Laboratory 12

12. Finite State Machines and Serial Communication (2)

12.1. Objectives

Study, design, implement and test
- **Finite State Machines**
- **Serial Communication**

Familiarize the students with
- Xilinx® ISE WebPack
- Digilent Development Boards **(DDB)**
  - ➢ Digilent Basys Board – Reference Manual
  - ➢ Digilent Basys 2 Board – Reference Manual
  - ➢ Digilent Basys 3 Board – Reference Manual

12.2. Theoretical Background

Oversampling mechanism for UART Receive

When transmitting a byte, the UART first sends a START BIT followed by the data (general 8 bits, but could be 5, 6, 7, or 8 bits), followed by STOP BITs. The sequence is repeated for each byte sent.

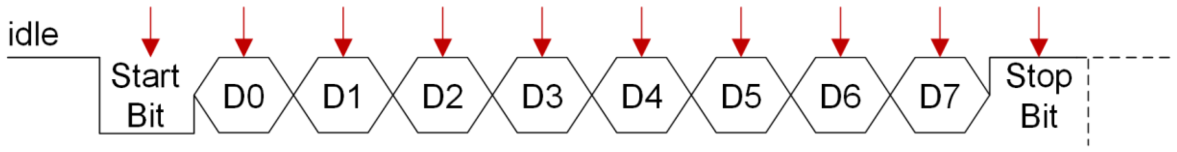

Figure 12-1: Timing Diagram for serial transmission (8-bit Data Example). The red arrows indicate when the bits of data should be read at the receiver.

Serial transmission does not involve a clock signal. The information is included in the baud rate (**number of bits per second**). Common baud rates are 2400, 4800, 9600 and 19200. This means that a bit transmitted through the serial line is valid for a given time period (the inverse of the baud rate). More details on the transmission over the serial line can be found in the previous laboratory.

When receiving a UART packet, one must read (sample) the input signal and extract the data bits sent over the serial line bit by bit. At a first glance, the sample rate for the receiver should coincide with the sample rate (baud rate) of the transmitter; i.e. the rate at which the data was sent. However, this is WRONG and can yield in bad transfers at the receiver end, due to imperfect synchronizations (the receiver and the transmitter are in two different clock domains, the baud rate is generated independent at the receiver and the transmitter, asynchronous communication – no common clock signal) between the receiver and the transmitter (double reading the same bit, missing the start bit, not reading the first bit and reading the sign bit, etc.). The frequency at which such events can occur depends on the difference between the sampling rates of the transmitter and receiver. Even if the differences would be very small at a significant number of samplings for successive bits, the error in communication can occur. For example, a difference of 0.1% between the two sampling rates, when transmitting 1000 bits the error appears once. When we perform the serial transfer with 10-bits per character (1 start bit, 8 data bits and 1 stop bit) it results that one character from 100 will be falsely received.

This problem is tackled using oversampling: the input receive signal is read (sampled) at a higher rate than the one used at the transmitter. This permits the detection of the middle of the start bit interval, thus allowing the data bits to be read approximately in the middle of the bit interval, thus eliminating the risk of gaps and receiving false data. For each new character, the middle of the start bit will be determined so this is the only synchronization mechanism used between the receiver and the transmitter.

Oversampling rates are multiple of the transmitter baud-rate: 2, 4, 8, etc. The most usual oversampling rate is 16 times the baud rate of the sender. Each bit that is received over the serial line is sampled (read) 16 times, but only one of the samples is saved (the middle one). The maximum delay for detecting the start bit is 1/16 from the bit interval.

Any UART circuit contains a shift register that is used for converting the received serial data into its parallel form.

12.3. Laboratory Assignments

12.3.1. Serial Receive FSM

Design a baud rate generator that would ensure a 9600 baud rate (9600 bits per second) communication over the serial cable. Use a counter to generate the BAUD_ENable signal (generate a '1' every bit time interval). For the serial receive communication you need to implement an oversampling mechanism of 16.

Baud rate generation for oversampling of 16:

- For 25 MHz, clock period ~ 40 ns, input clock must be divided by ~ 163.

- For 50 MHz, clock period ~ 20 ns, input clock must be divided by ~ 326.
- For 100 MHz, clock period ~ 10 ns, input clock must be divided by ~ 651.

Define a new entity for the receive FSM. The next figure presents the ports of this entity.

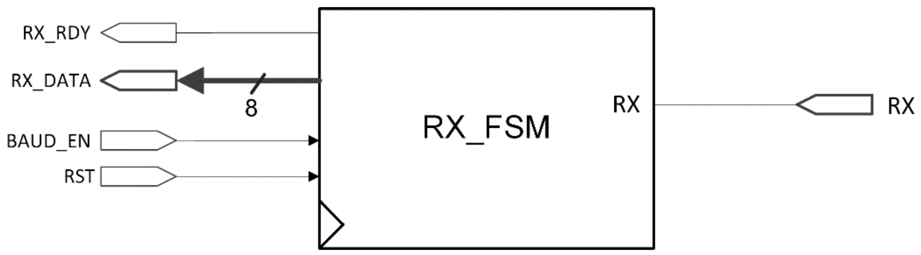

Figure 12-2: RX_FSM Entity Description

The detailed FSM implementation is presented in the figure below. A state transition is triggered only in the clock cycle when BAUD_ENable is ‘1’. This ensures that a bit will be valid for the baud rate period.

For the RX_FSM you have to use two auxiliary counters: BAUD_CNT and BIT_CNT.

The BIT_CNT is similar to the one in the TX_FSM, i.e. a signal with the functionality of a counter inside the RX_FSM; it holds the current transmitting bit number. It should be incremented in the bit state and should be reset after each serial transfer (you can do that in the idle state, or in all states except the bit state).

The BAUD_CNT is a signal is a signal with the functionality of a counter inside the RX_FSM; it counts the number of BAUD_ENables in order to ensure a correct oversampling mechanism. Remember that you use an oversampling factor of 16.

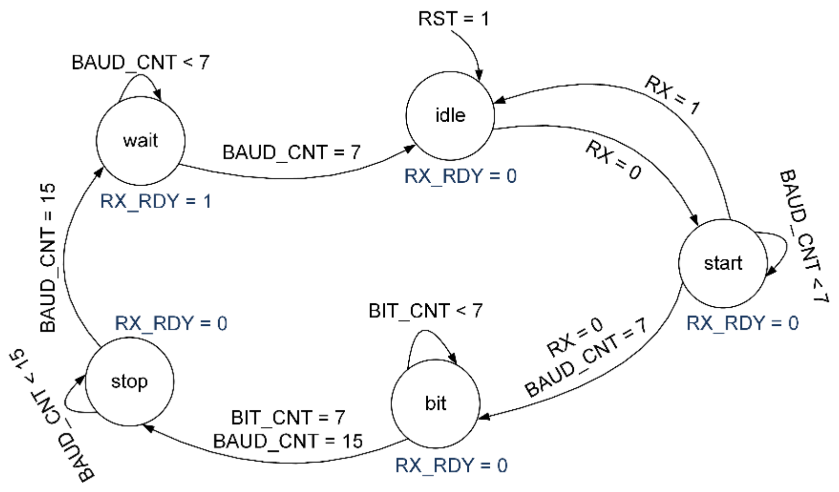

Figure 12-3: RX_FSM Implementation

Write the VHDL code and implement in the “test_env” project the RX_FSM state machine. Use a FSM with 2 or 3 processes (see appendix 7, laboratory 11). You also have to implement a shift register in order to receive the correct data from the serial input line. The RX signal will be shifted in this shift register only once per bit interval; i.e. in the middle of the transmitting interval. Test the communication between the FPGA board and the PC. The parameters of the serial communication are: 1 start bit, 8 data bits, 1 stop bit, no parity bit, 9600 baud rate. Make sure that these settings are also configured in the HTERM / hyper-terminal application. You have to identify the serial port where the module is connected – exactly like in the previous lab.

In order to test the serial transmission from the computer to the FPGA board, connect the RX_DATA output to the SSD (2 digits), RST to ‘0’ or a MPG enable signal. On the SSD, you will see the 8-bit ASCII code representation of the characters that you are sending from the PC.

12.3.2. I/O from the MIPS CPU – optional

Connect the RX_FSM into your own MIPS processor implementation. At this point, you are allowed to use your finished and complete processor (single-cycle or pipeline).

You have to receive 16-bits of valid data from the computer and feed this data into your MIPS processor. Depending on your program you can define what fields will be written with the data coming from the computer (register from the Register File, Data Memory location or even the Instructions from the Instruction Memory).

Remember that when receiving data from the serial RX line the 8-bits from a data transfer represent an ASCII character, hence you are required to make 4 transfers in order to receive the alphanumerical encoding of the 4 x 4-bit hexadecimal value (use a decoder/ROM to generate the 4-bit hexadecimal data and then concatenate 4 receive transfers in order to obtain the correct 16-bit data that will be fed to your processor).

Define the methodology to receive the 16-bit data over the serial RX line. Use the RX_RDY signal to control the writing of the data into your processor.

12.4. References

[1] XST User Guide
[2] Digilent Basys Board – Reference Manual
[3] Digilent Basys 2 Board – Reference Manual
[4] Digilent Basys 3 Board – Reference Manual
[5] Digilent Pmod USB-UART – Reference Manual
[6] http://www.asciitable.com/
[7] http://www.der-hammer.info/terminal/