SAM R34

SAM R34 MLS Getting Started Guide

Introduction

The Microchip LoRaWAN™ Stack (MLS) provides a solution for the LoRaWAN end-device that is used for Internet of Things (IoT) applications.

LoRa® is a wireless modulation technique designed to allow low-power end-devices to communicate over long range and at low data rates.

LoRaWAN is a wireless networking protocol which operates over LoRa communication layer and acts as Medium Access Control (MAC) layer.

LoRaWAN specification and its development is overseen by LoRa Alliance™. The specification is meant for secure communication of end-devices and ensures inter-operability within the LoRa network.

Features

- Low-Power LoRaWAN Solution
- SAM R34 SiP Based on Low Power ARM M0+ Core
- Application Integration Ready
- Persistent Data Server (PDS)
- Power Management Module (PMM)
- Dynamic Regional Band Selection Support

Table of Contents

1. Quick Reference Info

1.1 Reference Documentation

Following documents can be used for further details:

- SAM R34/R35 Microchip LoRaWAN™ Stack Software API Reference Manual (DS70005382)
- SAM R34/R35 Low Power LoRa® Sub-GHz SiP Data Sheet (DS70005356)
- SAM R34 Xplained Pro User Guide (DS50002803)
- WLR089 Xplained Pro User Guide (DSxxxxxxxx)

1.2 Acronyms and Abbreviations Used

Table 1-1. Acronyms and Abbreviations Used

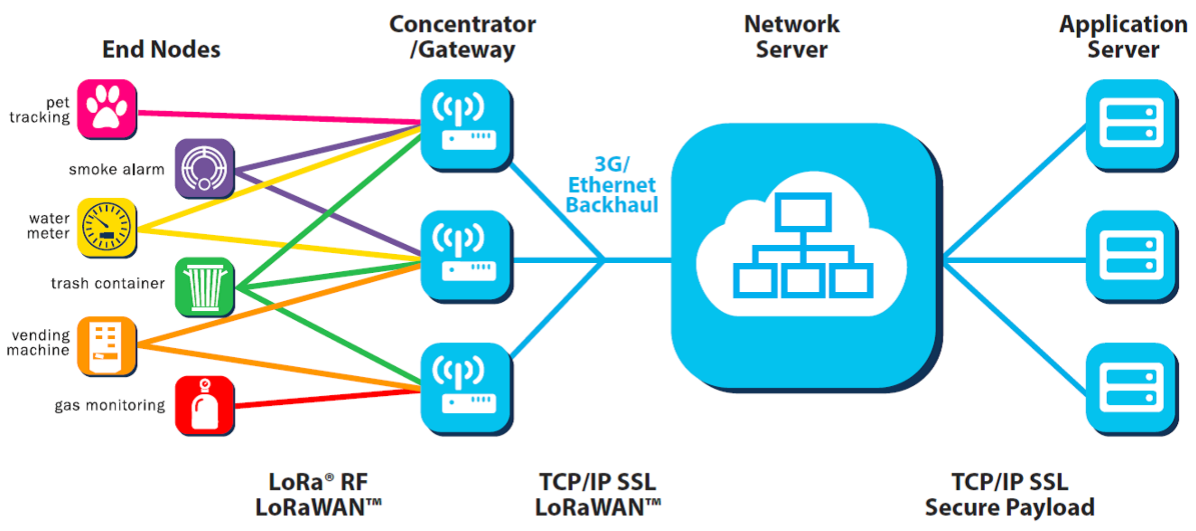
Acronym	Abbreviation
ABP	Activation By Personalization
AES	Advanced Encryption Standard
APPEUI	Application End Unique Identifier
ASF	Advanced Software Framework
DEVEUI	End Device Unique Identifier
EDBG	Embedded Debugger
LoRa	Long Range Modulation
LoRaWAN	Long Range Wide Area Network
MAC	Media Access Controller
MLS	Microchip LoRaWAN Stack
OTAA	Over-The-Air Activation
PDS	Persistent Data Storage
PMM	Power Management Module
TAL	Transceiver Abstraction Layer
UART	Universal Asynchronous Receiver/Transmitter

2. Network Architecture

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 wireless connection between an end-device and the gateway is set up through a LoRa wireless link. The gateways, network server and application servers communicate over an IP back haul linked using Ethernet, 3G, LTE, and so on. The following figure shows the typical system architecture of a LoRa network.

Figure 2-1. LoRaWAN Architecture



2.1 End-Device Architecture

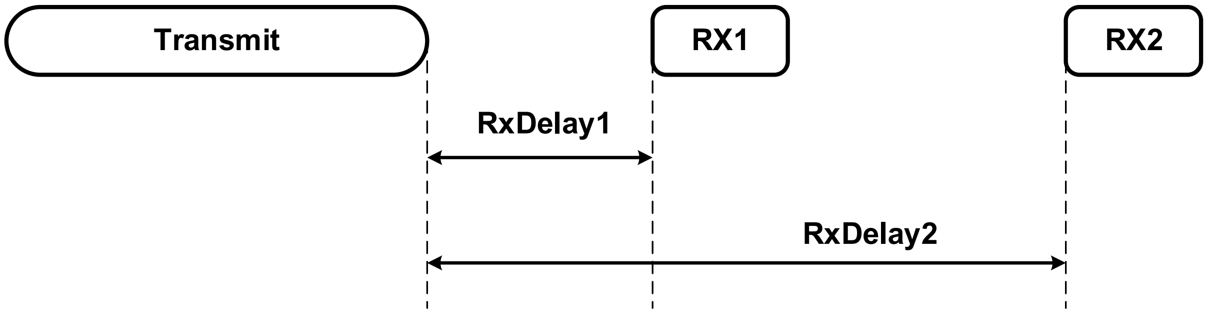
The LoRaWAN specifications define three different classes of end-devices, based on the latency, when the end-device listens to the network server.

2.2 LoRaWAN Device Types

2.2.1 Battery Powered - Class A (All Devices)

Every transaction in a Class A end-device starts with an uplink transmission, which is then followed by two downlink receive windows. The network server sends the downlink message after receiving the uplink. At the end of downlink message, the end-device enters into Sleep mode, thereby saving power. Therefore, Class A devices consume the least power and provide a long battery life. All LoRaWAN end-devices support Class A by default. The following figure shows the data transmission and reception sequence for a typical Class A end-device.

Figure 2-2. Class A Tx/Rx Sequence

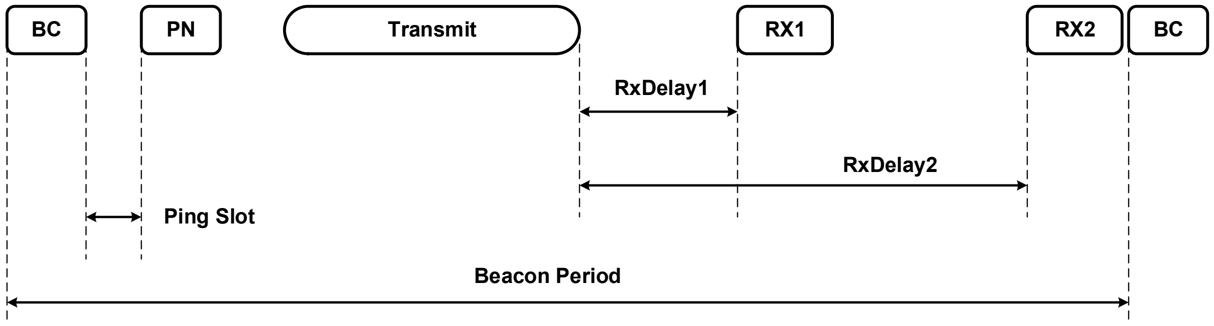


2.2.2 Low Latency - Class B (Beacon)

In Class A, the downlink is non-deterministic since it depends on random uplinks from a sleeping end-device. In Class B the end-device reduces the downlink latency by opening periodic downlink receive windows. The periodicity of the downlink windows is maintained by synchronizing the clocks of the end-device and the network server. For the synchronization, the network server commands the gateways to send a beacon at regular intervals. During uplink, the Class B end-device behaves similar to that of a Class A end-device.

A Class B end-device manages to reduce power consumption and yet reduces the downlink latency. The following figure shows the data transmission and reception sequence for a typical Class B end-device.

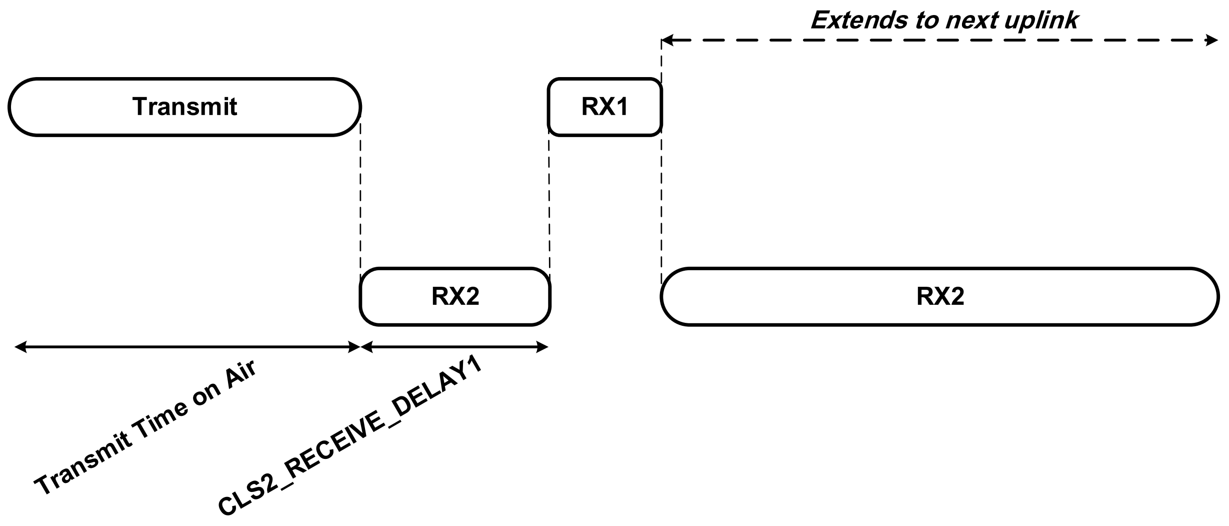
Figure 2-3. Class B Tx/Rx Sequence



2.2.3 No Latency - Class C (Continuous)

Except for the uplink period, the end-device in Class C continuously opens the receive windows, which reduces latency, but increases its power consumption considerably. The following figure shows the data transmission and reception sequence for a typical Class C end-device.

Figure 2-4. Class C Tx/Rx Sequence

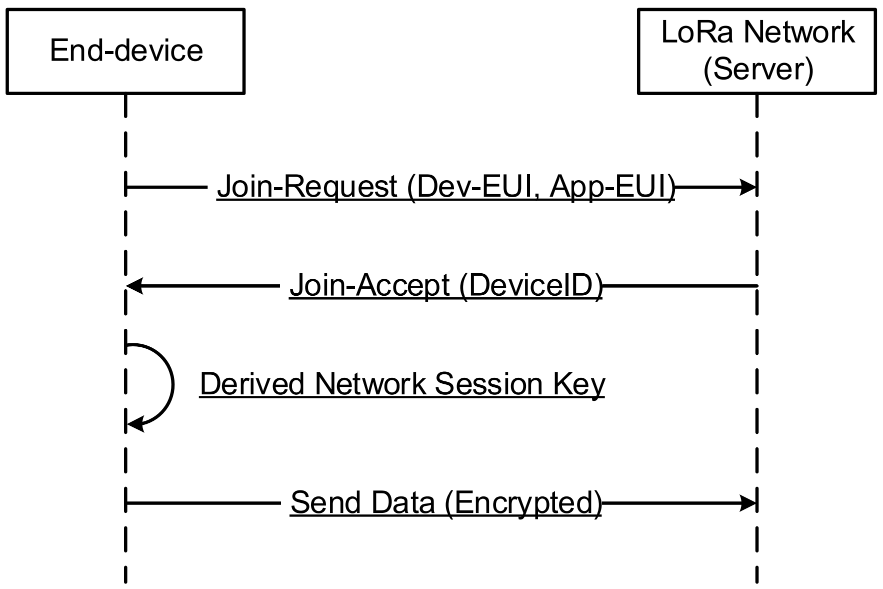


2.3 End-device Activation (Joining)

2.3.1 Over-The-Air Activation

The Over-The-Air Activation (OTAA) is an on demand joining procedure for an end-device to join the LoRa network. The end-device initiates the joining procedure by sending the Dev-EUI and App-EUI to the network server; the network server returns the join accept signal along with the device ID (DevID). The end-device then derives the Network Session Key (NwkSKey) for MAC commands encryption and also derives Application Session Key (AppSKey) for application data encryption. The following figure shows the OTAA joining sequence.

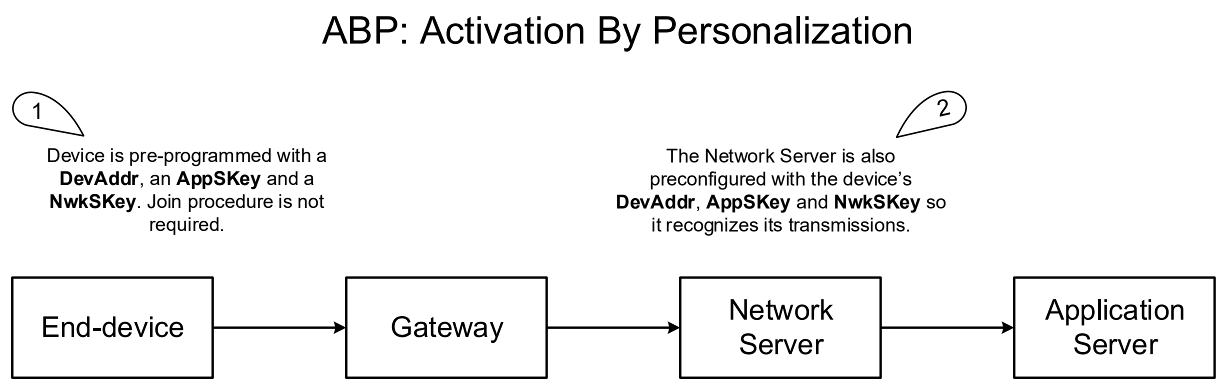
Figure 2-5. OTAA Joining Procedure



2.3.2 Activation By Personalization (ABP)

With the Activation By Personalization (ABP) joining procedure, the service provider preconfigures the Network Session Key (NwkSKey) and Application Session Key (AppSKey). These are stored inside the end-device. The end-device uses this preconfigured data to directly join the LoRa network. The following figure shows the ABP joining process.

Figure 2-6. ABP Joining Procedure

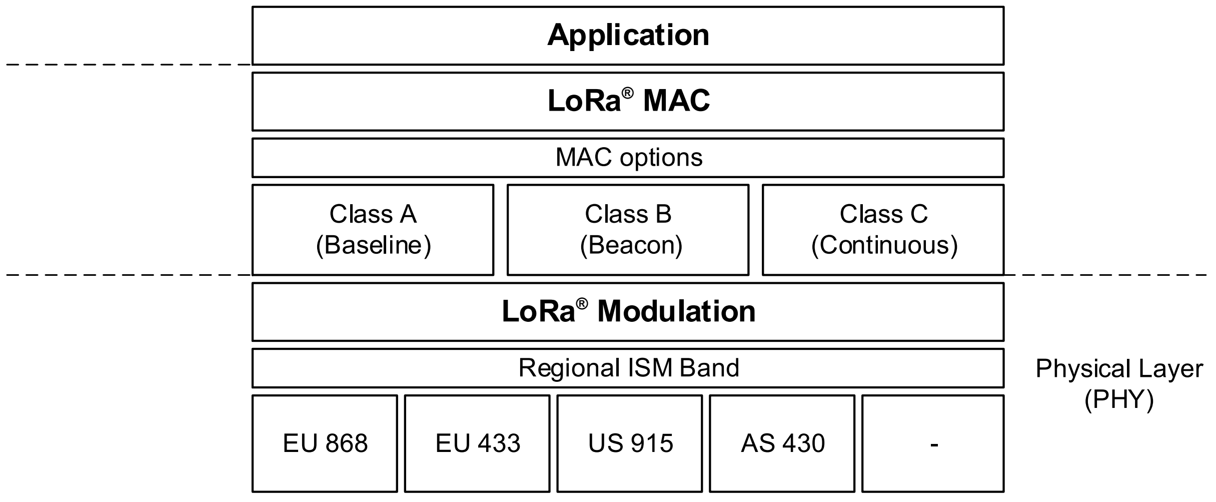

ABP: Activation By Personalization
1
Device is pre-programmed with a **DevAddr**, an **AppSKey** and a **NwkSKey**. Join procedure is not required.
2
The Network Server is also preconfigured with the device's **DevAddr**, **AppSKey** and **NwkSKey** so it recognizes its transmissions.
End-device
Gateway
Network Server
Application Server

2.4 LoRaWAN Layers

The LoRaWAN architecture is defined in terms of blocks known as "layers". Each layer is responsible for realizing a portion of the standard and offers services to the next higher layers.

An end-device contains at least one Physical Layer (PHY), which embeds the radio frequency transceiver. A MAC layer provides access to the physical channel. The application layer provides access to the MAC layer that is used to send and receive the data. The following figure shows the stack architecture of the LoRa end-device.

Figure 2-7. LoRaWAN Layers



3. Package Overview

The Microchip LoRaWAN Stack contains:

- An Atmel Studio 7.0 project, which provides a reference application
- A set of LoRaWAN stack components in a static library (`libLORAWAN_LIBGEN.a`)
- Drivers, software timer, PDS, PMM and radio drivers for the LoRaWAN stack
- The facility to support dynamic regional band switching within the supported bands

3.1 LoRaWAN Stack Directory Structure

The following table provides the directory structure of the LoRaWAN stack code base (`src/ASF/thirdparty/wireless/lorawan`).

Table 3-1. Directory Structure of LoRaWAN Stack

Directory	Description
`/hal`	Contains the implementation for the radio hardware interface, timers, etc.
`/inc`	Contains commonly included file(s)
`/mac`	Contains the headers of the LoRaWAN MAC layer specification independent of regional parameters
`/regparams`	Contains the implementation of the MAC layer functionality specific to the regional bands
`/services`	Contains modules such as software timer, PDS and AES
`/sys`	Contains system modules such as task manager, power management and initialization
`/tal`	Contains transceiver-related headers, drivers for supported transceivers
`/pmm`	Contains the Power Management Module (PMM)
`/libgen`	Contains the static library for the LoRaWAN MAC and TAL

The following table lists the supported hardware platforms and IDE.

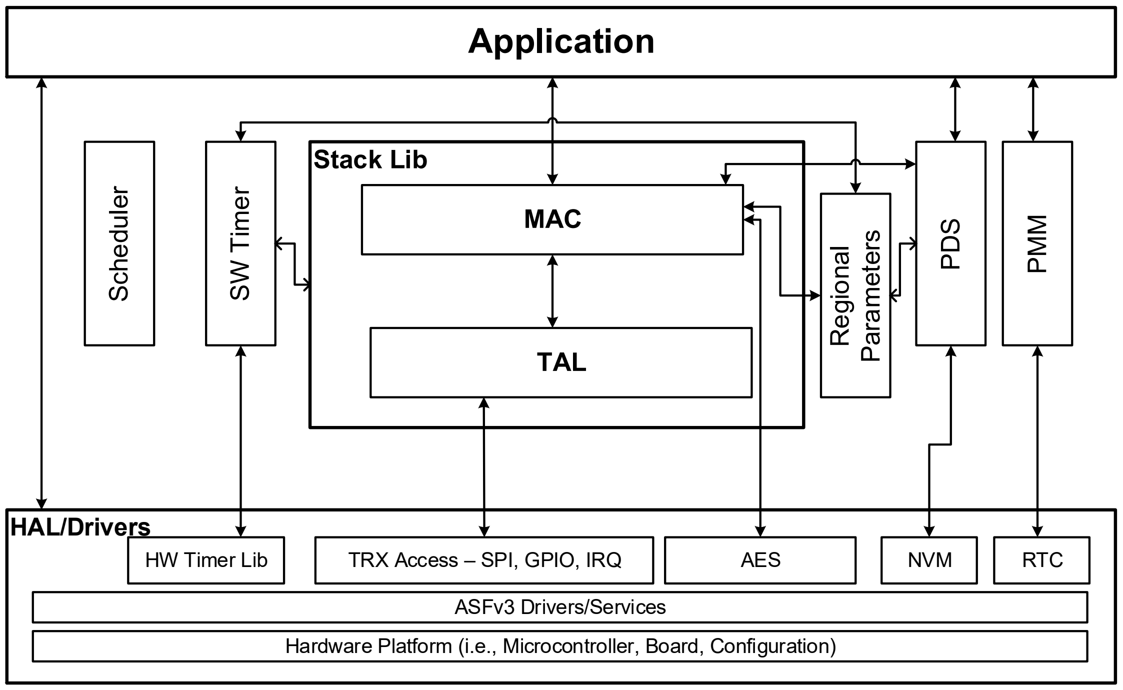
Table 3-2. Supported Hardware Platforms and IDE

Platform	MCU	Transceiver	Evaluation Kits	Supported IDE
SAM R34	ATSAMR34J18B	Semtech SX1276	SAM R34 Xplained Pro	Atmel Studio 7.0
WLR089	ATSAMR34J18B	Semtech SX1276	WLR089 Xplained Pro	Atmel Studio 7.0

4. Architecture

The following figure shows the architecture of the MLS LoRaWAN stack and application.

Figure 4-1. Architecture of MLS Stack



1. The MAC Layer provides the functionality of operations defined in the LoRaWAN Specification.
2. The TAL layer uses the radio drivers and provides access to the transceiver.
3. The radio drivers use the SPI, GPIO and IRQ to communicate with the Semtech Radio Transceiver.
4. The MLS stack supports multiple regional bands. Provision is provided to enable or disable the supported bands in the stack. The regional parameters are provided outside the library to optimize the RAM and the Flash memory based on the requirement.
5. The PDS stores the LoRaWAN parameters in the Flash. This feature is mainly used to restore the data between the power cycles.
6. The PMM helps to reduce the power consumption by putting the processor into Sleep mode when the stack is in Idle mode.
7. The system has a non-preemptive priority based Scheduler, which does scheduling for the MAC, TAL, PMM, PDC, Timer and application sub systems.
8. The ASFv3 provide drivers or services for the interfaces such as, I2C, SPI, GPIO, and UART.
9. The hardware timer library is a library package that is used to produce a 1 µs (1 MHz frequency) tick.
10. The software timer services all the timer requirements for the stack, using Hardware timer TC0.

5. Example Demo Project

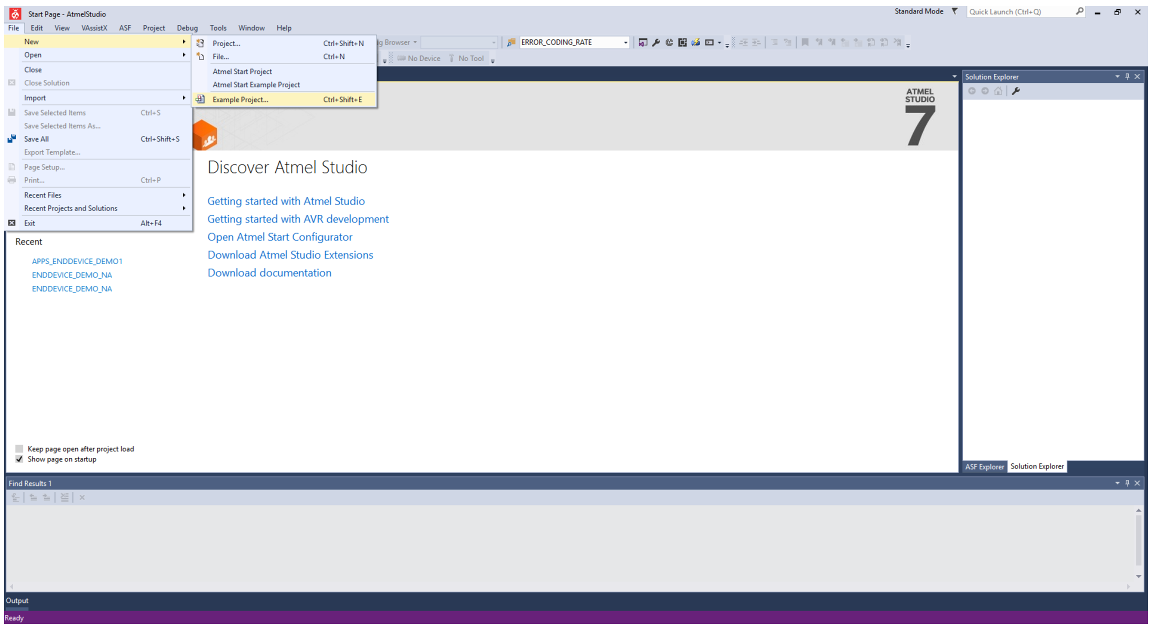
The ASFv3 installer for the Microchip LoRaWAN stack is an extension to Atmel Studio which provides a solution for the LoRaWAN end-device in SAM R34 devices. This extension allows a user to plug and play the SAM R34 drivers or sensor modules from ASF into the Microchip LoRaWAN stack and create easily demonstrable solutions.

5.1 Building the Firmware

Perform the following steps in Atmel Studio 7 to build the firmware for EndDevice_Demo.

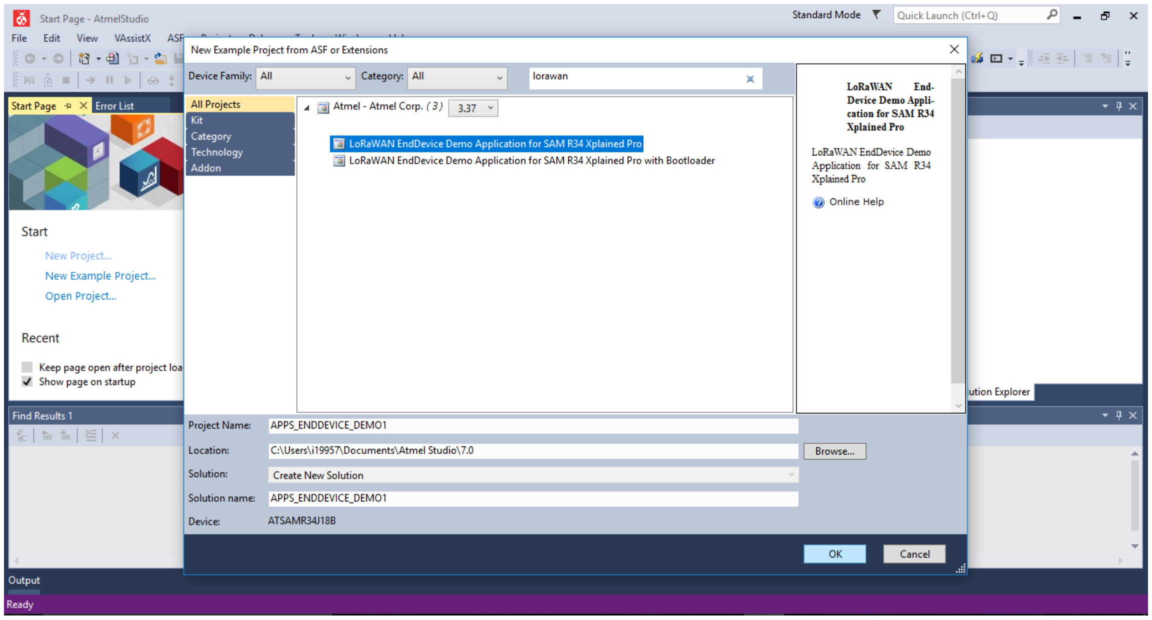
1. Open Atmel Studio and select `File > New > Example Project`.

   Figure 5-1. Opening an Example Project in Atmel Studio



2. In the New Example Project from ASF or the Extensions window:
   2.1. Enter "lorawan" keyword in the search box, which lists all the LoRaWAN EndDevice Demo Applications for SAM R34 Xplained Pro board or WLR089 Xplained Pro board.
   2.2. Select the respective example application of the SAM R34 by expanding the "Atmel - Atmel Corp." in the **All Projects** tab. This selection automatically populates the Project Name, Location, Solution, Solution Name and Device.
   2.3. Click **OK**.

Figure 5-2. Searching for LoRaWAN Example Project



New Example Project from ASF or Extensions

Device Family: All
Category: All
lorawan

Atmel - Atmel Corp. (3) 3.37

- LoRaWAN EndDevice Demo Application for SAM R34 Xplained Pro
- LoRaWAN EndDevice Demo Application for SAM R34 Xplained Pro with Bootloader

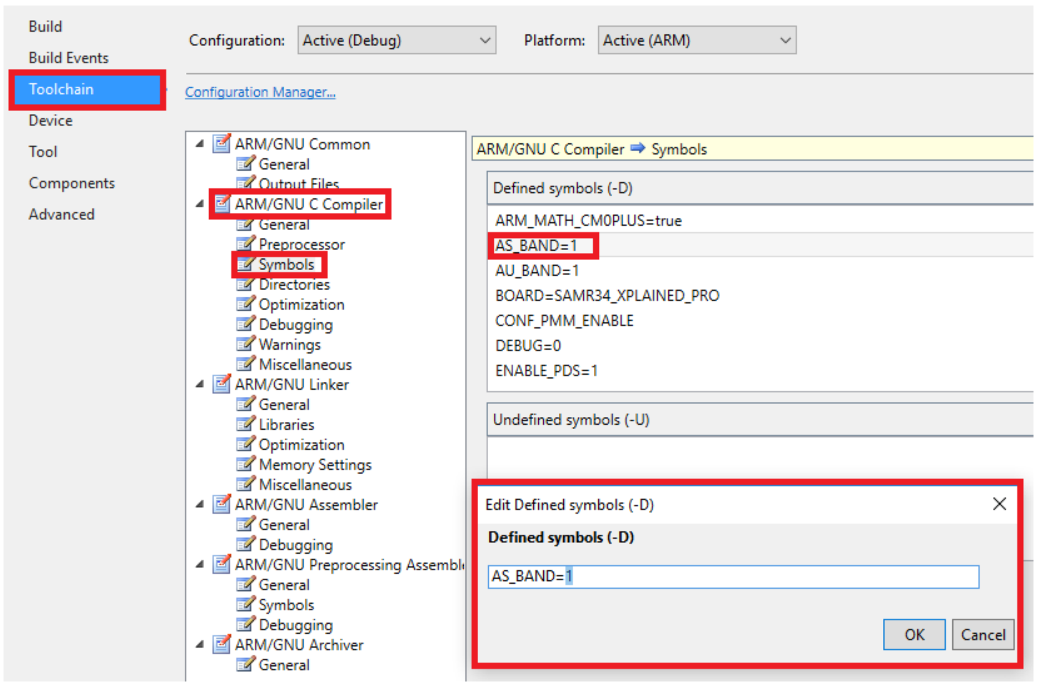
LoRaWAN End-Device Demo Application for SAM R34 Xplained Pro

LoRaWAN EndDevice Demo Application for SAM R34 Xplained Pro

Project Name:	APPS_ENDDEVICE_DEMO1
Location:	C:\Users\i19957\Documents\Atmel Studio\7.0
Solution:	Create New Solution
Solution name:	APPS_ENDDEVICE_DEMO1
Device:	ATSAMR34J18B

3. Select the “Accept the License Agreement” check box, then click **Finish**.
4. The Atmel Studio generates the project files for the selected application example that can be used in the SAM R34 Xplained Pro board or WLR089 Xplained Pro board.
5. MLS supports multiple regional bands; all the regions are enabled in the project by default. To disable certain regions, perform the following steps:
   5.1. Go to `Project > Properties` or press <Alt + F7>.
   5.2. From the left-hand pane, select **Toolchain**.
   5.3. In the right-hand pane, go to `ARM/GNU C Compiler > Symbols`.
   5.4. The regional band macros are listed in the “Defined Symbols” pane.
   - AS_BAND = 1
   - AU_BAND = 1
   - EU_BAND = 1
   - IND_BAND = 1
   - JPN_BAND = 1
   - KR_BAND = 1
   - NA_BAND = 1

Figure 5-3. Modifying Regional Configuration/Band



Defined symbols (-D)

```
ARM_MATH_CM0PLUS=true
AS_BAND=1
AU_BAND=1
BOARD=SAMR34_XPLAINED_PRO
CONF_PMM_ENABLE
DEBUG=0
ENABLE_PDS=1
```

Defined symbols (-D)

```
AS_BAND=1
```

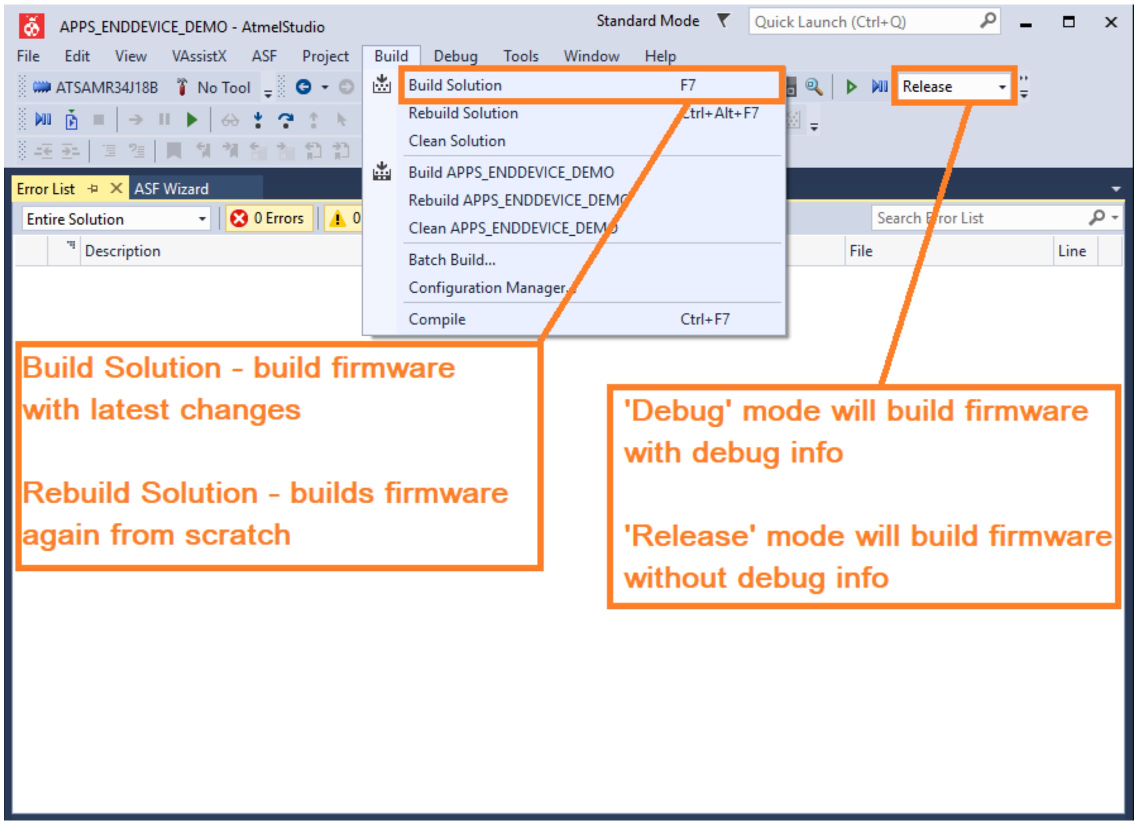
5.5. Runtime support for the regional bands are enabled when the macro for the corresponding regional band is set to `1`, and is disabled when the macro is set to `0`. For example, the following are the macro values to enable only the North American region (NA_BAND).

- AS_BAND = 0
- AU_BAND = 0
- EU_BAND = 0
- IND_BAND = 0
- JPN_BAND = 0
- KR_BAND = 0
- NA_BAND = 1

Note: The above mentioned select bands are from specific releases, provided as an example. Ensure the required band is supported before defining any band. The release notes contain the list of supported regional bands.

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

Figure 5-4. Building the Firmware



Build Solution - build firmware with latest changes

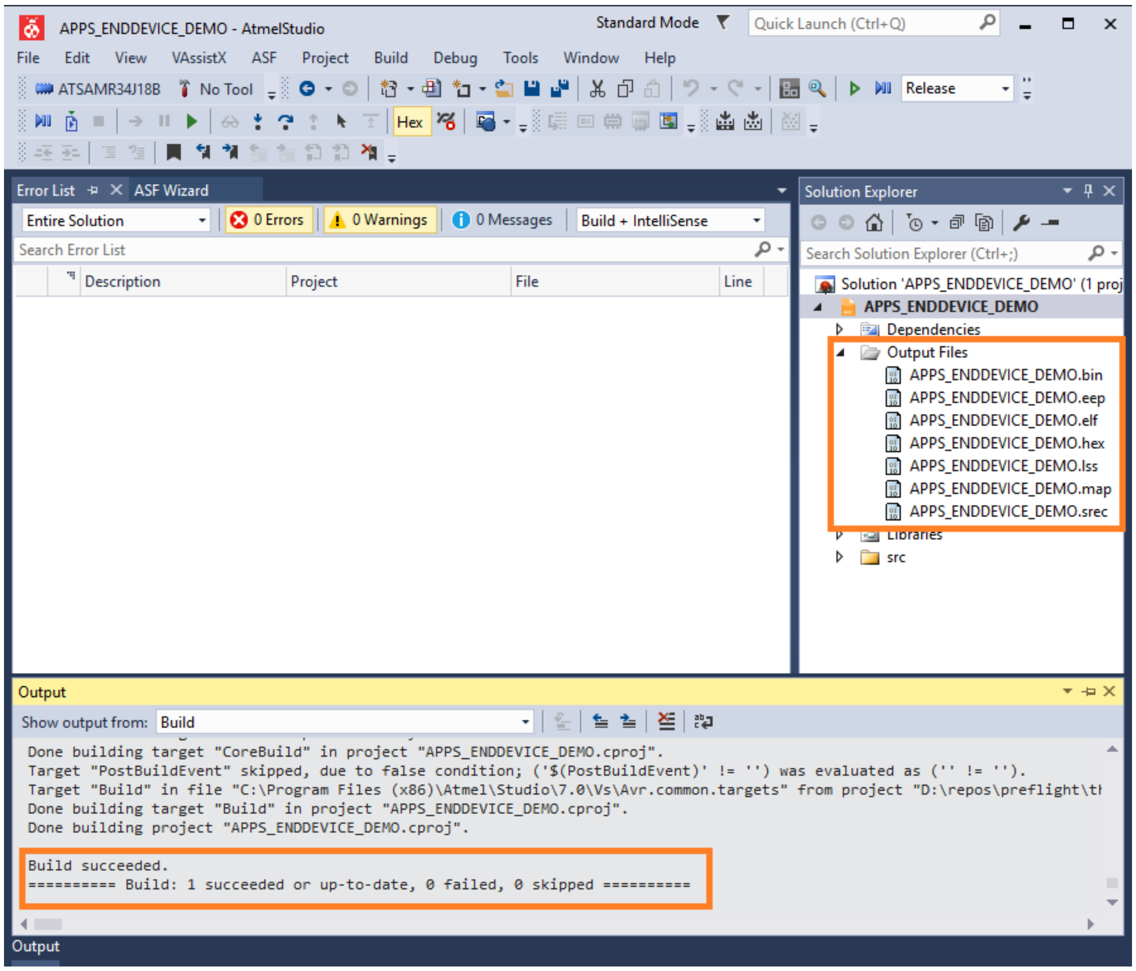
Rebuild Solution - builds firmware again from scratch

'Debug' mode will build firmware with debug info

'Release' mode will build firmware without debug info

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

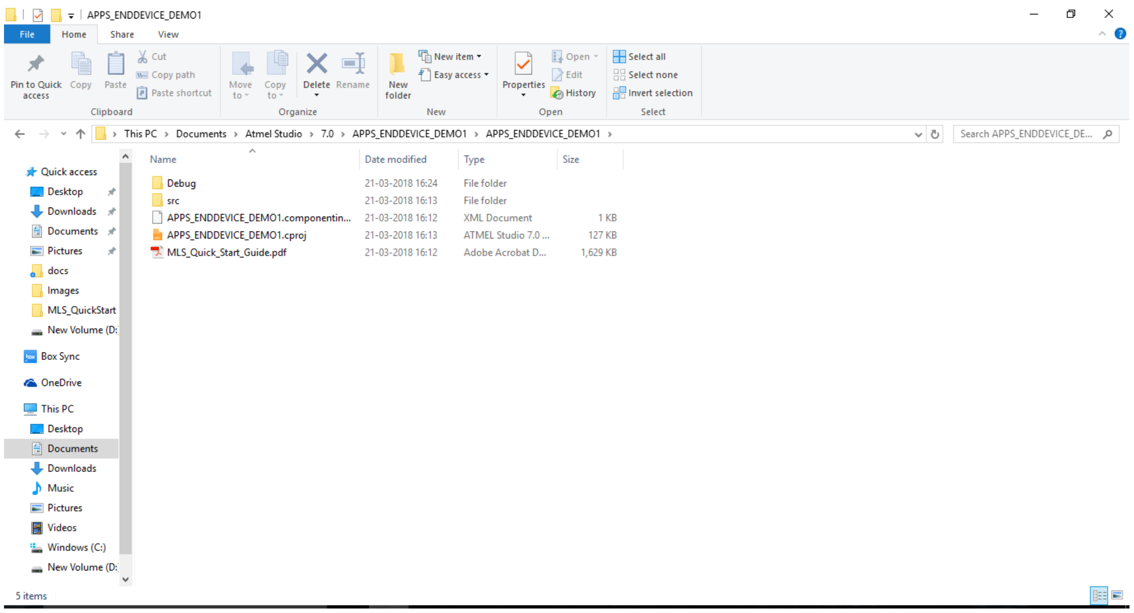
Figure 5-5. Build Output



```
Done building target "CoreBuild" in project "APPS_ENDDEVICE_DEMO.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "D:\repos\preflight\tl
Done building target "Build" in project "APPS_ENDDEVICE_DEMO.cproj".
Done building project "APPS_ENDDEVICE_DEMO.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

8. In the file system, Output files are saved in `Documents\Atmel Studio \7.0\APPS_ENDDEVICE_DEMO1\APPS_ENDDEVICE_DEMO1/[BuildConfiguration]`. Based on the build configuration, the BuildConfiguration directory can be either Debug or Release.

Figure 5-6. Executable File Location



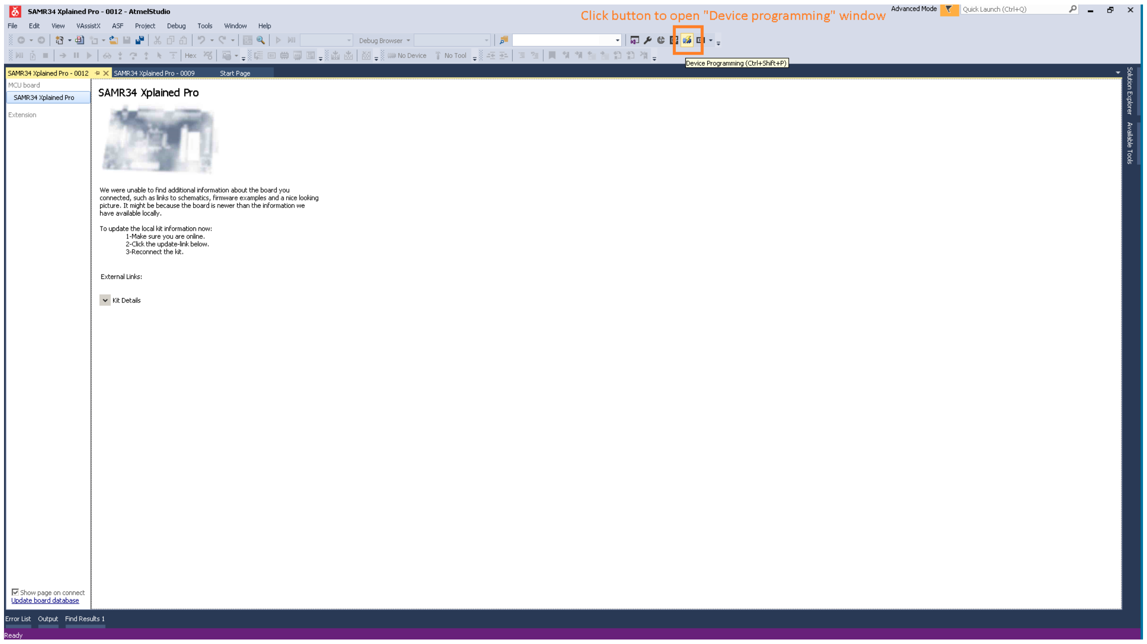
Name	Date modified	Type	Size
Debug	21-03-2018 16:24	File folder	
src	21-03-2018 16:13	File folder	
APPS_ENDDEVICE_DEMO1.componentin...	21-03-2018 16:12	XML Document	1 KB
APPS_ENDDEVICE_DEMO1.cproj	21-03-2018 16:13	ATMEL Studio 7.0 ...	127 KB
MLS_Quick_Start_Guide.pdf	21-03-2018 16:12	Adobe Acrobat D...	1,629 KB

5.2 Flashing the Firmware

Perform the following steps to Flash the firmware on the SAM R34 Xplained Pro board or the WLR089 Xplained Pro board.

1. After successfully building the firmware, connect the SAM R34 Xplained Pro board or the WLR089 Xplained Pro board to the PC through the USB cable. Atmel Studio detects the board after completing the driver installation.
2. Go to `Tools > Device Programming` or press <Ctrl + Shift + P>.

Figure 5-7. Opening the Device Programming Window


SAMR34 Xplained Pro - 0012 - AtmelStudio
File Edit View VAssistX ASF Project Debug Tools Window Help
Click button to open "Device programming" window
Device Programming (Ctrl+Shift+P)
SAMR34 Xplained Pro - 0012
SAMR34 Xplained Pro - 0009
Start Page
MCU board
SAMR34 Xplained Pro
Extension
Show page on connect
Update board database
Error List
Output
Find Results 1
Ready

SAMR34 Xplained Pro

We were unable to find additional information about the board you connected, such as links to schematics, firmware examples and a nice looking picture. It might be because the board is newer than the information we have available locally.

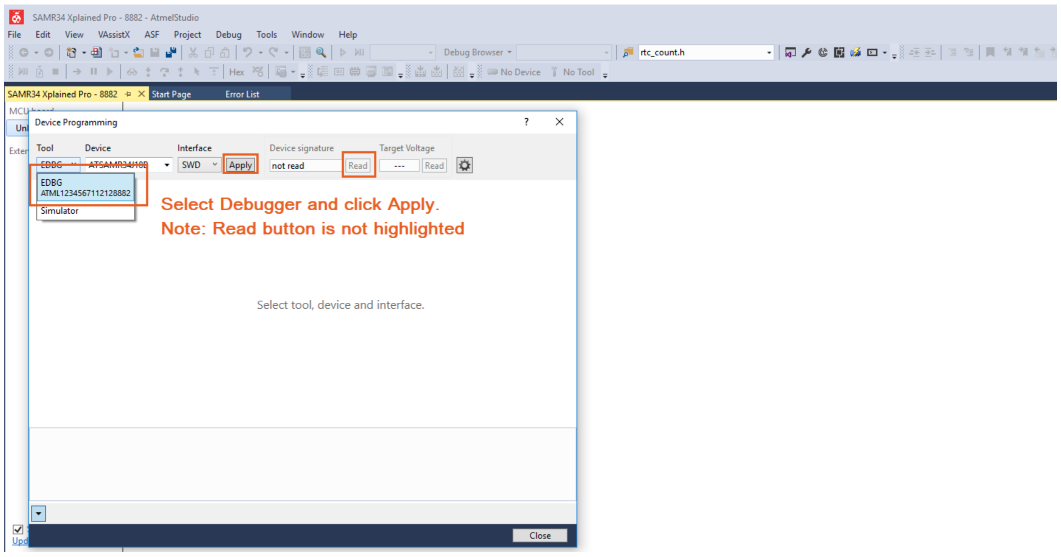
To update the local kit information now:
1-Make sure you are online.
2-Click the update-link below.
3-Reconnect the kit.

External Links:

Kit Details

3. The Device Programming window displays. Perform the following steps:
   3.1. From the Tool list, select **EDBG ATMLXXXXXXX**. This automatically fills the Device field. Click **Apply**.
   3.2. Click **Read** to read the Device Signature value.

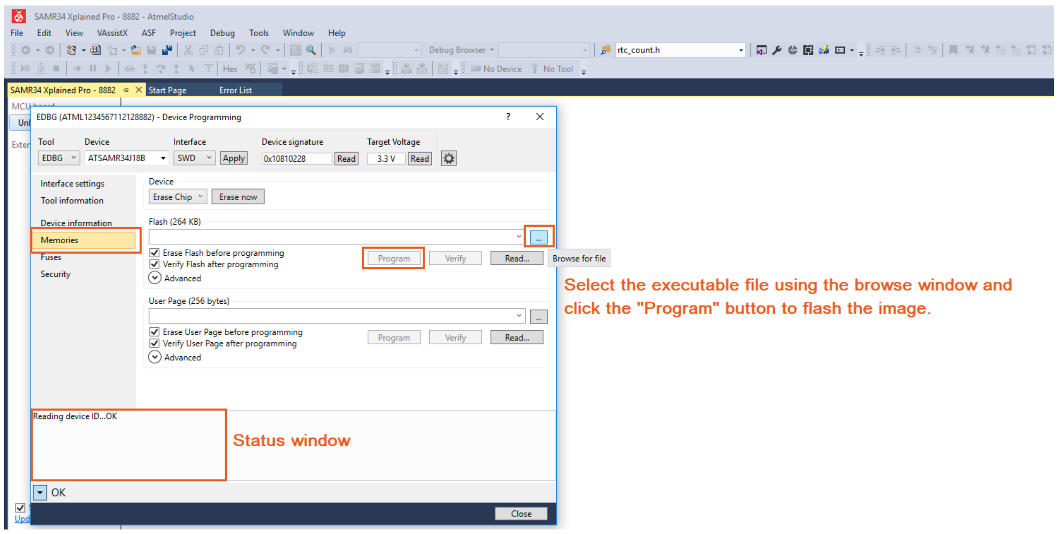
Figure 5-8. Selecting the Debugger


SAMR34 Xplained Pro - 8882 - AtmelStudio
File Edit View VAssistX ASF Project Debug Tools Window Help
rtc_count.h
SAMR34 Xplained Pro - 8882
Start Page
Error List
Device Programming
Tool
Device
Interface
Device signature
Target Voltage
EDBG
ATSAMR34J18B
SWD
Apply
not read
Read

Read
EDBG
ATML1234567112128882
Simulator
Select Debugger and click Apply.
Note: Read button is not highlighted
Select tool, device and interface.
Close

   3.3. From the left-hand menu list, click **Memories**.
   3.4. In the Flash pane, browse for the `elf` file, then click **Program**.

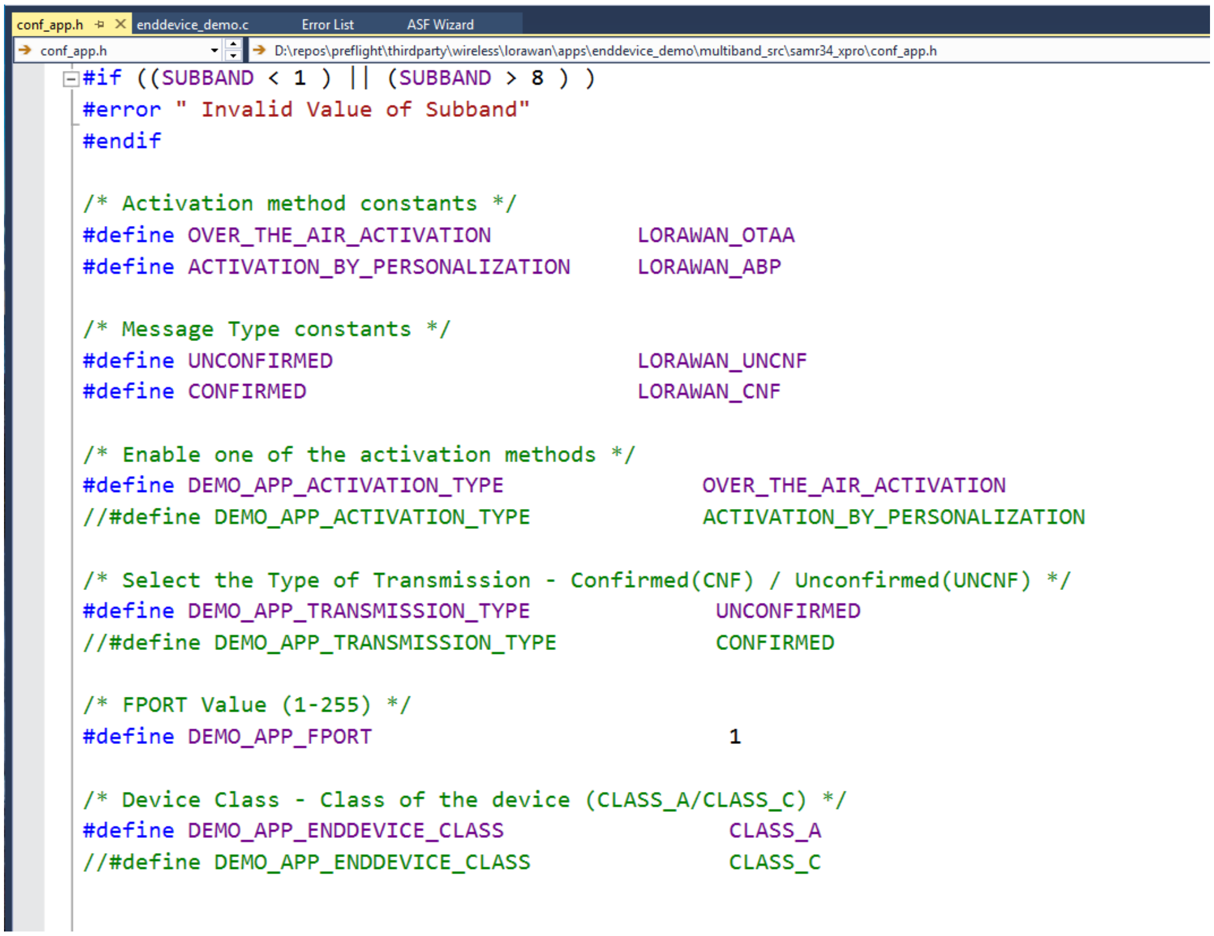
Figure 5-9. Flashing the Image


SAMR34 Xplained Pro - 8882 - AtmelStudio
EDBG (ATML1234567112128882) - Device Programming
Tool: EDBG
Device: ATSAMR34J18B
Interface: SWD
Device signature: 0x10810228
Target Voltage: 3.3 V
Memories
Flash (264 KB)
Program
Browse for file
Select the executable file using the browse window and click the "Program" button to flash the image.
Reading device ID...OK
Status window

5.3 Application Configuration

The EndDevice_Demo application provides configurable parameters in `conf_app.h`. This file is available at `PACKAGE_ROOT/src/config`.

Figure 5-10. Configurable Parameters in LoRaWAN


conf_app.h
enddevice_demo.c
Error List
ASF Wizard
conf_app.h
D:\repos\preflight\thirdparty\wireless\lorawan\apps\enddevice_demo\multiband_src\samr34_xpro\conf_app.h

```
#if ((SUBBAND < 1 ) || (SUBBAND > 8 ) )
#error " Invalid Value of Subband"
#endif

/* Activation method constants */
#define OVER_THE_AIR_ACTIVATION          LORAWAN_OTAA
#define ACTIVATION_BY_PERSONALIZATION    LORAWAN_ABP

/* Message Type constants */
#define UNCONFIRMED                      LORAWAN_UNCNF
#define CONFIRMED                        LORAWAN_CNF

/* Enable one of the activation methods */
#define DEMO_APP_ACTIVATION_TYPE               OVER_THE_AIR_ACTIVATION
//#define DEMO_APP_ACTIVATION_TYPE             ACTIVATION_BY_PERSONALIZATION

/* Select the Type of Transmission - Confirmed(CNF) / Unconfirmed(UNCNF) */
#define DEMO_APP_TRANSMISSION_TYPE              UNCONFIRMED
//#define DEMO_APP_TRANSMISSION_TYPE            CONFIRMED

/* FPORT Value (1-255) */
#define DEMO_APP_FPORT                           1

/* Device Class - Class of the device (CLASS_A/CLASS_C) */
#define DEMO_APP_ENDDEVICE_CLASS                 CLASS_A
//#define DEMO_APP_ENDDEVICE_CLASS               CLASS_C
```

1. This application provides the method of end-device activation.
   ```
   #define DEMO_APP_ACTIVATION_TYPE OVER_THE_AIR_ACTIVATION
   //#define DEMO_APP_ACTIVATION_TYPE ACTIVATION_BY_PERSONALIZATION
   ```
2. This application provides the message type for sending data from an end-device.
   ```
   #define DEMO_APP_TRANSMISSION_TYPE UNCNF
   //#define DEMO_APP_TRANSMISSION_TYPE CNF
   ```
3. This application mentions the port for the uplink data.
   ```
   #define DEMO_APP_FPORT 1
   ```
4. This application can modify or set the Device Address (32-bit) to be used with ABP.
   ```
   #define DEMO_DEVICE_ADDRESS 0x001AD9BB
   ```
5. This application can modify or set the network and application session keys to be used with ABP.
   ```
   #define DEMO_APPLICATION_SESSION_KEY {0x41, 0x63, 0x74, 0x69, 0x6C, 0x69,
   0x74, 0x79, 0x00, 0x04, 0xA3, 0x0B, 0x00, 0x04, 0xA3, 0x0B}

   #define DEMO_NETWORK_SESSION_KEY {0x61, 0x63, 0x74, 0x69, 0x6C,
   0x69, 0x74, 0x79, 0x00, 0x04, 0xA3, 0x0B, 0x00, 0x04, 0xA3, 0x0B}
   ```
6. This application can modify or set the DevEUI (64-bit) to be used with OTAA. SAMR34 Xplained Pro board or WLR089 Xplained Pro board has the DevEUI stored in its EDBG controller; the user has to define

EDBG_EUI_READ as 1 to read DevEUI from edbg and to set it. Otherwise, the value DEMO_DEVICE_EUI configured in conf_app.h will be used as a DevEUI.

Note: By default, EDBG_EUI_READ is defined in conf_board.h.

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

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

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

8. This application can modify or set the AppKey (128-bit) to be used with OTAA.

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

9. This application can modify or set the downlink multicast network and application session keys to be used with Class C.

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

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

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

```
#define DEMO_APP_MCAST_GROUP_ADDRESS 0x0037CC56
```

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

```
#define DEMO_APP_MCAST_ENABLE true
```

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

```
#define DEMO_APP_ENDDEVICE_CLASS CLASS_A
//#define DEMO_APP_ENDDEVICE_CLASS CLASS_C
```

13. Gateways usually support only 8+1 channels and a set of 8 + 1 channels is called as SUBBAND. The NA/AU Regional band has 64 + 8 channels. Therefore, there are eight SUBBAND's in the case of NA/AU region. The application by default is configured to work in SUBBAND 1.

    Change the SUBBAND value according to the gateway/NS configuration. If the gateway supports 64 + 8 channels, then the SUBBAND definition must be commented.

5.4 Stack Attributes

5.4.1 Regional Configuration Parameters

The following table provides regional configuration parameters.

Table 5-1. 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

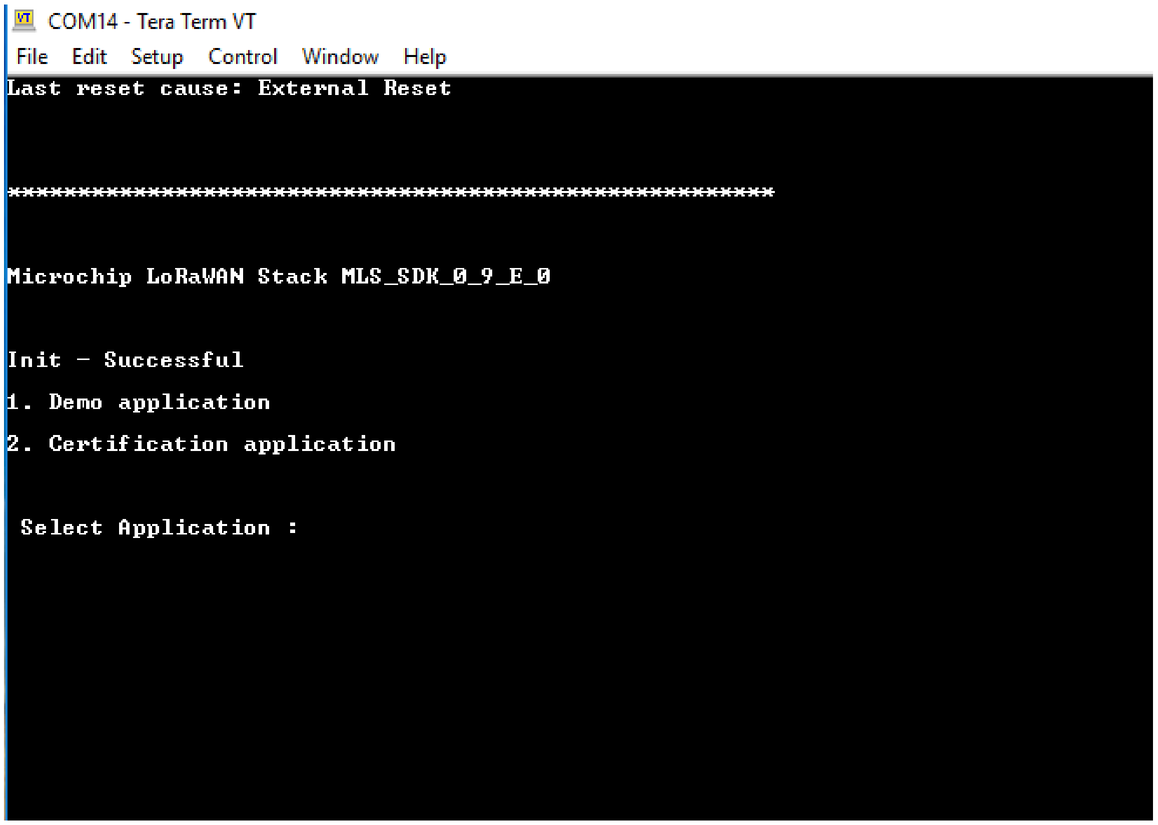
5.5 Demo Application Usage

The EndDevice_Demo_application available in Atmel Studio, is used to send the temperature sensor data through the LoRaWAN network to the network server. It uses UART serial interface with 115200 bps 8N1 configuration and the UART is used to display the menu options. The user input is provided through keyboard.

1. First level menu option is shown in the following figure.
   1.1. Option 1 : Runs the demo application.
   1.2. Option 2 : Runs the EU certification application.

   When option 2 is selected, it will display only one option to run EU certification, which initiates the EU certification application.

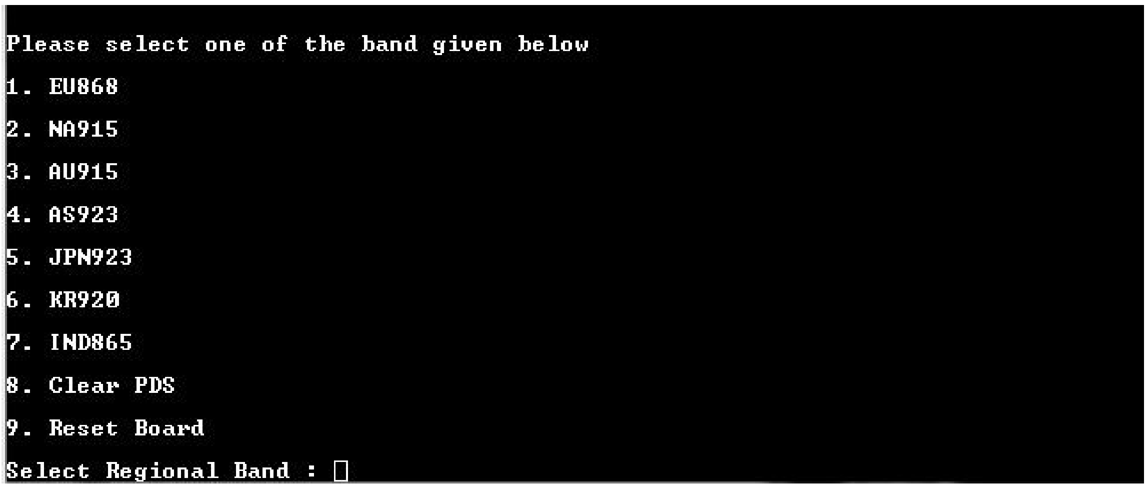
   Figure 5-11. Demo Application First Level Menu Options



```
Last reset cause: External Reset


*******************************************************


Microchip LoRaWAN Stack MLS_SDK_0_9_E_0


Init - Successful

1. Demo application

2. Certification application


 Select Application :
```

2. The second level menu option for the demo application is shown in the following figure.
   2.1. Options 1 to 7 : The regional bands which have the prefix letter as provided in Table 6-1, followed by band frequency.
   2.2. Option 8 : Clears the Flash storage memory (refer to, 6.3 Persistent Data Server).
   2.3. Option 9 : Resets the board (soft Reset), which displays the first level menu option as shown in Figure 5-11.

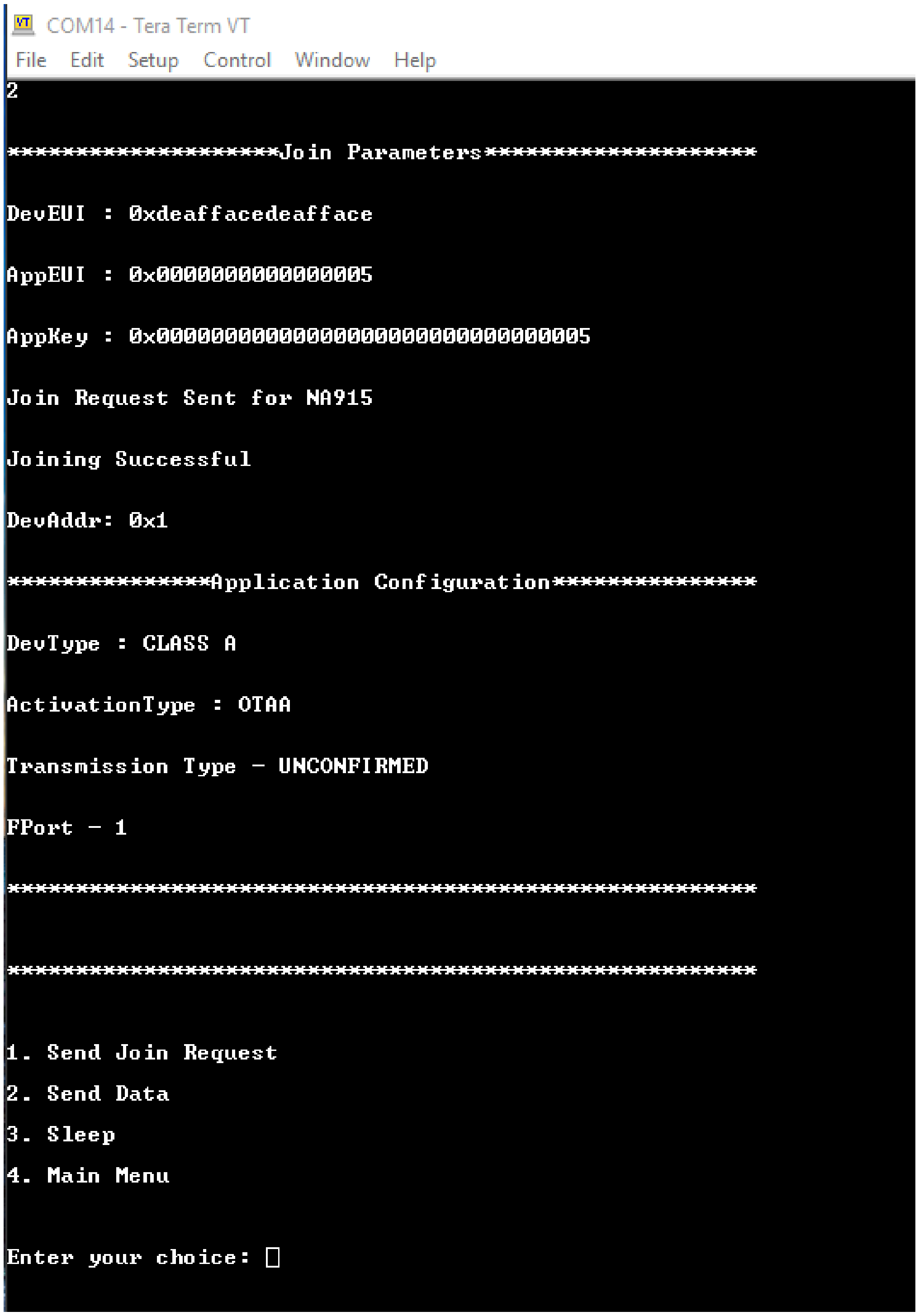
Figure 5-12. Demo Application Second Level Menu Option



```
Please select one of the band given below
1. EU868
2. NA915
3. AU915
4. AS923
5. JPN923
6. KR920
7. IND865
8. Clear PDS
9. Reset Board
Select Regional Band : 
```

3. The end-device joins the network server on selecting any of the regional bands listed in Figure 5-12. It then shows the menu options as shown in the following figure.
   3.1. Option 1 : Sends the join request to the network server.
   3.2. Option 2 : Sends the temperature data to the network server.
   3.3. Option 3 : Puts the end-device into sleep for 1 sec (PMM Standby mode, refer to 6.2 Power Management Module).
   3.4. Option 4 : Displays the main menu, as shown in Figure 5-12.

Figure 5-13. Demo Application Regional Band Menu Option



```
2

********************Join Parameters********************

DevEUI : 0xdeaffacedeafface

AppEUI : 0x0000000000000005

AppKey : 0x00000000000000000000000000000005

Join Request Sent for NA915

Joining Successful

DevAddr: 0x1

***************Application Configuration***************

DevType : CLASS A

ActivationType : OTAA

Transmission Type - UNCONFIRMED

FPort - 1

*******************************************************


*******************************************************


1. Send Join Request
2. Send Data
3. Sleep
4. Main Menu


Enter your choice:
```

6. Supporting MAC Layers

6.1 Regional Configurations

The MLS stack supports multiple regional configurations. It is possible to disable one or more regional configurations at the compile time, but at least one regional configuration must be enabled. MLS also supports run time switching between the supported regional configurations. For the supported regional configuration in the package, refer to the release notes.

The following are the advantages of multiband:

1. Single firmware supporting multiple regional bands.
2. Provision to add or remove the supported regional bands at compile time. This also reduces or increases the Flash and RAM size accordingly.
3. Run time switching between the supported regional bands.

Table 6-1. Supported Regional Band Macro

Macro Switch Name	Band(s) Supported
AS_BAND	Brunei, Cambodia, Indonesia, Laos, New Zealand, Singapore, Taiwan, Thailand, Vietnam
AU_BAND	Australia
EU_BAND	Europe 868 MHz
IND_BAND	India
JPN_BAND	Japan
KR_BAND	Korea
NA_BAND	North America

6.1.1 Dynamic Regional Configuration Support in Application

The following are the steps to add dynamic regional configuration:

1. Enable the required regional band configuration in the MLS. For more information on enabling/disabling the regional band configuration, refer to 5.1 Building the Firmware.
2. Application must call the MAC reset API for every band switching with the required regional band as a parameter. For the MAC reset API and `inc/stack_common.h` for the regional band enumeration, refer to API document.

6.2 Power Management Module

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 SAM R34 MCU and it can be configured either in STANDBY or BACKUP Sleep mode. By default, PMM is enabled and is configured in STANDBY Sleep mode.

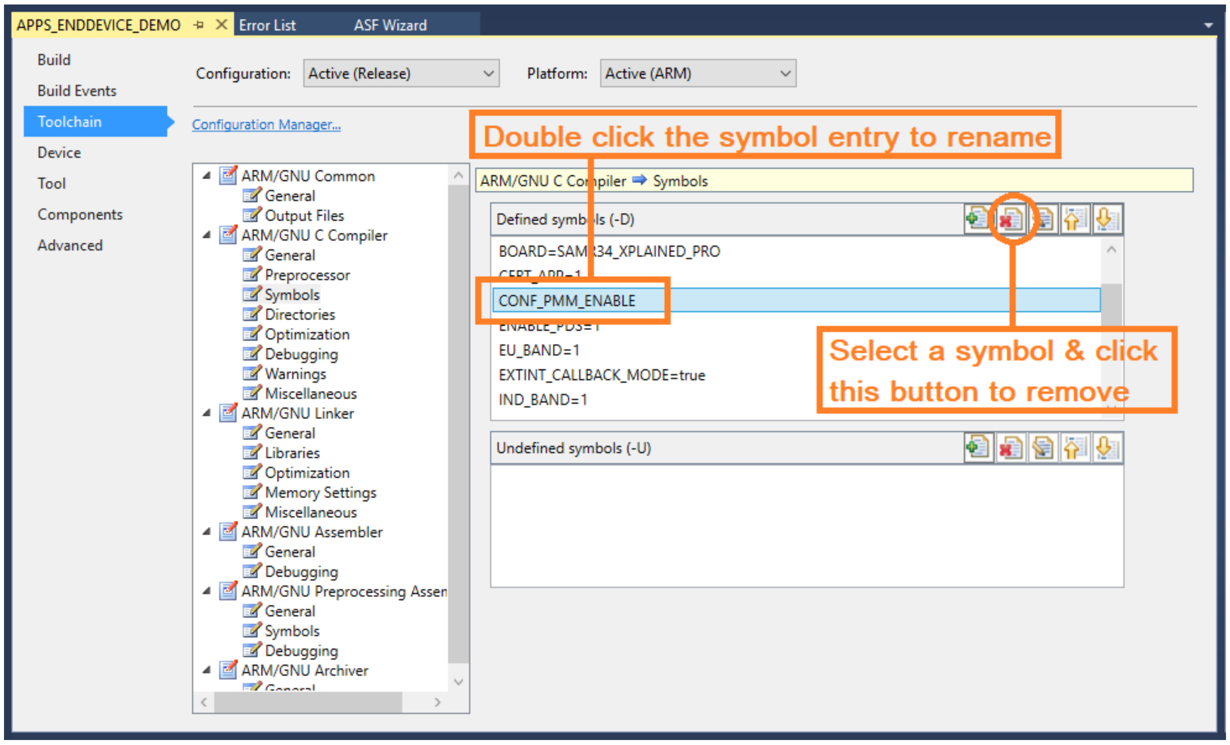
This section describes how to use PMM in the user application. The end-device demo application is used as example in this section. PMM is already included in the end-device demo application. When PMM is included in an application, it defines “CONF_PMM_ENABLE” macro as part of compiler flags. This flag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

To remove PMM from application:

1. Click the **Toolchain** tab listed in project properties window.

2. Select `ARM/GNU C Compiler > Symbols` and then remove or rename the “CONF_PMM_ENABLE” macro from the list of compiler flags. It will then remove PMM from application firmware.

Figure 6-1. PMM Configuration



Configuration: Active (Release)
Platform: Active (ARM)

ARM/GNU C Compiler ➡ Symbols

Defined symbols (-D)

```
BOARD=SAMR34_XPLAINED_PRO
CERT_APP=1
CONF_PMM_ENABLE
ENABLE_PDS=1
EU_BAND=1
EXTINT_CALLBACK_MODE=true
IND_BAND=1
```

6.2.1 Using PMM in Application

Perform the following steps to use PMM in application:

1. Include `pmm.h` to application. In end-device demo application, PMM is included in `main.c`.
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`.
3. Invoke `PMM_Sleep` function from the 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 wake-up callback function is not implemented.

Application sleep request time is configured by the macro “DEMO_CONF_DEFAULT_APP_SLEEP_TIME_MS”. It is present in `conf_app.h` file. By default, application sleep time is 1 second and it can be changed to the desired values. But, the sleep duration must fall within the acceptable range which is given in the following table.

Table 6-2. PMM Parameters

Parameter	Value	Unit	Description
PMM_SLEEP_TIME_MIN	100	milliseconds	Minimum allowed sleep time
PMM_SLEEP_TIME_MAX	0x7CED900	milliseconds	Maximum allowed sleep time is approximately 36 hours, 26 minutes
PMM_WAKEUP_TIME	10	milliseconds	Time to account for wake up

When the end-device is put to sleep, it can wake up from interrupt by either sleep timer, or transceiver interrupt or GPIO interrupt. When the end-device wakes up, the `PMM_Wakeup()` function is called and it returns the elapsed duration from sleep to application. In case of the 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 a sleep timer interrupt or external interrupt. However, the end-device must also call `PMM_Wakeup` for GPIO interrupts. For those GPIO used by the application that can generate interrupts during sleep, the user must call `PMM_Wakeup` in those ISR callbacks. In case of polling, this is not required since polling code works only after wake up.

6.3 Persistent Data Server

Persistent Data Server (PDS) module facilitates storing of stack parameters or attributes in Nonvolatile Memory (NVM) of MCU. The PDS module interfaces between NVM driver and stack.

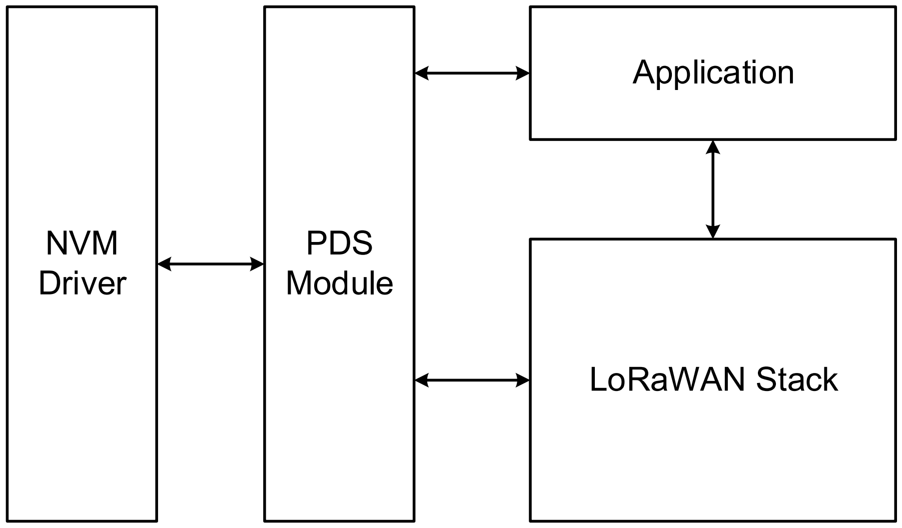
6.3.1 PDS Module Overview

Persistent Data Server (PDS) is a service layer on top of NVM (Nonvolatile Memory). PDS is required because of underlying limitations of the NVM. The following are the limitations of the NVM:

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

A sensor-based application or stack is expected to last long for years and is sometimes time-critical. Therefore, waiting for the milliseconds that are required 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 required which takes care of all the limitations and acts 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 the NVM. The PDS is a component in MLS which manages the storage of any parameter of the stack or application to NVM.

Figure 6-2. PDS Module in MLS Stack



6.3.2 PDS Module Sub-Layers

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

1. NVM sub-layer
2. Wear Leveling sub-layer

3. Files and Items sub-layer
4. Task Handler

6.3.2.1 NVM Sub-Layer

The NVM abstraction manages the following functionality:

- Abstract the address for the EEPROM emulation area and the Flash storage are into logical address so that it is easy to combine both memories.
- Manage the integrity of the information stored.
- The Flash memory in SAM R34 is organized into pages and rows. The following points explain how the NVM is organized:
  - Each row has 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 gets corrupted. So, write granularity is page wise.
  - Data can be erased 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. The data stored in the other pages will be lost due to erasure.
    To prevent this, before issuing an erase, the row must be read to RAM and written back after erasure with the new data.

From the above points, the 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 SAM R34, the size of the NVM Row is 256 bytes. In the NVM sub-layer, each row is given a logical row number in EEPROM Flash section or code Flash section. So, this abstraction manages the map that involves in the translation of logical row number to physical address. If more memory is required it can be added by updating this mapping table.

The integrity of the data storage is done by calculating the 16-bit 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.

6.3.2.2 Wear Leveling Sub-layer

The Wear Leveling sub-layer manages the following functionality:

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 the datasheet, an endurance cycle is a write and erase operation. For NVM Flash present in SAM R34 this endurance is around 100K cycles which is less when 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 Leveling sub-layer provides a translation of physical address abstraction and maintains the information.

The Wear Leveling sub-layer stores data in NVM in the form of Files. The Files are defined by Files and Items sub-layer. Each File is written in a NVM row and therefore, the maximum size of each File must not exceed 255 bytes.

The Wear Leveling sub-layer maintains used and free NVM rows. Based on that information, the Flash physical address is calculated and data is given to NVM sub-layer for storing. On the occurrence of all NVM rows used, the Wear Leveling sub-layer clears older data stored in NVM and frees some rows.

At any given time, the Wear Leveling sub-layer ensures that one copy of all the data stored NVM exists.

6.3.2.3 Files and Items Sub-layer

The Files and Items sub-layer manages the following functionality:

- Organizes the storing/retrieving/deleting of MLS parameters
- Provides APIs to stack and application to perform PDS operations

The basic element in Files and Item abstraction is that an Item and a File is a collection of Items. Therefore, if the user identifies the File ID and Item ID one can easily store/retrieve/delete an item in PDS. The Item is a parameter or 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 requires the following information about the parameter:

- RAM Address
- Size
- File ID mapping
- Item ID allocated to variable
- Items 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 storing or deleting of an Item takes a significant amount of time (unlike read operation in Flash) and these operations are not done synchronously, but instead scheduled by the Task Handler. For more information on the Task Handler execution, refer to 6.3.2.4 Task Handler.

To perform these operations, stack or application must inform the PDS which operation needs to be performed along with Item information. For this purpose, stack or application needs to register in the 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 scan the array registered above to know about the operation to perform. So, from the PDS perspective, it just needs to know which File IDs are used and does not care what is 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 needs to know for which File an operation is pending and not for individual items in a File. So, the PDS maintains a File mask which tracks the items that need an 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 posts a mask for that File ID to inform PDS that an action is pending and it needs to update the Files' marks that the layer maintains. In this case it is "store". Now when the PDS is scheduled, it checks the File mask and it recognizes that an action is pending for a File ID. Then it searches the File array data that the layer registered with PDS and knows 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 is the first time an operation is done for the File. After populating the buffer, it is sent to the Wear Leveling abstraction layer which adds increments the File counter. Then the buffer is passed to the NVM abstraction which adds the 16-bit CRC. Similarly, for PDS delete, except that the Delete bit is set and no RAM copy is performed.

6.3.2.4 Task Handler

The Files and Items abstraction manages the following design goals:

- Manage the latencies of the NVM
- Service the PDS module by scheduling PDS activities

The PDS has organized the data in the form of logical rows. This aids in the management of the Flash access latencies. To erase and write to a row takes about 10 ms, on average. When writing to the Flash a row at a time, the delays are manageable. PDS has the lowest priority in the MLS scheduler. Only after all the layers have completed their execution is the PDS 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 tasks to complete their execution.

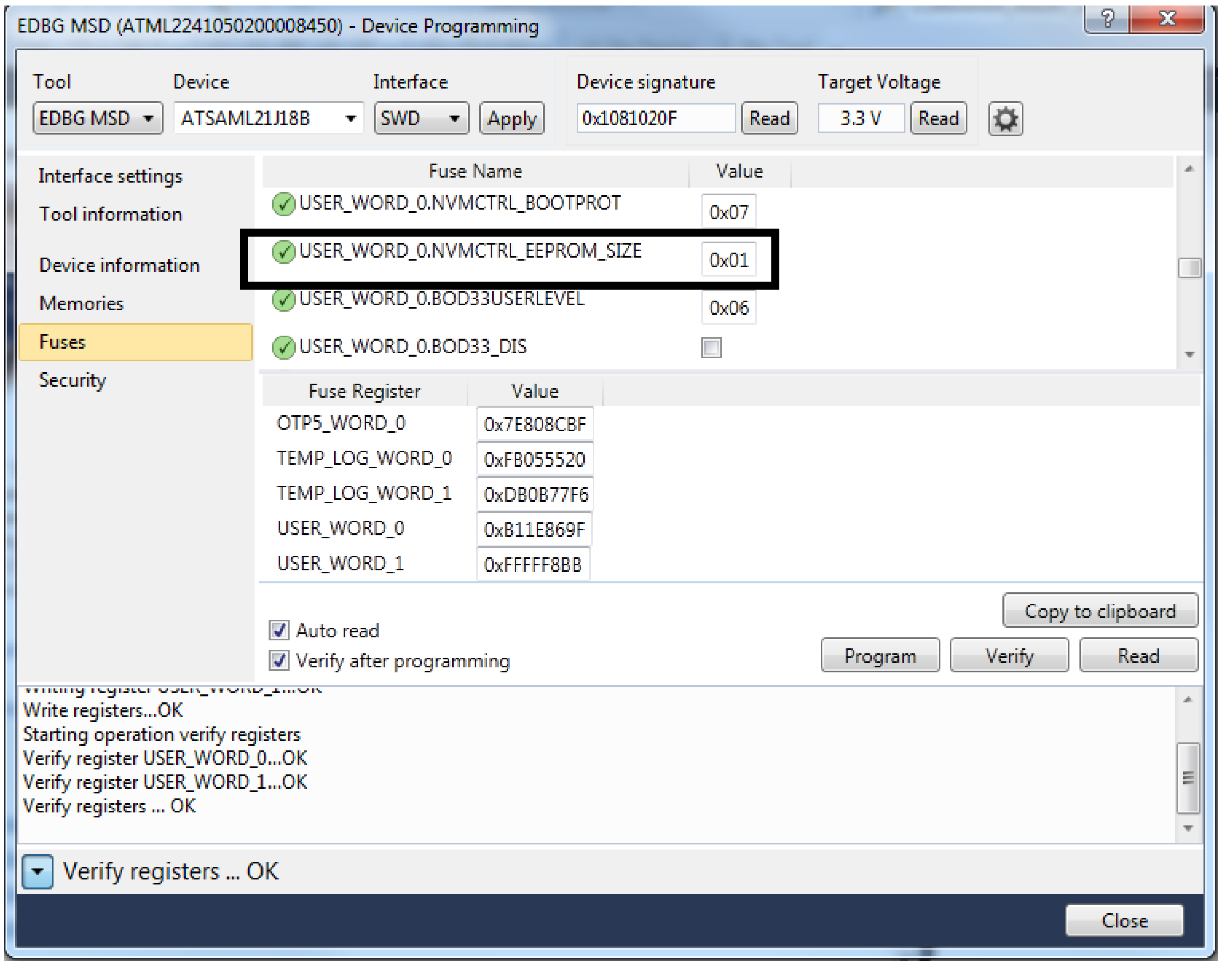
6.3.3 PDS Configuration

PDS is enabled in the project files by default. The "ENABLE_PDS" macro is used for enabling and disabling PDS module. RWW section of SAM R34 SiP is used for storing PDS data.

Perform the following steps to enable RWW section in SAM R34:

1. Configure EEPROM_SIZE macro in `conf_nvm.h` in configuration folder.
2. By default, EEPROM_SIZE is configured as 8192 (8K).
3. Based on the requirement of application size, RWW section can be increased, if required.
4. MLS stack requires 4096 (4K) memory for storing PDS data in RWW section.
5. SAM R34 RWW section allows 4, 8, and 16K memory configuration.
6. The user must to enable RWW section of SAM R34 using fuse settings from Atmel Studio.
7. In the fuse settings, `USER_WORD_0.NVMCTRL_EEPROM_SIZE` setting needs to be updated based on the EEPROM_SIZE macro.

Figure 6-3. Configuring EEPROM



EDBG MSD (ATML2241050200008450) - Device Programming

Tool	Device	Interface	Device signature	Target Voltage
EDBG MSD	ATSAML21J18B	SWD	0x1081020F	3.3 V

Fuse Name	Value
USER_WORD_0.NVMCTRL_BOOTPROT	0x07
USER_WORD_0.NVMCTRL_EEPROM_SIZE	0x01
USER_WORD_0.BOD33USERLEVEL	0x06
USER_WORD_0.BOD33_DIS	☐

Fuse Register	Value
OTP5_WORD_0	0x7E808CBF
TEMP_LOG_WORD_0	0xFB055520
TEMP_LOG_WORD_1	0xDB0B77F6
USER_WORD_0	0xB11E869F
USER_WORD_1	0xFFFFF8BB

☑ Auto read
☑ Verify after programming

```
Write registers...OK
Starting operation verify registers
Verify register USER_WORD_0...OK
Verify register USER_WORD_1...OK
Verify registers ... OK
```

Verify registers ... OK

Table 6-3. EEPROM Field Value and Row and Size Allocation

EEPROM [2:0]	Rows Allocated to EEPROM	EEPROM Size in Bytes
7	None	0
6	1	256
5	2	512
4	4	1024
3	8	2048

...........continued

EEPROM [2:0]	Rows Allocated to EEPROM	EEPROM Size in Bytes
2	16	4096
1	32	8192
0	64	16384

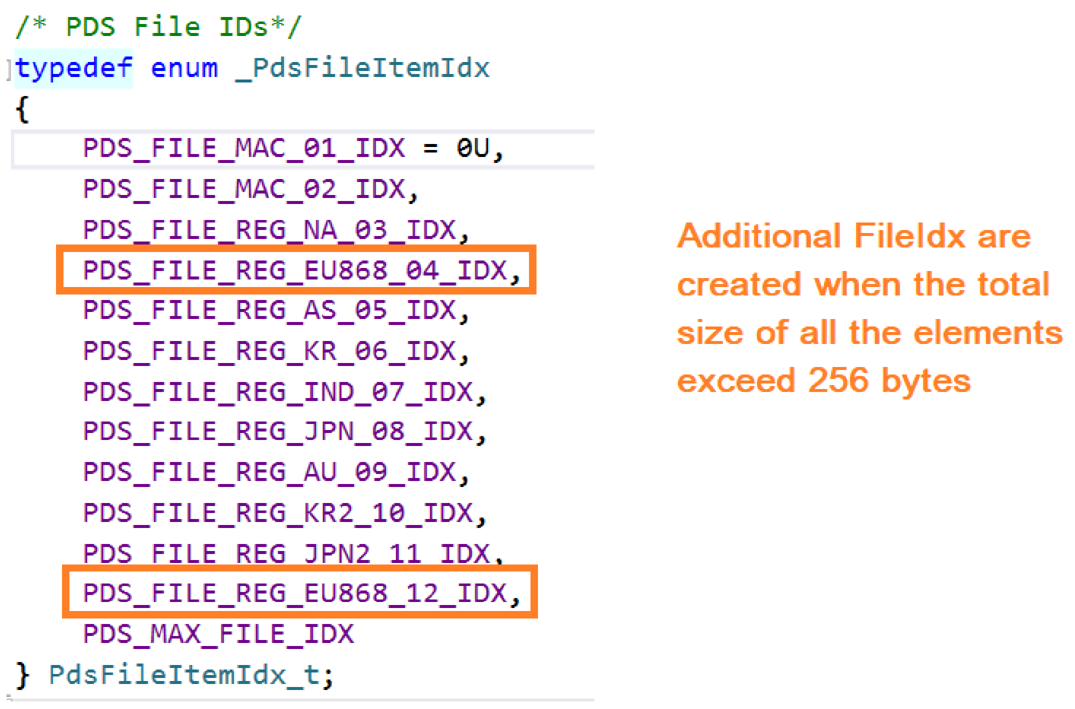
Note: The RWW section must be enabled for each SAM R34 board. By default, the RWW section is disabled in SAM R34. Before flashing firmware into the SAM R34, enable the RWW section in `Tools > Device Programming > Fuses` and change the `USER_WORD_0.NVMCTRL_EEPROM_SIZE` fuse value to one of the above table values based on the EEPROM_SIZE configured in the project.

6.3.4 Using PDS in the Application

The following are the sequence of steps to use PDS in the application:

1. Include `pds_interface.h` in the application
2. Create an instance of the structure `PdsFileMarks_t` and update the fields. The maximum size combining all the elements in `PdsFileMarks_t` is 256 bytes.
3. Each instance of `PdsFileMarks_t` represents the data to be stored in one NVM row, which is 256 bytes. So, additional instances of the `PdsFileMarks_t` structure are created when the size of all the elements exceeds 256 bytes.
4. Create a File ID for each instance of `PdsFileMarks_t` and append it in the enum list `PdsFileItemIdx_t` available in file `pds_interface.h`.

Figure 6-4. PDS File IDs



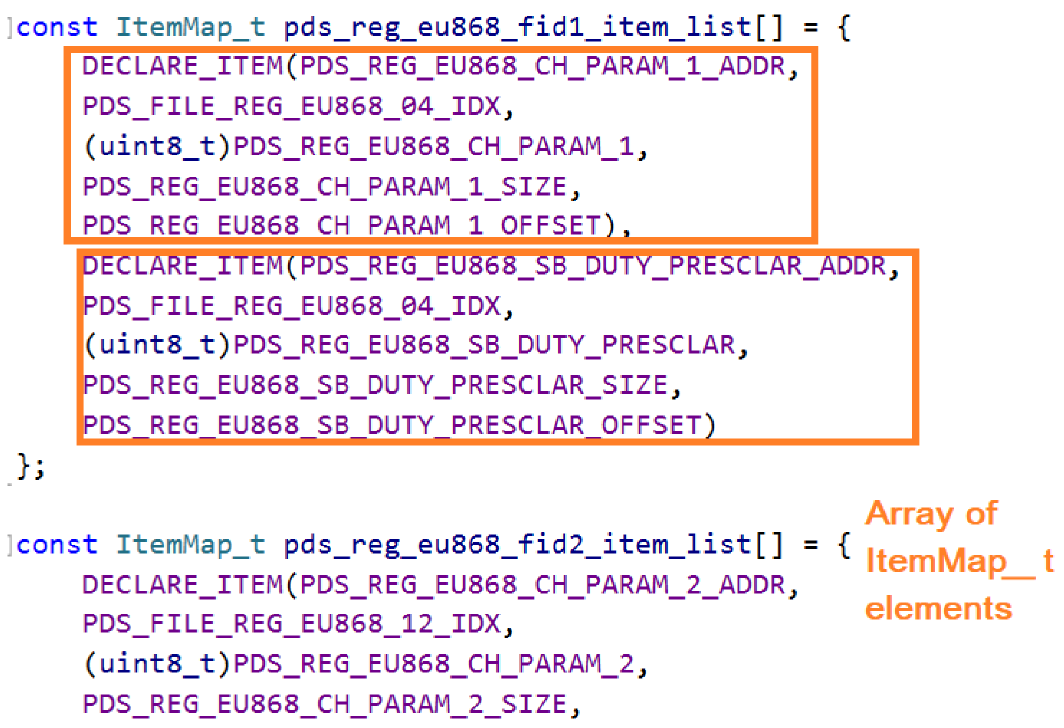
```
/* PDS File IDs*/
typedef enum _PdsFileItemIdx
{
    PDS_FILE_MAC_01_IDX = 0U,
    PDS_FILE_MAC_02_IDX,
    PDS_FILE_REG_NA_03_IDX,
    PDS_FILE_REG_EU868_04_IDX,
    PDS_FILE_REG_AS_05_IDX,
    PDS_FILE_REG_KR_06_IDX,
    PDS_FILE_REG_IND_07_IDX,
    PDS_FILE_REG_JPN_08_IDX,
    PDS_FILE_REG_AU_09_IDX,
    PDS_FILE_REG_KR2_10_IDX,
    PDS_FILE_REG_JPN2_11_IDX,
    PDS_FILE_REG_EU868_12_IDX,
    PDS_MAX_FILE_IDX
} PdsFileItemIdx_t;
```

Additional FileIdx are created when the total size of all the elements exceed 256 bytes

5. Create array instances of the structure `ItemMap_t`, according the number of elements to be stored. Ensure the elements have File ID in the MSB byte of the 16-bit data as shown in Figure 6-6. Macro `DECLARE_ITEM()` shown in Figure 6-5 assigns appropriate value to the `ItemMap_t` structure and it is declared in the

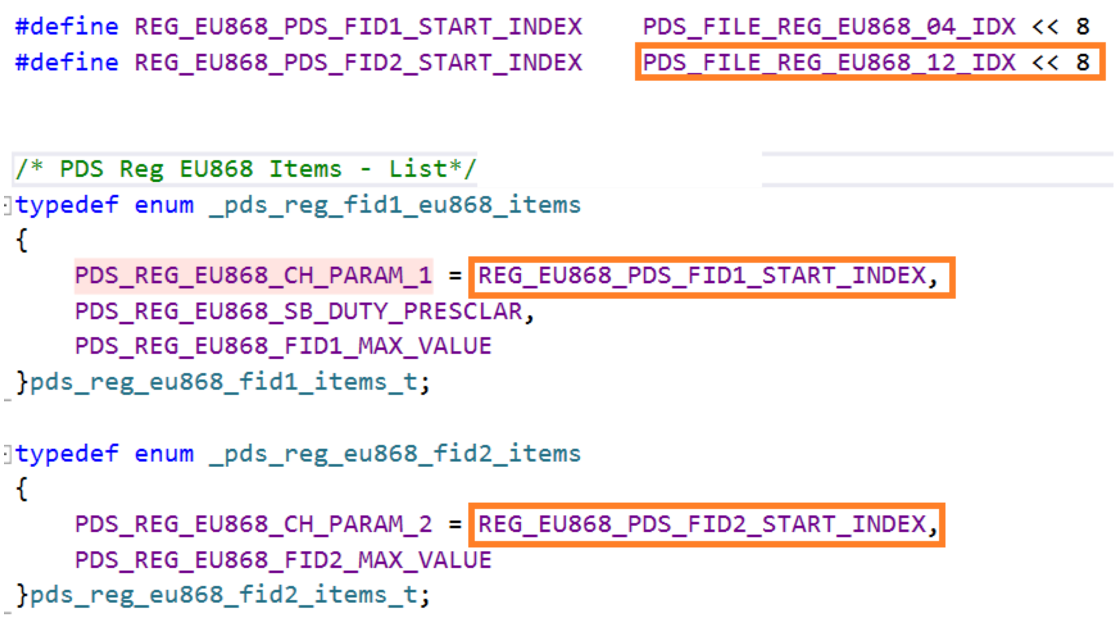
`pds_interface.h`. The third parameter to the macro function is the element value and it has the File ID in the MSB. The macro function masks the File ID and assigns the element value to the structure element.

Figure 6-5. Array of ItemMap Elements



```
const ItemMap_t pds_reg_eu868_fid1_item_list[] = {
    DECLARE_ITEM(PDS_REG_EU868_CH_PARAM_1_ADDR,
    PDS_FILE_REG_EU868_04_IDX,
    (uint8_t)PDS_REG_EU868_CH_PARAM_1,
    PDS_REG_EU868_CH_PARAM_1_SIZE,
    PDS_REG_EU868_CH_PARAM_1_OFFSET),
    DECLARE_ITEM(PDS_REG_EU868_SB_DUTY_PRESCLAR_ADDR,
    PDS_FILE_REG_EU868_04_IDX,
    (uint8_t)PDS_REG_EU868_SB_DUTY_PRESCLAR,
    PDS_REG_EU868_SB_DUTY_PRESCLAR_SIZE,
    PDS_REG_EU868_SB_DUTY_PRESCLAR_OFFSET)
};

const ItemMap_t pds_reg_eu868_fid2_item_list[] = {
    DECLARE_ITEM(PDS_REG_EU868_CH_PARAM_2_ADDR,
    PDS_FILE_REG_EU868_12_IDX,
    (uint8_t)PDS_REG_EU868_CH_PARAM_2,
    PDS_REG_EU868_CH_PARAM_2_SIZE,
```

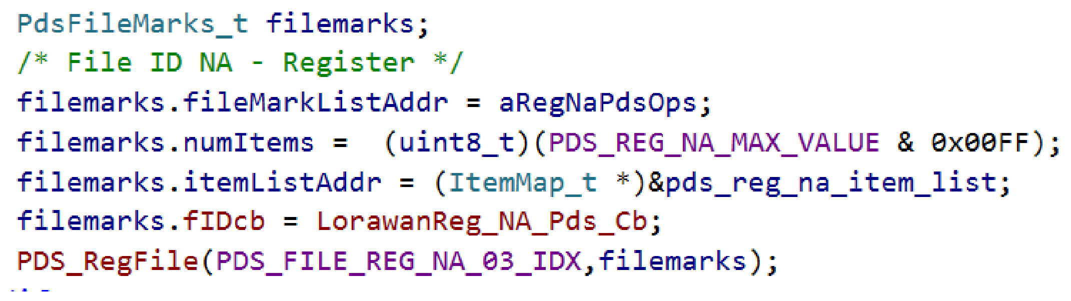
Figure 6-6. File ID Appended to Element List



```
#define REG_EU868_PDS_FID1_START_INDEX    PDS_FILE_REG_EU868_04_IDX << 8
#define REG_EU868_PDS_FID2_START_INDEX    PDS_FILE_REG_EU868_12_IDX << 8


/* PDS Reg EU868 Items - List*/
typedef enum _pds_reg_fid1_eu868_items
{
    PDS_REG_EU868_CH_PARAM_1 = REG_EU868_PDS_FID1_START_INDEX,
    PDS_REG_EU868_SB_DUTY_PRESCLAR,
    PDS_REG_EU868_FID1_MAX_VALUE
}pds_reg_eu868_fid1_items_t;

typedef enum _pds_reg_eu868_fid2_items
{
    PDS_REG_EU868_CH_PARAM_2 = REG_EU868_PDS_FID2_START_INDEX,
    PDS_REG_EU868_FID2_MAX_VALUE
}pds_reg_eu868_fid2_items_t;
```

6. Register each instance of the `PdsFileMarks_t` with the File ID and the instance name.

Figure 6-7. FileMarks Structure Registration



```
PdsFileMarks_t filemarks;
/* File ID NA - Register */
filemarks.fileMarkListAddr = aRegNaPdsOps;
filemarks.numItems =  (uint8_t)(PDS_REG_NA_MAX_VALUE & 0x00FF);
filemarks.itemListAddr = (ItemMap_t *)&pds_reg_na_item_list;
filemarks.fIDcb = LorawanReg_NA_Pds_Cb;
PDS_RegFile(PDS_FILE_REG_NA_03_IDX,filemarks);
```

7. Once registered, the application can use the PDS APIs with File ID and instance name. For more information on the PDS APIs, refer to the API document.

6.4 Software Timer

A timer provides the facility to measure time. SAM R34 has five hardware timers. Every component in MLS such as the RADIO, MAC, APP needs a timer, therefore, the timers need to be efficiently shared among all the components.

A software timer is used to provide the necessary abstractions for MLS to use the hardware timers. It manages the operation of the hardware timer, thus freeing the user from managing the hardware timers directly. The hardware timer TC0 is used in the stack to configure all the software timers.

The software timer provides a set of interfaces to initialize, create, start, and stop timers. The start of a timer function has parameters for measuring the time duration as well as a callback function. Once the duration of time has elapsed, the user-supplied callback function will be invoked.

More than one software timer can be started simultaneously. They are automatically sorted according to their duration and expiration accordingly. Along with running for a user-specified duration, the software timer module also keeps track of system time. System time is measured from the initialization of the software timer during Reset.

MLS currently supports a maximum of 25 software timer instances. It can be customized as per application requirements by changing the following macro in `conf_app.h`.

```
/* Number of software timers */
#define TOTAL_NUMBER_OF_TIMERS (25u)
```

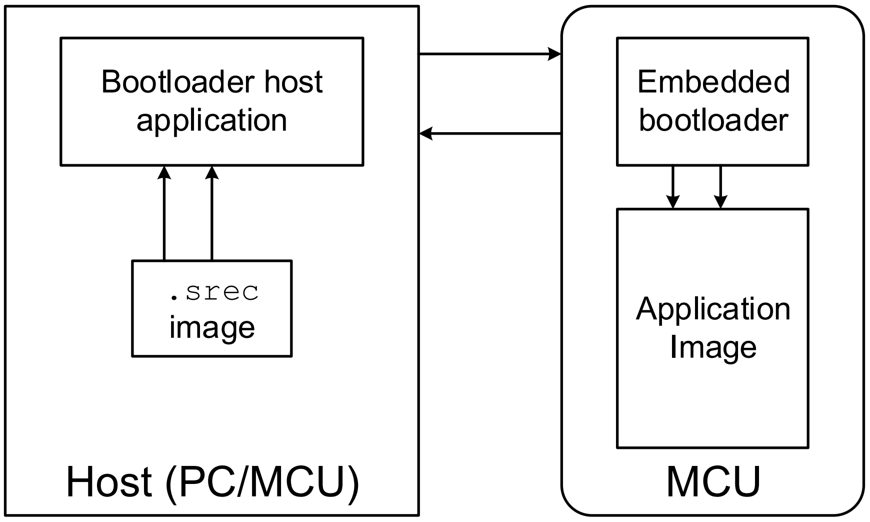
Note: Changing the number of software timers requires a rebuilding of application firmware.

6.5 Bootloader

The bootloader is available for the SAM R34 as a separate application, which allows the user to Flash an application image into the board without the use of an external debugger. After Reset, the board listens on the serial port (UART) for the image update request for 200 ms. If an update request is received within that time, the MCU then waits for the image, which it then writes into the board's internal Flash and loads the new image during the next Reset.

The PC utility image `Bootloader_PC_tool_setup` supplied with the package is necessary to Flash the image into SAM R34. Upon Reset the software takes the application image in Motorola, S-record hexadecimal format (S-REC) and sends the image via the serial interface to the target MCU. The bootloader is configured to write the application image to Flash, starting from location `0x2000`. The application must be build to load from location `0x2000`. The package contains a preconfigured linker script for this purpose. The linker script for non-bootloader project is available in the path `sam0\utils\linker_scripts\samr34\gcc\samr34j18b_flash.ld`. The linker script for bootloader project is available in the path `thirdparty\wireless\lorawan\utils\bootloader\samr34j18b_flash.ld`.

Figure 6-8. Bootloader Organization in SAM R34



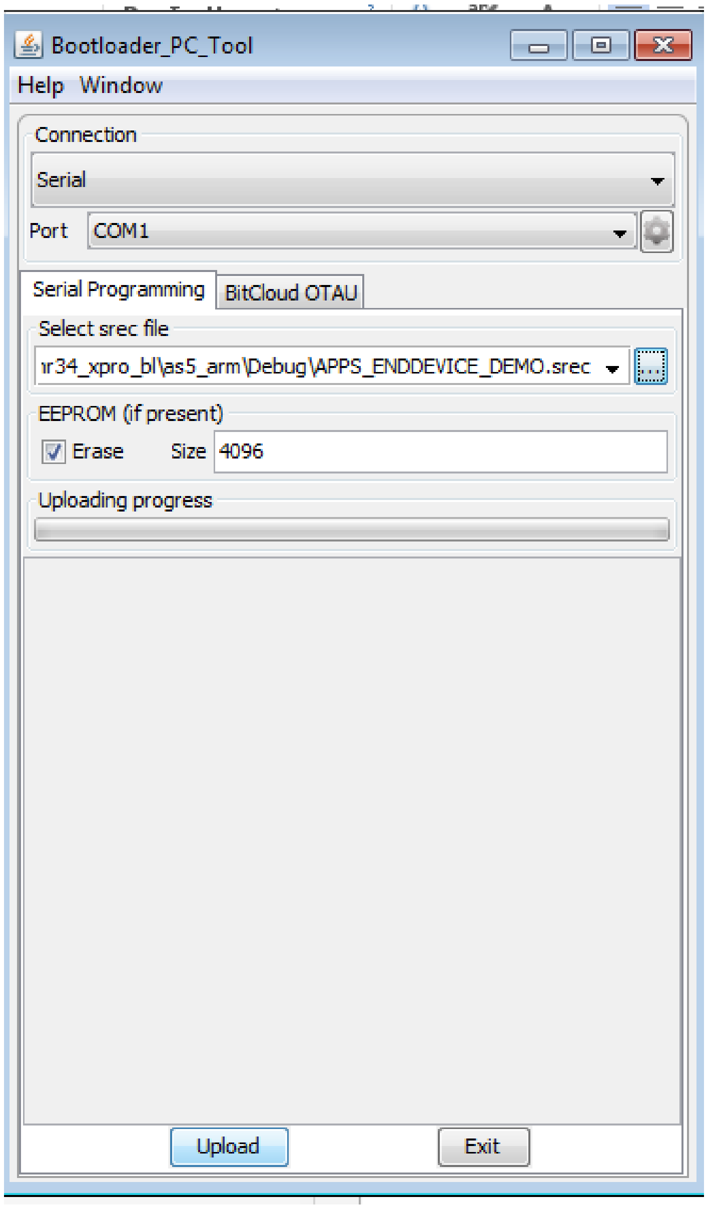
6.5.1 Bootloader PC Tool Usage

Install the Bootloader PC tool, launch the installation file from the `\PC_Bootloader_Setup\` and proceed with the instructions.

Perform the following steps to program an MCU using the serial bootloader:

1. Press <F7> in Atmel Studio to generate the application srec `APPS_ENDDEVICE_DEMO.srec`.
2. Program `wireless\lorawan\utils\bootloader\SerialBootloader_SAMR34J18B_XPRO_BAUDRATE_115200.elf` via Embedded debugger/JTAG programming interface in SAM R34 Xplained Pro board.
3. Connect the board to the PC via a serial connection (EDBG).
4. Double-click and open the Bootloader PC tool application for the GUI version of the serial bootloader.
5. Specify uploading parameters:
   5.1. Select the connection type as “Serial”.
   5.2. Select the COM port from the drop-down list. Set the bit rate as 115200.
   5.3. Select the firmware srec file to be uploaded; there is a restriction on the size of firmware downloaded by serial booting process. The serial bootloader cannot rewrite the area where the bootstrap code resides.
6. Press the **Upload** button if the Bootloader PC GUI tool is used. For the console bootloader, press <Enter> on the keyboard to start uploading.
7. Press the HW reset button on the device if requested. The Bootloader PC tool waits for approximately 30 seconds for the button to be released. If this does not happen, programming will be aborted.
8. The Bootloader PC tool indicates the programming progress. Once loading is successfully finished, the device automatically restarts. If loading fails, the Bootloader PC tool will indicate the reason. If the new image upload fails (for example, because of random communication errors), the device must be reprogrammed. If the reprogramming does not resolve the issue, then the previously programmed image code in the device may be corrupted. The device must be erased and reprogrammed via JTAG.

Figure 6-9. Bootloader PC Tool Configuration



Connection
Serial
Port COM1
Serial Programming | BitCloud OTAU
Select srec file
ır34_xpro_bl\as5_arm\Debug\APPS_ENDDEVICE_DEMO.srec
EEPROM (if present)
☑ Erase Size 4096
Uploading progress

Note: If needed, install the Bootloader PC tool *Bootloader_PC_Tool_Setup_1.2.2.233.exe,* which comes with the LoRaWAN Mote Application with Bootloader project.

7. Document Revision History

Revision	Date	Section	Changes
A	10/2018	Document	Initial release
B	09/2020	Document	Added WLR089 Xplained Pro board support related changes across the document
		1. Quick Reference Info	Added new chapter as per the standard

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 (FAQs), technical support requests, online discussion groups, Microchip design partner 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

Product Change Notification Service

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Legal Notice

Trademarks

ISBN: 978-1-5224-6636-9

Quality Management System

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
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

Australia - Sydney
Tel: 61-2-9868-6733
China - Beijing
Tel: 86-10-8569-7000
China - Chengdu
Tel: 86-28-8665-5511
China - Chongqing
Tel: 86-23-8980-9588
China - Dongguan
Tel: 86-769-8702-9880
China - Guangzhou
Tel: 86-20-8755-8029
China - Hangzhou
Tel: 86-571-8792-8115
China - Hong Kong SAR
Tel: 852-2943-5100
China - Nanjing
Tel: 86-25-8473-2460
China - Qingdao
Tel: 86-532-8502-7355
China - Shanghai
Tel: 86-21-3326-8000
China - Shenyang
Tel: 86-24-2334-2829
China - Shenzhen
Tel: 86-755-8864-2200
China - Suzhou
Tel: 86-186-6233-1526
China - Wuhan
Tel: 86-27-5980-5300
China - Xian
Tel: 86-29-8833-7252
China - Xiamen
Tel: 86-592-2388138
China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
India - New Delhi
Tel: 91-11-4160-8631
India - Pune
Tel: 91-20-4121-0141
Japan - Osaka
Tel: 81-6-6152-7160
Japan - Tokyo
Tel: 81-3-6880- 3770
Korea - Daegu
Tel: 82-53-744-4301
Korea - Seoul
Tel: 82-2-554-7200
Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906
Malaysia - Penang
Tel: 60-4-227-8870
Philippines - Manila
Tel: 63-2-634-9065
Singapore
Tel: 65-6334-8870
Taiwan - Hsin Chu
Tel: 886-3-577-8366
Taiwan - Kaohsiung
Tel: 886-7-213-7830
Taiwan - Taipei
Tel: 886-2-2508-8600
Thailand - Bangkok
Tel: 66-2-694-1351
Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393
Denmark - Copenhagen
Tel: 45-4485-5910
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
Germany - Garching
Tel: 49-8931-9700
Germany - Haan
Tel: 49-2129-3766400
Germany - Heilbronn
Tel: 49-7131-72400
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: 47-72884388
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