APPLIED HOMEWORK #4

Sequential Circuits

1 Description and Homework Objectives

Have you watched the video entitled "Proper Coding of State Machines" under the heading "Applied Homework Resources"? If not watch that first. Then return to this document.

This applied homework is primarily about creating the state machine to control a Manchester serial receiver. To make a digital circuit to control "something"…you first have to understand how that "something" is supposed to behave. So before we can jump into designing the state machine you have to read some background on how Manchester serial transmission/reception works.

After this homework, you should be able to:
1. Understand serial transmission/reception
2. Properly code a state machine in verilog
3. Use an FPGA prototyping system to verify sequential circuit funtionality

2 Applied Homework Tasks

2.1 Understand the Protocol

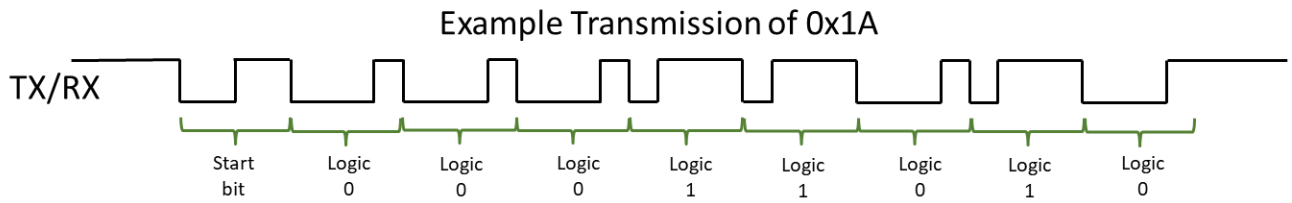
As mentioned there are many serial protocols. They all involve sending data one bit at a time over a medium, and often a quite a distance. The two systems communicating over a serial link are typically far enough apart that they don't share the same system clock. So how does the receiver of the bits know when in time the bits are valid?

Some protocols use only a single line, but have an agreed upon bit rate (UART, Modbus, …)

Some protocols use a data line and a clock line (SPI, I2C, …)

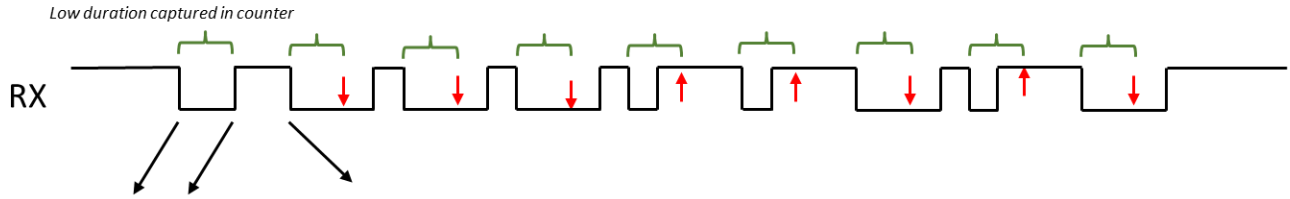
Some protocols embed clock and data information in the same line (USB, Manchester, …)

Manchester encoding is a style that embeds clock (timing) and data information into a single line. There are actually several versions of Manchester encoding. We are using one where the line is normally high when idle. Meaning when nothing is being transmitted the TX/RX line is high. The protocol is transmitting 8-bit words with the MSB first. However, prior to the MSB there is a "start bit" that encodes timing information as to the period of a bit time. Each bit starts with a falling edge. The protocol encodes a logic 0 as 25% duty, and a logic 1 as 75% duty. See the following diagram for clarification.



The protocol transmits a byte (8-bits) at a time. The line is normally high when idle. All bytes start with a "start bit". Timing information is built into the start bit. The signal has been high, then a start bit occurs. It is low for 50% of the duration of a bit period. The receiver times the duration of the low period of the start bit, and captures that value in a period register. Remember in AHW3 you made a registers **per_cnt** & **per_capture**.

For the next 8 subsequent falling edges of the ***RX*** signal, the receiver will start a timer (re-use of **per_cnt**). When that timer matches the captured value of the start bit low period it will sample (and shift into a shift register) the value of the ***RX*** line. When finished the shift register will contain the 8-bit station ID. The MSB of the byte is sent first.

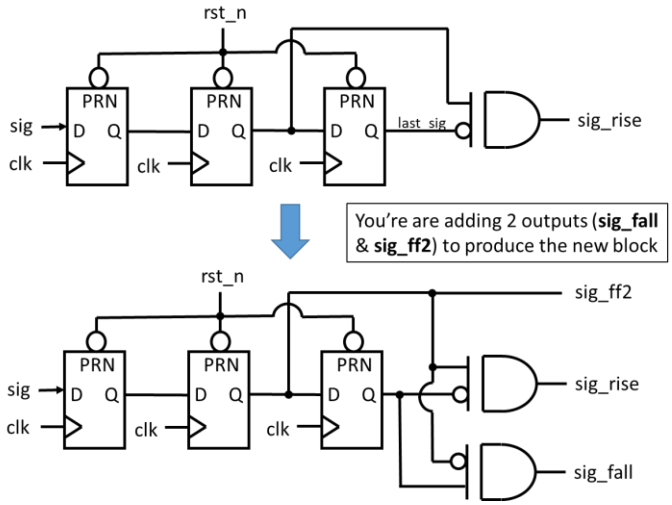


Receiver sees falling edge of start bit and starts a counter timing the low duration (**per_cnt**). When the low duration ends it captures this value (**per_capture**).	For the next 8 falling edges the receiver will "see" the falling edge and start a counter. When the count value matches the captured low duration of the start bit it will shift the shift register, thus sampling the ***RX*** line value. ***RX*** will be feeding into the LSB of the **shft_reg** we made in AHW3.	The red arrows indicate the times at which the ***RX*** line is (sampled) i.e. the shift register is shifted.

2.2 Copy Your rise_edge_detect.sv & Modify

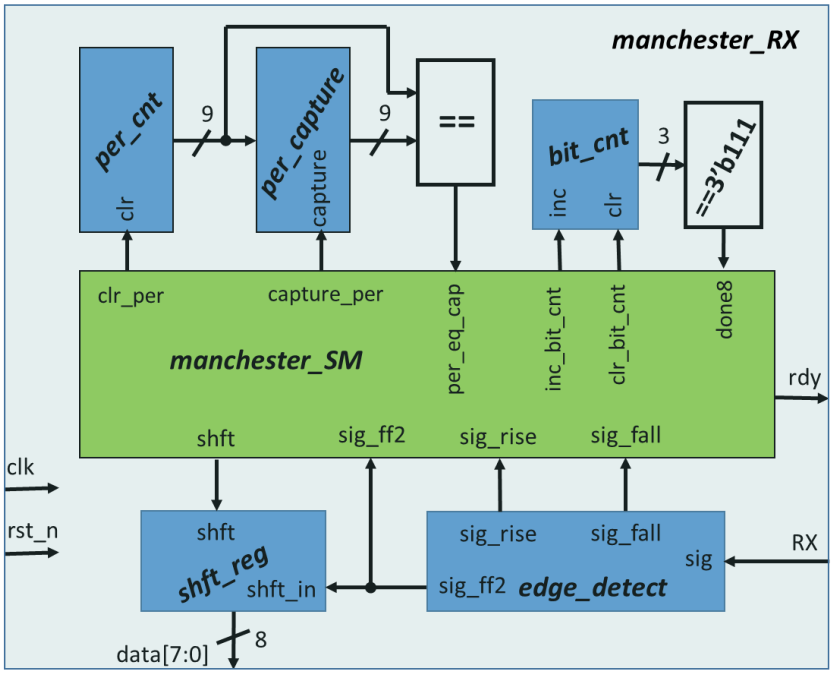
In AHW3 you made a block called **rise_edge_detect.sv** that double flopped (synchronized to your clock domain) an incoming signal and performed rising edge detection on it. If you look at the above protocol you can see that we are interested in both rising and falling edges of the signal. Also…since serial communication typically occurs between two systems that don't share a common clock it is wise for the receiver to synchronize the signal to avoid meta-stability issues.

Copy your **rise_edge_detect.sv** from AHW3 and call it **edge_detect.sv**. Now modify it as specified below.



2.3 Understand Manchester Receiver Block Diagram

To make the state machine you need to understand how the whole receiver will work. Study the following block diagram and think about what needs to occur to receive a serial byte in this protocol.



Interface of the state machine:

Signal:	Dir:	Notes:
clr_per	out	Clear at every falling edge
capture_per	out	Capture at start bit rise edge
clr_bit_cnt	out	Clear at start
inc_bit_cnt	out	Inc when know done with bit
shft	out	Shift when per_eq_cap
rdy	Out	Pulses high for 1 clk cycle when data is ready
per_eq_cap	in	When per_cnt eq cap we sample (i.e. shift)
done8	in	When we have shifted 8 times we are done
sig_rise	in	
sig_fall	in	
sig_ff2	in	SM needs to look for a high on sig_ff2 in some instances

2.4 Finish the State Diagram

At the end of this file you will find a state (bubble) diagram. A few of the transitions have been labeled. Print it out and complete it by hand. Take a clear picture of your completed state diagram (**state_diagram.jpg**) with your phone and submit to the dropbox for AHW4.

2.5 Copy all Your Collateral from AHW3

In AHW3 you made a bunch of different register cells. **per_cnt**, **per_capture**, **bit_cnt**, **state5_reg**, **shft_reg, & rise_edge_detect**. Copy all that over to your AHW4 working directory. You will also need files from your AHW2 like **hex7seg**.

2.6 Implement & Test the State Machine

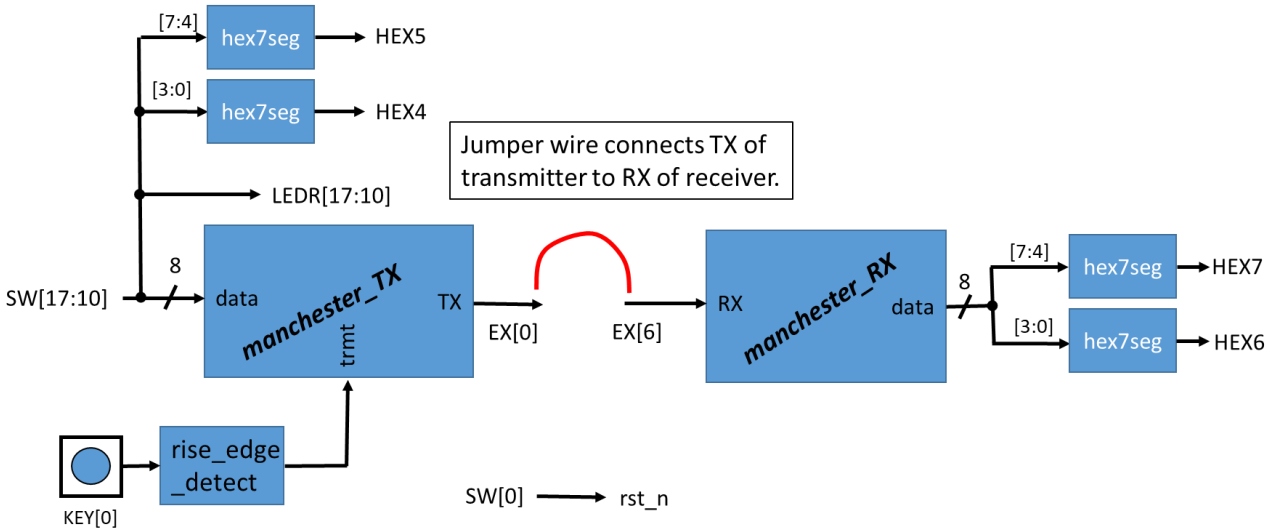
A file called **manchester_RX.sv** is provided. It instantiates all the blocks shown in the block diagram, and also the state flops needed (state5_reg). The main intellectual property of any state machine is the ***always_comb*** (with ***case*** statement) block that determines the state transitions and state machine outputs. Complete this ***case*** statement using your state diagram as a guide.

As always a self checking test bench (**manchester_RX_tb.sv**) has been provided. Using ModelSim compile, simulate and debug. Grab a screen capture of the waveforms for all signals of the DUT for the length of the simulation (**manchester_RX_sim.jpg**). Also capture the "YAHOO test passed" from the transctipt window (**manchester_RX_yahoo.jpg**). Submit **manchester_RX.sv**, **manchester_RX_sim.jpg** and **manchester_RX_yahoo.jpg** to the dropbox for AHW4.

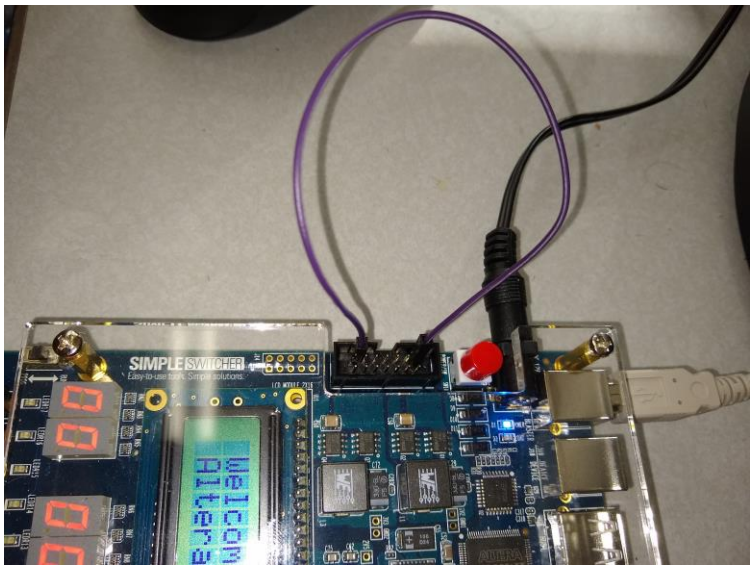
3 Demo Tasks

DO NOT wait until your demo to read this!

During your Applied Homework Demo, you will compile AHW4_top.sv in Quartus and map it to the DE2 FPGA board. A **manchester_TX** (transmitter) is provided and instantiated. Switches SW[17:10] form the byte to be transmitted (and are displayed as LEDs and 2 hex nibbles {HEX5,HEX4}. KEY[0] is hooked to a rise edge detector and initiates a transmission. Your **manchester_RX** design is also instantiated and its received 8-bit data drives 2 hex nibbles {HEX7,HEX6}. If you hook the TX of the transmitter to the RX of the receiver (via an external wire) then when you press KEY[0] then whatever byte you setup to be transmitted should show as received. Rst_n is mapped to SW[0] so ensure you set that switch high, or the system is always in reset. See the block diagram below:



NOTE: For this to work you have to connect pin 0 of the EX connector to pin 6 of the EX connector via a jumper wire. See image below.



4 Applied Homework Submission

If working with a partner, your group should submit only <u>one</u> report, and both group members will receive the same credit for the report component of the Applied Homework. The grade for the demo component may differ based on demonstrated knowledge and understanding of the submitted work.

You will upload a number of files to the AHW4 dropbox, be sure that your files are named correctly.

- **state_diagram.jpg** – picture of your completed state (bubble) diagram
- **manchester_RX.sv** – Completed Verilog of Manchester receiver.
- **manchester_RX_sim.jpg** – Image of waveforms for simulation of Manchester_RX
- **manchester_RX_yahoo.jpg** – capture of transcript window

Be prepared to answer questions about the design during the demo. Some sample questions follow:

1. Why did we use/need sig_rise, sig_fall, and sig_ff2?
2. When is the proper time to increment bit_cnt.

The deadline for your report appears online on the course webpage.

Name(s):_____

Only indicate state machine outputs when they are high. It is assumed they are defaulted low.

Draw as a mealy machine. Even though we teach mealy and moore real designers use mealy in real life.