STM32W108xx Simple MAC library

1 Introduction

The STM32W108 Simple MAC (media access control) library provides a set of APIs to access the lower-MAC functionality of the STM32W108HB and STM32W108CB microcontrollers (STM32Wxx). These devices integrate a 2.4 GHz IEEE 802.15.4-compliant transceiver featuring 16 channels and supporting 250 Kbps transfers.

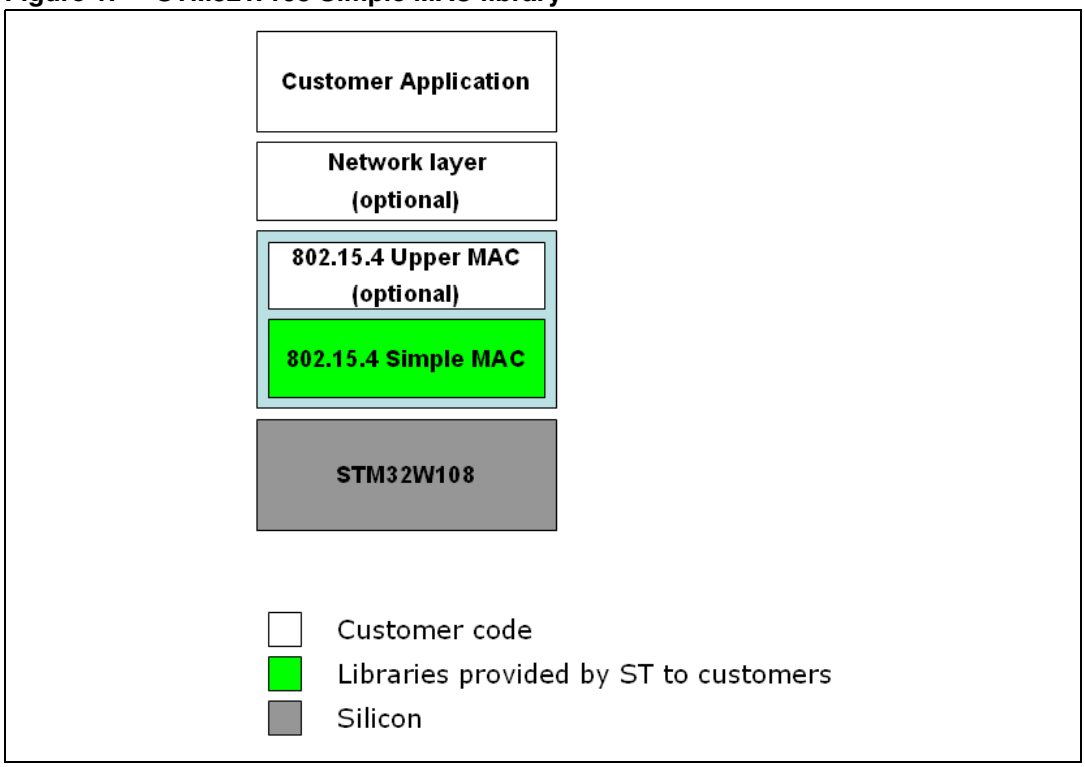
The Simple MAC library is designed to run on STM32W108 engineering samples and on the STM32W108xBU64 microcontroller.

In addition, the Simple MAC library will allow developing specific stacks based on the IEEE 802.15.4 standard.

The present document provides information on:

- IEEE 802.15.4 protocol
- STM32W108 Simple MAC library features
- How to build and run STM32W108 Simple MAC demonstration applications
- How to design an application using the STM32W108 Simple MAC library APIs

Figure 1. STM32W108 Simple MAC library



Contents

List of tables

List of figures

2 IEEE 802.15.4 protocol

The IEEE 802.15.4 is a standard which specifies the physical layer (PHY) and the media access control (MAC) layer for low-rate wireless personal area networks (LR-WPANs).

The main features are as follows:

- Channel access via carrier sense multiple access with collision avoidance (CSMA-CA)
- Optional time slotting and network beaconing
- Message acknowledgement
- Multilevel security
- Suitable for long battery devices, with selectable latency to match different power requirements (sensors, remote monitoring, …)

  The IEEE 802.15.4 standard distinguishes between two types of devices:

  – Full function devices (FFD):

    Full function devices can be common nodes or coordinators of personal area networks (PANs). They can communicate with any device connected to the network and relay messages. FFD devices are suitable for any type of network topology.

  – Reduced function devices (RFD):

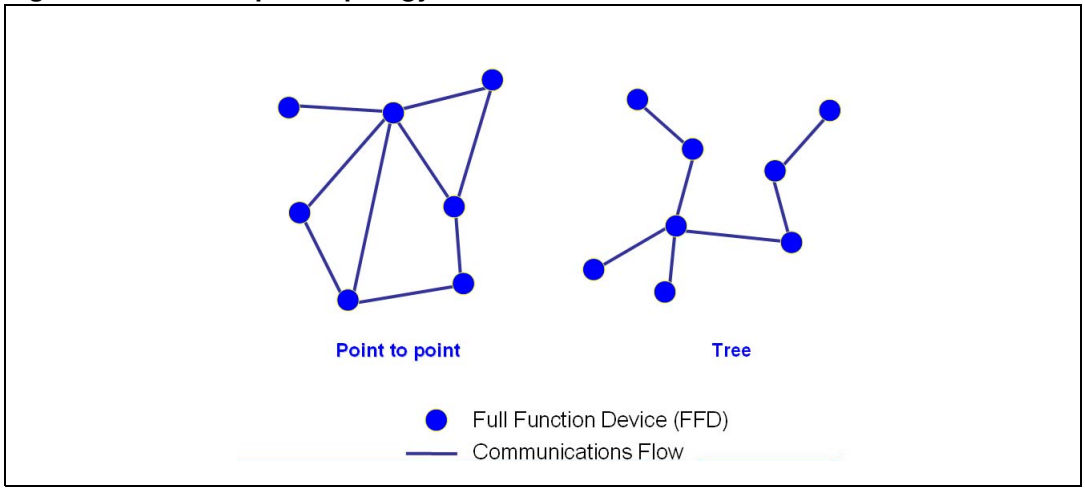
    Reduced function devices are simple devices with limited resources and communication capabilities. They can only communicate with FFDs and can never act as coordinators. RFD devices are only suitable for star networks.

2.1 IEEE 802.15.4 networks

Two type of network topologies are supported:

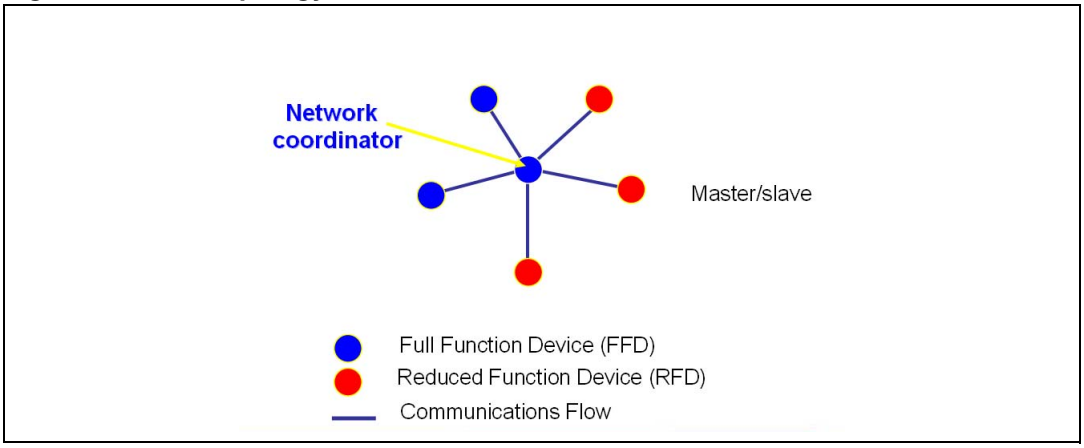
- Peer-to-peer networks where each device can communicate with any other device as long as they are in range of one another.

Figure 2. Peer-to-peer topology



- Star networks where communications are established between devices and a single central controller, called the PAN coordinator.

Figure 3. Star topology



2.2 IEEE 802.15.4 device addressing

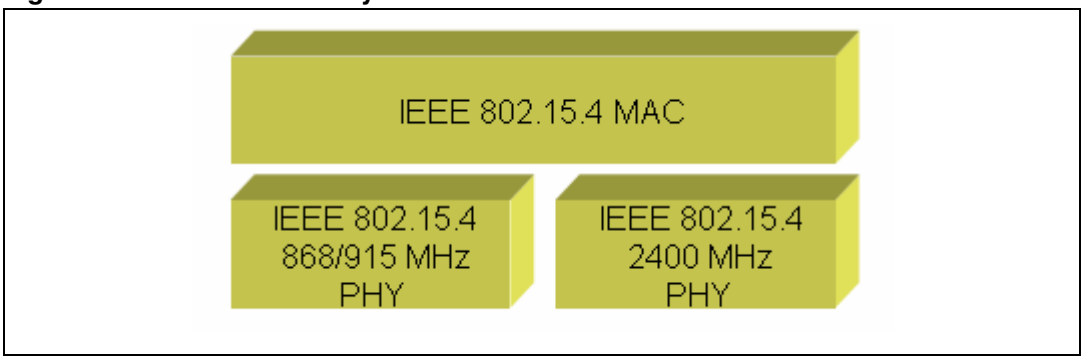
A unique PAN identifier is assigned to each independent PAN built on a channel.

All devices connected to the network have a unique 64-bit extended address. This address is used for direct communications through the PAN. A device can also use a short 16-bit address which is allocated by the PAN coordinator when the device is associated to the network.

2.3 PHY and MAC IEEE 802.15.4 protocol layers

Figure 4 shows the IEEE 802.15.4 MAC and PHY layers.

Figure 4. PHY and MAC layers



2.3.1 IEEE 802.15.4 PHY layer

The PHY layer manages the physical RF transceiver. It is also in charge of the following tasks:

- Activating and deactivating of the radio transceiver
- Energy detection for the current channel
- Indicating link quality for the received packets
- CCA for the CSMA-CA
- Selecting channel frequency
- Data transmission and reception

The standard specifies two PHY layers:

- 868/915 MHz direct sequence spread spectrum (DSSS) PHY
  One 20 Kbps channel in the European 868 MHz band
  Ten 40 Kbps channels in the 915 MHz ISM band (from 902 to 928 MHz)
- 2 450 MHz DSSS PHY supporting sixteen 250 Kbps channels in the 2.4 GHz band

Table 1. IEEE 802.15.4 PHY parameters

Parameter	2.4 GHz PHY	868/915 MHz PHY
Sensitivity @ 1% PER	-85 dBm	-92 dBm
Receiver maximum input level	-20 dBm	
Adjacent channel rejection	0 dB	
Alternate channel rejection	30 dB	
Output power (lowest maximum)	-3 dBm	
Transmit modulation accuracy	EVM < 35% for 1000 chips	
Number of channels	16	1/10
Channel spacing	5 MHz	Single-channel at 2 MHz
Transmission rates: Data rate Symbol rate Chip rate	 250 kbps 62.5 ksymbol/s 2 Mchip/s	 20/40 kbps 20/40 ksymbol/s 300/600 kchip/s
Chip modulation	O-QPSK with half-sine pulse shaping	BPSK with raised cosine pulse shaping

IEEE 802.15.4 PHY CCA

The following CCA modes are supported:

- CCA mode 1: energy above threshold (lowest)
- CCA mode 2: carrier sense (medium)
- CCA mode 3: carrier sense with energy above threshold (strongest)

IEEE 802.15.4 PHY link quality indication (LQI)

The LQI characterizes the strength and/or quality of a received packet. The measurement may be implemented using:

- Receiver energy detection
- Signal-to-noise ratio estimation

2.3.2 IEEE 802.15.4 MAC layer

The MAC layer is in charge of the following tasks:

- Generating network beaconing for devices acting as PAN coordinators
- Synchronizing with the beacons
- Supporting PAN association and dissociation
- Supporting device security
- Using the CSMA-CA for channel access
- Managing the GTS mechanism
- Providing a reliable link between peer-to-peer MAC entities.

IEEE 802.15.4 MAC data transfer

The IEEE 802.15.4 MAC protocol supports two data transfer models that can be selected by the PAN coordinator:

- The non beacon-enabled mode, in which the MAC is simply ruled by unslotted CSMA-CA.
- The beacon enabled mode in which beacons are periodically sent by the coordinator to synchronize the associated nodes and identify the PAN. Access to the channel is ruled by slotted CSMA-CA using the superframe structure.

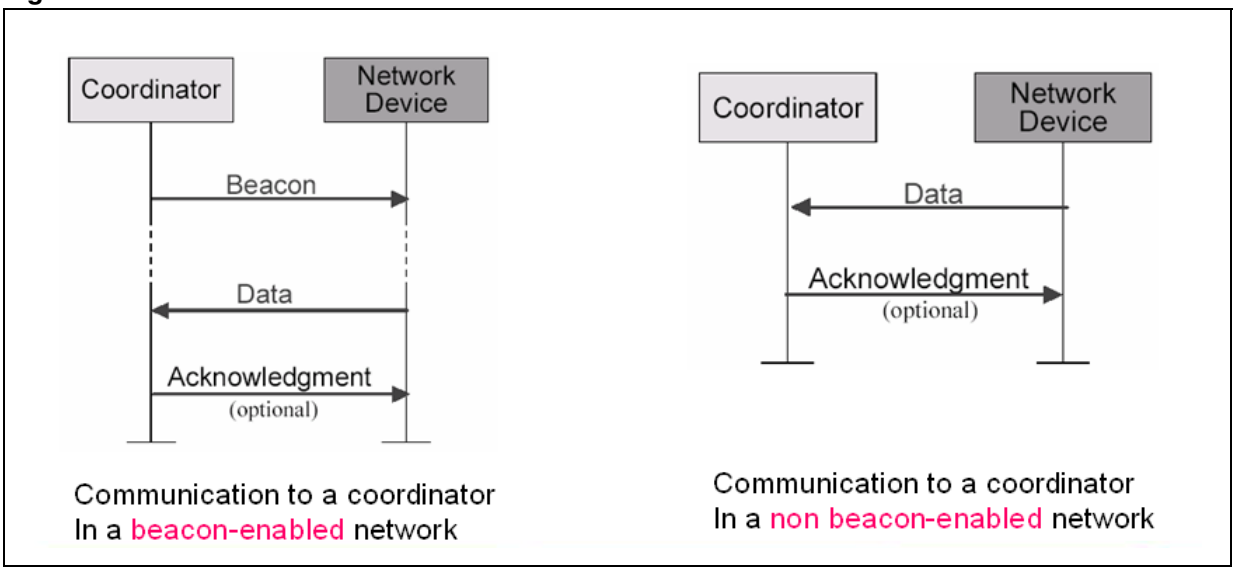
For detailed information about the IEEE 802.15.4 slotted/unslotted CSMA-CA and the superframe structure, refer to the related IEEE 802.15.4 standard specification.

IEEE 802.15.4 MAC device-to-coordinator data transfer

In beacon-enabled networks, the devices search for the beacon to synchronize with the superframe structure. They then transmit data using slotted CSMA-CA.

In non beacon-enabled networks, the devices simply transmit data using unslotted CSMA-CA.

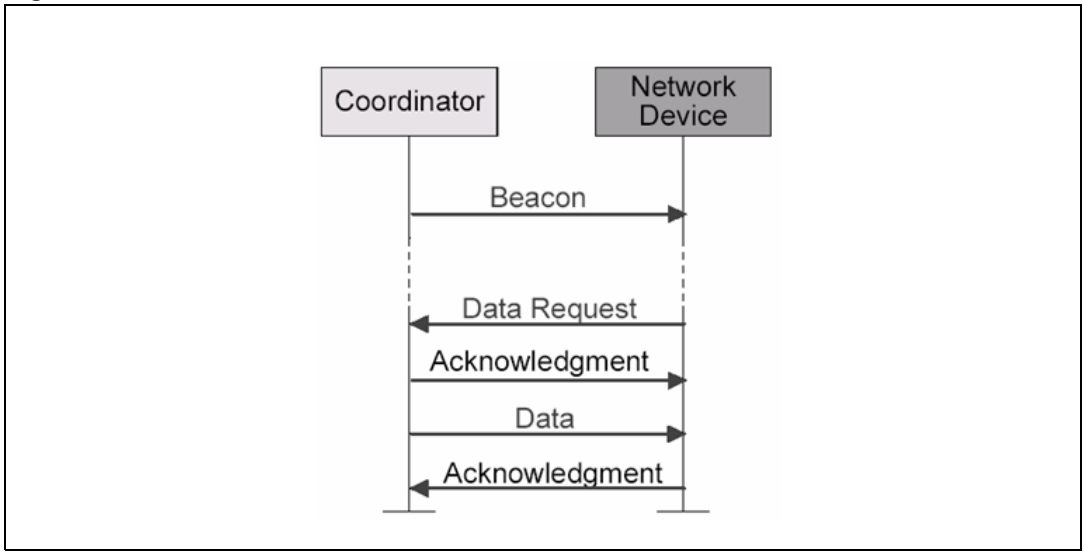
Figure 5. Device-to-coordinator communications



IEEE 802.15.4 MAC coordinator-to-device data transfer

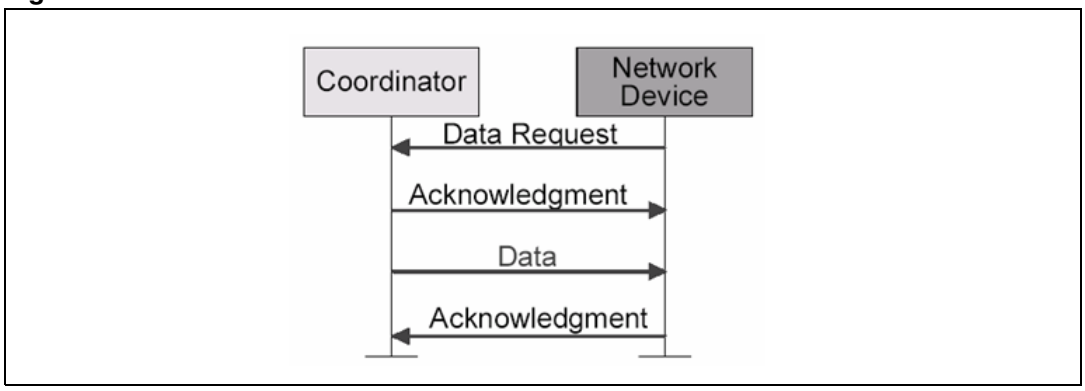
In beacon-enabled networks, the coordinator indicates in the beacon that data are pending. The device periodically listens to the beacons and transmits a data request MAC command using slotted CSMA-CA if necessary.

Figure 6. Coordinator-to-device communications in beacon-enabled networks



In non beacon-enabled networks, devices transmit a data request MAC command using unslotted CSMA-CA. If a coordinator data transmission is pending, the coordinator transmits a data frame using unslotted CSMA-CA. Otherwise, the coordinator transmits a data frame containing a zero-length payload. Refer to *Section : IEEE 802.15.4 general MAC frame format* for a description of the MAC frame.

Figure 7. Coordinator-to-device communications in non beacon-enabled networks



IEEE 802.15.4 general MAC frame format

The MAC frame is composed of a MAC header (MHR), a MAC payload, and a MAC footer (MFR). The MHR is composed of fixed fields. The address fields are optional.

Table 2. General MAC frame format

2 bytes	1 bytes	0/2 bytes	0/2/8 bytes	0/2 bytes	0/2/8 bytes	Variable	2 bytes
Frame control field (FCF)	Sequence number	Destination PAN identifier	Destination address	Source PAN identifier	Source address	Frame payload	FCS
		Addressing fields					
MHR						MAC payload	MFR

The MAC frame control field (FCF) has the following structure:

15 14	13 12	11 10	9 8 7	6	5	4	3	2 1 0
Source addressing mode	Reserved	Destination addressing mode	Reserved	Intra-PAN	Ack. request	MAC frame pending	Security enabled	MAC frame type

Bits [15:14] **Source addressing mode**

00: PAN identifier and address field are not present.
01: reserved
10: Address field contains a 16 bit short address.
11: Address field contains a 64 bit extended address.

Bits [12:13] Reserved

Bits [11:10] **Destination addressing mode**

00: PAN identifier and address field are not present.
01: reserved
10: Address field contains a 16 bit short address.
11: Address field contains a 64 bit extended address.

Bits [7:9] Reserved

Bit 6 **Intra-PAN**
1: frame to be sent within same PAN
0: frame to be sent to another PAN

Bit 5 **Acknowledge request**
1: device sends an acknowledgment frame when it receives a valid frame
0: device does not send an acknowledgment frame when it receives a valid frame

Bit 4 **MAC frame pending**
1: sender device has additional data to send to the receiver
0: sender device does not have any more data for the receiver

Bit 3 **Security enabled**
1: frame is cryptographically protected by the MAC layer
0: frame is not cryptographically protected by the MAC layer

Bits [2:0] **MAC frame type**
000: Beacon
001: Data
010: Ack
011: MAC command
100: reserved
111: reserved

IEEE 802.15.4 PHY frame format

Table 3 shows the PHY frame structure.

Table 3. PHY frame format

4 bytes	1 byte	1 byte		Variable
Preamble	SFD	Frame length (7 bits)	Reserved (1 bit)	PHY payload field (PSDU)
SHR		PHR		PHY payload

- The preamble field is used by the transceiver to perform chip and symbol synchronization with an incoming message. The preamble is composed of 32 binary zeros.
- The SFD field (start-of-frame delimiter) is an 8-bit field indicating the end of the synchronization preamble and the start of the packet data. The SFD must be equal to 10100111b.
- The frame length field is 7-bit long. It specifies the total number of bytes contained in the PHY payload field (PSDU). Its value ranges from 0 to 127.
- The PHY payload field (PSDU) contains the MAC frame.

For more details about the PHY and MAC frames, refer to the IEEE 802.15.4 standard specification.

3 STM32W108 Simple MAC library overview

This section describes the STM32W108 Simple MAC library features:

- IEEE 802.15.4 PHY features supported by the library
- IEEE 802.15.4 MAC features supported by the library
- Simple MAC library APIs naming conventions and classes.

3.1 IEEE 802.15.4 PHY layer supported features

The Simple MAC library supports the following IEEE 802.15.4 PHY features:

- Radio channel selection on 2.4 GHz band
- Radio calibration
- Transmission power control
- Boost mode control
- Selection of alternate transmission path for external power amplifier
- Radio sleep and wakeup control
- Time stamp of received and transmitted packets
- LQI and RSSI for received packets
- Transmit single carrier frequency (diagnostic function)
- Transmit continuous stream of random symbols (diagnostic function)
- Automatic seeding pseudo random number generator using hardware random number source

3.2 IEEE 802.15.4 MAC supported features

The Simple MAC library supports the following IEEE 802.15.4 MAC features:

- Transmit functionalities
  - Unslotted CSMA transmit support including CCA
  - Backoff periods determined by pseudo random number generator
  - CRC generation and CRC data insertion into packets
  - Automatic reception and verification of acknowledgement
- Receive functionalities
  - Packet reception with hardware filtering, correlator error and CRC checking
  - Ability to set node addresses and PAN identifier (for receive filtering only)
  - Hardware filters fully exposed
  - Automatic transmission of acknowledgement with software control over frame pending indication
  - Promiscuous mode
  - Ability to enable/disable receivers

3.3 Simple MAC library API naming conventions

The following naming conventions are used:

- **General prefix**

  All Simple MAC APIs are prefixed with “ST_” followed by the general API family (e.g. Radio, AES).

- **Callback suffix**

  The functions which are implemented in the application and called from the Simple MAC library are suffixed with “Callback”.

- **ISR callback suffix**

  The functions which are implemented in the application and called from Simple MAC library in interrupt context are suffixed with “IsrCallback”.

- **ISR suffix**

  The functions which are implemented in the Simple MAC library and must be called by the application in response to hardware events are suffixed with “Isr”.

3.4 Simple MAC Library APIs classes

The following API classes are supported:

- Radio power state control APIs which control the overall radio initialization and power state.
- Radio channel APIs which control channel selection and calibration.
- Radio transmit APIs which control the transmission of packets.
- Radio receive APIs which control the reception of packets.
- Radio cryptography APIs which provide an interface to the hardware AES coprocessor.
- Radio MAC timer APIs to interface with the MAC timer.
- Radio miscellaneous APIs which perform MAC diagnostic and configuration.

For a detailed description of the Simple MAC library APIs, refer to the STM32W108 Simple MAC Library APIs documentation.

4 STM32W108 Simple MAC demonstration applications

Four Simple MAC demonstration applications are delivered within the Simple MAC installer file which is available from http://www.st.com/mcu in the STM32W section:

- Simple MAC sample demonstration applications (sun and planet roles)
- Simple MAC talk demonstration application
- Simple MAC OTA bootloader demonstration application
- Simple MAC nodetest application

The demonstration applications designed to run on the MB851 board could require a serial communication interface.

To set a serial communication channel for the MB851 board, follow these steps:

1. Fit the MB851 jumper P2 into position 5-6 (power via USB).
2. Connect a mini USB cable to the MB851 mini USB connector and to a PC USB port.
3. Right click **My Computer**, select **Manage**, then **Device Manager**, and open **Ports** (COM and LPT) to display the related USB COMx port
4. Open an hyperterminal on the corresponding USB virtual COMx port with the following configuration:
   - Bit rate: 115200
   - Data bits: 8
   - Parity: None
   - Stop bits: 1
   - Flow control: None.

4.1 Simple MAC sample demonstration application

The Simple MAC sample demonstration application allows creating a basic star topology network based on the Simple MAC library. It supports parent and child roles called sun and planet, respectively.

The following top level commands are supported:

Table 4. Top level command versus hyperterminal characters

Hyperterminal character	Command
?	Display this help menu (sun and planet)
c	Clear indirect transmit queue (sun only)
f	Form a network (sun only)
i	Display status information (sun and planet)
j	Join a network (planet only)
l	Leave a network (sun and planet)
o	Enter OTA bootloader mode (sun and planet)
p	Poll for data (planet only)
r	Adjust send/poll rates (sun and planet)

Table 4. Top level command versus hyperterminal characters

Hyperterminal character	Command
s	Send data (sun and planet)
t	Display the planet table (sun only)
u	Enter UART bootloader mode (sun and planet)

A planet node can also join to the network by pushing the button S1 on the MB851 board (LED D3 turns on).

For a detailed description of the Simple MAC sample application features, roles and commands refer to the STM32W108 Simple MAC demonstration applications documentation.

4.1.1 Building and downloading the Simple MAC sample demonstration applications

Two Simple MAC sample demonstration applications are available, one for the sun role and one for the planet role.

The Simple MAC sample demonstration applications run on the STM32W-SK and STM32W-EXT starter and extension kits:

- STM32-Primer2 with MB850 board already preprogrammed with the sample application (sun image).
- MB851 board to be programmed with a sample planet binary image (*sample_planet.s37*).

The following sample application scenarios are supported:

- You can build a sun-planet network by using the STM32-Primer2 and the MB850 sun application in combination with the MB851 programmed with the *sample_planet.s37* image. The sun-planet network can be further extended by loading the *sample_planet.s37* image in the four MB851 boards, and configuring each of them as a planet by joining to the network formed by the sun.
- Alternatively, you can configure one of the MB851 boards as a sun (using the *sample_sun.s37* prebuilt image) and the other boards as planets.

*Note: For information about the STM32W108 kits refer to user manual UM0894 "*STM32W-SK and STM32W-EXT starter and extension kits for STM32W108xx", *available from http://www.st.com/mcu.*

Using the prebuilt sample binary images

To launch the preprogrammed sample application (sun role) on the STM32-Primer2 and MB850 board, follow these steps:

1. Power on the STM32-Primer2.
2. Press the joystick button.
3. Select **SM SUN** from the application menu by moving the joystick down and pressing again the button.

To download and run the prebuilt sample binary image on the MB851 board, launch the stm32w_flasher utility with the prebuilt *sample_planet.s37* binary file.

For information on how to use the stm32w_flasher utility, refer to user manual UM0894 "*STM32W-SK and STM32W-EXT starter and extension kits for STM32W108xx".*

Using the sample IAR project

An IAR workspace is also provided for the Simple MAC sample application. It supports the MB851 board only.

Follow these steps to build, download and run the Simple MAC sample application on a MB851 board:

1. Power on the MB851 board through the JTAG interface: P2 jumper fitted into position 5-6.
2. Connect the JTAG Flash programmer to MB851 P4 connector and to a PC through an USB cable.
3. Open the IAR toolset.
4. From the **File**, **Open**, **Workspace** menu, open the *sample.eww* IAR project and the workspace related to the sun or planet role.
5. From the **Project** menu, select **Rebuild All**. A binary file is built in the specific demonstration application directory under the chosen installation path.
6. From the **Project** menu, select **Download and Debug**. The related binary image is downloaded into the STM32W108xx Flash memory.
7. Connect the MB851 board to the PC USB port using a USB cable connected to the board mini-USB connector. Open a hyperterminal on the corresponding USB virtual COMx port with the configuration previously described. You can now run the demonstration application.

Note: For each application type (sun, planet), two workspaces are provided:

1. *MB851_SUN and MB851_PLANET used to build a binary image with no IAP bootloader support.*
2. *MB851_SUN-btl and MB851_PLANET-btl used to build a binary image supporting the IAP bootloader (file iap_bootloader.s37 in the prebuilt folder).*

4.1.2 Setting a basic star topology network using the sample demonstration application

This section describes the steps to run sun and planet applications.

Running the sample demonstration application on the STM32-Primer2 with MB850 board

The STM32-Primer2 with MB850 sun application supports some of the sample demonstration events and input commands described in *Section 4.1* through the STM32-Primer2 interface resources (LCD, joystick with button, touch screen).

These events and commands are mapped to graphic events displayed on the STM32-Primer2 LCD as shown below.

Figure 8. Star network created by sun

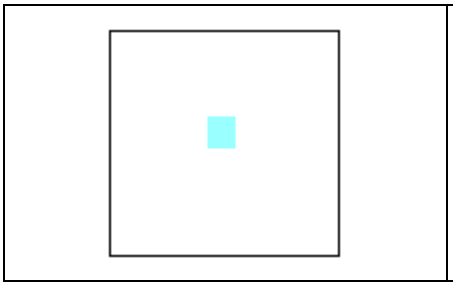


Figure 9. Planet associated to sun

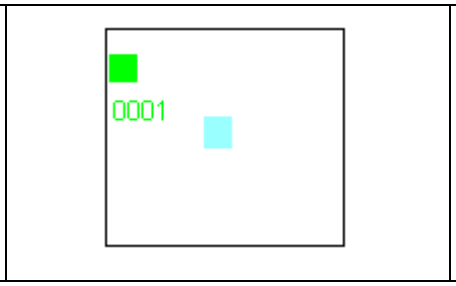


Figure 10. Data sent from planet to sun

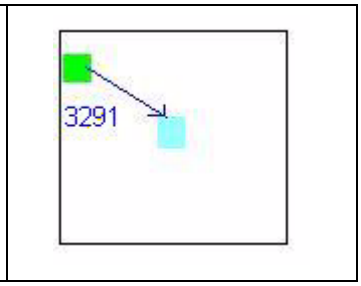


Figure 11. Sun plus 5 planets

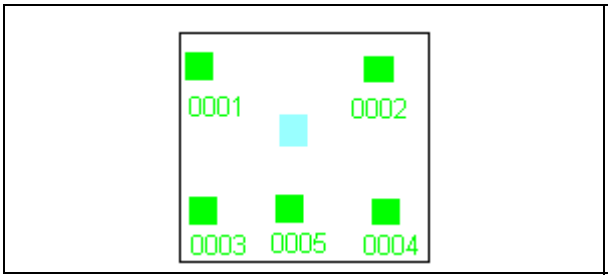
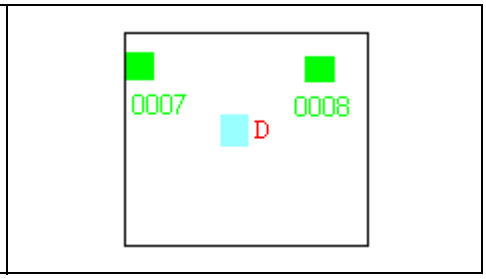


Figure 12. Network down



The scenario shown in *Figure 8* is obtained by selecting **SM SUN** from the LCD menu. The input commands listed in *Table 5* can then be issued by selecting the corresponding item from the related LCD menu:

Table 5. STM32-Primer2 SM SUN menu versus input commands

Menu items	Description
Planet table	Displays the list of planets
Leave network	Quits the network
Sun infos	Displays the sun node status and information.

Note: 1 The STM32-Primer2 with MB850 sun application also supports an interface communication channel through a virtual USB COM. Connect a mini USB cable to the bottom-right side of the STM32-Primer2 and to a PC USB port, and configure the related hyperterminal to 115 200 bps, 8-bits, no parity and flow control, and one stop bit.

2 The STM32-Primer2 sun application displays the V_{DD} values (in mV) received from each MB851 planet.

3 The Send data command is not supported by the STM32-Primer2 LCD menu.

4 The STM32-Primer2 sun application does not display data polled from a planet.

Running a planet application on the MB851 board

Press the button S1 on the MB851 board to join the network formed by the sun node.

To access the full set of commands and node information, you can open an hyperterminal connected to the planet node.

Note: When interacting with the STM32-Primer2 sun application, it is recommended to keep the planet sending rate to a value not lower than the default value.

4.2 Simple MAC talk demonstration application

The Simple MAC talk demonstration application is a simple chat program that demonstrate point-to-point IEEE 802.15.4 wireless communications using the STM32W108xx microcontroller.

4.2.1 Building and downloading the Simple MAC talk demonstration application

The Simple MAC talk demonstration application runs only on the MB851 board.

Use two MB851 boards and program each of them with the talk binary image (*talk.s37*).

Note: The STM32-Primer2 and the MB850 board do not support the talk demonstration application.

Using the prebuilt talk binary image

To download and run the prebuilt talk binary image on the MB851 board, use the stm32w_flasher utility with the prebuilt *talk.s37* binary file. For information on how to use the stm32w_flasher utility, refer to user manual UM0894 "*STM32W-SK and STM32W-EXT starter and extension kits for STM32W108xx*".

Using the talk IAR project

An IAR workspace is also provided for the Simple MAC talk demonstration application.

Follow these steps to build, download and run a talk demonstration application on the MB851 board:

1. Power on the MB851 board through the JTAG: P2 jumper fitted into position 5-6.
2. Connect the JTAG Flash programmer to MB851 P4 connector and to a PC through an USB cable.
3. Open the IAR toolset.
4. From the **File**, **Open**, **Workspace** menu, open the *talk.eww* IAR project and the workspace related to the MB851 board.
5. From the **Project** menu, select **Rebuild All**. A binary file is built in the specific demonstration application directory under the chosen installation path.
6. From the **Project** menu, select **Download and Debug**. The related binary image is downloaded into the STM32W108xx Flash memory.
7. Connect the MB851 board to the PC USB port using a USB cable connected to the board mini-USB connector. Open a hyperterminal on the corresponding USB virtual COMx port with the configuration previously described. You can now run the demonstration application.

Note: For the talk application, two workspaces are provided:

1. *MB851 used to build a binary image with no IAP bootloader support.*
2. *MB851-btl used to build a binary image supporting the IAP bootloader (file iap_bootloader.s37 in the prebuilt folder).*

4.2.2 Setting a “chat communication” using the talk demonstration application

Once configured the serial communication channels of the two MB851 talk boards, the two talk demonstration applications can communicate by typing the “chat text” on the corresponding hyperterminal.

4.3 Simple MAC OTA bootloader demonstration application

The Simple MAC OTA bootloader demonstration application is a simple program that demonstrates the Over the Air upload of a fixed binary image in a STM32W108 device.

4.3.1 Building and downloading the Simple OTA bootloader demonstration application

The Simple MAC OTA bootloader demonstration application runs only on the MB851 board.

Use one MB851 board and program it with the OTA bootloader binary image (bootloader_demo.s37).

Note: The STM32-Primer2 and the MB850 boards do not support the OTA bootloader demonstration application.

Using the prebuilt bootloader binary image

To download and run the prebuilt bootloader binary image on the MB851 board, use the stm32w_flasher utility with the prebuilt bootloader_demo.s37 binary file. For information on how to use the stm32w_flasher utility, refer to user manual UM0894 “STM32W-SK and STM32W-EXT starter and extension kits for STM32W108xx”.

Using the talk IAR project

An IAR workspace is also provided for the Simple MAC bootloader demonstration application.

Follow these steps to build, download and run the bootloader demonstration application on the MB851 board:

1. Power on the MB851 board through the JTAG: P2 jumper fitted into position 5-6.
2. Connect the JTAG Flash programmer to MB851 P4 connector and to a PC using an USB cable.
3. Open the IAR toolset.
4. From the **File**, **Open**, **Workspace** menu, open the bootloader_demo.eww IAR project and the workspace related to the MB851 board.
5. From the **Project** menu, select **Rebuild All**. A binary file is built in the specific demonstration application directory under the selected installation path.
6. From the **Project** menu, select **Download** and **Debug**. The related binary image is downloaded into the STM32W108xx Flash memory.
7. Connect the MB851 board to the PC USB port using a USB cable connected to the board mini-USB connector. Open a hyperterminal on the corresponding USB virtual COMx port with the configuration previously described. You can now run the demonstration application.

4.3.2 Use the bootloader demonstration application for over-the-air upload of a STM32W108 device

Once the serial communication channel on the MB851 bootloader board is configured, type `help` to display the list of supported commands.

For performing the over-the-air upload of a STM32W108 destination device, follow these steps:

1. Power on the related destination MB851 board (powered through the USB or batteries).
2. The destination STM32W108 device is requested to have the IAP bootloader (iap_bootloader.s37). This image is delivered in the prebuilt folder (for downloading the iap_bootloader.s37 image in the destination STM32W108 device, use the stm32w_flasher utility through RS-232 or J-Link interfaces).
3. Put the destination STM32W108 in Bootloader mode. To activate the bootloader on the destination device, press together S1 and RST1 buttons on the MB851 board and then release RST1. The destination board LED D3 turns on for indicating that the device is in Bootloader mode (hardware activation).
4. To launch the OTA upload, type `loadImage`. The destination device is uploaded with a simple test application.

Note: For a detailed description about the bootloader demonstration application commands, please refer to the STM32W108 Simple MAC demonstration applications documentation.

4.4 Simple MAC nodetest application

The Simple MAC nodetest application is a low-level test program meant for the functional testing of RF modules (either your own custom-manufactured devices or those provided in the STM32W108 Kits), including token viewing, range testing, RSSI measurement, and special test modes of transmission as required for FCC and CE certification.

4.4.1 Building and downloading the Simple MAC nodetest application

The Simple MAC nodetest demonstration application runs only on the MB851 board.

Use two MB851 boards and program each of them with the nodetest binary image (simplemac-test.s37).

Note: The STM32-Primer2 and the MB850 board do not support the nodetest demonstration application.

Using the prebuilt simplemac-test.s37 binary image

To download and run the prebuilt nodetest binary image on the MB851 board, use the stm32w_flasher utility with the prebuilt simplemac-test.s37 binary file. For information on how to use the stm32w_flasher utility, refer to user manual UM0894 “STM32W-SK and STM32W-EXTstarter and extension kits for STM32W108xx”.

Using the IAR project

No IAR workspace is provided for the Simple MAC nodetest application. The Simple MAC nodetest is delivered only in binary format.

4.4.2 How use the Simple MAC nodetest application commands

For detailed information on how to use the Simple MAC nodetest, refer to user manual UM0978 “Using the Simple MAC nodetest application”.

5 Designing an application using the Simple MAC Library APIs

This section provides information and code examples on how to design and implement a Simple MAC application.

The following functionalities are described:

- Initialization
- Configuring the radio
- Transmitting packets and managing transmit callbacks
- Enabling/disabling SFD event notifications
- Receiving packets and managing reception callbacks
- Configuring the coordinator filter mode
- Using AES security features
- Miscellaneous: energy detection, packet trace, CCA assessment.

5.1 Initialization

The following steps are required before starting using the Simple MAC library APIs:

1. Initialize the HAL layer.
2. Seed the random number generator.
3. Enable the interrupts.
4. Initialize the serial communication channel.
5. Perform one-time radio initialization and calibration (radio analog module, digital baseband and MAC) and leave the radio in the specified default mode (on or off).

The following pseudocode example illustrates the required initialization steps:

```
/* include hal header files */
#include "hal/hal.h"

/* include library header file */
#include "include/phy-library.h"

void main(void)
{
  u32 seed;
   StStatus status;
   /* Init hal */
  halInit();
  /* seed random number generator */
  ST_RadioGetRandomNumbers((u16 *)&seed, 2);
  halCommonSeedRandom(seed);
  /* init serial */
  uartInit(115200, 8, PARITY_NONE, 1);
  /* enable interrupts */
  INTERRUPTS_ON();
```

```
   /* Init radio in powered up mode */
   assert(ST_RadioInit(ST_RADIO_POWER_MODE_RX_ON)==ST_SUCCESS);
   while(1)
   {
   /* Simple MAC application specific steps */
   }
}
```

5.2 Configuring the radio

5.2.1 Radio sleep and wakeup

Call `ST_RadioSleep` and `ST_RadioWake` APIs to turn the radio off and on, respectively:

```
/* Turning off the radio */
ST_RadioSleep();

 /* Turning on the radio */
ST_RadioWake();
```

5.2.2 Calibrating the radio

The radio needs to be recalibrated to ensure the same performance as environmental conditions changes (temperature, etc.).

The following pseudocode example illustrates the required calibration steps:

```
void main(void)
{
   /* Initialization steps */
   ...
   ...
   while(1)
   {
      /* Periodically check if radio calibration conditions
         occur (due to temperature change)*/
      if (ST_RadioCheckRadio() == TRUE)
      {
         /* Perform necessary recalibration to counteract the
            effects of temperature changes since the last
            calibration */
        ST_RadioCalibrateCurrentChannel();
      }
   }
}
```

5.2.3 Setting radio channel, power level and power mode

Before starting transmitting or receiving a packet, select the radio channel and configure the transmit power and transmit power mode.

Pseudocode example for setting the radio channel

```
u8 channel = USER_CHANNEL;
/* Set radio channel */
ST_RadioSetChannel(channel);
```

Default radio channel is 11. The radio channel ranges between 11 and 26. The first time a channel is selected, all radio parameters are calibrated for this channel. This full calibration process can take up to 200 ms. Subsequent calls to *ST_RadioSetChannel()*, with the same channel takes less time (around 10 ms) because the values are retrieved from the Flash memory tokens.

Pseudocode example for setting the radio transmit power

```
s8 power = USER_TX_POWER;

/* Set radio transmit power level */
ST_RadioSetPower(power);
```

Default transmit power level is 3 dBm. The radio power ranges from -43 to 3 dBm. The *ST_RadioSetPower()* function can set the power level to a value other that the one specified in the power parameter since not all integer power levels are available as lower power levels. When a specific power level is not available, the next higher power level is used.

Pseudocode example for setting the radio power mode

```
u16 txPowerMode = USER_TX_POWER_MODE;
/* Set radio transmit power mode */
ST_RadioSetPowerMode(txPowerMode);
```

The *txPowerMode* parameter can take the following values:

bit 0

0: Normal mode.

1: Boost mode. Selecting the Boost mode increases Rx sensitivity by approximately 1 dBm and Tx power by approximately 0.5 dBm.

bit 1

0: Enables bidirectional transmit path

1: Enables alternate transmit when using an external power amplifier.

5.3 Transmitting packets and managing transmit callbacks

5.3.1 Configuring the radioTransmitConfig variable

Before transmitting a packet, declare a variable, radioTransmitConfig, of *RadioTransmitConfig* type. The *RadioTransmitConfig* type corresponds to a structure containing the parameters and modes related to the packet transmission features (see *Table 6*). It must be initialized prior to calling the packet transmit API.

Table 6. *RadioTransmitConfig* members

Member	Description
boolean waitForAck	Wait for ACK if ACK request set in FCF.
boolean checkCca	Backoff and check CCA before transmitting the packet.
u8 ccaAttemptMax	Number of CCA attempts before failure. The value ranges from 0 to 5. The default value is 4.
u8 backoffExponentMin	Backoff exponent for the initial CCA attempt.The value ranges from 0 to 3. The default value is 3.
u8 backoffExponentMax	Backoff exponent for the final CCA attempt(s). The default value is 5.
boolean appendCrc	Append CRC to transmitted packets.

If *radioTransmitConfig.checkCca* is TRUE, *ST_RadioTransmit()* performs CSMA-CA backoffs and CCA verifications before transmitting a packet. Otherwise it starts the transmission process immediately. The STM32W108xx supports only CCA mode 1. CSMA-CA reports busy medium if the energy level expressed in dBm exceeds this threshold.

The related *radioTransmitConfig* variable can be modified only when no transmit operation is ongoing.

The pseudocode example below explains how to configure the related *radioTransmitConfig* variable**:**

```
RadioTransmitConfig radioTransmitConfig = {
  TRUE,   // waitForAck;
  TRUE,   // checkCca;
  4,      // ccaAttemptMax;
  3,      // backoffExponentMin;
  5,      // backoffExponentMax;
  TRUE    // appendCrc;
};
```

5.3.2 Setting and transmitting a packet

The following steps are required to transmit a packet:

1. The first field of the packet is the related length. If the *radioTransmitConfig.appendCrc* is TRUE the packet length byte must also take into account the two bytes of CRC. A packet with a two-byte payload is represented in memory as:{0x04, 0x00, 0x01, 0xc0, 0xc1} where 0xc0 and 0xc1 are the CRC bytes generated by hardware.
2. The packet has to be configured according to the MAC frame format described in *Section : IEEE 802.15.4 general MAC frame format* :
   a) Set the packet FCF, the packet source and the destination address mode. Refer to *Table 7* for an example of FCF configuration.

Table 7. Packet FCF configuration as 0x0821

Bits [0:2]	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bits [8:9]	Bits [10:11]	Bits [12:13]	Bits [14:15]
Frame type	Security enabled	Frame pending	Ack. request	Intra-PAN	Reserved		Destination addressing mode	Reserved	Source addressing mode
100	0	0	1	0	0	00	10	00	00
0x21						0x08			

b) Specify the packet PAN identifier as well as its short or long addresses, according to the addressing mode selected on the packet FCF.

The pseudocode below shows an example of configuration and 13-byte packet transmission (plus 2 bytes for the CRC):

```
u8 txPacket[] = {
  0x0f, // packet length including two bytes for the CRC
  0x21, // FCF - data frame type, no security, no frame
                pending, ack request, no intra   PAN
  0x08, // FCF - 16 bits short destination address, no
                source pan id and address
  0x00, // sequence number
  0x12, // destination pan id (lowest byte)
  0x23, // destination pan id  (highest byte)
  0x02, // destination short address/node id (lowest byte)
  0x22, // destination short address/node id (highest byte)
  0x12, // packet data
  0x34, // packet data
  0x56, // packet data
  0x78, // packet data
  0x9A, // packet data
  0xBC  // packet data
      };

St_Status status;

/* Sent the txPacket */
status = ST_RadioTransmit(txPacket);
```

If *radioTransmitConfig.chechCCA equals* TRUE, the CSMA-CA backoff(s) and CCA check(s) are performed using the default energy level (-75 dBm).

Below is a pseudocode example of energy threshold setting for the CCA assessment:

```
s8 ed_cca_value = USER_ED_CCA_VALUE
/* Set the energy level used for CCA assessment */
ST_RadioSetEdCcaThreshold (ed_cca_value);
```

5.3.3 ISR callbacks for packet transmission

If the transmission process has successfully started, the *ST_RadioTransmitCompleteIsrCallback()* is called to indicate the completion status.

If the radio is busy transmitting, the *ST_RadioTransmit()* function returns an error and *ST_RadioTransmitCompleteIsrCallback()* will not be called.

Before sending a new packet, check the *ST_RadioTransmitIsrCompleteCallback()* status parameter to decide which action to perform. As an example, retransmit the packet if no acknowledgment has been received when the sent packet has the FCF ACK request bit set to 1.

The pseudocode below gives an example of the transmit ISR callback:

```
void ST_RadioTransmitCompleteIsrCallback(St_Status status, u32
sfdSentTime, boolean framePending)
{
  switch(status) {
    case ST_SUCCESS:
      break;
    case ST_PHY_TX_CCA_FAIL:
      /* Insert here user specific action */
      break;
    case ST_MAC_NO_ACK_RECEIVED:
      /* Insert here user specific action */
      break;
    case ST_PHY_ACK_RECEIVED:
      if (framePending) {
        /* Insert here user specific action */
      }
      else {
        /* Insert here user specific action */
      }
      break;
    default:
        /* Insert here user specific action */
      break;
  }
}
```

Note: The "framePending" parameter is TRUE if the received ACK indicates that a frame is pending. This is a very important information for notify the sender node that the destination node has pending data for it.

5.3.4 SFD event

It is possible to be notified of an SFD event (see *Section : IEEE 802.15.4 general MAC frame format*) through the *ST_RadioSfdSentIsrCallback (u32 sfdSentTime)* callback.

The SFD event notification is disabled by default. To enable it, call the *ST_RadioEnableSfdSentNotification* function:

```
/* Enable SFD event notification */
ST_RadioEnableSfdSentNotification(TRUE);
```

Once enabled, SFD event notifications is sent through the related callback:

```
void ST_RadioSfdSentIsrCallback(u32 sfdSentTime)
{
    /* user code */
}
```

5.4 Receiving packets

5.4.1 Configuring radio filters for packet reception

To receive a packet, the radio device filters must have been configured. The following steps are required to configure radio filters:

1. Set the radio filtering mode according to the user application targets:

```
/* Set promiscuous mode: receive any packet on the selected radio
channel */
/* Disable address filtering*/
ST_RadioEnableAddressFiltering(FALSE);
/* Turn off automatic acknowledgment */
ST_RadioEnableAutoAck(FALSE);

or

/* Receive packets only on a specific pan id and long or short
address (default configuration )*/
ST_RadioEnableAddressFiltering(TRUE);
```

2. Enable/disable the automatic transmission of an acknowledgment on packet reception (only for packets with FCF acknowledge request bit set to 1). Address filtering must be enabled for the automatic transmission of acknowledgment packet to occur:

```
/* Enable automatic packet acknowledgement (default is enabled) */
ST_RadioEnableAutoAck(TRUE);
```

3. Enable/disable the discarding of received packets for which CRC verification has failed. When this feature is enabled, the library automatically removes the CRC bytes from the packets that passed CRC verification:

```
/* Enable discarding packets which fail CRC (default is enabled) */
ST_RadioEnableReceiveCrc(TRUE);
```

4. When the address filtering mode is enabled, perform the following actions
   a) Set the radio PAN identifier:

```
u16 panid = USER_PAN_ID;
/* set the panid for filtering received packets */
ST_RadioSetPanId(panid);
```

   b) Set the node identifier if the user application requires filtering on the 16-bit short address (or node identifier):

```
u16 nodeid = USER_NODE_ID;
```

```
/*set the nodeid (short address) for filtering received packets*/
ST_RadioSetNodeID(nodeid);
```

The *ST_RadioDataPendingforLongIdIsrCallback* and *ST_RadioDataPendingforShortIdIsrCallback* callbacks are called by the library when the packet source short or long address has been received. The library sets the frame pending bit in the outgoing acknowledgment only if the related callback returns TRUE. The user callback implementation must check if the receiving node has pending data for the detected sender source. This can be done by searching the source address in a lookup table containing the sender addresses for which there are data pending, and returning TRUE if there are data pending, and FALSE otherwise.

It is critical that these callback functions complete as quickly as possible to ensure that the frame pending bit is set before the acknowledgment is sent back by the library. The sender node will be notified through the transmit callback that the frame pending bit has been received in the acknowledgment frame. The pseudocode below gives an example of the two callbacks:

```
boolean ST_RadioDataPendingShortIdIsrCallback(u16 shortId)
{
/* Verify there are pending data for the sender node:
look for the packet short address in a application table …>
        */
  if <pending data>
    return TRUE;
   else
    return FALSE;
}

boolean ST_RadioDataPendingLongIdIsrCallback(u8* longId)
{
   /* Verify there are pending data for the sender node:
look for the packet long address in an application table …>
        */

   if <pending data>
    return TRUE;
  else
    return FALSE;
}
```

5.4.2 ISR callbacks for packet reception

Each time a packet is received, the *ST_RadioReceiveIsrCallback(packet, ackFramePendingSet, time, errors, rssi)* callback is automatically called by the library.

The pseudocode below gives an example of a simple implementation of the *ST_RadioReceiveIsrCallback()* callback**:**

```
/* buffer where storing the received packet */
u8 rxPacket[128];

/* flag for checking that there's a packet being processed */
boolean packetReceived = FALSE;
```

```
void ST_RadioReceiveIsrCallback(u8 *packet,
                                boolean ackFramePendingSet,
                                u32 time,
                                u16 errors,
                                s8 rssi)
{
   /* note this is executed from interrupt context */
   u8 i;

   /* Copy the packet to a buffer that can be accessed from the main
   loop. Don't do the copy if there is already a packet there being
   processed  (packetReceived   = TRUE) */
   if(packetReceived == FALSE) {
     for(i=0; i<=packet[0]; i++) {
       rxPacket[i] = packet[i];
     }
     packetReceived = TRUE;
   }
}
```

The `rxpacket[]` will be processed depending on the application target (see pseudocode below for an example):

```
void main(void)
{
...
...
 while(1) {
       /* print out any packets that were received */
       if(packetReceived == TRUE) {
       for(i=0; i<=rxPacket[0]; i++) {
         <print rxPacket[i]>;
      }
      /* The packet has been processed, so free the single entry
      queue up */
      packetReceived = FALSE;
    }
  }
}
```

5.5 Configuring the coordinator filter mode

The IEEE 802.15.4 standard supports coordinator filter mode configuration. This feature allows receiving 802.15.4. data frames which have no destination address: the packets sent must have a destination addressing mode set to 00b (destination PAN identifier and destination short address not present). The source PAN identifier must match the coordinator PAN identifier, and the coordinator has to enable address filtering and set the same PAN identifier used for filtering the received packets.

Note: A node which is not a coordinator will not receive these packets.

Call the related APIs to enable the coordinator feature:

```
/* Enable coordinator feature (default is disabled)*/
ST_RadioSetCoordinator(TRUE).
```

Follows a pseudo code example showing a packet configured to be sent to a node with coordinator feature enabled and source pan id 0x2311:

```
u8 txPacket[] = {
  0x08, // packet length including two bytes for the CRC
  0x21, // FCF - data frame type, no security, no frame
                pending, ack request, no intra   PAN
  0xc0, // FCF - no destination address mode (only source pan id)
  0x00, // sequence number
  0x11, // source pan id (lowest byte)
  0x23, // source pan id (highest byte)
  0x71  // packet data
      };
```

The `txpacket[]` FCF field is configured as follows:

Table 8. FCF field of the packet sent to a coordinator node

Bits [0:2]	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bits [8:9]	Bits [10:11]	Bits [12:13]	Bits [14:15]
Frame type	Security enabled	Frame pending	Ack. request	Intra-PAN	Reserved		Destination addressing mode	Reserved	Source addressing mode
100	0	0	1	0	0	00	00	00	11 (or 10)
0x21						0xc0 (or 0x80)			

5.6 Radio AES security

Refer to the pseudocode below for an example of how to encrypt a block of data using an encryption key:

```
/* pointer to 128 bits of key data */
u8 key[128] = USER_KEY;
u8 block[128];

<Set the block of data>
...
/* Set the AES key */
ST_AesSetKey(key);

/* Encrypts the 128 'block' with the 'key' previously configured.
   The resulting encrypted data are stored at the same 'block'
   data.
*/
ST_AesEncrypt(block);
```

5.7 Radio MAC timer

The MAC timer is 20-bit long. Each LSB tick represents 1 µs, and the MAC timer rolls over to zero approximately once every second. The MAC timer starts operating in free running mode when *ST_RadioInit()* is called. To retrieve the MAC timer value, call the *ST_RadioGetMacTimer()* API as shown below:

```
u32 mac_timer ;
/* Returns an instantaneous reading of the free-running MAC timer*/
mac_timer =ST_RadioGetMacTimer();
```

Note: It is possible to enable/disable MAC timer compare event (ST_RadioEnableMacTimerCompare(enable) and to set the related reference compare value (ST_RadioSetMacTimerCompare(compare_value)). The ST_RadioMacTimerCompareIsrCallback() callback will be called by the library when a MAC timer comparison event occurs.

5.8 Other radio features

5.8.1 Radio energy detection

To read the average energy level calculated over the previous eight symbol periods (128 µs), call the *ST_RadioEnergyDetection()* API:

```
s8 energy_level;

/* get the energy level on the selected radio channel */
energy_level  = ST_RadioEnergyDetection()
```

Note: ST_RadioEnergyDetection() returns the chip sensitivity limits (e.g. -97 dBm up to approximately -25 dBm).

5.8.2 Radio CCA

To read the current status (clear or busy) of the selected channel, call the *ST_RadioChannelIsClear* API:

```
boolean channel_status;

/* Get the channel status (clear or busy) */
channel_status = ST_RadioChannelIsClear();
```

5.8.3 Radio packet trace interface (PTI)

The STM32W108xx includes a packet trace interface (PTI) module which performs robust packet-based debugging. The PTI lines (PTI_EN, PTI_DATA on GPIO[4:5]) allows accessing all the received radio packets in a non-intrusive way.

To enable PTI, call the *ST_RadioEnablePacketTrace* API:

```
/* enable packet trace interface (default is enabled) */
ST_RadioEnablePacketTrace(TRUE);
```

To read the current packet trace status (enabled or disabled), call the *ST_RadioPacketTraceEnabled* API, as shown below:

```
boolean packet_trace_status;
packet_trace_status = ST_RadioPacketTraceEnabled();
```

5.8.4 Send a tone or a carrier wave

The Simple MAC library provides some APIs which allow performing demodulated carrier wave (“tone”) transmission or modulated carrier wave transmission on the current channel. These APIs can be used to test scenarios such as transmit power level measurement.

To start/stop demodulated carrier wave (“tone”) transmission, call the following APIs:

```
/* Start sending a tone */
ST_RadioStartTransmitTone()

/* Stop the tone transmission */
ST_RadioStopTransmitTone(void);
```

To start/stop modulated carrier wave transmission, call the following APIs:

```
/* Start sending a carrier wave */
ST_RadioStartTransmitStream()
/* Stop modulated carrier wave transmission */
ST_RadioStopTransmitStream(void);
```

6 References

Table 9. List of references

Title	Content
IEEE 802.15.4 standard specification	IEEE 802.15.4 standard description
STM32W108 Simple MAC library APIs documentation(1)	HTML document describing the Simple MAC library APIs
STM32W108 Simple MAC demonstration applications documentation(1)	HTML document describing the Simple MAC demonstration application features, roles and commands.

1. This HTML file is provided within the Simple MAC library.

7 List of acronyms

Table 10. List of acronyms

Term	Meaning
ACK	Acknowledgment
API	Application programming interfaces
CCA	Clear channel assessment
CSMA-CA	Carrier sense multiple access with collision avoidance
FCF	Frame control field
FCS	Frame check sequence
MAC	Medium access control
MFR	MAC footer
MHR	MAC header
PAN	Personal area network
PHY	Physical layer
PHR	PHY header
PSDU	PHY service data unit
RSSI	Received signal strength indication
SFD	Start-of-frame delimiter
SHR	Synchronization header

8 Revision history

Table 11. Document revision history

Date	Revision	Changes
12-Feb-2010	1	Initial release.
21-Apr-2010	2	Updated API prefixes in *Section 3.3: Simple MAC library API naming conventions*. Updated *Section 4.1: Simple MAC sample demonstration application*. Modified sample image and removed Note 1 in *Section 4.1.1*. *Section 4.1.2*: changed *Figure 9*, *Figure 10*, *Figure 11*, and *Figure 12*; updated *Table 5* and *Running a planet application on the MB851 board*. Removed section 4.1.3 Limitations, as well as Limitations in *Section 4.2.2*. Updated function name and radio power range in *Pseudocode example for setting the radio transmit power*. Updated function name in *Pseudocode example for setting the radio power mode*. Updated structure and variable name in *Section 5.3.1* and *Section 5.3.2*. Function names updated in *Section 5.3.3*, *Section 5.3.4*, *Section 5.4.1*, *Section 5.4.2*, *Section 5.7*, *Section 5.8.3*. Updated whole *Section 5.8.4*.
30-Jul-2010	3	Added *Section 4.3: Simple MAC OTA bootloader demonstration application* and *Section 4.4: Simple MAC nodetest application*. Reviewed initialization steps in *Section 5.1: Initialization*.
25-Aug-2010	4	Reviewed bootloader name as iap_bootloader.s37. Reviewed bootloader_demo running steps.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com