LSCE-P1

Alejandro Gómez Molina
Luis Felipe Velez Flores

October 2020

1 Introduction

This document explains the RS323 module implementation using VHDL. In the first place, we will explain TX and RX sub-modules design as well as the implementation.Extended Finite State machines are used to design the modules behaviour. Finally, we will validate the RS232 top module with a testbech.

RS232 is a telecommunications standard designed for serial transmission data. This implementation uses two wires for transmission (TX) and reception (RX) without flow control. This provides a full duplex communication. In idle state, the line is pulled to a logic high, the data interchange starts when the line is pulled down. As we can see in Figure 1, the RS232 data frame is composed by a start bit, flowed by data (1 byte) and a stop bit at the end.

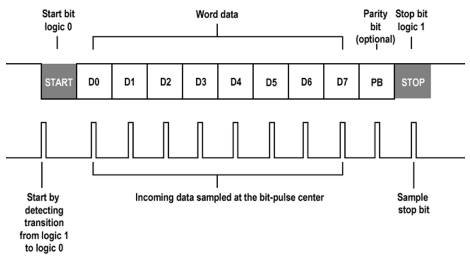


Figure 1: RS232 data frame

The transmission rate is also called baudrate and typically is in the range of 1200 to 115200 bauds. In this case, we will use a baudrate of 115200 bauds. This rate defines the clock prescaler used by the modules.

In order to avoid glitches and perform a more robust FSM we have registered the outputs and the counters. We have also created a package that contains the log_2 function (to get the exact numbers bits needed) and the following constants:

- **PULSE_WIDTH:** = 174
- **HALFCOUNT:** $\frac{PULSE\_WIDTH}{2}$
- **WORD_LENGTH :** 8

Respected to the counters, we compare its value with a threshold. If the value is equal, we set to zero. Otherwise, we add 1. The reason is that the mod function uses more resources and it is not necessary.
Both FSM use three process in their implementation:

- Next state: where the next state of the FSM is changed according the inputs.
- Output value: where counters and output values change.
- Register: the next state signal is asigned to the registered signal.

2 RS232 RX module

This module is in charged of transmiting data from memory to the RX line whit a specific baudrate. As shown in the Figure 2, this behavior has been modeled using a FSM with 4 states:

- **IDLE:** During this state, the module will be waiting for the start bit. Once this occurs, all the counters are reset (clock prescaler and bit counter) and go to the START_BIT state.
- **START_BIT:** In this state, the module will wait until the clock counter reach the PULSE_WIDTH value which is the number of clock pulses on one bit. Once this value is reached, the module go to RVCDATA state.
- **RVCDATA:** During this state, the system will sample the RX line every $\frac{PULSE\_WIDTH}{2}$ clock pulses. When one bit is sampled, the valid_out line is pulled to high and the bit counter is incremented. When the bit counter reach the word size the module go to STOP_BIT state.
- **STOP_BIT:** In this state, the module will wait until the clock counter reach the PULSE_WIDTH value which is the number of clock pulses on one bit. Once this value is achieved, the store_out signal is pulled to high and the module go to IDLE state.

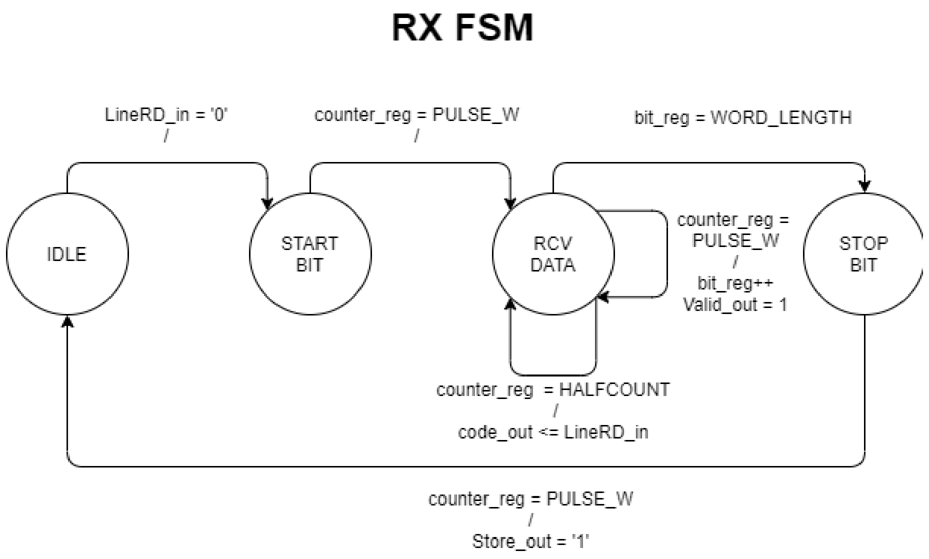

RX FSM
LineRD_in = '0' /
counter_reg = PULSE_W /
bit_reg = WORD_LENGTH
IDLE
START BIT
RCV DATA
STOP BIT
counter_reg = PULSE_W / bit_reg++ Valid_out = 1
counter_reg = HALFCOUNT / code_out <= LineRD_in
counter_reg = PULSE_W / Store_out = '1'

Figure 2: RX Finite state machine diagram.

3 RS232 TX module

We have used the proposed scheme to perform the Tx module. According to that, Figure 3 shows its behaviour.

- **IDLE:** this is the waiting state, where EOT and TX are set to 1. To start the sequence of sending new data, start must be set to '1', and in consequence we pass to START BIT state.
- **START_BIT:** Now TX is set to zero indicating that new data is coming. The duration of this value is PULSE_WIDTH pulses in order to get 115200 bps of baudrate.
- **SEND_DATA:** in this state the information is sended. We send one bit during PULSE_WIDTH each time. When the last bit is sended, we pass to STOP_BIT state.
- **STOP_BIT:** in this state TX is set to 1 during PULSE_WIDTH, and after that EOT is set to 1 the same time. This indicates the end of the transmission and in consequence we back to IDLE state.

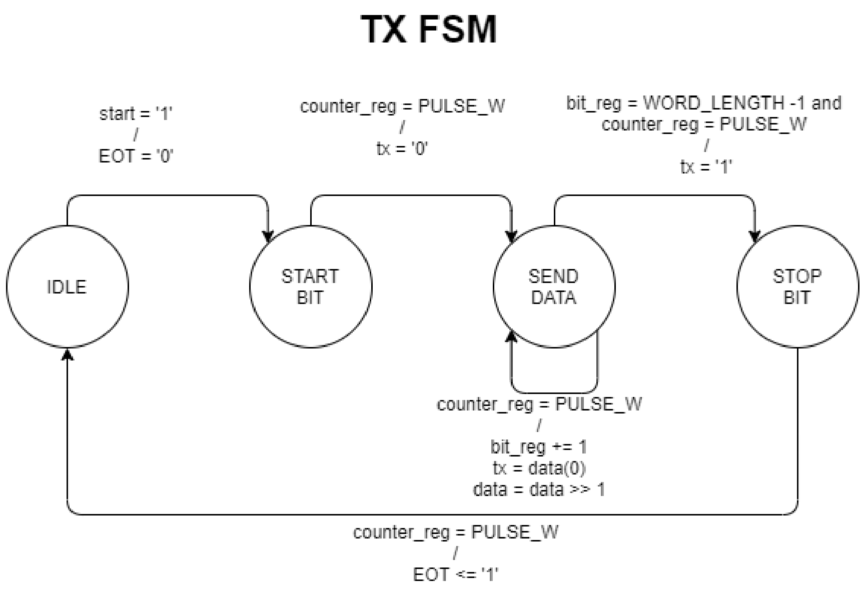
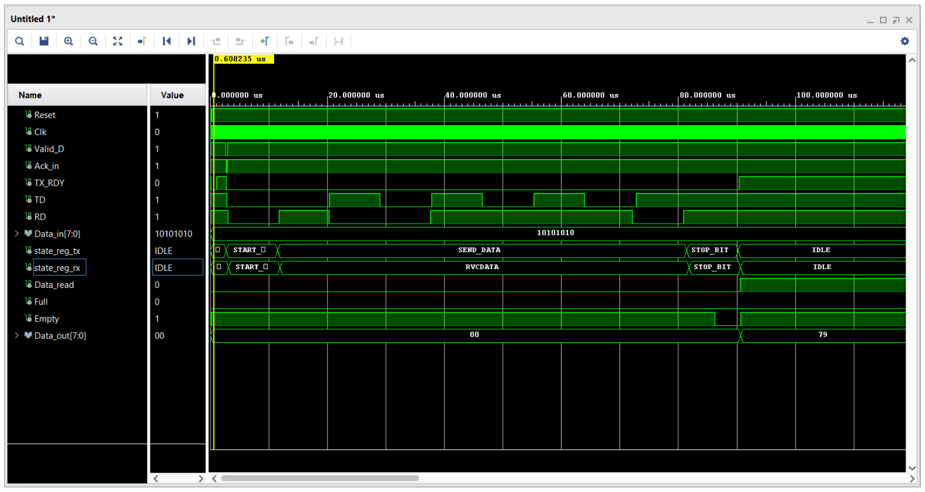

TX FSM
start = '1' / EOT = '0'
counter_reg = PULSE_W / tx = '0'
bit_reg = WORD_LENGTH -1 and counter_reg = PULSE_W / tx = '1'
IDLE
START BIT
SEND DATA
STOP BIT
counter_reg = PULSE_W / bit_reg += 1 tx = data(0) data = data >> 1
counter_reg = PULSE_W / EOT <= '1'

Figure 3: TX Finite state machine diagram.

4 Full module validation

In order to validate the system, we used the testbench provided. As we can see in the Figure 4, the module works properly according to the modeled behavior.



Name	Value
Reset	1
Clk	0
Valid_D	1
Ack_in	1
TX_RDY	0
TD	1
RD	1
Data_in[7:0]	10101010
state_reg_tx	IDLE
state_reg_rx	IDLE
Data_read	0
Full	0
Empty	1
Data_out[7:0]	00

Figure 4: Test results