Lab 8
Serial Communication

M. Briday

December 7, 2020

The objective of this lab is to make a basic driver for a serial line (UART). The driver will use a transmission buffer in order to avoid any blocking wait. This will allow to use it in under interrupt.

1 Basic raw transmission

1.1 Periodic interrupt

In this section, we first configure the UART to transmit one char, without any buffer. To transmit a char periodically, we first configure a timer@10Hz.

Question 1 *Configure TIM6@10Hz that toggles les D3 (PB0), under interrupt.*

1.2 Connection

The hardware connection is as in Fig.1.

The UART2 is connected on the nucleo to the USB to provide a serial over USB connection. We will use the connection:

- the emitter is the nucleo, on pin PA2: UART2, line TX
- the receiver is the host computer.

There is not enough pins availables on the nucleo board and the UART2 RX is used for other purpose. We only send data in this lab.

The communication client on the host computer can be done either:

- the monitor embedded in the arduino IDE
- `minicom` on Linux or Mac. This is a small, easy to use tool. With the virtual machine, you can install it with `sudo apt install minicom` and run it in the terminal with `minicom -s`.

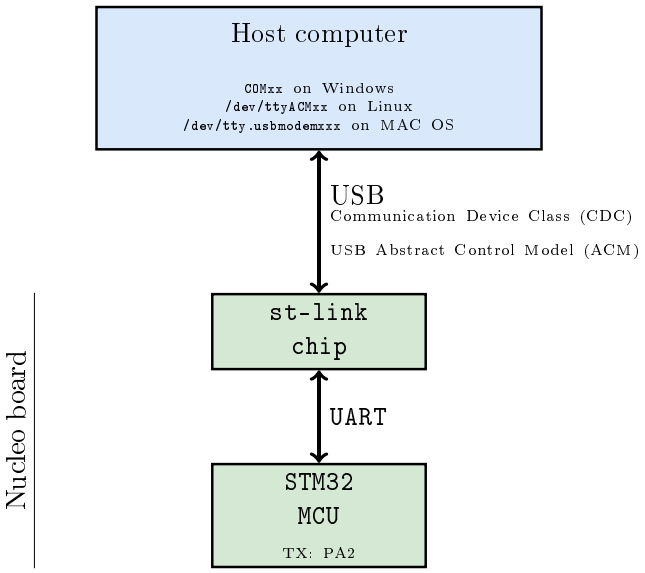

Figure 1: UART connection between STM32 and host computer

1.3 Driver setup

The setup of the serial implies the definition all the configuration registers of the peripheral. In this lab, we will just use a specific configuration: `115200 8N1` (115200 bits/s, 8 bits of data, no parity, 1 stop bit). We use `oversampling=8`.

At this first step: no interrupt for the UART.

For the configuration, you have to look at the datasheet (reference manual). The init should:

- configure the pin (alternative mode)
- set a clock and reset the peripheral (as for timers)
- configure the speed with `BRR register`
- configure `CR1`. `CR2` and `CR3` remain at 0.

Question 2 *Write the function* `void serialSetup();` *that inits the driver.*

To validate the initialization function, we first write a very basic print function that just write a character on the transmission register, without any control.

Question 3

- *write the function* `void serialPrintChar(char c);` *that sends an ASCII char over the serial line without any control on* `TDR`.
- *Test with the transmission of a character each 100ms inside the interrupt handler. The character should not be the same each time (for instance a loop from* `'A'` *to* `'Z'`*).*
- *use the logic analyzer to capture the transmission (use the protocol decoder to get a readable information). How much time is required?*

Question 4

- *what is the behavior if we try to send 2 bytes in the interrupt? Why?*
- *update the* `void serialPrintChar(char c);` *to check that the TX register is free before a transmission (still no UART interrupt).*
- *test the implementation with a transmission of 2 characters in the interrupt. Validate with the logic analyzer.*

Question 5 *Add the function so that we can have a basic driver in polling mode:*

```
/** send a NULL terminate ASCII string on the serial line. */
void serialPrintString(char *s);
```

Question 6 *Add the functions to print an integer (32 bits). Of course, non-significant zeros should not be displayed!*[1]

```
/** send a 32 bits integer on the serial line.  */
void serialPrintInt(uint32_t i);
```

2 FIFO implementation

A fifo (First In First Out) can be implemented with a circular buffer and 2 read and write indexes. This is called a circular buffer because when an index reaches the maximum value, it restarts from 0.

We first define the data structure:

```
#define FIFO_SIZE 128

typedef struct {
    volatile char tab[FIFO_SIZE];
    volatile int readIndex;  //point to the next value to read
```

[1] It's better to handle signed value, but handling an unsigned is ok.

```
    volatile int writeIndex; //point to the next free place
} fifo;
```

The principle is as follow:

- at startup, both indexes are reset;
- to insert a new value, we:
  - check that the fifo is not full;
  - at a new value in the tabular;
  - update the write index to point to the next place.
- to read a new value, we:
  - check that the fifo is not empty;
  - read the value in the tabular;
  - update the read index to point to the next place.

The fifo will be used for data transmission with the uart, so we make the choice to drop out characters that are written when the fifo is full.

The functions prototypes are:

```
/** init the fifo structure */
void fifoInit(fifo *f);

/** write a char in the fifo.
 * @return 0 if the fifo was full and the data was droped out
 * @return 1 if the value is written in the fifo
 */
int fifoWrite(fifo *f, char c);

/** read a char in the fifo.
 * @return 0 if the fifo was empty. c is not updated.
 * @return 1 if the value is read. c is the read character.
 */
int fifoRead(fifo *f, char *c);
```

Question 7 *Implement these 3 functions. Test your implementation and explain why your tests are sufficient to validate your implementation (normal behavior, full fifo, empty fifo...)*

3 Adding the FIFO to the driver

UART with a fifo works as follows:

- the `serialPrintChar()` function does not write directly to the TX buffer, but writes into the fifo.
- when a char has been transmitted (*i.e.* removed from the TX data register), an interrupt is generated and the next value from the fifo is taken.

We have to declare a global fifo, as it will be shared by different interrupts

```
fifo txFifo;
```

Be careful: there is a global structure that is shared between interrupts!! You have to make fifo accesses atomic!!

Attention will have to be paid to the particular case where there is no more data to be transmitted. In this case, the interrupt must be deactivated (the TX data register is always empty). Obviously, the interrupt should be reactivated for the next transmission.

Question 8 *Implement the driver with the fifo.*

Use 2 GPIOs (PB0 (D3) and PB3 (D13)) that are set at the beginning and reset at the end of of TIM6 and UART interrupt handler. So that, we are able to trace with the logic analyser the code execution timings.

Question 9 *Validate your solution with the transmission of a character string with the logic analyzer.*

For a defined character string, measure the time in the different interrupt handlers and the transmission time.