UM1076
User manual

C Library for ST7590 and STM32™

1 Introduction

This document describes the library (driver) for interfacing STMicroelectronics™ power line modem demonstration board (EVALST7590), with an STM32 microcontroller. This helps the developer to test the solution and more quickly design their application. The library is designed to be universal and easily adapted to any microcontroller with minor changes. The minimum requirements for the microcontroller are a UART and a timer. The library was tested and compiled for the STM32 microcontroller in IAR™ programming environment.

For example, the complete communication node shown in *Figure 5* (containing the ST7590 demonstration board (EVALST7590-1) and a microcontroller connectivity gateway demonstration board (STEVAL-PCC012V1) is used. An example application, which is part of the library, can be run on the mentioned communication node directly without any necessary adaptations. The example shows communication between the nodes, where the service and base node establish a logical connection. The service node periodically sends, every two seconds, a message which is received by the base node. Both service node and base node indicate sending or receiving of the packet by toggling of the LED.

The library implements all the ST7590 commands that this device offers. The library was tested on this communication node.

The use of this library with other platforms and microcontrollers and its interconnection is described in the ST7590 databrief, the UM1038 user manual, and the ST7590 datasheet and product related documents.

Contents

List of tables

List of figures

2 Library details

Project created in:	IAR programming environment version 5.50
Target processor:	Any, project tested on STM32
Language:	C language
Reference:	ST7590 databrief
	ST7590 datasheet and product related documents - for all details regarding the commands
	UM1038 - hardware interconnection of the STM32™, SPEAr™3xx, and ST75xx.

3 Modem commands

This section shows all the commands available with their codes together with corresponding filenames that the commands are located in. The library implements all the commands the modem offers.

Table 1 shows in which library file the specific command can be found. The “xxx” in the command name stands for: request, response, indication or confirm.

Table 1. Assignment of command names to the library files

Command group	Command codes	Filename
CL432_Establish_xxx	<0xA0> <0xA1>	CL432_Establish.c
CL432_Release_xxx	<0xA2> <0xA3>	CL432_Release.c
CL432_Join_xxx	<0xA4>	CL432_Join.c
CL432_Leave_xxx	<0xA5>	CL432_Leave.c
CL_432_GetSNSession	<0xB4>	CL432_GetSNS.c
DL_Data_xxx	<0xA6> <0xA7> <0xA8>	DL_Data.c
DL_Broadcast_xxx	<0xA9>	DL_Broadcast.c
DL_Reply_xxx	<0xAA> <0xAB> <0xAC>	DL_Reply.c
DL_UpdateReply_xxx	<0xAD><0xAE><0xAF>	DL_UpdateReply.c
PIB_LIST_GET_xxx	<0xB1><0xB2>	PIB_LIST_GET.c
PIB_GET_xxx	<0xB5><0xB6>	PIB_GET.c
PIB_SET_xxx	<0xB7><0xB8>	PIB_SET.c
MAC_MLME_REGISTERSTATE_Indication(1)	<0xB9>	MAC_MLME.c
MAC_TESTmode	<0xBA>	MAC_TESTmode.c
MAC_Establish_xxx	<0xC1><0xC2><0xC3><0xC4>	MAC_Establish.c
MAC_Relase_xxx	<0xC5><0xC6><0xC7>	MAC_Release.c
MAC_Join_xxx	<0xC8><0xC9><0xCF><0xD0>	MAC_Join.c
MAC_Leave_xxx	<0xCA><0xCB><0xDA>	MAC_Leave.c
MAC_Data_xxx	<0xCC><0xCD><0xCE>	MAC_Data.c
MAC_Unregister_xxx	<0xD1><0xD2><0xD3>	MAC_Unregister.c
MAC_Promotion_xxx	<0xD4><0xD5><0xD6>	MAC_Promote.c
MAC_Demotion_xxx	<0xD7><0xD8><0xD9>	MAC_Demote.c
PHY_DATAxxx	<0xE1><0xE2><0xE3>	PHY_Data.c
PRIME_GetState, PRIMEConfig, SW_Reset	<0xB0><0xB3><0x90>	cmd_msg.c

1. Referenced also as MAC_MLME_LIST_GETxxx.

4 Global data structure (in cmd_msg.h)

The global data structure is used to inform the user about the result of the last used command:

```
#define PHYDefine_PHY_HDR_DATA_SIZE 7
#define MAX_MESSAGE_LENGTH 1024
#define MAX_SN_LENGTH 1024

struct ST7590_Status_type{
  unsigned char Last_Snd_CMD_ID;
  unsigned char Last_Rcv_CMD_ID;
  unsigned char LastErrCode;
  unsigned short LastStateField;
  unsigned char LastResultCode;
  unsigned char LastType;
  unsigned short LastCfB;
  unsigned char LastReleaseConnectionReason;
  unsigned char LastConnectionHandler[4];
  unsigned char LastSNA[6];
  unsigned char LastEUI48[6];
  unsigned char LastReceivedMessage[MAX_MESSAGE_LENGTH];
  unsigned short LastReceivedMessageLength;
  unsigned short TX_Buffer_Len_Used;
  unsigned short RX_Buffer_Len_Used;
  //---------------CL432-----------
  unsigned short LastSNLength;
  unsigned char LastSN[MAX_SN_LENGTH];
  unsigned short LastDA; //Destination Address
  unsigned short LastSA; //Source Address
  unsigned char LastDSAP;
  unsigned char LastSSAP;
  //---------------DL--------------
  unsigned char LastTXStatus; //TXS
  unsigned char LastLC; //Link Class
  unsigned char LastStatus; //Link Class
```

```
  unsigned char LastLR; //Length requested
  //---------------PIB--------------
  unsigned short PIBAttr;
  unsigned short PIBListAttr;
  //---------------PHY--------------
  unsigned char Level;
  unsigned char Scheme;
  unsigned long Timer;
  unsigned char SNR;
  unsigned char RQ; // Copy the data in the Data Indication message: 10 are the LEN...RQ fields size
  unsigned char PHY_Header[PHYDefine_PHY_HDR_DATA_SIZE];
  //--------------------------------
  char GetStateOk;
};

extern struct ST7590_Status_type ST7590_Status;
```

Table 2. Example of the field description and its link to the commands

Field	Note
Last_Snd_CMD_ID	Send command ID filled by all CmdMsgSnd_xxx
Last_Rcv_CMD_ID	Received command ID filled by all CmdMsgRcv_xxx
LastErrCode	Always filled by all CmdMsgRcv_xxx
LastStateField	Always filled by all CmdMsgRcv_xxx
LastResultCode	Filled by some CmdMsgRcv_xxx commands
LastType	Filled by CmdMsgRcv_MACEstConf
LastCfB	Filled by CmdMsgRcv_MACEstInd
LastReleaseConnectionReason	Filled by CmdMsgRcv_MACRelInd
LastConnectionHandler	Filled by some CmdMsgRcv_xxx commands
LastSNA	Filled by CmdMsgRcv_MACRegisterIndication
LastEUI48	Filled by some CmdMsgRcv_xxx commands
LastReceivedMessage	Filled by some CmdMsgRcv_xxx commands
LastReceivedMessageLength	Filled by some CmdMsgRcv_xxx commands
TX_Buffer_Len_Used	Filled by some CmdMsgRcv_xxx commands
RX_Buffer_Len_Used	Filled by some CmdMsgRcv_xxx commands(1)

1. See the ST7590 datasheet and product related documents for the meaning and usage of the remaining fields that are not mentioned in *Table 2*.

5 Commands

The user calls only send commands: CmdMsgSnd_xxx. The corresponding receiving command CmdMsgRcv_xxx is called automatically according to the CMD_ID that is answered by modem ST7590.

Communication with PLM ST7590 is done by the procedure shown in *Table 3*.

Table 3. Steps used to communicate with the modem

Step	User	System, library
1	Calls: CmdMsgSnd_xxx	Sends xxx command Wait for the first character coming from ST7590 Wait 50 ms Receive complete incoming message Sets ST7590_Status data structure
2	Checks: ST7590_Status	

In *Section 5.1* to *Section 5.5* the first five files are described in detail. These five files make it possible to build the simplest application:

- cmd_msg.c: checking if the modem is operational
- MAC_MLME.c: checking if the modem is registered in the network
- MAC_Establish.c: logical channel establishment
- MAC_Data.c: data transfer from “Base node” to “Service node” or vice versa
- MAC_Leave.c: logical channel break-down management

The rest of the information about the function and commands in the library can be found in related files: the ST7590 datasheet and product related documents and directly in the header files of corresponding commands.

5.1 cmd_msg.h

The function of the command and appropriate usage is described in detail in the ST7590 datasheet and product related documents.

Table 4. Available commands in cmd_msg.c file

Commands	Command code	Filename
PRIME_GetState	<0xB0>	cmd_msg.c
PRIMEConfig	<0xB3>	
SW_Reset	<0x90>	

Send commands:

void CmdMsgSnd_PrimeGetState(**void**);

void CmdMsgSnd_PrimeConfig(**unsigned char** Configuration);

void CmdMsgSnd_SWReset(**void**);

Receive commands (not called by user):

void CmdMsgRcv_PrimeGetState(**unsigned char** Buffer[]);

void CmdMsgRcv_PrimeConfig(**unsigned char** Buffer[]);

5.2 MAC_MLME.h

Table 5. Available commands in MAC_MLME.c file

Commands	Command code	Filename
MAC_MLME_REGISTERSTATE_Indication	<0xB9>	MAC_MLME.c

Send commands:

void CmdMsgSnd_MACRegisterStateIndication(**void**);

Receive commands:

void CmdMsgRcv_MACRegisterIndication(**unsigned char** Buffer[]);

5.3 MAC_Data.h

Table 6. Available commands in MAC_Data.c file

Commands	Command code	Filename
MAC_Data_Request	<0xCC>	MAC_Data.c
MAC_Data_Confirm	<0xCD>	
MAC_Data_Indication	<0xCE>	

Send commands:

void CmdMsgSnd_MACDataReq(**unsigned char** ConnectionHandler[4], **char** DataToSend[], **short** DataToSendLength);

void CmdMsgSnd_MACDataConf(**void**);

void CmdMsgSnd_MACDataInd(**void**);

Receive commands:

void CmdMsgRcv_MACDataReq(**unsigned char** Buffer[]);

void CmdMsgRcv_MACDataConf(**unsigned char** Buffer[]);

void CmdMsgRcv_MACDataInd(**unsigned char** Buffer[]);

5.4 MAC_Establish.h

Table 7. Available commands in MAC_Establish.c file

Commands	Command code	Filename
MAC_Establish_Request	<0xC0>	MAC_Data.c
MAC_Establish_Indication	<0xC1>	
MAC_Establish_Response	<0xC2>	
MAC_Establish_Confirm	<0xC3>	

Send commands:

void CmdMsgSnd_MACEstReq(**unsigned char** EUI48[6], **char** Type, **char** ARQ, **unsigned short** CfB, **char** AdditionalDataToSend[], **short** AdditionalDataToSendLength);

void CmdMsgSnd_MACEstInd(**void**);

void CmdMsgSnd_MACEstRes(**unsigned char** ConnectionHandler[4], **char** Answer, **char** AdditionalDataToSend[], **short** AdditionalDataToSendLength);

void CmdMsgSnd_MACEstConf(**void**);

Receive commands:

void CmdMsgRcv_MACEstReq(**unsigned char** Buffer[]);

void CmdMsgRcv_MACEstInd(**unsigned char** Buffer[]);

void CmdMsgRcv_MACEstRes(**unsigned char** Buffer[]);

void CmdMsgRcv_MACEstConf(**unsigned char** Buffer[]);

5.5 MAC_Release.h

Table 8. Available commands in MAC_Release.c file

Commands	Command code	Filename
MAC_Relase_Request	<0xC4>	MAC_Release.c
MAC_Relase_Indication	<0xC5>	
MAC_Relase_Response	<0xC6>	
MAC_Relase_Confirm	<0xC7>	

Send commands:

void CmdMsgSnd_MACRelReq(**unsigned char** ConnectionHandler[4]);

void CmdMsgSnd_MACRelInd(**void**);

void CmdMsgSnd_MACRelRes(**unsigned char** ConnectionHandler[4], **char** Answer);

void CmdMsgSnd_MACRelConf(**void**);

Receive commands:

void CmdMsgRcv_MACRelReq(**unsigned char** Buffer[]);

void CmdMsgRcv_MACRelInd(**unsigned char** Buffer[]);

void CmdMsgRcv_MACRelRes(**unsigned char** Buffer[]);

void CmdMsgRcv_MACRelConf(**unsigned char** Buffer[]);

6 Library support files

6.1 time_counter.h

This file contain the prototype of the functions which implement the timers being used for precise timing when communicating with the modem or can be used for timing events in the user's own application. The user can use up to six timers numbered from 0 to 5:

```
typedef enum{
  TimerCounter0 = 0,
  TimerCounter1 = 1,
  TimerCounter2 = 2,
  TimerCounter3 = 3,
  TimerCounter4 = 4,
  TimerCounter5 = 5,
} TimerNumberEnum;

//Results
#define TimerElapsed 1
#define TimerNotElapsed 0
#define TimerDisabled 2000000000

//Time in milliseconds
#define SHORTTIME 50
#define ANSWERTIME7590 50
#define TIMERSMAXCOUNT 6
```

void TIMER_timeToElapse(**TimerNumberEnum** TimerNo, **int** ms);

- set the timer TimerNo to ms milliseconds, enables it and starts countdown.

int TIMER_timeElapsed(**TimerNumberEnum** TimerNo);

- if the timer TimerNo reaches zero, this function returns TimerElapsed value
- if the timer is disabled, this function returns TimerElapsed value
- if the timer has not yet reached zero, this function returns TimerNotElapsed

void TIMER_DisableTimer(**TimerNumberEnum** TimerNo);

- disables the timer TimerNo

void TIMER_waitFor(**TimerNumberEnum** TimerNo, int ms);

- finishes after ms milliseconds using the timer TimerNo.

void TIMER_HW_Init(**void**);

- configures the timer or SysTick hw used for timing.

6.2 history.h

This file contains the prototype of the functions which implement a simple circular data logging system. It logs 4 values: ErrorCode, StatusCode, CommandCodeSnd and CommandCodeRcv whenever the putIn function is called. An array of the logged bytes, ErrorEvidence, can be investigated in the programming environment during debugging. Array capacity is 4 x 255 bytes.

#define maxerrors 255

unsigned char pointer;

struct{

unsigned char Error;

unsigned short Status;

unsigned char CommandSnd;

unsigned char CommandRcv;

} ErrorEvidence[maxerrors];

void initErrorList(**void**);

- inits the data logger.

void putIn(**unsigned char** ErrorCode, **unsigned short** StatusCode, **unsigned char** CommandCodeSnd, **unsigned char** CommandCodeRcv);

- inserts the new values into the ErrorEvidence array.

6.3 serial_port.h

This file contains functions that take care of incoming packets that were received by the power line modem. According to the received packet, the main function calls corresponding CMD_rcv_xxx functions from other files from *Section 5*.

Function exported from file: serial_port.h

Called by user:

void SerialportDataRcv(**void**);

This function is called automatically after some period of time after calling any CmdMsgSnd_xxx command by user. In order not to miss any incoming message, the user should check the status of the CTS or DFU_FORCE line of ST7590 (see the ST7590 datasheet and product related documents) or call e.g. PRIME_GetState regularly and according to the received State Field react with the corresponding action.

6.4 stm32_uart.h

This file implements a buffered UART interface.

The beginning of the file is dedicated for physical definition of the pins of the microcontroller used for UART interface:

#define TXD_RXD_remap 1 // 1: Remap (TX/PB6, RX/PB7)

#define TXD_pin GPIO_Pin_6 //used UART

#define TXD_port GPIOB //used UART

#define RXD_pin GPIO_Pin_7 //used UART

#define RXD_port GPIOB //used UART

#define TREQ_pin GPIO_Pin_9 //T_REQ

#define TREQ_port GPIOB //T_REQ

#define US1 0

#define PLM US1

- This directive defines that PLM constant is UART0 of the microcontroller.

#define DirectWrite

This directive defines which function is used to write to UART:

- ComWrt = ComWrt_direct (if DirectWrite defined) all the data requested to be sent over UART is sent in one row and the application does not continue unless the transfer has finished.
- ComWrt = ComWrt_buff data is stored in buffer and sent by interrupt whenever there is computational time.

Functions for buffered UART (parameter portNumber is present for legacy reason only, has no influence on functionality):

void UART1_init(**void**);

void UART2_init(**void**);

- configures UART interfaces of the used microcontroller.

int GetInQLen (**int** portNumber, **char** UART);

- gives the length of the data present in the input buffer for the chosen UART.

int FlushOutQ (**int** portNumber, **char** UART);

- clears the output buffer for the chosen UART.

int FlushInQ (**int** portNumber, **char** UART);

- clears the input buffer for the chosen UART.

int ComWrtByte (**int** portNumber, **char** byte, **char** UART);

- writes one byte to the chosen buffered UART.

int ComWrt_buff (**int** portNumber, **char** buffer[], size_t count, **char** UART);

- writes array buffer of size_t length to chosen buffered UART.

void ComWrt_direct(int portNumber, **unsigned char*** data_buffer, **unsigned short** Nb_bytes, c**har** USART);

- writes array buffer of size_t length to chosen buffered UART.

int ComRd (**int** portNumber, **char** buffer[], **int** count, **char** UART);

- reads data from chosen buffered UART to array buffer. Count indicates number of bytes read.

char ComRdByte (**int** portNumber, **char** UART);

- reads one byte from chosen buffered UART.

7 Communication example

This communication example demonstrates two nodes in PRIME network: base node and service node. The service node sends a packet with data 0x11 or 0x22 every two seconds. Whenever the base node receives 0x11, it switches on an LED. If it receives 0x22, it switches it off.

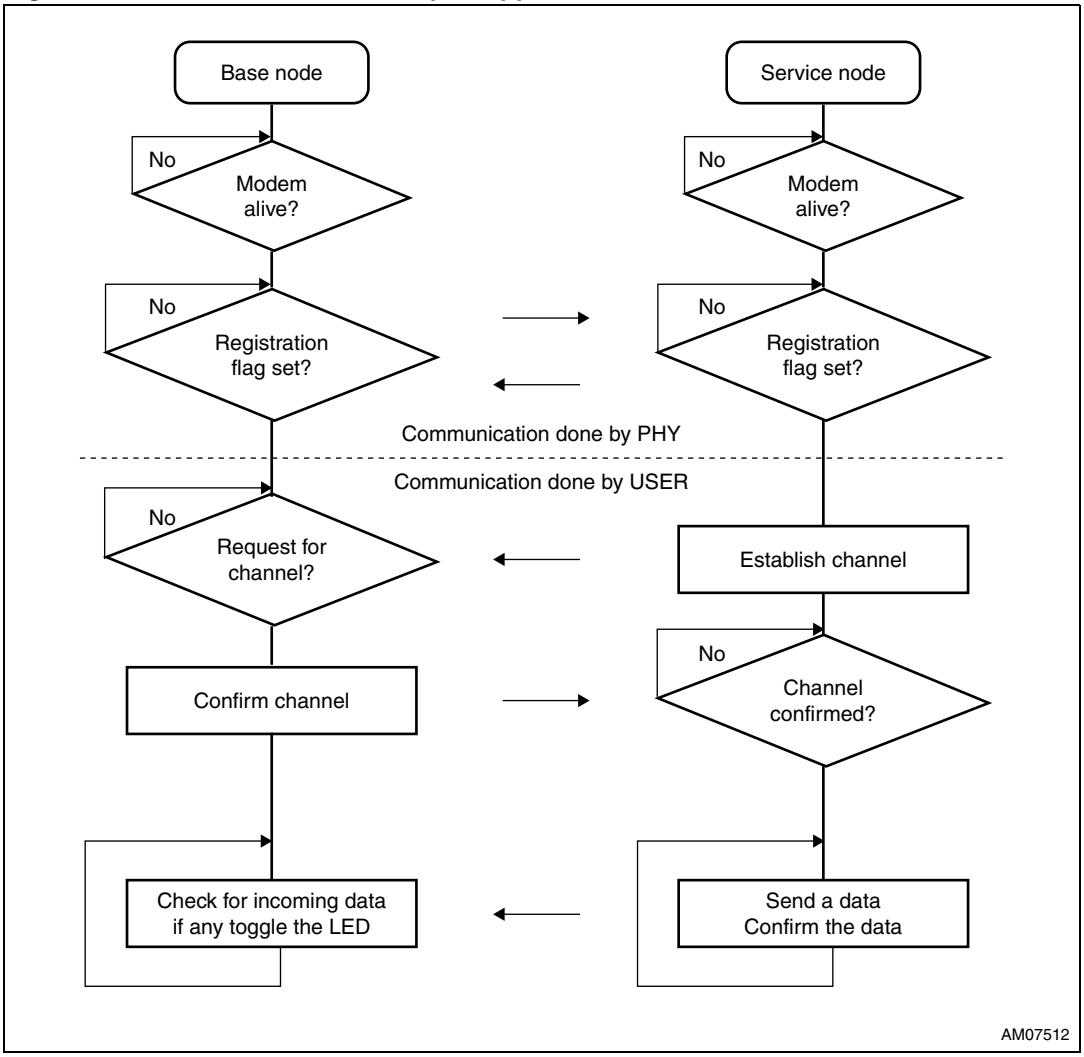
This example application uses:

- ST7590 library described in the chapters above
- Two files: main.c and functions.c.

7.1 Application - flowchart

Communication example application is described in detail by the flowchart in *Figure 1*.

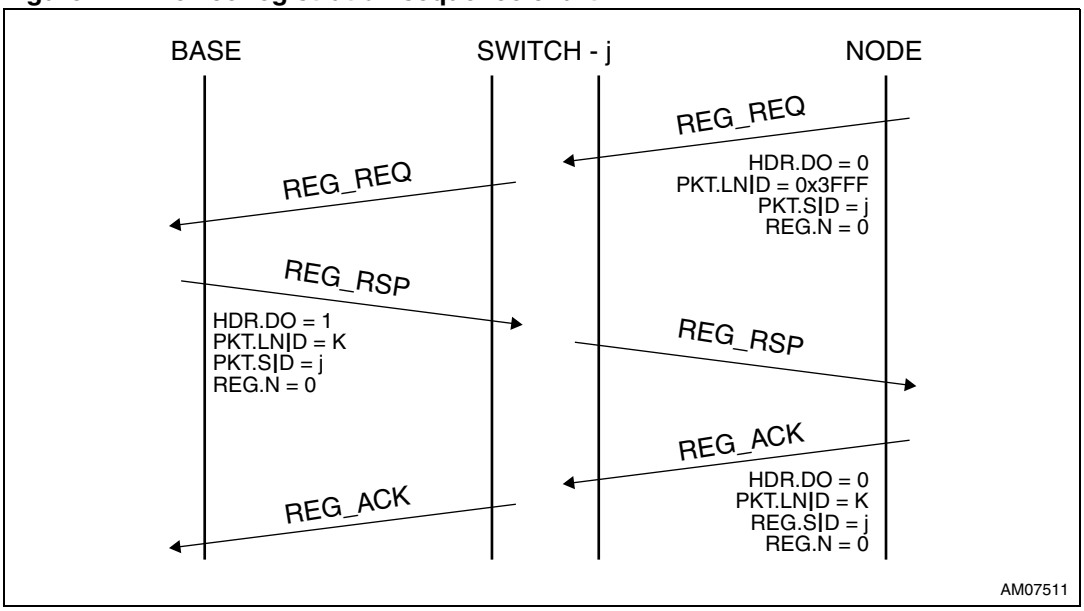
Figure 1. Communication example: application flowchart



7.2 Communication - sequence charts

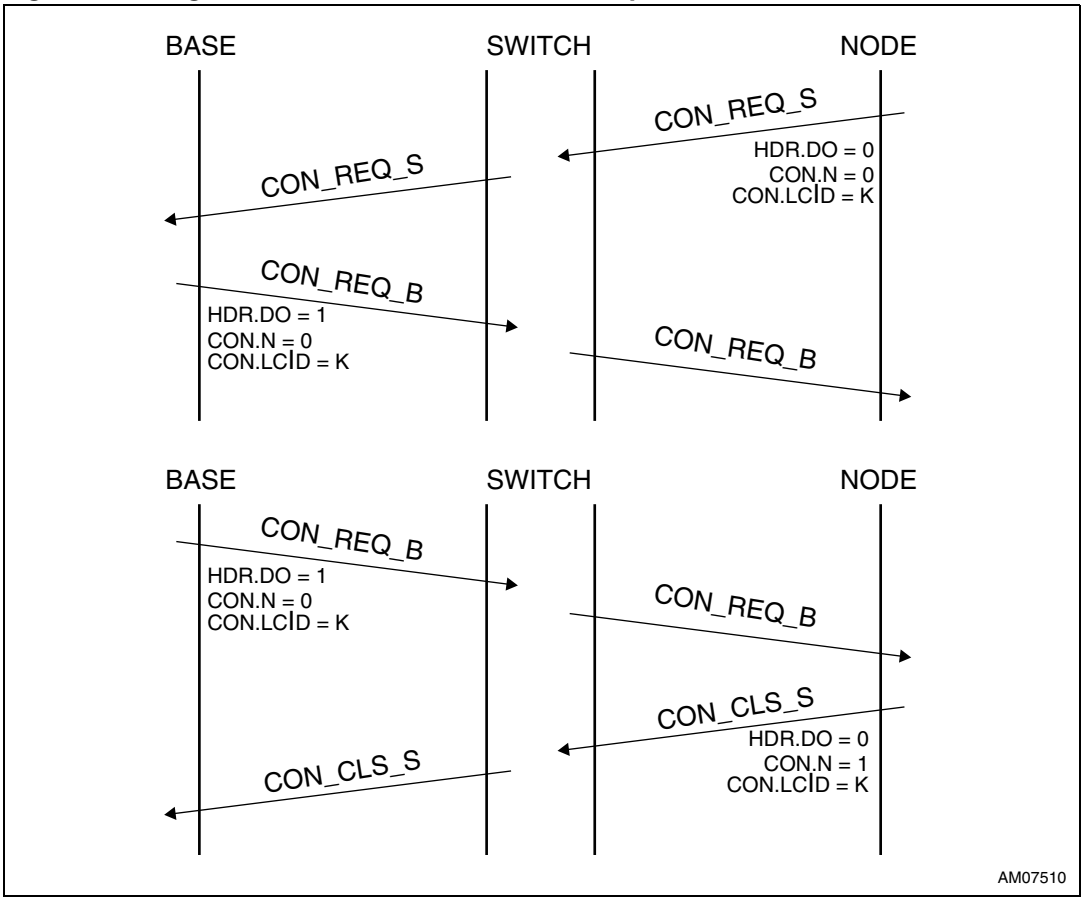
The registration process that is done automatically according to the “PRIME specification” (see www.prime-alliance.org) is shown in *Figure 2*.

Figure 2. Device registration sequence chart



Usage of the “PRIME primitives” during establishment of the channel is described in detail in *Figure 3*.

Figure 3. Logical connection establishment sequence chart



7.3 main.c

This file contains only the state machine that calls the corresponding function of each state from function.c file:

```
ActualStateEnum ActualState = MODEM_CHECK;

  while(1){
    switch(ActualState){
    case MODEM_CHECK: ActualState = modem_state_check();     break;

    case CLIENT_MASTER_DATA_SEND_req:     ActualState =
client_master_data_send_req();     break;
```

```
      case CLIENT_MASTER_DATA_CONFIRM_ind:    ActualState =
client_master_data_confirm_ind(); break;
      case CLIENT_MASTER_DATA_RECEIVE_ind:    ActualState =
client_master_data_receive_ind();     break;


      case SERVER_SLAVE_DATA_SEND_req:       ActualState =
server_slave_data_send_req();     break;
      case SERVER_SLAVE_DATA_RECEIVE_ind:    ActualState =
server_slave_data_receive_ind();     break;


       case IDLE_STATE::;    break;
    }
       ActualState = check_external_events(ActualState); //check for incoming packet
    }
```

7.4 functions.c

This file contains the corresponding functions for each state of the state machine implemented in main.c:

```
ActualStateEnum modem_state_check(void){
   while(GetFPMAstatus(NO_EFFECT_HARD_WIRED_UART1_REMAPED, 0, NULL, 0,
NULL)!= 0){
   }
      InitDevice();
      if(Demonstration_data.nodeIdentification == DEVICE_CLIENT_MASTER_NODE)
      return CLIENT_MASTER_DATA_SEND_req;
    or else
     return SERVER_SLAVE_DATA_SEND_req;
}

//---------------------------- DEVICE_CLIENT_MASTER_NODE ----------------------
ActualStateEnum client_master_data_send_req(void){
   ST7570_Status__Data.Last_CONFIRM_CODE = LP_NOT_VALID;
   CMD_snd_PHY_DataRequest(&(Demonstration_data.dataToSend), 1);
   TIMER_timeToElapse(TimerCounter3, 700); //Confirm response interval is 290 - 560 ms
   return CLIENT_MASTER_DATA_CONFIRM_ind;
}
```

```
ActualStateEnum client_master_data_receive_ind(void){
    return IDLE_STATE;
}

ActualStateEnum client_master_data_confirm_ind(void){
  if(TIMER_timeElapsed(TimerCounter3))
    return IDLE_STATE;
   switch(ST7570_Status__Data.Last_CONFIRM_CODE){
  case LP_NOT_VALID:
    return CLIENT_MASTER_DATA_CONFIRM_ind;
  case LP_OK:
     if(Demonstration_data.dataToSend == 0x11)
       GPIO_ResetBits(ORANGE_LED1_Port, ORANGE_LED1_Pin);
     or else
      GPIO_ResetBits(RED_LED_Port, RED_LED_Pin);
    TIMER_timeToElapse(TimerCounter2, 300);
  default:
    return IDLE_STATE;
  }
}

//---------------------------- DEVICE_SERVICE_SLAVE_NODE ----------------------
ActualStateEnum server_slave_data_send_req(void){
  return IDLE_STATE;
}

ActualStateEnum server_slave_data_receive_ind(void){
  if(ST7570_Status__Data.Last_P_SDU[0] == 0x11)
    GPIO_ResetBits(ORANGE_LED1_Port, ORANGE_LED1_Pin);
  or else
    GPIO_ResetBits(RED_LED_Port, RED_LED_Pin);
   return IDLE_STATE;
}
```

```
//====================== END - Exchange data
==================================

ActualStateEnum check_external_events(ActualStateEnum InState){
   int incomming_packet_COMMAND_ID;
     ActualStateEnum returnValue = InState;

    if(Demonstration_data.nodeIdentification == DEVICE_CLIENT_MASTER_NODE)
     if(TIMER_timeElapsed(TimerCounter1))
     {
     TIMER_timeToElapse(TimerCounter1, 2000); //timer to send message every 2 s
      returnValue = CLIENT_MASTER_DATA_SEND_req;
       }

   incomming_packet_COMMAND_ID = check_incomming_packets();

    if(incomming_packet_COMMAND_ID == CMD_DATA_INDICATION_CODE){
     TIMER_timeToElapse(TimerCounter2, 300);        //timer to switch on the LED
     returnValue = SERVER_SLAVE_DATA_RECEIVE_ind;
   }

   if(incomming_packet_COMMAND_ID == CMD_SYNCHRO_INDICATION_CODE)
     TIMER_timeToElapse(TimerCounter4, 5000);

   if(TIMER_timeElapsed(TimerCounter4)){//automatic desynchro request every 5 s
     CMD_snd_DesynchroRequest();
      TIMER_DisableTimer(TimerCounter4);
   }

   if(TIMER_timeElapsed(TimerCounter2)){ //timer to switch off the LEDs
      GPIO_SetBits(ORANGE_LED1_Port, ORANGE_LED1_Pin);
     GPIO_SetBits(RED_LED_Port, RED_LED_Pin);
      TIMER_DisableTimer(TimerCounter2);
   }
```

```
    return returnValue;
}

int check_incomming_packets(void){
    if(RX_buffer_internal_not_empty){
      return CheckFPMAevents(0);
    }
    return 0;
}
```

8 Real application

Figure 5 shows an application NODE consisting of the EVALST7590-1 power line demonstration board and STEVAL-PCC012V1 connectivity gateway demonstration board. This setup represents a complete node (base node or service node). More details about interconnection of different platforms to power line demonstration boards can be found in the UM1038 user manual.

Figure 4. EVALST7590 and STEVAL-PCC012V1, block diagram

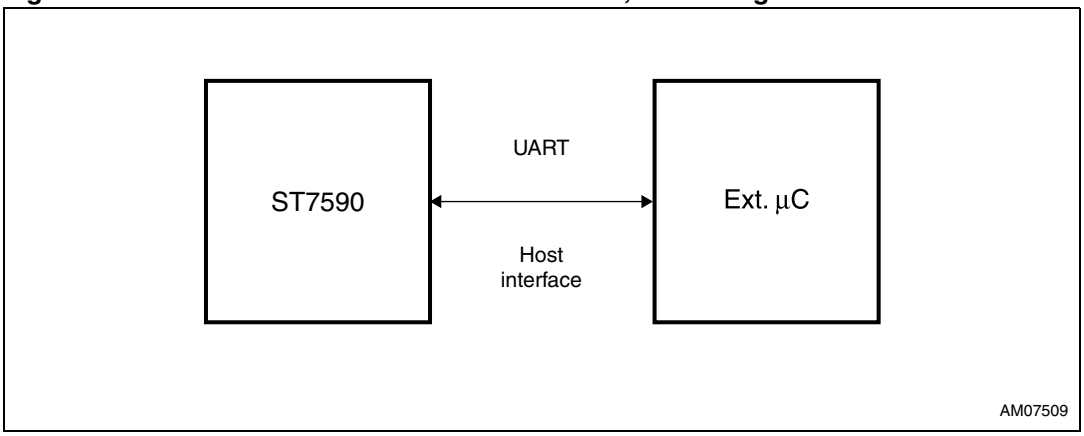
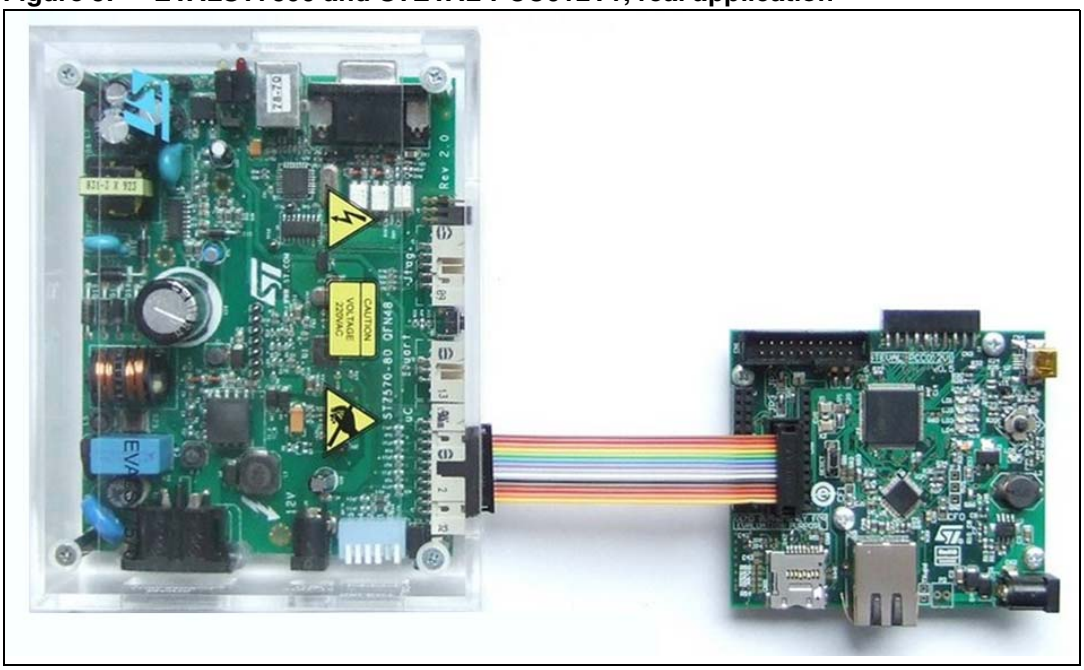


Figure 5. EVALST7590 and STEVAL-PCC012V1, real application



9 Revision history

Table 9. Document revision history

Date	Revision	Changes
15-Nov-2011	1	Initial release.