Microchip LoRaWAN Stack

Quick Start Guide

Introduction

The Microchip LoRaWAN stack (MLS) provides a solution for the LoRaWAN end-device that is used for Internet of Things (IoT) applications. A typical LoRaWAN network is composed of end-devices, gateways, network servers and application servers.

End-devices are simple objects such as sensors and actuators, and are the “things” in the IoT. An end-device communicates to the network server through one or many gateways. The gateway acts as a concentrator for the end-devices and relays the data between end-devices and the network server.

The network server is the manager of a given network that handles the activation and manages end-devices. A network server can communicate to many application servers. The application servers process the data sent by the end-devices.

The wireless connection between an end-device and the gateway is setup through a LoRa wireless link. The gateways, network server and application servers communicate over IP back haul linked using Ethernet, 3G, LTE etc.

Table of Contents

1. Kit Overview

Microchip LoRa Stack

The main items provided as part of the LoRaWAN stack are:

- Atmel Studio 7.0 project for reference application:
  - End-device application
- LoRaWAN stack components as a static library (libLORAWAN_LIBGEN.a)
- Drivers, timer module for LoRaWAN stack

LoRaWAN Stack Directory Structure

The LoRaWAN stack code base is available in the directory present in the package (`src/ASF/thirdparty/wireless/lorawan`).

Table 1-1. Directory Structure of LoRaWAN Stack

Directory	Description
`./hal`	This directory contains implementation for radio's hardware interface, timers etc.
`./inc`	This directory contains common include file(s)
`./mac`	This directory contains headers of LoRaWAN MAC layer specification independent of regional parameters
`./regparams/<region>`	This directory contains implementation of MAC layer functionality specific to a particular region. For example: “eu” directory contains implementations for EU (863 MHz – 870 MHz) region
`./services`	This directory contains modules such as software timer and AES
`./sys`	This directory contains system modules such as task manager, power management and initialization
`./tal`	This directory contains transceiver related headers, drivers for supported transceivers
`./pmm`	This directory contains power mangement module (PMM)
`./libgen`	This directory contains the static library for LoRaWAN MAC and TAL.

Table 1-2. Supported Hardware Platforms and IDE

Name in this document	MCU	Transceiver	Evaluation Kits	IDE
SAML21XPRO +SX1276	ATSAML21J18B	Semtech SX1276	SAML21 Xplained Pro and SX1276 wing board	Atmel Studio 7.0
SAMR34 XPLAINED PRO	ATSAMR34J18B	Semtech SX1276	SAMR34 XPLAINED PRO	Atmel Studio 7.0

2. Stack Configuration

The LoRaWAN stack is configured according to the application needs. This requires a variety of build switches (macros) to be set for both the core stack and for the regional configurations. The stack configuration parameters are available in `conf_stack.h`. The regional configuration parameters are available in `conf_reg_params.h`.

Table 2-1. Stack configuration parameters

Macro definition	Default value	Description
FEATURE_CLASSC	1	Includes the functionality of Class C
FEATURE_DL_MCAST	1	Includes downlink multicast functionality
LORAWAN_SUPPORTED_ED_CLASSES	5 (enables Class A and C)	Default device classes supported

Table 2-2. Regional Configuration parameters

Macro definition	Default value	Description
MAC_DEF_TX_POWER	For EU: 1; For NA: 7	Transmission power table index
MAC_TX_CURRENT_DATARATE	For EU: DR5; For NA: DR0	Initial data rate to be used by application for uplink
MAC_DATARATE_MIN	For EU: DR7; For NA: DR4	Minimum data rate to be used by end-device
MAC_DATARATE_MAX	DR0 (for both EU and NA)	Maximum data rate to be used by end-device

3. Accessing and Modifying Stack Parameters

Besides configuring parameters at compile time, the LoRaWAN stack provides APIs for the application to access and modify the stack parameters. The application can access and modify both MAC and TAL layer parameters. These parameters are listed in the table at Stack Attributes.

1. **Accessing LoRaWAN/MAC parameters**
   The MAC layer provides an API to read the MAC parameters. It is available at `lorawan.h`.

   ```
   StackRetStatus_t LORAWAN_GetAttr(LorawanAttributes_t attrType, void *attrInput, void
   *attrOutput);
   ```

   Example: EndDevice_Demo application uses this API to read the Tx Power.

   ```
   uint8_t retValue;
   StackRetStatus_t status = LORAWAN_GetAttr(TX_POWER, NULL, &retValue);
   ```

2. **Modifying LoRaWAN/MAC parameters**
   The MAC layer provides an API to write the MAC parameters. It is available at `lorawan.h`.

   ```
   StackRetStatus_t LORAWAN_SetAttr(LorawanAttributes_t attrType, void *attrValue);
   ```

   Example: EndDevice_Demo application uses this API to write the Application Session Key.

   ```
   uint8_t appsKey[16] = APP_SESSION_KEY;
   StackRetStatus_t status = LORAWAN_SetAttr (APPS_KEY, appsKey);
   ```

3. **Accessing RADIO/TAL parameters**
   The TAL layer provides an API to write the TAL parameters. It is available at `radio_get_set.h`.

   ```
   RadioError_t RADIO_GetAttr(RadioAttribute_t attribute, void *value);
   ```

   Example: In `IncludeMacCommandsResponse()`, MAC uses this API to read the SNR of the latest packet in response to `DevStatusReq` command.

   ```
   int8_t packetSNR;
   RadioError_t status = RADIO_GetAttr(PACKET_SNR,(void *)&packetSNR);
   ```

4. **Modifying RADIO/TAL parameters**
   The TAL layer provides an API to write the TAL parameters. It is available at `radio_get_set.h`.

   ```
   RadioError_t RADIO_SetAttr(RadioAttribute_t attribute, void *value);
   ```

   Example: In `LORAWAN_Reset()`, the MAC uses this API to set the sync word of LoRa modulation.

   ```
   loRa.syncWord = MAC_LORA_MODULATION_SYNCWORD;
   RadioError_t status = RADIO_SetAttr(LORA_SYNC_WORD, (void *)&(loRa.syncWord));
   ```

Note:

- **StackRetStatus_t** is an enum available in `lorawan.h`
- **RadioError_t** is an enum available in `radio_interface.h`

4. Stack Attributes

4.1 LoRaWAN/MAC Attributes

Table 4-1. LoRaWAN/MAC Attributes

Name	Type	Range	Access	Default
ACKTIMEOUT	Unsigned 16-bit integer	0x0000-0xffff	Read/Write	0x7d0
ADR	Boolean	True (Enabled), False (Disabled)	Read/Write	--
ADR_ACKDELAY	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	0x20
ADR_ACKLIMIT	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	0x40
APPS_KEY	Unsigned 8-bit integer array[16]	--	Read/Write	--
APP_EUI	Unsigned 8-bit integer array[8]	--	Read/Write	--
APP_KEY	Unsigned 8-bit integer array[16]	--	Read/Write	--
AUTOREPLY	Boolean	True, False	Read/Write	--
BATTERY	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	0xff (BATTERY_LEVEL_INVALID)
CH_PARAM_FREQUENCY	Unsigned 32-bit integer	0x00000000 - 0xffffffff	Read/Write	--
CH_PARAM_DR_RANGE	Unsigned 8-bit integer	--	Read/Write	--
CH_PARAM_DUTYCYCLE	Unsigned 16-bit integer	0x0000 - 0xffff	Read/Write	--
CH_PARAM_STATUS	Boolean	True, False	Read/Write	--
CURRENT_DATARATE	Unsigned 8-bit integer	DR0 - DR7	Read/Write	DR0
DEV_ADDR	Unsigned 32-bit integer	0x00000000 - 0xffffffff	Read/Write	--

Name	Type	Range	Access	Default
DEV_EUI	Unsigned 8-bit integer array[8]	--	Read/Write	--
DOWNLINK_COUNTER	Unsigned 32-bit integer	0x00000000 - 0xffffffff	Read/Write	0x00000000
EDCLASS	Unsigned 8-bit integer	0 (Class A), 1 (Class B - not supported), 2 (Class C)	Read/Write	0 (Class A)
EDCLASS_SUPPORTED	Unsigned 8-bit integer	1 (Class A), 5 (Class A and Class C)	Read/Write	LORAWAN_SUPPORTED_ED_CLASSES
FHSS_CALLBACK	FHSSCallback_t (function pointer)	Valid function address	Read/Write	--
ISMBAND	Unsigned 8-bit integer	0x00 - 0xff	Read Only	ISM_EU868
JOINACCEPT_DELAY1	Unsigned 16-bit integer	0x0000 – 0xffff	Read/Write	0x1388
JOINACCEPT_DELAY2	Unsigned 16-bit integer	0x0000 – 0xffff	Read/Write	0x1770
LINK_CHECK_GWCNT	Unsigned 8-bit integer	0x00 – 0xff	Read Only	0x00
LINK_CHECK_MARGIN	Unsigned 8-bit integer	0x00 – 0xff	Read Only	0xff
LINK_CHECK_PERIOD	Unsigned 16-bit integer	0x0000 – 0xffff	Read/Write	0x0000
LORAWAN_STATUS	Unsigned 32-bit integer	0x00000000 – 0xffffffff	Read Only	0x00
MAX_FCOUNT_GAP	Unsigned 16-bit integer	0x0000 -0xffff	Read/Write	0x4000
MCAST_APPS_KEY	Unsigned 8-bit integer array[16]	--	Read/Write	--
MCAST_ENABLE	Boolean	True, False	Read/Write	0x00
MCAST_FCNT_DOWN	Unsigned 16-bit integer	0x0000 - 0xffff	Read	--
MCAST_GROUP_ADDR	Unsigned 32-bit integer	0x00000000 - 0xffffffff	Read/Write	--
MCAST_NWKS_KEY	Unsigned 8-bit integer array[16]	--	Read/Write	--

Name	Type	Range	Access	Default
NWKS_KEY	Unsigned 8-bit integer array[16]	--	Read/Write	--
PRESCALAR	Unsigned 16-bit integer	0x0000 -0xffff	Read Only	0x0001
CNF_RETRANSMISSION_NUM	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	--
UNCNF_REPETITION_NUM				
RX2_WINDOW_PARAMS	ReceiveWindow2Params_t (enum)	--	Read/Write	Specific to the region
RX_DELAY1	Unsigned 16-bit integer	0x0000 -0xffff	Read/Write	0x3e8
RX_DELAY2	Unsigned 16-bit integer	0x0000 - 0xffff	Read Only	0x7d0
SYNC_WORD	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	0x34
TX_POWER	Unsigned 8-bit integer	For EU: 0 to 5, For NA: 5, 7, 8, 9, 10	Read/Write	For EU: 1, For NA: 7
UPLINK_COUNTER	Unsigned 32-bit integer	0x00000000 - 0xffffffff	Read/Write	0x00000000

4.2 RADIO/TAL Attributes

Table 4-2. Radio/TAL Attributes

Name	Type	Range	Access	Default
BANDWIDTH	RadioLoRaBandwidth_t (enum)	Only the enumerated values	Read/Write	BW_125KHZ
CHANNEL_FREQUENCY	Unsigned 32-bit integer	Only valid frequency value	Read/Write	For EU: FREQ_868100KHZ, For NA: FREQ_923300KHZ
CHANNEL_FREQUENCY_DEVIATION	Unsigned 32-bit integer	0x00000000 – 0xffffffff	Read/Write	0x61a8
CRC_ON	Unsigned 8-bit integer	0x00 (Disabled), 0x01 (Enabled)	Read/Write	0x01 (Enabled)
ERROR_CODING_RATE	RadioErrorCodingRate_t (enum)	Only the enumerated values	Read/Write	CR_4_5

Name	Type	Range	Access	Default
FREQUENCY_HOP_PERIOD	Unsigned 16-bit integer	0x0000 - 0xffff	Read Only	0x0000
FSK_AFC_BW	RadioFSKShaping_t (enum)	Only the enumerated values	Read/Write	FSKBW_83_3KHZ
FSK_BIT_RATE	Unsigned 32-bit integer	0x00000000 – 0xffffffff	Read/Write	0xc350
FSK_DATA_SHAPING	RadioFSKShaping_t (enum)	Only the enumerated values	Read/Write	FSK_SHAPING_GAUSS_BR_0_5
FSK_RX_BW	RadioFSKShaping_t (enum)	Only the enumerated values	Read/Write	FSK_BW_50_0KHZ
FSK_SYNC_WORD	Unsigned 8-bit integer array [8]	0x00 - 0xff (in each octet)	Read/Write	{0xc1, 0x94, 0xc1}
FSK_SYNC_WORD_LEN	Unsigned 8-bit integer	0x00 - 0x08	Read/Write	0x03
IQINVERTED	Unsigned 8-bit integer	0x00 (Disabled), 0x01 (Enabled)	Read/Write	0x00 (Disabled)
LORA_SYNC_WORD	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	0x34
MODULATION	RadioModulation_t (enum)	Only the enumerated values	Read/Write	MODULATION_LORA
OUTPUT_POWER	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	0x01
PABOOST	Unsigned 8-bit integer	0x00 - 0xff	Read/Write	0x01
PACKET_SNR	Signed 8-bit integer	-128 to +127	Read Only	-128
PREAMBLE_LEN	Unsigned 16-bit integer	0x0000 - 0xffff	Read/Write	0x08
RADIO_CALLBACK	RadioCallback_t (function pointer)	Valid function address	Read/Write	--
RADIO_LBT_PARAMS	RadioLBT_t (structure)	Depends on the data type of each member of the structure	Read/Write	{0x00}
SPREADING_FACTOR	RadioDataRate_t (enum)	Only the enumerated values	Read/Write	SF_12
WATCHDOG_TIMEOUT	Unsigned 32-bit integer	0x00000000 – 0xffffffff	Read/Write	0x3a98

5. LoRaWAN/MAC API

LORAWAN_Init

```
void LORAWAN_Init(AppDataCb_t appdata, JoinDataCb_t joindata)
```

Parameters:

- `appdata` -- pointer to function that is called when a downlink is received
- `joindata` -- pointer to function that is called when join response is received

This function initializes the LoRaWAN stack and the radio module.

LORAWAN_Join

```
StackRetStatus_t LORAWAN_Join(ActivationType_t activationTypeNew)
```

Parameters:

- `activationTypeNew` -- activation method to be used for joining

This function starts the LoRaWAN activation procedure.

Returns: `StackRetStatus_t` is an enum. Status of the activation procedure.

LORAWAN_Send

```
StackRetStatus_t LORAWAN_Send (LorawanSendReq_t *lorasendreq)
```

Parameters:

- `*lorasendreq` -- pointer to `LorawanSendReq_t` structure. This structure contains transmissionType (unconfirmed or confirmed), FPort, pointer to application data and length of the application data

This function starts the uplink from the end-device. The application can optionally retain the lorasendreq pointer until the completion of the transaction, as the same pointer will be returned by LORAWAN/MAC to help the application identify the specific send request.

Returns: `StackRetStatus_t` is an enum. Status of the uplink procedure.

LORAWAN_GetAttr

```
StackRetStatus_t LORAWAN_GetAttr(LorawanAttributes_t attrType, void *attrInput, void *attrOutput)
```

Parameters:

- `attrType` -- type of the attribute
- `*attrInput` -- pointer to input. This input corresponds to the attribute that is received from the LoRaWAN/MAC attribute list. For example, when getting channel status, this parameter is used as the input to the channel index.
- `*attrOutput`-- pointer to the output variable. This is where the attribute value that is read by the application is written.

LORAWAN_GetAttr

This function reads the value of given attribute from the LoRaWAN/MAC attribute list.

Returns: `StackRetStatus_t` is an enum. Status of the get attribute procedure.

LORAWAN_SetAttr

```
StackRetStatus_t LORAWAN_SetAttr(LorawanAttributes_t attrType, void *attrValue)
```

Parameters:

- `attrType` -- type of the attribute
- `*attrOutput` -- pointer to the variable that contains the value to be written into the attribute

This function writes the given value to the specified attribute.

Returns: `StackRetStatus_t` is an enum. Status of the set attribute procedure.

LORAWAN_Pause

```
uint32_t LORAWAN_Pause (void)
```

This function determines whether the LoRaWAN protocol functionality of the MAC is paused and allows the application to use it for sending and receiving payload of some other protocol. If this function finds the pause is possible, then it pauses the MAC and returns the amount of time that is paused.

Returns: Amount of time the LoRaWAN protocol functionality of the MAC is paused.

LORAWAN_Resume

```
void LORAWAN_Resume (void)
```

This function resumes the LoRaWAN protocol functionality of the MAC. This is used to force the resume from a previous `LORAWAN_Pause` before the pause time elapses.

LORAWAN_ForceEnable

```
void LORAWAN_ForceEnable (void)
```

The transmission by the end-device is disabled from the network server by sending the `DutyCycleReq` command with the `maxDutyCycle` parameter as the highest possible value. If the transmission is disabled, this function is used to enable the transmission again in the LoRaWAN MAC by the end-device.

LORAWAN_ReadyToSleep

Parameters:

- `deviceReadyToSleep` - indicates whether the device will reset at wake-up. ‘true’ means the device will reset at wake-up. ‘true’ is used in case of sleep modes where execution resumes at the

LORAWAN_ReadyToSleep
reset vector. ‘false’ means the device will not reset at wake-up. This is used in sleep modes where execution resumes at instruction immediately after the sleep instruction.
`bool LORAWAN_ReadyToSleep(bool deviceResetAfterSleep)`
This function checks whether or not the stack is ready to sleep.
Returns: 'true' if stack is in ready state for sleep, otherwise 'false'

6. RADIO/TAL API

RADIO_Init

```
void RADIO_Init(void);
```

This function initializes the radio module.

RADIO_Receive

```
RadioError_t RADIO_Receive(RadioReceiveParam_t *param)
```

Parameters:

- `param` -- pointer to `RadioReceiveParam_t` structure. This structure contains receive type(start or stop), duration of receive in symbols

This function either starts or stops the receive. Duration in symbols is applied for start.

Returns: `RadioError_t` is an enum. Status of the request whether it is succeeded or not.

RADIO_Transmit

```
RadioError_t RADIO_Transmit (RadioTransmitParam_t *param)
```

Parameters:

- `param` -- pointer to `RadioTransmitParam_t` structure. This structure contains the pointer to the transmit buffer, and the length of bytes to be transmitted

This function transmits requested length of bytes from the buffer.

Returns: `RadioError_t` is an enum. Status of the request whether it is succeeded or not.

RADIO_TransmitCW

```
RadioError_t RADIO_TransmitCW(void)
```

This function transmits a continuous wave.

Returns: `RadioError_t` is an enum. Status of the request whether it is succeeded or not.

RADIO_StopCW

```
RadioError_t RADIO_StopCW(void)
```

This function stops the transmission of continuous wave.

Returns: `RadioError_t` is an enum. Status of the request whether it is succeeded or not.

RADIO_FHSSChangeChannel

```
void RADIO_FHSSChangeChannel(void)
```

This function sets the receive frequency of the transceiver while hopping in FHSS

RADIO_SetAttr

```
RadioError_t RADIO_SetAttr(RadioAttribute_t attrType, void *attr)
```

Parameters:

- `attrType` -- type of the radio attribute
- `*attr` -- pointer to the variable that contains the value to be written into the attribute

This function writes the given value to the specified attribute.

Returns: `RadioError_t` is an enum. Status of the set attribute procedure.

RADIO_GetAttr

```
RadioError_t RADIO_GetAttr (RadioAttribute_t attrType, void *attr)
```

Parameters:

- `attrType` -- type of the radio attribute
- `*attr` -- pointer to the variable that contains the value to be read for the attribute

This function reads the parameters of the requested attribute.

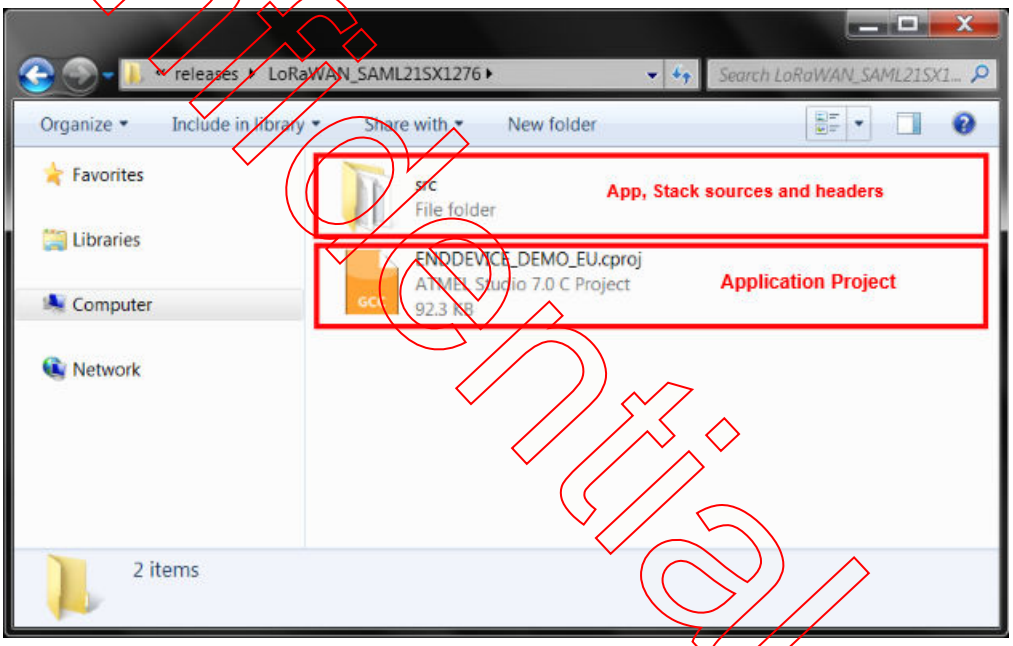
Returns: `RadioError_t` is an enum. Status of the set attribute procedure.

7. Example Application - EndDevice_Demo

The LoRaWAN stack software package includes an example application that is built and programmed on the supported hardware platform and is executed immediately. The source code of the application is provided, thus helping application developers to understand the proper use of the stack and allowing modifications that are required for the custom applications to be made.

This example application (EndDevice_Demo) demonstrates the LoRaWAN end-device joining, and data uplink and downlink over LoRaWAN. There are two separate EndDevice_Demo application projects available for EU and NA regions. The application projects for both regions are available in the root location of the corresponding packages.

Figure 7-1. Example Application - EndDevice_Demo

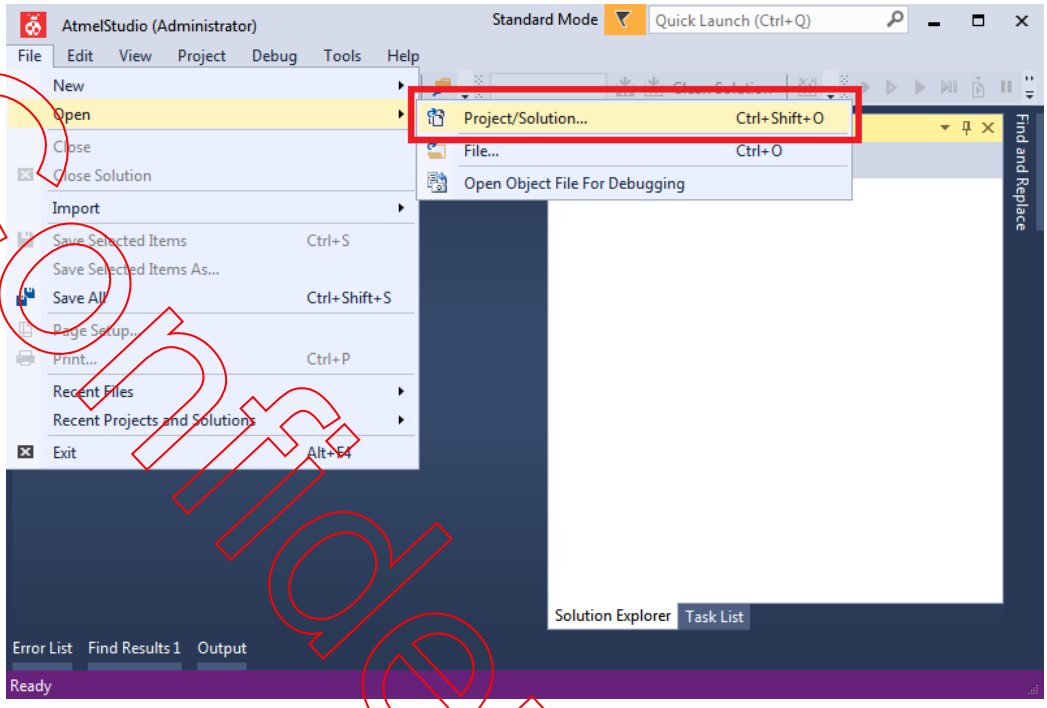


7.1 Building the Firmware

Perform the following steps in Atmel Studio to build the firmware for EndDevice_Demo.

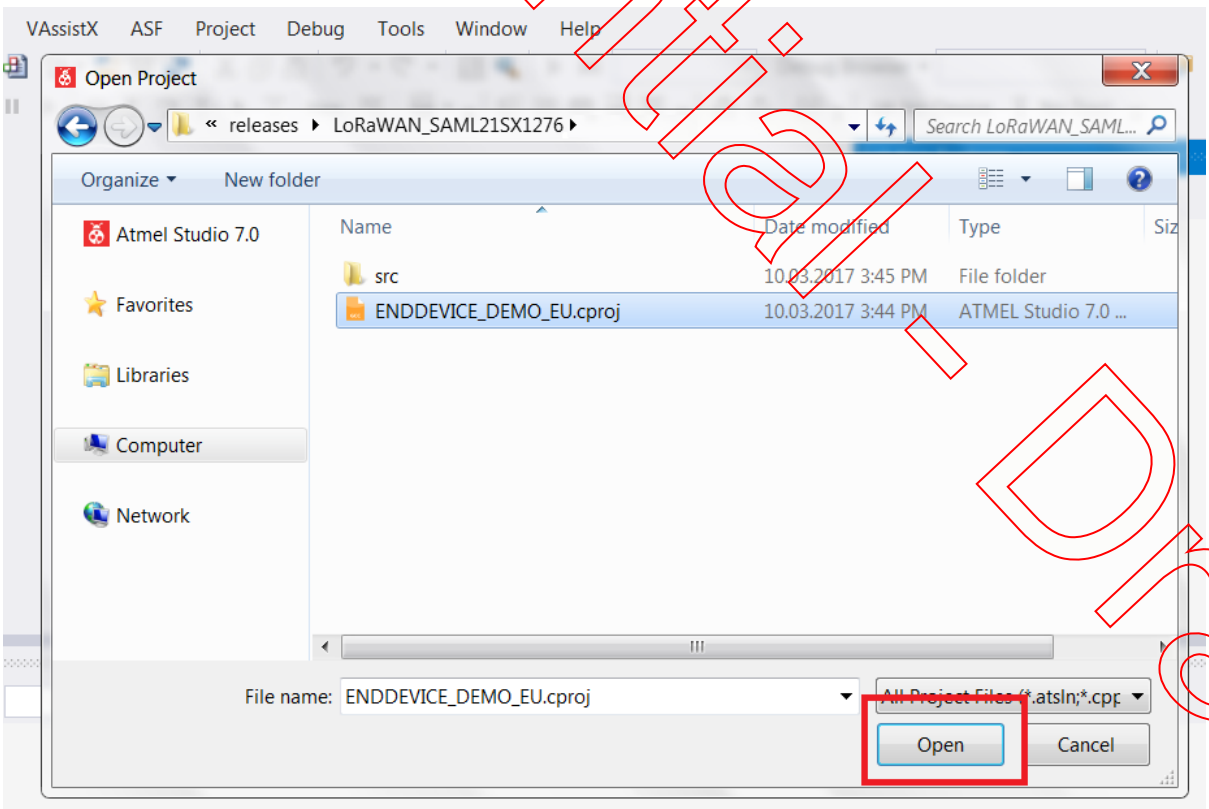
1. Download and unzip the package “ZIP file” (for example: `apps_enddevice_demo_multiband.zip`) at any location. For example: *D:\releases \LoRaWAN_SAML21SX1276*. For ease, we refer the package location as **“PACKAGE_ROOT”**.
2. Open Atmel Studio 7, from *Start Menu* > *Atmel Studio 7.0* > *Atmel Studio 7.0*
3. Go to *File* > *Open* > *Project/Solution*.

Figure 7-2. Selecting Project/Solution



4. Select the application project file from the package.

Figure 7-3. Selecting the Application Project



5. By default, all the regions are enabled in the project. If user, wants disable certain regions, follow the steps given below.

Goto Project->Properties or Alt+F7

From the left hand pane, Choose “Toolchain”.

In the right hand pane, the “ARM/GNU c compiler”-> ”Symbols”.

In the next pane, the regional band macros are listed in the “Defined Symbols”,

AS_BAND=1

AU_BAND=1

EU_BAND=1

IND_BAND=1

JPN_BAND=1

KR_BAND=1

NA_BAND=1

Runtime support for the regional bands are enabled, when the macro for the corresponding regional band is set to ‘1’ and is disabled, when the macro is set to ‘0’
Eg:

The following are the macro values to enable only North American region (NA_BAND)

AS_BAND=0

AU_BAND=0

EU_BAND=0

IND_BAND=0

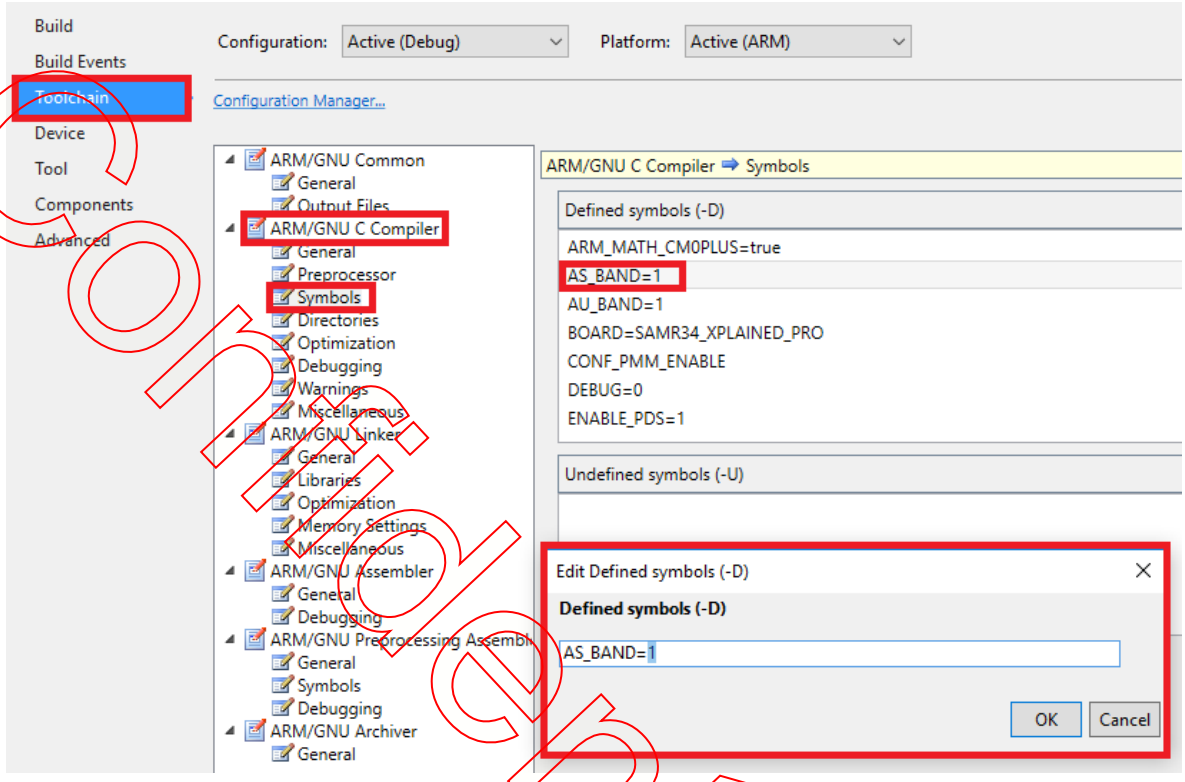
JPN_BAND=0

KR_BAND=0

NA_BAND=1

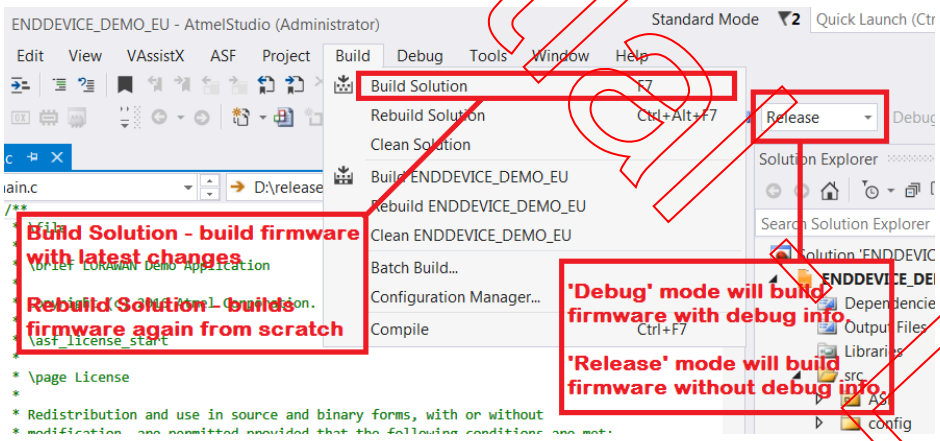
Note: The above select bands mentioned are from particular release, given as example. Make sure, before defining any band, that particular band is supported from release notes.

Figure 7-4. Selecting the Band



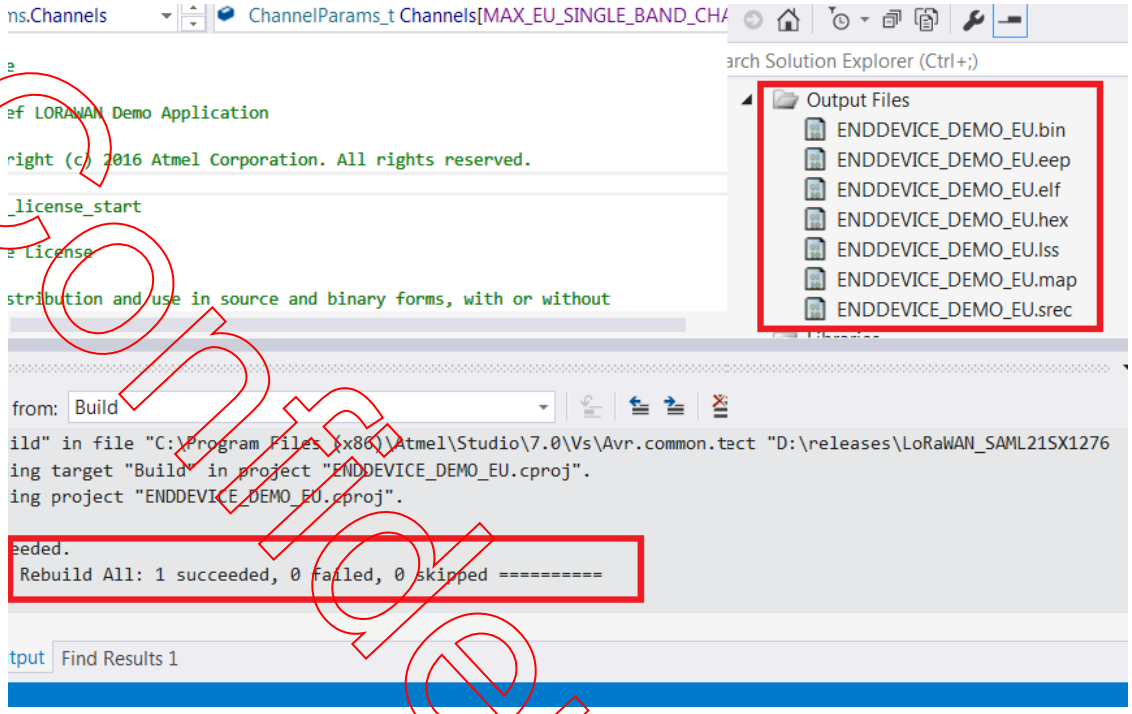
6. Go to *Build* > *Build Solution* to build the firmware.

Figure 7-5. Building the Firmware



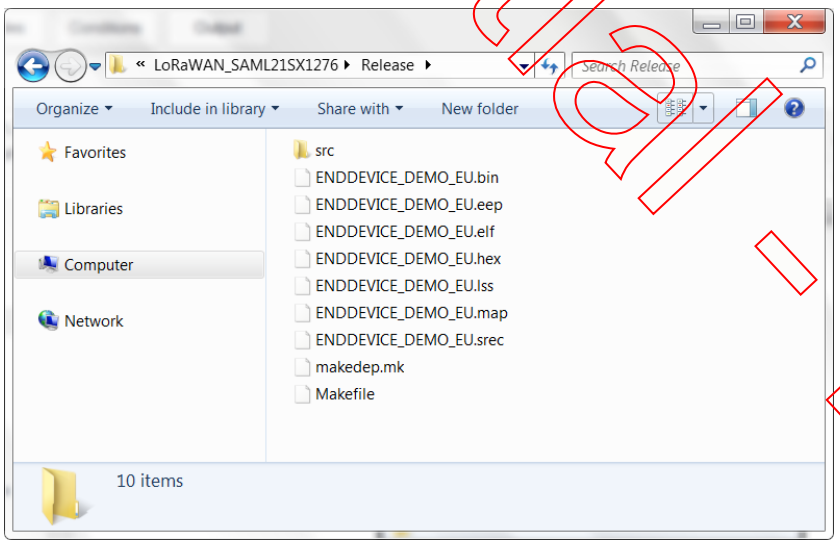
7. After the successful compilation and linking, firmware appears in the **Output Files** section of "Solution Explorer".

Figure 7-6. Output Files



In the file system, output files are saved in *PACKAGE_ROOT/<Build_Configuration>*. As per this example, it is present at *D:\releases\LoRaWAN_SAML21SX1276\Debug*. If **Release** is selected as the build configuration, then the firmware is saved at *D:\releases\LoRaWAN_SAML21SX1276\Release*.

Figure 7-7. Output File Location

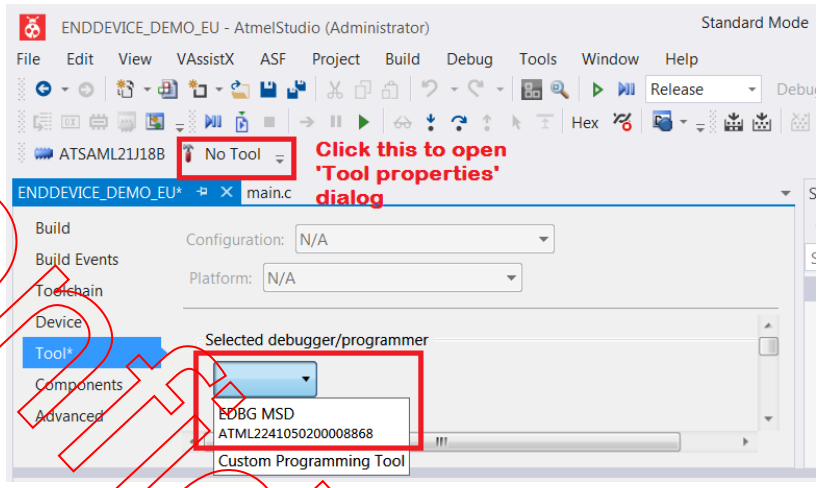


7.2 Flashing the Firmware

1. After successfully building the firmware, connect the SAML21 Xplained Pro board with the PC. Atmel Studio detects the board after completing the driver installation.
2. Go to *Project > ENDDEVICE_DEMO_EU Properties*. The Project Properties window appears. From the vertical tab of options, click **Tool**.

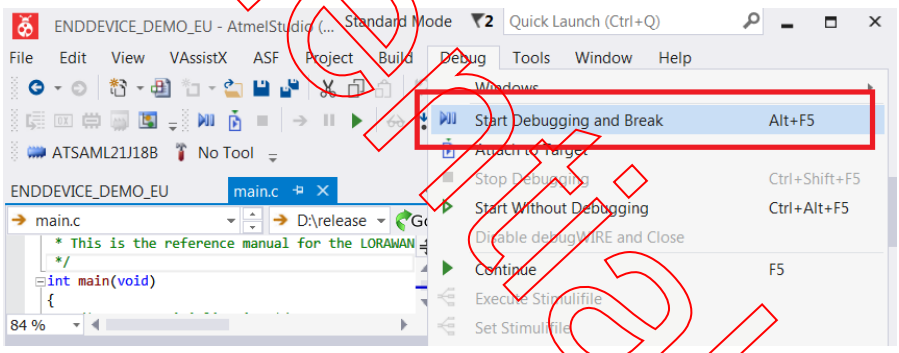
3. In the “Selected debugger/programmer” drop-down menu, select **SAML21 Xplained Pro board**.

Figure 7-8. Selecting the Supported Hardware Platform



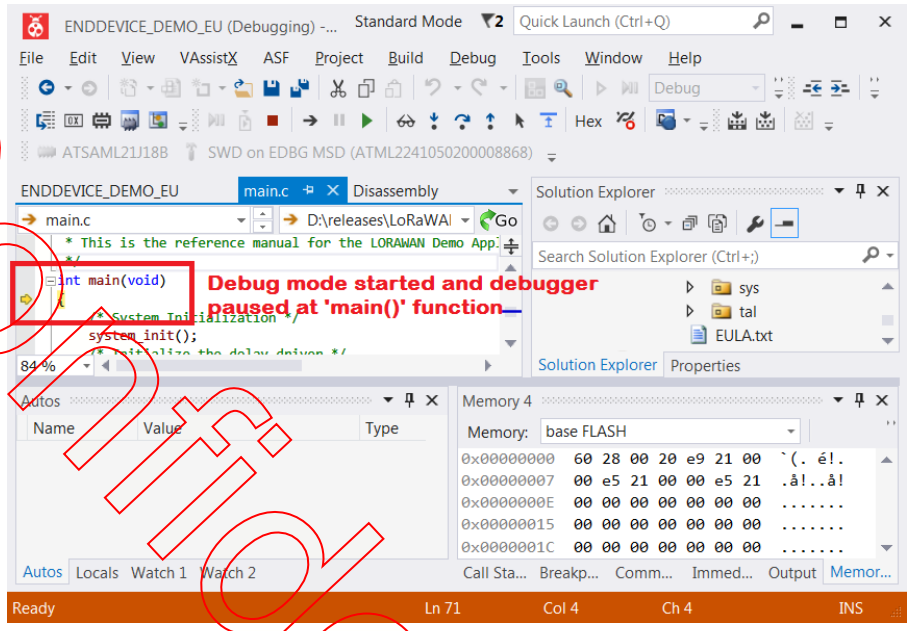
4. When the board is selected, it is ready for programming and debugging. Go to *Debug* > *Start Debugging and Break*.

Figure 7-9. Programming the Device



5. The firmware is flashed into the MCU and starts debugging automatically.

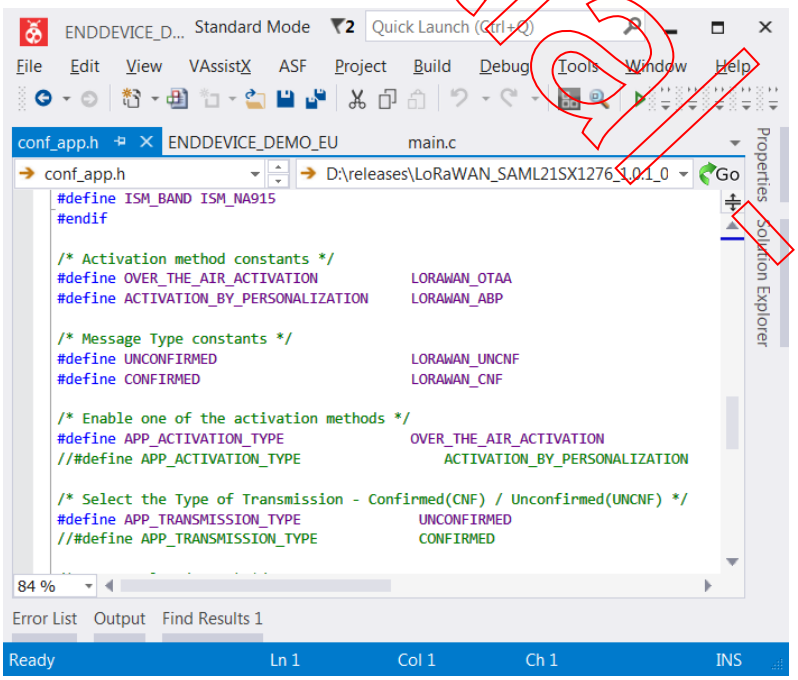
Figure 7-10. Debugging the Device



7.3 Application Configuration

The EndDevice_Demo application provides configurable parameters in `conf_app.h`. This file is available at `PACKAGE_ROOT/src/config`. These parameters are modified in the EndDevice_Demo application in conjunction with the stack configuration. Refer to Stack Configuration.

Figure 7-11. Configuring the Application



1. The application provides the method of device **activation**.

```
#define APP_ACTIVATION_TYPE              OVER_THE_AIR_ACTIVATION
//#define APP_ACTIVATION_TYPE            ACTIVATION_BY_PERSONALIZATION
```

2. The application provides the **message type** for sending data from end-device.

```
#define APP_TRANSMISSION_TYPE             UNCNF
//#define APP_TRANSMISSION_TYPE           CNF
```

3. The application mentions the **port** for uplink data.

```
#define APP_FPORT                          1
```

4. The application can modify/set the **DeviceAddress** (32-bit) to be used with ABP.

```
#define DEVICE_ADDRESS                     0x001AD9BB
```

5. The application can modify/set the **network and application session keys** to be used with ABP.

```
#define APPLICATION_SESSION_KEY  {0x41, 0x63, 0x74, 0x69, 0x6C, 0x69, 0x74, 0x79,
                                  0x00, 0x04, 0xA3, 0x0B, 0x00, 0x04, 0xA3, 0x0B}

#define NETWORK_SESSION_KEY      {0x61, 0x63, 0x74, 0x69, 0x6C, 0x69, 0x74, 0x79,
                                  0x00, 0x04, 0xA3, 0x0B, 0x00, 0x04, 0xA3, 0x0B}
```

6. The application can modify/set the **DevEUI** (64-bit) to be used with OTAA.

```
#define DEVICE_EUI               {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07}
```

7. The application can modify/set the **AppEUI** (64-bit) to be used with OTAA.

```
#define APPLICATION_EUI          {0xDA, 0xBB, 0xAD, 0x00, 0xDA, 0xBB, 0xAD, 0x00}
```

8. The application can modify/set the **AppKey** (128-bit) to be used with OTAA.

```
#define APPLICATION_KEY          {0xBA, 0xAD, 0xF0, 0x0D, 0xBA, 0xAD, 0xF0, 0x0D,
                                  0xBA, 0xAD, 0xF0, 0x0D, 0xBA, 0xAD, 0xF0,
0x0D}
```

9. The application can modify/set the **downlink multicast network and application session keys** to be used with Class C.

```
#define APP_MCAST_APP_SESSION_KEY  {0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
                                    0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6}

#define APP_MCAST_NWK_SESSION_KEY  {0x3C, 0x8F, 0x26, 0x27, 0x39, 0xBF, 0xE3, 0xB7,
                                    0xBC, 0x08, 0x26, 0x99, 0x1A, 0xD0, 0x50, 0x4D}
```

10. The application can modify/set the **Downlink Multicast Group Address** (32-bit) to be used with Class C.

```
#define APP_MCAST_GROUP_ADDRESS            0x0037CC56
```

11. The application can modify/set the **Downlink Multicast Enable** (Boolean) to be used with Class C.

```
#define APP_MCAST_ENABLE                   true
```

12. The application can modify/set the **End Device Class**. When Class C is chosen, and when the selection succeeds, the downlink multicast functionality is enabled by default

```
#define APP_ENDDEVICE_CLASS        CLASS_A
//#define APP_ENDDEVICE_CLASS      CLASS_C
```

8. Power Management

MLS provides power management module (PMM) in the stack. An application running on top of MLS can choose to use PMM to save power during idle times. Besides saving power during idle, PMM tries to reduce power consumption even during transaction. Power saving is done by switching the MCU to one of the available low-power modes. Currently, PMM is supported only on SAMR34 MCU. In SAMR34, only STANDBY sleep mode is currently supported in PMM. By default, PMM is added and enabled.

Table 8-1. Files that are implemented for power management

Filename	Description
`pmm/inc/pmm.h`	This file contains the public API for the power management module. This needs to be included by the application if power management is needed.
`pmm/src/pmm.c`	This file contains the implementation of power management i.e., the conditions for entering to sleep, calculating the sleep time, configuring and backing up the timers etc.
`hal/inc/sleep_timer.h`	This file contains the APIs for the sleep timer. It is used by the PMM to keep track of the sleep duration and also to wake up the device in timed sleep scenarios.
`hal/src/sleep_timer/sam0/sleep_timer.c`	This file contains the implementation of sleep timer. Currently, sleep timer uses RTC internally.
`hal/inc/sleep.h`	This file contains the API to switch the MCU to desired sleep mode.
`hal/src/sleep/sam0/src/sleep.c`	This file contains the implementation to switch the MCU to sleep mode. Currently, only STANDBY mode in SAMR34 is supported.

Table 8-2. Power Management API

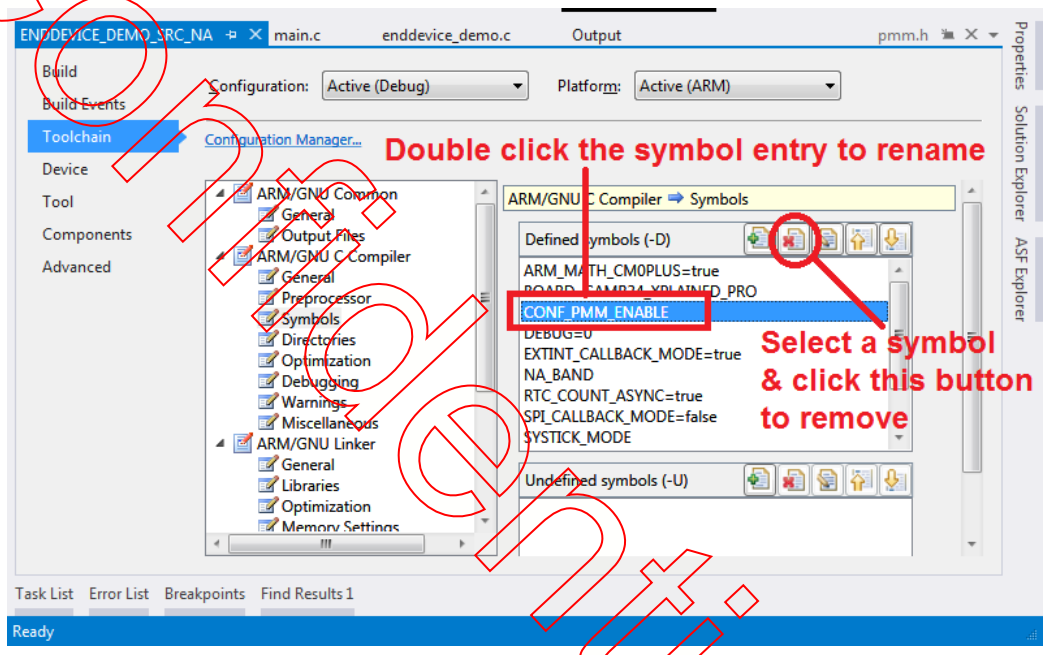
PMM_Sleep
`PMM_Status_t PMM_Sleep(PMM_SleepReq_t *req)`
Parameters: • `*req` -- Pointer to sleep request structure when being called from application
This function puts the system to sleep if possible
Returns: PMM_Status_t enumeration • **PMM_SLEEP_REQ_DENIED** when sleep is not possible at the instance • **PMM_SLEEP_REQ_PROCESSED** when sleep is possible and have already done

- PMM_SleepReq_t is a structure available in pmm.h
- PMM_Status_t is a enum available in pmm.h

8.1 Using PMM in Application

This section describes how to use PMM in user application. End Device demo application is used as example in this section. PMM is already included in end device demo application. When PMM is included in an application, it will define “CONF_PMM_ENABLE” macro as part of compiler flags. This flag controls the addition and removal of PMM in application. By default it is added to end device demo application.

Figure 8-1. Adding or removing PMM from application

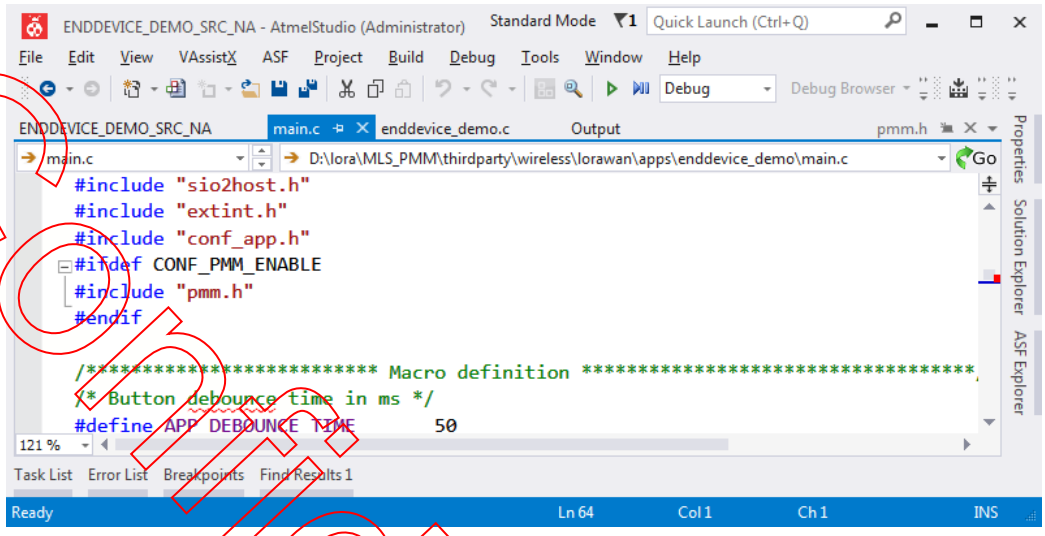


To remove PMM from end device demo application, remove or rename the “CONF_PMM_ENABLE” macro from the list of compiler flags. As shown in the above figure, “Symbols” dialog from “Toolchain” tab listed in project properties window. Then, either rename or remove this entry. It will then remove PMM from application firmware.

To use PMM in application, following steps are to followed. Before the following steps, PMM must be added to end device demo application.

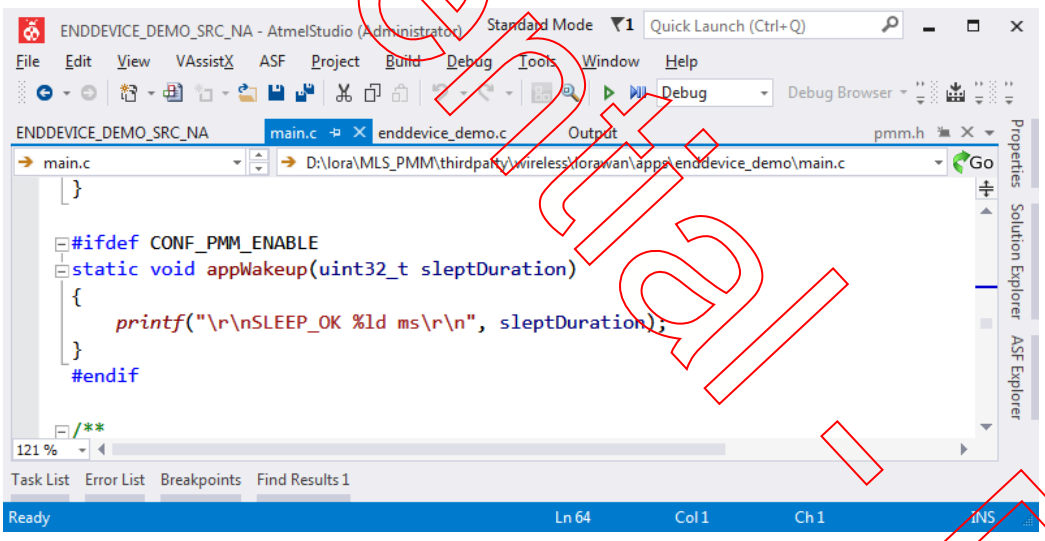
1. Include pmm.h to application. In end device demo application, PMM is included in main.c

Figure 8-2. Including PMM in application



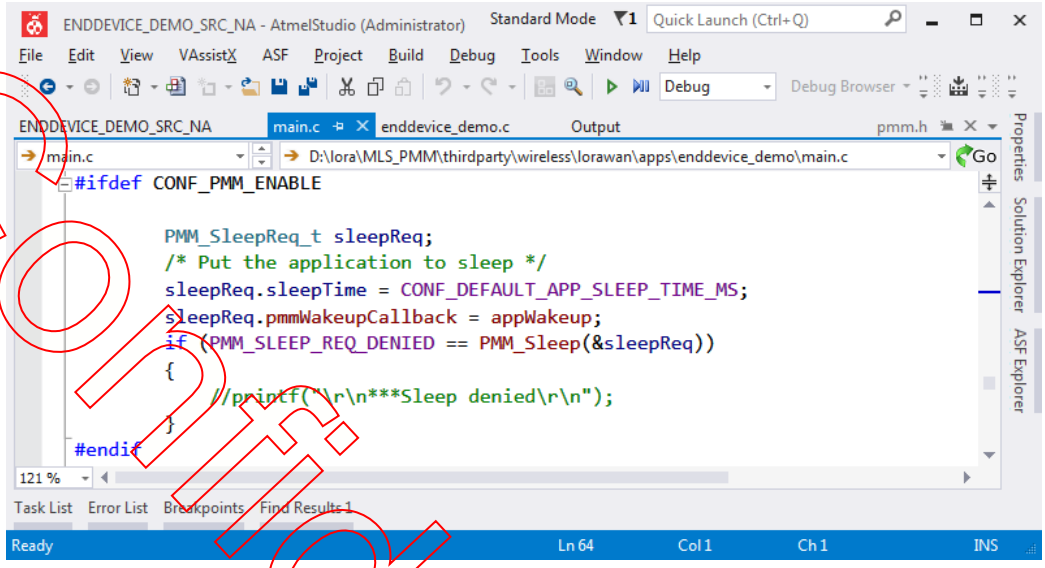
2. Implement a call back function to be invoked after wake up. This function must have the prototype of “pmmWakeupCallback” function pointer defined in PMM_SleepReq_t structure. PMM_SleepReq_t is available in pmm.h

Figure 8-3. Wake up callback function



3. Invoke PMM_Sleep function from application to request the PMM to put the system to sleep. PMM may deny a sleep request if the stack is not ready to sleep. User can supply NULL pointer to “pmmWakeupCallback” if wakeup callback function is not implemented.

Figure 8-4. Calling PMM_Sleep



Application sleep request time is configured by the macro “CONF_DEFAULT_APP_SLEEP_TIME_MS”. It is present in “conf_app.h” file. By default, application sleep time is given as 3 seconds. It can be changed to desired values. But, the sleep duration must fall within acceptable range which is given in the following table.

Table 8-3. Sleep duration values

Macro	Value	Description
PMM_SLEEP_TIME_MAX	130996480	Unit is milliseconds. This macro defines the highest acceptable sleep request duration by PMM_Sleep(). If any value lesser than PMM_SLEEP_TIME_MAX then, the sleep request will be deined. This value is defined in pmm.h
PMM_SLEEP_TIME_MIN	100	Unit is milliseconds. This macro defines the lowest acceptable sleep request duration by PMM_Sleep(). If any value lesser than PMM_SLEEP_TIME_MIN then, the sleep request will be deined. This value is defined in pmm.h
PMM_WAKEUP_TIME	10	Unit is milliseconds. This is the guard time used by PMM to perform wakeup. PMM will subtract PMM_WAKEUP_TIME from the requested sleep time when entering to sleep.

When end device is put to sleep, it can wake up from interrupt by either sleep timer or transceiver interrupt or GPIO interrupt. When wake up happens in end device PMM_Wakeup() function is called and it returns the duration elapsed during sleep to application. In case of application maintaining its own timers, this slept duration returned from PMM_Wakeup can be used to resume those timers. MLS automatically calls PMM_Wakeup whenever it receives sleep timer interrupt or external interrupt. But, the end device must call PMM_Wakeup for GPIO interrupts also. For those GPIO used by the application that can generate interrupts during sleep, user must call PMM_Wakeup in those ISR callbacks - as shown in the following code snippet. In case of polling, this is not required since, polling code works only after wake up.

```
user_gpio_isr()
```

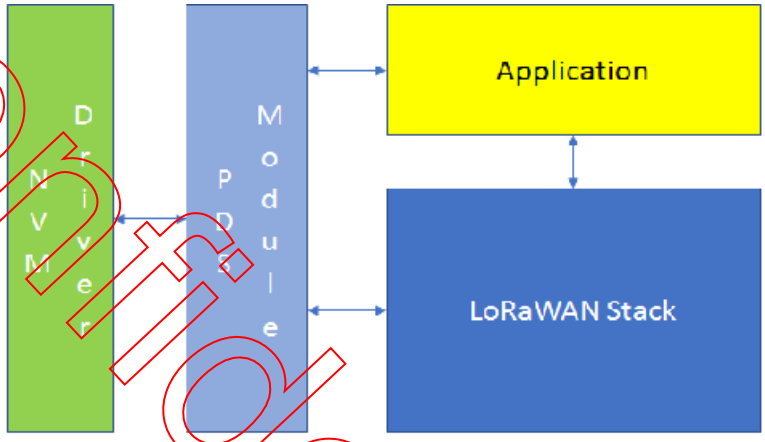
```
{
        PMM_Wakeup(); /* This MUST be called to properly resume after wakeup */

        /* TODO: Add code to service this interrupt */
        .
        .
        .
}
```

9. Persistent Data Server

Persistent Data Server(PDS) module facilitates storing of Stack paramters/attributes in Non-Volatile Memory(NVM) of MCU. PDS module interfaces between NVM driver and Stack

Figure 9-1. PDS Module



PDS Module Overview

PDS stands for Persistent Data Server, a service layer on top of NVM (Non-Volatile Memory). PDS is required because of underlying limitations of the NVM. The following are the limitations of the NVM:

1. The NVM takes some amount of time to store or erase the data and it unusually takes a few milliseconds.
2. NVM has an endurance associated with it and one can only store and erase a certain number of times usually few thousands of times before it makes unusable.

A Sensor based Application or Stack expected last long for years and sometimes time-critical, thus waiting for the milliseconds that is needed by the NVM code to execute and maintaining the endurance level of a section in NVM is very critical to any successful product. So, to solve these issues, an abstraction layer is need which will take care of all the limitations and act as an intermediary between the Application or Stack and the NVM. This intermediary is the PDS. It abstracts and manages the NVM so that the Application or Stack can run without waiting on NVM.

The PDS is a component in MLS which manages the storage of any parameter of the Stack or Application to NVM.

PDS Module Sub-layers

This section describes various PDS module sub-layers in detail. The following are the sub-layers inside PDS module.

1. The NVM Sub-layer.
2. The Wear Levelling Sub-layer.
3. The Files and Items Sub-layer.
4. The Task Handler Sub-layer.

The NVM Sub-layer

The NVM abstraction takes care of the following functionalities:

- Abstract the address for the EEPROM emulation area and the Flash storage area into logical address so that it's easy combine both memories.
- Manage the integrity of the information stored.

The flash memory in SAMR34 is organized into Pages and Rows. The following points discusses about how the NVM is organized:

- Each Row had four Pages.
- Data can be written once per Page given a Row. If writing is done more than once per Page in a Row data corruption happens. So, write granularity is Page wise.
- Data erasure can be done for a Row and not for a Page. This means that data stored in all four Pages will be erased. So, erase granularity is Row wise.
- If data needs to be re-written to a Page in a Row, first the Row must be erased and then data to that page can be written. Unfortunately, data stored in the other pages will be lost due to erasure. To prevent this from happening, before issuing an erase, the row must be read to ram and written back after erasure with the new data.

From the above points, PDS module is designed in such a way that a row can be treated as the smallest possible NVM element that can be maintained with least possible code. In SAMR34, the size of the NVM Row is 256bytes.

In the NVM Sub-layer, each row is given a logical row number be it in EEPROM flash section or code flash section. So, this abstraction takes care of the mapping involves with the translation of logical row number to physical address. If more memory is needed it can be added by updating this mapping table.

The integrity of the data stored is done by calculating the 16bit CRC for the data to be stored. This calculated CRC is also stored along with the data in NVM so that while reading back the integrity of the data can be checked.

The Wear Levelling Sub-layer

The Wear Levelling Sub-layer takes care of the following functionalities:

1. Increase the endurance of the NVM.
2. Translation of logical to Physical address.
3. Maintaining information of File ID mapping to Physical address.

As per datasheet an endurance cycle is a write and erase operation. For NVM Flash present in SAMR34 this endurance is around 100K cycles which is less compared to millions of cycles for EEPROM. To emulate the endurance of EEPROM in Flash, instead of writing to the same Physical row in Flash and reducing the endurance, PDS module will write to a new Physical row each time the data is updated. The wear levelling sub-layer provides translation of physical address abstraction and maintain the information.

The WL sub-layer stores data in NVM in the form of Files. Files are defined by Files and Items Sub-layer. Each file is written in a NVM Row and hence, the maximum size of each file shall not exceed 255 bytes.

WL sub-layer will maintain used and free NVM Rows. Based on that information Flash physical address will calculated and data will be given to NVM sub-layer for storing.

On the occurrence of all NVM Rows been used, WL sub-layer will clear older data stored in NVM and free some Rows.

At any given time, WL sub-layer make sure one copy of all data that stored NVM exists.

The Files and Items Sub-layer

The Files and Items Sub-layer takes care of the following functionalities:

- Organizes the storing/retrieving/deleting of MLS parameters.
- Act as external entity from PDS module and provide APIs to Stack & Application to perform PDS operations.

The basic element in Files and Item abstraction is an Item and a File is nothing but a collection of Items. So, if one knows the File id and Item id one can easily store/retrieve/delete an item in PDS. The Item is nothing but a parameter/variable and is private to a layer in the stack and is not exposed to the outside world. If the PDS must store an item it will need the following information about the parameter:

- RAM Address
- Size
- File ID mapping
- Item ID allocated to variable.
- Item's offset inside the File.

The following operation can be performed for every Item in a file:

- Store.
- Delete.
- Restore.

The above information is essential because to store or delete of an Item will take significant amount of time unlike read operation in flash and so it these operations are not done synchronously but instead scheduled by the task handler. More information on the task handler execution is detailed in the next section.

To do these operations, Stack or Application must inform the PDS which operation needs to be performed along with item information. For this purpose, Stack/Application needs to register in PDS module during initialization with an array containing flags for each item per file.

The information is organized in the form of arrays for each File id and it is the duty of each layer to register this array with the following information to the PDS:

- File ID Item Array address.
- Array size.
- File marks array address.
- File marks array size.

The PDS will just simply scan the array registered above to know about operation to perform.

So, from the PDS perspective, it just needs to know which File ids are used and does not need to care what's inside them. It is the responsibility of each layer that needs PDS to create a mapping for each File to an Item and maintain offsets. The PDS just needs to know for which File an operation is pending and not for individual items in a File. So, the PDS maintains something called as the File mask which will track which items need update and this File mask must be set by the individual layers using the File.

For example, if an item needs to be stored into PDS, the layer using the File id needs to post a mask for that File id to inform PDS that an action is pending and it also needs to update the files marks that the layer maintains, now in this case it is "store". Now when the PDS gets scheduled it will check the File mask and it will recognise that an action is pending for a File id. Now it will search the File array data that the layer registered with PDS and will know about the action to be performed i.e. "store", the address of the item in ram, the offset within the file and the size of the file. So, with all this information the PDS will copy the data from ram to the PDS buffer. This buffer may already contain data if the file is already present or an empty buffer if it's the first time an operation is done for the File. After populating the buffer,

it is sent to the WL abstraction layer which adds increments the File counter. Then the buffer is passed on to the NVM abstraction which adds the 16 CRC. The same happens for PDS delete except that the delete bit will be set and no ram copy is performed.

The Task Handler Sub-layer

The Files and Items abstraction takes care of the following design goal:

- Manage the latencies of the NVM.

The way in which PDS has organized the data in the form of Logical rows aides to take care of latencies because to erase and write to a row on average it takes about 10ms. And since we are writing to the flash row after row its manageable time. PDS is also has the lowest priority in MLS. This means that only after all the layers have finished their execution the PDS gets scheduled. The task manager in MLS breaks the context after each File write so that if a task gets posted for another layer, it will schedule that, as it has more priority than PDS while PDS waits for the other task to finish their execution.

PDS Configuration

PDS is enabled in the project files by default. Macro "ENABLE_PDS" is used for enabling and disbaling PDS module.

RWW section of SAMR34 SIP is used for storing PDS data.

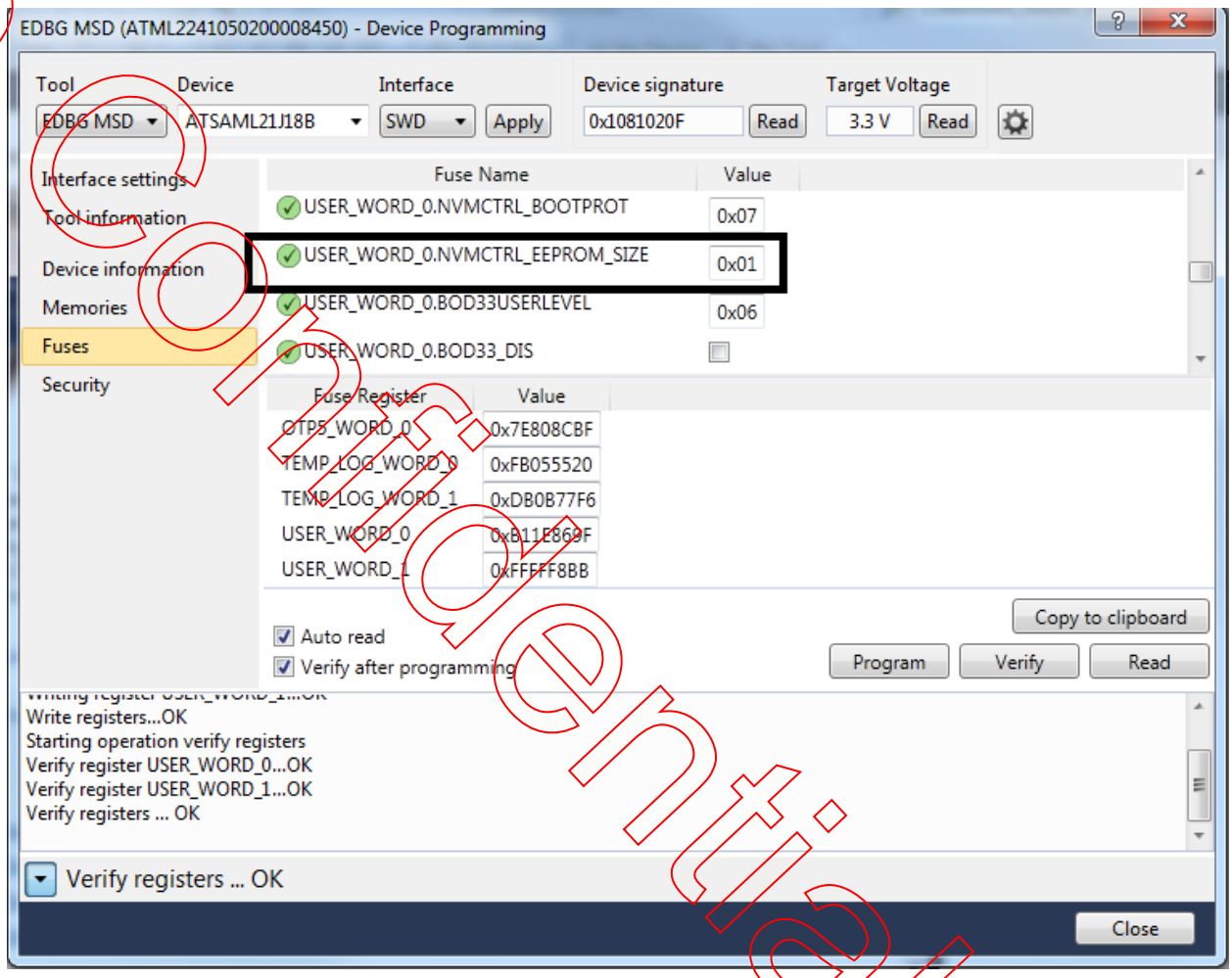
To enable RWW section in SAMR34 following steps to needs to be done.

- Configure EEPROM_SIZE macro in "conf_nvm.h" in config folder.
- By deafult, EEPROM_SIZE is configured as 8192 (8K).
- Based on the requirement of application size of RWW section can be Increased, if needed.
- MLS stack requires 4096(4K) memory for storing PDS data in RWW section.
- SAMR34 RWW section allows 4, 8, 16K memory configuration.
- User must to enable RWW section of SAMR34 using fuse settings from Atmel Studio.
- In the fuse settings, "USER_WORD_0.NVMCTRL_EEPROM_SIZE" setting needs to be updated based on the EEPROM_SIZE macro.

EEPROM_SIZE	RWW Fuse Setting value
4096	0x02
8192	0x01
16384	0x00

Note: RWW section must be enabled for each SAMR34 board. By default, RWW section is disabled in SAMR34. Before flashing FW into the SAMR34, enable RWW section in Tools->Device Programming->Fuses and change the USER_WORD_0.NVMCTRL_EEPROM_SIZE fuse value to one of the above table values based on the EEPROM_SIZE configured in the project.

Figure 9-2. Enabling RWW section in SAMR34



9.1 PDS Module API Definitions

PDS module APIs

Table 9-1. Table 10.1 PDS Initialization API

PDS_Store
`PdsStatus_t PDS_Store(PdsFileItemIdx_t pdsFileItemIdx, uint8_t item)`
Parameters • `pdsFileItemIdx` The file id to regsiter file to PDS. • `item` The item id of the item in PDS.
This function will set the store operation bit in the filemarks for the item in PDS
Returns: PdsStatus_t enumeration • **PDS_OK** Success status • **PDS_INVLIAD_FILE_IDX** Wrong file ID

Table 9-2. Table 10.2 PDS Store API

PDS_Store
`PdsStatus_t PDS_Store(PdsFileItemIdx_t pdsFileItemIdx, uint8_t item)`
Parameters • `pdsFileItemIdx` The file id to regsiter file to PDS. • `item` The item id of the item in PDS.
This function will set the store operation bit in the filemarks for the item in PDS
Returns: PdsStatus_t enumeration • **PDS_OK** Success status • **PDS_INVLIAD_FILE_IDX** Wrong file ID

Table 9-3. Table 10.3 PDS Restore API

PDS_Restore
`PdsStatus_t PDS_Restore(PdsFileItemIdx_t pdsFileItemIdx, uint8_t item)`
Parameters: • `pdsFileItemIdx` The file id to regsiter file to PDS. • `item` The item id of the item in PDS.
This function will restore the item from the PDS to RAM.
Returns: PdsStatus_t enumeration • **PDS_OK** Success status • **PDS_CRC_ERROR** CRC error ocuurs in PDS data • **PDS_ERROR** Error from NVM driver • **PDS_NOT_FOUND** If file or item not found • **PDS_INVLIAD_FILE_IDX** Wrong file ID • **PDS_ITEM_DELETED** Indicateing delete operation performed for an item.

Table 9-4. Table 10.4 PDS Delete API

PDS_Delete
`PdsStatus_t PDS_Delete(PdsFileItemIdx_t pdsFileItemIdx, uint8_t item)`
Parameters: • `pdsFileItemIdx` The file id to regsiter file to PDS. • `item` The item id of the item in PDS.
This function will set the delete operation for the item in the filemask.
Returns: PdsStatus_t enumeration

- **PDS_OK** Success status
- **PDS_CRC_ERROR** CRC error ocuurs in PDS data
- **PDS_ERROR** Error from NVM driver
- **PDS_NOT_FOUND** If file or item not found
- **PDS_INVLIAD_FILE_IDX** Wrong file ID
- **PDS_ITEM_DELETED** Indicateing delete operation performed for an item.

Table 9-5. Table 10.5 API to check valid data is stored in PDS

PDS_IsRestorable
`bool PDS_IsRestorable(void)`
Parameters • `void`
This function checks if all the registered files are restorable.
Returns: Boolean **TRUE** **FALSE**

Table 9-6. Table 10.6 PDS Delete All API

PDS_DeleteAll
`PdsStatus_t PDS_DeleteAll(void)`
Parameters • `void`
This function will erase all the items stored in the PDS
Returns: PdsStatus_t enumeration • **PDS_OK** Success status • **PDS_CRC_ERROR** CRC error ocuurs in PDS data • **PDS_ERROR** Error from NVM driver • **PDS_NOT_FOUND** If file or item not found • **PDS_INVLIAD_FILE_IDX** Wrong file ID • **PDS_ITEM_DELETED** Indicateing delete operation performed for an item.

Table 9-7. Table 10.7 PDS Restore All API

PDS_RestoreAll
`PdsStatus_t PDS_Init(void)`
Parameters PdsStatus_t

PDS_RestoreAll
• `void`
This function will restore all the items from the PDS to RAM from all registered files.
Returns: PdsStatus_t enumeration • **PDS_OK** Success status • **PDS_CRC_ERROR** CRC error ocuurs in PDS data • **PDS_ERROR** Error from NVM driver • **PDS_NOT_FOUND** If file or item not found

Table 9-8. Table 10.8 PDS Store All API

PDS_StoreAll
`PdsStatus_t PDS_StoreAll(void)`
Parameters • `void`
This function will set the store operation to all the items stored in all the registered files in PDS
Returns: PdsStatus_t enumeration • **PDS_OK** Success status

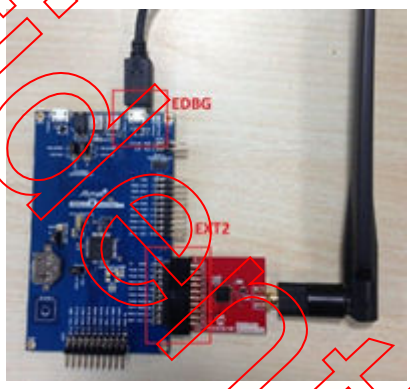
Table 9-9. Table 10.9 PDS Register File

PDS_RegFile
`PdsStatus_t PDS_RegFile(PdsFileItemIdx_t argFileId, PdsFileMarks_t argFileMarks)`
Parameters • `argFileId` The file id to regsiter file to PDS • `argFileMarks` Structure with details of Items Lsit, Number of items, Callback function.
This function will set the store operation to all the items stored in all the registered files in PDS
Returns: PdsStatus_t enumeration • **PDS_OK** Success status

10. Application Setup

1. This application requires two devices: SAML21Xplained Pro with SX1276 wing board, and the MultiTech MultiConnect® Conduit™ Gateway (it has both network and application server functionality).
2. Build the firmware (refer to Building the Firmware) and program it into the SAML21 Xplained Pro.
3. Connect the LoRa transceiver board (SX1276 wing board) in EXT2 of SAML21 Xplained Pro.
4. The board has a software switch(SW0) configured in two modes :
   4.1. Short press - SW0 button pressed and released immediately
   4.2. Long press - SW0 button pressed and holded for more than 3 sec before releasing

Figure 10-1. Connection Setup



10.1 MultiTech MultiConnect® Conduit™ Gateway Setup

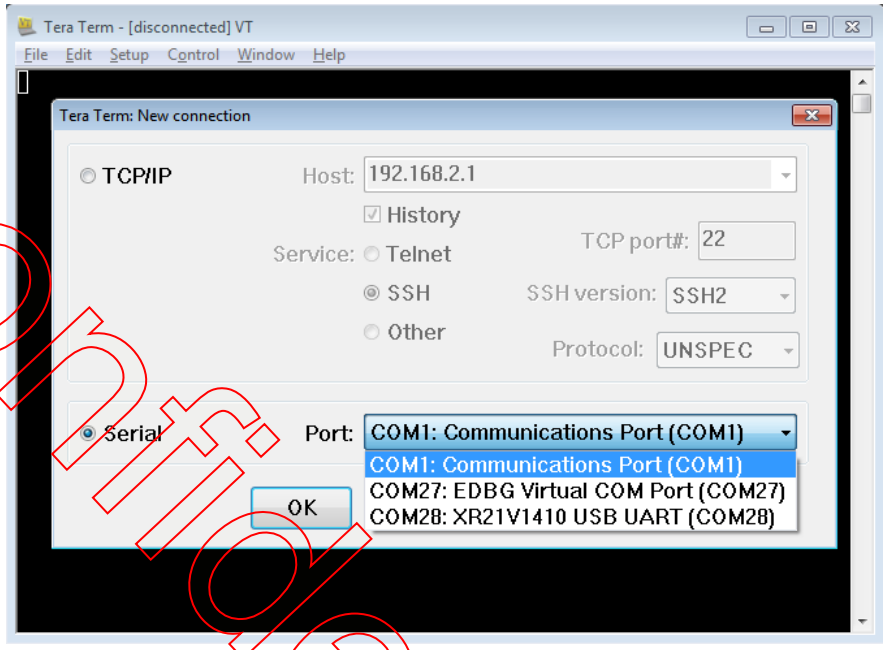
Prerequisites

- SAML21-Xplained Pro board with SX1276 wing board
- MultiTech MultiConnect Conduit model MTCDT-210A

Perform the following steps to setup Multitech MultiConnect Conduit Gateway:

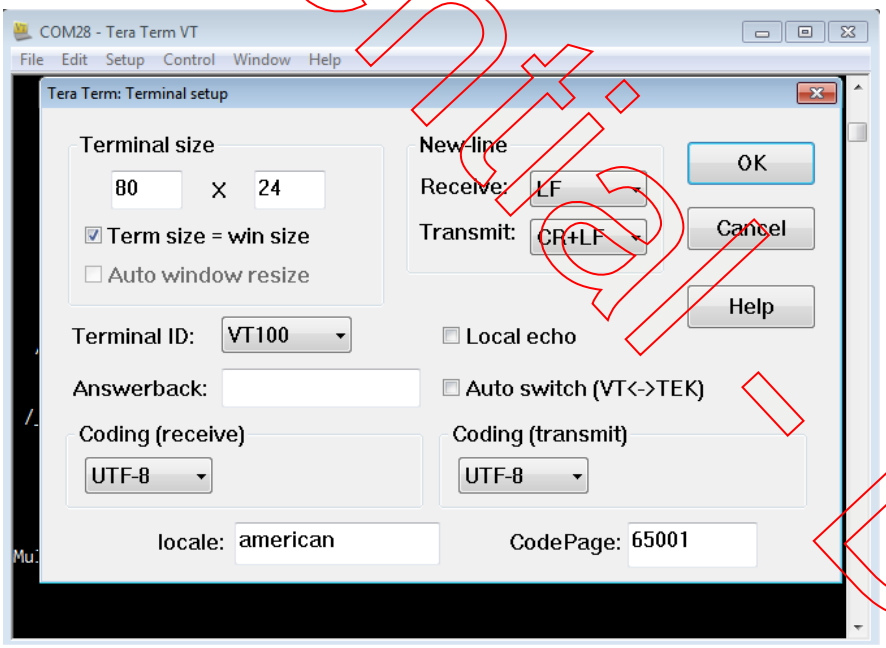
1. Power-up and connect MultiTech MultiConnect Conduit with the PC.

Figure 10-2. Connecting MultiTech Multiconnect Conduit with PC



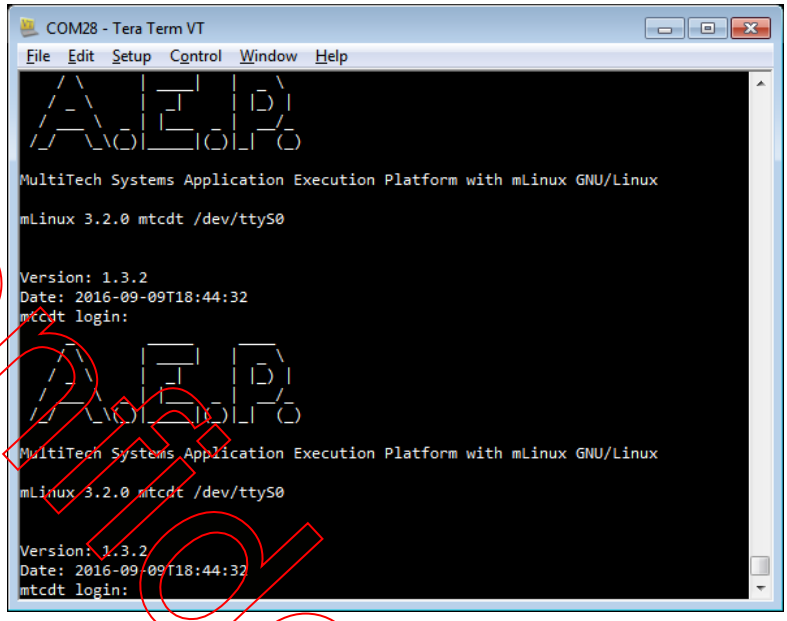
2. The MultiTech MultiConnect Conduit enumerates a virtual COM port (for example: COM178).

Figure 10-3. Enumerating a Virtual COM Port



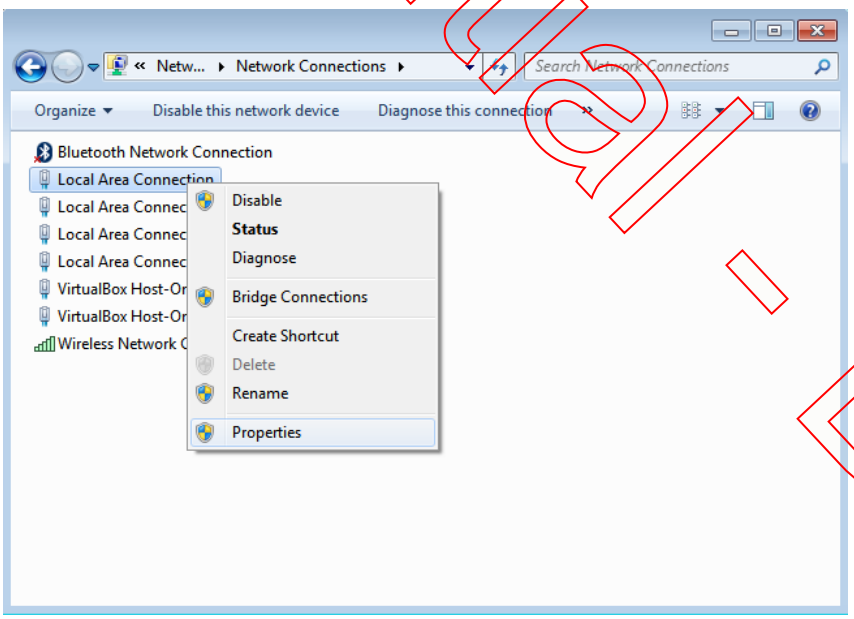
3. Open TeraTerm or any other terminal program. Set the UART as COM178 and Baudrate = 115200.

Figure 10-4. Application Execution Platform



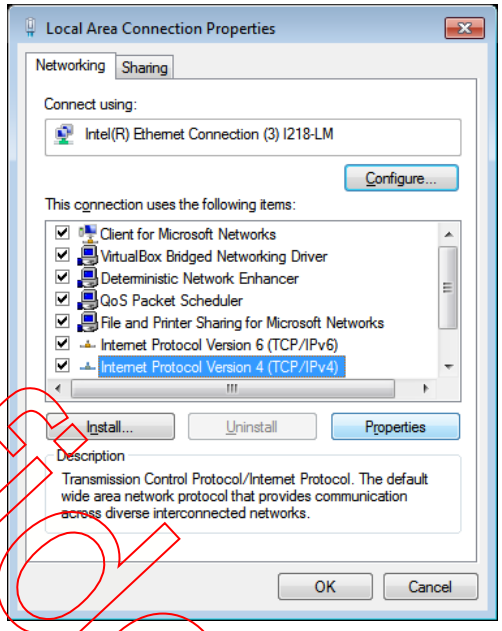
4. Log in to the MultiTech MultiConnect Conduit (default: “username = admin and password = admin”).
5. Open *Control Panel > Network and Internet > Network connections*.
6. Select the Ethernet card in the PC.
7. Right click and select **Properties**.

Figure 10-5. Network Connections



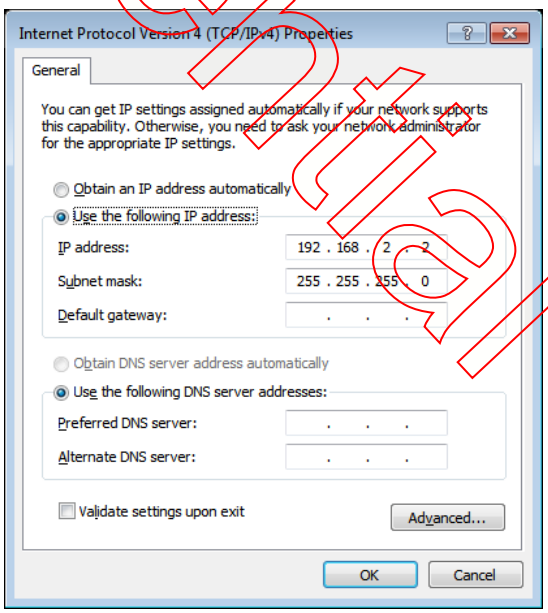
8. Select “IPv4” adapter settings.

Figure 10-6. Local Area Connection Properties



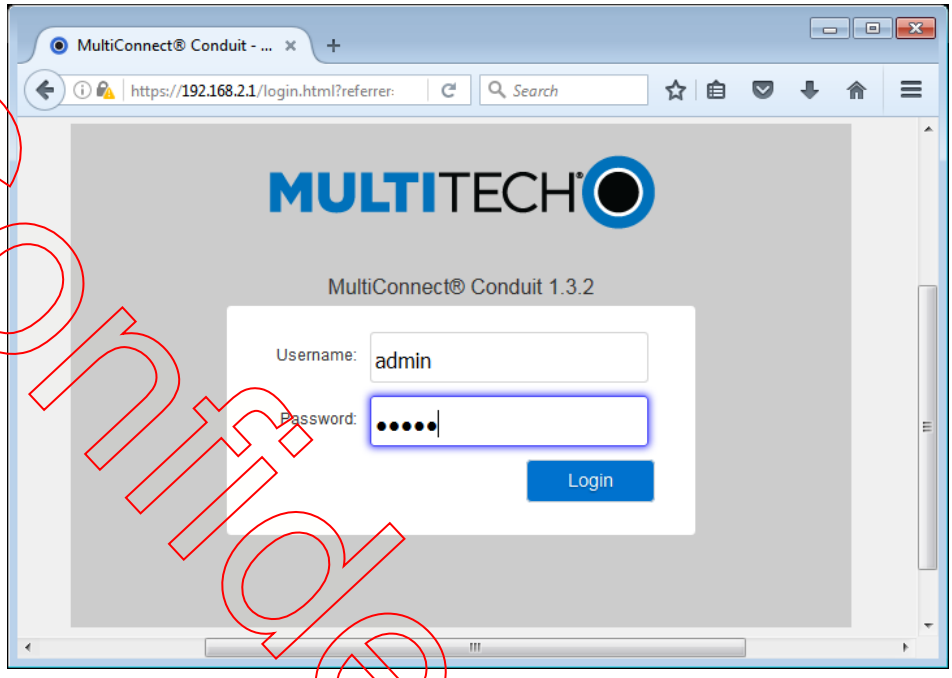
9. Set the IP address to “192.168.2.2” in the IPv4 adapter properties.

Figure 10-7. Setting IP Address



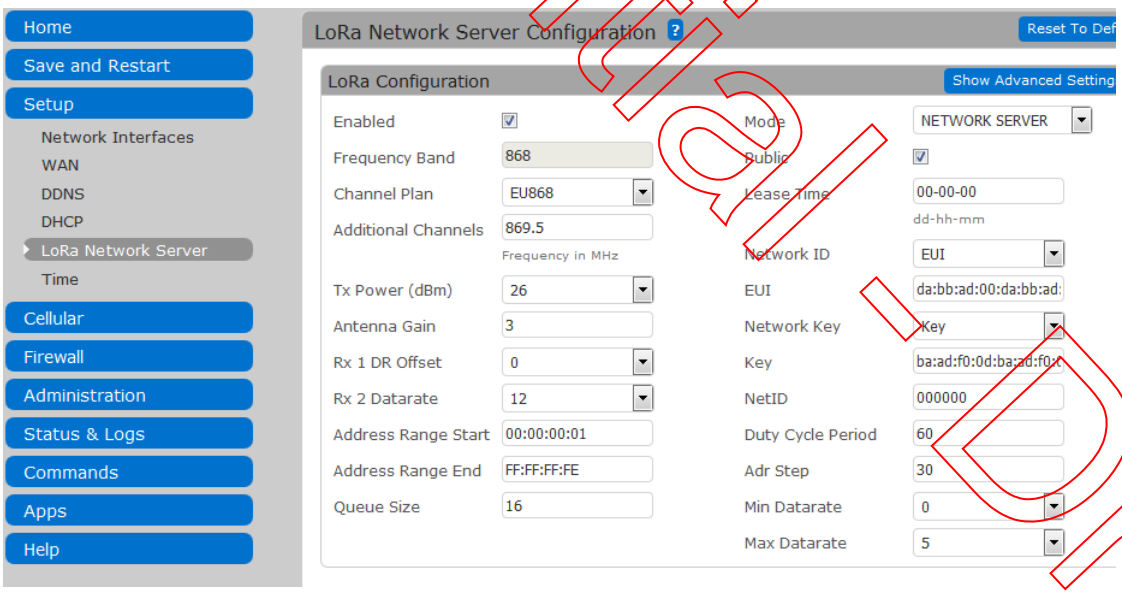
10. Open a web browser and go to https://192.168.2.1. A login page appears. Enter the username and password (default: admin/admin). By default, this is the IP address configured in the Conduit when it is reset.

Figure 10-8. MultiTech MultiConnect Conduit Web Page



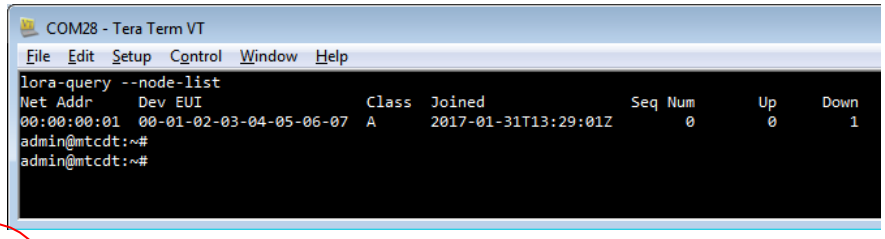
11. Open https://192.168.2.1/lora_network.html page. Configure the LoRa parameters as illustrated in the following figure (it is set for EU). In this page, the gateway and the network server parameters are configured. In the figure, the MultiTech MultiConnect Conduit is configured for the EU region.

Figure 10-9. LoRa Network Server Configuration



12. Information such as end-device address, uplink and downlink status can also be queried from the MultiTech MultiConnect Conduit root console that is opened in the terminal.

Figure 10-10. MultiTech MultiConnect Conduit Console



Now the MultiTech gateway is ready to accept the end-device and communicate with it. For more information on MultiTech MultiConnect Conduit, refer to http://www.multitech.net.

Note: For additional information on configuring MultiTech MultiConnect Conduit MTCDT-210A, refer the following links:

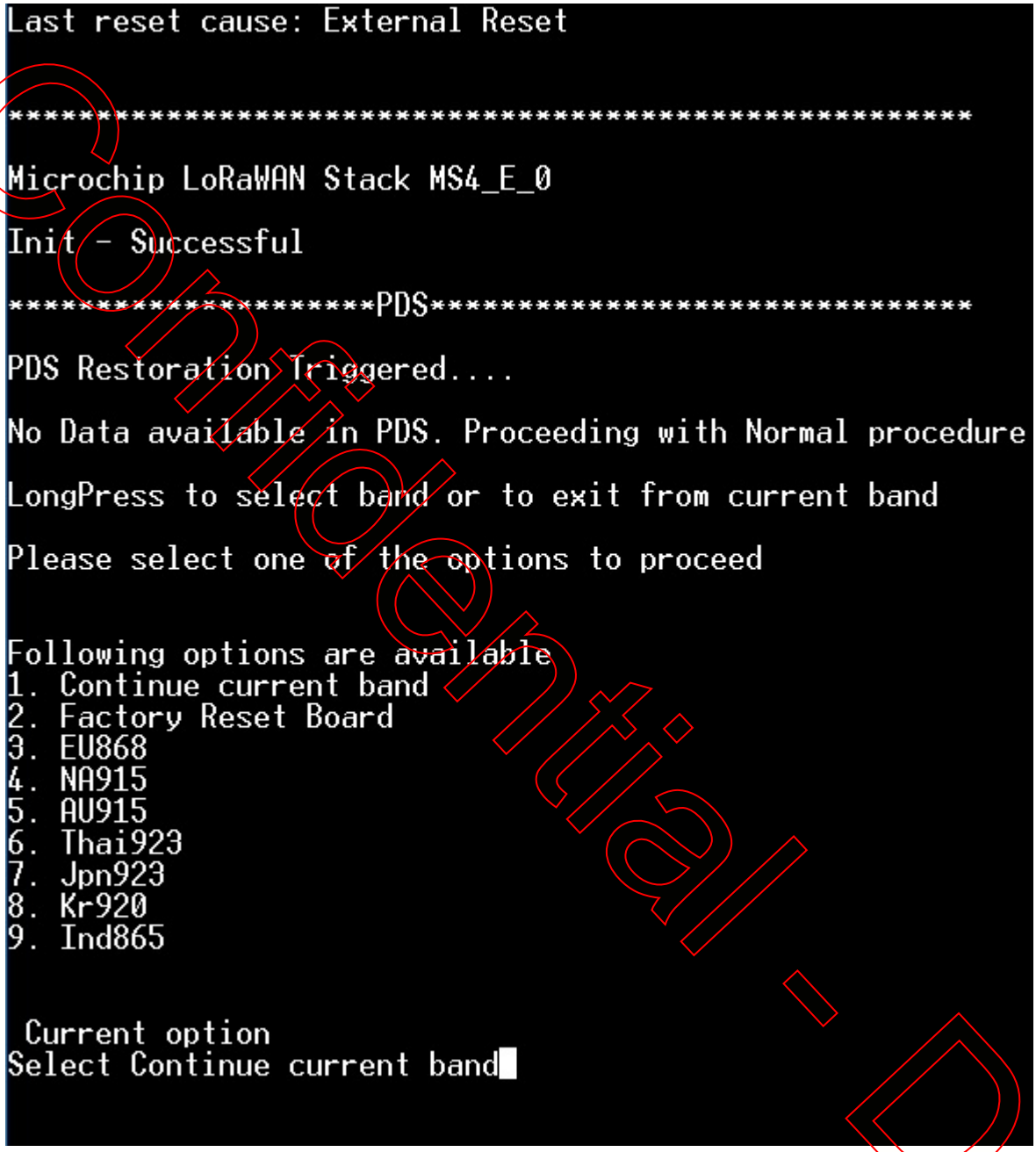
- http://www.multitech.com/models/94557201LF
- http://www.multitech.net/developer/products/conduit/connecting-to-the-debug-interface/
- http://www.multitech.net/developer/software/lora/conduit-aep-lora-communication/
- http://www.multitech.net/developer/software/lora/getting-started-with-lora-conduit-aep/
- http://www.multitech.net/developer/software/lora/

10.2 Application Sequence

The following is the sequence to execute the example application:

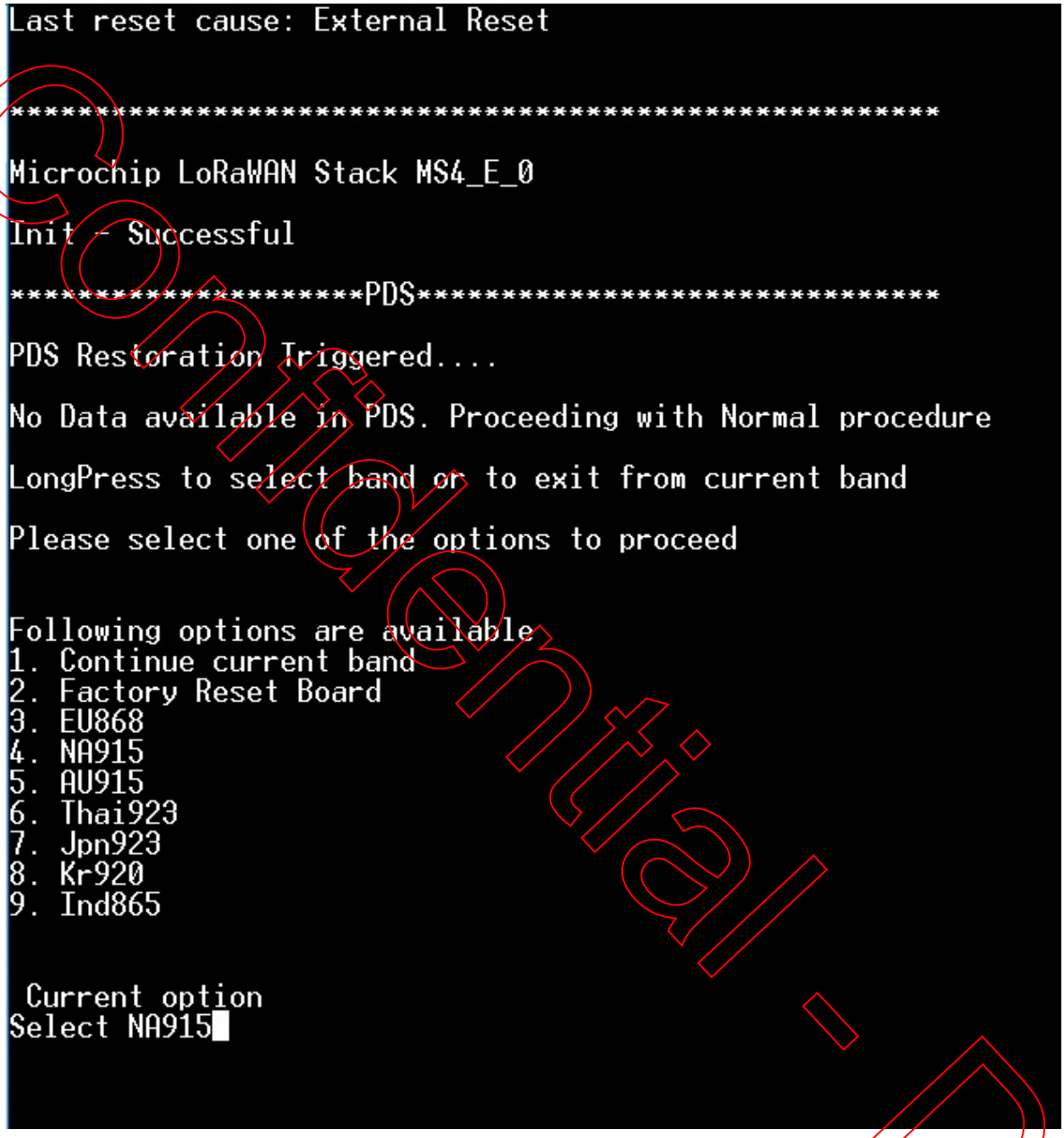
1. Initialization:
   - First, the application initializes the system resources such as clock, board, drivers, AES, system timer and radio hardware interfaces. This is done in `main()` at `main.c`.
   - Then, it initializes the LoRaWAN stack by calling `LORAWAN_Init` from `mote_demo_init()` function at `mote_demo.c`
2. In the terminal, the demo app lists all the supported bands during the startup.

Figure 10-11. Terminal output after flashing



3. In the terminal, under the “Current option” the app displays one of the supported band for every short press.

Figure 10-12. Current band option

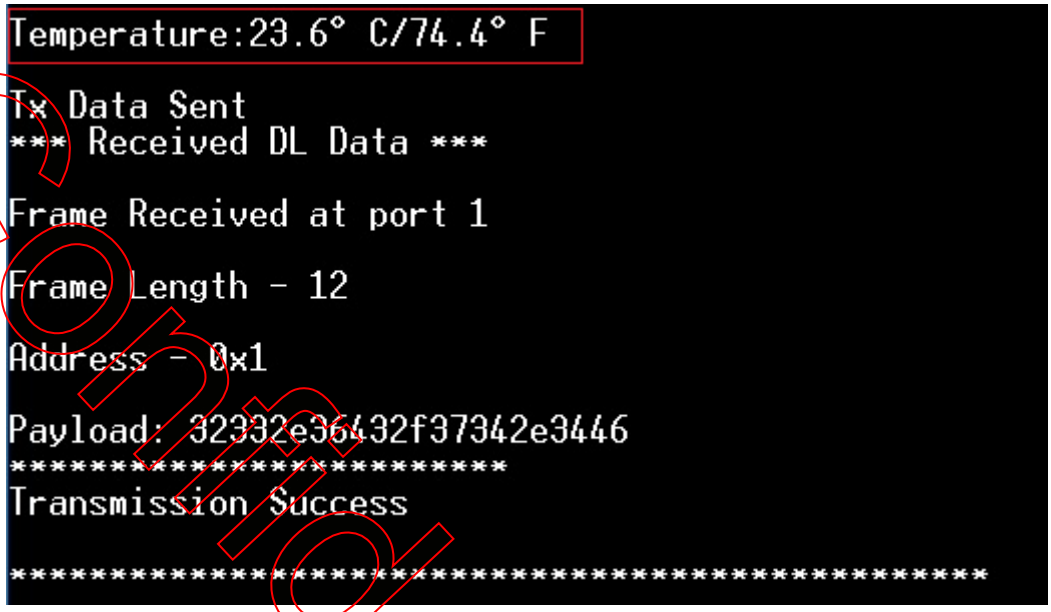


4. Long press of the button SW0, will select the band shown in the “Current option”
5. Once band is selected, application starts the activation process through either ABP or OTAA.
6. Once the activation is completed, it contains DeviceAddress, NwkSKey and AppSKey.

Figure 10-13. Terminal join output

7. At any point after complete activation of the band, Long press of button (SW0), will exit from the band and allows user to choose a different band
8. If join request is failed for any reason, the short press of button SW0, will send the join request to the NS
9. Set the configured device class. If the class chosen in "Class C", downlink multicast reception functionality is configured with the parameters provided in `conf_app.h`
10. On every short button press (SW0) in the joined state, end-device sends temperature data, read from the MCU internal temperature.

Figure 10-14. Data transmit output



11. If the network server downlinks any data, the end-device prints the data that is connected to it in the terminal.

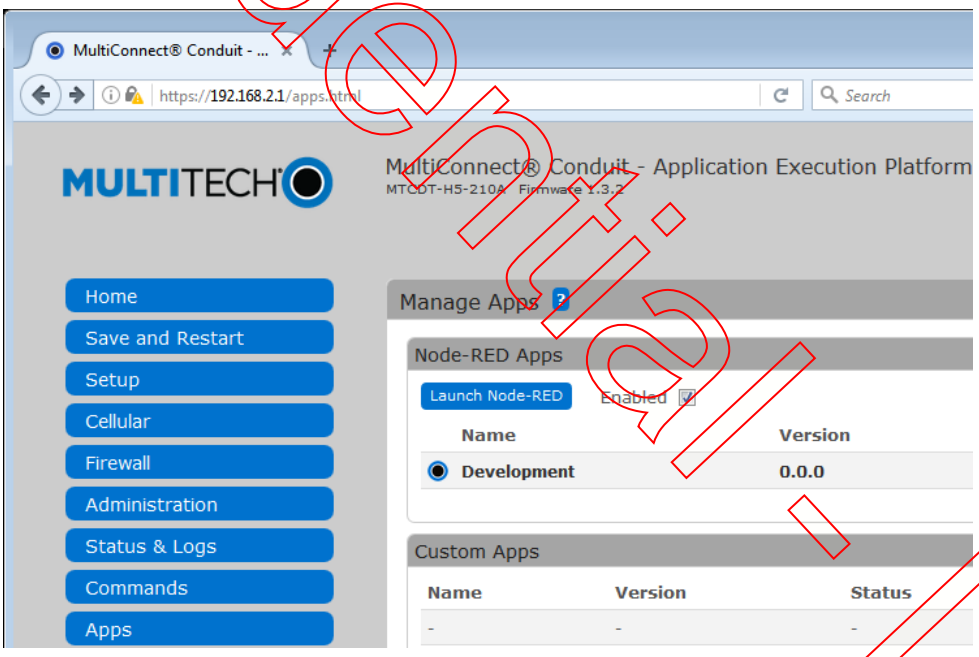
Note: These steps are provided for reference. The user can modify the example application according to their own requirements.

11. Uplink and Downlink

The transmission of data from an end-device to a network server or an application server through a gateway is called “uplink”. An end-device must be activated before sending an uplink. The transmission of data from a network server or an application server through a gateway to an end-device is called “downlink”. An end-device must be activated before it can receive a downlink. In case of the ENDDEVICE_DEMO application, the end-device gets connected to the MultiTech Conduit. The Conduit provides a way to view the uplink data and set the downlink data, which is done through the Node-RED interface. Before the uplink, the Node-RED interface has to be configured in the Conduit. Perform the following steps to configure the Node-RED to dump the uplink data in a debug window and to echo the uplink data as downlink.

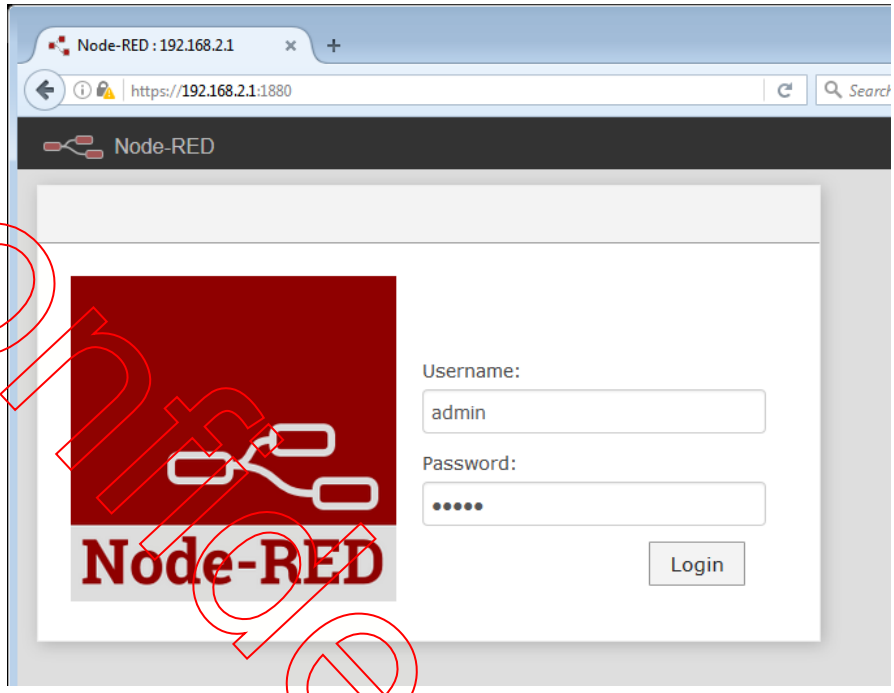
1. Open an internet browser. In the address field, type the default address of the Conduit – http://192.168.2.1. A login page appears.
2. Enter the default username/password (admin/admin).
3. The “MultiTech MultiConnect Conduit – Application Execution Platform” webpage appears.
4. Click **Node-RED** under the **Apps** tab on the left side of the webpage.

Figure 11-1. MultiTech MultiConnect Conduit - Application Execution Platform Webpage



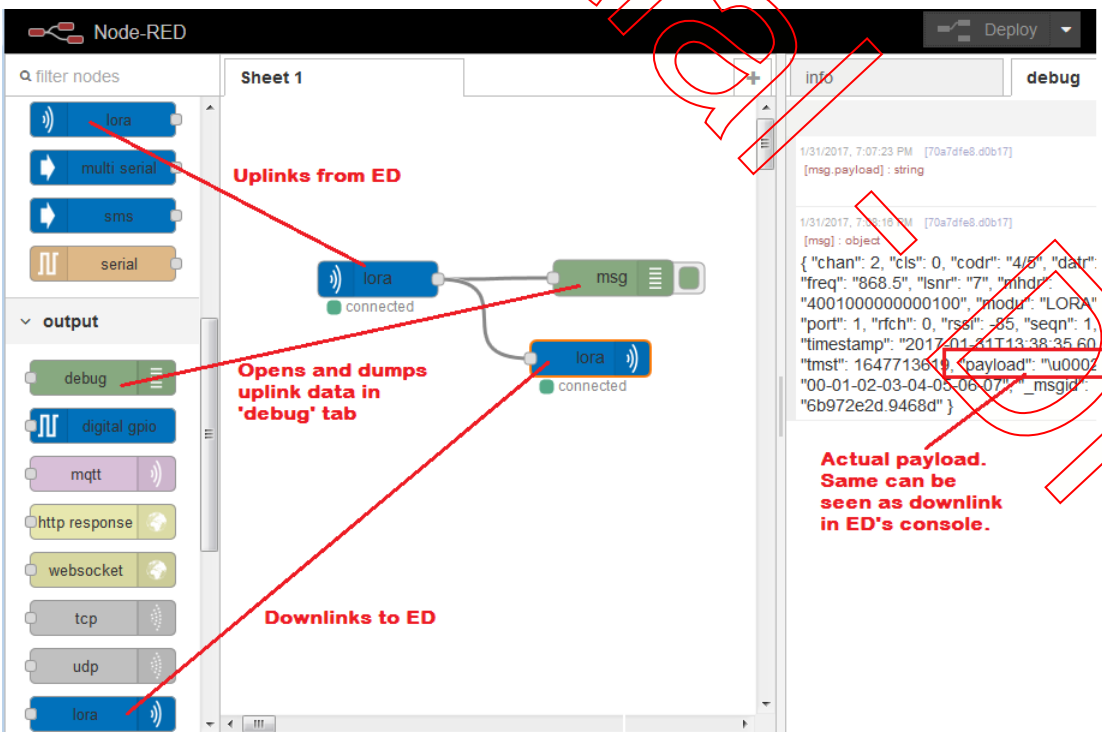
5. The Node-RED login page appears.
6. Enter the default username/password (admin/admin).

Figure 11-2. Node-RED Login Page



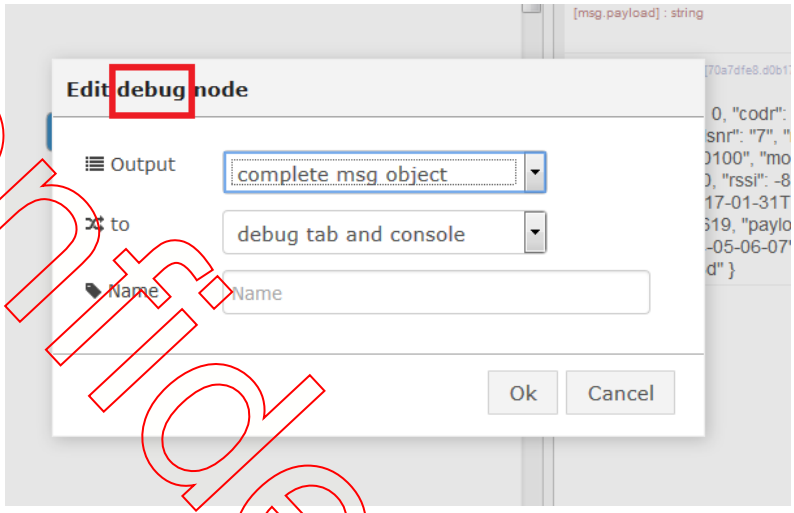
7. The Node-RED configuration page appears. It displays the list of nodes in the left pane and a configuration sheet. The nodes are categorized into “Input” and “Output”. Among them, **lora** node represents a LoRa end-device. When taken from input, it captures the uplink data received at network server and from output, it represents the downlink. The output **lora** node is used to setup the downlink data. The **debug** node creates a “debug” console window that dumps the information.

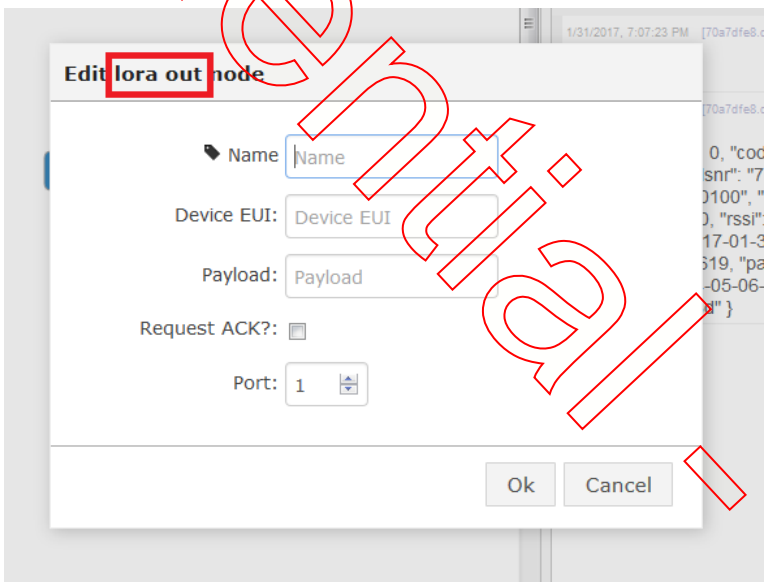
Figure 11-3. Node-RED Configuration



8. Now set up the sheet, “Sheet 1” as illustrated in the following figure. Drag the nodes to “Sheet 1” and link them. When the **debug** node is dragged on to the “Sheet 1”, it displays as `msg.payload`. This configuration makes the Node-RED dump the uplink to the “debug” window and echo the received payload in downlink to the end-device.

Figure 11-4. Editing Debug Node

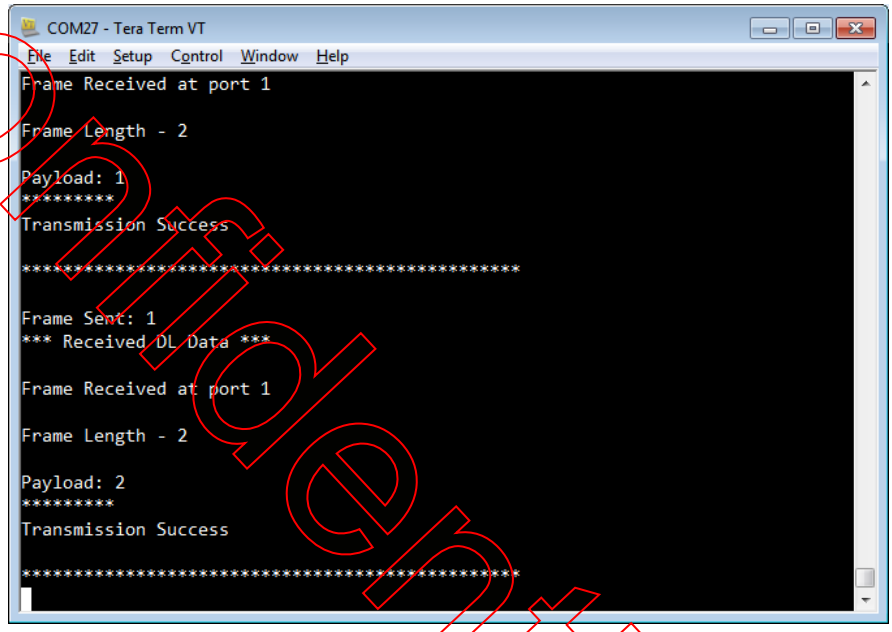




9. Build the EndDevice_Demo firmware by setting APP_ACTIVATION_TYPE to the desired activation method in `conf_app.h`.
10. Connect the reference board to a PC. The reference board is enumerated with a COM port number. For example: COM27.
11. If the build is successful, flash the firmware into the reference board. Follow the steps in Building the Firmware and Flashing the firmware.
12. Open TeraTerm or a similar terminal program. Connect the board's COM port in TeraTerm. Set the UART: COM27and Baudrate = 115200.
13. Depending on the selected activation method, the end-device is activated. Refer to Activating the EndDevice”. If activation is successful, then it can uplink and downlink.

14. Press the button “SW0” in reference board to initiate an uplink. Each time the button “SW0” is pressed, the end-device increments a counter and uplinks the value in payload. For example, `1` for the first press, `2` for the second press and so on.
15. As per the Node-RED configuration, the uplink received at network server is dumped to “debug” window and the same data is echoed in downlink.

Figure 11-5. Console Output



16. In Tera Term, the downlink received by the end-device is displayed. The same data can be seen in Node-RED’s debug window.

12. Glossary

LoRa	Long Range Modulation
LoRaWAN	Long Range Wide Area Network
LPWAN	Low-Power Wide Area Network
SAML21 XPLAINED PRO	Microchip/Atmel SAML21 Xplained Pro
SX1276	Semtech SX1276 LoRa transceiver module
EXT2	Extension 2
UART	Universal Asynchronous Receiver/Transmitter
EDBG	Embedded Debugger
MAC	Medium Access Control
DR	Data Rate
FREQ	Frequency
ASF	Atmel Software Framework
TAL	Transceiver Abstraction Layer
PMM	Power Management Module
PDS	Persistent Data Server
NVM	Non-Volatile Memory

13. Document Revision History

Revision A (April 2017)

This is the initial released version of this document.

Revision B (May 2017)

Added SAMR34 Platform details to the document.

Revision C (June 2017)

Added/Modified some stack configuration parameters.

Revision D (Novemeber 2017)

Added Power management module and PDS.

Revision E (February 2018)

Added Multiband feature details to the document

14. References

- Atmel Studio 7: http://www.microchip.com/development-tools/atmel-studio-7
- Atmel Software Framework: http://asf.atmel.com/docs/latest/
- SAML21 Xplained Pro: http://www.microchip.com/developmenttools/productdetails.aspx?partno=atsaml21-xpro-b
- LoRa Technology Evaluation Suite user's guide: http://ww1.microchip.com/downloads/en/DeviceDoc/40001847A.pdf

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Product Identification System

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

Note:

1. Tape and Reel identifier only appears in the catalog part number description. This identifier is used for ordering purposes and is not printed on the device package. Check with your Microchip Sales Office for package availability with the Tape and Reel option.
2. Small form-factor packaging options may be available. Please check http://www.microchip.com/packaging for small-form factor package availability, or contact your local Sales Office.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BeaconThings, BitCloud, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, KeeLoq logo, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, RightTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, chipKIT, chipKIT logo, CodeGuard, CryptoAuthentication, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PureSilicon, QMatrix, RightTouch logo, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-63277-605-1

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Asia Pacific Office** Suites 3707-14, 37th Floor Tower 6, The Gateway Harbour City, Kowloon **Hong Kong** Tel: 852-2943-5100 Fax: 852-2401-3431 **Australia - Sydney** Tel: 61-2-9868-6733 Fax: 61-2-9868-6755 **China - Beijing** Tel: 86-10-8569-7000 Fax: 86-10-8528-2104 **China - Chengdu** Tel: 86-28-8665-5511 Fax: 86-28-8665-7889 **China - Chongqing** Tel: 86-23-8980-9588 Fax: 86-23-8980-9500 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 Fax: 86-571-8792-8116 **China - Hong Kong SAR** Tel: 852-2943-5100 Fax: 852-2401-3431 **China - Nanjing** Tel: 86-25-8473-2460 Fax: 86-25-8473-2470 **China - Qingdao** Tel: 86-532-8502-7355 Fax: 86-532-8502-7205 **China - Shanghai** Tel: 86-21-3326-8000 Fax: 86-21-3326-8021 **China - Shenyang** Tel: 86-24-2334-2829 Fax: 86-24-2334-2393 **China - Shenzhen** Tel: 86-755-8864-2200 Fax: 86-755-8203-1760 **China - Wuhan** Tel: 86-27-5980-5300 Fax: 86-27-5980-5118 **China - Xian** Tel: 86-29-8833-7252 Fax: 86-29-8833-7256	**China - Xiamen** Tel: 86-592-2388138 Fax: 86-592-2388130 **China - Zhuhai** Tel: 86-756-3210040 Fax: 86-756-3210049 **India - Bangalore** Tel: 91-80-3090-4444 Fax: 91-80-3090-4123 **India - New Delhi** Tel: 91-11-4160-8631 Fax: 91-11-4160-8632 **India - Pune** Tel: 91-20-3019-1500 **Japan - Osaka** Tel: 81-6-6152-7160 Fax: 81-6-6152-9310 **Japan - Tokyo** Tel: 81-3-6880- 3770 Fax: 81-3-6880-3771 **Korea - Daegu** Tel: 82-53-744-4301 Fax: 82-53-744-4302 **Korea - Seoul** Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934 **Malaysia - Kuala Lumpur** Tel: 60-3-6201-9857 Fax: 60-3-6201-9859 **Malaysia - Penang** Tel: 60-4-227-8870 Fax: 60-4-227-4068 **Philippines - Manila** Tel: 63-2-634-9065 Fax: 63-2-634-9069 **Singapore** Tel: 65-6334-8870 Fax: 65-6334-8850 **Taiwan - Hsin Chu** Tel: 886-3-5778-366 Fax: 886-3-5770-955 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 Fax: 886-2-2508-0102 **Thailand - Bangkok** Tel: 66-2-694-1351 Fax: 66-2-694-1350	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **France - Saint Cloud** Tel: 33-1-30-60-70-00 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820