Lab Four: Shift registers and counters

Ben Smith

***Abstract*—This document introduces concepts related to the implementation of Registers and Timers in the System Verilog hardware descriptive language. The effects of register timing, metastability and the FPGA timing closure process are breifly discussed.**

I. Introduction

THIS lab will reinforce the idea of modularity as we design modules which depend on a clocking signal. Clocking circuits allows the synchronization of large blocks of logic. The concepts this lab introduces are at the core of design with Verilog. The sensitivity list will allow you to communicate with a number of different devices on different system clocks greatly expanding the usefulness of the FPGA. It is important to remember that each one of these blocks evaluates simultaneously, the behavioral block is at the core of Verilog's parallel nature. The purpose of this lab is to introduce the following concepts:

- Using synchronous logic with a system clock
- The implications of timing constraints

A. Clocked designs in Verilog

1) Posedge and Negedge in the sensitivity list: These operators are commonly used in the sensitivity list of a logical block in Verilog. They indicate where operations are performed on a square wave clock signal. Posedge performs the specified operation at the low to high transition and negedge the high to low edge.

2) Register metastability and timing closure: "Timing Closure" is the process of making sure the physical circuits on the FPGA (or any digital system) and produce a desired functionality. The electrical properties of capacitance and inductance limit the maximum operating frequency of a circuit, even on the nanometer scale on the FPGA. The systems that are built in this lab do not approach the abilities of the Cyclone FPGA we are using. But the topic is very important, it has been one of the first questions I have been asked on a job interview. If you would like to learn about the topic (not required) Altera's white paper 01082 offers a technical discussion of metastability and how to mitigate the effects of the phenomenon.

B. Blocking and non blocking operators

Verilog statements can be structured by using two different assignments. These two types of statements must be used to avoid race conditions when your code is synthesized. Once again Clifford Cummings offers a great insight into the operation of these language constructs.

1) Blocking assignments (=): Blocking assignments occur in one step, all other operations are "paused" while waiting for the operation to complete.Evaluate the RHS (right-hand side equation) and update the LHS (left-hand side expression) of the blocking assignment without interruption from any other Verilog statement." [1]

2) Non-Blocking assignments (<=): The evaluation of nonblocking assignments is a bit more complex than the blocking operator and requires special consideration. The execution of these statements is defined as follows. This information is particularly relevant in the simulator as you try to debug your design. [1]

1) Evaluate the RHS of nonblocking statements at the beginning of the time step.
2) Update the LHS of nonblocking statements at the end of the time step.

3) Tips for the effective use of behavioral blocks: The following tips come form A paper discussing the use of blocking and nonblocking statements in Verilog [1]. Mixing the operators without thinking may lead to errors that are very hard to track down. In general following these guidelines will help you write error free code.

- When modeling sequential logic, use nonblocking assignments.
- When modeling latches, use nonblocking assignments.
- When modeling combinational logic with an always block, use blocking assignments.
- When modeling both sequential and combinational logic within the same always block, use nonblocking assignments.
- Do not mix blocking and nonblocking assignments in the same always block.
- Do not make assignments to the same variable from more than one always block.
- Use $strobe to display values that have been assigned using non blocking assignments.

C. Clock division with Counters

The idea of a clock divider in verilog is identical to a divider with discrete logic. Because of the timing constraints on the FPGA, a synchronous counter is the best option for our requirements. If one d flip-flop divides a clock by two, then the addition of another will divide the input clock by four and so on. this can be generalized to

$$f_{out} = \frac{f_{in}}{2^n}$$

where n is the number of bits in the register.

II. Lab Procedure

A. Investigating timing closure

Altera provides a tutorial on the basics of using the TimeQuest Timing analyzer. The effective use of this tool is critical to a successful design. Complete the Altera tutorial and include the contents of the generated timing constraints file (.SDC) in your lab report.

B. Application of registers: Timers

The DE0-Nano has a 50MHz crystal oscillator and this is input to the FPGA clock pin. This clock must used to trigger operations every second and every 2 milliseconds. The operations are triggered by a pulse longer than 100 microseconds. Measure the frequency of this trigger signal using a debugging tool like Signaltap or output the signal to pins and measure the output with an oscilloscope.

Laboratory Demo: Timed pulse generator

Specification: Generate a 100μs pulse every two ms and every second and output these signals to GPIO pins on the DE0-Nano.

Deliverable: DE0-Nano attached to Oscilloscope for verification

Process : The signals need to be verified with the Oscilloscope on the DE0-Nano. Attach the two channel scope to the GPIO pins and use the frequency measurement function.

C. Application of Shift Registers: IO expansion

This sections focuses on a application of the shift register. It can be found in a number of uses like the serialization and de-serialization of data streams. A common trick to expand IO on a device is to use the venerable HC595 shift register. The data for the output pins can be output serially to the shift register instead of directly parallel. This trades update speed and code complexity for IO pin usage. Let's take some indicator LED's for example. They do not need to be updated frequently and we can save the complex IO pins on the FPGA for other purposes. Figure 1 shows how a Shift Register could be used in this configuration.

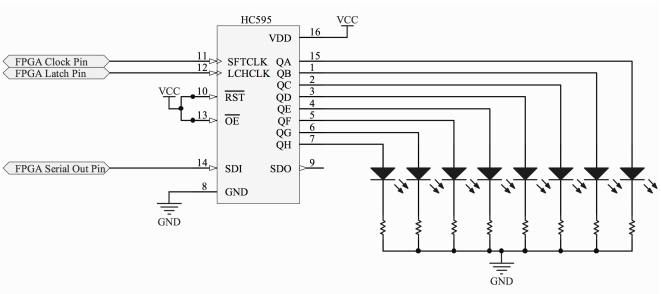

Fig. 1: Example application of 595 Shift Register

Laboratory Demo: IO Expansion with Shift Register

Specification: Use External DIP switches for 8-bit input and control 8 LEDs attached to a Shift Register though a serial line.

Deliverable: DE0-Nano configured with Comparitor, Clean timing report from TimeQuest

Process : Output LEDs will be inspected for proper operation as the student cycles through the inputs

III. Lab Report

THE writeup for this lab should be fairly short. This particular lab report will require:

A. figures to include

- SignalTap captures for the function generator and shift register.
- Code listings for the function generator and shift register
- Netlist schematics from the Quartus RTL Netlist viewer.

B. Questions to Answer

1) The quartus timing analyzer will report a maximum operating frequency for a circuit. This is dictated by how your design synthesizes and the technology of the FPGA. Can you find the maximum operating frequency of your circuit. *hint*

IV. Conclusion

LAB four introduced some very important topics. Most applications for FPGA devices will involve frequency requirements that create very difficult timing problems to solve. This is another area where practicing a skill can have a direct effect on employability. The shift register demo is very similar to the implementation of a *Serial Periphrial Interface* or SPI in industry. Almost all periphrial devices (Sensors, ADCs, DACs, ...) offer the SPI bus to interact with a control device.

References

[1] C. Cummings, "Nonblocking assignments in verilog synthesis, coding styles that kill!" in *SNUG San Jose*, 2000.