Getting started with STSW-ST8500G3 firmware package

Introduction

This user manual describes the content of STSW-ST8500G3 firmware package.

This package provides an example of application that runs on STM32 that operates ST8500 running G3 power line communication firmware.

The main features of STSW-ST8500G3 firmware package are:

- image download of PE/RTE images towards ST8500 and configuration for G3 PLC operation
- communication between one PAN coordinator and one or several PAN devices through G3 power line communication

1 General information

1.1 Terms and definitions

ADP = ADaPtation layer
GUI = Graphical User Interface
HI = Host Interface
ISR = Interrupt Service Routine
MAC = Medium Access Control layer
MSC = Message Sequence Chart
NVM = Non-Volatile Memory
PAN = Personal Area Network
PE = Protocol Engine (ARM Cortex in ST8500)
PHY = Phisical layer
PLC = Power Line Communication
RTE = Real Time Engine (DSP in ST8500)
RTOS = Real Time Operating System

1.2 References

1. UM2343: EVALKITST8500-1: getting started with ST8500 evaluation kit
2. Host Interface Driver G3 Application Note
3. SmartGrid LabTool_Quick Start Guide Demo.pdf from SmartGrid LabTool help menu

2 Package description

2.1 General description

The STSW-ST8500G3 firmware package provides software components running on the host STM32 MCU. These software components provide APIs to communicate with the ST8500 when it runs its Boot ROM code or the G3 application code.

The STM32 MCU acts as the host controller and it interfaces with ST8500 that implements a PLC modem. The PLC modem firmware is not in the scope of this document.

The following integrated development environment is supported:

- IAR Embedded Workbench® for Arm® (EWARM)

Note: Refer to the release note available in the root folder of the delivery package for information about the IDE versions supported.

2.2 Architecture

2.2.1 Architecture overview

This STM32 project provides a basic application example to operate ST8500 for G3 PLC operation.

It starts by PE and RTE images download. During download phase, ST8500 runs from its Boot ROM and thus STM32 implements APIs that allows to download and start images for both RTE and PE.

After PE and RTE images download, ST8500 runs the G3 software and STM32 implements APIs for G3 PLC operation.

In the STM32 project example, the ST8500 operates G3 PLC in BOOT/ADP mode.

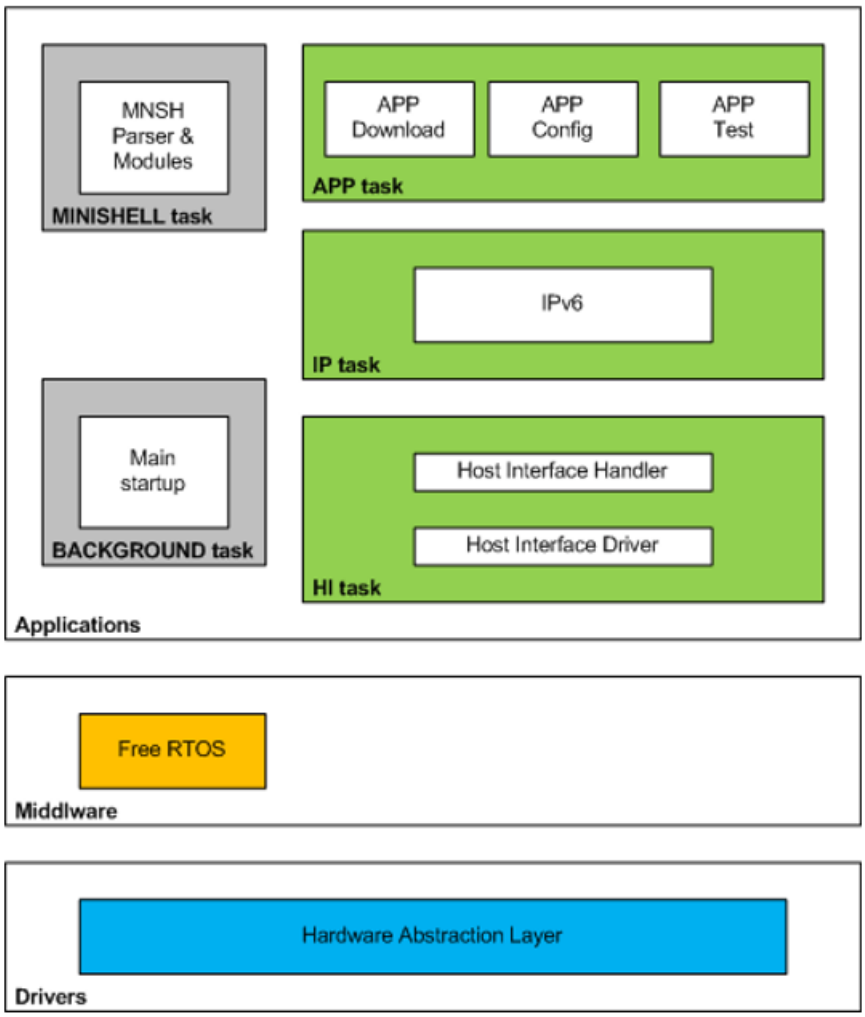
This means that:

- G3 bootstrap procedure is fully handled by ST8500
- for data transfer ST8500 handles PHY, MAC and ADP layers, while STM32 handles IP/UDP and application layers

The STSW-ST8500G3 firmware package includes a driver for all ST8500 host interface APIs. It must be noted that in the application example only a subset of these APIs are used, mainly the APIs related to configuration and data transfer at ADP level.

2.2.2 Static architecture view

Figure 1. **Static architecture view**



Application layer implements mainly 3 features:

- APP Download: RTE and PE images download towards ST8500
- APP Config: ST8500 configuration for G3 operation
- APP Test: Test applications that consist in exchanging UDP messages for
  - PAN device Remote reset from the PAN coordinator
  - Ping from the PAN coordinator or from any PAN device
  - PAN device time set from PAN coordinator
  - PAN device time read from the PAN coordinator
  - PAN device LED set/clear from the PAN coordinator
  - Packet sending from the PAN coordinator or any PAN device

The µIPv6 IPv6 protocol stack implements UDP/IP layers. The corresponding source code comes from open source Contiki solution.

Host Interface layer provides APIs to interface with ST8500 running G3 PLC:

- The Host Interface handler manages request messages going to ST8500 and confirm/indication messages coming from ST8500
- The Host Interface driver provides an extensive list of functions for all ST8500 Host Interface APIs

– Please refer to 2. for details

For the sake of demonstration, a command line parser (MINISHELL) was added to the main and startup functions. This parser allows triggering various tests from the PAN coordinator. This parser is not described in details as it is just there to interface with the SmartGrid LabTool GUI but is not needed for a typical application.

2.2.3 Host Interface buffer management

Request messages are handled the following way:

- Request messages are handled sequentially
- These messages are prepared in TX buffer (hiReqBuf[])
- Once correctly formatted, CRC is computed
- Once complete, the message is sent on the USART TX line
- The TX buffer cannot be used until the current message is fully transmitted

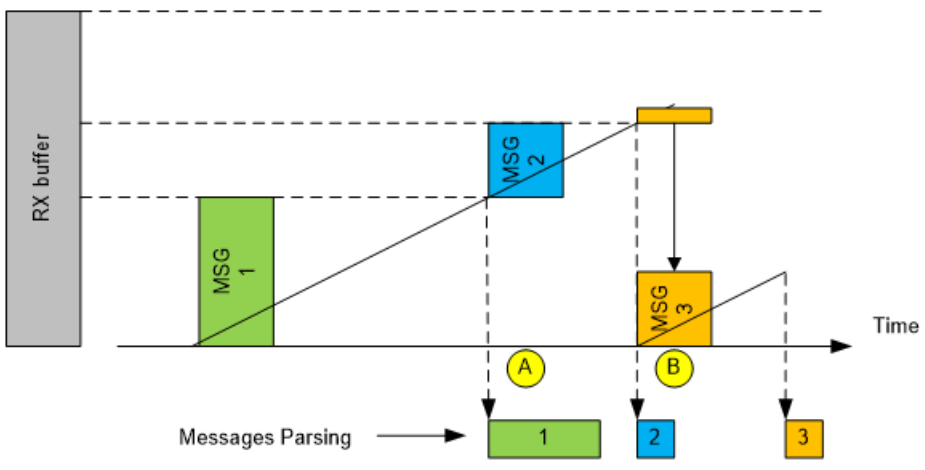
A mechanism is implemented in case of multiple requests coming from the application:

- Up to 2 parallel request messages can be issued towards ST8500 (except 2 data requests)
- The extra requests are memorized and processed later when the TX buffer becomes available.

Confirm/Indication messages are handled the following way:

- These messages come from ST8500 asynchronously and can be received consecutively without any delay
- Time to parse the messages can be more than a character time on the USART RX line
- This is the reason why the RX buffer (hiRespBuf[]) can contain more than a message as shown on the diagram below
  - When RX buffer is empty, the first message is naturally stored at the beginning of the buffer
  - After the reception of the 5 first bytes of a new message (to compute the message size), it is checked that the previously parsed messages have freed enough space at the beginning of the RX buffer to copy this new message to the beginning of the RX buffer
    - This is not the case at time A on diagram below, as parsing of first message is not completed
    - This is the case at time B as third message is small enough to be stored at the place of first and second messages that have already been parsed.

Figure 2. HI RX buffer management



2.2.4 Dynamic architecture view

STSW-ST8500G3 firmware package dynamic behavior is presented in the following message sequence charts (MSC). A few examples are provided:

- FW download to ST8500
- Configuration of ST8500
- Bootstrap for PAN coordinator and PAN device

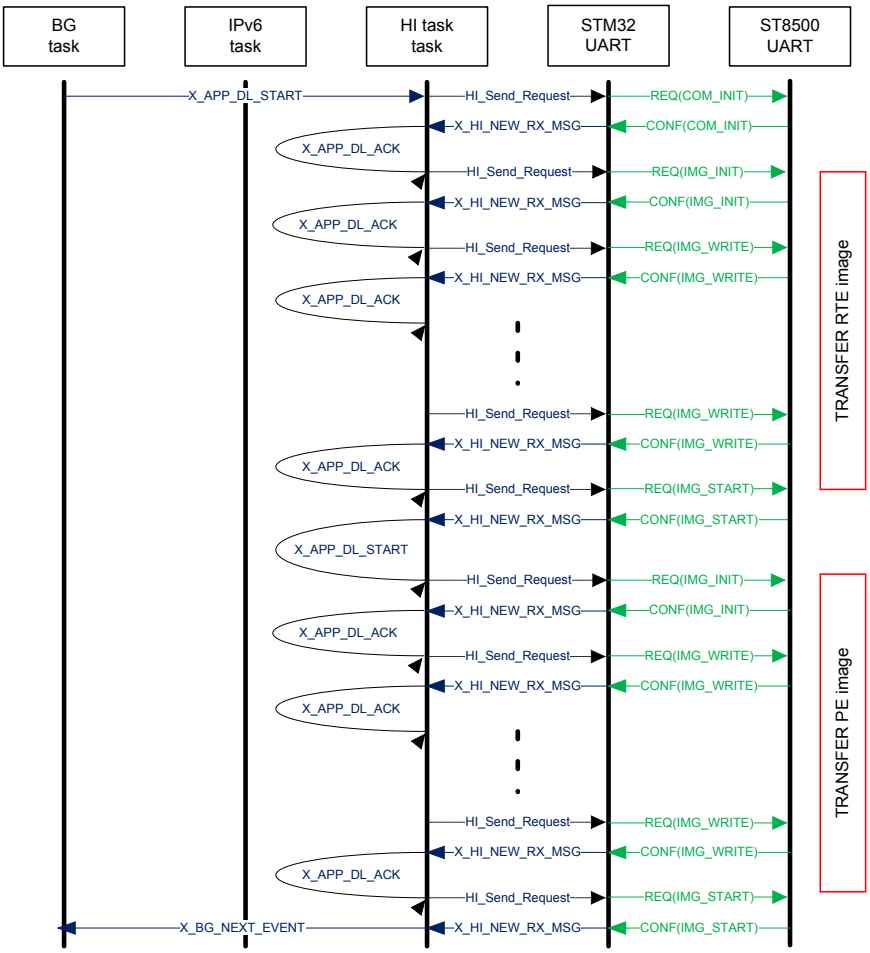
- UDP transfer

In the MSC, the following applies:

- Plain arrows are used to represent RTOS messages
- Dashed arrows are used to represent function call
- Green arrows represent messages on the Host Interface (UART between STM32 and ST8500)

2.2.4.1 *FW Download to ST8500*

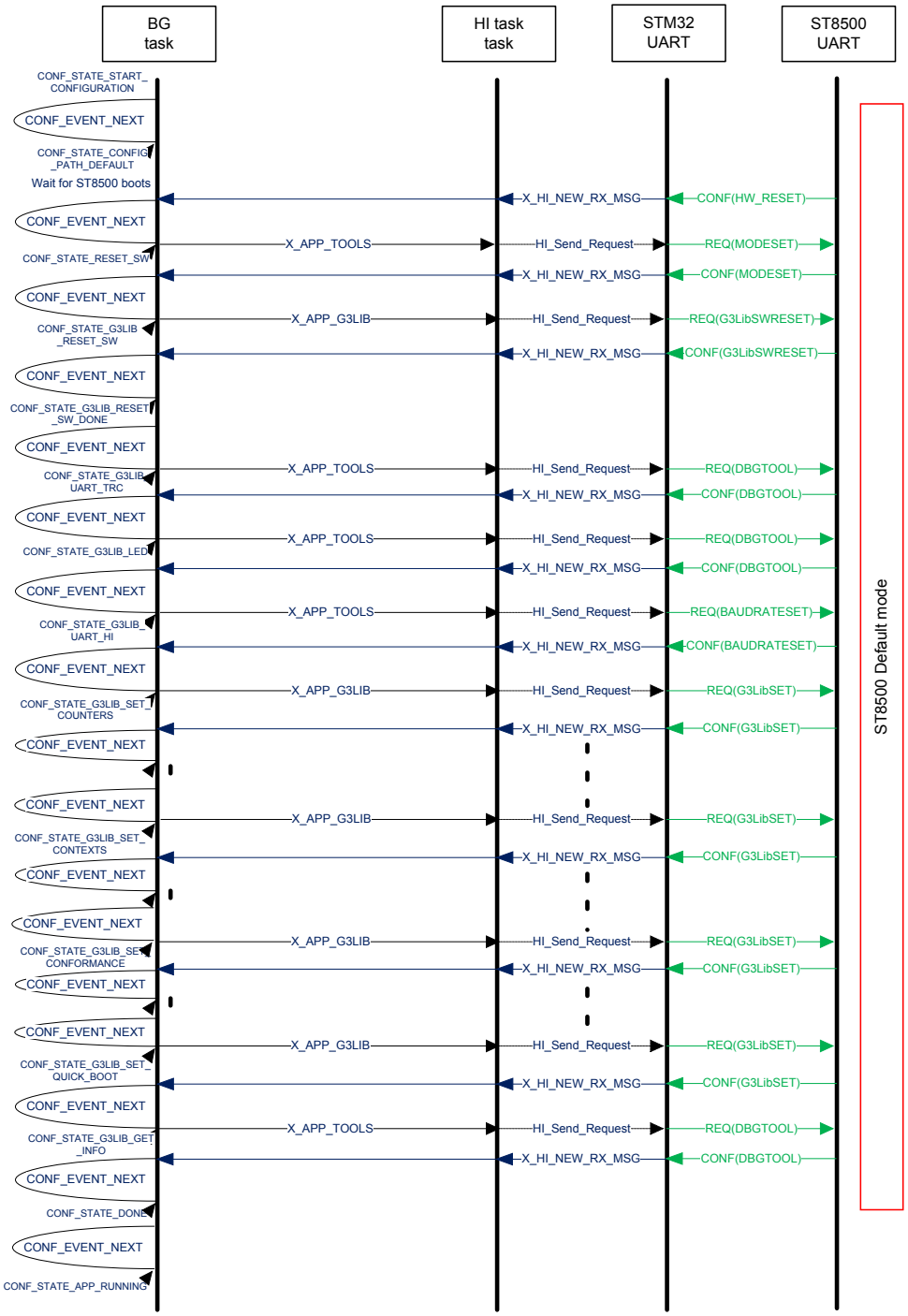
Figure 3. Message sequence for RTE/PE images download



2.2.4.2 *Configuration*

Here is the configuration flow chart

Figure 4. **Message sequence for ST8500 G3 configuration**

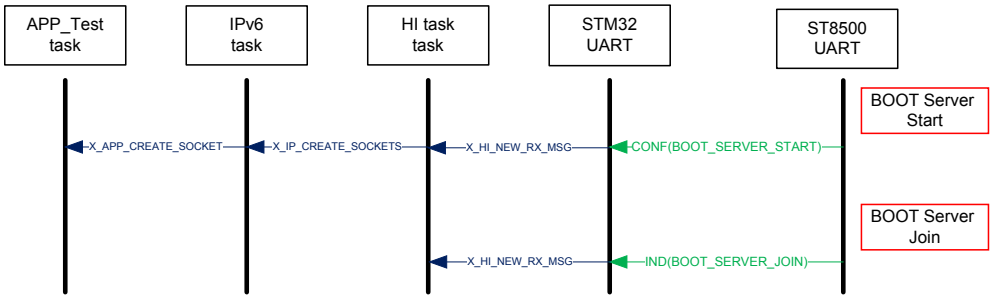


2.2.4.3 *Bootstrap*

At power up, on the PAN Coordinator, the ST8500 issues a BOOT_SERVER_START confirm message to the STM32 in order to assess that a PAN network has been started properly.

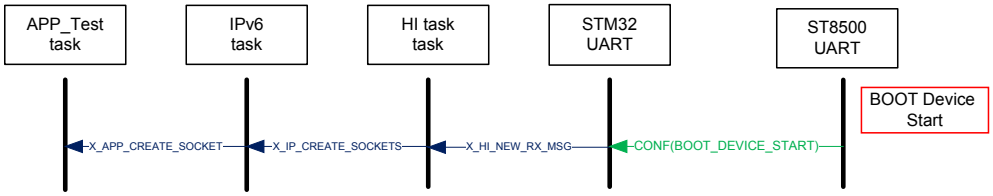
When a joining procedure is successful for any PAN device, a BOOT_SERVER_JOIN message is issued towards the PAN coordinator.

Figure 5. Message sequence for BOOT for PAN Coordinator



At power up, the PAN device automatically starts the joining procedure and, as soon as this is successful, a BOOT_DEVICE_START indication is issued by the ST8500 to the STM32.

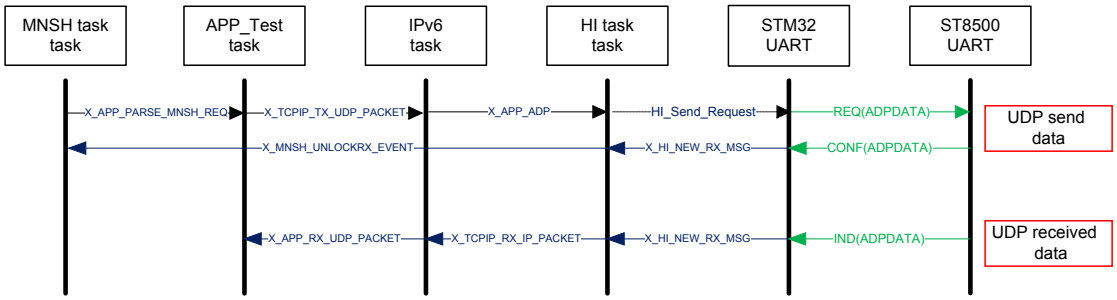
Figure 6. Message sequence for BOOT for PAN Device



2.2.4.4 *UDP Transfer*

The following diagram provides details when sending or receiving UDP packets:

Figure 7. Message sequence for UDP SEND and RECEIVE



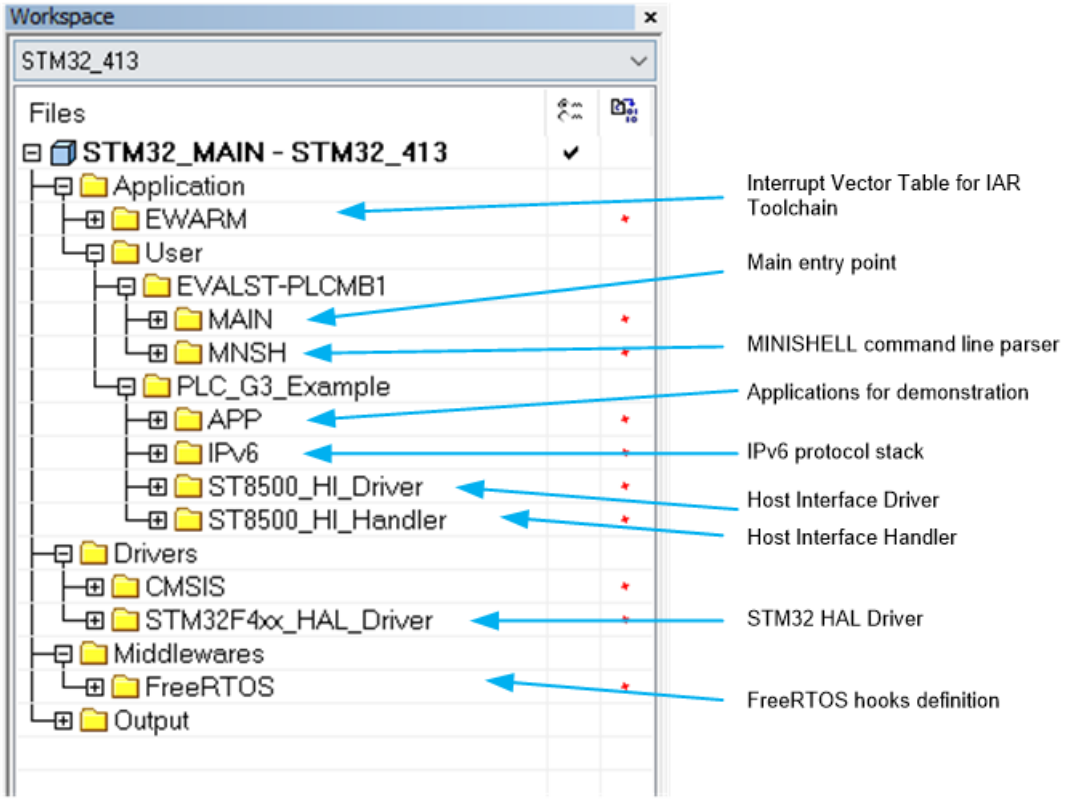
2.2.5 Package description

STSW-ST8500G3 firmware package is composed of:

- STM32CubeHAL
  - The HAL driver layer provides a generic multi-instance simple set of APIs to interact with the upper layers (application, libraries and stacks).
- Middleware
  - FreeRTOS is needed to run the tasks for the application example
- Applications
  - EVALST-PLCMB1 framework
  - PLC_G3_Example
    - Application example running on top of UDP/IP
    - Host Interface handler/driver for ST8500

2.2.6 Folder structure

Figure 8. **Folder structure**



2.2.7 Board related information

For details on board information, please refer to ST8500 1.

2.2.7.1 *Reset button*

The reset push-button is used to reset the board at any time. This action forces the reboot of the evaluation kit including STM32 and ST8500.

2.2.7.2 *External serial flash*

An external serial flash is connected to the STM32.

The sectors 0 and 1 contain the RTE image.

The sectors from 2 to 5 contain the PE image.

The sectors 30 and 31 contain configuration parameters.

2.2.7.3 *Pass through switch*

The switch called “**STM32 mode**” allows passing through the STM32 and interacts to the ST8500 through PLC GUI.

With switch pushed down, the platform is in pass through mode

With switch pulled up, the platform is in default mode.

2.2.7.4 *ST8500 boot switch*

The switch called “**PLC mode**” is used to choose the ST8500 boot (through UART or SPI flash attached to ST8500).

With switch pushed down, the ST8500 boots from its serial flash.

With switch pulled up, the ST8500 boots from UART.

2.2.7.5 *LED*

There are 4 LEDs:

- LED0 is an image of OS tick.
- LED1 is blinking each time the OS enters in Idle task. In case this LED stops blinking, the STM32 is stuck in a higher priority task and shall be reset using the reset push button.
- LED2 is used to show that evaluation kit is in pass through mode.
- LED3 is used to show that evaluation kit is booting and is cleared at the end of the configuration.
- In default mode, after configuration, LED2 and LED3 can be used by application test API. The PAN Coordinator may send a message to PAN device for setting or clearing these LEDs.

2.2.7.6 *RTC*

The STM32 embeds the RTC block to set and get the RTC time and calendar.

3 Service connectivity description

Power-Line Communication (PLC) carries data on a conductor that is simultaneously used for AC electric power distribution to consumers.

This chapter describes the PLC connectivity: from the automatic joining procedure to the data exchanges between a PAN coordinator and one or several PAN devices.

3.1 Automatic bootstrap

At start-up and as long as they have not joined a PAN network, PAN devices discover the network environment on which they are connected. If any PAN coordinator is present on this network, it is responding to beacon requests sent by the PAN devices with the beacon frame. Upon reception of the beacon frame, the PAN device initiates a joining procedure to the PAN coordinator.

During the joining procedure, the PAN coordinator authenticates the PAN device that requests access to the PAN. Then, it provides the PAN device with a short address (2 bytes) and with the encryption key to secure the communication link.

This bootstrap procedure is handled autonomously by the ST8500

When the joining procedure is successful, the PAN device initiates a route establishment between itself and the PAN coordinator it is connected to. Furthermore, the creation and the management of the routes to or from the destination PAN Device or Coordinator is handled autonomously by the ST8500.

3.2 PING example

The PING feature – i.e. the transmission of ICMP request – may be triggered from any PAN Coordinator or any PAN Device that are part of the same PLC network. A PLC network is a set of PAN Devices together with the PAN Coordinator they are connected to.

Upon reception of an ICMP request, any PLC device (PAN coordinator or PAN device) replies with an ICMP reply including the same payload.

The processing linked to the ICMP layer is done on the STM32 side within the µIPv6 protocol stack. Once the ICMP request is built, it is provided to the ST8500 Modem, on the Host Interface, for further processing and transmission on the PLC physical network.

Reversely, any ICMP reply, or request, that is detected and received on the PLC physical network is processed within the ST8500 Modem before it is provided to the STM32, through the Host Interface, to be processed within the µIPv6 protocol stack.

3.3 UDP example

The UDP datagram transmission may be triggered from any PAN Coordinator or PAN Device that is part of the same PLC network.

The processing linked to the UDP layer is done on the STM32 side within the µIPv6 protocol stack. Once the UDP datagram is built, it is provided to the ST8500 Modem, on the Host Interface, for further processing and transmission on the PLC physical network.

At the destination, the ST8500 Modem detects any received UDP datagram before it is provided to the STM32, through the Host Interface, to be processed within the µIPv6 protocol stack.

4 Hardware and software environment setup

For detailed hardware and software configuration please refer to 1. and 3. .

In order to update the STM32 firmware, 2 options are possible:

- Either create a .dfu file and download it to EVALST-PLCMB1 evaluation board through USB DFU connector
  - You can refer to Annex for details

Or in IAR IDE, download through JTAG STM32 interface using ST-LINK/V2 probe

5 Debug

Several debug features are made available in the STSW-ST8500G3 firmware package.
To use these features, it is needed to connect SmartGrid LabTool to the platform.

5.1 Host Interface messages

The MINISHELL command “dbg filt 8” activates the display of the messages passing through the Host Interface (on the UART) between the STM32 Host Controller and the ST8500 Modem. The display of the messages is propagated to the SmartGrid LabTool (refer to 3.).

5.2 Debug messages

The MINISHELL command “dbg hiinfo” activates the display of debug messages within the SmartGrid LabTool (refer to3.). These debug messages are triggered either as generic debug prints or on reception of confirm and indication messages from ST8500.

```
dbg hiinfo <level>
0 → display of debug messages is disabled
1 → display of generic debug prints
2 → same as 1 plus display of type of confirm/indication message
3 → same as 2 plus display of most important parameters from confirm/indication message
4 → same as 2 plus display of all parameters from confirm/indication message
```

6 Annexes

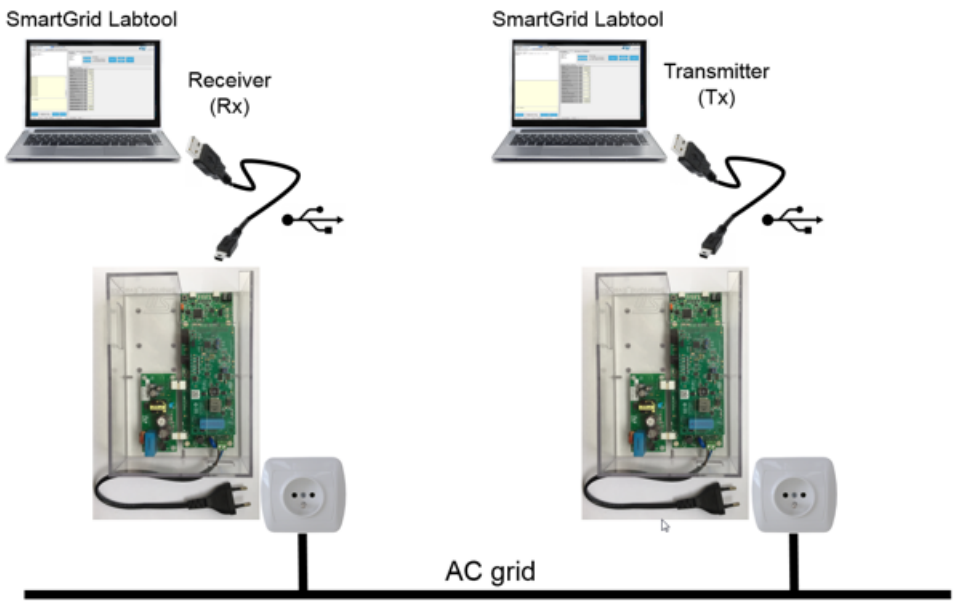
6.1 FAQ

6.1.1 How to check the PLC link?

To perform a PLC link between 2 EVALST-PLCMB1 evaluation boards at PHY level, one will be configured as receiver (Rx) and the other one as transmitter (Tx).

Both will need a PC connected to the USB connector J204. On each PC, start a SmartGrid LabTool session and connect to the correct COM port (please refer to 3. for details).

Figure 9. Basic PHY test setup



Plug both EVALST-PLCMB1 evaluation boards to the AC plug and wait few seconds till the end of startup.

Then to configure the Rx, use the following MNSH command:

- Set the platform in PHY mode

```
plc mode 0
```

- Set the platform in Rx:

```
plc perf 0 0
```

And to configure the Tx, use the following MNSH command as example:

- Set the platform in PHY mode

```
plc mode 0
```

- Configure the Tx to send PHY frames

```
plc perf 0 <nb_packets> <delay> <packet_size> <modulation> <TX_power> <tone_map>

nb_packets: from 0 to 0xFFFE, 0xFFFF for continuous mode
```

```
delay: Inter-Frame Spacing (IFS) in ms
packet_size: nb of bytes
modulation:
    0x0000    drobo
    0x0001    dbpsk
    0x0002    dqpsk
    0x0003    d8psk
    0x0100    crobo
    0x0101    cbpsk
    0x0102    cqpsk
    0x0103    c8psk
TX_power: power in dB, from 0 to 32
tone_map:
          0x3F: default tone-map for CENELEC-A
          0x0F: default tone-map for CENELEC-B
          0xFFFFFF: default tone-map for FCC
```

In the following example 100 frames are transmitted with IFS=200ms, a 120Bytes payload, in ROBO mode, TxPower=32 and full tone map in CENA

```
plc perf 0 100 200 120 0x00 32 0x3f
```

In the following example 100 frames are transmitted with IFS=200ms, a 120Bytes payload, in ROBO mode, TxPower=32 and full tone map in CENB

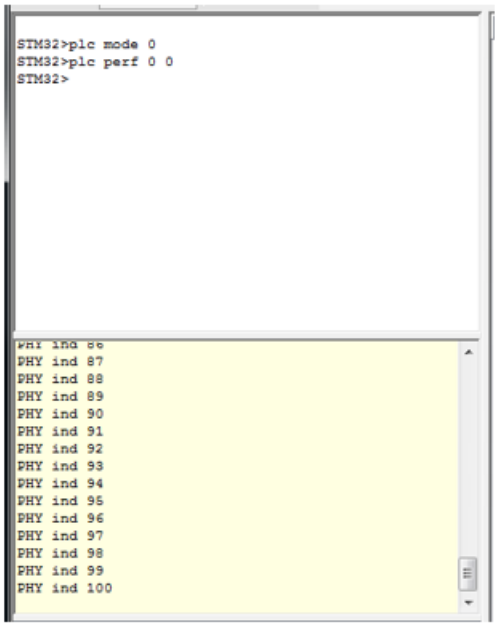
```
plc perf 0 100 200 120 0x00 32 0xf
```

In the following example 100 frames are transmitted with IFS=200ms, a 120Bytes payload, in DBPSK mode, TxPower=32 and full tone map in FCC

```
plc perf 0 100 200 120 0x01 32 0xffffff
```

Then on Rx, PHY indication messages will be printed for each frame received.

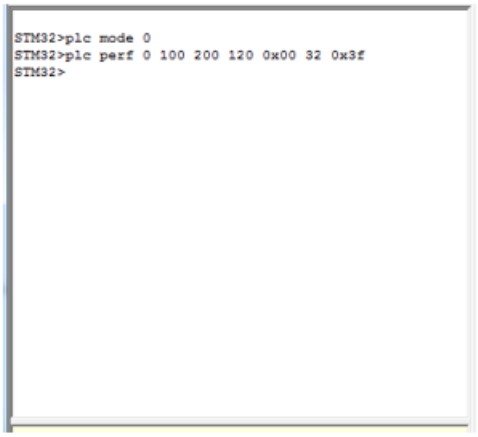
Figure 10. SmartGrid LabTool terminal on Rx – PHY mode



To reset the Rx PHY ind counter before starting a new test, use the following MNSH command:

```
plc perf 0 0
```

Figure 11. **SmartGrid LabTool terminal on Tx – PHY mode**



To stop the Tx test still running, use the same MNSH command:

```
plc perf 0 0
```

To retrieve the statistics of the test, use the MNSH command:

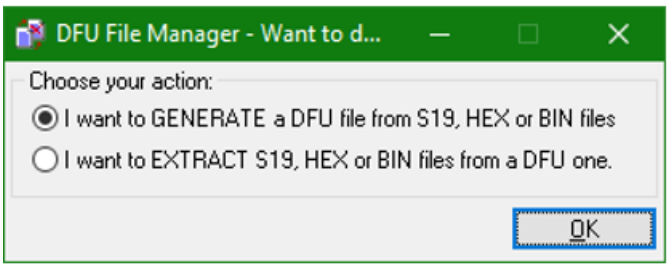
```
plc perf 9 0
```

6.1.2 How to generate .dfu file?

DFU File Manager allows generating from .hex file a .dfu file:

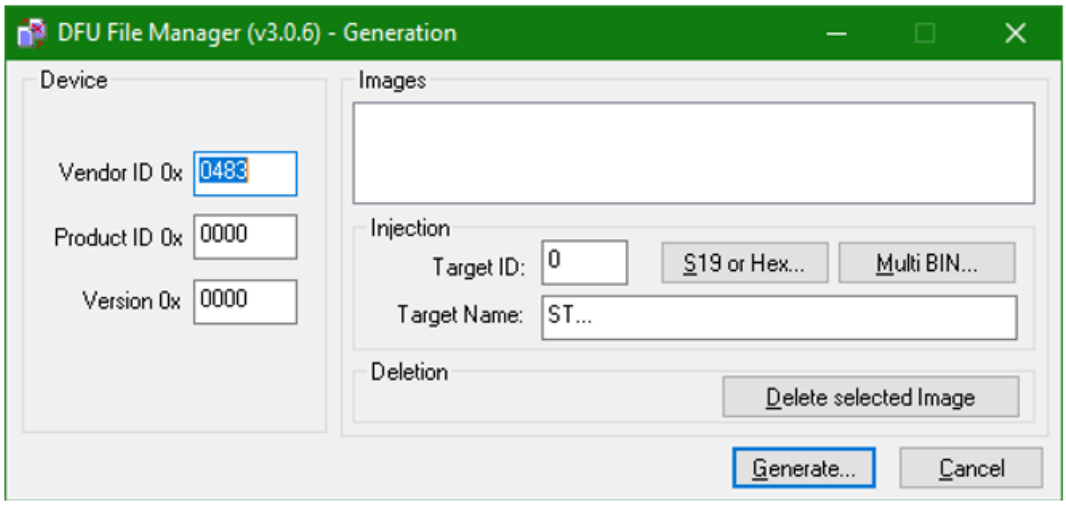
- Launch the DFU FM tool
- Choose HEX file in the dialog box below

Figure 12. **DFU File Manager Window**



- Click on “S19 or Hex…” and select the .hex file generated from IAR build

Figure 13. **DFU File Manager - Generation Window**



- Click on “Generate” to get the .dfu file

6.2 Application task API details

6.2.1 ST8500 images download

ST8500 PE and RTE images are downloaded at boot by STM32 when “PLC mode” switch is configured accordingly. In such case, the images are read from the serial flash attached to the STM32 and transferred to ST8500 using Host Interface.

This requires various steps (baudrate change, image transfer initialization with image header, image transfer by small packets, image start).

All this is handled in a state machine that is implemented in app_Download.c file:

```
/**
* @brief  This function implements the state machine of image download
* @param  event that triggers transitions
* @retval None
*/
static void APP_ImgDlStateMachine(uint32_t event)
```

6.2.2 ST8500 configuration

Once PE and RTE images are started on ST8500, the STM32 proceeds with some configuration:

- Trigger G3lib reset using desired band and profile
- Set some attributes used for G3 operation
- Configure LED and Trace port on ST8500

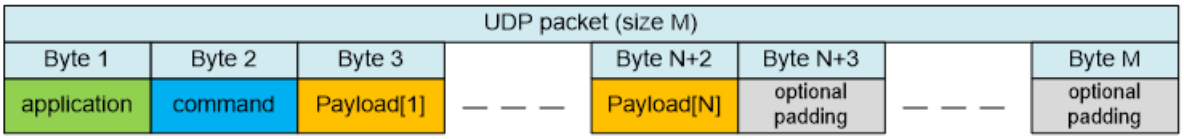
All this is handled in a state machine that is implemented in app_Config.c file:

```
/**
  * @brief  This function handles the state machine for ST8500 configuration
  * @param  event to trigger state transitions
  * @retval next event
  */
configEvent_t APP_G3ConfigStarcom(configEvent_t event)
```

6.2.3 Sending and receiving UDP messages

In the demo examples, the UDP messages are formatted as shown below.

Figure 14. UDP message format



In order to send an UDP packet, the following function in app_Task.c must be invoked. PayloadSize is the size of payload and msgSize is the total size for payload + optional padding.

```
typedef struct {
u8    application;
  u8    command;
u16   destAddress;
  u16   msgSize;
u16   payloadSize;
  u8    *payload;
 } appSendMsg_t;

/**
* @brief   Format and Send UDP message
* @param    msg is a pointer to the structure containing required parameters
to send an UDP message
* @retval  None
*/
void APP_SendMessage(appSendMsg_t *msg)
```

On reception of an UDP packet, the following functions are invoked in app_Test.c.

On the PAN coordinator:

```
/**
* @brief   Handles received UDP packet (Coordinator)
* @param   pData is a pointer to UDP packet
* @retval  None
*/
void APP_TestHandleRxUdpPacket_Coord(u8 *pData)
```

```
/**
* @brief   Handles received UDP packet (Device)
* @param   pData is a pointer to UDP packet
* @retval  None
*/
void APP_TestHandleRxUdpPacket_Device(u8 *pData)
```

Revision history

Table 1. **Document revision history**

Date	Version	Changes
11-June-2019	1	Initial release.

Contents

List of tables

List of figures