0_15	AP_UART[0]_TXD_GPIO[31]
13	P0_14	AP_I2C[0]_SS_SDA
14	P0_13	AP_I2C[0]_SS_SCL
15	P0_09	AP_GPIO_SS[2]_ADC[10]
16	P0_08	AP_GPIO_AON[0]
17	P0_07	AP_GPIO[0]_ADC[0]
18	P0_06	AP_GPIO[24]_SPI[0]_M_CS[0]
19	P0_05	AP_GPIO[23]_SPI[0]_M_MOSI
20	P0_04	AP_GPIO[22]_SPI[0]_M_MISO
21	P0_03	AP_I2C[1]_SS_SDA
22	P0_02	AP_I2C[1]_SS_SCL
23	P0_01	AP_GPIO[21]_SPI[0]_M_SCK
24	P0_00	AP_GPIO_SS[3]_ADC[11]
25	P0_22	AP_GPIO_SS[10]_PWM[0]
26	P0_23	AP_GPIO_SS[11]_PWM[1]
27	P0_24	AP_GPIO_SS[12]_PWM[2]
28	P0_25	AP_GPIO_SS[13]_PWM[3]
29	P0_28	AP_GPIO_SS[0]_ADC[8]_UART[1]_CTS
30	P0_29	AP_GPIO_SS[1]_ADC[9]_UART[1]_RTS
31	P0_30	AP_GPIO_SS[8]_ADC[16]_UART[1]_TXD
32	P0_31	AP_GPIO_SS[9]_ADC[17]_UART[1]_RXD
33	P0_21	AP_SPI[0]_SS_MISO
34	P0_20	AP_SPI[0]_SS_MOSI
35	P0_19	AP_SPI[0]_SS_SCK
36	P0_18	AP_SPI[0]_SS_CS[0]
37	P0_17	AP_GPIO[1]_ADC[1]
38	P0_12	AP_GPIO_SS[4]_ADC[12]
39	P0_11	AP_GPIO_AON[1]
40	P0_10	AP_GPIO_AON[2]

*Debug port



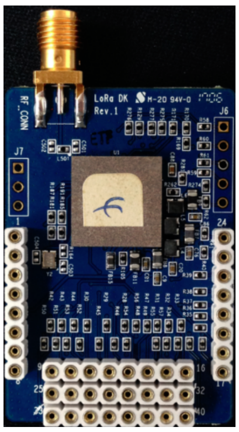


Figure 2: DK Pin Arrangement

*Inter Quark C1000 use UART(1) as debug port. When user use “QM_PRINT” function, the output will at UART(1).

3.3.2. Step 2: Check the JTAG interface setting correct.

Please reference : "LoRa DK JTAG setup guide".

3.3.3. Step 3: Run Intel System Studio for Microcontrollers.

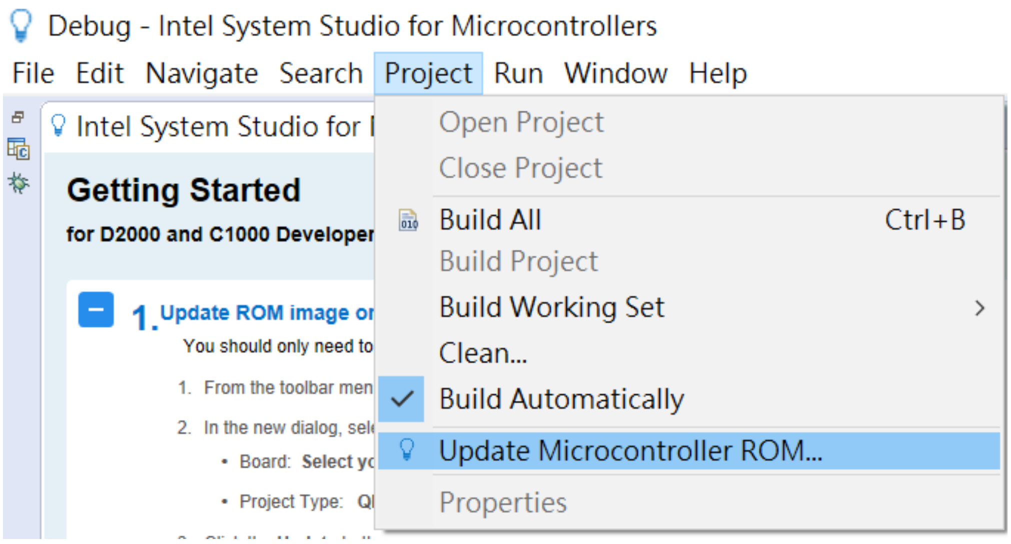
Using File Explorer, navigate to the C:\IntelSWTools\ISSM_2016.2.090 directory and doubleclick the file named iss_mcu_ide_eclipse-launcher.bat or doubleclick the shortcut at the top-disk..

3.3.4. Step 4: Update target ROM

> Note: Updating the boot ROM only needs to be performed once. Check with your lab instructor to determine if this step has already been done for your board

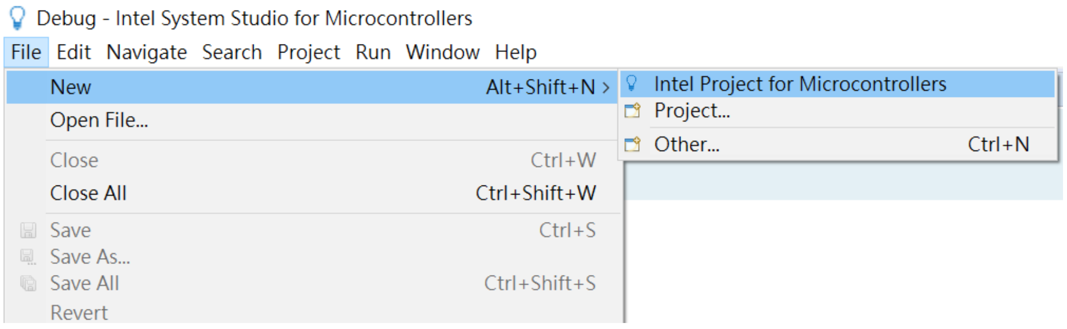
From the Project menu select the **[Update Microcontroller ROM...]** menu item.

The Update Microcontroller ROM dialog box will appear. Make sure the Board is the **Intel® Quark™ SE C1000 Developer Board**, and the Project type is **QMSI 1.4.0**. Click the Update button to re-flash the microcontroller's ROM.

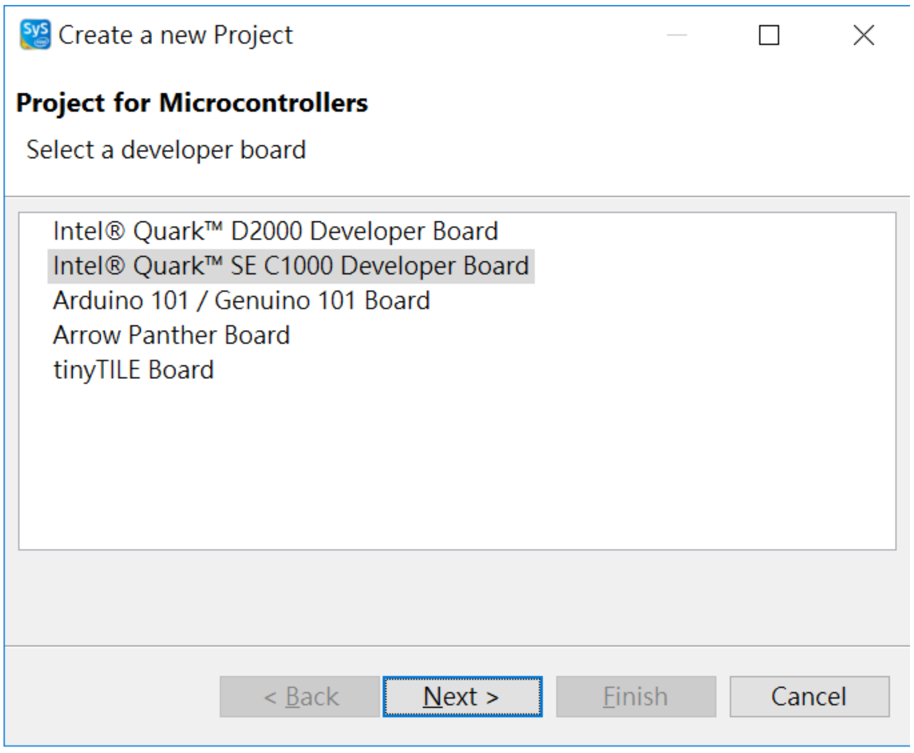


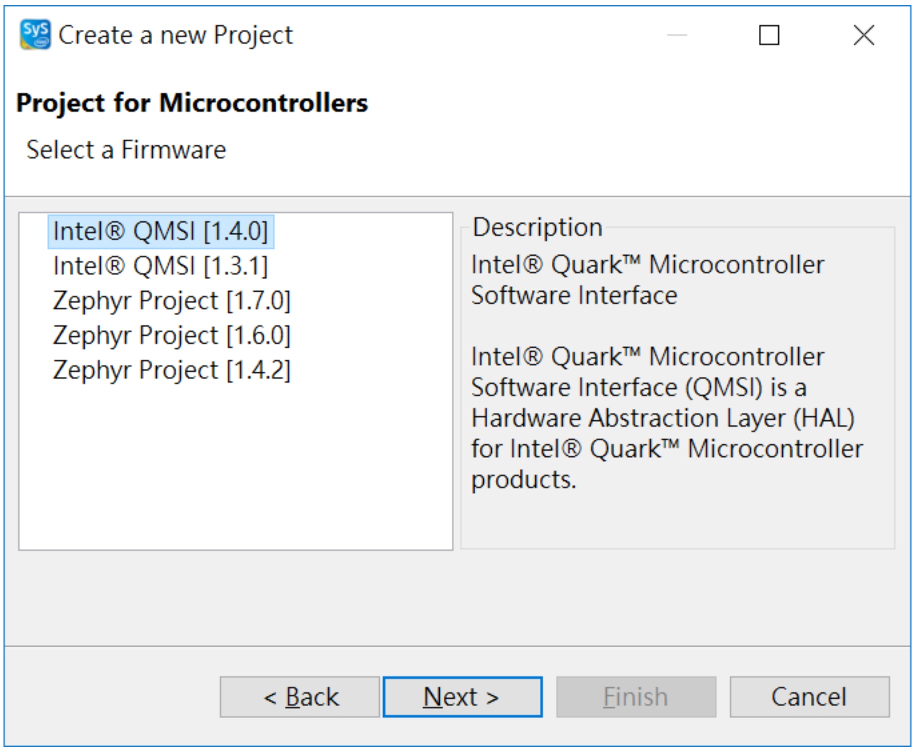
3.3.5. Step 5: Create a new project

From the toolbar select **File > New > Intel Project** for Microcontrollers:



Select the **[Intel® Quark™ SE C1000 Developer Board]** in the Select a developer board dialog.
Then, select the **[Intel® QMSI (1.4.0)]** project type.
Select **[Intel® Quark™]** as your platform configuration:
In the Setup project dialog, name the project **[project name you want]** and make sure the **\Driver\Serial Peripheral Interface example** is selected (this sample project will be the basis for your project).


Create a new Project
Project for Microcontrollers
Select a developer board
Intel® Quark™ D2000 Developer Board
Intel® Quark™ SE C1000 Developer Board
Arduino 101 / Genuino 101 Board
Arrow Panther Board
tinyTILE Board
< Back
Next >
Finish
Cancel


Create a new Project

Project for Microcontrollers

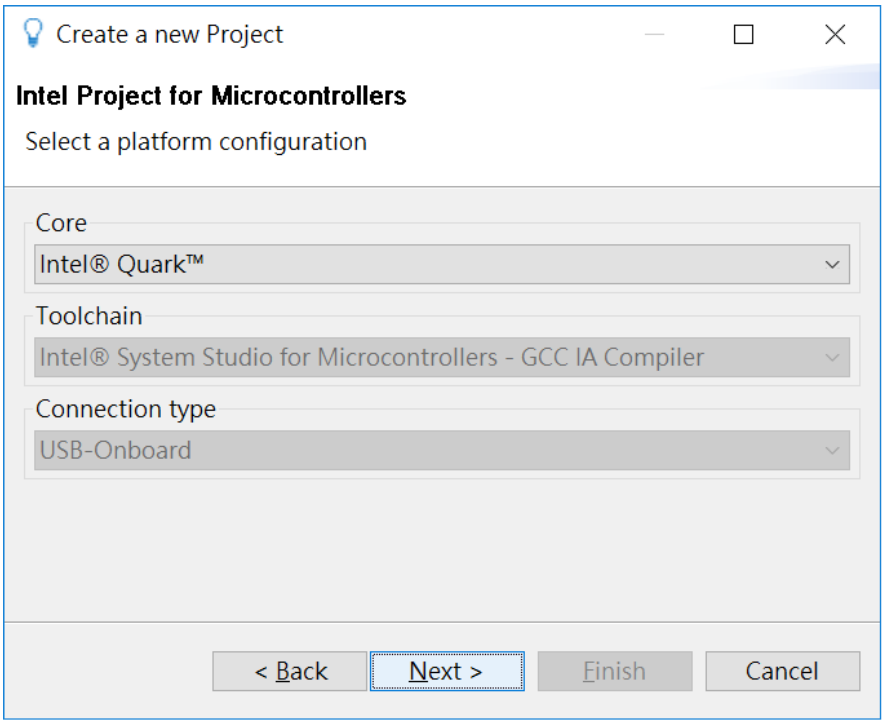
Select a Firmware

- Intel® QMSI [1.4.0]
- Intel® QMSI [1.3.1]
- Zephyr Project [1.7.0]
- Zephyr Project [1.6.0]
- Zephyr Project [1.4.2]

Description

Intel® Quark™ Microcontroller Software Interface

Intel® Quark™ Microcontroller Software Interface (QMSI) is a Hardware Abstraction Layer (HAL) for Intel® Quark™ Microcontroller products.


Create a new Project

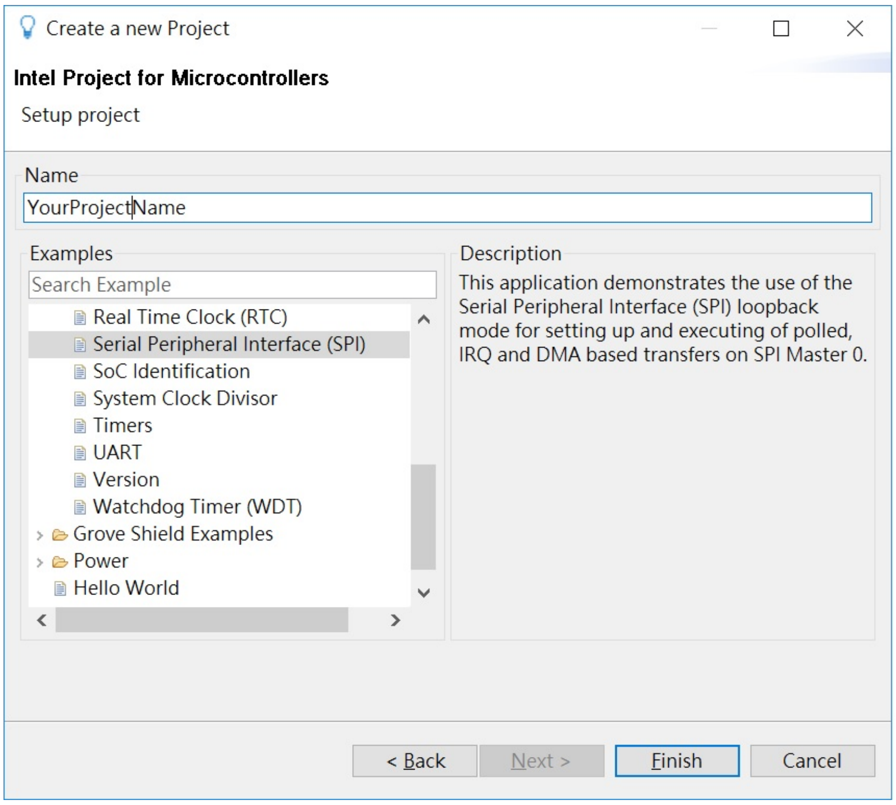
Intel Project for Microcontrollers

Select a platform configuration

Core: Intel® Quark™

Toolchain: Intel® System Studio for Microcontrollers - GCC IA Compiler

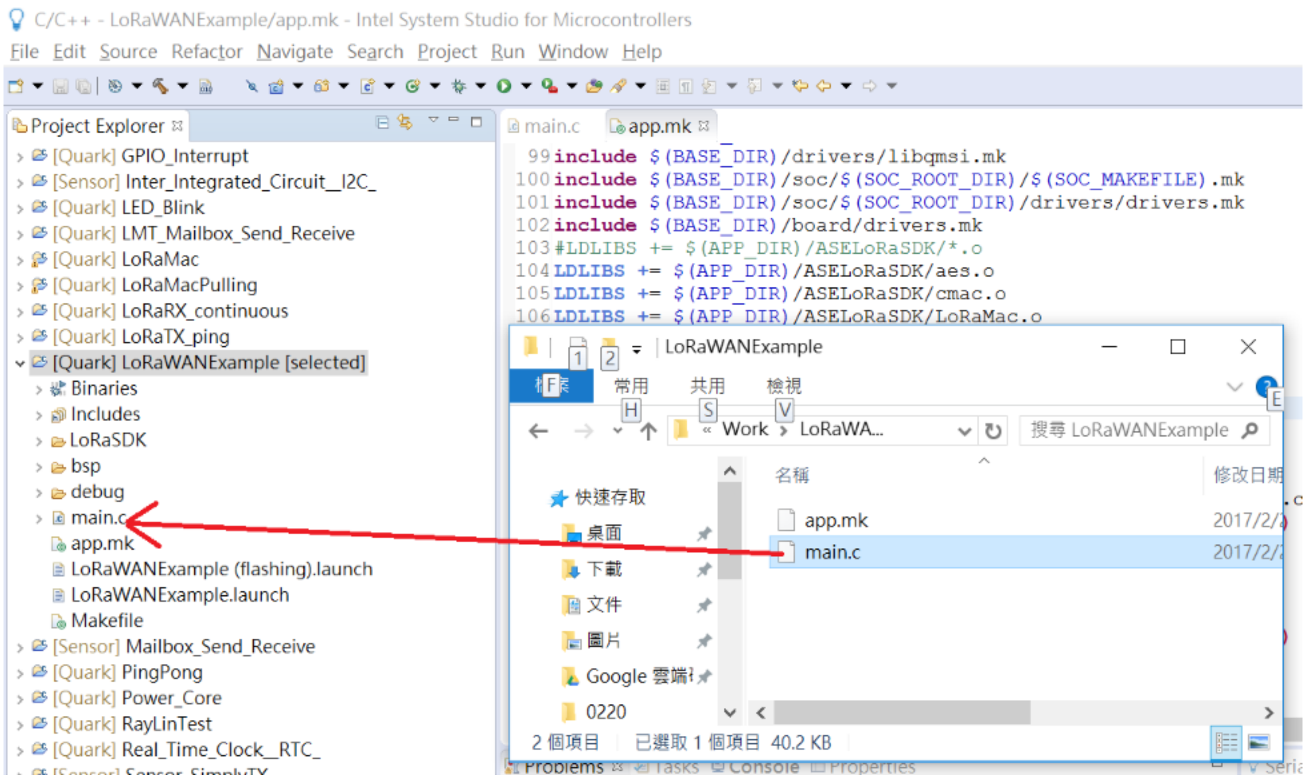
Connection type: USB-Onboard



This application demonstrates the use of the Serial Peripheral Interface (SPI) loopback mode for setting up and executing of polled, IRQ and DMA based transfers on SPI Master 0.

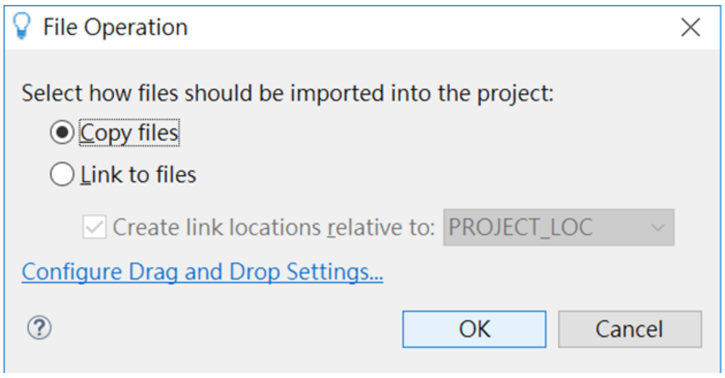
Click the **Finish** button to create the new project (in your workspace).

After unzip LoRaSDK file, it include two folders first one is SDK file, another one is example file (main.c & app.mk). Use Windows File Explorer to locate the example main.c file on your development system and replace the contents of the main.c file in your project with the example main.c file.
You can “drag and drop” the example main.c file onto the old main.c file, from your Windows File Explorer window to the Project Explorer window (in the Intel System Studio for Microcontrollers window):



```
include $(BASE_DIR)/drivers/libqmsi.mk
include $(BASE_DIR)/soc/$(SOC_ROOT_DIR)/$(SOC_MAKEFILE).mk
include $(BASE_DIR)/soc/$(SOC_ROOT_DIR)/drivers/drivers.mk
include $(BASE_DIR)/board/drivers.mk
#LDLIBS += $(APP_DIR)/ASELoRaSDK/*.o
LDLIBS += $(APP_DIR)/ASELoRaSDK/aes.o
LDLIBS += $(APP_DIR)/ASELoRaSDK/cmac.o
LDLIBS += $(APP_DIR)/ASELoRaSDK/LoRaMac.o
```

Make sure you select Copy files when you perform the "drag and drop" operation between windows.


File Operation

Select how files should be imported into the project:

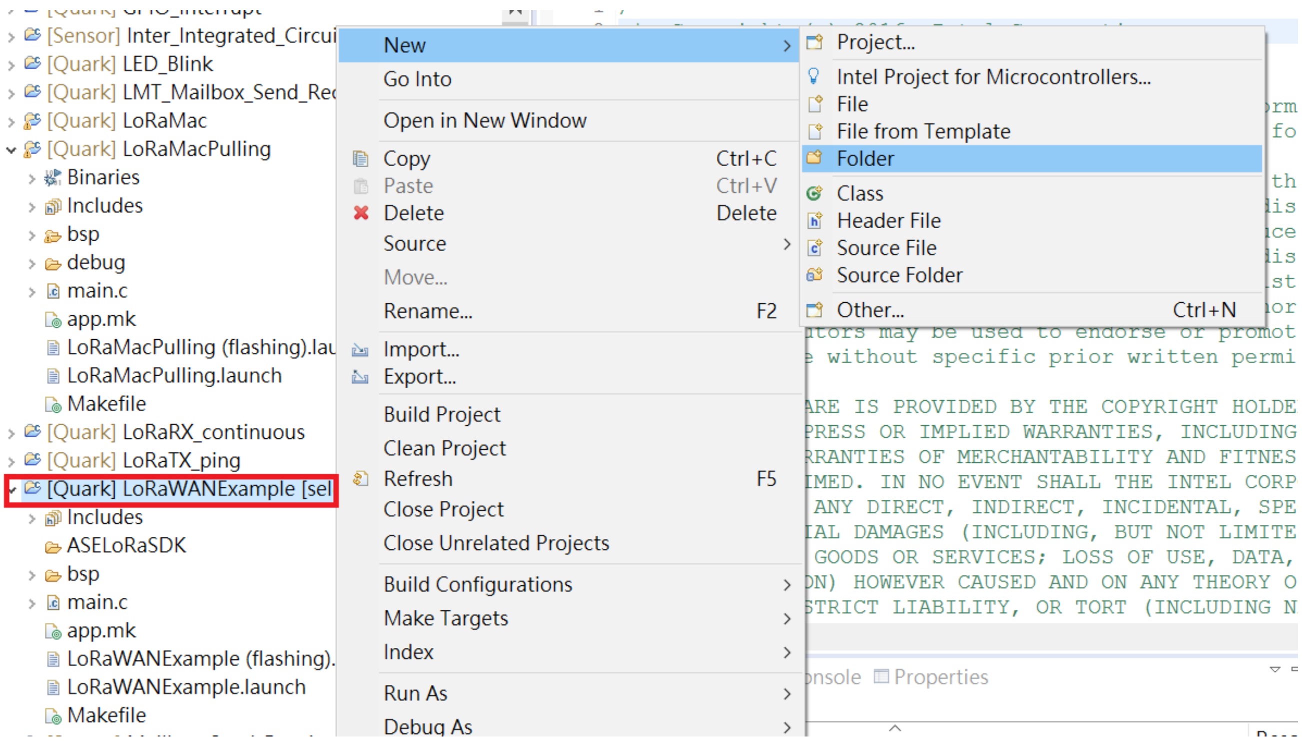
[✓ Copy files]
Link to files

Create link locations relative to: PROJECT_LOC

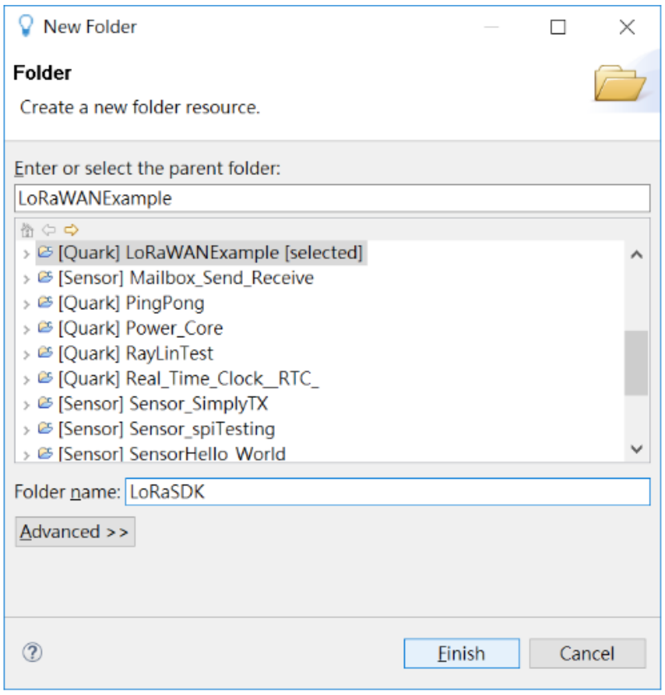
Save the file by pressing Ctrl-S key combination.

3.3.6. Step 6: Setup LoRaWAN SDK

New a [LoRaSDK] folder in your project. Tap the right click in your project, select [New], select [Folder].



Input new folder name: LoRaSDK.

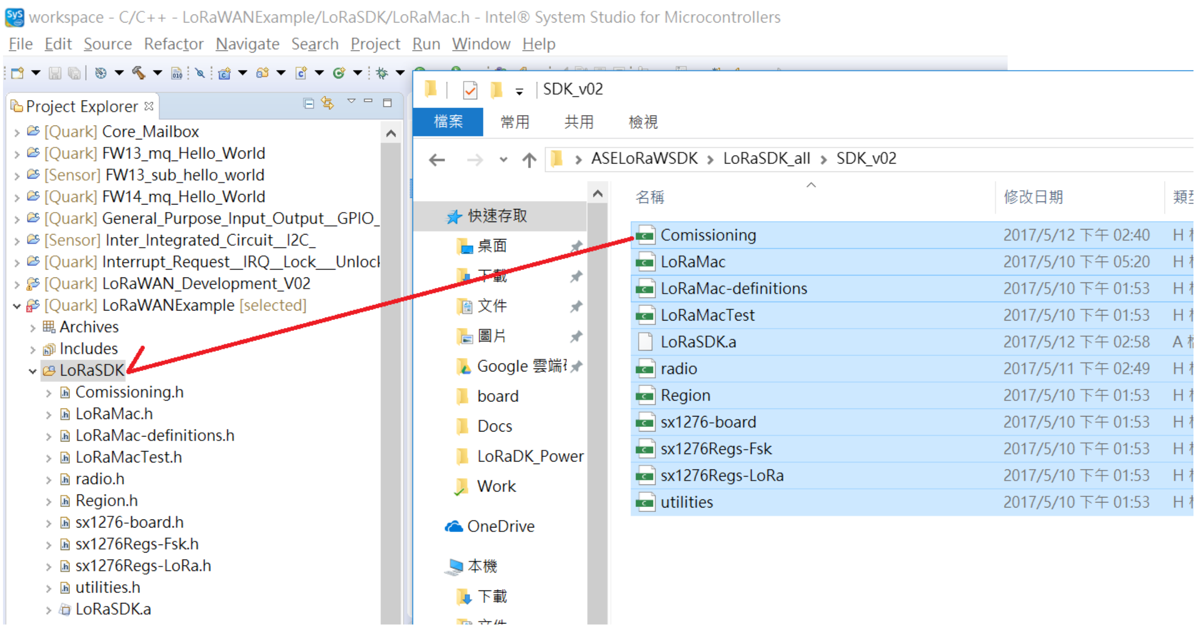

New Folder
Folder
Create a new folder resource.

Enter or select the parent folder: LoRaWANExample

Folder name: LoRaSDK

[Quark] LoRaWANExample [selected]
[Sensor] Mailbox_Send_Receive
[Quark] PingPong
[Quark] Power_Core
[Quark] RayLinTest
[Quark] Real_Time_Clock__RTC_
[Sensor] Sensor_SimplyTX
[Sensor] Sensor_spiTesting
[Sensor] SensorHello_World
Advanced >>
Finish
Cancel

Use Windows File Explorer to locate the LoRaSDK files on your development system and copy LoRaSDK files into LoRaSDK folder.


workspace - C/C++ - LoRaWANExample/LoRaSDK/LoRaMac.h - Intel® System Studio for Microcontrollers

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- [Quark] Core_Mailbox
- [Quark] FW13_mq_Hello_World
- [Sensor] FW13_sub_hello_world
- [Quark] FW14_mq_Hello_World
- [Quark] General_Purpose_Input_Output__GPIO_
- [Sensor] Inter_Integrated_Circuit__I2C_
- [Quark] Interrupt_Request__IRQ__Lock__Unlocl
- [Quark] LoRaWAN_Development_V02
- [Quark] LoRaWANExample [selected]
  - Archives
  - Includes
  - LoRaSDK
    - Comissioning.h
    - LoRaMac.h
    - LoRaMac-definitions.h
    - LoRaMacTest.h
    - radio.h
    - Region.h
    - sx1276-board.h
    - sx1276Regs-Fsk.h
    - sx1276Regs-LoRa.h
    - utilities.h
    - LoRaSDK.a

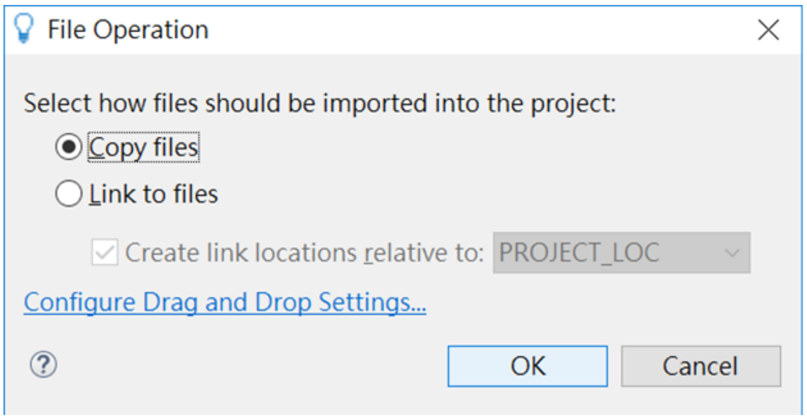
SDK_v02

檔案 常用 共用 檢視

ASELoRaWSDK › LoRaSDK_all › SDK_v02

快速存取: 桌面, 下載, 文件, 圖片, Google 雲端, board, Docs, LoRaDK_Power, Work; OneDrive; 本機: 下載

名稱	修改日期	類
Comissioning	2017/5/12 下午 02:40	H
LoRaMac	2017/5/10 下午 05:20	H
LoRaMac-definitions	2017/5/10 下午 01:53	H
LoRaMacTest	2017/5/10 下午 01:53	H
LoRaSDK.a	2017/5/12 下午 02:58	A
radio	2017/5/11 下午 02:49	H
Region	2017/5/10 下午 01:53	H
sx1276-board	2017/5/10 下午 01:53	H
sx1276Regs-Fsk	2017/5/10 下午 01:53	H
sx1276Regs-LoRa	2017/5/10 下午 01:53	H
utilities	2017/5/10 下午 01:53	H


File Operation

Select how files should be imported into the project:

[✓ Copy files]

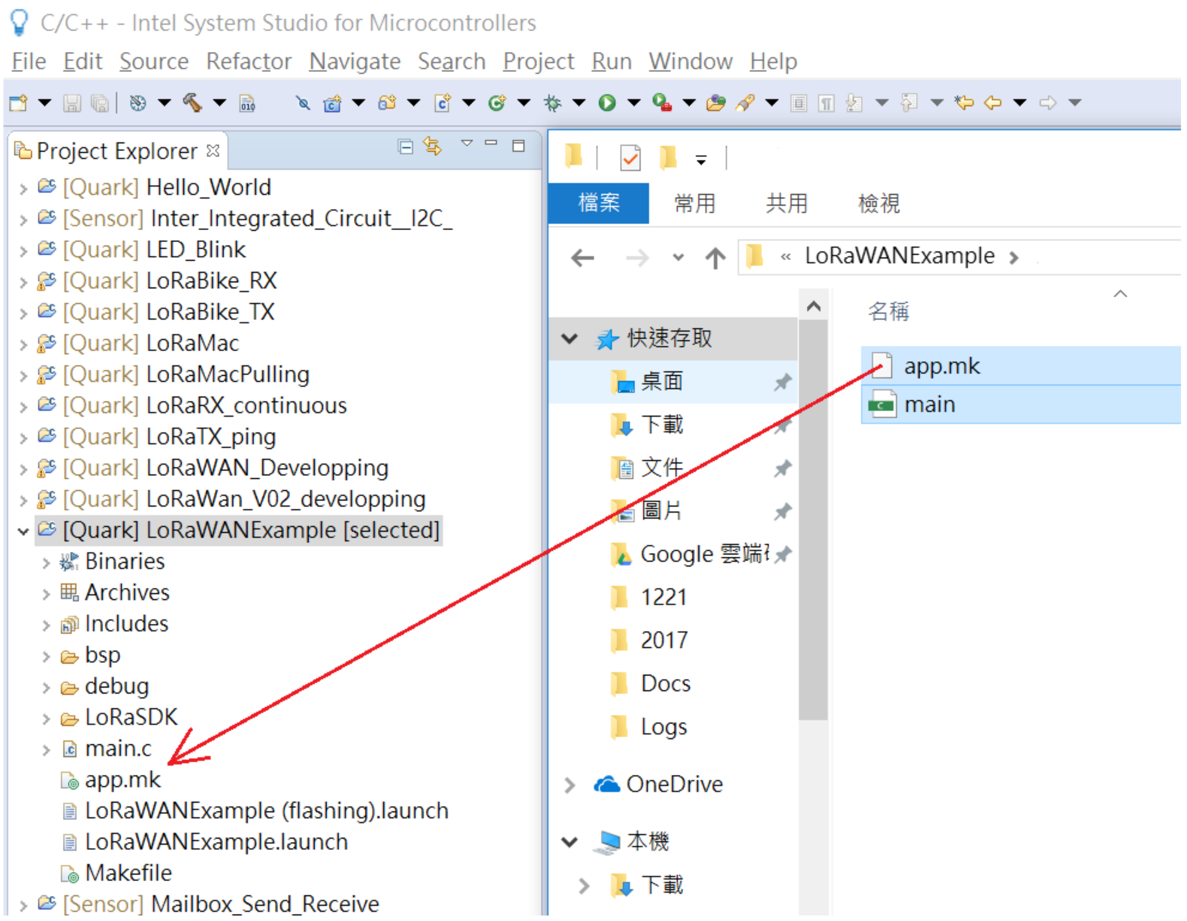
Link to files

Create link locations relative to: PROJECT_LOC

Configure Drag and Drop Settings...

OK Cancel

Replace the **app.mk** file with LoRaSDK provided **app.mk** file.



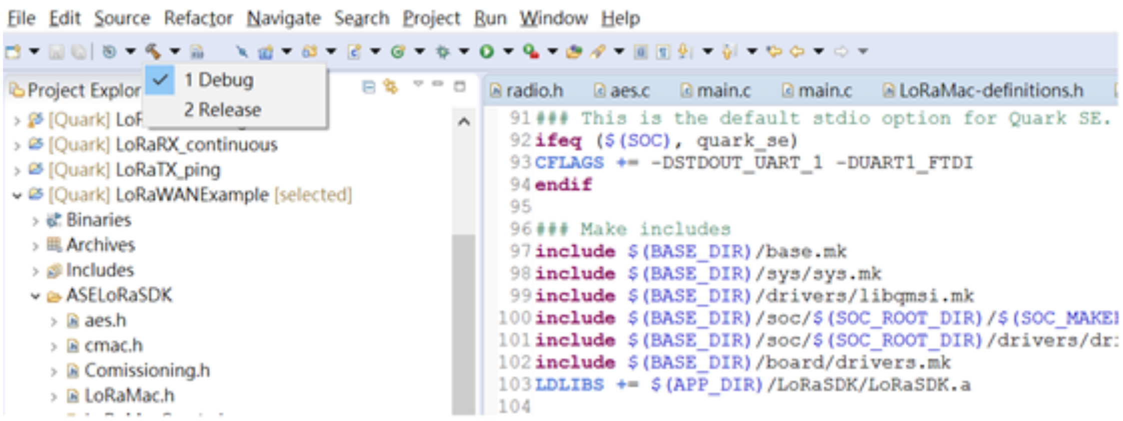
app.mk of LoRaSDK file add below settings.

LoRaSDK Setting :

"LDLIBS += $(APP_DIR)/LoRaSDK/LoRaSDKa"

3.3.7. Step 7: Build the project

Click on the Build button ("hammer" icon) on the toolbar to build the project. Alternatively, you can select the build configuration using the drop down menu next to the Build button. Intel® Systems Studio for Microcontrollers provides two build configurations: debug and release. The debug configuration includes debug symbols in the compiled files to facilitate source-code debugging. Debug is the default build configuration. The release configuration optimizes the compiled code for deployment.

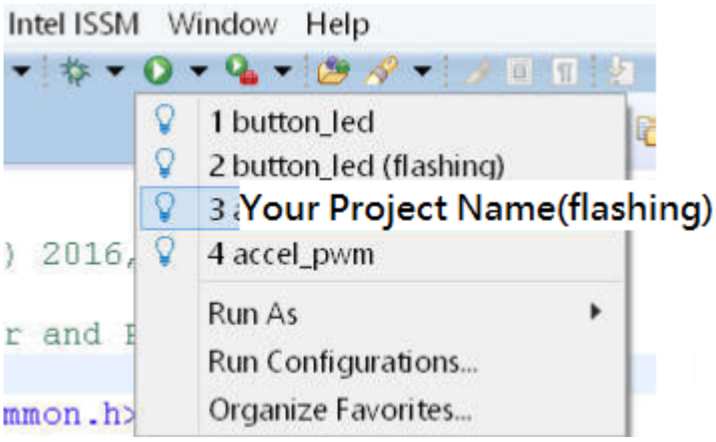


```
### This is the default stdio option for Quark SE.
ifeq ($(SOC), quark_se)
CFLAGS += -DSTDOUT_UART_1 -DUART1_FTDI
endif

### Make includes
include $(BASE_DIR)/base.mk
include $(BASE_DIR)/sys/sys.mk
include $(BASE_DIR)/drivers/libqmsi.mk
include $(BASE_DIR)/soc/$(SOC_ROOT_DIR)/$(SOC_MAKEI
include $(BASE_DIR)/soc/$(SOC_ROOT_DIR)/drivers/dr:
include $(BASE_DIR)/board/drivers.mk
LDLIBS += $(APP_DIR)/LoRaSDK/LoRaSDK.a

```

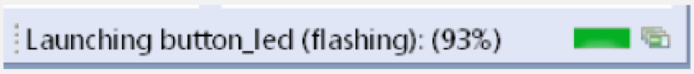
3.3.8. Step 8: Flash and run the project

Click on the drop down menu next to the Run button ("play" icon). Select [your project name] (flashing) from the menu ("flashing" means you are going to program your built application onto the target board's Flash ROM). Intel® Systems Studio for Microcontrollers will recompile the code and "flash" it to the microcontroller.

Intel® System Studio for Microcontrollers provides two options for running your app:



NOTE: The microcontroller "flashing" process can take time. You'll find a progress bar at the bottom right corner of Intel® Systems Studio for Microcontrollers:



3.4. LoRaWAN Example Settings

There are several settings about how to use this example:

(1) Include SDK files.
(2) Timer settings.
(3) Define Radio ➔ Radio Driver : Radio.h
(4) Region and custom channel setting
(5) Define LoRaWAN Network parameters. ➔ Comissioning.h
(6) Define LoRaMac connection parameters. ➔ LoRaMac.h
(7) Used GPIO pins setting.

3.4.1. Include SDK files and define parameters

- After copy example main.c file, please make sure that include file path are correct.

The default path is "LoRaSDK/xxs.h"

- If used our "Sensor Data Example" on sub-system, we set the MailBox channel to Channel 0. (QM_MBOX_CH_0)

Define the MailBox call back function in main.c named "mbox_cb".

- In this example, the LoRa DataRate : **DR_0**, disable confirmed message(**unconfirmed**), and LoRaWan application port is "**1**".

3.4.2. Timer settings

Intel Quark C1000 support 4 HW timers. LoRaSDK used 1 timer(timer0). In this example, it used another 1 timer(timer1) for send Tx data event. So, there are still have 2 HW timers could be use (timer 2 and timer 3).
We are setting the timer functions in our example "main.c" file. Those are (1) TimerInit(), (2)Time0Stop(), (3) Time1Stop(), and (4)timer_callback(). The Timerinit() function initialize the timer0 clock which SDK used, and define the callback function to timer_callback(). The timer1 used in main.c is a Tx duty cycle timer. It could be change duty cycle time by setting TxDutyCycleTime parameter.
If want stop Timer0, please reference Timer0Stop() function in "main.c". Set the sw_timer_running to false, then stop Timer0.

3.4.3. Define Radio Ⓟ Radio Driver : Radio.h

After include Radio.h file, user could use Radio functions to control LoRa transceiver directly.
Ex :

- Set Radio to sleep/standby mode : Radio.Sleep();/Radio.Standby();
- Read Radio Register (address: 0x42) : uint8_t value = Radio.Read(0x42);
- Write Radio Register(address:0x12 , value:0xff) : Radio.Write(0x12,0xff);

3.4.4. Region and Custom channel Setting

LoRaSDK V02 supported LoRa SPEC V1.0.2. This version supported many Regions those are:

(1) AS923
(2) AU915
(3) CN470
(4) CN779
(5) EU433
(6) EU868
(7) KR920
(8) IN865
(9) US915
(10) US915-Hybrid

And two custom regions for customize channels. The custom regions are CustomAType and CustomBType. A type is based on EU868 band parameter. B type is based on US915 band parameter.

When user want to setting custom channels, (1)select a custom type region, (2)setting your channels, and (3) Modify band parameters in RegionCustomA(B)Type.h.

(1) Select one Region

Main.c :

```
#define REGION_CustomBType
```

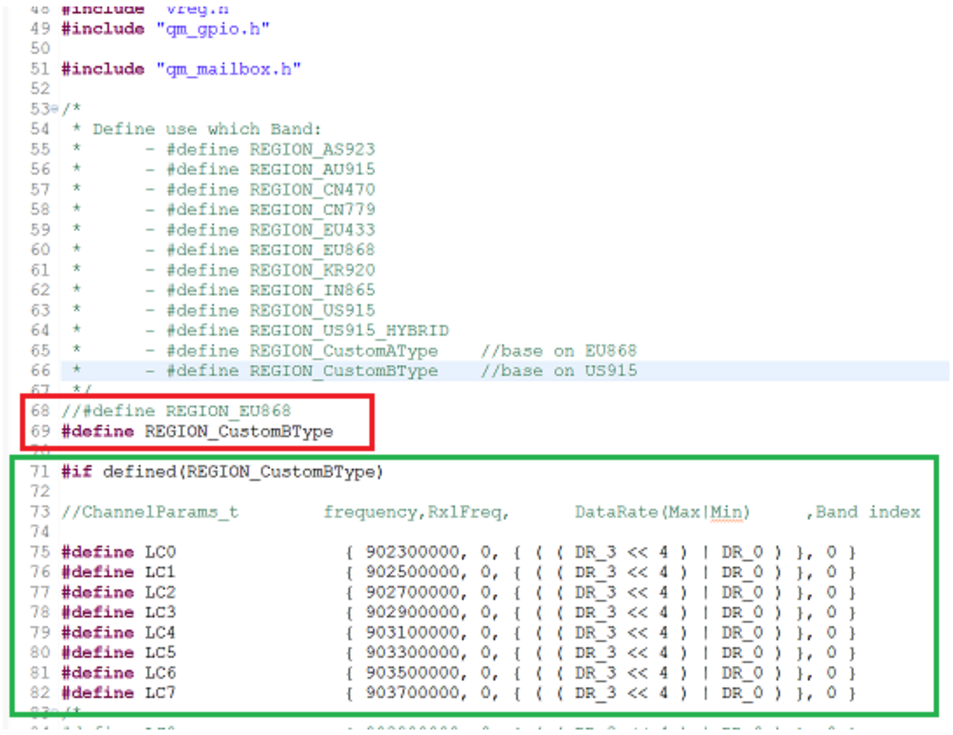
(2) Setting channels:

```
#define LC0 {902300000,0, {    (   (    DR_3 << 4 ) | DR_0) },    0 }
                    Frequency,Rx1Freq, DataRate(Max | Min)       , Band
```

The channels stricture is use ChannelParams_t which defined in LoRaMac.h. The first parameter is channel's frequency(Hz). The second is alternative frequency for RX window 1.

DataRate is setting the range. The last is band index.

User usually just need to modify first parameter and DataRate range(Green block). Other parameters are copy the example.



```
#include "vreg.h"
#include "qm_gpio.h"

#include "qm_mailbox.h"

/*
 * Define use which Band:
 *        - #define REGION_AS923
 *        - #define REGION_AU915
 *        - #define REGION_CN470
 *        - #define REGION_CN779
 *        - #define REGION_EU433
 *        - #define REGION_EU868
 *        - #define REGION_KR920
 *        - #define REGION_IN865
 *        - #define REGION_US915
 *        - #define REGION_US915_HYBRID
 *        - #define REGION_CustomAType    //base on EU868
 *        - #define REGION_CustomBType    //base on US915
 */
//#define REGION_EU868
#define REGION_CustomBType

#if defined(REGION_CustomBType)

//ChannelParams_t         frequency,RxlFreq,      DataRate(Max|Min)     ,Band index

#define LC0                 { 902300000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
#define LC1                 { 902500000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
#define LC2                 { 902700000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
#define LC3                 { 902900000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
#define LC4                 { 903100000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
#define LC5                 { 903300000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
#define LC6                 { 903500000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
#define LC7                 { 903700000, 0, { ( ( DR_3 << 4 ) | DR_0 ) }, 0 }
/*
```

(3) Modify band parameters in RegionCustomA(B)Type.h.

EX :

#define CustomBType_RX_WND_2_FREQ 869525000

For an Example : If want set 8 channels & Rx2 window like below.

Channel	Frequency	Modulation / BW
0	865.400 MHz	MultiSF 125 kHz
1	865.600 MHz	MultiSF 125 kHz
2	865.800 MHz	MultiSF 125 kHz
3	866.000 MHz	MultiSF 125 kHz
4	866.200 MHz	MultiSF 125 kHz
5	866.400 MHz	MultiSF 125 kHz
6	866.600 MHz	MultiSF 125 kHz
7	866.800 MHz	MultiSF 125 kHz

RX2 channel (downlink)		
RX2	865.200 MHz	SF12 125 kHz

(1) Chose Region_CustomAType (Because, that seems base on 868band)
In Main.c add the line "#define REGION_CustomAType"

(2) Setting channels:

```
#define LC0 {865040000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
#define LC1 {865060000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
#define LC2 {865080000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
#define LC3 {866000000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
#define LC4 {866020000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
#define LC5 {866040000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
#define LC6 {866060000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
#define LC7 {866080000,0, {     (    (    DR_5 << 4 ) | DR_0) },    0}
```

(3) Modify band parameter

(3-1) Remove initilized channels:
Use LoRaMacChannelRemove() function to remove channel.

(3-2) Add Custom Channels:
Use LoRaMacChannelAdd() function to Add channel.

(3-3) Setting Rx2 Window Freq like below:

```
RegionSetCustomRX2WindowFreq(LORAMAC_REGION_CustomAType,865200000);
mibReq.Type = MIB_RX2_CHANNEL;
mibReq.Param.Rx2Channel = (Rx2ChannelParams_t){865200000,DR_0};
LoRaMacMibSetRequestConfirm(&mibReq);
```

3.4.5. Define LoRaWAN Network parameters. ➔Comissioning.h

LoRaWan has two method to join to LoRa Network. The first is “Over the Air Activation” (OTAA). When use this method, user need fill out below parameters into “commissioning.h” file:

(1) OVER_THE_AIR_ACTIVATION set to “**1**”.
(2) DevEUI : A globally unique end-device identifier.
(3) AppEui : The application identifier.
(4) AppKey : An AES-128 key.

Each time use OTAA success join to Lora network, LoRa Gateway will response some information to end-device. Those include (1) NetID, (2)DevAddr, (3)NwKSKey, and (4) AppSKey.

The second join network method is “Activation by Personalization” (ABP).
User fill out known parameters (after OTAA) into “commissioning.h” file, those parameters are:

(1) OVER_THE_AIR_ACTIVATION set to “**0**”
(2) NetID : A network identifier.
(3) DevAddr: An end-device address.
(4) NwkSKey : A network session key.
(5) AppSKey : An application session key.

Above network parameters could be find at the “**case DEVICE_STATE_JOIN**” of “switch(DeviceState)” of “while loop” of “main.c” of “LoRaWAN Example”.

3.4.6. Define LoRaWAN parameters ➔Main.c

There are two parameters often to set when use this example. That are “Datarate” and “Application port”, both parameters could be set by follow two methods:

(a) Define the value:

```
#define LORAWAN_DEFAULT_DATARATE        DR_0
#define LORAWAN_APP_PORT                1
```

(b) Set the vale when call MCPS (MAC Common Part Sublayer) services.

Ex:

```
mcpsReq.Type=MCP_UNCONFIRMED;
mcpsReq.Req.Unconfirmed.fBuffer = AppPort;
mcpsReq.Req.Unconfirmed.fBuffer = AppData;
mcpsReq.Req.Unconfirmed.fBufferSize = AppDataSize;
mcpsReq.Req.Unconfirmed.Datarate = DR_0;
LoRaMacMcpsRequest(&mcpsReq);
```

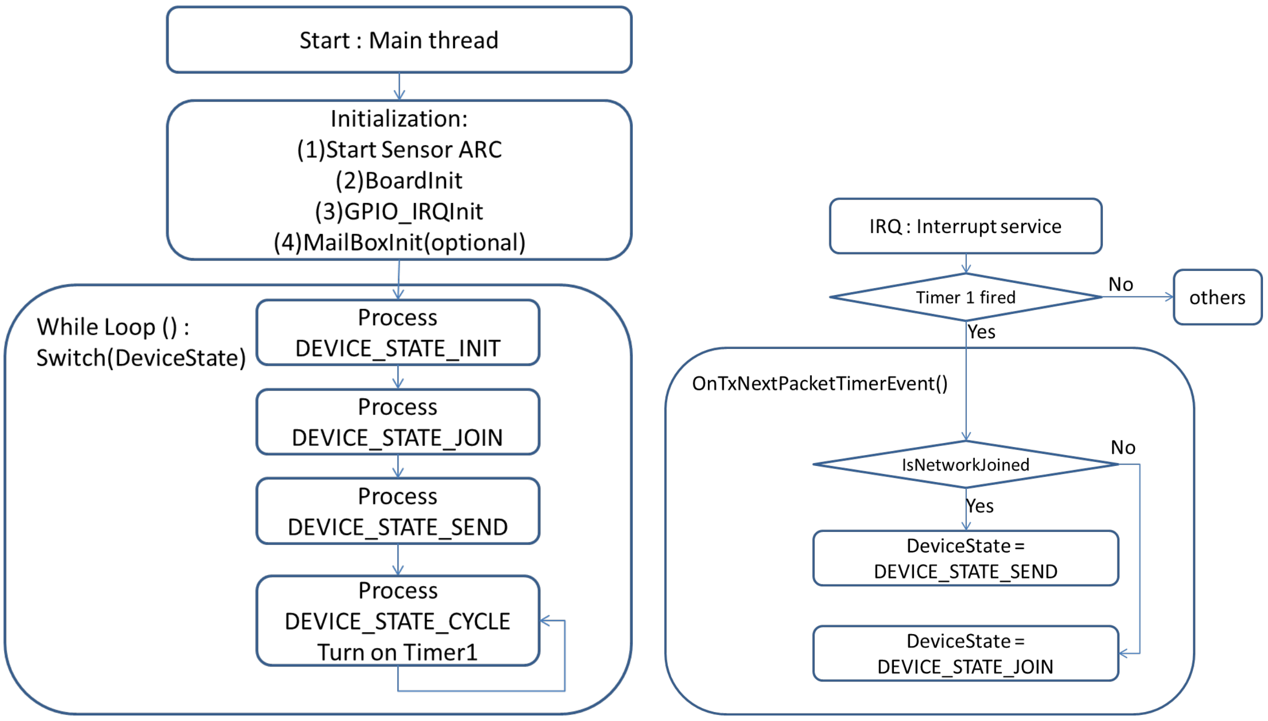
3.4.7. Define used GPIO configuration.

LoRa DK used some GPIO pins to control RF antenna and interrupt from LoRa chip. Please note don't re-setting those pins and keep them workable. Those pins are GPIO Pin **4 ~ 7 ,20 and 28**. The configuration is defined in " Main_IrqInit()" function of Example. And like below in main.c :

```
/* Request IRQ and write GPIO port config */
cfg.direction = |BIT(4)|BIT(5)|BIT(6)|BIT(7);
cfg.int_en = BIT(SX1276.DIO0);                /* Interrupt enabled */
cfg.int_type = BIT(SX1276.DIO0);              /* Edge sensitive interrupt */
cfg.int_polarity = BIT(SX1276.DIO0);      /* Rising edge */
cfg.int_debounce = BIT(SX1276.DIO0);          /* Debounce enabled */
cfg.int_bothedge = 0x0;                   /* Both edge disabled */
cfg.callback = gpio_example_callback;
```

3.5. LoRaWAN Example flow chart

The example use “DeviceState” parameter to switch process flow. After the flow into Case: “DEVICE_STATE_CYCLE”, main thread will turn on Timer1. Until interrupt occur, interrupt service will check the “Network” state and set the “DeviceState” to the right state.



Process description:

(1) DEVICE_STATE_INIT:

Initialize LoRaMacPrimitives, LoRaMacInitialization(Region) , special settings.

(2) DEVICE_STATE_JOIN:

if use OTAA, set necessary parameters: DevEui, AppEui, and AppKey.

If use ABP, set networkID, DevAddr, NWK_SKEY, and APP_SKEY.

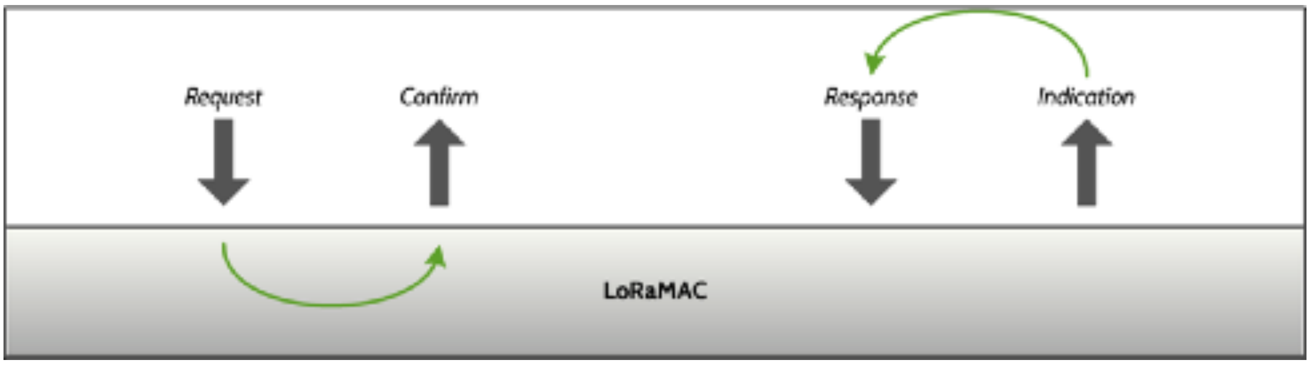
(3) DEVICE_STATE_SEND:

If ready for send, prepare the TxFrame, then send the Frame.

(4) DEVICE_STATE_CYCLE:

Set the cycle time, and then turn on time1. When timer fired, check the state. If joined network send data, else request join network.

3.6. LoRaWAN Example Receive data



When LoRa DK receive data from gateway, the function “McpsIndication( )” will called. Reference McpsIndication_t (LoRaMac.h) to know the data structure. Check receive data from mcpsIndication->[parameter].

Ex : Check Application Port , RSSI , Snr.

In McpsIndication( ) :

```
QM_PRINTF(“ Port :%d”, mcpsIndication->Port);
QM_PRINTF(“ RSSI :%d”, mcpsIndication->RSSI);
QM_PRINTF(“ Snr :%d”, mcpsIndication->Snr);
```

Note : McpsIndication_t Data structure :

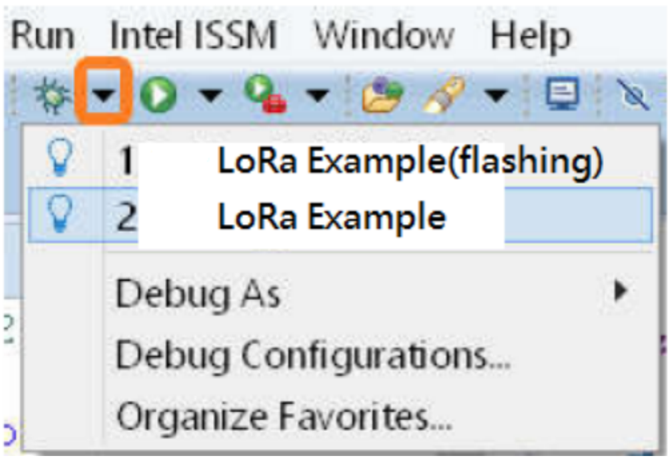
t

```
typedef struct sMcpsIndication
{
  .. .
  uint8_t Multicast;
  uint8_t Port;
  uint8_t RxDatarate;
  uint8_t *Buffer;
  uint8_t BufferSize;
  int16_t Rssi;
  uint8_t Snr;
  bool AckReceived;
  uint32_t DownLinkCounter; //The downlink counter value for the received frame
}McpsIndication_t;
```

4. Debugging with LoRaWAN Example

After your app “flashed” and is running on the board you can debug the application without flashing it (again) to the board. Click the drop down menu next to the Debug button (“bug” icon) and select **LoRa Example** (second option) from the drop-down menu:

Establish a serial connection to the board



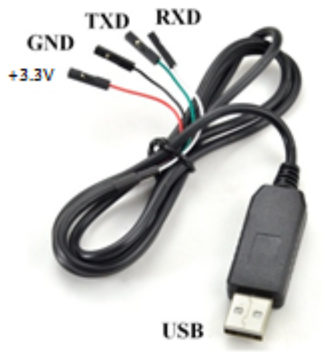
Intel® System Studio for Microcontrollers provides two options for debugging your app:

1. The LoRa Example (flashing) option compiles and programs the application onto your target system Flash ROM before the application is run. Use this option if your application’s source code has changed since the last time the board was “flashed.”

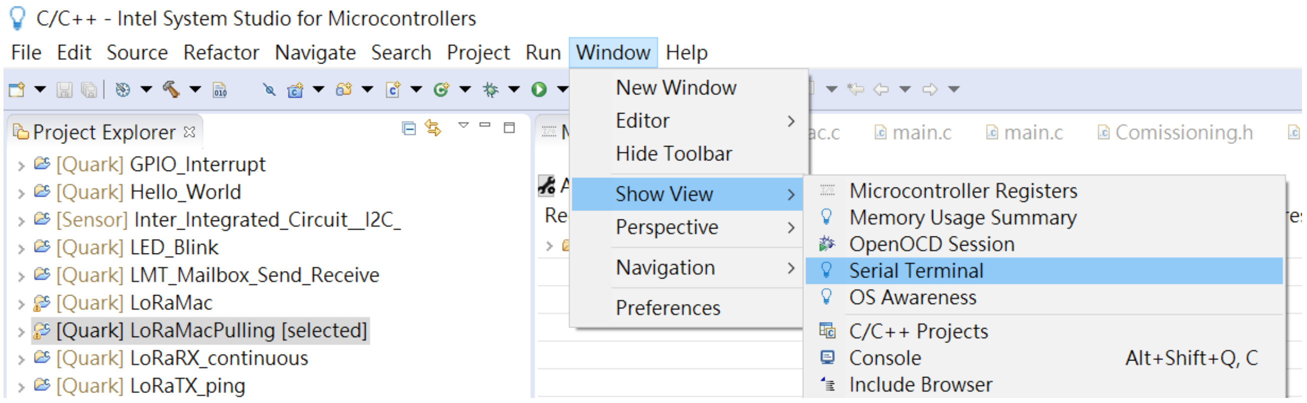
2. The LoRa Example option assumes that your board has already been programed with the application. Use this option if you do not need to update the application on the board since the last time it was “flashed” and run.

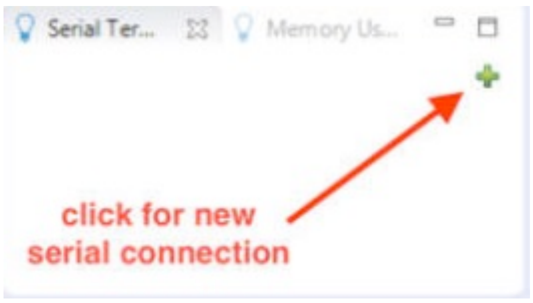
4.1. Establish a serial connection to the board

Please take an USB to RS232 cable, and connect TX, RX and GND pins to DK debug port and GND (Ref. Figure 2). USB-RS232-TX connect DK pin 32(RXD); USB-RS232-RX connect DK pin 31(TXD)

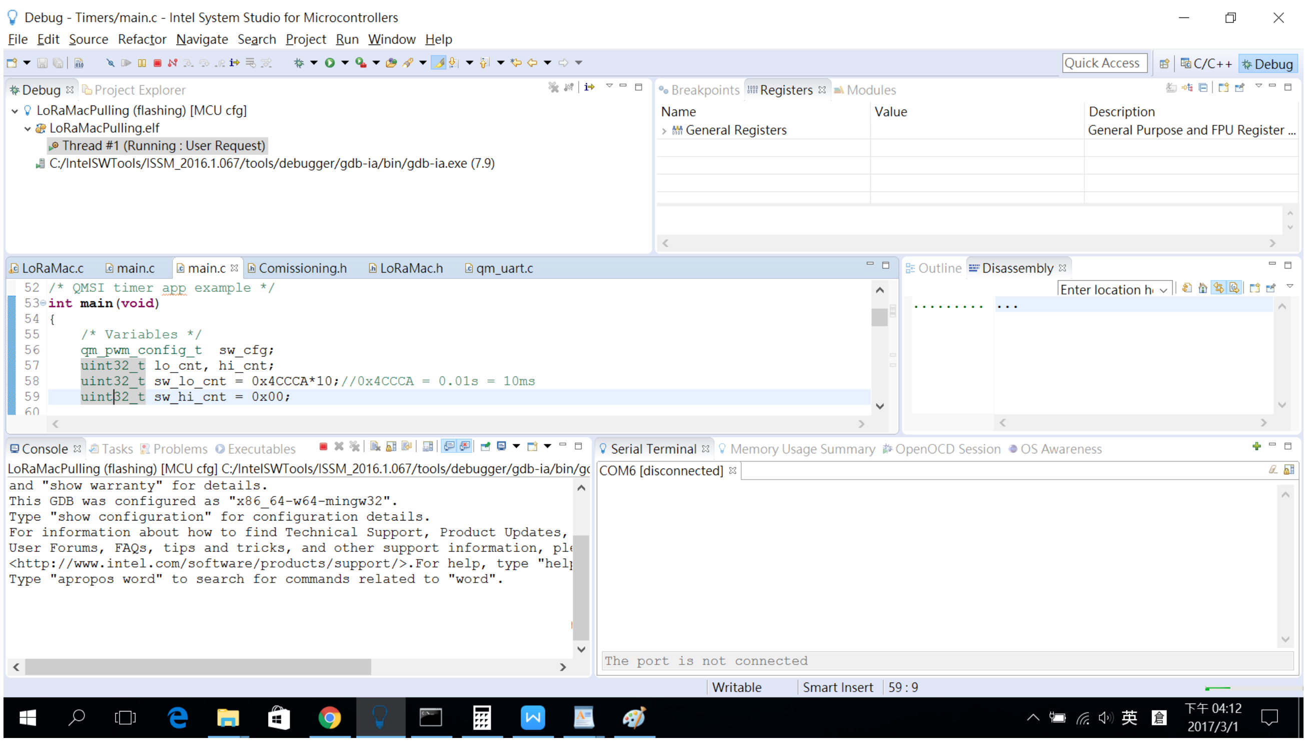


From the toolbar in Intel® System Studio for Microcontrollers select Window > Show View > Serial Terminal:





In this new window click on the Green plus sign to start a serial connection. Then set the COM port to the one you observed from your host PC and leave the other fields in their default settings.


Debug - Timers/main.c - Intel System Studio for Microcontrollers

Debug: LoRaMacPulling (flashing) [MCU cfg] › LoRaMacPulling.elf › Thread #1 (Running : User Request); C:/IntelSWTools/ISSM_2016.1.067/tools/debugger/gdb-ia/bin/gdb-ia.exe (7.9)

Registers: Name | Value | Description — General Registers — General Purpose and FPU Register ...

```
/* QMSI timer app example */
int main(void)
{
    /* Variables */
    qm_pwm_config_t  sw_cfg;
    uint32_t lo_cnt, hi_cnt;
    uint32_t sw_lo_cnt = 0x4CCCA*10;//0x4CCCA = 0.01s = 10ms
    uint32_t sw_hi_cnt = 0x00;
```

```
LoRaMacPulling (flashing) [MCU cfg] C:/IntelSWTools/ISSM_2016.1.067/tools/debugger/gdb-ia/bin/gc
and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For information about how to find Technical Support, Product Updates,
User Forums, FAQs, tips and tricks, and other support information, ple
<http://www.intel.com/software/products/support/>.For help, type "hel
Type "apropos word" to search for commands related to "word".
```

Serial Terminal: COM6 [disconnected] — The port is not connected

Writable | Smart Insert | 59 : 9

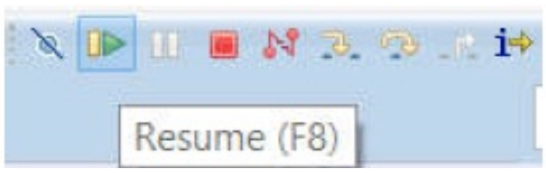
4.2. Setting and removing breakpoints

The simplest way to set or remove a breakpoint is by using the “Toggle Breakpoint” function. One way to do this is by right-clicking the blue vertical bar immediately to the left of the code line on which you want to set (or remove) a breakpoint; and selecting the Toggle Breakpoint menu item that appears in the right-click context menu.

4.3. Controlling code execution

You control code execution during a debug session by using the toolbar buttons shown in the screenshot below; by using their corresponding shortcut keystrokes (for example F5 > Step Into, F6 > Step Over, F8 > Resume, etc.); or by using the commands on the Run menu to control the code execution.

Follow the next steps to practice debugging your app:



🎬 Click the **Resume** toolbar icon once; your app will run to the breakpoint set in the previous step.

- Press the Resume icon again to continue running your app.
- Push **Suspend** to pause the app.

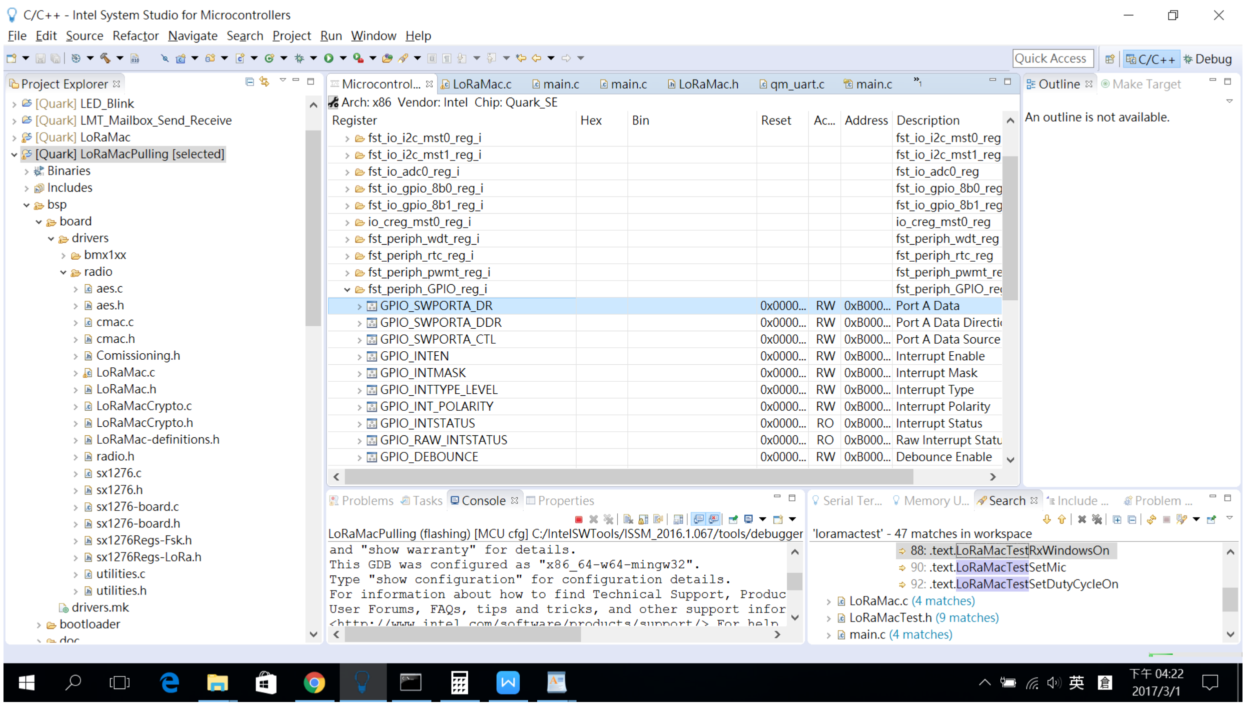
These commands are also accessible using the right-click menu inside the Debug window.

4.4. Adding a conditional breakpoint

Conditional breakpoints are useful for suspending code execution when a specific condition is met. Right-click on the vertical blue bar to the left of the code line where you want to add a conditional breakpoint and select the Add Breakpoint... menu item. The Properties for C/C++ Line Breakpoint dialog will appear. Enter the condition expression in the Condition text box.

4.5. The Microcontroller Registers view

The Microcontroller Registers view allows you to view and modify Intel® Quark™ Microcontroller peripheral registers while the app is running in Debug mode. This feature is useful when debugging on-chip peripherals and their interaction with external hardware.



```
[Quark] LED_Blink
[Quark] LMT_Mailbox_Send_Receive
[Quark] LoRaMac
[Quark] LoRaMacPulling [selected]
  Binaries
  Includes
  bsp
    board
      drivers
        bmx1xx
        radio
          aes.c
          aes.h
          cmac.c
          cmac.h
          Comissioning.h
          LoRaMac.c
          LoRaMac.h
          LoRaMacCrypto.c
          LoRaMacCrypto.h
          LoRaMac-definitions.h
          radio.h
          sx1276.c
          sx1276.h
          sx1276-board.c
          sx1276-board.h
          sx1276Regs-Fsk.h
          sx1276Regs-LoRa.h
          utilities.c
          utilities.h
      drivers.mk
    bootloader
    doc
```

Arch: x86 Vendor: Intel Chip: Quark_SE

Register	Hex	Bin	Reset	Ac...	Address	Description
fst_io_i2c_mst0_reg_i						fst_io_i2c_mst0_reg
fst_io_i2c_mst1_reg_i						fst_io_i2c_mst1_reg
fst_io_adc0_reg_i						fst_io_adc0_reg
fst_io_gpio_8b0_reg_i						fst_io_gpio_8b0_reg
fst_io_gpio_8b1_reg_i						fst_io_gpio_8b1_reg
io_creg_mst0_reg_i						io_creg_mst0_reg
fst_periph_wdt_reg_i						fst_periph_wdt_reg
fst_periph_rtc_reg_i						fst_periph_rtc_reg
fst_periph_pwmt_reg_i						fst_periph_pwmt_re
fst_periph_GPIO_reg_i						fst_periph_GPIO_re
GPIO_SWPORTA_DR			0x0000...	RW	0xB000...	Port A Data
GPIO_SWPORTA_DDR			0x0000...	RW	0xB000...	Port A Data Directi
GPIO_SWPORTA_CTL			0x0000...	RW	0xB000...	Port A Data Source
GPIO_INTEN			0x0000...	RW	0xB000...	Interrupt Enable
GPIO_INTMASK			0x0000...	RW	0xB000...	Interrupt Mask
GPIO_INTTYPE_LEVEL			0x0000...	RW	0xB000...	Interrupt Type
GPIO_INT_POLARITY			0x0000...	RW	0xB000...	Interrupt Polarity
GPIO_INTSTATUS			0x0000...	RO	0xB000...	Interrupt Status
GPIO_RAW_INTSTATUS			0x0000...	RO	0xB000...	Raw Interrupt Statu
GPIO_DEBOUNCE			0x0000...	RW	0xB000...	Debounce Enable

An outline is not available.

```
LoRaMacPulling (flashing) [MCU cfg] C:/IntelSWTools/ISSM_2016.1.067/tools/debugger
and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For information about how to find Technical Support, Produc
User Forums, FAQs, tips and tricks, and other support infor
<http://www.intel.com/software/products/support/> For help
```

```
'loramactest' - 47 matches in workspace
    88: .text.LoRaMacTestRxWindowsOn
    90: .text.LoRaMacTestSetMic
    92: .text.LoRaMacTestSetDutyCycleOn
  LoRaMac.c (4 matches)
  LoRaMacTest.h (9 matches)
  main.c (4 matches)
```

4.6. Other debug views

There are multiple debug views available – feel free to explore them on your own. Here are just a few of the more commonly used views:

- Disassembly view
- Registers view
- Memory view

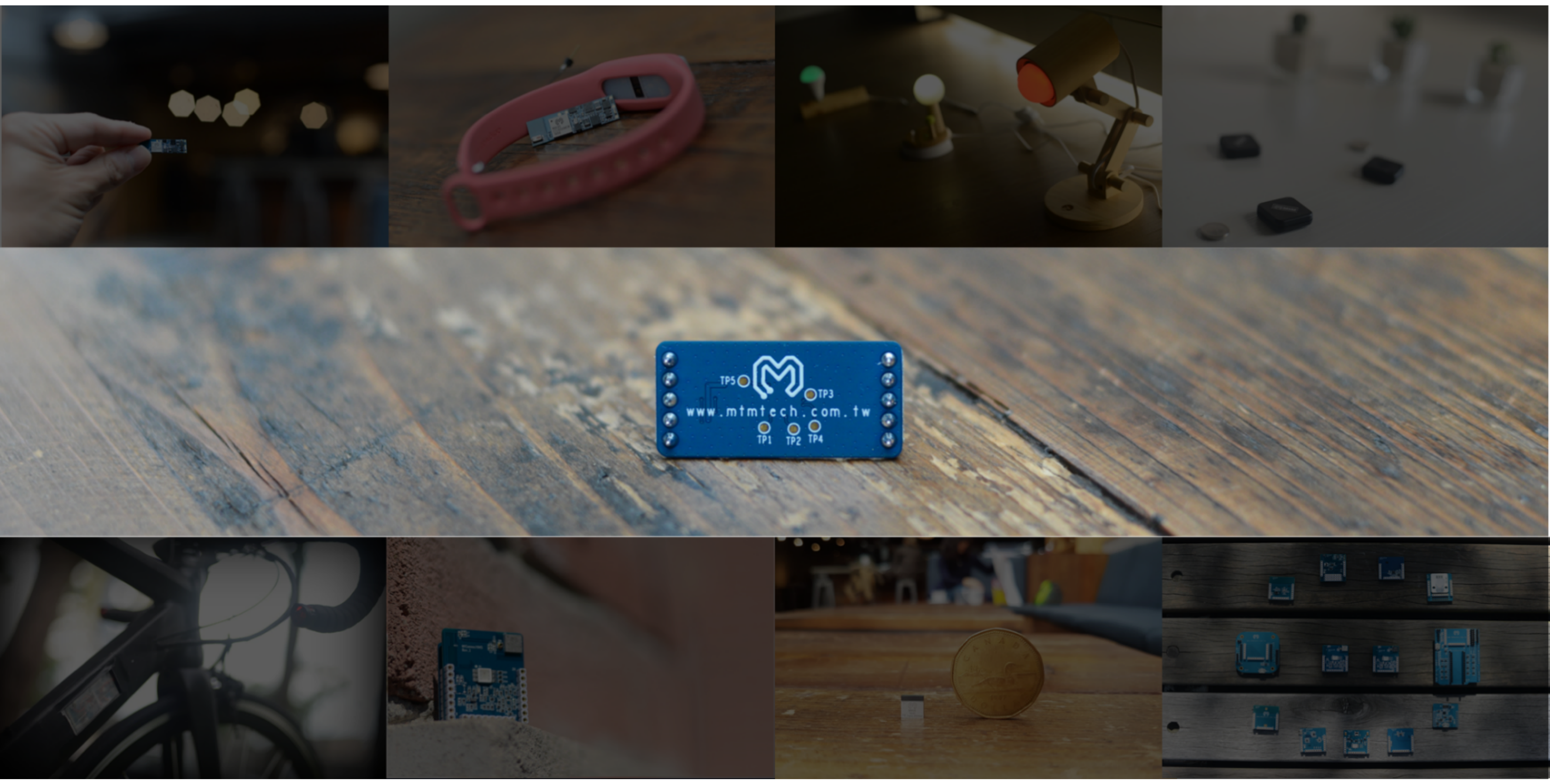
Use Windows > Show View menu to select the views. Note that the views available will vary depending on the active perspective.

4.7. OpenOCD view

The Intel® System Studio for Microcontroller uses OpenOCD software to communicate with the Intel® Quark™ SE Microcontroller C1000 through its JTAG interface. This software drives the OpenOCD view. In some cases, for example when the board is disconnected and reconnected from its USB connection, it might be necessary to restart OpenOCD. This can be done by clicking the Stop OpenOCD (red “stop” icon) button, and then clicking the Start OpenOCD (green “play” icon) button at the top right corner of the OpenOCD view. You must restart the debugging session after OpenOCD had been restarted.

5.Reference:

- Quark-c1000-datasheet
- Intel Quark SE Microcontroller C1000 Platform Design Guide
- SX1276 Datasheet
- LoRaWAN Specification 1R02
- LoRaWAN_Regional_Parameters_v1_0-20161012_1397_1
- http://stackforce.github.io/LoRaMac-doc/



MtM+ Technology Corporation

sales@mtmtech.com.tw

7F, 178, Sec. 3, Minquan East Rd.
Songshan District, Taipei,
Taiwan (R.O.C.)

+886-2-7736-7386

https://www.mtmtech.com.tw/

https://blog.mtmtech.com.tw/

https://www.facebook.com/MtMTechnologyCorporation

https://www.Instagram.com/mtmtech