Alan Achtenberg

Lab 7 IR remote hardware

ECEN 449 Sec:505

Due: 11/7/2014

Introduction:

Lab 7 was very similar to lab 2 in the fact that we created a hardware peripheral to implement some logic and then used software to read and display the results. However, the difficulty of implementing the hardware was much harder than lab 2. This fact only exacerbated the difficulty of debugging in hardware.

Procedure:

The first step in the lab was to build an IR receiver circuit. This circuit consisted of a diode sensitive to infrared light and a comparator to output a high(3.3) or low(0) voltage signal. Once we set up our simple circuit we tested it by reading the value on the oscilloscope.

Before we could create our IR decoder we needed to know a little more about our signal. Setting the oscilloscope to a time scale of 4 ms and setting our trigger to occur on the negative edge, we could get a good view of our signal. Right away it was clear that our signal unclear if not held at a good angle and extremely close to the diode. Once we could see our signal clearly using the oscilloscope we determined the approximate length of each of the bits that are transmitted. The Start bit was 2.4 ms long. The “0” bit was 600 us long. The “1” bit was 1.2 ms long.

Once we knew our periods to interpret our signal. We then calculated the period of our clock signal. Using this information we wrote a Verilog module that would count the number of periods our IR signal was active and then store the bit of information in our buffer. After 12 such occurrences the module would then store the decoded message into slv_reg0 to be read later by our software program.

The main difficulty that occurs is managing edge detection of the IR signal to determine when to start counting and when to stop, as well as when to store the values into our message buffer. Debugging

was extremely difficult and in several cases our module passed our test bench but failed on the hardware.

Results: C Source Files

Conclusion:

The number one thing I will take from this lab is to always write a good test bench file. Waiting for xps to generate a bit stream every time you make a small change in your logic is simply too slow to be productive. The ability to simulate our module is invaluable.

Questions:

(a) What are the hexadecimal control codes for the following buttons

Vol-up	490
Vol-down	C90
Channel-up	90
Channel-down	890
Play	FB0
Stop	7B0
Ch1	10
Ch2	810
Ch3	410
Ch4	C10

(b) When a button is pressed on the remote, multiple copies of the same command message are sent.

Approximately how many of the same command message are transmitted after each press of a button? Provide some intuition as to why multiple messages are sent.

Between 3 to 5 messages depending on how long the button was pressed by the user. Multiple messages are probably sent to allow for better error prevention of decoding. For example if the probably of a single message failing is 20 percent then the probability of 2 messages failing would be 4 percent.

(c) What modifications would you make to your code to provide an internal signal that goes high when a new message comes in? You do not have to synthesize this modification, but please provide the Verilog code that would do this. Hint: you can use the message count register. If this signal was made available to the processor, what might this signal be used for?

See modified_user.v for code. I simply added a register that is assigned high whenever my message buffer is stored in slv_reg0 and my message count register is incremented. This signal could be used as a software interrupt to let the operating system know when an IR message has been received.