AN4049
Application note

In-Home Display with HTTP gateway

Introduction

In this period of energy optimization and metering, there is an ever increasing demand for the possibility to perform in-house monitoring and management of energy use.

For this reason, STMicroelectronics offers a system which allows the monitoring and management of home energy consumption.

This system, "In-Home Display gateway", is made up of three main components:

- The STM322xG-EVAL demonstration board connected to an Ethernet network
- The STEVAL-IDZ001V1 adapter board connected on top of the STM322xG-EVAL board
- Remote web server.

Through this infrastructure, the system has the ability to publish the home automation device (i.e. smartplugs) energy data to a remote web server. The user can consult the energy consumption data on the web server from wherever they are, and at home, they can manage the entire system directly through an LCD touch panel.

This application note describes a smartplugW network system managed through the STM322xG-EVAL board.

The firmware and board allow the possibility to manage a ZigBee® network system, collecting ZigBee device information, and managing it through the LCD, touchscreen, pushbutton and joystick, and publishing the collected data to a HTTP server over the Internet by means of an Ethernet interface.

- *Section 1* describes the document and library rules.
- *Section 2* highlights the features of the ZigBee smartplug and explains its hardware interface with the microcontroller (STM32).
- *Section 3* describes briefly the "Multi-Input Embedded GUI Library".
- *Section 4* describes the relevant blocks of the STM322xG-EVAL demonstration board.
- *Section 5* shows the demonstration firmware/board system setup.
- *Section 6* describes how the "In-Home Display" firmware is structured, its architecture and its exported APIs in detail.
- *Section 7* explains how to get started with the system, how to configure and use the IAR workspace, and it contains an example application source code.
- *Section 8* illustrates how the "In-Home Display" GUI application works.
- *Section 10* illustrates the hardware schematics.

Contents

List of tables

List of figures

1 Document and library rules

This document uses the conventions described in the sections below.

1.1 Acronyms

Table 1 lists the acronyms used in this document.

Table 1. List of acronyms

Acronym	Meaning
API	Application programming interface
HAL	Hardware abstraction layer
MCU	Microcontroller unit
I^2C	Inter-integrated circuit
SPI	Serial to parallel interface
UART	Universal asynchronous receiver-transmitter
HTTP	HyperText transfer protocol

2 System description

The network configuration parameters (IP address, subnet mask, and default gateway) of the board as well as the IP address of the web server are stored in the dual interface EEPROM “M24LR64r” onboard, accessible from both the STM32W108CB and the STM32F2x7x via the I^2C interface and remotely by means of an RF reader. Therefore, it is possible to read and update parameters from inside (I^2C) and outside (RF) the application. In this way, an operator can easily change the network configuration of the board without any programming skills, and using just an RF reader.

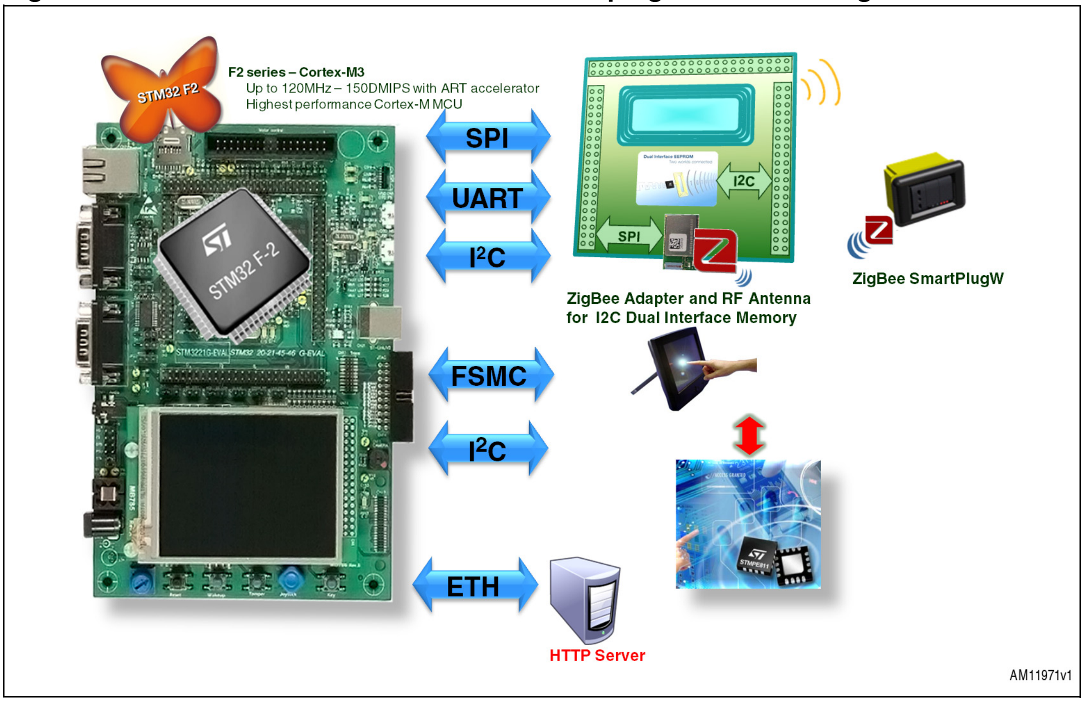
In order to use the STM322xG-EVAL board to interface with the ZigBee coordinator, a specific adapter has been developed. The ZigBee coordinator is connected to the demonstration board through the adapter that allows the UART communication. The two peers use a serial protocol to exchange data and commands.

The STM32W108CB is used as a co-processor of the STM32F2x7x, so the STM32F2x7x runs the application while the STM32W108CB runs the ZigBee stack and protocol. The two microcontrollers communicate by means of a UART interface.

An embedded graphical user interface (GUI) based on the “Multi-Input Embedded GUI Library 2.1" described in the document AN3128, and working on an LCD TFT 320 x 240 display and 5-position joystick, allows the user to interact with the smartplugW system made up of one coordinator and two smartplugs connected.

In *Figure 1* it is possible to take a quick look of the STM322xG-EVAL board and a smartplug node block diagram.

Figure 1. STM322xG-EVAL board and a smartplug node block diagram



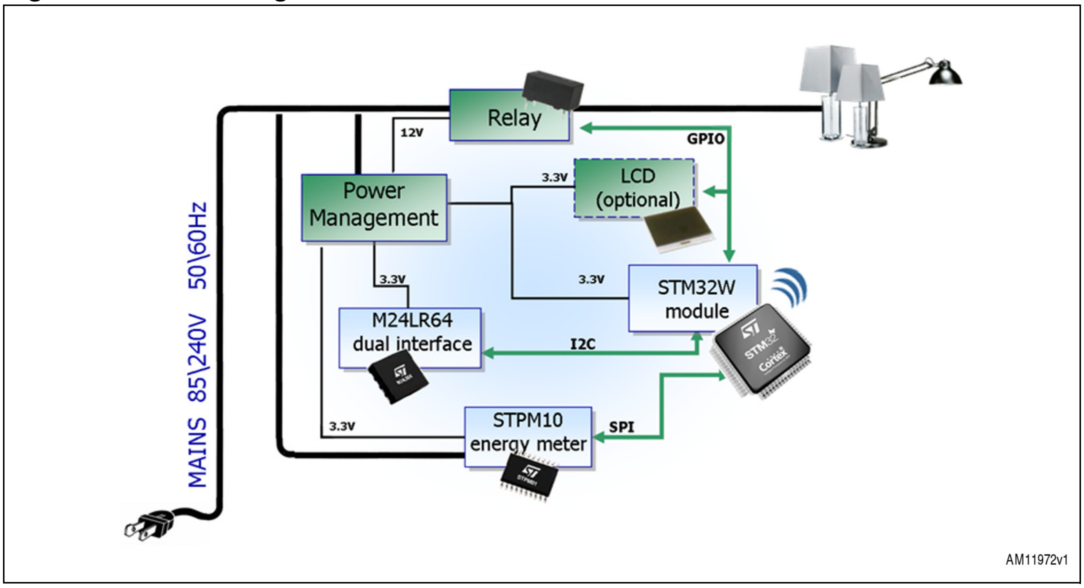
3 ZigBee smartplug

3.1 Smartplug description

The smartplug coordinator is connected via UART to the STM322xG-EVAL through an “ad-hoc” adapter. The adapter is connected to the extended connectors CN2, CN3 and CN4 on the STM322xG-EVAL (for more detailed information, please refer to the UM1065 user manual); it allows the connection of a ZigBee coordinator and the I2C/RF dual interface EEPROM M24LR64-r. The Gerber files of the adapter board are included in the setup package of this project.

The ZigBee smartplug board can be used as a guide to build a home/building automation subsystem for energy management. In a typical application, the board is plugged into an electrical wall socket and supplies an electrical load, monitoring the energy consumption; using several smartplugs it is possible to monitor and control the home/building energy consumption socket by socket.

Figure 2. Block diagram



The STEVAL-IHP004V1 is a smartplug board based on an STM32W108CB microcontroller, a SPZB32W1x2.1 module, and an STPM10 energy metering IC.

It implements a ZigBee metering node which allows the final user to monitor and manage energy consumption.

The board has been developed to provide a guide to build a home/building automation subsystem for energy management. In a typical home system implementation, the board is plugged into an electrical wall socket and supplies a home appliance or other generic electrical load.

3.2 ZigBee module SPZB32W1x2.1

ZigBee communication is based on the SPZB32W module. The SPZB32W is a low power consumption ZigBee module optimized for embedded applications. It enables OEMs to easily add wireless capability to electronic devices.

The module is FCC compliant (FCC ID: S9NZB32C2) and it is based on the STM32W108CB MCU which integrates a 2.4 GHz IEEE 802.15.4-compliant SMD module based on the ST single-chip STM32W108CB featuring:

- Integrated 2.4 GHz transceiver
- PHY and MAC IEEE 802.15.4 features
- Integrated ARM® Cortex-M3 core
- Integrated 128 kB embedded Flash and 8 kB embedded RAM
- Integrated encryption (AES-128) accelerator
- +3 dBm output power (+8 dBm in Boost mode)
- -99 dBm receiver sensitivity.

The module is controlled by means of a standard serial interface (SPI) allowing connection to a variety of host microcontrollers.

For further details, please refer to the SPZB32W module and the STM32W108CB datasheet.

For more information, see the user manual, the STEVAL-IHP004V1 schematics diagram, and the ZigBee smartplug firmware user manual.

4 Multi-Input Embedded GUI Library

4.1 Description

This solution enables users, comfortable with the use of standard microcontrollers, to create higher-end “look and feel” human interfaces by replacing conventional electromechanical switches with touch-sensing controls.

The user can combine touch-sensing functions using multiple configurations (touchscreen, joystick, and keys) with traditional MCU features (communication, LED control, beeper, LCD control, etc.).

The E-multi-input graphic library is part of the application firmware.

The graphic objects are a set of controls that can be printed on the screen and associated to an action when pressed.

The library has been developed and tested on an LCD panel of QWGA resolution (320x240) which is the default, but the library is independent of the LCD resolution, although it has not been tested with others.

The library supports touchscreen features and includes a low level driver which handles the analog input (for 12-bit ADC), and a function for the touchscreen calibration based on an algorithm that uses 5 points.

The Multi-Input Embedded GUI Library is fully developed in 'ANSI-C' following an OOP approach. This means that the final application uses instances of page and graphic objects, according to their public methods and properties. In the end, the PageObj is a structure containing public properties (data fields) and methods (functions pointers). The OOP encapsulation feature is assured. The library has been developed and tested on the STMicroelectronics demonstration boards.

The library can be included in the final application as a library file (multi-input embedded GUI Library.a) and used as a black box through its exported public API, or can be included in the final application as source files (.c and .h) if the user wants to debug the library itself, or if they want to change the HAL functions in order to port the library on a different LCD (in model and resolution) from the one attached to the ST demonstration board.

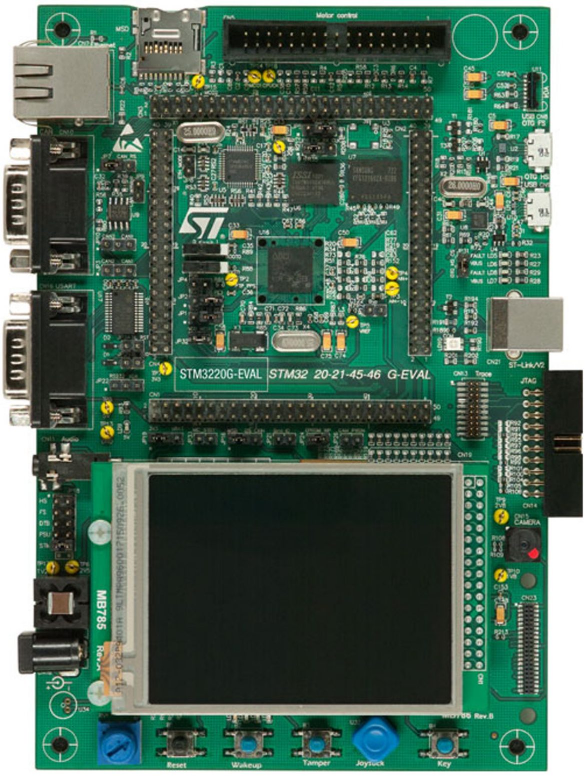
For more information on the graphic library, see the AN3128 rev.4 application note.

The calibration process is part of the post-processing layer. The touchscreen must be calibrated at first power-on and/or upon user request.

Once the calibration is done, final adjustments on future power-on of the board are not necessary because the calibration parameters are saved on the Flash memory.

5 STM3220G-EVAL demonstration board

Figure 3. STM3220G-EVAL board



AM11973v1

The STM3220G-EVAL demonstration board is a complete evaluation and development platform for the STM32 F-2 series and includes an embedded in STM32F207IGH6 high-performance ARM® Cortex™-M3 32-bit microcontroller.

The full range of hardware features on the board is provided to help the user evaluate all peripherals (USB OTG HS, USB OTG FS, Ethernet, motor control, CAN, MicroSD card, smartcard, USART, audio DAC, RS-232, IrDA, SRAM, MEMS, EEPROM… etc.) and develop their own applications. Extension headers make it possible to easily connect a daughterboard or wrapping board for specific applications.

The in-circuit ST-LINK tool can be easily used for JTAG and SWD interface debugging and programming.

5.1 Features

- STM32F207IGH6 microcontroller
- 16 Mbit SRAM
- 1 GByte or more MicroSD card
- Boot from user Flash, system memory or SRAM
- Both ISO/IEC 14443 type A and B smartcards support
- I^2C compatible serial interface 8 Kbytes EEPROM, MEMS and I/O expander
- IEEE 802.3-2002 compliant Ethernet connector
- Two CAN 2.0 A/B channels on the same DB connector
- RS-232 communication
- IrDA transceiver
- USB OTG (HS and FS) with Micro-AB connector
- Inductor motor control connector
- I2S audio DAC, stereo audio jack for headset
- 3.2" 240x320 TFT color LCD with touchscreen
- 4 color LEDs
- Camera module and extension connector for ST camera plug-in
- Joystick with 4-direction control and selector
- Reset, wakeup, tamper and user button
- RTC with backup battery
- Extension connector for daughterboard or wrapping board
- JTAG, SW and trace debug support
- Embedded ST-LINK/V2
- Five 5 V power supply options: Power jack, USB FS connector, USB HS connector, ST-LINK/V2 or daughterboard
- MCU consumption measurement circuit.

5.2 Hardware layout and configuration

The STM3220G-EVAL demonstration board is designed around the in STM32F207IGH6 microcontroller in the UFBGA176 package. The hardware block diagram, *Figure 2*, illustrates the connection between the in STM32F207IGH6 and peripherals (camera module, LCD, SRAM, EEPROM, MEMS, USART, IrDA, USB OTG HS, USB OTG FS, Ethernet, audio, CAN bus, smartcard, MicroSD card and motor control) and *Figure 3* helps to locate these features on the actual demonstration board.

Figure 4. STM3220G-EVAL board hardware block diagram

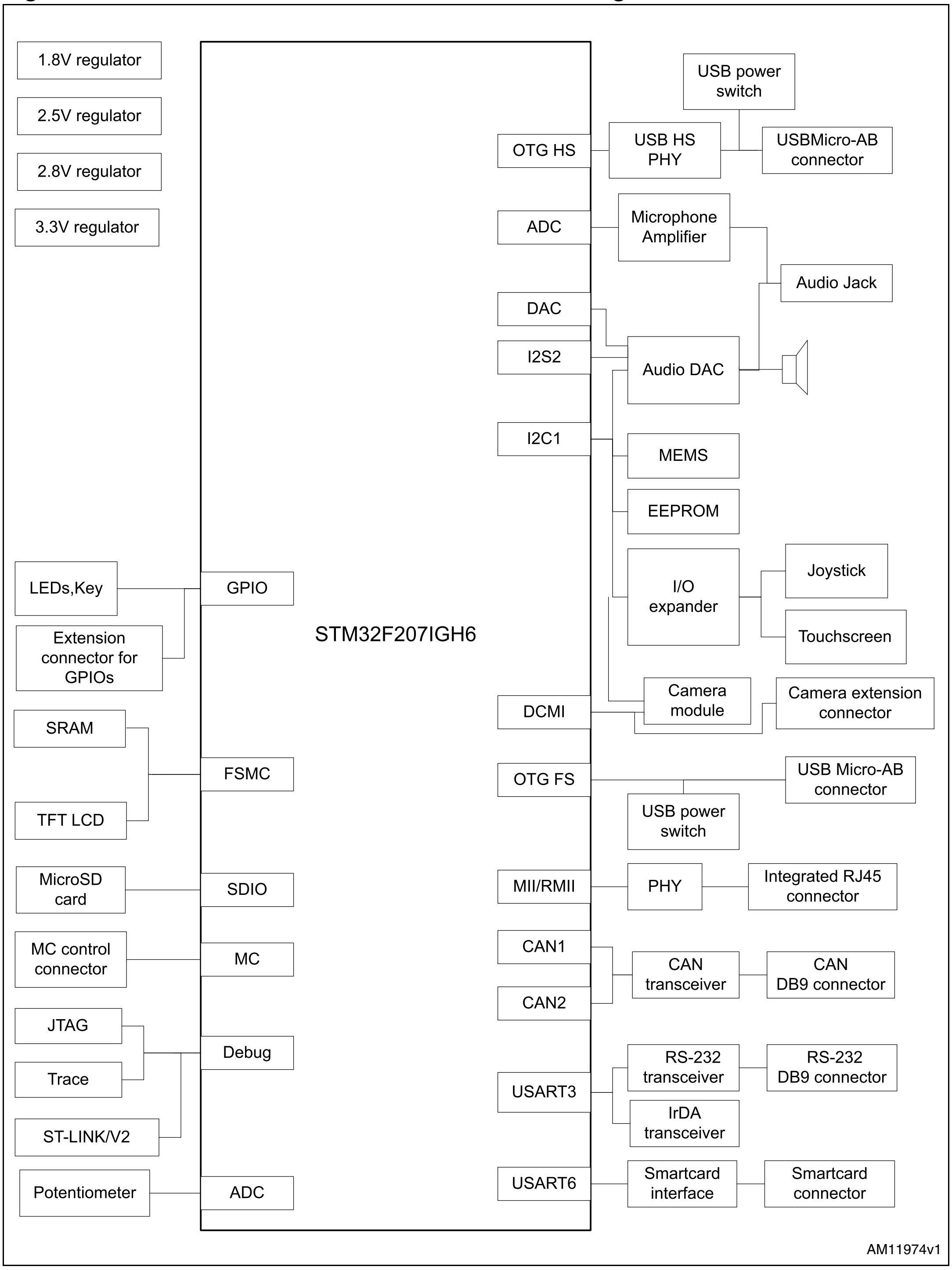


Figure 5. STM3220G-EVAL demonstration board layout

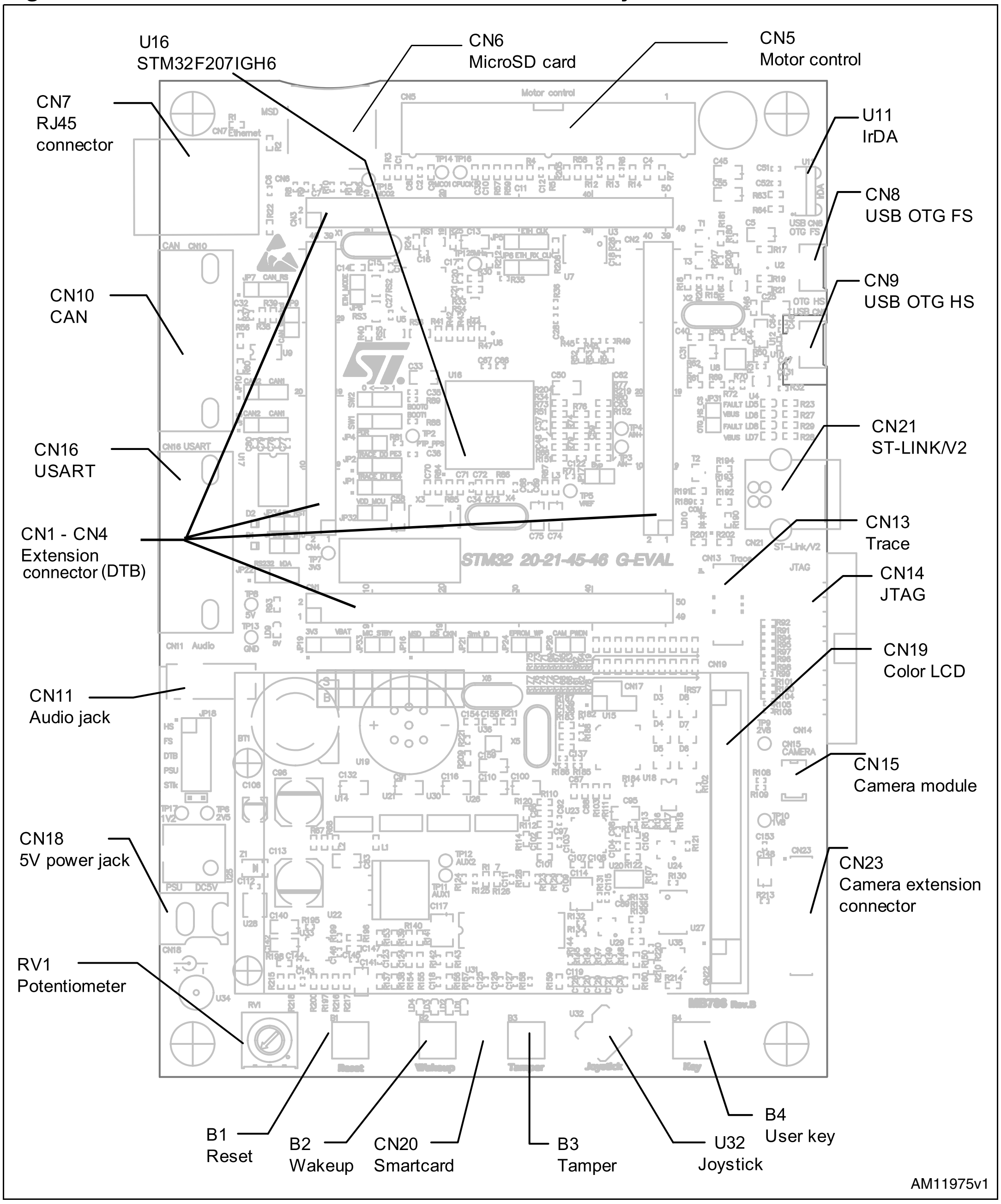
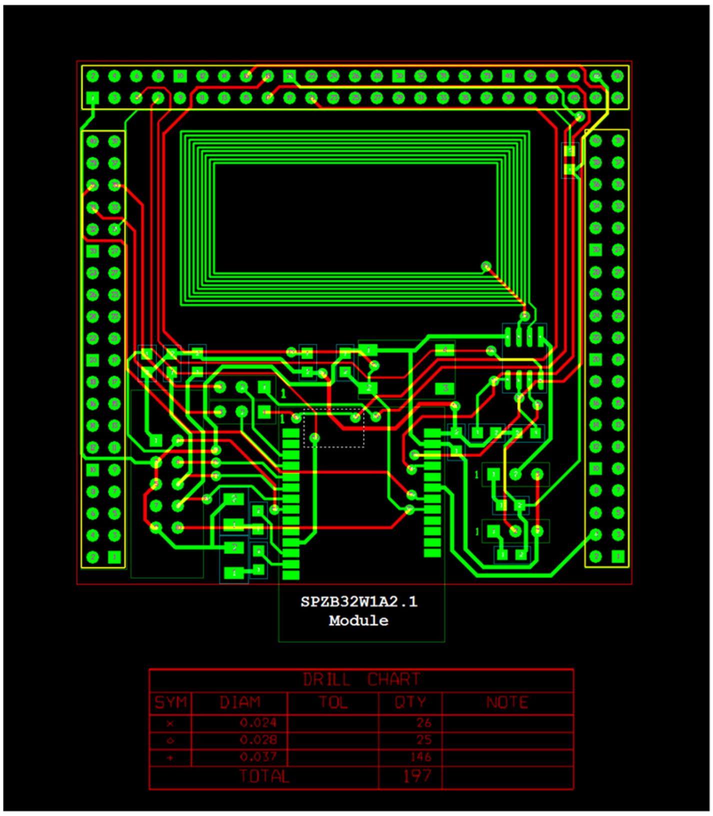


Figure 6. ZigBee adapter layout description



5.3 Power supply

The STM322xG-EVAL demonstration board is designed to be powered by a 5 V DC power supply and to be protected by PolyZen from a wrong power plug-in event. It is possible to configure the demonstration board to use any of the following five sources for the power supply:

- 5 V DC power adapter connected to JP18, the power jack on the board
- 5 V DC power with 500 mA limitation from CN8, the USB OTG FS Micro-AB connector
- 5 V DC power with 500 mA limitation from CN9, the USB OTG HS Micro-AB connector
- 5 V DC power with 500 mA limitation from CN21, the ST-LINK/V2 USB connector
- 5 V DC power from both CN1 and CN3, the extension connector for the daughterboard (DTB for daughterboard on silkscreen).

The power supply is configured by setting the related jumpers JP4, JP32, JP18 and JP19, as described in *Table 1*.

Table 2. Power related jumpers and solder bridges

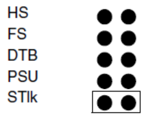
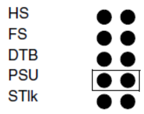
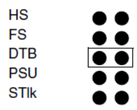
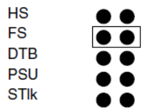
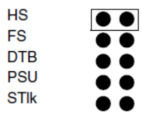
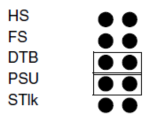
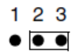
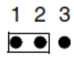
<table>
<tr><th>Jumper</th><th>Description</th></tr>
<tr><td>JP4</td><td>Jumper reserved for future use (RFU).
Default setting: fitted</td></tr>
<tr><td>JP32</td><td>MCU_VDD is connected to 3.3 V power when JP32 is closed and MCU current consumption measurement can be done manually by multi-meter when JP32 is open.
Default setting: fitted</td></tr>
<tr><td rowspan="6">JP18</td><td>JP18 is used to select one of the four possible power supply sources.
To select the ST-LINK/V2 USB connector (CN21) power supply, set JP18 as shown:
(default setting)
</td></tr>
<tr><td>For power supply jack (CN18) to the STM322xG-EVAL, JP18 is set as shown on the right: (default setting)
</td></tr>
<tr><td>To select daughterboard connector (CN1 and CN3) power supply, set JP18 as shown:
</td></tr>
<tr><td>To select USB OTG FS (CN8) power supply, set JP18 as shown:
</td></tr>
<tr><td>To select USB OTG HS (CN9) power supply, set JP18 as shown:
</td></tr>
<tr><td>To select power supply jack (CN18) power supply to both STM322xG-EVAL and daughterboard connected on CN1 and CN3, set JP18 as shown
Warning: daughterboard must not have its own power supply connected
</td></tr>
</table>

Table 2. Power related jumpers and solder bridges (continued)

Jumper	Description
JP19	To connect Vbat to the battery, set JP19 as shown: 
	To connect Vbat to 3.3 V power, set JP19 as shown: (default setting) 

Note: The LED LD9 is lit when the STM322xG-EVAL demonstration board is powered by the 5 V correctly.

5.4 Boot option

The STM322xG-EVAL demonstration board is able to boot from:

- Embedded user Flash
- System memory with boot loader for ISP
- Embedded SRAM for debugging.

The boot option is configured by setting switch SW1 (BOOT1) and SW2 (BOOT0). The BOOT0 can be configured also via RS232 connector CN16.

Table 3. Boot related switch

BOOT	BOOT1	Boot source
0	Do not care	STM322xG-EVAL boots from user Flash (default setting)
1	1	STM322xG-EVAL boots from embedded SRAM
1	0	STM322xG-EVAL boots from system memory

5.5 Clock source

Four clock sources are available on the STM3220G-EVAL demonstration board for the STM32F207IFT6 and RTC embedded.

- X1, 25 MHz crystal for Ethernet PHY with socket. It can be removed when clock is provided by the MCO pin of the MCU
- X2, 26 MHz crystal for USB OTG HS PHY
- X3, 32 kHz crystal for embedded RTC
- X4, 25 MHz crystal with socket for the in STM32F207IGH6 microcontroller (it can be removed from socket when internal RC clock is used).

5.6 Reset source

The reset signal of the STM322xG-EVAL demonstration board is low active and the reset sources include:

- Reset button B1
- Debugging tools from JTAG connector CN14 and trace connector CN13
- Daughterboard from CN3
- RS232 connector CN16 for ISP
- ST-LINK/V2.

5.7 EEPROM

A 64 KBit EEPROM is connected to the I2C1 bus of the in STM32F207IGH6.

Table 4. EEPROM related jumper and solder bridge

Jumper	Description
JP24	The EEPROM is in Write Protection mode when JP24 is fitted. Default setting: not fitted

5.8 Ethernet

The STM322xG-EVAL demonstration board supports 10M/100M Ethernet communication by a PHY DP83848CVV (U5) and integrated RJ45 connector (CN7). Both MII and RMII interfaces can be selected by setting jumpers JP5, JP6 and JP8, as listed below:

Table 5. Ethernet related jumpers and solder bridges

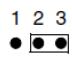
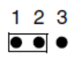
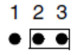
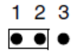
Jumper	Description
JP8	JP8 is used to select MII or RMII interface mode. To enable MII, JP8 is not fitted. To enable RMII interface mode, JP8 is fitted. Default setting: not fitted
JP6	To enable MII interface mode, set JP6 as shown (default setting): 
	To enable RMII interface mode, set JP6 as shown: 

Table 5. **Ethernet related jumpers and solder bridges (continued)**

Jumper	Description
JP5	To provide 25 MHz clock for MII or 50 MHz clock for RMII by MCO at PA8, set JP5 as shown (default setting): 
	To provide 25 MHz clock by external crystal X1 (for MII interface mode only) set JP5 as shown: 
	When clock is provided by external oscillator U3, JP5 must not be fitted (default setting).
SB1	SB1 is used to select clock source only for RMII mode. To connect the clock from MCO to RMII_REF_CLK, close SB1. The resistor R212 must be removed in this case. Default setting: open

Note: 1 *A test point (TP2) is available on the board for the PTP_PPS feature test.*

2 *The Ethernet PHYU5 can be powered down by regulating PB14.*

3 *In RMII mode it is not possible to use MCO to output the 50 MHz clock to PHY due to the PLL limitation explained in chapter 2.6.5 of STM32F2x7x and STM32F2x7x errata sheet (ES0005). In such a case it is possible to provide the 50 MHz clock by soldering a 50 MHz oscillator (ref SM7745HEV-50.0M or equivalent) on the U3 footprint located under CN3 and also removing jumper on JP5. This oscillator is not provided with the board.*

5.9 SRAM

The 16 Mbit SRAM is connected to the FSMC bus of the in STM32F207IGH6 which shares the same I/Os with the CAN1 bus. JP3 and JP10 must not be fitted for SRAM and LCD application.

Table 6. **SRAM related jumpers**

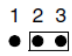
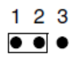
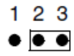
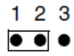
Jumper	Description
JP1	Connect PE4 to SRAM as A20 by setting JP1 as shown (default setting): 
	Connect PE4 to trace connector CN13 as TRACE_D1 by setting JP1 as shown: 

Table 6. **SRAM related jumpers (continued)**

Jumper	Description
JP2	Connect PE3 to SRAM as A19 by setting JP2 as shown (default setting): 
	Connect PE3 to trace connector CN13 as TRACE_D0 by setting JP2 as shown: 

5.10 Development and debug support

The version 2 of the ST-LINK, called ST-LINK/V2, is embedded on the board. This tool allows onboard program loading and debugging of the STM32F using the JTAG or SWD interface.

Third-party debug tools are also supported by the JTAG (CN14) or trace (CN13) connectors.

To communicate with the embedded ST-LINK/V2, a specific driver needs to be installed on the PC. To download and install this driver, refer to the software and development tools page for the STM32F family available on www.st.com (the install shield is called ST-LINK_V2_USBdriver.exe).

The embedded ST-LINK/V2 connects to the PC via a standard USB cable from connector CN21. The bicolor LED LD10 (COM) indicates the status of the communication as follows:

- Slow blinking red/off: at power-on before USB initialization
- Fast blinking red/off: after the first correct communication between PC and ST-LINK/V2 (enumeration)
- Red LED on: when initialization between PC and ST-LINK/V2 is successfully finished
- Green LED on: after successful target communication initialization
- Blinking red/green: during communication with target
- Green on: communication finished and OK
- Orange on: communication failure.

Note: It is possible to power the board via CN21 (embedded ST-LINK/V2 USB connector) even if an external tool is connected to CN13 (trace) or CN14 (external JTAG and SWD).

5.11 Display and input devices

The 3.2" TFT color LCD connected to the FSMC bus and 4 general purpose color LEDs (LED 1, 2, 3, 4) are available as display devices. A touchscreen connected to an I/O expander (U24), 4-direction joystick with selection key, general purpose button (B4), wakeup button (B2) and tamper detection button (B3) are available as input devices.

Table 7. LCD modules

3.2” TFT LCD with touchscreen (CN19)		
Pin on CN19	Pin name	Pin connection
1	CS	FSMC_NE3 (PG10)
2	RS	FSMC_A0
3	WR/SCL	FSMC_NWE
4	RD	FSMC_NOE
5	RESET	RESET#
6	PD1	FSMC_D0
7	PD2	FSMC_D1
8	PD3	FSMC_D2
9	PD4	FSMC_D3
10	PD5	FSMC_D4
11	PD6	FSMC_D5
12	PD7	FSMC_D6
13	PD8	FSMC_D7
14	PD10	FSMC_D8
15	PD11	FSMC_D9
16	PD12	FSMC_D10
17	PD13	FSMC_D11
18	PD14	FSMC_D12
19	PD15	FSMC_D13
20	PD16	FSMC_D14
21	PD17	FSMC_D15
22	BL_GND	GND
23	BL_Control	+5 V
24	VDD	+3.3 V
25	VCI	+3V3
26	GND	GND
27	GND	GND
28	BL_VDD	+5V
29	SDO	NC
30	SDI	NC
31	XL	IO expander U24
32	XR	IO expander U24

Table 7. LCD modules (continued)

3.2” TFT LCD with touchscreen (CN19)		
Pin on CN19	**Pin name**	**Pin connection**
33	YD	IO expander U24
34	YU	IO expander U24

5.12 Daughterboard extension connector CN1, 2, 3 and CN4

Four male headers CN1, 2, 3 and CN4 can be used to connect the daughterboard or the standard wrapping board to the STM322xG-EVAL demonstration board. A total number of 140 GPIOs is available on the board.

Each pin on CN1, 2, 3 and 4 can be used by a daughterboard after disconnecting it from the corresponding function block on the STM322xG-EVAL demonstration board.

The ZigBee adapter is connected to the extended connectors CN2 and CN4 on the STM322xG-EVAL (for more detailed information, please refer to *Table 8* to *Table 10*), it allows the communication between the STM32F2x7x on the eval board, the ZigBee smartplug coordinator (STM32W108CB) and the I2C/RF dual interface EEPROM M24LR64-r on the ZigBee adapter.

Table 8. Daughterboard extension connector CN1

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
1	GND	-	-
3	PE3	Trace_D0 and FSMC_A19	Keep JP2 on 2<->3
5	PE5	Trace_D2	
7	PI8	LCD_HSYNC	-
9	PC14	OSC32_IN	Remove R84, SB4 closed
11	PC15	OSC32_OUT	Remove R85, SB5 closed
13	PI10	MII_RX_ER	Remove RS3
15	PF0	FSMC_A0	-
17	PF2	FSMC_A2	-
19	GND	-	-
21	PF5	FSMC_A5	-
23	PF7	SmartCard_RST	-
25	PF9	Potentiometer	Remove R151
27	PH0	OSC_IN	SB6 open
29	PC0	ULPI_STP	-
31	PC1	MII_MDC	SB11 open
33	PC3	MII_TX_CLK	Remove R51

Table 8. Daughterboard extension connector CN1 (continued)

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
35	PA0	WakeUP	Remove R139
37	PA2	MII_MDIO	-
39	GND	-	-
41	PH4	ULPI_NXT	Remove R61
43	NC	-	-
45	NC	-	-
47	EMU_3V3	-	-
49	EMU_5V	-	-
2	PE2	Trace_CLK	-
4	PE4	Trace_D1 & FSMC_A20	Keep JP1 on 2<->3
6	PE6	Trace_D3	-
8	PC13	Anti-Tamper	Remove R143
10	GND	-	-
12	PI9	LED3	Remove R141
14	PI11	ULPI_DIR	Remove R62
16	PF1	FSMC_A1	-
18	PF3	FSMC_A3	-
20	PF4	FSMC_A4	-
22	PF6	SmartCard_OFF	Remove R126
24	PF8	LCD_CS	
26	PF10	Audio_IN	Remove R196
28	PH1	OSC_OUT	Remove R86, SB7 closed
30	GND	-	-
32	PC2	MII_TXD2 & MC	SB12 open
34	VREF+	-	-
36	PA1	MII_RX_CLK	JP6 open
38	PH2	MII_CRS	Remove RS3
40	PH3	MII_COL	Remove RS3
42	PH5	OTG_FS_PowerSwitchOn	Remove R18
44	NC	-	-
46	NC	-	-
48	APP_3V3	-	-
50	GND	-	-

Table 9. Daughterboard extension connector CN2

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
1	GND	-	-
3	PA3	ULPI_D0	-
5	PA5	ULPI_CLK	Remove R69
7	PA7	MII_RX_DV	Remove RS2, JP8 open
9	PC5	MII_RXD1	Remove R58
11	PB0	ULPI_D1	-
13	PB2	BOOT1	-
15	PF12	FSMC_A6	-
17	PF14	FSMC_A8	-
19	GND	-	-
21	PG1	FSMC_A11	-
23	PE8	FSMC_D5	-
25	PE10	FSMC_D7	-
27	PE12	FSMC_D9	-
29	PE14	FSMC_D11	-
31	PE15	FSMC_D12	-
33	PB11	ULPI_D4	-
35	PH7	MII_RXD3	Remove RS3
37	PH9	DCMI_D0	Remove camera module from CN15
39	GND	-	-
2	APP_VCC	-	-
4	PA4	Audio_DAC_OUT	Remove R115
6	PA6	DCMI_PIXCK	Remove camera module from CN15
8	PC4	MII_RXD0	Remove RS2
10	GND	-	-
12	PB1	ULPI_D2	-
14	PF11	OTG_FS_Overcurrent	Remove R15
16	PF13	FSMC_A7	-
18	PF15	FSMC_A9	-
20	PG0	FSMC_A10	-
22	PE7	FSMC_D4	-
24	PE9	FSMC_D6	-
26	PE11	FSMC_D8	-

Table 9. Daughterboard extension connector CN2 (continued)

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
28	PE13	FSMC_D10	-
30	GND	-	-
32	PB10	ULPI_D3	-
34	PH6	MII_RXD2	Remove RS5
36	PH8	DCMI_HSYNC & MC	Remove camera module from CN15. Disconnect motor control board from CN5
38	PH10	DCMI_D1 &MC	Remove camera module from CN15. Disconnect motor control board from CN5
40	PH11	DCMI_D2 &MC	Remove camera module from CN15. Disconnect motor control board from CN5

Table 10. Daughterboard extension connector CN3

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
1	GND	-	-
3	PI1	I2S_CK	-
5	PH15	SmartCard_3/5 V & MC	Disconnect motor control board from CN5
7	PH13	MicroSDCard_defect & MC	Remove MicroSD card from CN6. Disconnect motor control board from CN5
9	PC13	Anti-Tamper	Remove R143
11	RESET#	Reset button	-
13	PA11	OTG_FS_DM	Remove R17
15	PA9	VBUS_FS	Remove USB cable from CN8 Remove R18
17	PC9	MicroSDCard_D1 & I2S_CKIN	Keep JP16 on open
19	EMU_5V	-	-
21	PC6	I2S_MCK & SmartCard_IO	JP21 open
23	PG7	SmartCard_CLK	-
25	PG5	FSMC_A15	-
27	PG3	FSMC_A13	-
29	PD15	FSMC_D1	-
31	PD14	FSMC_D0	-

Table 10. Daughterboard extension connector CN3 (continued)

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
33	PD12	FSMC_A17	SB15 open
35	PD10	FSMC_D15	-
37	PD8	FSMC_D13	-
39	GND	-	-
41	PB13	ULPI_D6 & CAN2_TX	-
43	PH12	DCMI_D3 & MC	Remove camera module from CN15. Disconnect motor control board from CN5
45	NC	-	-
47	EMU_3V3	-	-
49	EMU_5V	-	-
2	PI2	IO_Expandor_INT	Remove R136
4	PI0	I2S_CMD	-
6	PH14	DCMI_D4 & MC	Remove camera module from CN15. Disconnect motor control board from CN5
8	PA13	TMS/SWDIO	-
10	GND	-	-
12	PA12	OTG_FS_DP	Remove R19
14	PA10	OTG_FS_ID	Remove R21
16	PA8	MCO	JP5 open
18	PC8	MicroSDCard_D0 & MC	Remove MicroSD card from CN6 disconnect motor control board from CN5
20	PC7	LED4	Remove R140
22	PG8	LED2	Remove R154
24	PG6	LED1	Remove R155
26	PG4	FSMC_A14	-
28	PG2	FSMC_A12	-
30	GND	-	-
32	PD13	FSMC/MC	SB14 open. Disconnect motor control board from CN5.
34	PD11	FSMC_A16	-
36	PD9	FSMC_D14	-
38	PB15	OneNAND_INT	Remove R53
40	PB14	MII_INT	Remove R41
42	PB12	ULPI_D5	-

Table 10. Daughterboard extension connector CN3 (continued)

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
44	NC	-	-
46	NC	-	-
48	APP_3V3	-	-
50	GND	-	-

Table 11. Daughterboard extension connector CN4

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
1	GND	-	-
3	PI6	DCMI_D6 & MC	Remove camera module from CN15 Disconnect motor control board from CN5
5	PI4	DCMI_D5 & MC	Remove camera module from CN15 SB16 open
7	PE0	FSMC_BL0	-
9	PB8	MII_TXD3 & MC	Remove RS5 SB10 open Disconnect motor control board from CN5
11	BOOT0	BOOT0	-
13	PB6	I2C1_SCK	Remove R103
15	PB4	TRST	-
17	PG15	User button	Remove R150
19	GND	-	-
21	PG12	Smartcard_CMDVCC & CLD_VSYNC	Remove R128
23	PG10	FSMC_NE3	Remove LCD board MB785 from CN19
25	PD7	FSMC_NE1	Remove R52
27	PD5	FSMC_NWE	-
29	PD3	FSMC_CLK	-
31	PD2	MicroSDCard_CMD	-
33	PD0	FSMC_D2 & CAN1_RX	JP10 open
35	PC11	MicroSDCard_D3 & RS232/IrDA_RX	JP22 open. Remove MicroSD card from CN6
37	PA15	TDI	-
39	GND	-	-

Table 11. Daughterboard extension connector CN4 (continued)

Pin	Description	Alternative function	How to disconnect with function block on STM322xG-EVAL board
2	PI7	DCMI_D7 & MC	Remove camera module from CN15. Disconnect motor control board from CN5
4	PI5	DCMI_VSYNC & MC	Remove camera module from CN15. Disconnect motor control board from CN5
6	PE1	FSMC_BL1	-
8	PB9	I2C1_SDA	Remove R111
10	GND	-	-
12	PB7	FSMC_NL	-
14	PB5	ULPI_D7 & CAN2_RX	JP10 open
16	PB3	TDO/SWO	-
18	PG14	MII_TXD1	Remove RS6
20	PG13	MII_TXD0	Remove RS6
22	PG11	MII_TX_EN	Remove RS6
24	PG9	FSMC_NE2	Remove R47
26	PD6	FSMC_NWAIT	Remove R54
28	PD4	FSMC_NOE	-
30	GND	-	-
32	PD1	FSMC_D3 &CAN1_TX	JP3 open
34	PC12	MicroSDCard_CLK	Remove MicroSD card from CN6
36	PC10	MicroSDCard_D2 & RS232/IrDA_TX	Remove MicroSD card from CN6
38	PA14	TCK/SWCLK	-
40	PI3	I2S_DIN	-

5.13 Ethernet RJ45 connector CN7

Figure 7. Ethernet RJ45 connector CN7

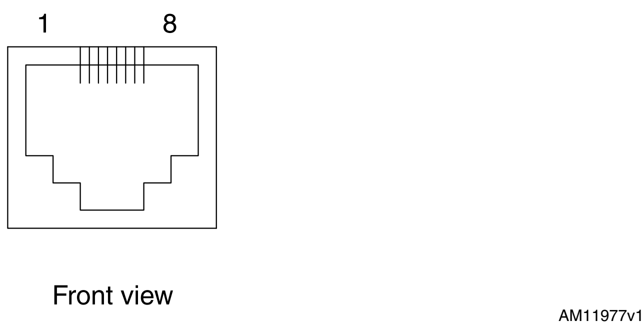


Table 12. Ethernet connector pinout

Pin number	Description	Pin number	Description
1	TxData+	2	TxData-
3	RxData+	4	Shield
5	Shield	6	RxData-
7	Shield	8	Shield

5.14 JTAG debugging connector CN14

Figure 8. JTAG debugging connector CN14

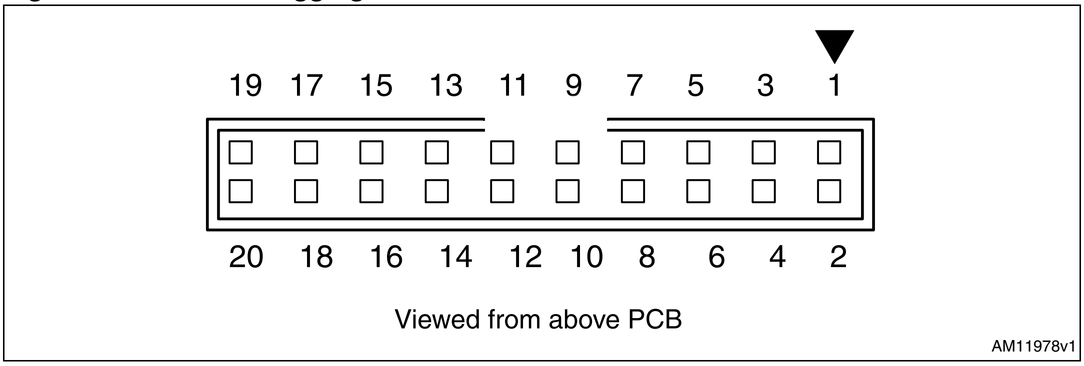


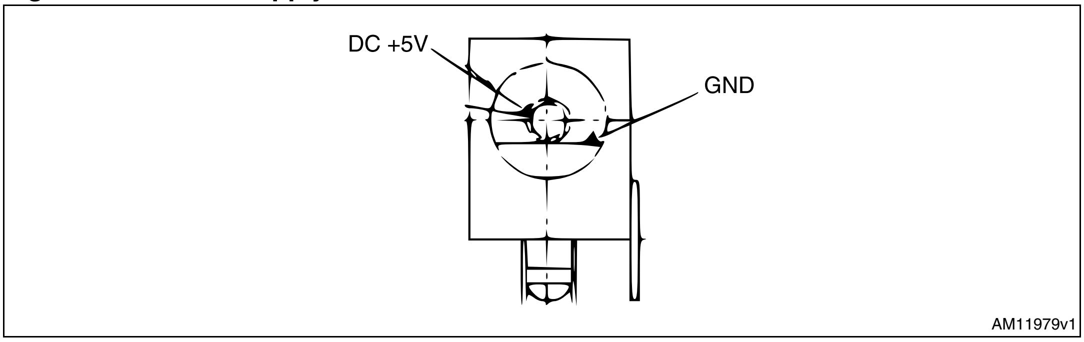
Table 13. JTAG debugging connector CN14

Pin number	Description	Pin number	Description
1	3.3 V power	2	3.3 V power
3	PB4	4	GND
5	PA15	6	GND
7	PA13	8	GND
9	PA14	10	GND
11	RTCK	12	GND
13	PB3	14	GND
15	RESET#	16	GND
17	DBGRQ	18	GND
19	DBGACK	20	GND

5.15 Power connector CN18

The STM322xG-EVAL demonstration board can be powered from a DC 5 V power supply via the external power supply jack (CN18) shown in *Figure 6*. The central pin of CN18 must be positive.

Figure 9. Power supply connector CN18



5.16 TFT LCD connector CN19

One 34-pin male header CN19 is available on the board for connecting the LCD module board MB785. Please refer to *Section 5.11* for details.

6 In-Home Display with HTTP gateway firmware

6.1 Firmware structure

Figure 10 shows the layers architecture of the firmware implementation.

Figure 10. Firmware architecture

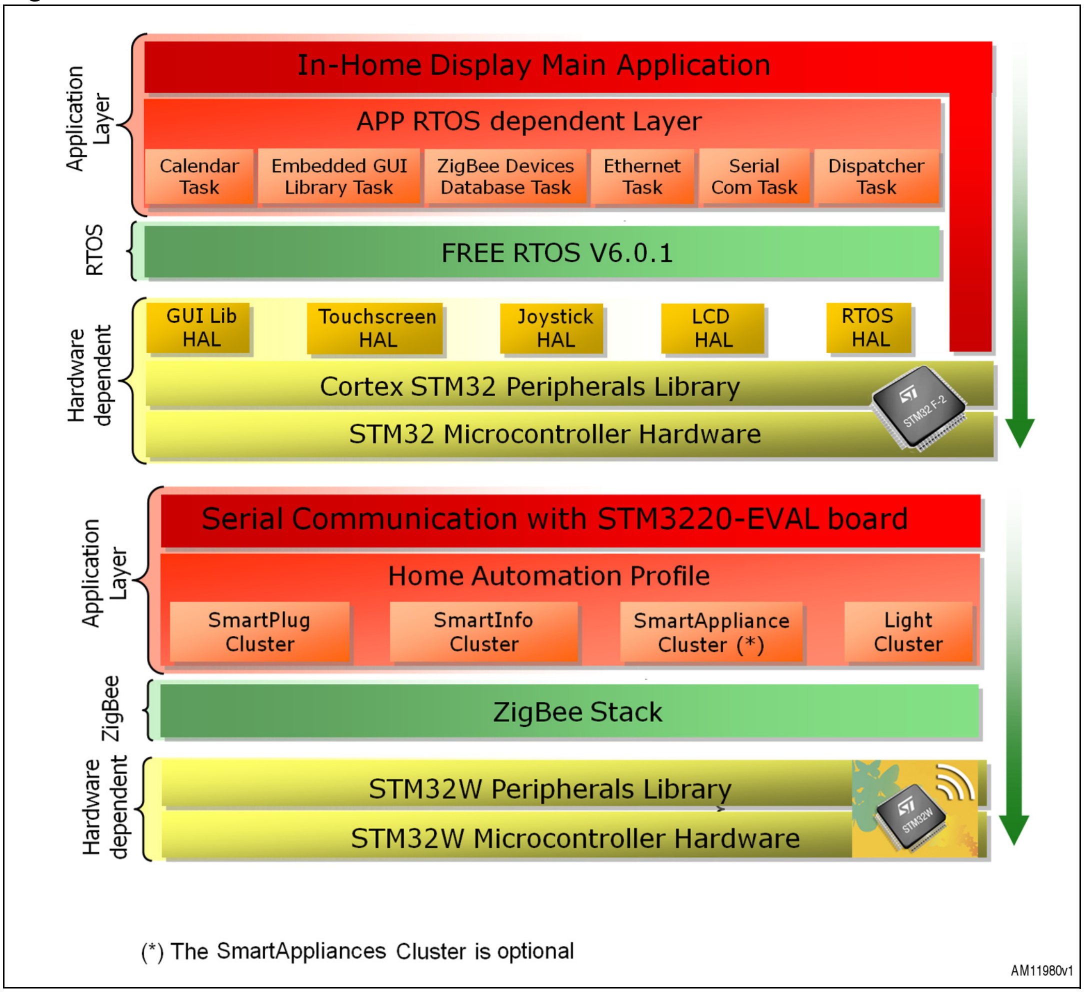
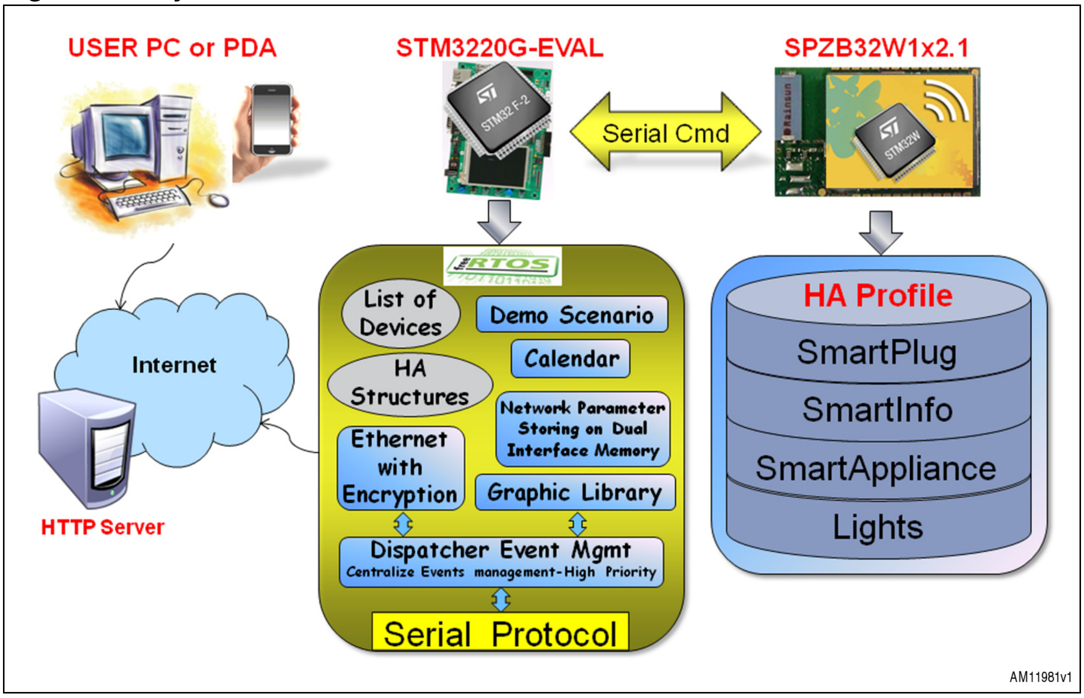


Figure 11 describes the system architecture and the hardware block interaction.

Figure 11. System architecture and hardware block interaction



This section describes the firmware implementation. The tasks/functions are described in the following format:

Table 14. Function description format

Function name	The name of the function
Function prototype	Prototype declaration of the function
Behavior description	Brief explanation of how the function is executed
Input parameter {x}	Description of the input parameters
Output parameter {x}	Description of the output parameters
Return value	Value returned by the function
Required preconditions	Requirements before calling the function
Called functions	Other library functions called

6.2 main.c

6.2.1 demo_initTask

Table 15 describes the demo_initTask task:

Table 15. demo_initTask task

Function name	demo_initTask
Function prototype	void demo_initTask (void *pvParameters)
Behavior description	Initialize the visualization environment, the RTC and check the touchscreen for the calibration
Input parameter {x}	None
Output parameter {x}	None
Return value	None
Required preconditions	None
Called functions	No API/HAL layer functions;

Example:

```
static void demo_initTask (void *pvParameters)
{
  #if TOUCH_SCREEN_CAPABILITY
    /* Check if Calibration has been done*/
    TS_CheckCalibration();
  #endif

  CursorInit(GL_NULL);

  /* Set the LCD Back Color */
  GL_SetBackColor(GL_White);
  /* Set the Font Text Color */
  GL_SetTextColor(GL_Black);
  GL_Clear(GL_White);
  GL_DrawButtonBMP( 210, 110,
                   (LCD_Height/10)*2 + 100,
                   (LCD_Height/10)*2,
                   (uint8_t*) STM32Logo);
  GL_DisplayAdjStringLine(3*(LCD_Height/5),(LCD_Width/3)*2 + 6,
                          "Loading.", GL_FALSE);
  GL_Delay(30);
```

```
  GL_DisplayAdjStringLine(3*(LCD_Height/5), (LCD_Width/3)*2 + 6,
                          "Loading..", GL_FALSE);
  GL_Delay(30);
  GL_DisplayAdjStringLine(3*(LCD_Height/5), (LCD_Width/3)*2 + 6,
                          "Loading...", GL_FALSE);
  GL_Delay(30);
  GL_DisplayAdjStringLine(3*(LCD_Height/5), (LCD_Width/3)*2 + 6,
                          "Loading.   ", GL_FALSE);
  GL_Delay(30);
  GL_DisplayAdjStringLine(3*(LCD_Height/5), (LCD_Width/3)*2 + 6,
                          "Loading..", GL_FALSE);
  GL_Delay(30);
  GL_DisplayAdjStringLine(3*(LCD_Height/5), (LCD_Width/3)*2 + 6,
                          "Loading...", GL_FALSE);
  GL_Delay(20);

  RTC_Settings();

  CursorShow(195, 50);

  vTaskResume(xGraphLibTaskHandle);

  while(!RTC_ClockIsSet)
    vTaskDelay(25);

  vTaskResume(xWEB_GetTaskHandle);
  vTaskResume(xZigNetMgmtHandle);
  vTaskResume(xCalendarTaskHandle);
  vTaskResume(xSmartPlugSamplingHandle);
  vTaskResume(xLedTaskHandle);
  vTaskSuspend(NULL);
}
```

6.3 web_client.c

6.3.1 Web_Get_Task

Table 16 describes the Web_Get_Task task:

Table 16. Web_Get_Task task

Function name	Web_Get_Task
Function prototype	void Web_Get_Task (void *pvParameters)
Behavior description	Start the WEB GET task: Upload the smartplug sampling data on the web server where they are stored in an SQL database.
Input parameter {x}	None
Output parameter {x}	None
Return value	None
Required preconditions	None
Called functions	No API/HAL layer functions;

Example:

```
void Web_Get_Task(void *pvParameters)
{
  struct netconn *conn;
  int ret = 0;
  int len = 0;
  int nErr = 0;
  vTaskSuspend( NULL );
  /* Create a new connection identifier. */
  conn = netconn_new(NETCONN_TCP);
  /* Bind the connection to WEB server port */
  ret = netconn_connect(conn, &WEB_SERVER_IP, WEB_SERVER_PORT);
  /* Infinite main loop ------------------------------------------------------*/
  while (1)
  {
    if(SendToWeb == 1)
    {
      /* Check if the conn allocation has been successed */
      if (conn == 0)
      {
        /* Delete the current session data */
        netconn_delete(conn);
        /* Create a new connection identifier. */
```

```
    conn = netconn_new(NETCONN_TCP);
    /* Start the connection */
    do
    {
      /* Bind the connection to WEB server port */
      ret = netconn_connect(conn, &WEB_SERVER_IP, WEB_SERVER_PORT);
      /* Check if the Connection Binding has been successed */
      if (ret == ERR_MEM || ret == ERR_BUF || ret == ERR_VAL || ret == ERR_ARG )
      {
        /* Delete the current session data */
        netconn_delete(conn);
        /* Create a new connection identifier. */
        conn = netconn_new(NETCONN_TCP);
        vTaskDelay(700);
      }
    }while(ret!=0);
  }

  /* Write the GET request to the buffer to be sent to WEB server */
  len = strlen((char*)buf);
  /* Send application data to be encrypted */
  ret = netconn_write(conn, buf, len, NETCONN_NOFLAG);
  if(ret < 0)
  {
    /* Write to WEB server failed */
    /* Bind the connection to WEB server port */
    ret = netconn_connect(conn, &WEB_SERVER_IP, WEB_SERVER_PORT);
    /* Check if the Connection Binding has been successes */
    if ( ret < 0 )
    {
      /* Delete the current session data */
      netconn_delete(conn);
      /* Create a new connection identifier. */
      conn = netconn_new(NETCONN_TCP);
      do{
          /* Bind the connection to WEB server port */
          ret = netconn_connect(conn, &WEB_SERVER_IP, WEB_SERVER_PORT);
          /* Send application data to be encrypted */
          ret = netconn_write(conn, buf, len, NETCONN_NOFLAG);
      }while(ret<0);
```

```
        }
      }
      SendToWeb = 0;
    }
    /* Insert 10 s delay */
    vTaskDelay(700);
  }
}
```

6.4 tasks_misc.c

6.4.1 GraphicLibTask

Table 17 describes the GraphicLibTask task:

Table 17. GraphicLibTask task

Function name	GraphicLibTask
Function prototype	void GraphicLibTask (void *pvParameters)
Behavior description	Manage the graphic library layer and input events
Input parameter {x}	None
Output parameter {x}	None
Return value	None
Required preconditions	None
Called functions	No API/HAL layer functions;

Example:

```
void GraphicLibTask( void *pvParameters)
{
    vTaskSuspend( NULL );

    /* Infinite main loop ---------------------------------------*/
    while (1)
    {
        /* Catching touch events */
        if ( TS_IsCalibrationDone() == SET )
        {
           /* Catch the user input through Touchscreen/Joystick */
           ProcessInputData();
         }
         /*Timeout calculate for power saving mode*/
```

```
        TimeOutCalculate();
        CatchInputEvents();
        vTaskDelay(3);
    }
}
```

6.4.2 ZigNetMgmtTask

Table 18 describes the ZigNetMgmtTask task:

Table 18. ZigNetMgmtTask task

Function name	ZigNetMgmtTask
Function prototype	void ZigNetMgmtTask (void *pvParameters)
Behavior description	Initialize ZigBee network add nodes sending DeviceAnnounce request
Input parameter {x}	None
Output parameter {x}	None
Return value	None
Required preconditions	None
Called functions	No API/HAL layer functions;

Example:

```
void ZigNetMgmtTask( void *pvParameters)
{
    vTaskSuspend( NULL );
    uint8_t NewDevFlag = 1;

    /* Load Device List stored into the Dual Interface EEPROM */
    if( Load_HA_DevList() == 0 && HA_DeviceCount > 0)
    {
        ZigBeeNet_InitDone = 1;
        ShowNetworkStatus();
        CursorShow(195, 50);
        vTaskResume(xWEB_GetTaskHandle);
    }
    /* Infinite main loop ---------------------------------------*/
    while (1)
    {
       USART_ITConfig(HA_USART, USART_IT_RXNE, ENABLE);
       while( (HA_DevAnn == 1) && (ZigBeeNet_InitDone == 1) )
```

```
          {
            NewDevFlag = 1;
            for(uint8_t i=0; i<HA_DeviceCount; i++)
              if(ppHAdeviceList[i]->NetworkID == HA_NodeID)
              {
                 NewDevFlag = 0;
                 break;
              }

            /* Adding the new device to the device list */
            if(NewDevFlag == 1)
            {
               ppHAdeviceList[HA_DeviceCount++] = NewHA_Device(0, HA_NodeID);
               Add_HA_DeviceToCombo(GL_FALSE,0);
            }

            HA_DevAnn = 0; /* Reset the FLAG to accept new device join */
          }
          vTaskDelay(20);
      }
}
```

6.4.3 CalendarTask

Table 19 describes the CalendarTask task:

Table 19. CalendarTask task

Function name	CalendarTask
Function prototype	void CalendarTask (void *pvParameters)
Behavior description	Simulate the scenario over 24 hours switching on/off the devices according to their timer
Input parameter {x}	None
Output parameter {x}	None
Return value	None
Required preconditions	None
Called functions	No API/HAL layer functions;

Example:

```
void CalendarTask( void *pvParameters)
{
```

```
vTaskSuspend( NULL );
uint8_t * NodePower[16];
uint8_t NodeLabel[20];


/* Infinite main loop ---------------------------------------*/
while (1) {
  if(HA_TimerActivated == 1){
    if(ppHAdeviceList[0]->Timer == HA_TIMER_ON)
    {
      if((ppHAdeviceList[0]->StartTime == HA_HourCounter) &&
         (ppHAdeviceList[0]->Status == 0))
      {
         SwitchDeviceON(ppHAdeviceList[0]->NetworkID);
         ppHAdeviceList[0]->Status = 1;
      }
      if((ppHAdeviceList[0]->StopTime <= HA_HourCounter) &&
          (ppHAdeviceList[0]->Status == 1))
      {
         SwitchDeviceOFF(ppHAdeviceList[0]->NetworkID);
         ppHAdeviceList[0]->Status = 0;
      }
      if(ppHAdeviceList[0]->Status == 1)
      {
         sprintf((char*)NodeLabel, "%s", ppHAdeviceList[0]->Label);
         if(strcmp((char*)NodeLabel, "WASHING_MACHINE") == 0)
            sprintf((char*)NodeLabel, "WashMach Status ON ");
         else
            sprintf((char*)NodeLabel, "%s Status ON ", NodeLabel);
         Set_Label( &pageS1E, 0x68, NodeLabel );
         sprintf((char*)NodePower, "Power %d Watt  ", ppHAdeviceList[0]->Power);
      }
      else
      {
         sprintf((char*)NodeLabel, "%s", ppHAdeviceList[0]->Label);
         if(strcmp((char*)NodeLabel, "WASHING_MACHINE") == 0)
            sprintf((char*)NodeLabel, "WashMach Status OFF");
         else
            sprintf((char*)NodeLabel, "%s Status OFF", NodeLabel);
         Set_Label( &pageS1E, 0x68, NodeLabel );
         sprintf((char*)NodePower, "Power 0 Watt  ");
```

```
        }
        Set_Label( &pageS1E, 0x71, (uint8_t*)NodePower );
        RefreshPageControl( &pageS1E, 0x68 );
        RefreshPageControl( &pageS1E, 0x71 );
    }
    if(ppHAdeviceList[1]->Timer == HA_TIMER_ON)
    {
        if((ppHAdeviceList[1]->StartTime == HA_HourCounter) &&
           (ppHAdeviceList[1]->Status == 0))
        {
           SwitchDeviceON(ppHAdeviceList[1]->NetworkID);
           ppHAdeviceList[1]->Status = 1;
        }
        if((ppHAdeviceList[1]->StopTime <= HA_HourCounter) &&
           (ppHAdeviceList[1]->Status == 1))
        {
           SwitchDeviceOFF(ppHAdeviceList[1]->NetworkID);
           ppHAdeviceList[1]->Status = 0;
        }
        if(ppHAdeviceList[1]->Timer == HA_TIMER_ON)
        {
          if(ppHAdeviceList[1]->Status == 1)
          {
             sprintf((char*)NodeLabel, "%s", ppHAdeviceList[1]->Label);
             if(strcmp((char*)NodeLabel, "WASHING_MACHINE") == 0)
                sprintf((char*)NodeLabel, "WashMach Status ON ");
             else
                sprintf((char*)NodeLabel, "%s Status ON ", NodeLabel);
             Set_Label( &pageS1E, 0x69, NodeLabel );
             sprintf((char*)NodePower, "Power %d Watt  ", ppHAdeviceList[1]->Power);
          }
          else
          {
            sprintf((char*)NodeLabel, "%s", ppHAdeviceList[1]->Label);
            if(strcmp((char*)NodeLabel, "WASHING_MACHINE") == 0)
                sprintf((char*)NodeLabel, "WashMach Status OFF");
            else
                sprintf((char*)NodeLabel, "%s Status OFF", NodeLabel);
            Set_Label( &pageS1E, 0x69, NodeLabel );
            sprintf((char*)NodePower, "Power 0 Watt  ");
          }
```

```
          Set_Label( &pageS1E, 0x72, (uint8_t*)NodePower );
          RefreshPageControl( &pageS1E, 0x69 );
          RefreshPageControl( &pageS1E, 0x72 );
        }
      }
      RTC_TimeStampShow();
    }
    vTaskDelay(90);
  }
}
```

6.4.4 SPSamplingTask

Table 20 describes the SPsamplingTask task:

Table 20. SPsamplingTask task

Function name	SPsamplingTask
Function prototype	void SPsamplingTask (void *pvParameters)
Behavior description	Read and sample the plug consumption values
Input parameter {x}	None
Output parameter {x}	None
Return value	None
Required preconditions	None
Called functions	No API/HAL layer functions;

Example:

```
void SPsamplingTask( void *pvParameters )
{
  uint8_t i=0, j=0, f=0, g=0;
  uint8_t index = 0;
  vTaskSuspend( NULL );

  /* Infinite main loop ------------------------------------------------------*/
  while (1)
  {
    if(HA_ReportReceived)
    {
      HA_ReportReceived = 0;
      for(; index < HA_DeviceCount; index++)
      {
```

```
  if(HA_NodeID == ppHAdeviceList[index]->NetworkID)
    break;
}
if(index == 0)
{
  if ( ppHAdeviceList[0]->Status == 1 )
  {
    if(HA_Attribute == 0x0000) /* Type of data = Energy */
    {
      smart_points1[i%N_SAMPLES] = ppHAdeviceList[0]->Energy;
      i++;
    }
    else if(HA_Attribute == 0x0400)/* Type of data = Power */
    {
      smart_points1B[i%N_SAMPLES] = ppHAdeviceList[0]->Power;
      j++;
    }
    sendData_ToWeb(ppHAdeviceList[0]->NetworkID, ppHAdeviceList[0]->Label,
                   ppHAdeviceList[0]->Power, ppHAdeviceList[0]->Energy);
  }
  else
  {
    smart_points1[i%N_SAMPLES] = 0;
    smart_points1B[j%N_SAMPLES] = 0;
    i++;
    j++;
  }
}
else if(index == 1)
{
  if ( ppHAdeviceList[1]->Status == 1 )
  {
    if(HA_Attribute == 0x0000) /* Type of data = Energy */
    {
      smart_points2[f%N_SAMPLES] = ppHAdeviceList[1]->Energy;
      f++;
    }
    else if(HA_Attribute == 0x0400)/* Type of data = Power */
    {
      smart_points2B[g%N_SAMPLES] = ppHAdeviceList[1]->Power;
```

```
              g++;
            }
            sendData_ToWeb(ppHAdeviceList[1]->NetworkID, ppHAdeviceList[1]->Label,
                           ppHAdeviceList[1]->Power, ppHAdeviceList[1]->Energy);
          }
          else
          {
            smart_points2[f%N_SAMPLES] = 0;
            smart_points2B[g%N_SAMPLES] = 0;
            f++;
            g++;
          }
        }
       else if(ppHAdeviceList[index] && HA_NodeID == ppHAdeviceList[index]->NetworkID)
         sendData_ToWeb(ppHAdeviceList[index]->NetworkID,ppHAdeviceList[index]->Label,
                        ppHAdeviceList[index]->Power, ppHAdeviceList[index]->Energy);
       if((i == N_SAMPLES) || (j == N_SAMPLES) || (f == N_SAMPLES) || (g == N_SAMPLES))
        {
          i=0;
          j=0;
          f=0;
          g=0;
          for(uint8_t k=0; k<N_SAMPLES; k++)
          {
            smart_points1[k]  = 0;
            smart_points1B[k] = 0;
            smart_points2[k]  = 0;
            smart_points2B[k] = 0;
          }
        }
      }
      index = 0;
      USART_ITConfig(HA_USART, USART_IT_RXNE, ENABLE);
      vTaskDelay(100);
    }
  }
```

6.4.5 ToggleLed Task

Table 21 describes the ToggleLed Task:

Table 21. ToggleLed Task

Function name	ToggleLed
Function prototype	void ToggleLed (void *pvParameters)
Behavior description	Start toggle Led task: Toggle LED 1, 2 and 3 every 200 ms
Input parameter {x}	None
Output parameter {x}	None
Return value	None
Required preconditions	None
Called functions	No API/HAL layer functions

Example:

```
void ToggleLed( void *pvParameters)
{
  vTaskSuspend( NULL );

  portTickType xLastWakeTime;
  const portTickType xFrequency = 200;

  /* Initialize the xLastWakeTime variable with the current time */
  xLastWakeTime = xTaskGetTickCount();

  /* Infinite loop */
  for( ;; )
  {
      STM_EVAL_LEDToggle(LED1);
      /* Insert 12 ms delay */
      vTaskDelay(12);
      STM_EVAL_LEDToggle(LED1);
      /* Insert 12 ms delay */
      vTaskDelay(12);
      STM_EVAL_LEDToggle(LED2);
      /* Insert 10 ms delay */
      vTaskDelay(12);
      STM_EVAL_LEDToggle(LED2);
      /* Insert 12 ms delay */
```

```
        vTaskDelay(12);
        STM_EVAL_LEDToggle(LED3);
        /* Insert 12 ms delay */
        vTaskDelay(12);
        STM_EVAL_LEDToggle(LED3);
        /* Insert 12 ms delay */
        vTaskDelay(12);
        STM_EVAL_LEDToggle(LED2);
        /* Insert 12 ms delay */
        vTaskDelay(12);
        STM_EVAL_LEDToggle(LED2);
        vTaskDelayUntil(&xLastWakeTime, xFrequency);
    }
}
```

7 Getting started with the system

7.1 Hardware requirements

- ZigBee adapter board with the ZigBee module SPZB32W onboard
- STM322xG-EVAL board with the jumpers set as described in *Section 7.2*
- VDC / 2 A isolated power supply is recommended
- One JTAG programmer/debugger dongle (J-Link from SEGGER or IAR Systems™ is recommended). It is unnecessary if no modifications to the firmware code have been performed.

7.2 STM322xG-EVAL demonstration board setup

Set up the STM322xG-EVAL board as follows:

- Keep jumper JP16, JP21 and JP22 not fitted
- Set JP5 in 1-2 position, and JP6 in 2-3 position
- Disconnect motor control board from CN5
- Remove MicroSD card from CN6
- Remove R115 and R140.

Set up the STEVAL-IDZ001V1 adapter as follows:

- Place the adapter on the STM322xG-EVAL board so that it is well fixed on connectors CN2, CN3 and CN4
- Set JP3 and JP4 in 1-2 position in order to set the communication between STM32W108CB and STM32F2x7x via UART
- Set JP1 and JP2 in position 1-2 to assign 0xA6 as I^2C address for the dual interface EEPROM (M24LR64r).

7.3 STM322xG-EVAL and ZigBee adapter pins mapping

The connections between the MCU and the ZigBee module + EEPROM dual interface are shown in *Table 12*.

Table 22. ZigBee adapter pinout description

STM32 pin name	STM322xG-EVAL I/O assignment	Extension connector pin. no	ZigBee adapter I/O assignment
-	APP_3V3	CN3	VCC_3V3
-	GND	CN3	GND
PA4	Audio_DAC_OUT (remove R115)	CN2 -Pin 4	ZIG_SS (SPI3)
PC10	MicroSDCard_D2 & RS232-IrDA_TX (remove MicroSD card from CN6)	CN4	ZIG_SCLK (SPI3)
PC11	MicroSDCard_D3 & RS232-IrDA_RX (JP22 open, remove MicroSD card from CN6)	CN4 -Pin 35	ZIG_MISO (SPI3)

Table 22. ZigBee adapter pinout description (continued)

STM32 pin name	STM322xG-EVAL I/O assignment	Extension connector pin. no	ZigBee adapter I/O assignment
PC12	MicroSDCard_CLK (remove MicroSD card from CN6)	CN4 -Pin 34	ZIG_MOSI (SPI3)
PA8	MCO (JP5 open)	CN3	I2C3_SCK
PC9	MicroSDCard_D1 & I2S_CKIN (keep JP16 on open)	CN3	I2C3_SDA
PD2	MicroSDCard_CMD	CN4	ZIG_HOST_INT
PC8	MicroSDCard_D0 & MC (remove MicroSD card from CN6, disconnect motor control board from CN5)	CN3 -Pin 18	ZIG_RSTB
PH13	MicroSDCard_detect & MC (remove MicroSD card from CN6, disconnect motor control board from CN5)	CN3	ZIG_WAKE
PH15	SmartCard_3/5V & MC (disconnect motor control board from CN5)	CN3 -Pin 5	VCC-GPIO (for M24LR64-r)
PC6	I2S_MCK & SmartCard_IO (JP21 open)	CN3	USART6_TX
PC7	LED4 (Remove R140)	CN3	USART6_RX

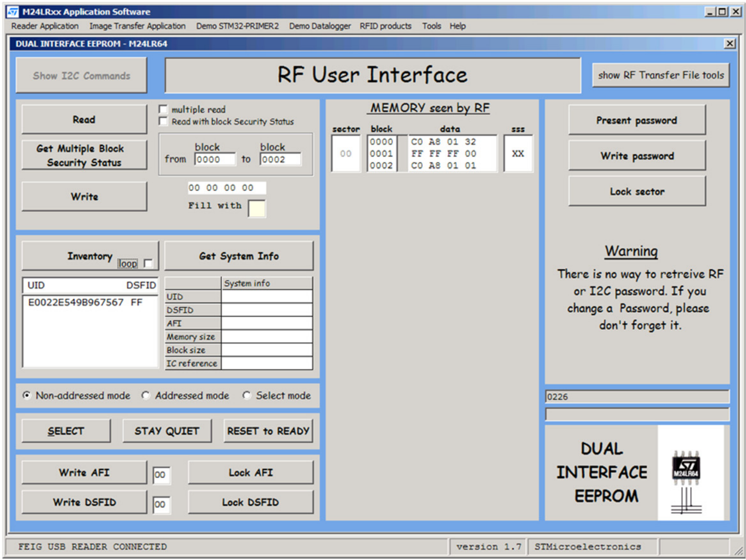
7.4 Program/update dual interface EEPROM with network parameters

The dual interface EEPROM M24LR64-r stores the network parameters of the STM322xG-EVAL board inside the first 3 blocks of the sector 1 and the IP address of the web server at the fourth block of the sector 1, where each block is 4 bytes. The schema is the following:

- Block 0000: IP address
- Block 0001: subnet mask
- Block 0002: default gateway
- Block 0003: web server IP address.

Figure 12 shows a screenshot of the “M24LRxx application software”, the GUI application that allows the user to program the M24LR64-r memory through an RF reader (i.e. FEIG) included in the demo kit (USB based) of the dual interface EEPROM:

Figure 12. M24LRxx application software



AM11982v1

For more information on the M24LRxx application software, see the UM0853 user manual, the GUI application is available from the ST website: http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/SW_FUNCTION/m24lr_application_software.zip.

7.5 Configure IAR tool for building, debugging, and programming application

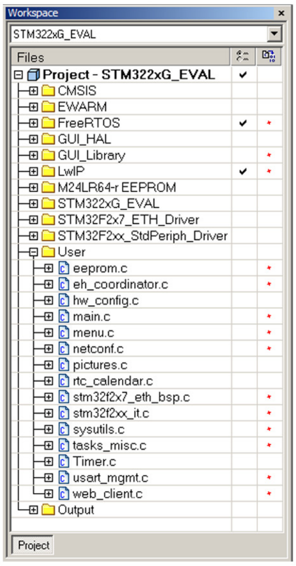
Together with the firmware library package, an example application is delivered in order to provide the final user with a real example of the “In-Home Display with HTTP gateway” application usage.

The delivered example application has been written and developed using IAR EWARM 6.30 IDE and can be built for the STM32F2x7x microcontroller.

The workspace is created using the IAR embedded workbench 6.30 IDE, using the ARM®-based 32-bit STM32F2x7x firmware library (ver. 1.0.0), CMSIS compliant, and in C language.

The tree structure of the project is organized separating and grouping the source files with the header files, both for the project files and the library files, as shown in *Figure 13*.

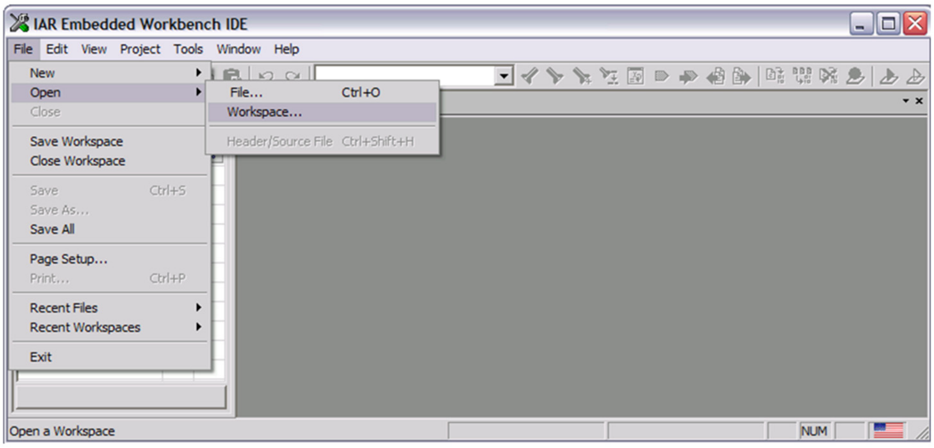
Figure 13. Application project files



AM11983V1

In order to load the project, click on File\Open\Workspace and in the window that appears select, in \project folder\EWARM, the file “Project.eww”.

Figure 14. IAR embedded workbench main window



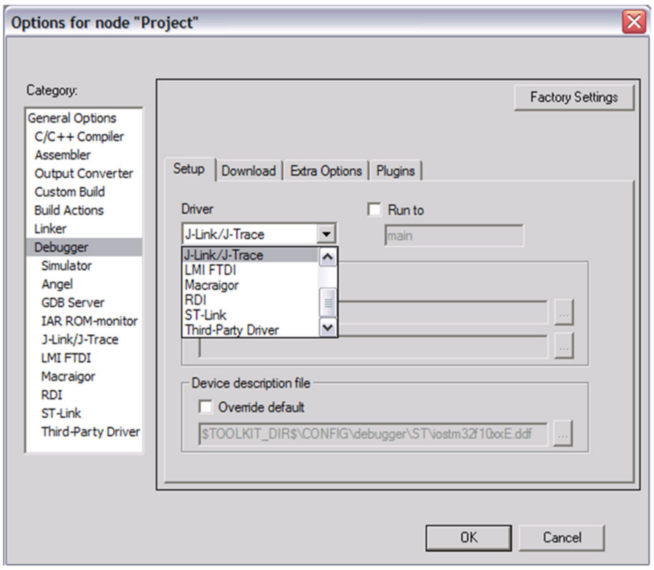
AM11984v1

On the main node, where the program name located in the files window is shown, right click and select Options. In the window which appears, select the Debugger item in the Category list box, and select the proper debugging tool in the Driver list box, then press the OK button. In the proposed example the J-Link dongle is used (*Figure 15*).

Press the “Make” icon or click on “Project \ Rebuild All”. Any error or warning should appear once the compiling has completed. Connect the J-Link tool to the USB port of the PC, and connect the flat cable with the programming adapter. Plug the adapter into the dongle connector. Press the Debug icon, CTRL+D or click Project\Debug. The debugger starts to download the firmware to the dongle through the J-Link debugger\programmer. Press the Go button, F5 or click Debug\Go in order to execute the firmware in Debug mode. To run the dongle in Standalone mode, press the Stop Debugging icon, CTRL+SHIFT+D or click

Debug\Stop Debugging. Then remove the J-Link adapter from the dongle and reset the board by unplugging and plugging the power cable back in.

Figure 15. IAR embedded workbench debugger options



AM11985v1

In order to use the application project, it is necessary to:

- Include all the firmware delivered in the In-Home Display with HTTP gateway package containing the FreeRTOS core files, see *Figure 13*.
- Create the desired menu application functions in the file menu.c.
- Put inside "picture.c" the HEX dump of the pictures to be used with the application GUI.
- Implement a main function as described in the following section.

7.6 Example application - main.c

An example of a main application is reported below. The main function contains an example of the In-Home Display with HTTP gateway application initialization/configuration and implements the classic operations:

```
/* Includes ----------------------------------------------------*/
#include "main.h"
#include "touchscreen.h"

#ifdef USE_FREERTOS
  #include "FreeRTOS.h"
  #include "task.h"
#endif

/* Private typedef ---------------------------------------------*/
/* Private define ----------------------------------------------*/
```

```
/*--------------- Tasks Priority -------------*/
#define WEB_TASK_PRIO   ( configMAX_PRIORITIES - 3 )
#define LED_TASK_PRIO   ( tskIDLE_PRIORITY + 1 )

/* Private macro ------------------------------------------------*/
/* Private variables --------------------------------------------*/
/* Task Handles */
xTaskHandle xSmartPlugSamplingHandle, xWEB_GetTaskHandle, xGraphLibTaskHandle,
            xLedTaskHandle, xCalendarTaskHandle, xZigNetMgmtHandle;

  /* Static IP Address */
uint8_t IP_ADDR0;
uint8_t IP_ADDR1;
uint8_t IP_ADDR2;
uint8_t IP_ADDR3;

  /* Subnet Mask */
uint8_t NETMASK_ADDR0;
uint8_t NETMASK_ADDR1;
uint8_t NETMASK_ADDR2;
uint8_t NETMASK_ADDR3;

  /* Default Gateway */
uint8_t GW_ADDR0;
uint8_t GW_ADDR1;
uint8_t GW_ADDR2;
uint8_t GW_ADDR3;

  /* Web Server Static IP Address */
uint8_t WEB_SERVER_IP_ADDR0;
uint8_t WEB_SERVER_IP_ADDR1;
uint8_t WEB_SERVER_IP_ADDR2;
uint8_t WEB_SERVER_IP_ADDR3;

extern uint8_t RTC_ClockIsSet;

/* Private function prototypes ----------------------------------*/
static void demo_initTask( void *pvParameters );
extern void (*pDelayFunc)(uint32_t nTime);
extern void (*pSysTickFunc)(void);
```

```
extern void DelayCounter(uint32_t nTime);
extern void DelayRTOS(uint32_t nTime);

#ifdef __GNUC__
/* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
   set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

/* Private functions -------------------------------------------*/

/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */
int main(void)
{
  pDelayFunc = DelayCounter;
  /*!< At this stage the microcontroller clock setting is already configured to
       120 MHz, this is done through SystemInit() function which is called from
       startup file (startup_stm32f2xx.s) before to branch to application main.
       To reconfigure the default setting of SystemInit() function, refer to
       system_stm32f2xx.c file
   */

  /* Setup STM32 system (USART, RNG, HASH, CRYP) and STM322xG-EVAL resources */
  System_Setup();

  EEprom_NetConfigReadFunc();

  /* Configure Ethernet (GPIOs, clocks, MAC, DMA) */
  ETH_BSP_Config();
  /* Initialize the LwIP stack */
  LwIP_Init();

  portBASE_TYPE res = xTaskCreate(demo_initTask, "DEMO_INIT",
                                configMINIMAL_STACK_SIZE*12, NULL,
                                configMAX_PRIORITIES-1, NULL);
```

```
/* Start the WEB GET task: Update the SmartPlug data on the WebServer */
res = xTaskCreate(Web_Get_Task, "WEB_GET", configMINIMAL_STACK_SIZE*22, NULL,
                  tskIDLE_PRIORITY+4, &xWEB_GetTaskHandle);
/* Start the Graphic Library task: Check the Touchscreen, Joystick, Button user
   interaction */
res = xTaskCreate(GraphicLibTask, (signed portCHAR *) "GraphicLibrary",
                  configMINIMAL_STACK_SIZE*16, NULL, tskIDLE_PRIORITY+6,
                  &xGraphLibTaskHandle);
/* Start the ZigBee Network Management task: manage the whole ZigBee network and
   keep trace of the alive Nodes */
res = xTaskCreate(ZigNetMgmtTask, (signed portCHAR *) "ZigNetMgmt",
                  configMINIMAL_STACK_SIZE*4, NULL, tskIDLE_PRIORITY+2,
                  &xZigNetMgmtHandle);
/* Start the Calendar task: Update the date/time and show it on the LCD */
res = xTaskCreate(CalendarTask, (signed portCHAR *) "Calendar",

                  configMINIMAL_STACK_SIZE*4, NULL, tskIDLE_PRIORITY+2,

                  &xCalendarTaskHandle);
/* Start the SmartPlug Measurement Sampling task: Sample the measured power/energy

   of each Node */
res = xTaskCreate(SPsamplingTask, (signed portCHAR *) "SmartPlugSampling",

                  configMINIMAL_STACK_SIZE*6, NULL, tskIDLE_PRIORITY+3,

                  &xSmartPlugSamplingHandle);
/* Start toggleLed task: Toggle LED 1, 2 and 3 every 200ms */
res = xTaskCreate(ToggleLed,"LED",configMINIMAL_STACK_SIZE/4, NULL, LED_TASK_PRIO,
                  &xLedTaskHandle);

pDelayFunc = DelayRTOS;

/* Start the scheduler */
vTaskStartScheduler();


/* We should never get here as control is now taken by the scheduler */
for( ;; );
}
```

The initialization process is charged with preparing the basic mechanism of the system:

- Hardware peripheral configuration and initialization
- ZigBee stack initialization
- FreeRTOS task creation
- Starting system.

The clock distribution and the interrupt settings are two components which are strongly dependent on the target project. An example of clock rate may be 72 MHz as the maximum speed of the current STM32 microcontroller. It can be decreased to reduce the power consumption. The clock rate assumptions are:

- System HCLK - 72 MHz
- Low speed peripheral PCLK1 - 72 MHz
- High speed peripheral PCLK2 - 36 MHz
- Analog-to-digital converter ADCCLK - 36 MHz.

The interrupt setting situation is very similar to clock distribution. The library functions involved with interrupt managing do not take the priorities into account; they only perform very necessary and absolutely common settings to make them serviceable.

8 In-Home Display with HTTP gateway GUI application

8.1 Application user interface

After a board reset, if the firmware is correctly loaded into the Flash memory and the board power is correctly supplied, the main screen is displayed, as shown in *Figure 16*.

Figure 16. Home screen

If the network has already been set up in the past, after some seconds, when the ZigBee network initialization is complete, the home screen looks like that in *Figure 17*.

Figure 17. Home screen after ZigBee network initialization done

At this point, the ZigBee nodes can be switched on and joined to the ZigBee network just created. The default channel is 16, so set it on the ZigBee nodes accordingly.

If “Management” is chosen, two situations are possible:

a) If no plug is connected to the smartplug coordinator, the following screen is shown:

Figure 18. No plug detected

From this screen it is possible to run the ZigBee network setup by which a new ZigBee network is formed. The screen in *Figure 19* is shown:

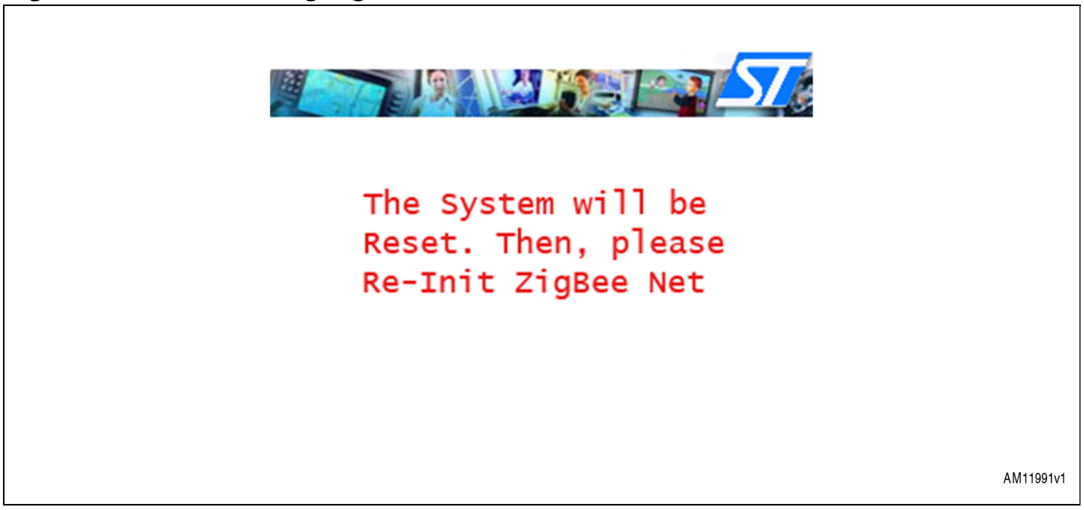
Figure 19. Setup ZigBee network

If no network already exists with the current coordinator, the screen in *Figure 20* is shown:

Figure 20. Setting ZigBee network up

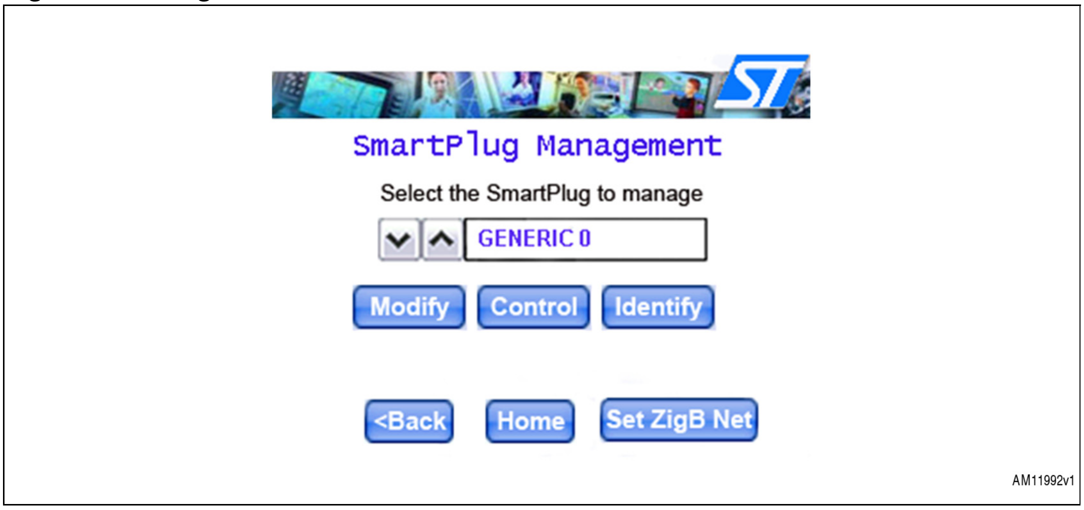
If a ZigBee network had already been set up, it must be reinitialized, so the screen in *Figure 21* is shown:

Figure 21. Reinitializing ZigBee network



b) If one or more plugs are detected the following screen is shown:

Figure 22. Plug detected



There are three main buttons: modify, control and identify. They allow the management of the smartplug device.

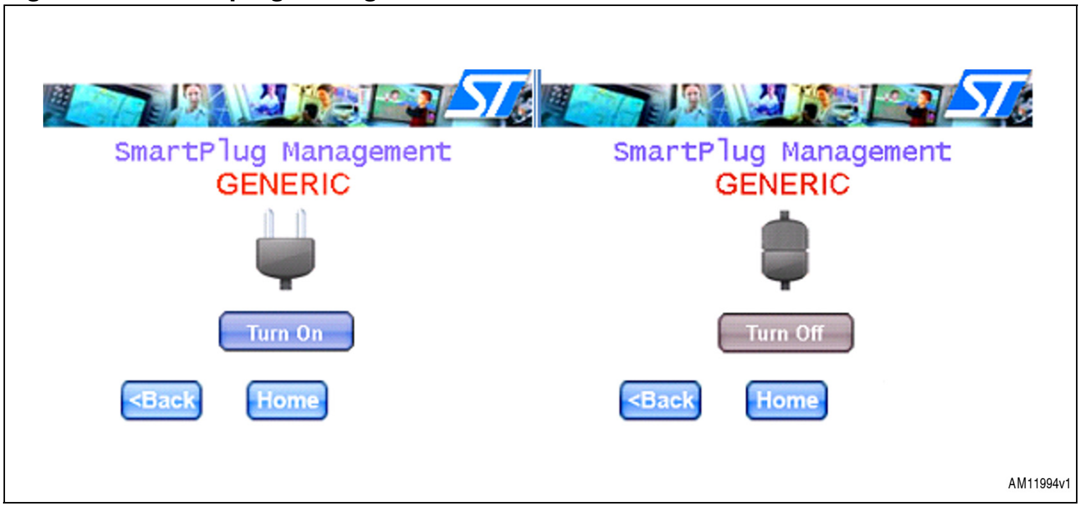
If "Modify" is chosen, the user can change the label of the selected plug, through the screen shown in *Figure 23*, choosing the new label from a list contained in a combobox.

Figure 23. Label changing

The user should scroll the list of the labels and select the "Apply" button in order to save the information and set the label also in the plug device via ZigBee protocol.

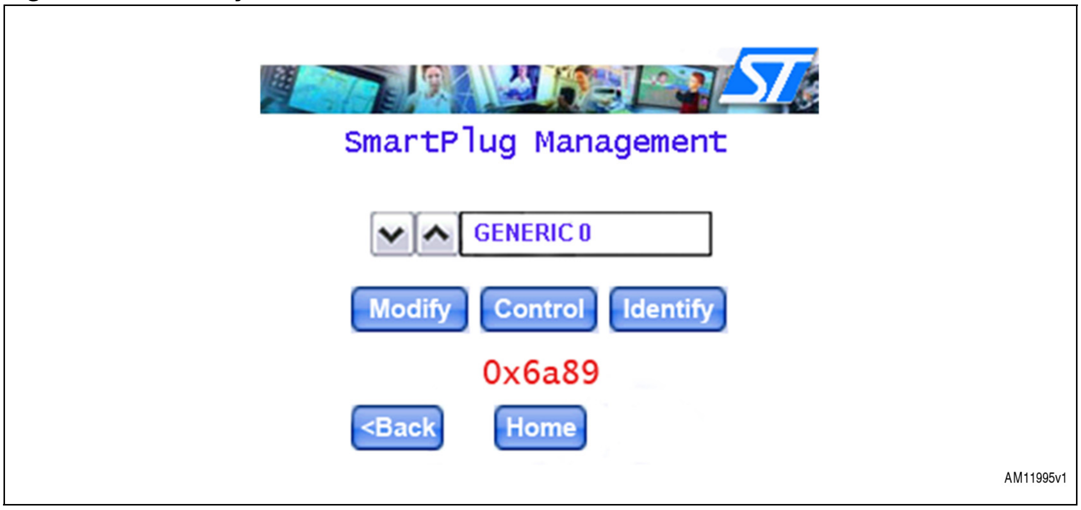
When “Control” is chosen, it is possible to switch on/off the smartplug device. The following screen shows the scenario:

Figure 24. Smartplug management



When “Identify” is chosen, the system shows the network ID of the selected node, which is a 16-bit number shown in hexadecimal format. *Figure 25* below shows the respective screen:

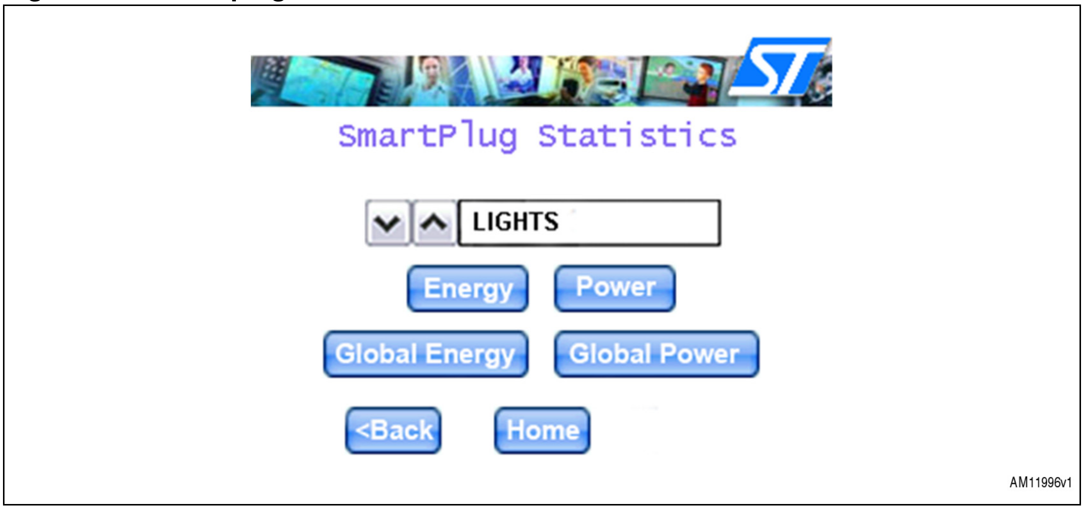
Figure 25. Identify



In the home screen, by clicking on “Statistics” it is possible to see the electrical consumption of the single smartplug device or the total consumption related to all plugs connected to the smartplug coordinator.

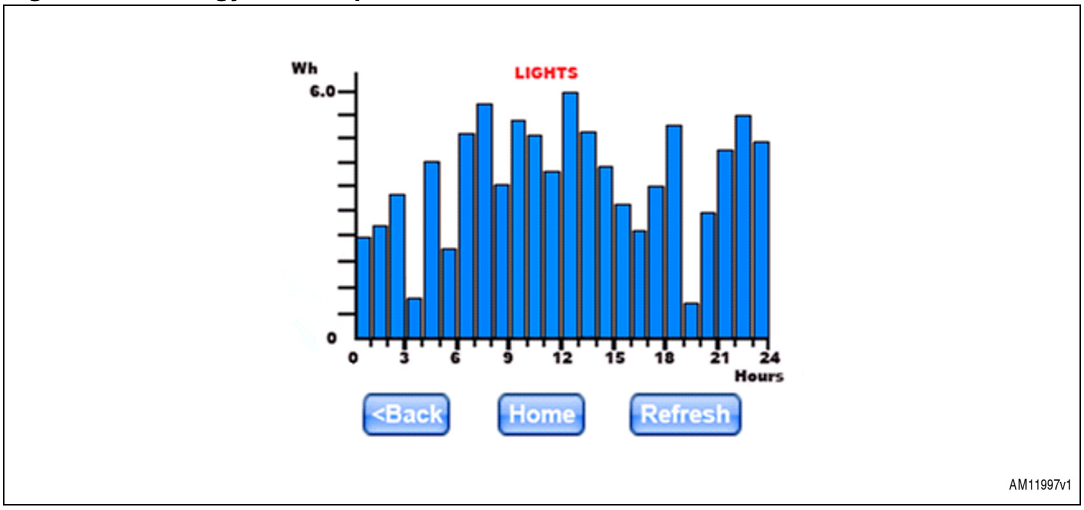
The following screen is shown:

Figure 26. Smartplug statistics



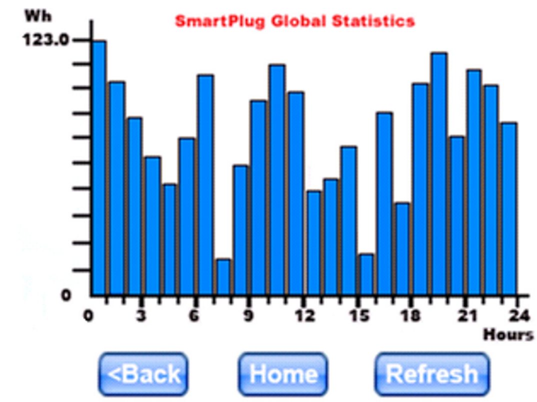
When energy is chosen, the following screen, representing the energy consumption of the selected smartplug device, is shown:

Figure 27. Energy consumption



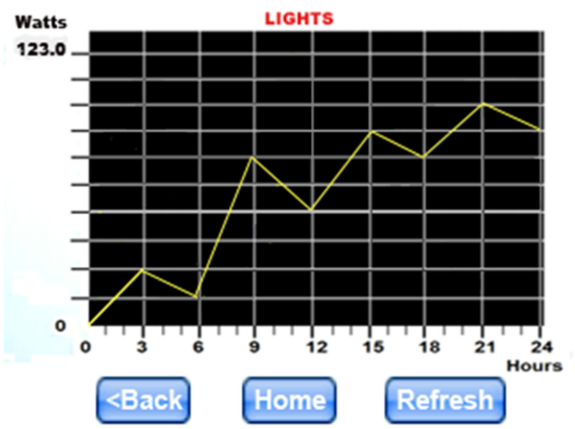
When global energy is chosen, the following screen, representing the energy consumption of the whole smartplug network, is shown:

Figure 28. Global energy consumption



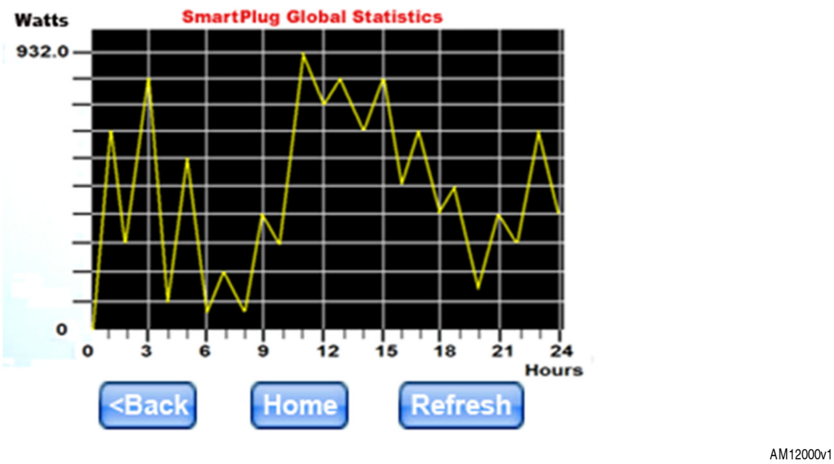
When power is chosen, the following screen, representing the power consumption of the selected smartplug device, is shown:

Figure 29. Power consumption



When global power is chosen, the following screen, representing the power consumption of the whole smartplug network, is shown:

Figure 30. Global power consumption



When the refresh button is chosen, it is possible to make a refresh of the graph chart in relation to the sampled points of the power consumption.

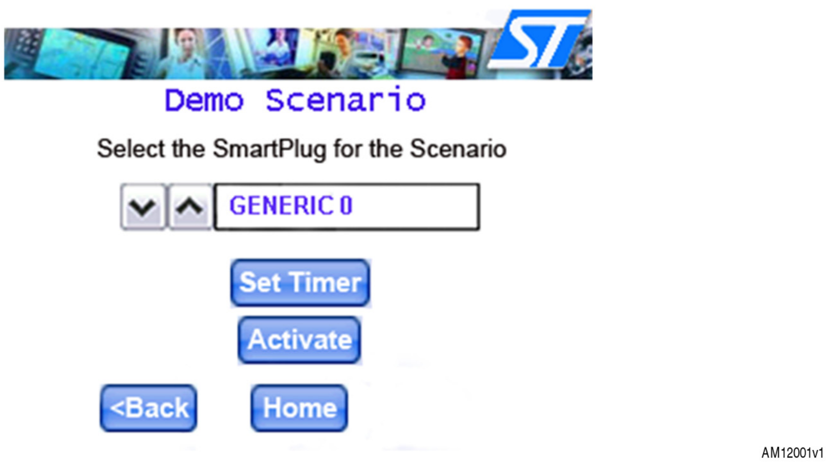
When the home button is chosen, it returns to the home screen.

On the home screen, by clicking on “Demo Scenario” it is possible to set a configuration for the nodes in order to enable automatic timers.

Only the first two nodes can be configured for the demo scenario.

The following screen is shown:

Figure 31. Demo scenario



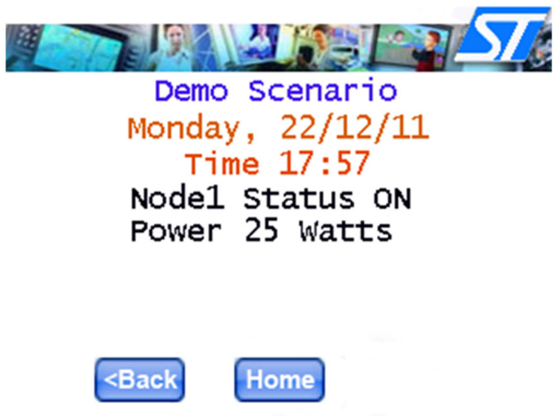
When the “Set Timer” button is chosen, a new screen is shown where the user can set up the parameter for the timer related to a smartplug node.

Figure 32. Set timer

After setting up the parameters, the user must choose “Save Parameters” in order to apply the changes. Then, the user must choose “Back”, bringing up the previous screen so that the timer can be activated with the respective button. Before activating the timer, the user can set it for another smartplug, so that the timer scenario is activated for 2 devices (the maximum allowed for the demo).

Clicking on the “Activate” button, the following screen is shown:

Figure 33. Activate timer



In this way, the demo scenario is running and the node for which the timer is activated is switched on and off according to the timer parameters set in the previous screen. In this screen the user can visualize the current date and time, the node status (off or on) and the instant power consumption. If the timer is activated also for a second node, its status and power consumption are shown after those of the first node.

When the home button is selected, it returns to the home screen.

9 Web server hosting energy consumption data

9.1 Features overview

Each time a smartplug node joins the network and its load type is configured through the GUI interface, it starts reporting power and energy consumption measured values.

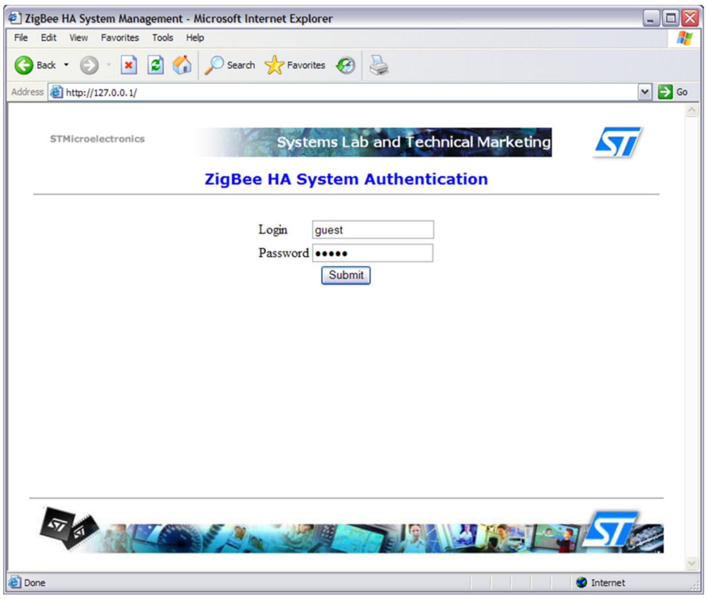
These data are received from the coordinator board and routed via Ethernet to the web server.

The user can browse these data from another PC connecting to the web server and logging in with a username and password. Then, the user can look at the reporting table, export data to an Excel file or plot them to graphical charts.

9.2 Accessing the website

The following image shows the login page of the web server:

Figure 34. Login page



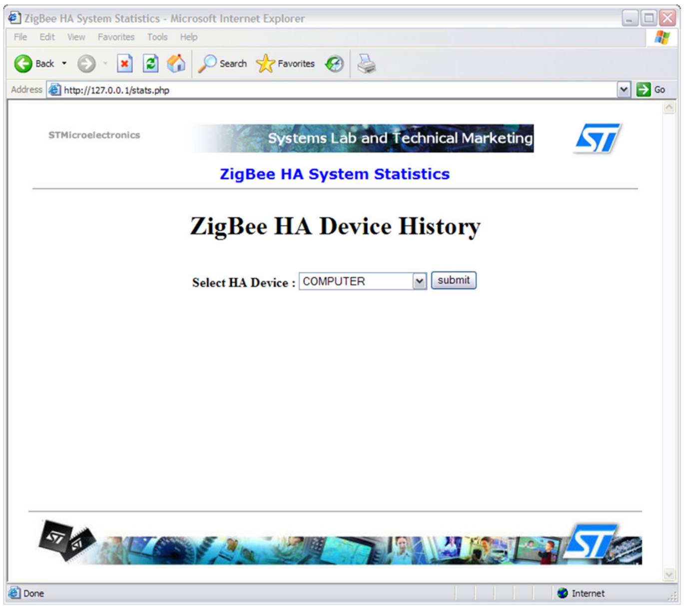
AM12124v1

For example, the user can log in using “guest” as both username and password.

9.3 Browsing reporting data

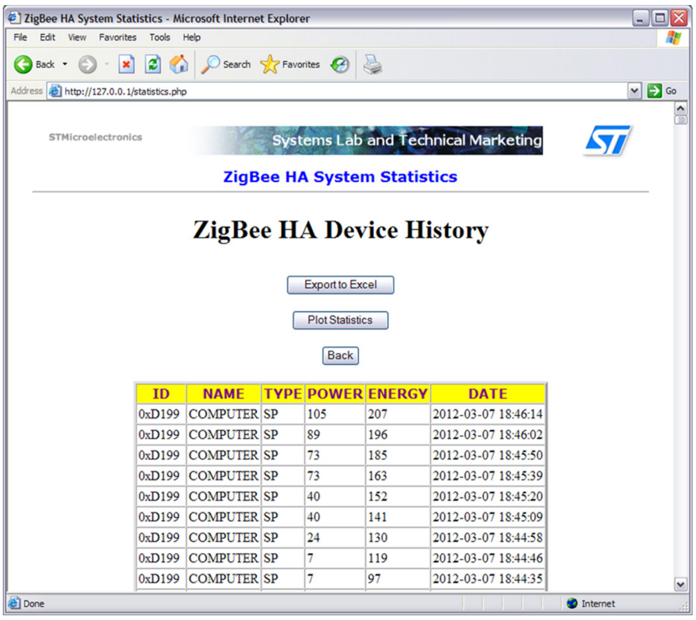
After login, the browser is redirected to the following page where the user can choose the smartplug of which they want to query the reporting data. It can be selected from a combobox containing a list of all the smartplugs that have sent reports. From this page on, the user can always return using the specific “Back” button.

Figure 35. Device selection page



After selecting the smartplug device, the following page is displayed:

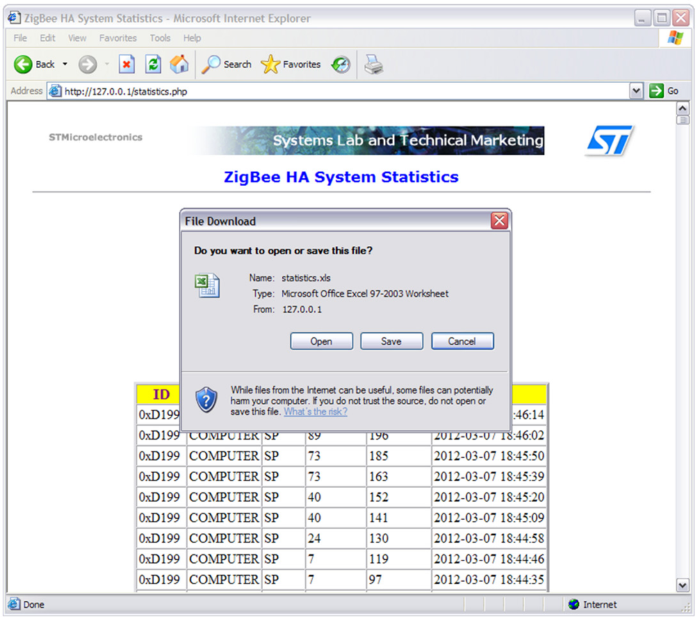
Figure 36. Energy data table page



At this point, the user can:

1. Export the table of data to an Excel file, clicking on the “Export to Excel” button. See the following image:

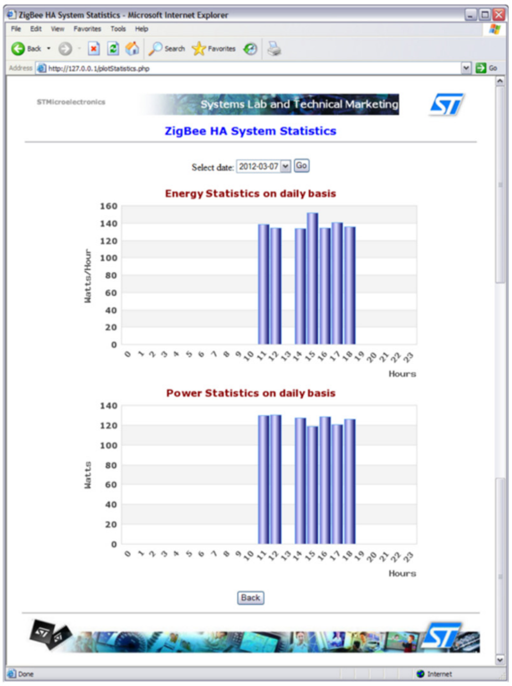
Figure 37. Export data page



ID					
0xD199					:46:14
0xD199	COMPUTER	SP	89	196	2012-03-07 18:46:02
0xD199	COMPUTER	SP	73	185	2012-03-07 18:45:50
0xD199	COMPUTER	SP	73	163	2012-03-07 18:45:39
0xD199	COMPUTER	SP	40	152	2012-03-07 18:45:20
0xD199	COMPUTER	SP	40	141	2012-03-07 18:45:09
0xD199	COMPUTER	SP	24	130	2012-03-07 18:44:58
0xD199	COMPUTER	SP	7	119	2012-03-07 18:44:46
0xD199	COMPUTER	SP	7	97	2012-03-07 18:44:35

2. Plot the data to graphical charts, where the user can select the day for which to plot the data, as shown from the following screen:

Figure 38. Plotting data page



3. Return in order to select another smartplug device using the “Back” button.

10 Schematics

Figure 39. ZigBee and dual Interface EEPROM adapter for STM322xG-EVAL

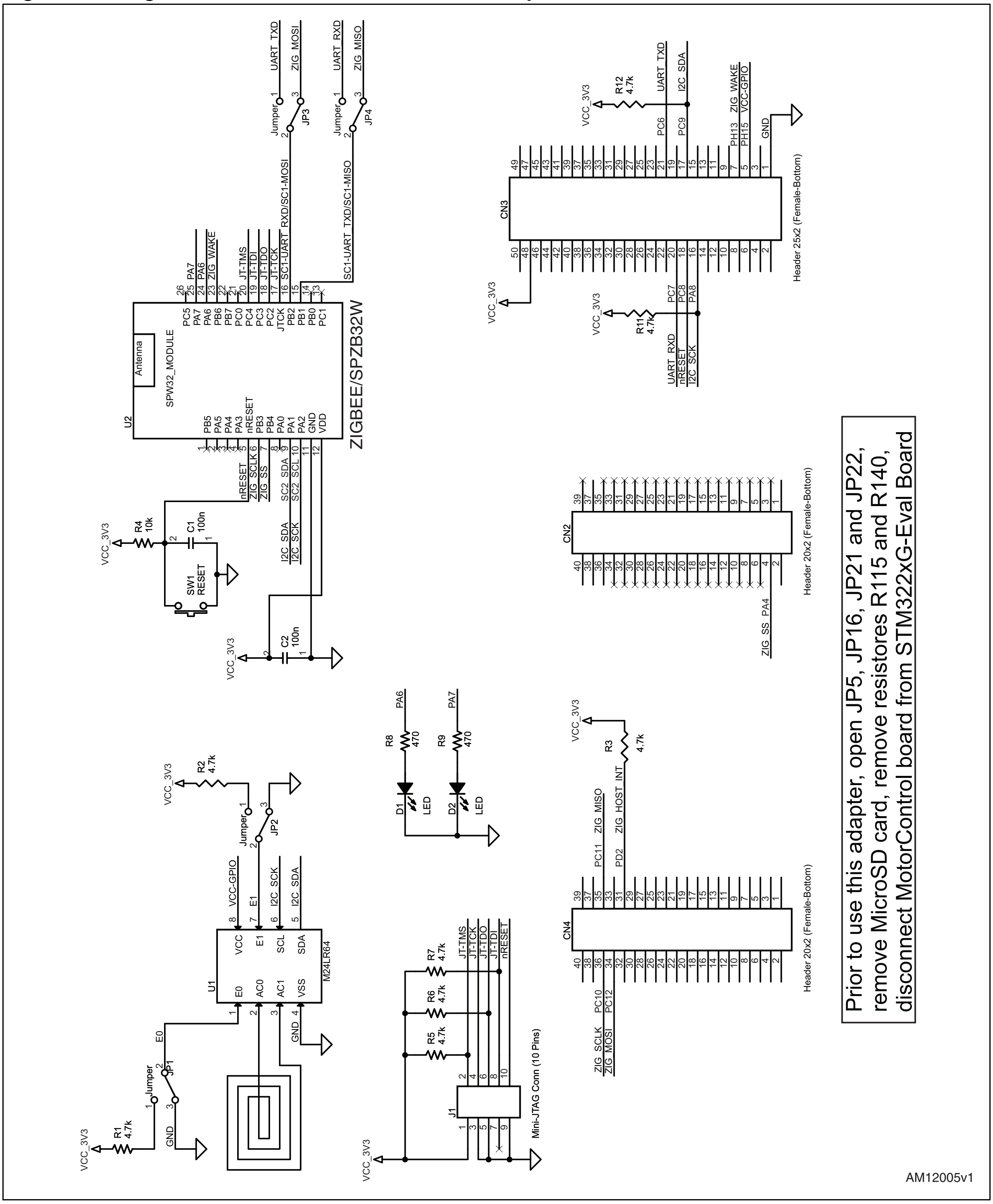


Figure 40. STM3220-21-45-46G-EVAL

Figure 41. STM3220-21-45-46G-EVAL_MCU

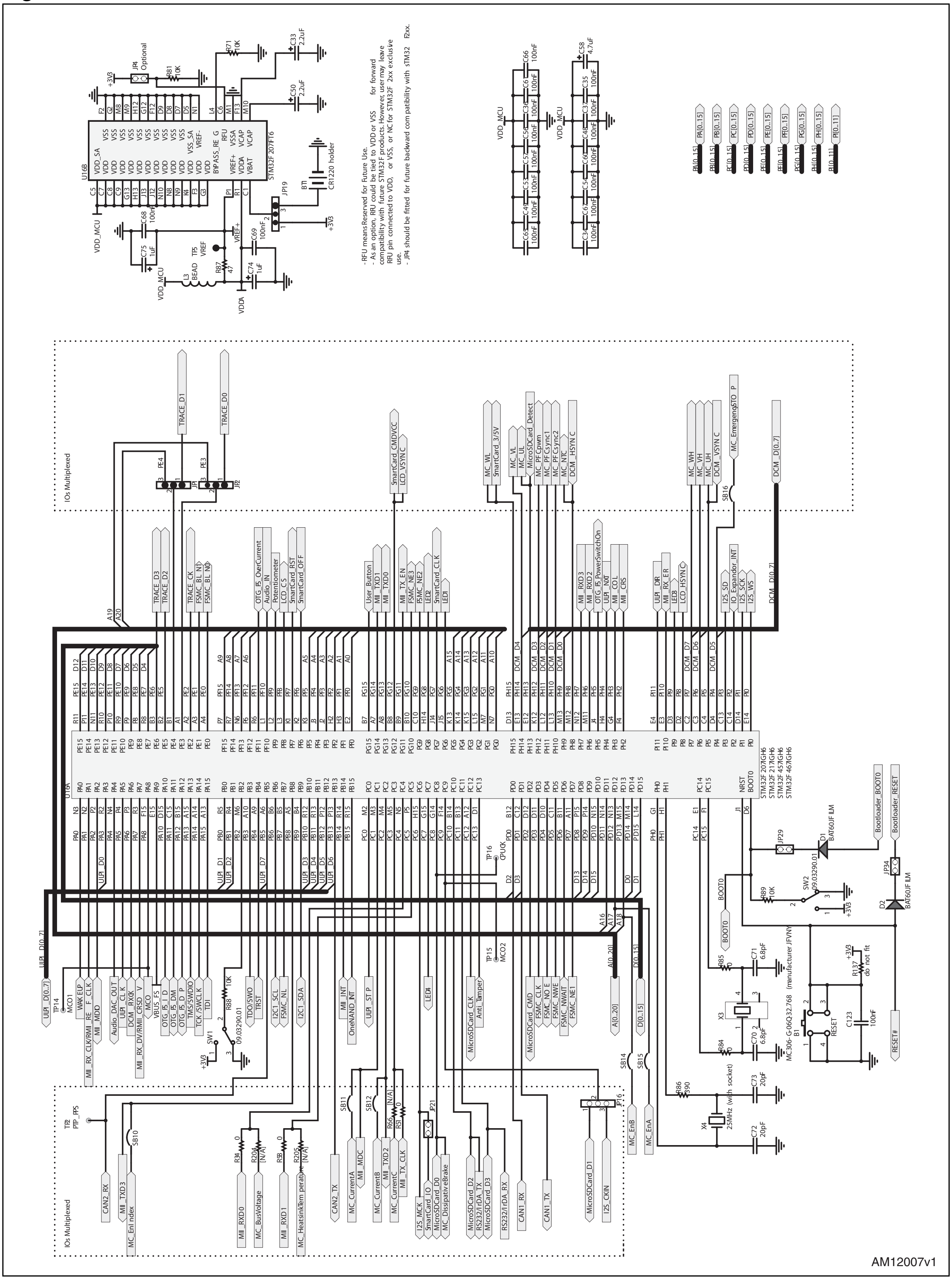


Figure 42. STM3220-21-45-46G-EVAL_Ethernet

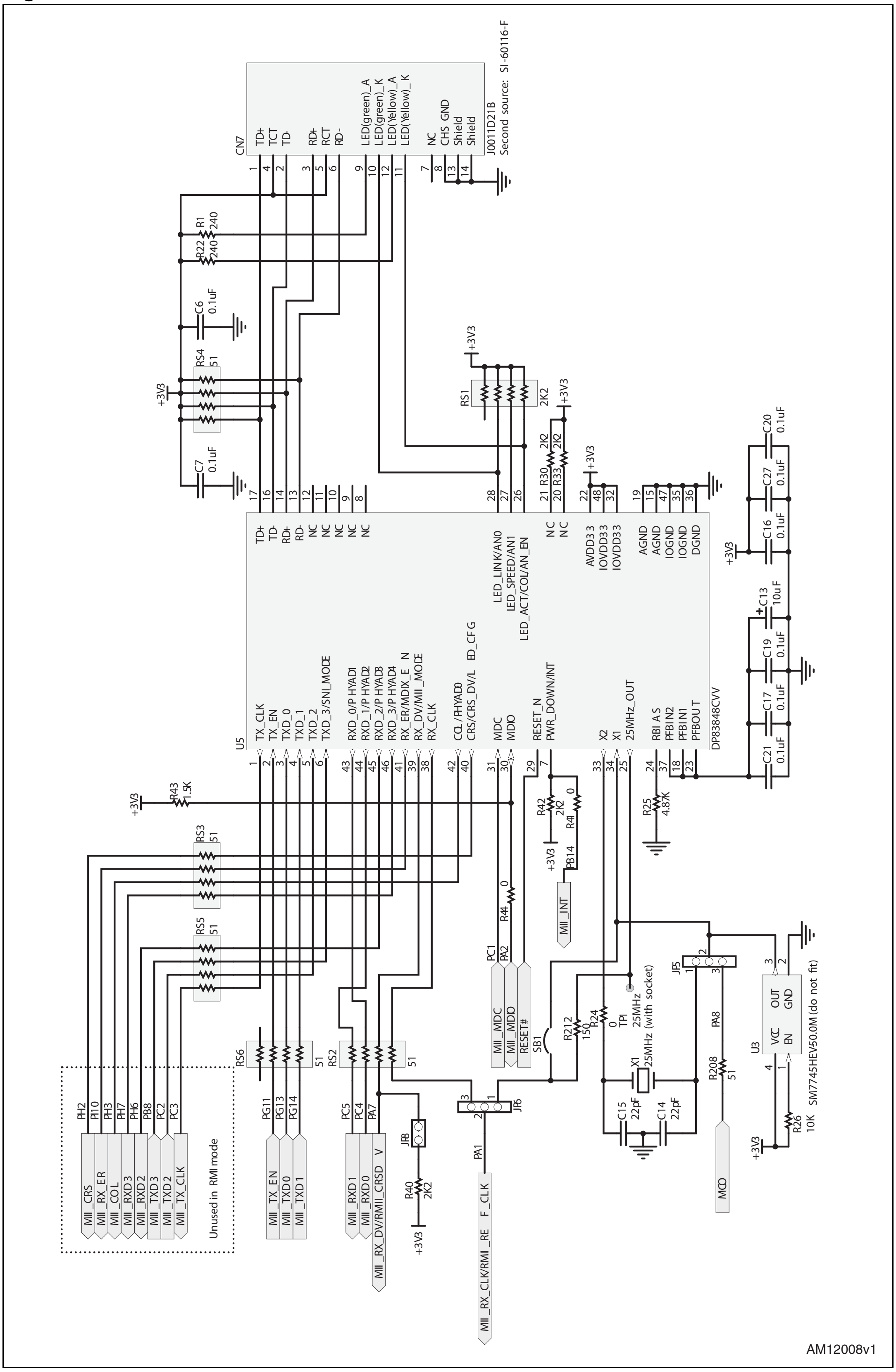


Figure 43. STM3220-21-45-46G-EVAL_LCD

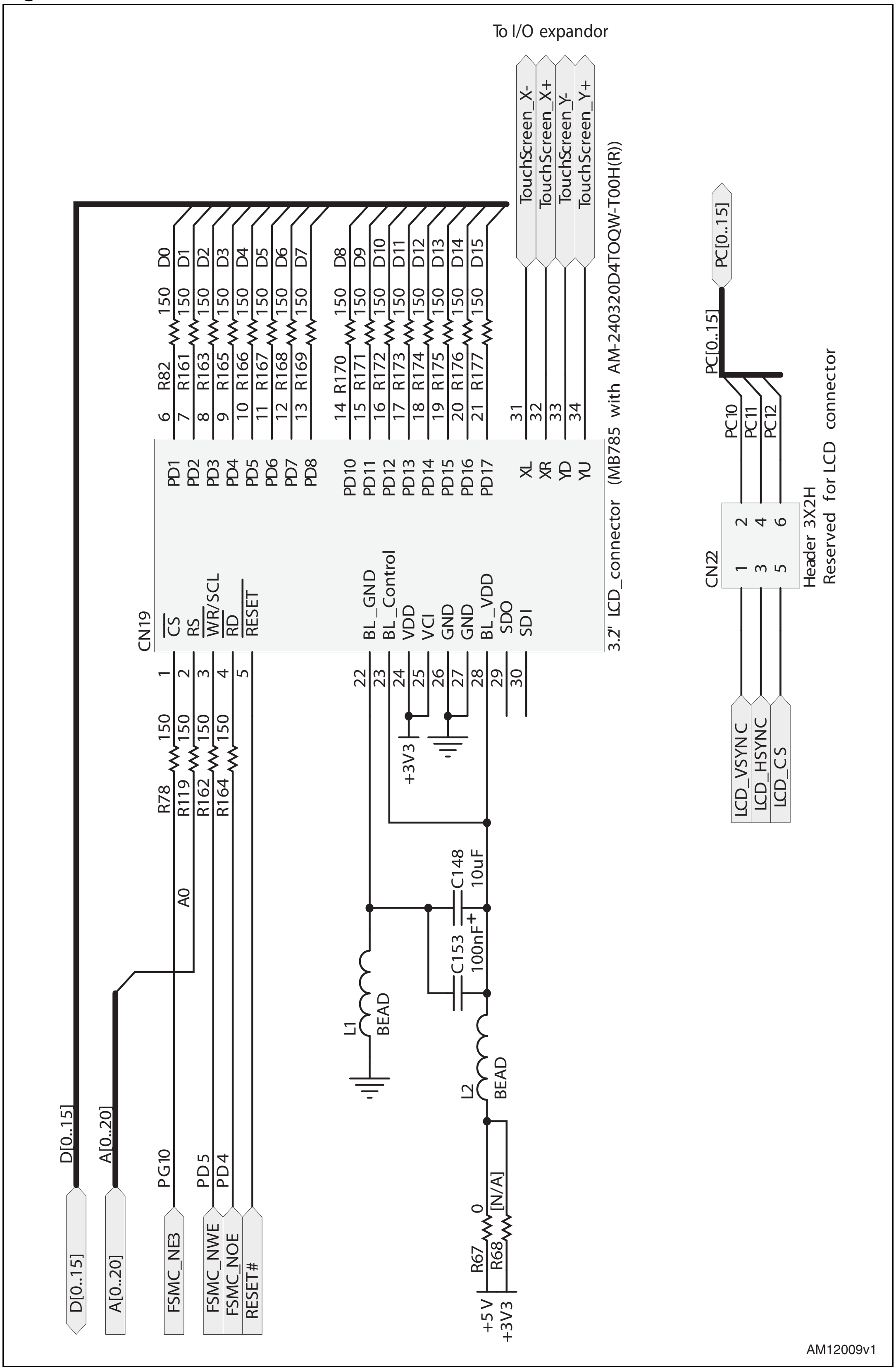


Figure 44. LCD module with both SPI and 16-bit interface

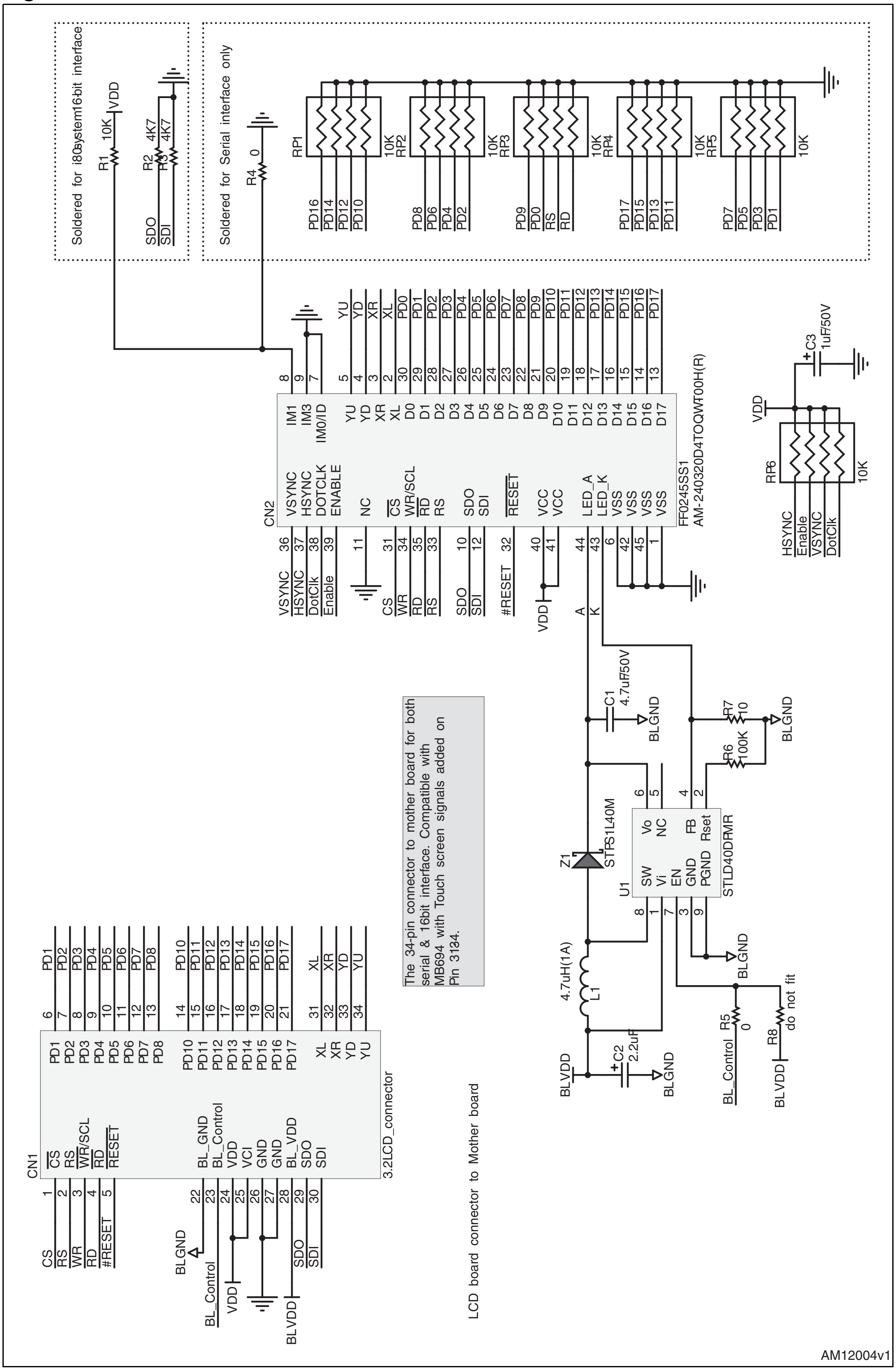


Figure 45. STM3220-21-45-46G-EVAL I/O peripherals

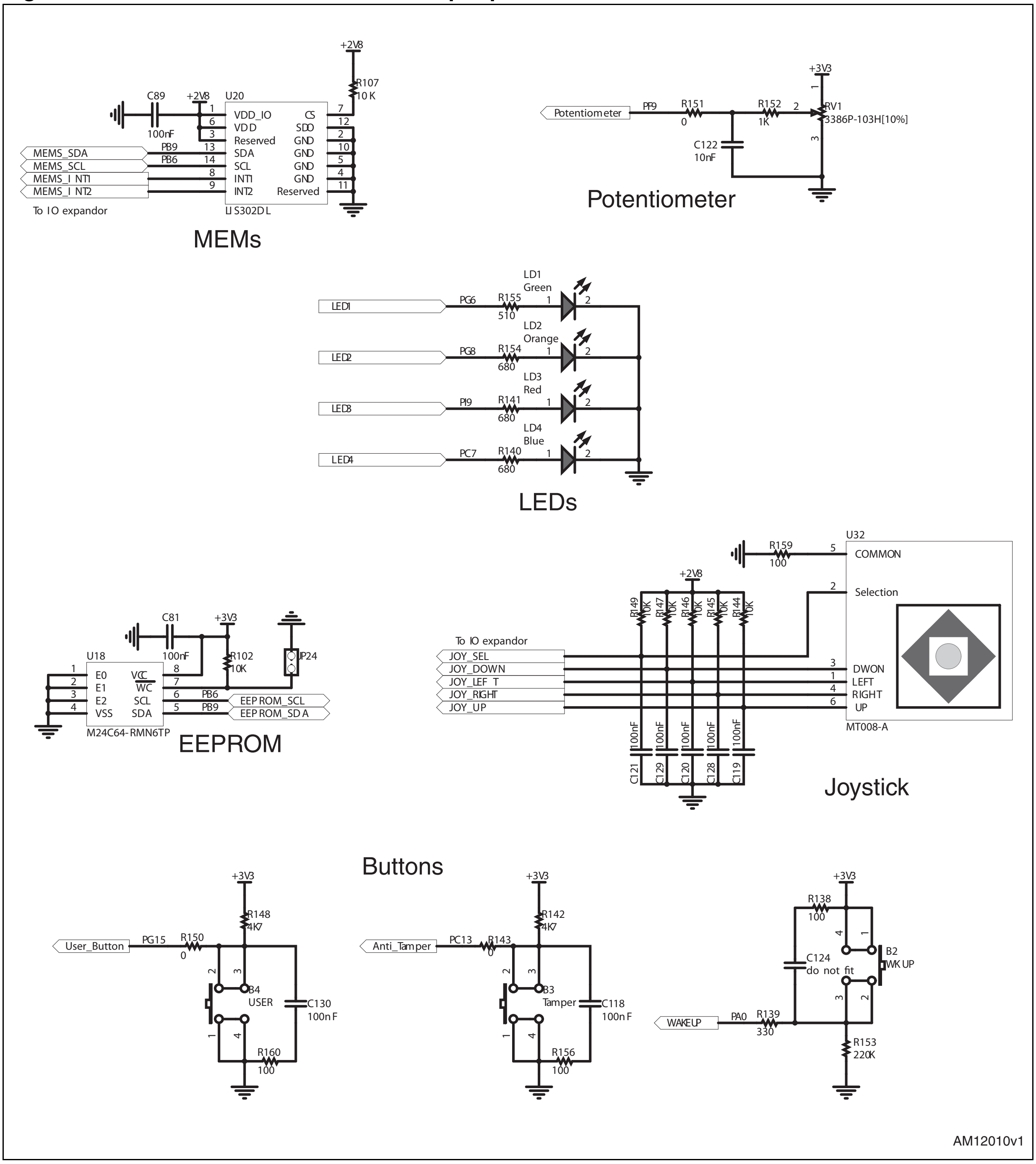


Figure 46. STM3220-21-45-46G-EVAL I/O_Expander

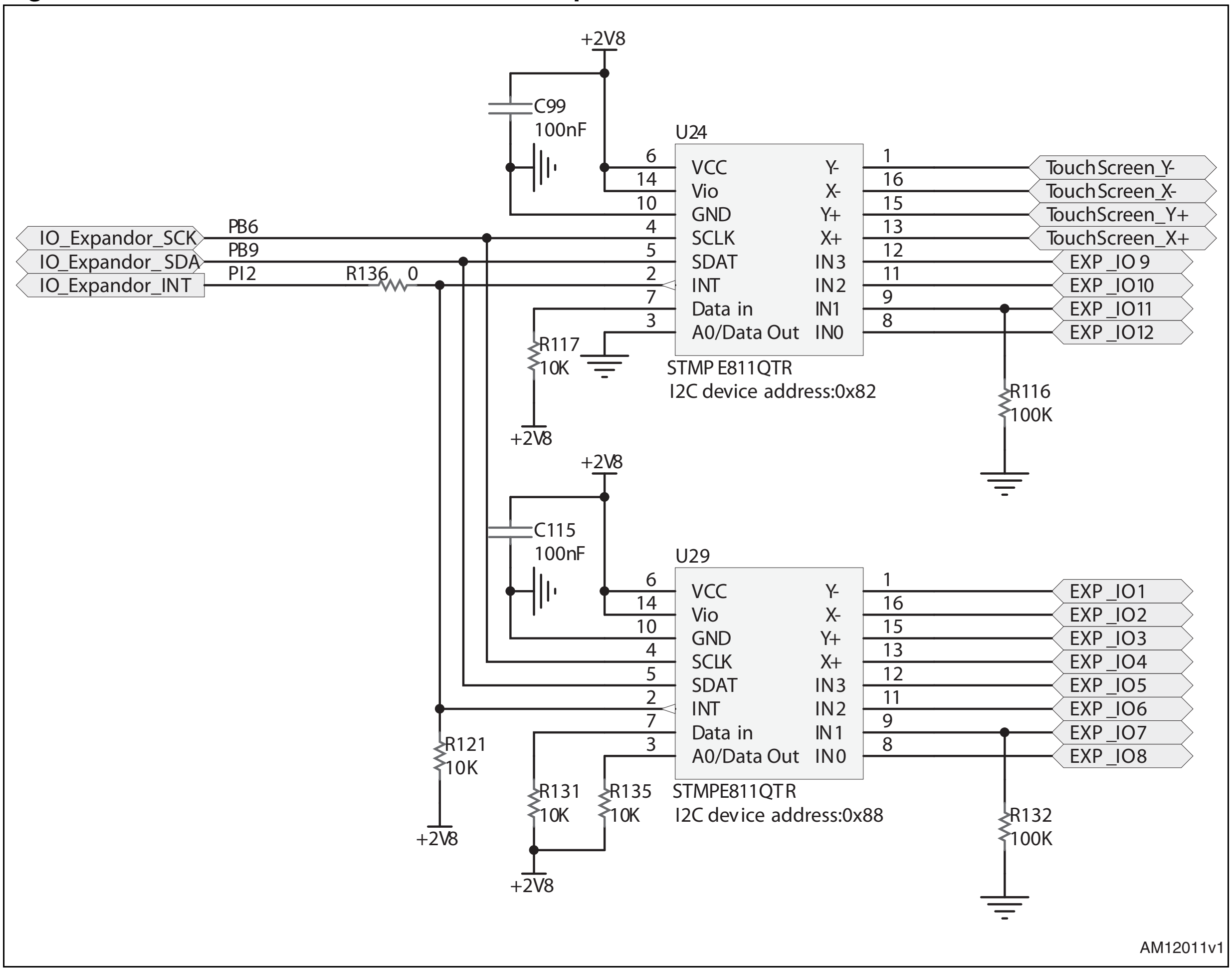


Figure 47. STM3220-21-45-46G-EVAL JTAG and trace

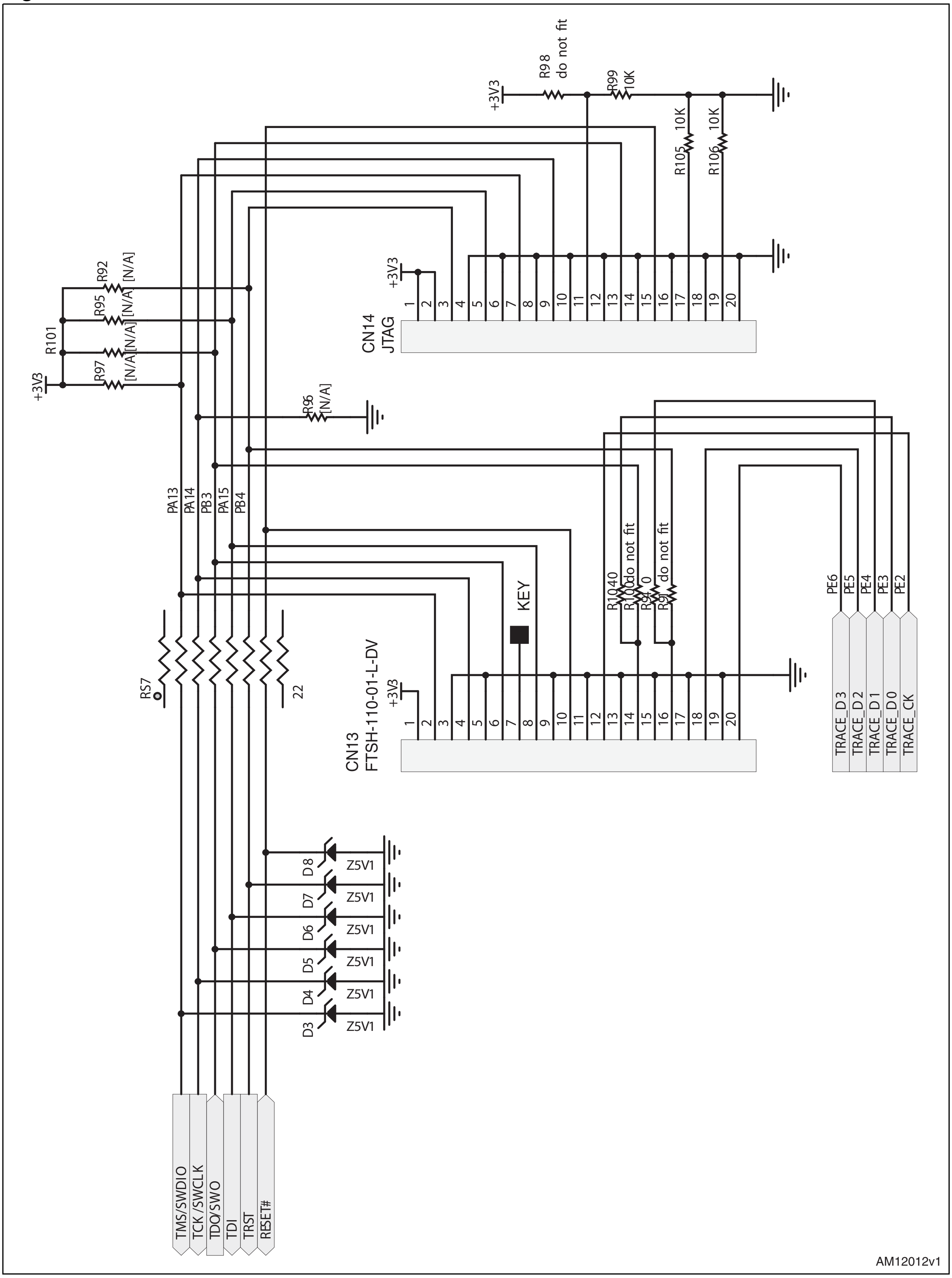


Figure 48. STM3220-21-45-46G-EVAL_Power

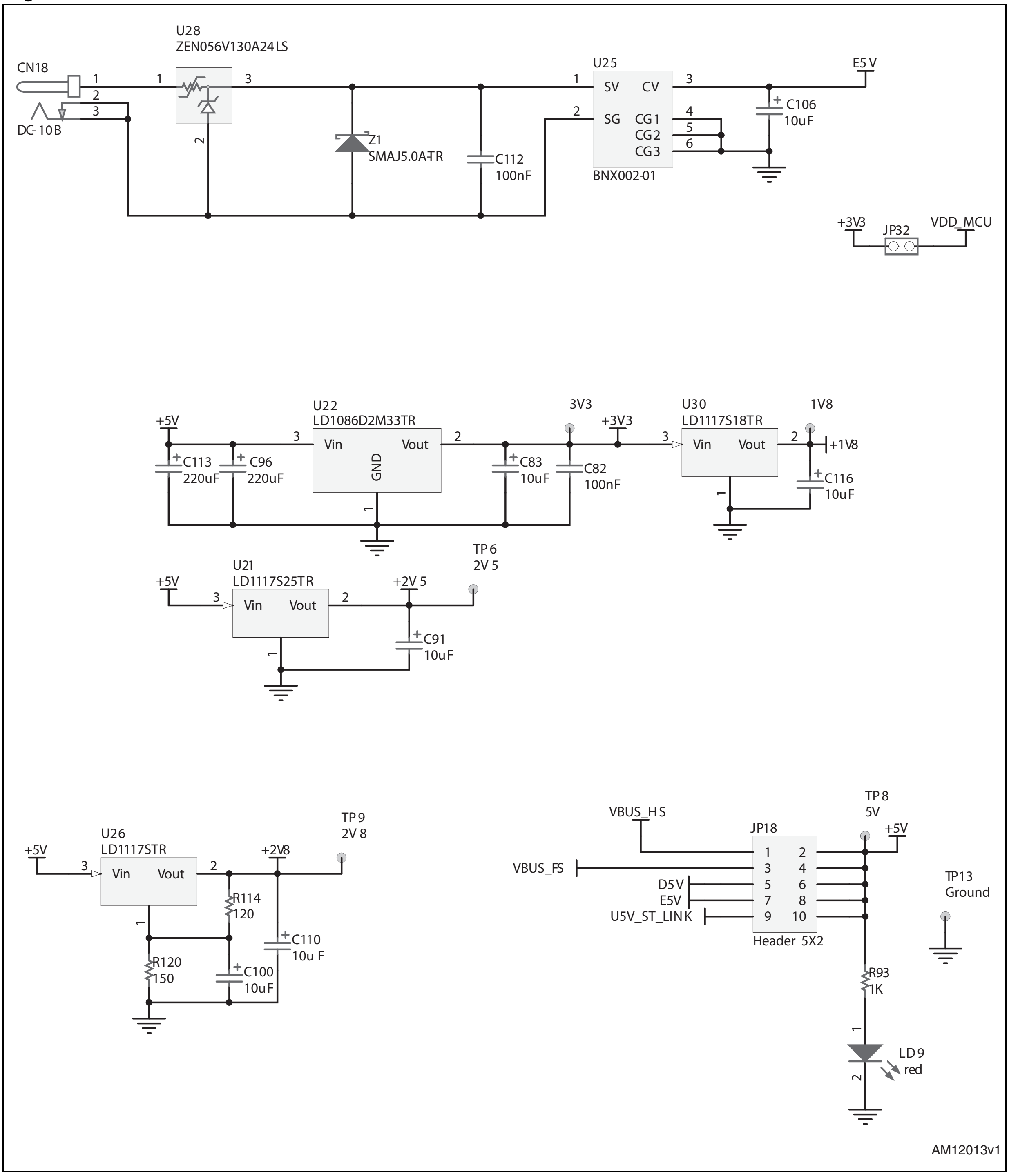
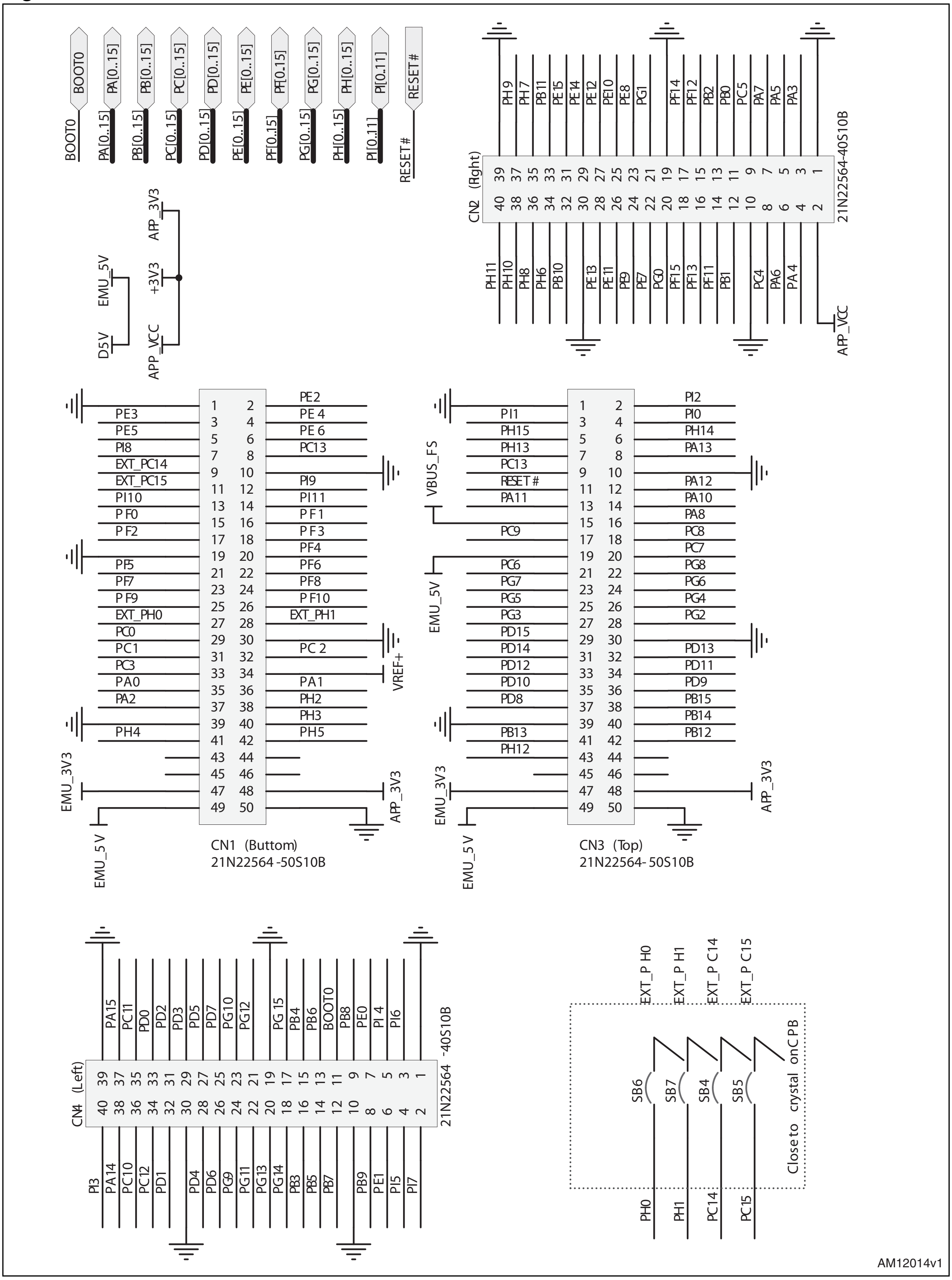


Figure 49. STM3220-21-45-46G-EVAL_Extension connector



11 References

1. STEVAL-IHP004V1 schematics diagram
2. M24LR64, datasheet
3. STM322xG-EVAL evaluation board - UM1057, user manual
4. AN3128, application note
5. ZigBee-Dual IFs EEPROM Adapter for STM322xG-EVAL - UM1519, user manual
6. SPZB32W1A2.1 module, datasheet
7. UM0853, user manual

12 Revision history

Table 23. Document revision history

Date	Revision	Changes
24-Jul-2012	1	Initial release.
12-Oct-2012	2	Modified: code in the *Section 6.3.1*