UM1818
User manual

Street lighting power line modem evaluation board based on ST7580 PLM and STM32 microcontroller

Introduction

This document explains how to use and set up the firmware and the software designed for the STEVAL-IHP007V1 board and all the necessary setup for using the hardware.

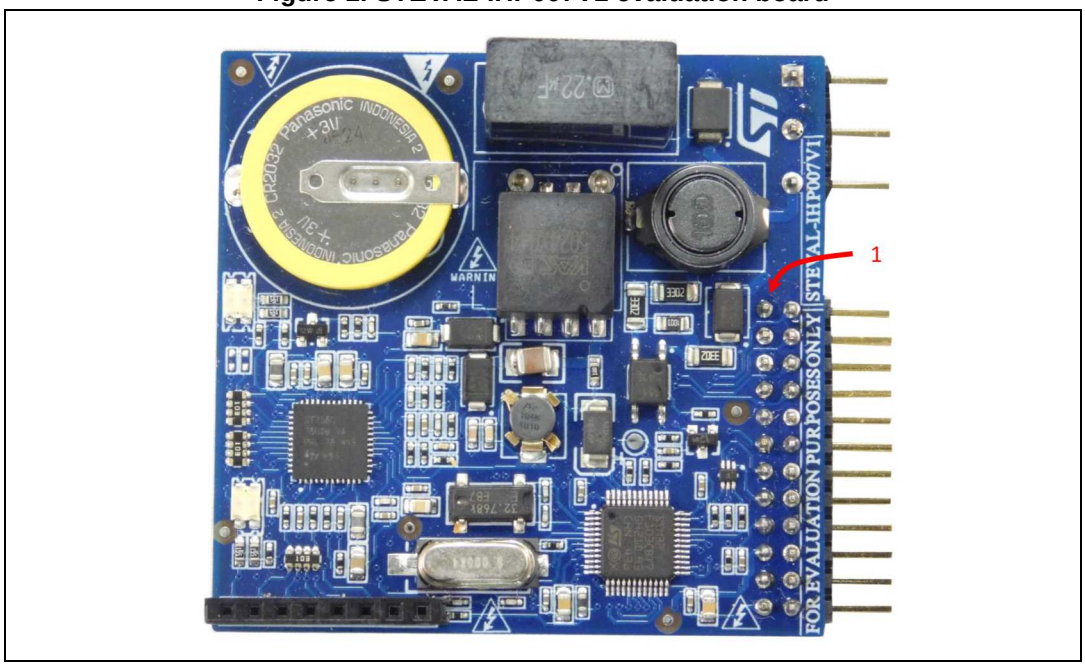
The system is based to the ST7580 data link protocol firmware, data link protocol described in the application note AN4018. ST7580 data link protocol firmware is organized in a layer structure. A dedicated layer allows the user to design its own application interfacing to the evaluation board features with very simple and easy to use APIs.

Dedicated software graphic user interface (GUI) allows the user to use all the embedded features interfacing the PLM evaluation board with the PC via an RS232 communication port.

This firmware is developed using STM32F10x standard peripherals library Rel.3.5.0 and IAR Embedded Workbench® IDE for STM32 microcontrollers Rel. 6.50x

The STEVAL-IHP007V1 hardware evaluation board embeds an ARM 32-bit Cortex™-M3 core-based STM32F103xB and an ST7580 PSK multi mode power line networking system-on-chip.

Figure 1. STEVAL-IHP007V1 evaluation board



Contents

1 Features

- Configurable PSK power line modem interface with an embedded firmware stack for a complete power line communication management
- 8 user configurable general purpose input/output pins
- USART communication channel for evaluation board interfacing
- Internal configurable RTC evaluation board with lithium backup battery
- Programmable user data and PLM parameters Flash memory area
- Remote firmware update
- Embedded AES 128 encryption evaluation board with programmable AES Key

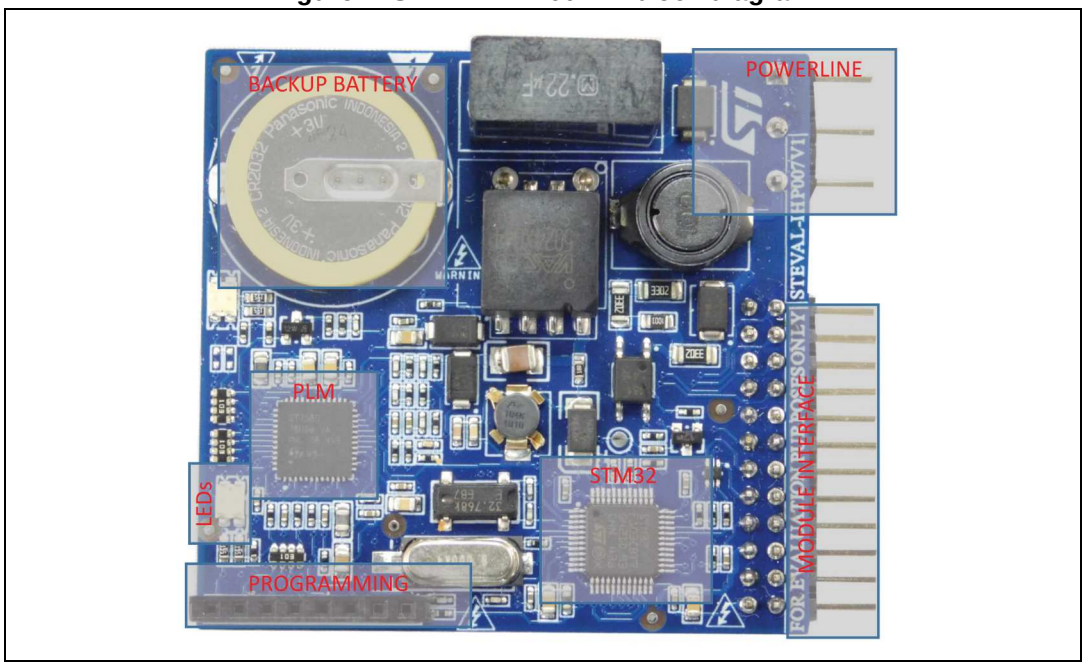
2 Description

The STEVAL-IHP007V1 block diagram is shown in *Figure 2*. The general purpose power line modem evaluation board is based on an ST7580 x-PSK power line modem device and an ARM 32-bit Cortex™-M3 core based STM32F103xB microcontroller. The PLM evaluation board is a fully functional communication evaluation board, with 8 programmable I/Os, a real time clock and a Flash memory area for modem parameters and user data storage. The firmware structure is made up of several layers, each dealing with a different feature. The application layer engine is the general interface between the user program and all the parts of the evaluation board. It manages the communication ports, the evaluation board peripherals such as SCI, RTC, I/Os, LEDs and timing management. It is also the interface between the PLM communication protocol and the user application layer. The PLM communication protocol, itself made up of several layers, implements and manages the power line communication, manages the conflicts, timing and repetitions, the addressing, and so on. Please refer to AN4018 for details on the ST7580 communication protocol features and application services provided. Some features are managed directly by the application engine, and are transparent to the user, such as the RTC management or the board parameter update, as well as the board programming and configuration, which is done by particular programming or service commands managed and acknowledged directly by the application engine. Even the remote firmware update is managed by the application engine and allows the firmware being update remotely by power line modem.

The STEVAL-IHP007V1 is powered by a dual regulated DC power source, +12VDC (pin 1) and +3.3 VDC (pin 2) from the power supply connector (J2). The pin 3 is the ground.

The communication is done via power line, which is applied to the board using the J1 connector, where the pin 1 must be connected to the neutral wire while the pin 3 to the phase wires (refer to the *Appendix A*).

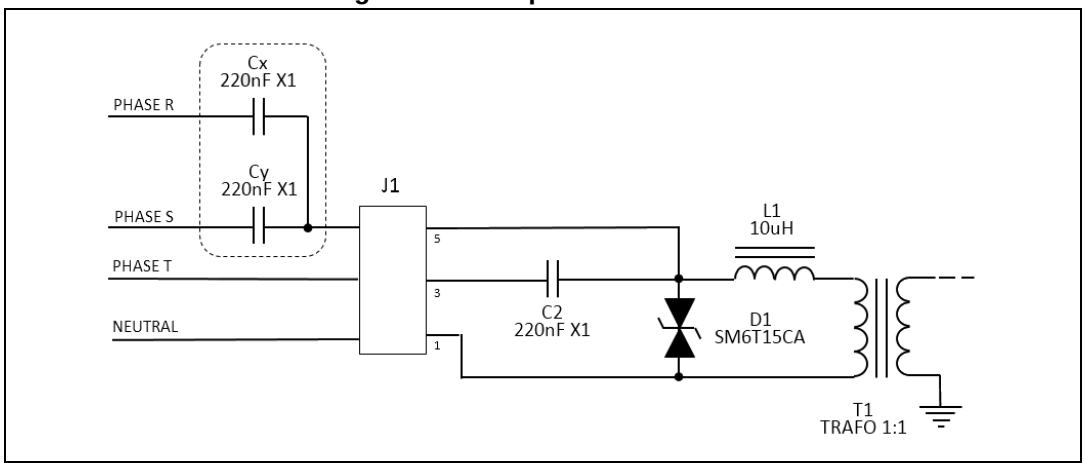
Figure 2. STEVAL-IHP007V1 block diagram



It is possible to connect the evaluation board in a three phase line (in case of communication modules are connected in all three phases), in this case an external

capacitor of 220 nF X1 must be connected to any additional phase, and then the other side of capacitors together with the common pin 5 of the J2 connector, following the schematic shown in the *Figure 3*, and the 0 Ω resistor R1 must be mounted.

Figure 3. Three phase connection



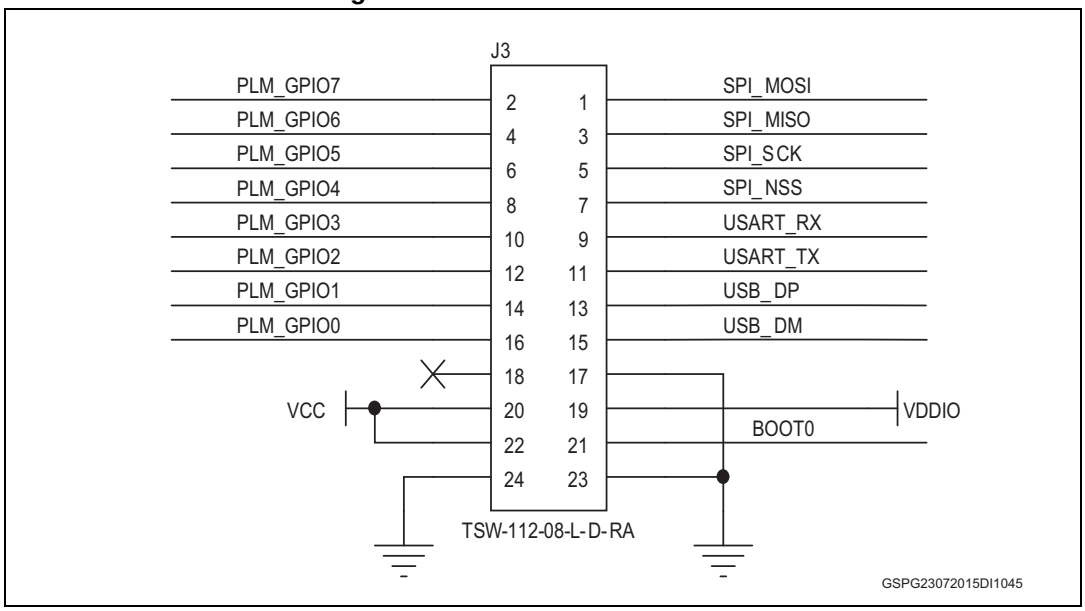
The GP PLM module is provided of a user interface (J3) shown in *Figure 4* where there are connected the SPI interface pins (MOSI, MISO, SCK and NSS), the RS232 interface pins (Tx and Rx), the USB interface pins (D+ and D-) and the user programmable general purpose I/O pins. Remark that those pins are directly connected to the microcontroller, so be careful to provide the correct insulation and

protection depending on the use those pins are addressed.

It is possible to power the PLM using the +3.3 VDC (pin 19), +12 VDC (pin 20 or 22) and GND (pins 17, 23, 20 and 24) of this connector instead of the connector J2, using only a single connector for power supply and control signals.

A lithium backup battery mounted on the board and 32 kHz quartz allows the use of the full functionality of the internal RTC of the microcontroller, allowing precise time-based operations.

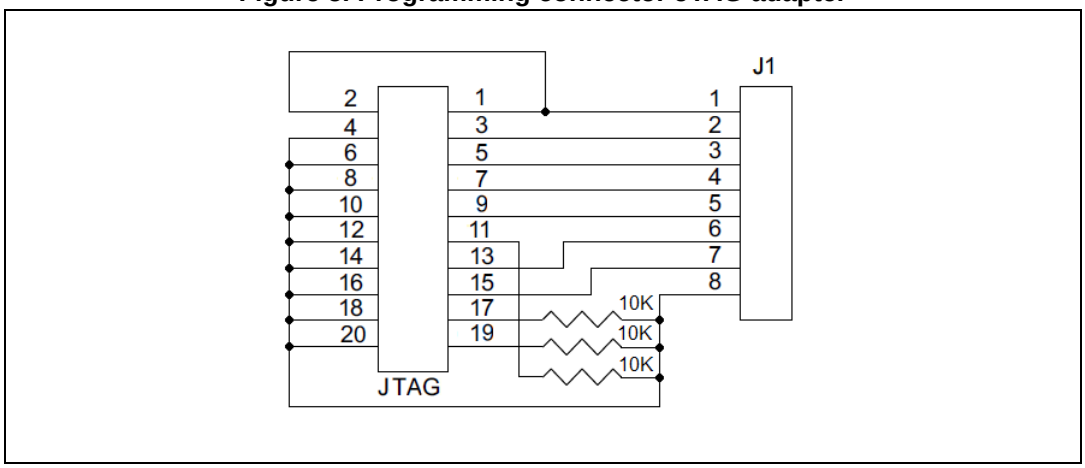
Figure 4. User interface connector



A two color LED allows the signaling the board operations, data transmission and reception.

Finally, a programming connector allows the firmware download and debug, even if it is possible to use the remote firmware update feature to remotely update the firmware using the PLM, as described further in this user manual. If the IAR - JLINK/JTRACE is used for the firmware downloading, a simple JTAG adapter is necessary. The *Figure 5* shows the adapter schematics.

Figure 5. Programming connector JTAG adapter



3 Hardware installation

Connect a regulated dual DC power supply to the power source pins of the connector J3 as described previously and power the board.

In order to download the firmware, plug the programmer adapter (*Figure 5*) in the programming connector J1 and the IAR J-Link programmer in the JTAG connector of the adapter.

Refer to the chapter firmware description (paragraph 6.3 for the firmware download procedure). As soon as the application is launched, the LEDs should quickly switch on indicating that the board has to be configured.

4 Firmware description

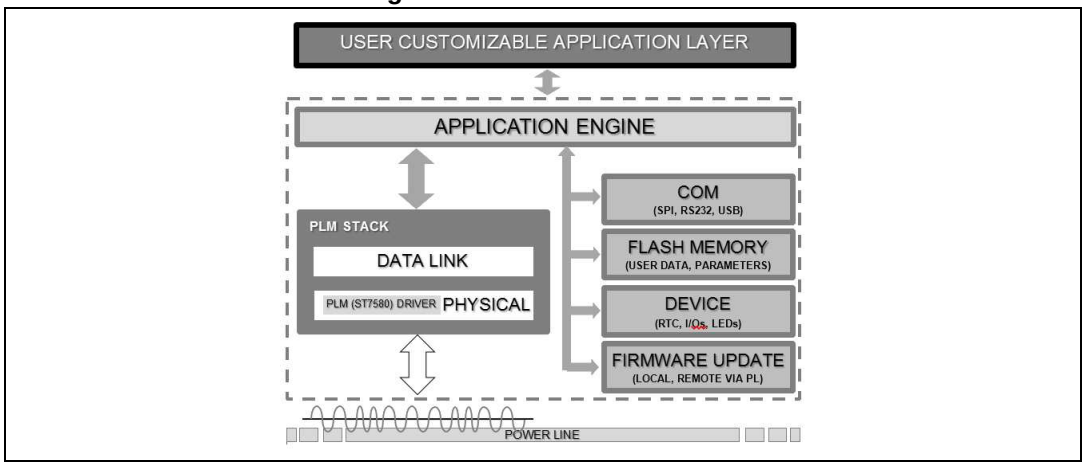
4.1 Introduction

The firmware structure is constituted of several layers each of it taking care of a different feature. The application layer engine is the general interface between the user program and all the parts of the module. It takes care of the communication ports, the board peripherals as RTC and I/Os, and timing management. It is also the interface between the PLM communication protocol and the user program. The PLM communication protocol, itself constituted by several layers, implements and manages the power line communication, manages the conflicts, timing and repetitions, the addressing and so on.

Some features are managed directly by the application engine, and are transparent to the user, as well as the board programming and configuration, which is done by particular programming or service commands managed and acknowledged directly by the application engine, the RTC management, the board parameter or the firmware update.

The user application can be interfaced to the application engine by simples APIs used for the data transfer and the evaluation board interface. The *Figure 6* shows the firmware structure.

Figure 6. Firmware structure



The file user.c/h and application.c/h are the owner of application management, this manage the data flow from/to the PLM physical and from/to the UART side.

4.2 Remote firmware update (RFU)

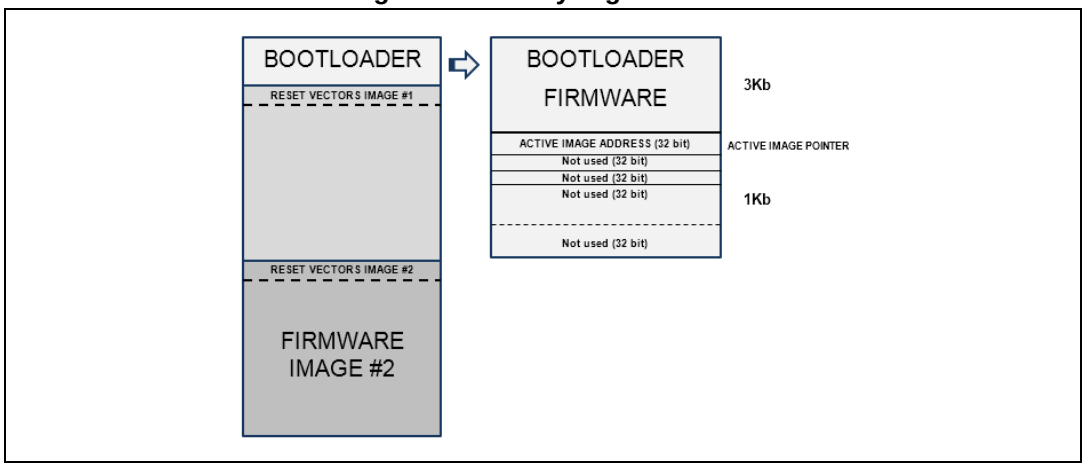
The remote firmware update (RFU) uses the power line modem as external communication channel for receiving a new firmware dump. The firmware dump is placed in the internal Flash memory of the microcontroller. Hence, the total memory size of the microcontroller must be at least the double of the estimated maximum size of the firmware application (in this application is set to 60 kbytes), plus 4 Kb of additional memory for a bootloader. The *Figure 7* shows the microcontroller memory organization.

The bootloader is loaded at the startup and checks the active segment containing the actual firmware. The implemented mechanism uses three partitions of the microcontroller's Flash

memory, one containing the bootloader and two containing the actual running firmware (active image) and the new firmware as soon as a RFU is needed.

As soon as the firmware transfer is completed, a “swap” command sent from the remote PLM causes the target PLM to check first the integrity of the firmware dump (actually a checksum is calculated and compared with the one sent by the remote PLM), and after the reset vector address of the new firmware is written in a dedicated Flash segment of the bootloader. Last the microcontroller is self reset, and the new firmware executed.

Figure 7. Memory organization



The RFU protocol manages the RFU “start”, “get new firmware segment” (with the segment address) and “swap” commands. The protocol is not embedded in the bootloader, hence it can be updated with the new firmware, bust the user must be careful in the modifications as any bugs can compromise the RFU mechanism.

As soon as a new firmware segment is received, the RFU manager checks if the address is within the firmware interrupt vector table. If it is the case, an offset depending on the free firmware image (1 or 2) allocation is added to each interrupt vector before being written in the free image Flash area.

In the *Figure 8* is shown the RFU flowchart.

Figure 8. RFU flowchart (1of 2)

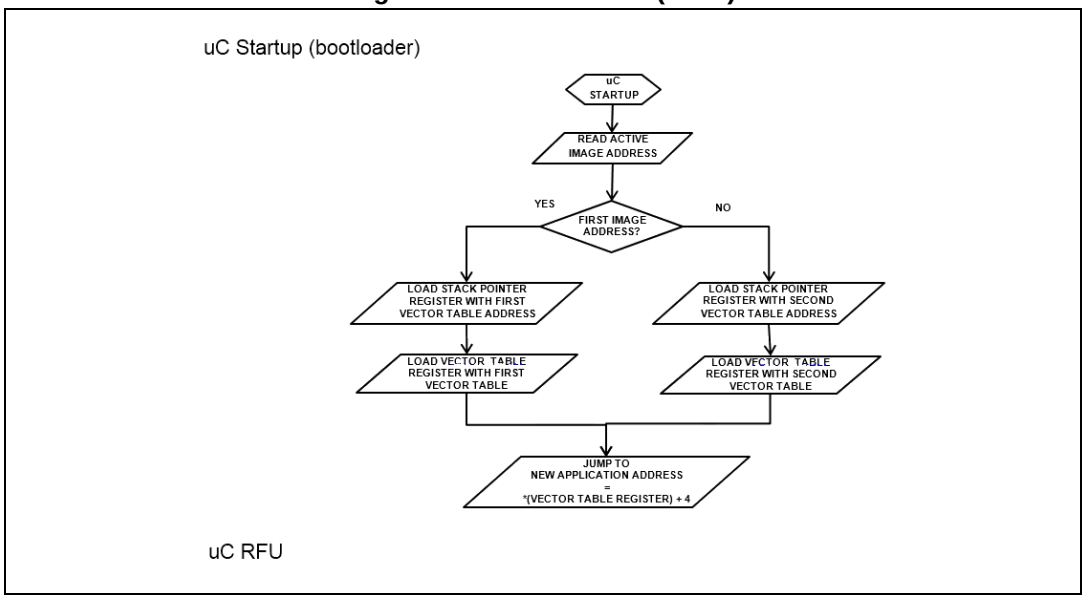
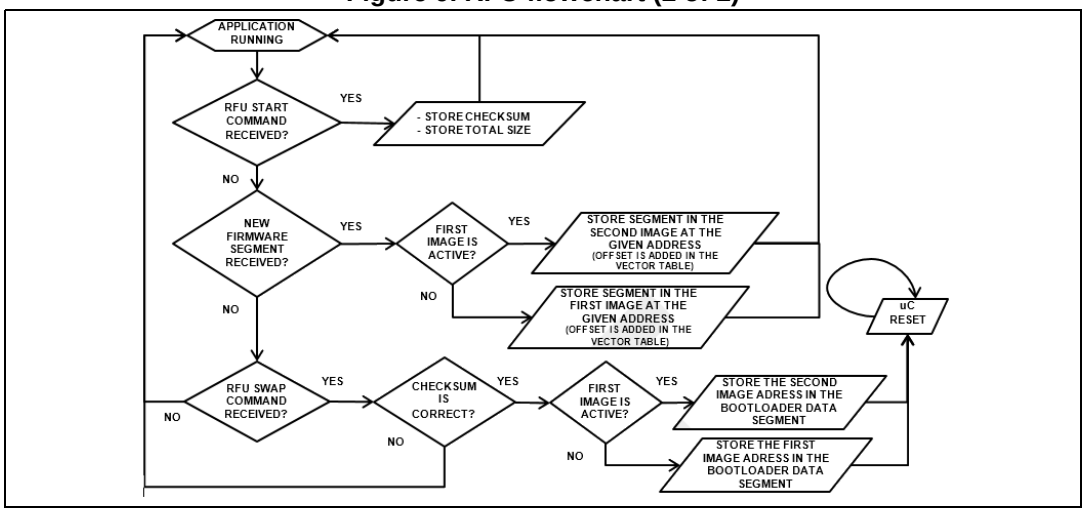


Figure 9. RFU flowchart (2 of 2)

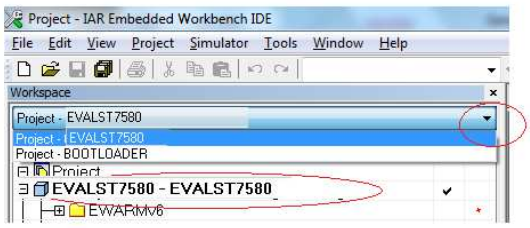


4.3 Firmware download

In the setup directory there are different workspaces stored in different directories. In order to implement the remote firmware update feature it is necessary to download the project located in the workspace “Firmware - Application and Bootloader”. This workspace contains two different projects, one is the bootloader and the other one is the application itself. If the board has never been programmed, this workspace must be downloaded before.

Open the IAR Embedded Workbench® IDE for STM32 microcontrollers Rel. 6.50 (or a more recent release). Click File\Open\Workspace and load the following workspace placed in the directory selected during the setup file installation: “Firmware - Application and Bootloader\EWARM\Project.eww”. Verify that the application project is the active project (the project name must be in bold), otherwise select the active project in the list below the workspace (*Figure 10*).

Figure 10. Active project selection



Click "Project - Batch Build" or press the key F8 in order to compile at the mean time the bootloader and the application.

After the compiling is completed, press "Project - Download and Debug" or press the keys CTRL+D. The both firmware download starts. As soon as the download is completed, press F5 in order to run the application (or exit from the debug mode pressing the keys CTRL+SHIFT+D and unplug the programmer).

If the procedure is done correctly, the orange LED should be on, indicating the board has never been set up before. If it is not the case, try first to erase the memory by clicking Project\Download\Erase Memory and download again the firmware as described before.

Use the GUI interface in order to setup the board and connect it to the power line as described in the dedicated paragraph.

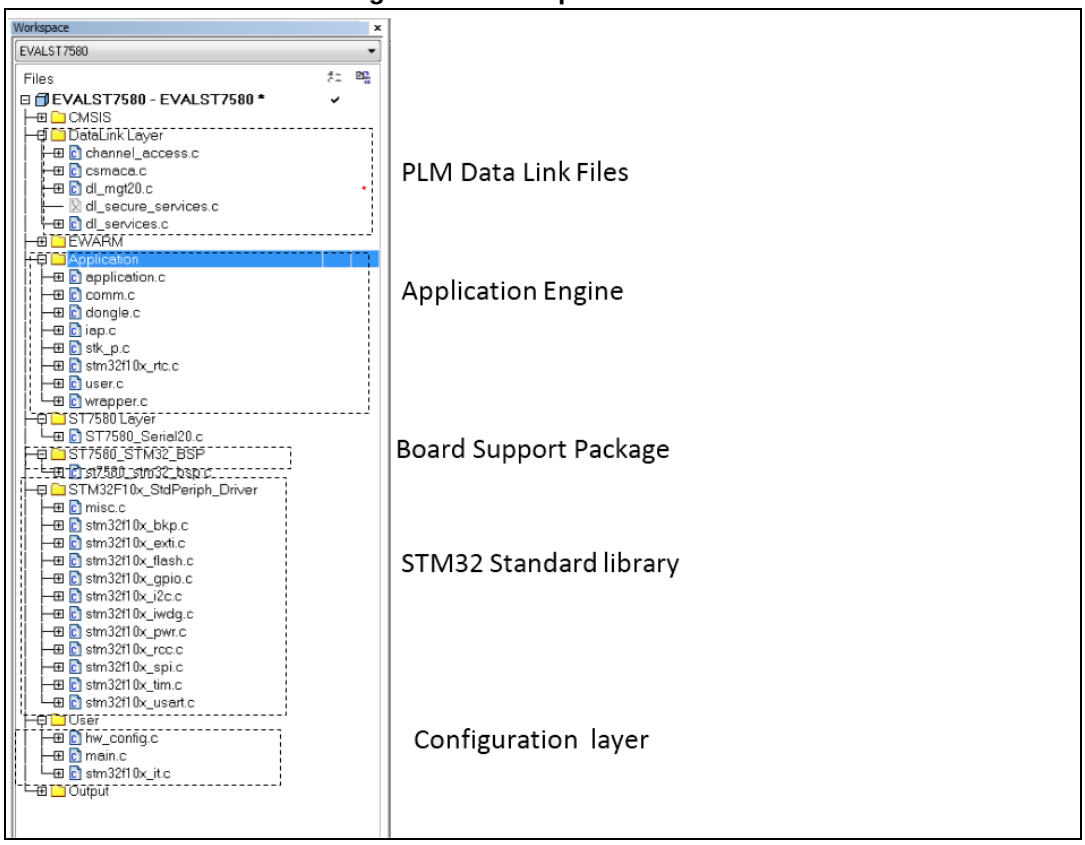
As soon as bootloader has been installed in the evaluation board is possible to remotely (via power line) update the firmware using the RFU feature. Each new firmware version has to be programmed using the workspace “Firmware - Application standalone\EWARM\Project.eww”. The bin file produced by this workspace that is located in the folder “Firmware - Application standalone\ EWARM\PLM_HID_STANDALONE.bin” can be directly loaded using the GUI interface. The difference of this application with the one contained in the workspace with the bootloader is mainly in the stm32f10x_flash.icf linker file and some workspace parameters that are not used in the application without the bootloader (as multiple build, simultaneous debug mode, etc.).

The setup folder contains also the Firmware - Bootloader folder, where inside there is the bootloader firmware; and the folder Firmware - Sniffer which contains the sniffer workspace to download in a PLM module useful if the data sniffing feature of the interface is used. In this case the PLM module will work only as a sniffer.

4.4 Firmware architecture

The structure of the workspace is divided in different sections as shown in the *Figure 11*.

Figure 11. Workspace structure



At this level, all the communication APIs and all the APIs for the application engine interface are available. In the main file, the following code is implemented for running the state machine engines:

```
while(1){
    DL_FSM();//PLM Main Loop

    APP_StackUpdate();// USART State machine

    USER_Program();// PLM State machine

#ifdef  USE_WDG //STM32 Window Watchdog
  IWDG_ReloadCounter();
#endif

#ifdef  KEEP_ALIVE //PLM Keep Alive systems
  Check_KA_Timeout();
#endif
}
```

After the initialization the infinite loop calls three main functions: the DL_FSM(), USER_Program() and the APP_StackUpdate() routine.

The DL_FSM() is the PLM stack main loop, this manage the PLM low level communication. The application engine “APP_StackUpdate()” is the state machine which runs inside the PLM application state machine; this uses the data link service provided from DL_Service layer.

The user program implemented in this user manual realizes a bridge between the power line communication and the COM port: all data arriving from the COM port addressed to another PLM module is sent via PLM, and in turn all data received from PLM is sent back to the COM port. It is necessary that the user program does not stop the core operations (looping instructions) without calling the application engine.

All the hardware configurations are contained in the board support package file, and the file DL_SimpleNodeConf contains the data link configuration parameters and the PLM modem configuration value (frequency, modulation zero crossing delay, etc.).

The next paragraph lists all the data types and the APIs used in the application engine that can be modified by the user if different needs arise.

4.5 Firmware data types

The data type found in the wrapper.h module, are listed hereafter:

```
/* USER FRAME STRUCTURE */

typedef struct
{
  APP_ftype_t   type;
  bool          broadcast;
  u8            address[6];
  u8            len;
  u8            data[USER_PAYLOAD_SIZE]; /* MAX PAYLOAD SIZE: 100 bytes */
  bool          EnableTX
}APP_frame_t;


/* APPLICATION FRAME TYPE */

typedef enum
{
  APP_DATA_FRAME                = 0x00,
  APP_SERVICE_FRAME             = 0x01,
  APP_PING_FRAME                = 0x02,
  APP_ERROR_FRAME               = 0x03,
  APP_PROGRAMMING_FRAME         = 0x04,
  APP_ACK_FRAME                 = 0x05,
  APP_MIB_FRAME                 = 0x06
}APP_ftype_t;
```

```
/* APPLICATION ERRORS */

typedef enum
{
  APP_ERROR_NONE                  = 0x00, // No error
  APP_ERROR_GENERIC               = 0x01, // Generic communication error
  APP_ERROR_COMM_TIMEOUT          = 0x02, // Communication timeout error
  APP_ERROR_SERVICE_GRP_UNKNOWN   = 0x03, // Service group unknown error
  APP_ERROR_SERVICE_CMD_ERROR     = 0x04, // Service command error
  APP_ERROR_COMMUNICATION         = 0x05, // Communication error
  APP_ERROR_ISOLATED_NODE         = 0x06, // Node unreachable error
  APP_ERROR_HARDWARE              = 0x07, // Hardware error
  APP_ERROR_WRONG_PROG_COMMAND    = 0x08, // Wrong programming command
error
  APP_ERROR_WRONG_PROG_GROUP      = 0x09, // Wrong programming group error
  APP_ERROR_DEVICE_BLANK          = 0x0a, // Device blank
  APP_ERROR_RTC_ERROR             = 0x0b, // Error setting the system time
  APP_ERROR_WATCHDOG_DISABLED     = 0x0c, // Hardware reset impossible
  APP_ERROR_NODE_INIT_FAILED      = 0x0d, // Node initialization failure
  APP_ERROR_RTC_DISABLED          = 0x0e  // Internal RTC disabled
}APP_ERROR_t;


/* PROGRAMMING GROUPS */
typedef enum
{
  PROG_GRP_DEVICE_DATA      = 0x00, // Device Data
  PROG_GRP_LL_STACK_PARAM   = 0x01, // Link layer stack parameters
  PROG_GRP_USER_DATA        = 0x02  // User program
}APP_PROG_GROUP_t;

/* SERVICE COMMANDS */

typedef enum
{
  /* NATIVE SERVICE COMMANDS */
  SERVICE_SOFTWARE_RESET    = 0x00,// Reset internal state machines
  SERVICE_HARDWARE_RESET    = 0x01,// Module hardware reset
  SERVICE_PARAM_SET         = 0x02,// Set service parameters
  SERVICE_PARAM_GET         = 0x03,// Get service parameters
  SERVICE_INPUTS_GET        = 0x04,// Get general purpose inputs pin status
  SERVICE_OUTPUTS_SET       = 0x05,// Set general purpose outputs pins
value
  SERVICE_FW_REL_GET        = 0x06,// Get the stack and the module firmware
release
  SERVICE_PLM_CLOCK_SET     = 0x07,// Set the internal time clock value
```

```
  SERVICE_PLM_CLOCK_GET     = 0x08,// Get the internal time clock value
  SERVICE_IO_CONFIG_SET     = 0x09,// Set the general purpose input and
output pins
  SERVICE_IO_CONFIG_GET     = 0x0a,// Get the general purpose input and
output pins
  SERVICE_NET_DISCOVER_REQ    = 0x0b,
  SERVICE_RFU_SET_IMG_HEADER  = 0x0c,
  SERVICE_RFU_SET_IMG_DATA    = 0x0d,
  SERVICE_RFU_SWAP_IMG        = 0x0e,
  SERVICE_SN_SET              = 0x0f,
  SERVICE_SN_GET              = 0x10
  /* USER DEFINED SERVICE COMANDS */
  // SERVICE_USER_CMD_xx       = 0x.., // User defined service commands
(0x0b to 0x7f)
}APP_SER_CMD_t;
```

4.6 Firmware frame types

This paragraph describes all the frame types that are implemented in this firmware. In each field there is also the description.

4.6.1 Communication frames types

Frames exchanged between two PLM modules or between a PLM module and an external device connected to the COMM interface.

From the COMM interface module (USART)

```
buffer[0] = n + 10;                              // Data frame payload length (n + 10)
buffer[1] = DATA_FRAME_TYPE | BROADCAST_FLAG;    // Data frame type
buffer[2,..7] = target_module.address;           // Target device address (6 bytes)
buffer[8,..8+n-1] = user_data[n];                // User data (n bytes, at least 1)
buffer[8+n,8+n+1] = CRC16;                       // CRC-16
```

```
/* BROADCAST_FLAG = 0x80 -> data sent in broadcast - BROADCAST_FLAG = 0x00
-> data sent in unicast */
```

From / to communication interface (PLM, USART)

```
frame.type = DATA_FRAME_TYPE;                    // Data frame type
frame.len = n;                                   // Data frame payload length
frame.broadcast = TRUE / FALSE;                  // TRUE = broadcast, FALSE = unicast
frame.address = target_module.address;           // Target device address (6 bytes)
frame.data[n] = service_data[n];                 // User data (n bytes)
```

4.6.2 Ping frames

This particular frame is used to ping a remote (via PLM interface) or a local (via COMM interface) module. When the ping frame is received, this is managed directly at the data link layer and is not notified at the application and consequently the user levels.

From the COMM interface module (USART).

```
buffer[0] = 10;                              // Ping frame payload length
buffer[1] = APP_PING_FRAME;                  // Ping frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8,9] = CRC16;                         // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                              // Frame payload length
buffer[1] = APP_ACK_FRAME;                   // ACK frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8]= APP_PING_FRAME;                   // Ping frame type
buffer[9]= command_echo;                     // Echo
buffer[10,11] = CRC16;                       // CRC-16
```

4.6.3 Error frames

Can be considered as data frames; they are user error frames from user application level addressed to a target PLM module.

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                              // Error frame payload length
buffer[1] APP_ERROR_FRAME;                   // Target device address (6 bytes)
buffer[2,..7] = target_module.address;       // Ping frame type
buffer[8,9]= user_error_code;                // Echo
buffer[10,11] = CRC16;                       // CRC-16
```

Error code list

```
0x0000 = APP_ERROR_NONE
0x0001 = APP_ERROR_GENERIC
0x0002 = APP_ERROR_COMM_TIMEOUT
0x0003 = APP_ERROR_SERVICE_GRP_UNKNOWN
0x0004 = APP_ERROR_SERVICE_CMD_ERROR
0x0005 = APP_ERROR_COMMUNICATION
0x0006 = APP_ERROR_ISOLATED_NODE
0x0007 = APP_ERROR_HARDWARE
0x0008 = APP_ERROR_WRONG_PROG_COMMAND
0x0009 = APP_ERROR_WRONG_PROG_GROUP
0x000a = APP_ERROR_DEVICE_BLANK
0x000b = APP_ERROR_RTC_ERROR
0x000c = APP_ERROR_WATCHDOG_DISABLED
0x000d = APP_ERROR_NODE_INIT_FAILED
0x000e = APP_ERROR_RTC_DISABLED
```

```
user_error_code = (APP_ERROR_t)(buffer[8]<<8)| buffer[9]);
```

4.6.4 Service frames

Frames containing service commands concerning both some native module features (internal clock, general purpose inputs and outputs, etc.) and user defined service frames. Native frames are managed directly by the application engine.

From the COMM interface module (USART)

```
buffer[0] = n + 11;                              // Service frame payload length (n + 11)
buffer[1] = APP_SERVICE_FRAME | BROADCAST_FLAG;// Service frame type
buffer[2,..7] = target_module.address;           // Target device address (6 bytes)
buffer[8] = (APP_SER_CMD_t)command;              // Service command
buffer[9,..9+n-1] = service_data[n];             // Service data
buffer[9+n, 9+n+1] = CRC16;                      // CRC-16
```

```
/* BROADCAST_FLAG = 0x80 -> data sent in broadcast - BROADCAST_FLAG = 0x00
-> data sent in unicast */
```

From / to communication interface (PLM, USART)

```
buffer[0] = n + 11;                               // length
buffer[1] = DATA_FRAME_TYPE;                      // Service or ACK frame type
buffer[2,..7] = target_module.address;            // Target device address (6 bytes)
buffer[8] = (APP_SER_CMD_t)command/APP_ACK_FRAME;// Service command/ACK
buffer[9,..n] = service_data[n]/Command_Echo;     // Service data or Command echo(n=1)
buffer[n+1,n+2] = CRC16;
```

Service command list

```
/* NATIVE SERVICE COMMANDS */
SERVICE_SOFTWARE_RESET    = 0x00,     // Reset internal state machines
SERVICE_HARDWARE_RESET    = 0x01,     // Module hardware reset
SERVICE_PARAM_SET         = 0x02,     // Set service parameters
SERVICE_PARAM_GET         = 0x03,     // Get service parameters
SERVICE_INPUTS_GET        = 0x04,     // Get general purpose inputs pin status
SERVICE_OUTPUTS_SET       = 0x05,     // Set general purpose outputs pins value
SERVICE_FW_REL_GET        = 0x06,     // Get the stack and the module firmware release
SERVICE_PLM_CLOCK_SET     = 0x07,     // Set the internal time clock value
SERVICE_PLM_CLOCK_GET     = 0x08,     // Get the internal time clock value
SERVICE_IO_CONFIG_SET     = 0x09,     // Set the general purpose input and output pins
SERVICE_IO_CONFIG_GET     = 0x0a,     // Get the general purpose input and output pins
SERVICE_NET_DISCOVER_REQ    = 0x0b,
SERVICE_RFU_SET_IMG_HEADER  = 0x0c,
SERVICE_RFU_SET_IMG_DATA    = 0x0d,
SERVICE_RFU_SWAP_IMG        = 0x0e,
SERVICE_SN_SET              = 0x0f,
SERVICE_SN_GET              = 0x10
/* USER DEFINED SERVICE COMANDS */
0x.. = SERVICE_USER_CMD_xx
```

PLM reset: software, hardware

From the COMM interface (USART)

```
buffer[0] = 11;                       // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;        // Service frame type
buffer[2,..7] = target_module.address; // Target device address (6 bytes)
/* FOR SOFTWARE RESET */
buffer[8] = SERVICE_SOFTWARE_RESET;
/* FOR HARDWARE RESET */
buffer[8] = SERVICE_HARDWARE_RESET;
buffer[9,10] = CRC16;                 // CRC-16
```

Note: Any response from PLM module

Set module parameters: programming user parameters

From the COMM interface (USART)

```
buffer[0] = 44                                  // Service frame payload length
buffer[1] = APP_SERVICE_FRAME | BROADCAST_FLAG;// Service frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8] = SERVICE_PARAM_SET;                  // Service command
buffer[9] = PROG_GRP_USER_DATA;                 // Type command
buffer[10 -> 41] = *user_data_buffer;           // Program data
buffer[42,43] = CRC16;                          // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                                 // Service frame payload length
buffer[1] = APP_ACK_FRAME;                      // ACK frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8]= APP_SERVICE_FRAME;                   // Service frame type
buffer[9]= command_echo;                        // Echo
buffer[10,11] = CRC16;                          // CRC-16
```

Get module parameters: programming user parameters

From the COMM interface (USART)

```
buffer[0] = 12;                                 // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;                  // Service frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8] = SERVICE_PARAM_GET;                  // Service command
buffer[9] = PROG_GRP_USER_DATA;                 // Type command
buffer[10,11] = CRC16;                          // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 44;                                 // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;                  // Service frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8] = SERVICE_PARAM_GET;                  // Service command
buffer[9] = PROG_GRP_USER_DATA;                 // Type command
buffer[10 -> 41] = *user_data_buffer;           // Program Data
buffer[42,43] = CRC16;                          // CRC-16
```

Get module general purpose inputs/outputs configuration

From the COMM interface (USART)

```
buffer[0] = 11;                          // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;           // Service frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8] = SERVICE_IO_CONFIG_GET;       // Service Command
buffer[9,10] = CRC16;                    // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                          // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;           // Service frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8] =SERVICE_IO_CONFIG_GET;        // Service command
buffer[9] = *sender.configuration_value; // input/output configuration
buffer[10,11] = CRC16;                   // CRC-16
```

Set module general purpose inputs/outputs configuration

From the COMM interface (USART)

```
buffer[0] = 12;                          // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;           // Service frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8] = SERVICE_IO_CONFIG_SET;       // Service command
buffer[9] = target.configuration_value;  // bit x = 1 -> IOx = output, bit x = 0
                                         -> IOx = input
buffer[10,11] = CRC16;                   // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                          // Service frame payload length
buffer[1] = APP_ACK_FRAME;               // ACK frame type
buffer[2,..7] = target_module.address    // Target device address (6 bytes)
buffer[8]= APP_SERVICE_FRAME;            // Service frame type
buffer[9]= command_echo;                 // Echo
buffer[10,11] = CRC16;                   // CRC-16
```

Get module general purpose inputs value

From the COMM interface (USART)

```
buffer[0] = 11;                          // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;           // Service frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8] = SERVICE_INPUTS_GET;          // Service command
buffer[9,10] = CRC16;                    // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                          // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;           // Service frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8] =SERVICE_INPUTS_GET;           // Service command
buffer[9] = *sender.inputs_value;        // Read GPIO Input value
buffer[10,11] = CRC16;                   // CRC-16
```

Set module general purpose outputs value

From the COMM interface (USART)

```
buffer[0] = 12;                          // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;           // Service frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8] = SERVICE_OUTPUTS_SET;         // Service command
buffer[9] = target.outputs_value;        // Set GPIO output value
buffer[10,11] = CRC16;                   // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                          // Service frame payload length
buffer[1] = APP_ACK_FRAME;               // ACK frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8]= APP_SERVICE_FRAME;            // Service frame type
buffer[9]= command_echo;                 // Echo
buffer[10,11] = CRC16;                   // CRC-16
```

Get module firmware release

From the COMM interface (USART)

```
buffer[0] = 11;                          // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;           // Service frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8] = SERVICE_FW_REL_GET;          // Service command
buffer[9,10] = CRC16;                    // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 15;                                 // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;                  // Service frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8] = SERVICE_FW_REL_GET;                 // Service command
buffer[9,10] = target_module.firmware_release; // Module firmware release (x.y)
buffer[11,12] = target_module.stack_release;   // Stack firmware release (x.y)
buffer[13,14] = CRC16;                          // CRC-16
```

Get module time clock value

From the COMM interface (USART)

```
buffer[0] = 11;                                 // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;                  // Service frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8] = SERVICE_PLM_CLOCK_GET;              // Service command
buffer[9,10] = CRC16;                           // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 14;                                 // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;                  // Service frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8] = SERVICE_PLM_CLOCK_GET;              // Service command
buffer[9] = target_module.hours;                // Hours
buffer[10] = target_module.minutes;             // Minutes
buffer[11] = target_module.seconds;             // Seconds
buffer[12,13] = CRC16;                          // CRC-16
```

Set module time clock value

From the COMM interface (USART)

```
buffer[0] = 14;                                 // Service frame payload length
buffer[1] = APP_SERVICE_FRAME | BROADCAST_FLAG;// Service frame type
buffer[2,..7] = target_module.address;          // Target device address (6 bytes)
buffer[8] = SERVICE_PLM_CLOCK_SET;              // Service command
buffer[9] = target_module.new_hours;            // Hours
buffer[10] = target_module.new_minutes;         // Minutes
buffer[11] = target_module.new_seconds;         // Seconds
buffer[12,13] = CRC16;                          // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                              // Service frame payload length
buffer[1] = APP_ACK_FRAME;                   // Service frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8]= APP_SERVICE_FRAME;                // Service frame type
buffer[9]= command_echo;                     // Echo
buffer[10,11] = CRC16;                       // CRC-16
```

GET PLM Module SNR ratio

From the COMM interface (USART)

```
buffer[0] = 11;                              // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;               // Service frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8] = SERVICE_SN_GET;                  // Service command
buffer[9,10] = CRC16;                        // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                              // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;               // Service frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8] = SERVICE_SN_GET;                  // Service command
buffer[9] = *sender.sn_value;                // SNR Ratio from 0 to 31
buffer[10,11] = CRC16;                       // CRC-16
```

Set PLM Module SNR ratio

From the COMM interface (USART)

```
buffer[0] = 12;                              // Service frame payload length
buffer[1] = APP_SERVICE_FRAME;               // Service frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8] = SERVICE_SN_SET;                  // Service command
buffer[9] = target.sn_value;                 // SNR Ratio from 0 to 31
buffer[10,11] = CRC16;                       // CRC-16
```

From / to communication interface (PLM, USART)

```
buffer[0] = 12;                              // Service frame payload length
buffer[1] = APP_ACK_FRAME;                   // ACK frame type
buffer[2,..7] = target_module.address        // Target device address (6 bytes)
buffer[8]= APP_SERVICE_FRAME;                // Service frame type
buffer[9]= command_echo;                     // Echo
buffer[10,11] = CRC16;                       // CRC-16
```

Appendix A HID commands

HID frames are data frames where the payload is customized for the HID ballast application.

From the COMM interface (USART)

```
buffer[0] = n + 10;                          // Data frame payload length (n + 10)
buffer[1] = APP_DATA_FRAME | BROADCAST_FLAG;  // Data frame type
buffer[2,..7] = target_module.address;        // Target device address (6 bytes)
buffer[8,..8+n-1] = user_data[n];             // User data (n bytes, at least 1)
buffer[8+n,8+n+1] = CRC16;                    // CRC-16
```

```
/* BROADCAST_FLAG = 0x80 -> data sent in broadcast - BROADCAST_FLAG = 0x00
-> data sent in unicast */
```

From / to communication interface (PLM, USART)

```
buffer[0] = n + 10;                          // Data frame payload length (n + 10)
buffer[1] = APP_DATA_FRAME;                  // Data frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8,..n] = HID_DATA;                    // HID data
buffer[n+1,n+2] = CRC16;                     // CRC-16
```

HID board reset

From the COMM interface (USART)

```
buffer[0] = 13;                         // Data frame payload length
buffer[1] = APP_DATA_FRAME;             // Data frame type
buffer[2,..7] = target_module.address; // Target device address (6 bytes)
buffer[8] = 0x73;                       // command set char 's'
buffer[9] = 0x00;                       // HID board command identify "reset"
buffer[10] = reset_type;                // Software reset = 0xaa, Hardware reset = 0x0c
buffer[11,12] = CRC16;                  // CRC-16
```

From / to communication interface (PLM, USART)

Case hardware reset, the PLM set the reset GPIO pin to reset the HID ballast (GPIO Port A Pin 6)

```
buffer[0] = 12;                          // Data frame payload length
buffer[1] = APP_ACK_FRAME;               // ACK frame type
buffer[2,..7] = target_module.address;   // Target device address (6 bytes)
buffer[8]= APP_DATA_FRAME;               // Data frame type
buffer[9]= command_echo;                 // Echo
buffer[10,11] = CRC16;                   // CRC-16
```

Case Software reset

```
buffer[0] = 12;                                  // Data frame payload length
buffer[1] = APP_ACK_FRAME;                       // ACK frame type
buffer[2,..7] = target_module.address;           // Target device address (6 bytes)
buffer[8]= APP_DATA_FRAME;                       // Data frame type
buffer[9]= 0x73;                                 // Command char 's'
buffer[10,11] = CRC16;                           // CRC-16
```

Lamp dimming

From the COMM interface (USART- Source “Concentrator”) and To the COMM interface (USART- Destination “HID ballast”)

```
buffer[0] = 13;                                  // Data frame payload length
buffer[1] = APP_DATA_FRAME | BROADCAST_FLAG;     // Data frame type
buffer[2,..7] = target_module.address;           // Target device address (6 bytes)
buffer[8] = 0x73;                                // command set char 's'
buffer[9] = 0x01;                                // HID board command identify "Dimming"
buffer[10] = dimming_value;                      // Dimming value 0 - 100 (0% - 100%)
buffer[11,12] = CRC16;                           // CRC-16
```

From the COMM interface (USART- Destination “HID ballast”)

```
buffer[0] = 2;                                   // Data frame payload length
buffer[1] = APP_DATA_FRAME;                      // Data frame type
buffer[2] = 0x73;                                // Char 's'
```

To the COMM interface (USART- Source “Concentrator”)

```
buffer[0] = 12;                                  // Data frame payload length
buffer[1] = APP_ACK_FRAME;                       // ACK frame type
buffer[2,..7] = target_module.address;           // Target device address (6 bytes)
buffer[8]= APP_DATA_FRAME;                       // Data frame type
buffer[9]= 0x73;                                 // Command char 's'
buffer[10,11] = CRC16;                           // CRC-16
```

Get HID parameter

From the COMM interface (USART- Source “Concentrator”) and To the COMM interface (USART- Destination “HID ballast”)

```
buffer[0] = 12;                              // Data frame payload length
buffer[1] = APP_DATA_FRAME;                  // Data frame type
buffer[2,..7] = target_module.address;       // Target device address (6 bytes)
buffer[8] = 0x67;                            // command set char 'g'
buffer[9] = param_to_get;                    // HID Param Value
buffer[10,11] = CRC16;                       // CRC-16


/* param_to_get value */

0x00  = HARDWARE VERSION
0x01  = BUS VOLTAGE
0x02  = LAMP VOLTAGE
0x03  = HARDWARE STATUS
0x04  = LAMP POWER
0x05  = LAST FAILURE
0x06  = POWER SUPPLY VOLTAGE
0x07  = BOARD TEMPERATURE (°C)
0x08  = N° OF POWER ON
0x09  = LAMP LIFETIME (Hours)
0x0a  = GET ALL PARAMETERS
```

From the COMM interface (USART- Destination “HID Ballast”)

```
/* GET HARDWARE VERSION */

buffer[0] = 5;                   // Data frame payload length
buffer[1] = 0x67;                // Get command 'g' ascii value
buffer[2] = 0x00;                // HARDWARE VERSION
buffer[3] = hardware_release_xx; // Hardware release xx.yy (xx, yy = bcd format)
buffer[4] = hardware_release_yy; // Hardware release xx.yy (xx, yy = bcd format)
```

```
/* GET BUS VOLTAGE */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x01;               // BUS VOLTAGE
buffer[3] = bus_voltage_h;      // High byte of bus voltage
buffer[4] = bus_voltage_l;      // Low byte of bus voltage


/* GET LAMP VOLTAGE */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x02;               // LAMP VOLTAGE
buffer[3] = lamp_voltage_h;     // High byte of lamp voltage
buffer[4] = lamp_voltage_l;     // Low byte of lamp voltage


/* GET HARDWARE STATUS */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x03;               // HARDWARE STATUS
buffer[3] = hardware_status_h;  // High byte of hardware status
buffer[4] = hardware_status_l;  // Low byte of hardware status


/* hardware_status value */

0x00 = IDLE
0x01,0x02,0x03 = STARTUP
0x04,0x05 = WARMUP
0x06 = RUN
0x07 = WAIT
0x08 = FAILURE
```

```
/* GET LAMP POWER */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x04;               // LAMP POWER
buffer[3] = lamp_power_h;       // High byte of lamp power
buffer[4] = lamp_power_l;       // Low byte of lamp power


/* GET LAST FAILURE */


buffer[0] = 9;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x05;               // LAST FAILURE
buffer[3] = hardware_status_h;  // High byte of hardware status before the failure
buffer[4] = hardware_status_l;  // Low byte of hardware status before the failure
buffer[5] = bus_voltage_h;      // High byte of bus voltage before the failure
buffer[6] = bus_voltage_l;      // Low byte of bus voltage before the failure
buffer[7] = lamp_voltage_h;     // High byte of lamp voltage before the failure
buffer[8] = lamp_voltage_l;     // Low byte of lamp voltage before the failure


/* GET POWER SUPPLY VOLTAGE */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x07;               // BOARD TEMPERATURE
buffer[3] = board_temperature_h; // High byte of board temperature (°C)
buffer[4] = board_temperature_l; // Low byte of board temperature (°C)


/* GET BOARD TEMPERATURE (°C) */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x07;               // BOARD TEMPERATURE
buffer[3] = board_temperature_h; // High byte of board temperature (°C)
buffer[4] = board_temperature_l; // Low byte of board temperature (°C)
```

```
/* GET N° OF POWER ON */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x08;               // N. OF POWER ON
buffer[3] = power_on_n_h;       // High byte of number of lamp power on
buffer[4] = power_on_n_l;       // Low byte of number of lamp power on


/* GET LAMP LIFETIME (Hours) */


buffer[0] = 5;                  // Data frame payload length
buffer[1] = 0x67;               // Get command 'g' ascii value
buffer[2] = 0x09;               // LAMP LIFETIME (hours)
buffer[3] = lamp_lifetime_h;    // High byte of lamp lifetime (hours)
buffer[4] = lamp_lifetime_l;    // Low byte of lamp lifetime (hours)


/* GET ALL PARAMETERS */


buffer[0] = 5;                    // Data frame payload length
buffer[1] = 0x67;                 // Get command 'g' ascii value
buffer[2] = 0x0a;                 // GET ALL PARAMETERS
buffer[3] = hardware_release_xx;  // Hardware release xx.yy (xx, yy = bcd format)
buffer[4] = hardware_release_yy;  // Hardware release xx.yy (xx, yy = bcd format)
buffer[5] = bus_voltage_h;        // High byte of bus voltage before the failure
buffer[6] = bus_voltage_l;        // Low byte of bus voltage before the failure
buffer[7] = lamp_voltage_h;       // High byte of lamp voltage before the failure
buffer[8] = lamp_voltage_l;       // Low byte of lamp voltage before the failure
buffer[9] = hardware_status_h;    // High byte of hardware status before the failure
buffer[10] = hardware_status_l;   // Low byte of hardware status before the failure
buffer[11] = lamp_power_h;        // High byte of lamp power
buffer[12] = lamp_power_l;        // Low byte of lamp power
buffer[13] = last_hardware_status_h;// High byte of hardware status before the failure
buffer[14] = last_hardware_status_l;// Low byte of hardware status before the failure
buffer[15] = last_bus_voltage_h;  // High byte of bus voltage before the failure
buffer[15] = last_bus_voltage_l;  // Low byte of bus voltage before the failure
buffer[17] = last_lamp_voltage_h; // High byte of lamp voltage before the failure
buffer[18] = last_lamp_voltage_l; // Low byte of lamp voltage before the failure
buffer[19] = input_voltage_h;     // High byte of power supply voltage
buffer[20] = input_voltage_l;     // Low byte of power supply voltage
```

```
buffer[21] = board_temperature_h;     // High byte of board temperature (°C)
buffer[22] = board_temperature_l;     // Low byte of board temperature (°C)
buffer[23] = power_on_n_h;            // High byte of number of lamp power on
buffer[24] = power_on_n_l;            // Low byte of number of lamp power on
buffer[25] = lamp_lifetime_h;         // High byte of lamp lifetime (hours)
buffer[26] = lamp_lifetime_l;         // Low byte of lamp lifetime (hours)
```

To the COMM interface (USART- source “concentrator”)

```
/* GET HARDWARE VERSION */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x00;                     // Response to g0 command
buffer[10] = hardware_release_xx;     // Hardware release xx.yy (xx, yy = bcd format)
buffer[11] = hardware_release_yy;     // Hardware release xx.yy (xx, yy = bcd format)
buffer[12,13] = CRC16;                // CRC-16


/* GET BUS VOLTAGE */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x02;                     // Response to g2 command
buffer[10] = lamp_voltage_h;          // High byte of lamp voltage
buffer[11] = lamp_voltage_l;          // Low byte of lamp voltage
buffer[12,13] = CRC16;                // CRC-16


/* GET HARDWARE STATUS */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x03;                     // Response to g3 command
buffer[10] = hardware_status_h;       // High byte of hardware status
buffer[11] = hardware_status_l;       // Low byte of hardware status
buffer[12,13] = CRC16;                // CRC-16
```

```
/* hardware_status value */
0x00                        = IDLE
0x01,0x02,0x03              = STARTUP
0x04,0x05                   = WARMUP
0x06                        = RUN
0x07                        = WAIT
0x08                        = FAILURE


/* GET LAMP POWER */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x04;                     // Response to g4 command
buffer[10] = lamp_power_h;            // High byte of lamp power
buffer[11] = lamp_power_l;            // Low byte of lamp power
buffer[12,13] = CRC16;                // CRC-16


/* GET LAST FAILURE */

buffer[0] = 18;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x05;                     // Response to g5 command
buffer[10] = hardware_status_h;       // High byte of hardware status before the
                                      failure
buffer[11] = hardware_status_l;       // Low byte of hardware status before the
                                      failure
buffer[12] = bus_voltage_h;           // High byte of bus voltage before the failure
buffer[13] = bus_voltage_l;           // Low byte of bus voltage before the failure
buffer[14] = lamp_voltage_h;          // High byte of lamp voltage before the failure
buffer[15] = lamp_voltage_l;          // Low byte of lamp voltage before the failure
buffer[16,17] = CRC16;                // CRC-16
```

```
/* GET POWER SUPPLY VOLTAGE */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x06;                     // Response to g6 command
buffer[10] = input_voltage_h;         // High byte of power supply voltage
buffer[11] = input_voltage_l;         // Low byte of power supply voltage
buffer[12,13] = CRC16;                // CRC-16


/* GET BOARD TEMPERATURE (°C) */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[1] = APP_DATA_FRAME;           // Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x07;                     // Response to g7 command
buffer[10] = board_temperature_h;     // High byte of board temperature (°C)
buffer[11] = board_temperature_l;     // Low byte of board temperature (°C)
buffer[12,13] = CRC16;                // CRC-16


/* GET N° OF POWER ON */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x08;                     // Response to g8 command
buffer[10] = power_on_n_h;            // High byte of number of lamp power on
buffer[11] = power_on_n_l;            // Low byte of number of lamp power on
buffer[12,13] = CRC16;                // CRC-16
```

```
/* GET LAMP LIFETIME (Hours) */

buffer[0] = 14;                       // Data frame payload length
buffer[1] = APP_DATA_FRAME;           // Data frame type
buffer[2,..7] = source_module.address;// Source device address (6 bytes)
buffer[8] = 0x67;                     // Get command 'g' ascii value
buffer[9] = 0x09;                     // Response to g9 command
buffer[10] = lamp_lifetime_h;         // High byte of lamp lifetime (hours)
buffer[11] = lamp_lifetime_l;         // Low byte of lamp lifetime (hours)
buffer[12,13] = CRC16;                // CRC-16


/* GET ALL PARAMETERS */

buffer[0] = 36;                        // Data frame payload length
buffer[1] = APP_DATA_FRAME;            // Data frame type
buffer[2,..7] = source_module.address; // Source device address (6 bytes)
buffer[8] = 0x67;                      // Get command 'g' ascii value
buffer[9] = 0x0a;                      // Response to g10 command
buffer[10] = hardware_release_xx;      // Hardware release xx.yy (xx, yy = bcd format)
buffer[11] = hardware_release_yy;      // Hardware release xx.yy (xx, yy = bcd format)
buffer[12] = bus_voltage_h;            // High byte of bus voltage before the failure
buffer[13] = bus_voltage_l;            // Low byte of bus voltage before the failure
buffer[14] = lamp_voltage_h;           // High byte of lamp voltage before the failure
buffer[15] = lamp_voltage_l;           // Low byte of lamp voltage before the failure
buffer[16] = hardware_status_h;        // High byte of hardware status before the
                                       failure
buffer[17] = hardware_status_l;        // Low byte of hardware status before the
                                       failure
buffer[18] = lamp_power_h;             // High byte of lamp power
buffer[19] = lamp_power_l;             // Low byte of lamp power
buffer[20] = last_hardware_status_h;   // High byte of hardware status before the
                                       failure
buffer[21] = last_hardware_status_l;   // Low byte of hardware status before the
                                       failure
```

```
buffer[22] = last_bus_voltage_h;      // High byte of bus voltage before the failure
buffer[23] = last_bus_voltage_l;      // Low byte of bus voltage before the failure
buffer[24] = last_lamp_voltage_h;     // High byte of lamp voltage before the failure
buffer[25] = last_lamp_voltage_l;     // Low byte of lamp voltage before the failure
buffer[26] = input_voltage_h;         // High byte of power supply voltage
buffer[27] = input_voltage_l;         // Low byte of power supply voltage
buffer[28] = board_temperature_h;     // High byte of board temperature (°C)
buffer[29] = board_temperature_l;     // Low byte of board temperature (°C)
buffer[30] = power_on_n_h;            // High byte of number of lamp power on
buffer[31] = power_on_n_l;            // Low byte of number of lamp power on
buffer[32] = lamp_lifetime_h;         // High byte of lamp lifetime (hours)
buffer[33] = lamp_lifetime_l;         // Low byte of lamp lifetime (hours)
buffer[34,35] = CRC16;                // CRC-16
```

If an error occurs from the HID board or from the network, the following frame throughout the COMM will be sent:

```
/* ERROR FRAME RESPONSE */

buffer[0] = 12;                          // Error frame payload length
buffer[1] = APP_ERROR_FRAME;             // Error frame type
buffer[4,5,6,7] = target_module.address; // Target device address (6 bytes)
buffer[8,9] = HID_error_code;            // HID board error code (2 bytes)
buffer[10,11] = CRC16;                   // CRC-16


/* HID_error_code value */

0x10 = HID response error
0x11 = HID command unknown
0x12 = HID board response timeout
0x13 = HID packet delivery impossible
```

Appendix B Schematic diagrams and bill of material

B.1 Schematic diagrams

Figure 12. Schematic diagram (1 of 5)

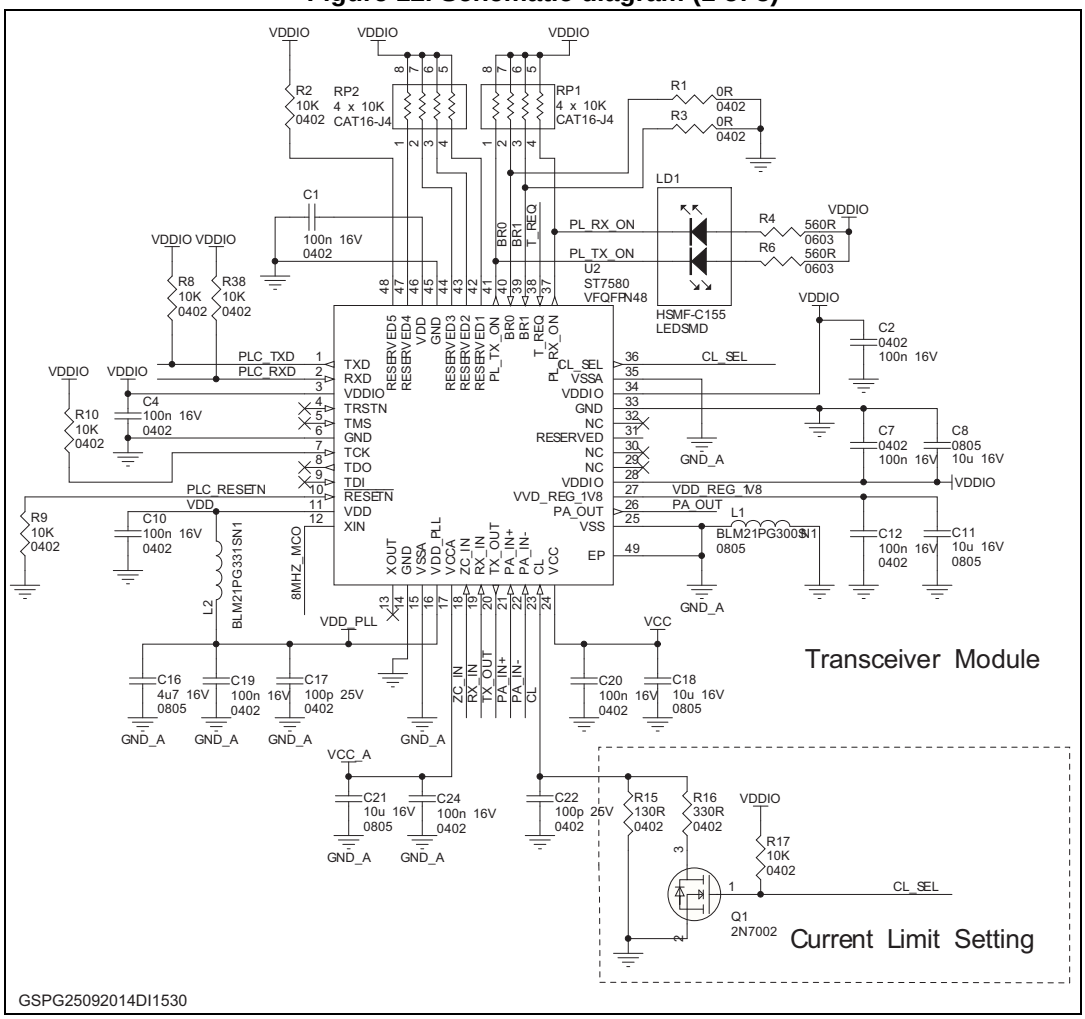
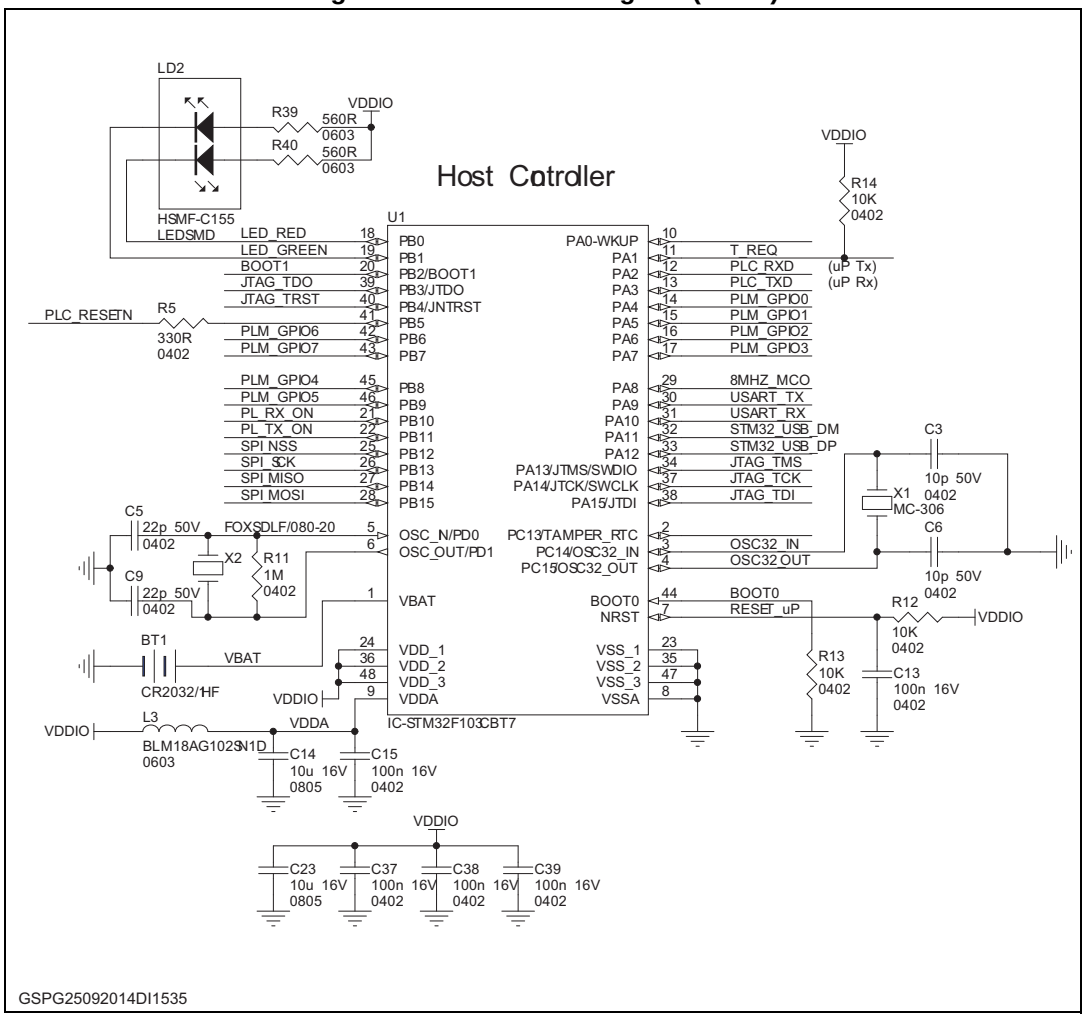
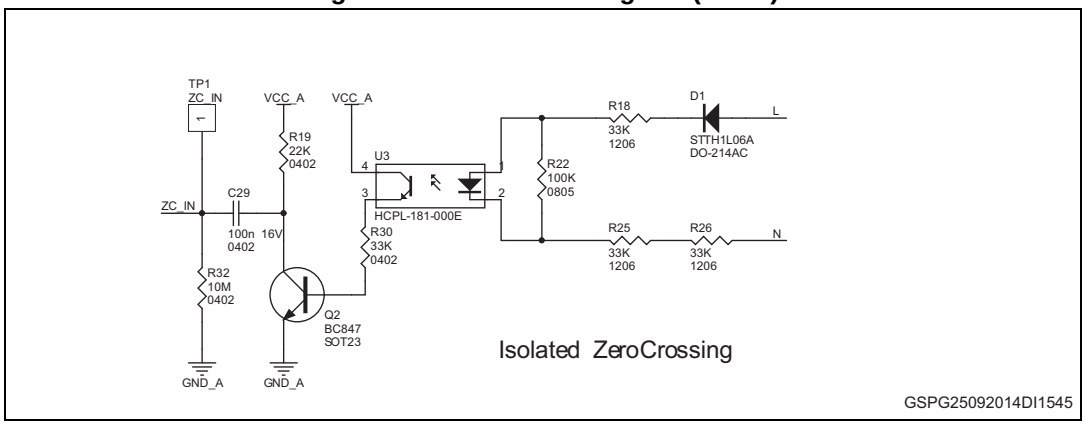


Figure 13. Schematic diagram (2 of 5)



GSPG25092014DI1535

Figure 14. Schematic diagram (3 of 5)



GSPG25092014DI1545

Figure 15. Schematic diagram (4 of 5)

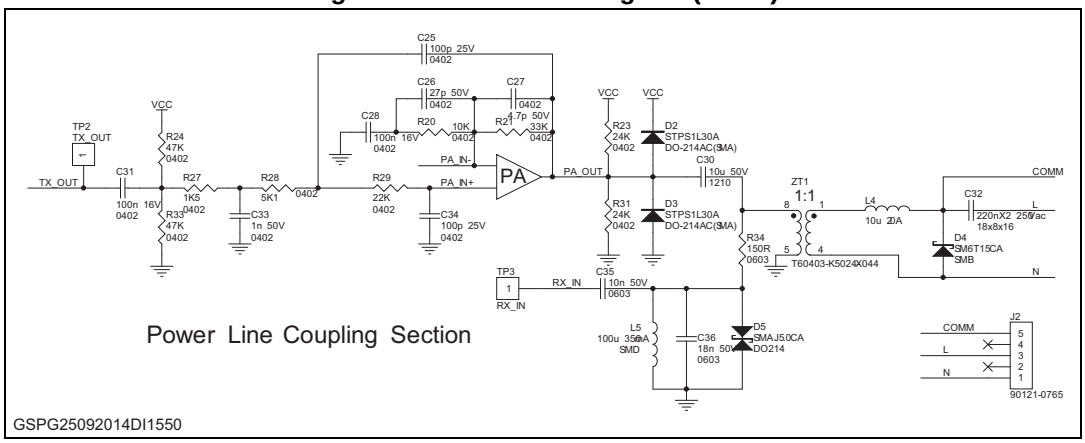
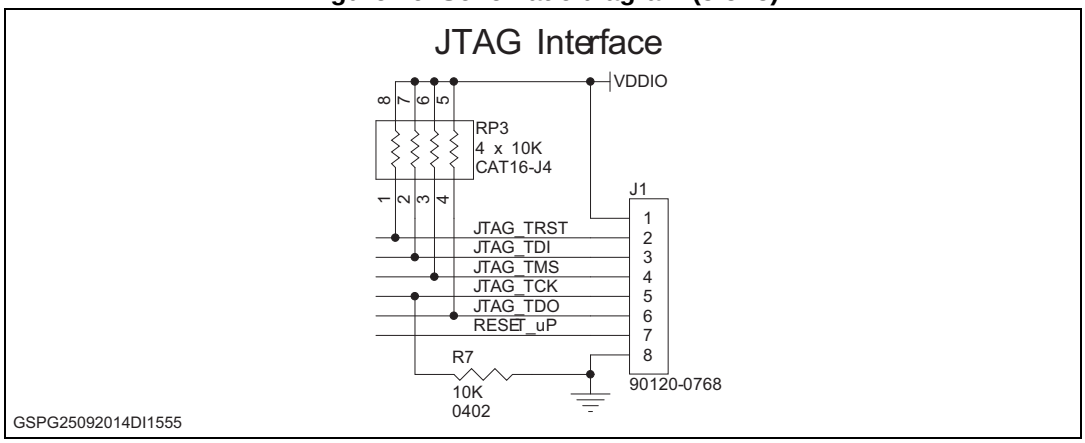


Figure 16. Schematic diagram (5 of 5)



B.2 Bill of material (BOM)

Table 1. Bill of material

Part type	MI code	Design.	Footprint	Description	Qty	Manufacturer	Order code	Part number	Order code 2
100n 16V	MI0156	C1, C2, C4, C7, C10, C12, C13, C15, C19, C20, C24, C28, C29, C31, C37, C38, C39	0402_C	Ceramic capacitor 100n 16V 0402	17	0402YG104ZAT2A	698-3190		
10p 50V	MI0207	C3, C6	0402_C	Ceramic capacitor 10p 50V X7R 5%	2	04025A100DAT2A	698-3065		
22p 50V		C5, C9	0402_C	Ceramic capacitor 22p 50V	2	GRM1555C1H220JZ 01D	624-2187		
10µ 16V	MI0136	C8, C11, C14, C18, C21, C23	0805S_C	Ceramic capacitor 10µ 16V X5R 10%	6	GRM21BR61C106K E15L	723-6039		
47µ 16V		C16	0805S_C	Ceramic capacitor 47µ 16V 0805 +/- 15%	1	GRM21BR61C475K A88L	723-6041		
100p 25V		C17, C22, C25, C34	0402_C	Ceramic capacitor 100p 25V 0402	4	04023A101JAT2A	698-3055		
27p 50V		C26	0402_C	Ceramic capacitor 27p 50V COG 0402	1	GRM1555C1H270JA 01D	723-5402		
4.7p 50V		C27	0402_C	Ceramic capacitor 4.7p 50V COG 0402	1	GRM1555C1H4R7B A01D	723-5436		
10µ 50V		C30	1210	Ceramic capacitor 10µ 50V 1210 X7R	1	UMK325BJ106MM-T	758-3486		

Table 1. Bill of material (continued)

Part type	MI code	Design.	Footprint	Description	Qty	Manufacturer	Order code	Part number	Order code 2
220n X2 250Vac		C32	RADX2-152	Metalized polyester film dielectric 220n 250Vac pitch 15mm	1	ECQU2A224ML	1198297		
1n 50V		C33	0402_C	Ceramic capacitor 1n 50V X7R 10%	1	04025C102KAT2A	698-3131		
10n 50V		C35	0603S_C	Ceramic capacitor 10n 50V X7R 10%	1	C0603C103K5RAC	264-4595		
18n 50V		C36	0603S_C	Ceramic capacitor 18n 50V	1	C0603C183K5RACTU	1865536RL	CL10B183KB8NNNC	766-5392
STTH1L06A	MI0020	D1	DO214AA	Diode fast recovery	1	STTH1L06A	8165505	STTH1L06A	485-8500
STPS1L30A		D2, D3	SMA_DIODE	Low drop power schottky rectifier 1A 30V 0.3V	2	STPS1L30	485-8061	STPS1L30A	497-6577-1-ND
SM6T15CA		D4	SMB	TRANSIL diode 15V 600W	1	SM6T15CA	436-9757		
SMAJ5.0CA		D5	DO214	Diode TVS bi-directional 5V 400W DO214AC	1	SMAJ5.0CA	764-5568	SMAJ5.0CA-TR	9802860RL
90121-0765		J2	SIP5_RA	Connector male 2.54mm pitch SIL RA	1	90121-0765			
010883121		J3, J4	FKV12LN_STRIP_MIRROR		2	010883121			
BLM21PG300SN1		L1	0805S	EMI filter 30R 100 MHz 0805	1	BLM21PG300SN1			
BLM21PG331SN1		L2	0805S	EMI filter 330R 100 MHz	1	BLM21PG331SN1	724-1545		

Table 1. Bill of material (continued)

Part type	MI code	Design.	Footprint	Description	Qty	Manufacturer	Order code	Part number	Order code 2
BLM21PG300SN1		L1	0805S	EMI filter 30R 100 MHz 0805	1	BLM21PG300SN1			
BLM21PG331SN1		L2	0805S	EMI filter 330R 100 MHz	1	BLM21PG331SN1	724-1545		
BLM18AG102SN1D	MI0160	L3	0603S	EMI filter BLM 1K 100MHz 400mA	1	BLM18AG102SN1D	230-3783		
10µ 2.0A		L4	IND_WE-PD3	Inductor 10µ 2.0A 74454010	1	74454010	74454010		
100µ 350mA		L5	IND_6x6	Inductor 100µ 350mA	1	B82462A4104K	495-8032		
HSMF-C155		LD1, LD2	HSMF-C655	LED Bi-color green/red SMD	2	HSMF-C155	486-0430		
2N7002	MI0127	Q1	SOT23	N-channel Trench MOSFET	1	2N7002	112-5526		
BC847	MI0135	Q2	SOT23A	NPN 45 V, 100 mA general-purpose transistors	1	BC847	445-2023		
10K	MI0167	R2, R7, R8, R9, R10, R12, R13, R14, R17, R20, R38	0402_R	Resistor 10K 0.0625W	11	CRG0402F10K	667-8848		
560R		R4, R6, R39, R40	0603S_R	Resistor 560R 0.1W 1%	4	CRG0603F560R	213-2238		
330R		R5, R16	0402_R	Resistor 330R 0.0625W	2	CRG0402F330R	667-8646		
1M		R11	0402_R	Resistor 1M 0.0625W	1	CRG0402F1M0	667-9065		
130R		R15	0402_R	Resistor 130R 0.0625W	1	CRG0402F130R	667-8618		
33K		R18, R25, R26	1206S_R	Resistor 33K 0.33W 1%	3	CRCW120633K0FKEA	679-2055		

Table 1. Bill of material (continued)

Part type	MI code	Design.	Footprint	Description	Qty	Manufacturer	Order code	Part number	Order code 2
22K		R19, R29	0402_R	Resistor 22K 0.0625W 1%	2	CRG0402F22K	667-8892		
33K		R21, R30	0402_R	Resistor 33K 0.0625W	2	CRG0402F33K	667-8933		
100K	MI0229	R22	0805S_R	Resistor 100K 0.125W 1%	1	CRG0805F100K	223-0691		
24K		R23, R31	0402_R	Resistor 24K 0.0625W 1%	2	CRG0402F24K	667-8909		
47K		R24, R33	0402_R	Resistor 47K 0.0625W 0.1%	2	CRG0402F47K	667-8943		
1K5		R27, R35	0402_R	Resistor 1K5 0.063W 1%	2	CRG0402F1K5	667-8707		
5K1		R28	0402_R	Resistor 5K1 0.0625W 1%	1	CRCW04025K10FKED	678-9478		
10M		R32	0402_R	Resistor 10M 0.0625W	1	CRCW040210M0FKED	667-8709		
150R		R34	0603S_R	Resistor 150R 0.1W 1%	1	CRG0603F150R	213-2165		
22R		R36, R37	0402_R	Resistor 22R 0.063W 1%	2	CRG0402F22R	667-8558		
4 x 10K	MI0069	RP1, RP2, RP3	RESPACK_CAT/CAY_16	Respack SMD 4 x 10K 0.25 5%	3	CAT16-103J4LF	522-5412	CAT16-103J4LF	1770124
IC-STM32F103CBT7		U1	LQFP48_STM	STM32 ARM-based 32-bit MCU with 64 kbytes Flash, 36-pin VFQFPN, industrial temperature	1	STM32F103CBT7	714-7689		

Table 1. Bill of material (continued)

Part type	MI code	Design.	Footprint	Description	Qty	Manufacturer	Order code	Part number	Order code 2
ST7580		U2	VFQFPN48_7x7x1	B-FSK, B-PSK, Q-PSK, 8-PSK Multi mode power line networking system on chip	1				
HCPL-181-000E		U3	SO-4	Phototransistor optocoupler SMD Mini-Flat type	1	HCPL-181	693-5277		
USBLC6-2	MI0211	U4	SOT666	USB very low capacitance ESD protection	1	USBLC6-2P6	624-7687		
MC-306		X1	TPSM32A	Crystal 32.768 kHz	1	MC-306	667-6117		
FOXSDLF/080-20		X2	XTAL2	SMD Crystal 8.000 MHz	1	8.000MHz 49USMX/30/50/-40+85/18pF/ATF	693-8869		
T60403-K5024-X044		ZT1	TRF_VAC5024X044	Signal transformer 1:1	1	T60403-K5024-X044	T60403-K5024-X044		

Appendix C CRC 16 calculation

The CRC 16 is based on the X16 + X15 + X2 + 1 polynomial.

```
/* Used CRC 16 table */

const uint16_t TableCRC16[256] = {
0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040
        };



/* CRC function */
```

```
/*******************************************************************************
******
* Function Name  : CalcCRC16
* Description    : Calculate a 16 bit CRC (X16 + X15 + X2 + 1)
* Input          : Buffer pointer, buffer length
* Return         : Calculated CRC
*******************************************************************************
*****/
uint16_t CalcCRC16(uint8_t *buf, uint8_t len)
{
  uint16_t crc = 0;

  while (len--)
    crc = (crc >> 8) ^ TableCRC16[(crc ^ (*(buf++))) & 0xff];

  return (crc);
}
```

5 Reference

1. ARM-based 32-bit MCU STM32F10x standard peripheral library Rel. 3.5.0 (2011)
2. ST7580 FSK, PSK multi-mode power line networking SoC datasheets (2012)
3. UM0932 ST7580 FSK, PSK multi-mode power line networking SoC user manual (2013)
4. AN4018 data link protocol for ST7580 PLM
5. IAR embedded workbench® IDE for STM32 microcontrollers Rel. 6.3 documentation (www.iar.com)

6 Revision history

Table 2. Document revision history

Date	Revision	Changes
29-May-2015	1	Initial release
29-Jul-2015	2	*Figure 4* has been updated
12-Dec-2016	3	Updated: - *Figure 1.: STEVAL-IHP007V1 evaluation board* - *Figure 2.: STEVAL-IHP007V1 block diagram*