AN4274
Application note

The serial communication driver between the ST7580 and the STM32Fx

By Vincenzo Mormina

Introduction

This document describes the serial communication driver between the ST7580 and the STM32Fx.

This driver is designed from ST7580 power line modem devices.

This firmware is available by software license agreement only:

- http://www.st.com/software_license_agreement_liberty_v2
- http://www.st.com/software_license_agreement_image_v2

Please, ask your local ST office for further information.

Contents

1 Overview

The ST7580 serial communication driver provides services and functions to manage the serial communication (UART, USART) from the ST7580 power line networking SoC and the STM32Fx microcontroller.

The STM32Fx hardware resources required to connect the STM32Fx to ST7580 PLM are:

- Serial peripheral (Uart/Usart)
- 3 GPIO
- Timer

The ST7580 serial communication driver provides the function to configure these peripherals.

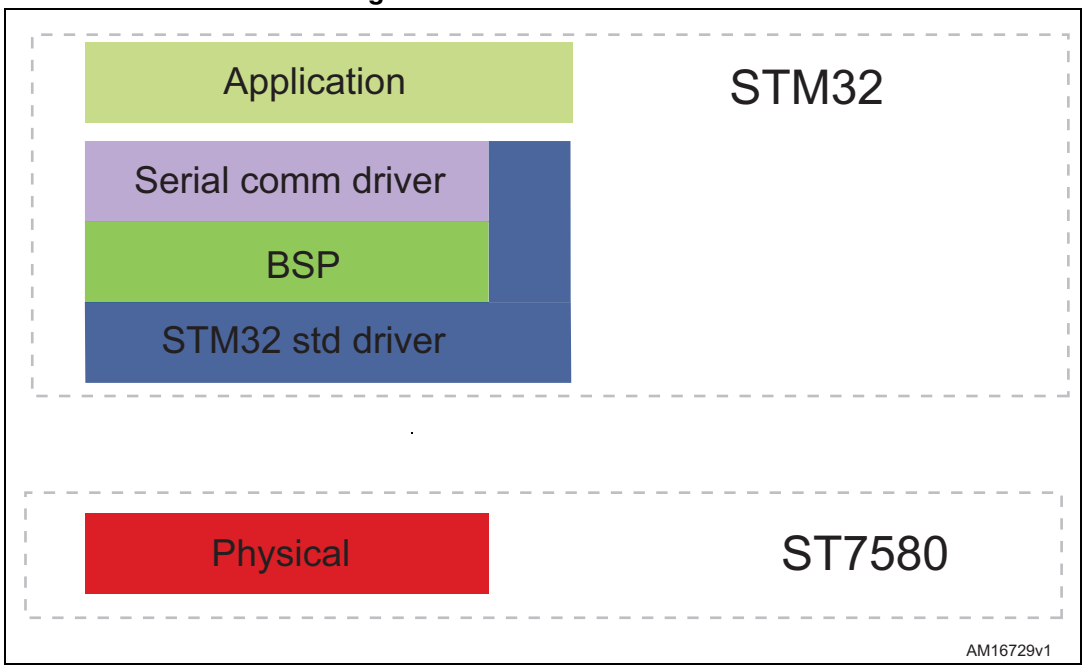
The ST7580 serial communication driver is based on the STM32Fx standard library to manage the peripherals used.

For further details about the serial peripherals and the accepted configuration parameters, please see the ST7580 and the STM32Fx documentation.

2 Firmware architecture

Figure 1 shows the firmware architecture:

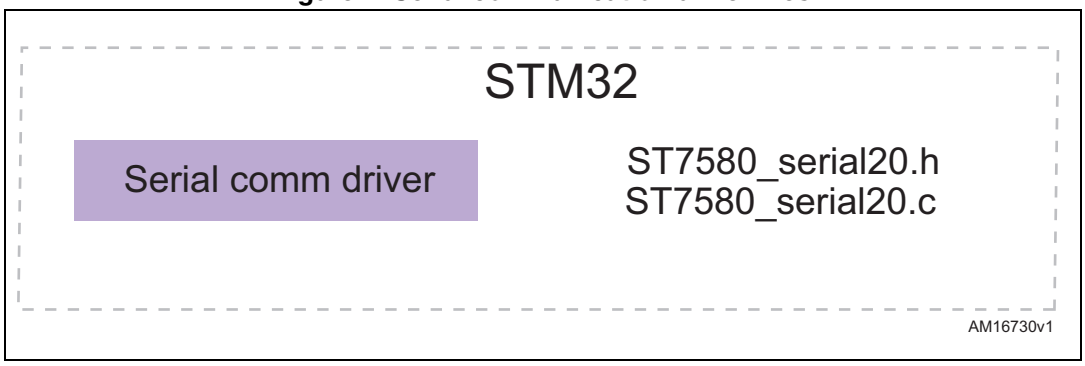
Figure 1. Firmware architecture



- Physical:
  - the physical layer of the ST7580 provides a basic MAC service through a serial communication interface.
- STM32Fx standard driver:
  - it provides the STM32Fx standard peripheral driver.
- BSP (board support package):
  - the BSP layer defines and configures the communication peripherals used to connect the ST7580 and the STM32Fx.
- Serial communication driver:
  - it provides a serial communication function to manage the communication from the ST7580 to STM32Fx and vice versa.
- Application layer:
  - the application layer can be customized by the user; in this section the user must design the application.

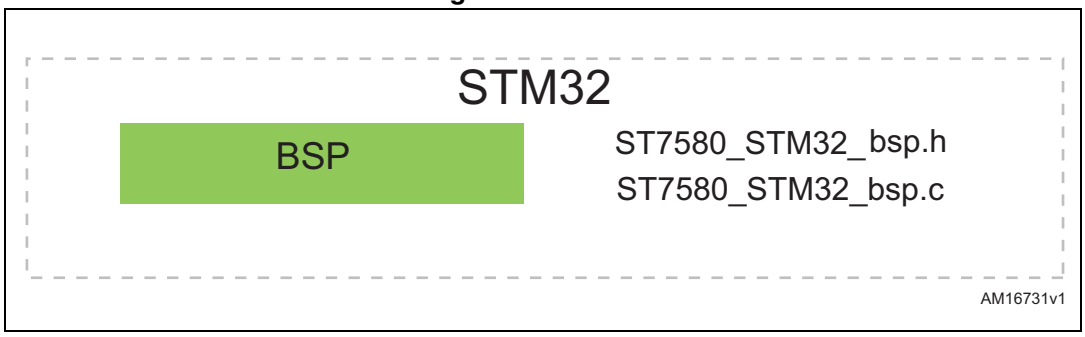
The following *Figure 2* and *Figure 3* show the component files.

Figure 2. Serial communication driver files



The serial communication firmware module contains the files ST7580_Serial20.c/h.They provide FSM (finite state machine) and the functions to manage the serial data flow from and to the ST7580 devices.

Figure 3. BSP files



The BSP firmware module contains all the definitions and the configurations of the hardware resources used by the system to perform and manage the connection and serial communication from the ST7580 to the STM32Fx and vice versa.

3 Board support package

This layer defines the STM32Fx peripherals used to perform the serial communication. In this file the following peripherals and linked functions have to be defined:

- Uart/Usart: serial connection.
- GPIO: used by Uart/Usart, to reset the ST7580 and MCO pin if used.
- Timer: used by serial synchronization and timeout.
- MCO: external clock, if used.

4 Serial communication layer

The serial communication module defines all the functions to manage the serial data flow.

The core of the serial communication driver is a "ModemComFSM" function. This is a finite state machine and must be called in an infinite loop, as follows:

```
while(1)
  {
   state = ModemComFSM( &SerCom, DL_Act);
    switch (state)
    {...}
  }
```

This function has the following input parameters:

- SerCom, input/output parameter. It is a pointer to "SER_CMD_FRAME_T" struct:

```
typedef struct{
      u8 id;//host interface command code
      u8 Rxpar_len;//reception buffer length
      u8 Txpar_len; //transmission buffer length
      u8 Rxparam[255];//reception buffer
      u8 Txparam[255];//transmission buffer
    } SER_CMD_FRAME_T;
```

- DL_Act, this parameter notifies that the data is ready to be sent; the user sets "1" to send the data.

Table 1 defines the return values of the finite state machine:

Table 1. FSM return values

Name	Value	FSM state	Description
M_FRAME_ACKED	04h	1	Frame sent to PLM and ACK arrived.
M_FRAME_NOT_ACKED	05h	1	Frame sent to PLM and NACK arrived (error state).
M_WAIT_ACK	06h	1	FSM waits for either ACK or NACK after sending data.
M_FRAME_RECEIVED	07h	2	FSM received the data from a PLM, and the data is ready to be read.
M_IDLE	08h	2	The FSM is ready to receive the data from STM32Fx to be sent to PLM.
M_FRAME_VALID	09h	2	The data received by STM32Fx is valid.
M_FRAME_NOT_VALID	0Ah	2	The data received by STM32Fx is not valid (error state).
M_FRAME_DELIVERED	0Bh	3	The FSM delivered the data to PLM.
M_WAIT_STATUS	0Ch	3	The FSM waits for the status of the message from PLM.
M_FRAME_CONFIRMED	0Dh	4	The data sent to PLM is confirmed.

Table 1. FSM return values (continued)

Name	Value	FSM state	Description
M_FRAME_NOT_CONFIRMED	0Eh	4	The data sent to PLM is not confirmed (error state).
M_WAIT_CONFIRM	0Fh	4	The FSM waits for the confirmed command from PLM.
M_TO_ELAPSED	10h	1-3-4	Timeout elapsed (error state).

The application layer can send frames to PLM only when the FSM return state is "M_IDLE. If the data sent to FSM is valid, the FSM return state is "M_FRAME_RECEIVED".

5 Reference documents

- ST7580 datasheet
- UM0932 user manual
- AN4068 application note
- STM32Fx datasheet

Appendix A Example of serial communication driver code

The following code shows how to manage the serial communication driver.

The "BSP_Init" is a configuration function, the parameter "COM_BAUD_RATE" is the serial speed (please see the ST7580 datasheet for further details about the accepted speed values).

The FSM must be inserted in an infinite loop, and the return state must be managed.

The state "M_FRAME_RECEIVED" notifies that a new data has arrived, the state "M_IDLE" means that the FSM is free and the data can be sent.

```
/* init BSP */
    BSP_Init(COM_BAUD_RATE);
/* Output HSE clock on MCO pin */
    EnableMCO();
/*start Modem*/
    MODEM_START;
while(1)
  {
  u8 state;
  state = ModemComFSM( &SerCom, DL_Act);
  switch (state)
    {
    case M_FRAME_RECEIVED:// Frame Receive Management
      {
        /* reception management, example */
        Id_Command = SerCom.id;
        memcpy(&RX_Buff, &SerCom.Rxparam, SerCom.Rxpar_len);
        RX_Buffer_Len = SerCom.Rxpar_len;
      }
      break;
    case M_IDLE:// Frame Transmit Management
      {
        /* Transmission management, example */
        if(TX_State == APP_TRUE){
          SerCom.id = 0x50;//DL_Data Req ID
          SerCom.Txparam[0] = 0x04;//Modem Transmission Parameters
          SerCom.Txparam[1] = 0x01;//Data 1
          SerCom.Txparam[2] = 0x03;//Data 2
          SerCom.Txpar_len = 3;
          DL_Act = ACT_TX;
          TX_State = APP_FALSE;
        }
      }
      break;
    case M_FRAME_CONFIRMED:
```

```
      {
       }
      break;
    case M_FRAME_NOT_CONFIRMED:
      {
        /* put here the Frames not Confirmed Management CODE */
        DL_Act = NO_ACT;
      }
      break;
    case M_TO_ELAPSED:
      break;
    case M_FRAME_VALID:
       break;
    default:
      {
       state = M_IDLE;
      }
      break;
   }
}
```

Revision history

Table 2. Document revision history

Date	Revision	Changes
23-Apr-2013	1	Initial release.