55:131 Introduction to VLSI Design
Project 2: Counter Simulation
Due Date: Friday, November 1, 2013

This is the second of three projects developed for our class. This project uses the Questasim digital simulator to design and simulate a synchronous 4-bit counter. The counter is created using (only) several inverting MUXes we created in project 1.

I. What is Questasim?

Questasim is a digital circuit simulator from Mentor Graphics. Questasim supports several languages, including VerilogHDL, SystemVerilog, VHDL, and SystemC. Tool Command Language (TCL) is a scripting language for controlling and extending Questasim. Like most digital simulators, Questasim is an event-driven simulator.

II. Installing Questasim

Questasim is installed on the Linux-based computers in the Elder and Herring labs. After logging in, you'll need to source the Mentor setup file by typing the line below at the Linux prompt (assuming you are using the *bash* shell):

```
source /usr/css/etc/mentor_setup.sh
```

To avoid having to source this file every time you start a session, you can add the source command line in your .bashrc file (which is in your home directory):

- open your .bashrc file with a text editor
- add the following two lines at the end of the file (the first line is only comment):

```
#source of setup file for Mentor Tools
source /usr/css/etc/mentor_setup.sh
```

Make sure you type the file's path correctly! This way, in the future, when you start a session you will not need to type the "source" command.

III. Counter Description

The requirements for this design are shown in the tables below. One of the key ideas in this project is the notion that all of the combinational and sequential logic in a design can be built from just one primitive (such as the inverting mux). The FPGA library could consist of just one gate! Although this may seem contrived, there are some highly granular FPGAs available which use thousands of primitive "tiles" which can implement combinational logic functions or flip-flops. This "tile" is thus used for all user logic except SRAM, IO blocks, and PLLs.

Table 1 IO Requirements

Req#	Signal	Function	In/Out	Asserted	Description
1	C	Clock	In	↑	Sample D input on rising edge
2	R	Reset	In	Low	Asynchronously clear outputs
3	Q	Output	Out	High	4-bit counter output, Q[3] is most significant bit, Q[0] is LSB

Table 2 Functional Requirements

Req#	Text	Comments
4	Counter outputs shall be “0000” when reset signal is asserted low	Asynchronously clear outputs regardless of clock
5	Counter outputs shall increment at every rising edge of the clock when reset is high	
6	Counter outputs shall rollover from “1111” to “0000”	4-bit counter output

Table 3 Implementation Requirements

Req#	Text	Comments
7	All combination logic and flip flops in the counter shall be built from the inverting mux primitive	Asynchronously clear outputs regardless of clock
8	The counter shall be synchronous	Global reset signal connected to all flip-flop “R” inputs with no intervening combinational logic, global clock input connected to all flip-flop “C” inputs with no intervening combinational logic.

For this project, the inverting mux is the only thing you will write using behavioral style modeling (other than the test bench). Everything else is built using structural style modeling using the inverting mux as the building block.

Designing, coding, and testing your most designs are significant tasks, even for a simple one like the counter in this project. With experience, the time needed to design and code will become fairly predictable. Debugging is not as predictable, even for “seasoned” engineers. So don’t wait to get started on this project – there may be bugs to fix! It often helps to get parts of the design and test bench working, and then add other capabilities and features incrementally after that.

IV. Recommended Steps

1) Design the inverting mux using behavioral style modeling. See figure 1. For now, don’t include any delays in the design.

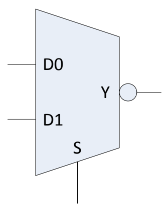

Figure 1 – Inverting MUX

2) Create a 2-input NAND gate, inverter, and positive-edge triggered flip-flop from the inverting mux, in structural style modeling. The inverter and NAND gate simply use one or two of the inputs and strap the others to power or ground to form the correct function. The flip-flop is a little more complicated, so it has been provided in the figure 2 below. It is advised that you label each node in this figure with a “wire” name and build the structural model using those wire names.

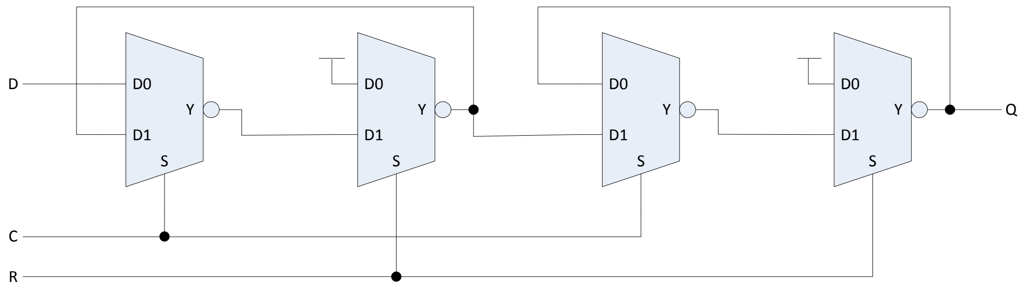

Figure 2 – Flip-flop Created from Inverting MUXes

3) Create the 4-bit counter using the flip-flop, inverting mux, nand gate, and inverter, again using structural style modeling. While designing the counter may be obvious to some students, hints are still in order.

   - Note how each counter output (other than the LSB) changes at the positive edge of the clock whenever all previous stages are ‘1’, otherwise it holds its current value. For example, in figure 3 at the transition from “3” to “4”, Q[0] and Q[1] are ‘1’. Q[1] and Q[2] therefore change (in this case from ‘1’ to ‘0’ and ‘0’ to ‘1’ respectively) and Q[3] doesn’t change. Craft a way to implement this logic using an inverting mux (providing the “change” and “hold” values to the flip-flop’s D input).

   - Q[0] changes unconditionally at every positive edge of the clock, so the logic needed to drive that flip-flop’s D input is much simpler.

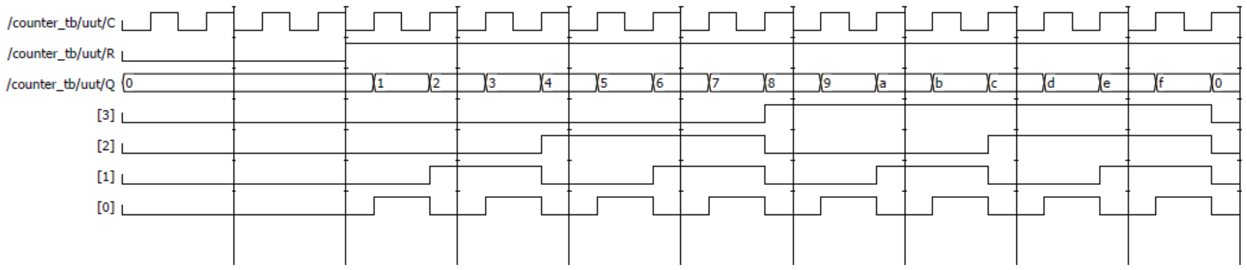

Figure 3 – Counter Signals

4) Create the counter test bench which instantiates the counter, provides the reset and clock inputs, and monitor the counter’s output. The full Verilog (or SystemVerilog) language is available for this module.

   For now, set the clock to a 100MHz period with a 50% duty cycle. Don’t forget to briefly assert the reset input for a few nanoseconds at the beginning of the simulation.

V. Test the Counter

It is recommended that the test bench tests the device against its requirements. In industry, the design and verification teams are usually independent, and the verification team writes its test cases based on the requirements, not on the design implementation team (the design team may have misinterpreted the requirements).

The test cases and test bench(es) usually automate testing and results verification. For example, during reset, the counter outputs are checked to be "0000" by the test bench. If not, the test fails and the verification engineer is notified. Similarly, the counter itself could be modeled behaviorally in the test bench to mimic the RTL, and the outputs can be compared.

VI. Measure Setup Time and Clock-to-Q (Propagation) Delay

Our goal now is to rerun the simulation and estimate the flip-flop's setup time and clock-to-Q delay. Note that setup (and hold) time is a constraints on the flip-flop D input which must be met for proper flip-flop operation, while clock-to-Q delay is directly measureable characteristic of the flop-flop itself. These times and delays are briefly described in Table 4.

Table 4: Delay Descriptions

Time/Delay	Description
Setup	Time that the data (D) input must be stable, before the rising edge of the clock, to guarantee that that value of D will be captured by the flip-flop. Note that it is possible for this time to be negative. It is recommended that the setup time be measured for both values of D (0 and 1).
Hold (information only)	Time that the data (D) input must be stable, after the rising edge of the clock, to guarantee that that value of D will be captured by the flip-flop. Note that it is possible for this time to be negative. It is recommended that the hold time be measured for both values of D (0 and 1).
Clock-to-Q	Maximum delay from the rising edge of clock to the change of state of the output. It is recommended that the clock-to-Q delay be measured for both transitions of Q (0 -> 1 and 1 -> 0).

First, edit the RTL for the inverting mux to add the propagation delay measured in project 1. This is done by using the `timescale directive and inserting a delay (#delay_amount) in the "assign" statement where the output signal was assigned in the mux. For example, for a delay of 1 nS, the RTL edits would look like:

```
// directive
`timescale 1ns/100ps
…
// assignment
assign #1 Y = (S == 1)? ~D1 : ~D0;
…
```

To estimate the setup time, adjust the clock period iteratively smaller until you find the point (to the nearest 100 pS) where the flip-flop just continues to work (i.e. any smaller and it will fail). Save a copy of these waveforms. To find the clock-to-Q time, measure the time from the rising edge of the clock to the edge of Q. Note that cursors are available in the wave window to aid making measurements.

VII. Asynchronous Reset and Metastability

Return the clock rate to 100MHz, run the simulation, and note the flip-flop input and output values while the counter is in reset. Recall that the clock and reset signals are asynchronous to one another in this design – the reset can be asserted, or removed, at any time in relation to the clock. Try removing reset at exact point of a rising edge of the clock. Save a copy of these waveforms.

VIII. What To Turn In

Please submit an electronic copy of each of the following items into ICON. Answers to questions can be submitted in a text file or Word document. Make sure the images are easy to read to avoid grading difficulties.

1. Please indicate how many hours you spent on this project. This will not affect your grade, but will be helpful for calibrating the workload for the future.
2. Code for the inverting mux, NAND2, inverter, flip-flop, and counter.
3. Code for the counter test bench.
4. Signal waveforms for maximum frequency counter operation.
5. Signal waveforms for metastable counter operation.
6. How fast could the counter run in the simulation (in MHz) before the delay was added to the inverting mux?
7. How fast could the counter run after the delay was added?
8. What were the setup times you measured for the flip-flop? Justify the times based on the flip-flop construction (i.e. the inverting mux delays).
9. What were the clock-to-Q delays you measured for the flip-flop? Justify this time based on the flip-flop construction.
10. What were the flip-flop input and output values during reset? Can you explain how any of these D/Q signal pair values could cause a problem?
11. Explain the metastability problem demonstrated in section VII.