Christopher Bero

27-2-2015

CPE 324

Lab Report 05

Serial Communication Interface

Introduction

In our previous labs, we've been introduced to Quartus and ModelSim workflows. The learning curve for Quartus has been an unsettling one; unlike most introductions to programming languages Verilog in Quartus presents a useful and capable system of design at varying and interchangeable layers of abstraction, but the editor itself leaves much to be desired and has a resolutely clunky feel to it along with an eyestraining visual interface. In this week's lab we address both aspects by transfering from block level design to behavioral Verilog in a portable instance of Emacs to provide a more capable editing interface.

The task set to us entails communication via RS-232 serial ports between a host computer and the DE2-115 with a scanning keypad, seven segment LEDs, and the LCD character array. If we felt so inclined, the project this week may incorporate the resulting module from last week's keypad excursion as a functional component. This serves as an interesting aspect of FPGA development: we can re-use existing modules from the aforementioned various abstractions in order to shorten design time for a project. In this regard, the goal of creating modules which we understand on a comprehensive level and are confident in the robust and efficiency of is now of higher importance.

Theory and Analysis

We will be communicating to our 'remote' system via an RS-232 port on the DE2-115 at 9600 Baud, 8 data bits, 1 stop bit, no parity. Converting the Baud rate from symbol rate to data rate, the connection is configured to 9600 bits per second. This would allow for a maximum of (9600/

(1+8+1)) = 960 characters every second, a transmission cap with plenty of headroom.

Our design for receiving and producing serial data with our static UART module was an unfortunately obfuscated aspect of the lab. Without a somewhat more details description or understanding of the mechanics of the DE2-115, we can only make assumptions which justify the provided FSM diagrams to implement our protocol. This is the wrong approach, ostensibly, and is similar to issues faced with the MSP-430 line of microcontrollers in another UAH course. The given difference being that students had access to a 900 page manual of internal workings to the line of MSP microcontrollers being examined. Nonetheless, we can infer a connection from the S0-S10 states and each bit in a transmitted byte, with a rather straightforward use of the state machine as a clock to churn out a bit at a time to the UART interface via the shift_reg_10_l_en module for sending, and a similar arrangement for receiving with additional parameters to transport our character to the LCD display.

Resources:
https://en.wikipedia.org/wiki/RS-232
https://en.wikipedia.org/wiki/Baud#Relationship_to_gross_bit_rate
https://en.wikipedia.org/wiki/Serial_port#Speed

Procedures

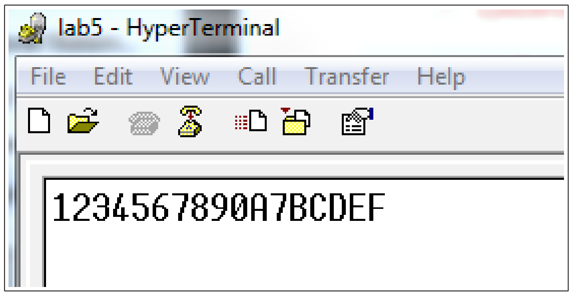
Procedure was followed from the lab manual “Serial Communication Interface” as closely as could be expected given the provided fog-of-war approach to programming in which deigning the utility of an assigned input or output of a module becomes a process of understanding another author's approach to a separate set of parent and children modules from which the 'data' traverses. Without a knowledge of the process by which characters are conveyed to the LCD screen, designing the receiving module became (eight) hours of rather frustrated poking and prodding at the project in Quartus, with an end result of "Oh, it worked." rather than a considerably more desirable "Ah, I understand the system."

At one point, an error in Quartus kept the program from compiling code which appeared to follow closely from examples in the book, and was eventually found to be an improperly placed assignment of 'state' in the receiving module.

Resources:
https://stackoverflow.com/questions/18385370/compiling-verilog-code-in-quartus

Results

Results for this lab were presented in class as an example of a working implementation on the DE2-115. A sample of collected characters from the scanning keypad is provided.



```
1234567890A7BCDEF
```

Conclusion and Discussion

This lab provided a very real example of the dangers involved in operating on a device or system which extends beyond the programmers immediate knowledge. Anecdotally, the team from Data General portrayed in *The Soul of a New Machine* by Tracy Kidder struck a chord with their efforts to develop the Eagle computer with many people grasping at straws for architecture and design principals. It is the observation of this lab participant that discussions and representations provided in only one non-rigorous form are a detriment to the mentality of implementing “the right thing” for a project, rather than following so many other coders of this era in tenuous and kludgy programming.

Provided Questions

1.
The chosen length of count1 and count2 are simply a product of the largest intended value for them to hold. In the case of count1, a 9 bit register is the smallest which can contain the value 325.
At 50,000,000Hz en_out16 is cycled at 50,000,000 / 326 = 153,375Hz. This is surprisingly close to the target of 153,600Hz.

2.
Covered as much as I care to in the “Theory and Analysis” section.

3.
Well… Assuming that the statement “incorporating the external shift register and counter elements” means that we

would be bringing the FSM modules together and continue to have the same input/output lines, this should be entirely possible with just a collection of `always @()` statements which generate the same state representations and outputs.