Laboratory 11

11. Finite State Machines and Serial Communication

11.1. 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

11.2. Theoretical Background

11.2.1. Finite State Machines

A finite state machine or FSM is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is used to describe an abstract model of a control unit. XST proposes a large set of templates to describe FSMs. By default, XST tries to distinguish FSMs from VHDL or Verilog code, and apply several state encoding techniques to obtain better performance or less area.

XST supports the following state encoding techniques:

- Auto – the best suited encoding algorithm for each FSM.
- One-hot – associate one code bit and one flip-flop per state. At a given clock cycle during operation, one and only one bit of the state variable is asserted. Only two bits toggle during a transition between two states. One-hot encoding is appropriate with most FPGA targets where a large number of flip-flops are available. It is also a good alternative when trying to optimize speed or to reduce power dissipation.
- Gray – guarantees that only one bit switches between two consecutive states. It is appropriate for controllers exhibiting long paths without branching. In

addition, this coding technique minimizes hazards and glitches. Very good results can be obtained when implementing the state register with T flip-flops.

- Compact – consists of minimizing the number of bits in the state variables and flip-flops. Compact encoding is appropriate when trying to optimize area.
- Johnson – like Gray, it shows benefits with state machines containing long paths with no branching.
- Sequential – consists of identifying long paths and applying successive radix two codes to the states on these paths. Next state equations are minimized.
- Speed1 – oriented for speed optimization. The number of bits for a state register depends on the particular FSM, but generally, it is greater than the number of FSM states.
- User – original encoding specified in the HDL file.

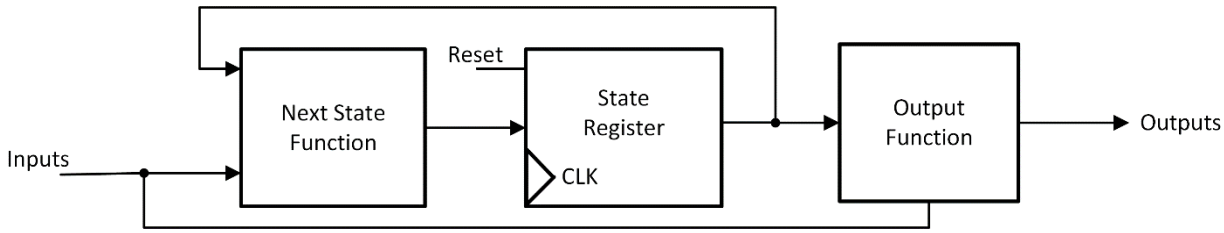

Figure 11-1: FSM Representation Incorporating Mealy and Moore Machines

When describing a finite state machine in VHDL you may have several processes (1, 2 or 3) depending upon how you consider and decompose the different parts of the preceding model. Appendix 7 (adapted from [1]) describes the VHDL finite state machine implementations.

11.2.2. Serial Communication – UART

Serial communication is the transmission or reception of data one bit at a time. Today's computers generally address data in bytes or some multiple thereof. A serial port is used to convert each byte to a stream of ones and zeroes as well as to convert streams of ones and zeroes to bytes. The serial port contains an electronic chip called a Universal Asynchronous Receiver/Transmitter (UART) that actually does the conversion. Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires.

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

Figure 11-2: Serial Transmission Timing Diagram

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).

The start bit is always 0, the data bits are transmitted with the LSB (least significant bit) first and MSB (most significant bit) last and the stop bit is always 1. In serial communication, the stop bit duration can have multiple values: 1, 1.5 or 2 bit periods in length. Besides the synchronization provided by the use of start and stop bits, an additional bit called a parity bit may optionally be transmitted along with the data. A parity bit affords a small amount of error checking, to help detect data corruption that might occur during transmission. One can choose even parity, odd parity, mark parity, space parity or none at all. When even or odd parity is being used, the number of marks (logical 1 bits) in each data byte is counted, and a single bit is transmitted following the data bits, to indicate whether the number of 1-bits just sent is even or odd.

The data sent through serial communication is encoded using ASCII codes (Appendix 8). Assume we want to send the letter 'A' over the serial communication channel. The binary representation of the letter 'A' is 01000001 ($0x41_{hex}$). Remembering that bits are transmitted from least significant bit (LSB) to most significant bit (MSB), the bit stream transmitted would be as follows for the line characteristics 8 bits, no parity, 1 stop bit, 9600 baud: **LSB (0 1 0 0 0 0 0 1 0 1) MSB**. This represents (Start Bit) (Data Bits) (Stop Bit). For a binary two-level signal, a data rate of one bit per second is equivalent to one Baud. To calculate the actual byte transfer rate, simply divide the baud rate by the number of bits that must be transferred for each byte of data. In the case of the above example, each character requires 10 bits to be transmitted for each character. As such, at 9600 baud, up to 960 bytes can be transferred in one second.

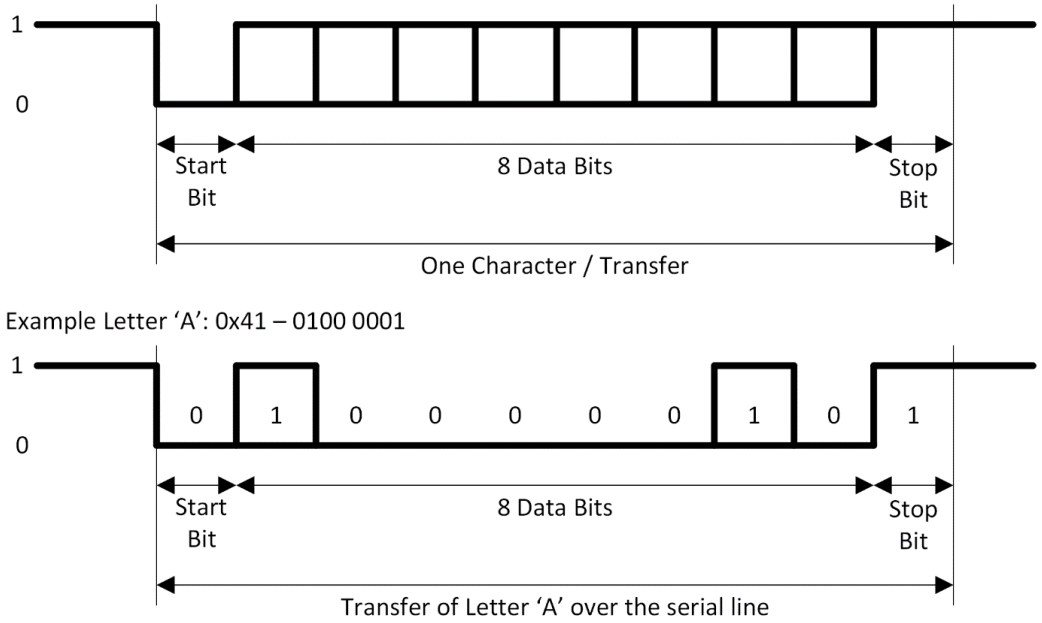

Figure 11-3: Serial Transmission Example (8 data bits, no parity)

For accurate serial communication, on the receiving end, an oversampling scheme is commonly used to locate the middle position of the transmitted bits, i.e., where the actual sample is taken. The most common oversampling rate is 16 times the baud rate. Therefore, each serial bit is sampled 16 times but only one sample is saved.

Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms.

11.3. Laboratory Assignments

11.3.1. USB-UART

For Basys1 & 2 Development boards: Read the Pmod USB-UART reference manual. The figure below shows the connection of the USB-UART peripheral module to the FPGA board.

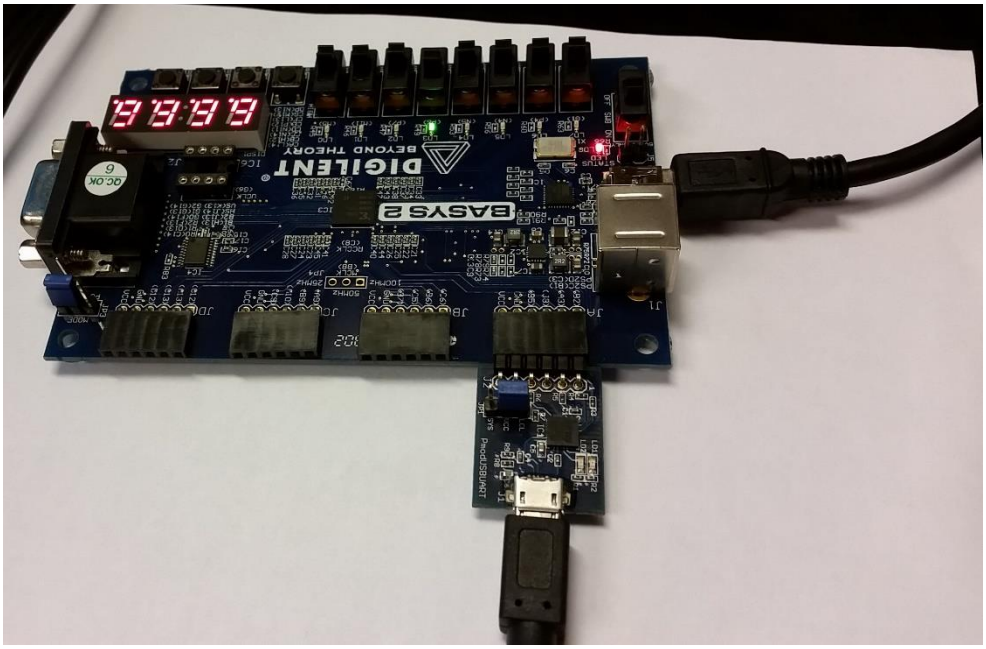

Figure 11-4: Pmod USB-UART connection to the FPGA board

Use the USB-Mini USB cable to power the board and the USB-Micro USB cable for serial data communication.

For Basys3 Development board: Your board is already equipped with an **USB UART Bridge** (FTDI chip) for serial communication. Refer to the Basys 3 reference manual for more details.

Download and open the HTERM terminal program. Alternatively, you can use the hyper-terminal software available in windows or download any other terminal software known to you / available on the web.

You need to define the RX (input) and TX (output) ports into your "test_env" project and in the UCF/XDC file. Use your board's reference manual to locate the correct pin numbers. Attention: The TX of the FPGA board is the RX of the Pmod USB-UART module / FTDI chip and the RX of the FPGA board is the TX of the Pmod USB-UART module.

11.3.2. Serial Transmit 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).

Baud rate generation:

- For 25 MHz, clock period ~40 ns, input clock must be divided by 2604.
- For 50 MHz, clock period ~20 ns, input clock must be divided by 5208.

- For 100 MHz, clock period ~10 ns, input clock must be divided by 10416.

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

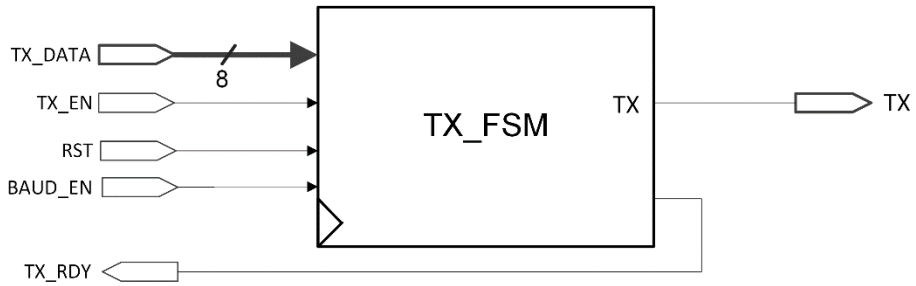

Figure 11-5: TX_FSM Entity Description

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. The BIT_CNT is a signal with the functionality of a counter inside the FSM; it holds the current transmitting bit value. 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).

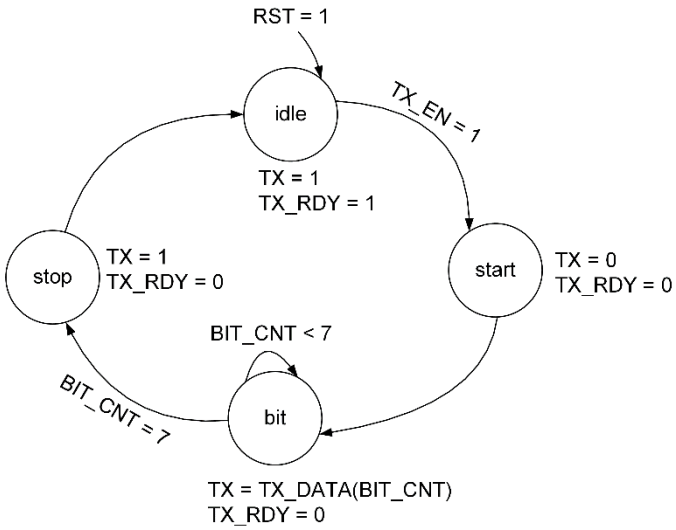

Figure 11-6: TX_FSM Implementation

Write the VHDL code and implement in the "test_env" project the TX_FSM state machine. Use a FSM with 2 or 3 processes (see appendix 7). 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.

In order to test the serial transmission from the FPGA board to the PC, connect the TX_DATA input to the switches, the TX_EN signal to a MPG enable, RST to '0' or another MPG enable. Make sure that the switches show a valid ASCII code.

Define the correct methodology of asserting the TX_EN signal in order to initiate a single serial data transfer (use a D flip-flop with a set and a reset).

11.3.3. I/O from the MIPS CPU

Connect the TX_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 send 16-bits of data from your MIPS processor to the PC. Depending on the result of your program, define what field you will send (register with the final result, memory location, etc.).

Example:
When your program has finished execution the result is in R7 and the PC is 0x0020. Add a new instruction to your program: addi R7, R7, 0. Define a 16-bit register whose value will be written from the RD1/ALURes signal, write it in this register (write enable with the value of the PC) and initiate the serial transfer.

Remember that when sending over the serial line the 8-bits represent an ASCII character, hence you are required to make 4 transfers in order to send the alphanumerical encoding of the 4 x 4-bit hexadecimal value (use a decoder/ROM to generate the 8-bit ASCII representation for a hexadecimal value).

Define the methodology to send the 16-bit data over the serial line. Use the TX_RDY signal to control the 4 serial transfers.

11.4. 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/