CE 126: Advanced Logic Design
Midterm Report

Darrell Ross
sdr85@cats.ucsc.edu

1. Introduction

The SRAM tester that I designed for the BORG board went through several design stages. The first design was one giant, twenty-eight state, FSM that did everything except generate the patterns. The main point of that design was to have a working design for the check-in point and to learn how everything worked. I actually had another design that had the SRAM memory control in a separate block but I could not get it to work in time.

After the check-in, I got rid of my huge FSM and worked on building a glitchless SRAM control block. I got my design working pretty quickly because I had learned all of the ways-of-the-road during the design of the previous machine. Another advantage was when I collaborated with other people— not really for design ideas but for possible bugs in individual designs. This helped me avoid a lot of possible bugs ahead of time, like tri-stating the SRAM control and making sure my OE SRAM signal went low for the right length of time. The people that I worked closest with were Vincent, Max, and Chris. Saar also gave me some good pointers.

2. Blocks

I split my design into three main blocks: the SRAM control, the pattern generator, and the main control unit which controlled the first two and all other blocks. Minor blocks include the thirteen bit counter, the edge detector, and the eight-bit comparator.

2.1 SRAM Control

The SRAM control was difficult to build. My first design used sequential encoding which left lots of room for glitches. For my final design, I followed Vincent's advice. The best way to design glitchless control was using one-hot encoding. However, my SRAM control has seven states and that would have meant seven bits. It was much easier to use a state table to design the state bits to fit the outputs as Vincent had suggested. In the end, the three-input, four-output FSM used only four flip-flops and was much easier to use.

2.2 Pattern Generator

My first pattern generator was a fairly large FSM that had nine states and seventeen outputs (eight for each pattern and one for the pattern TC). Long showed me the beauty of the shift register. Unfortunately, it had far too many inputs. I asked Martin and she fixed it by suggesting that we build our own and then showing us how! Her design works perfectly and is nice and compact using nine flip-flops and having the read pattern be one step behind the write pattern.

2.3 Main Control

The main control of my SRAM tester has thirteen states. I have diagramed the FSM in the following pages. The main control has five inputs, besides the clock:

GO ~ go signal to start test
EQ ~ result of comparator
TC13 ~ terminal count for the 13-bit counter
TCP ~ terminal count for the pattern generator
DONE ~ done signal from SRAM instructions

and eight outputs:

CE13 ~ 13-bit counter clock enable
CEP ~ pattern generator clock enable
RD ~ read signal sent to SRAM
WR ~ write signal sent to SRAM
ERR ~ error signal (only low in ERR state)
PASS ~ pass signal (only low in PASS state)
IN ~ bit shift value for pattern generator
TRI ~ tri-state enable for the address bus

One very convenient way of debugging with this software is to make sure that Mustang encodes the FSM exactly the way you want it so that you can tap the PS or NS wires and tell what state you are in. I was able to look at the screen and think "yes, that's the right sate."

The main control basically follows the psuedocode laid out in the assignment: a simple two-state pattern of writing the first pattern and then incrementing the 13-bit address counter is first. Followed by a read/check and write cycle which tests each patter on each address except for the last patter. The final part checks just that last pattern.

2.4 Miscellaneous

My edge detector is simply two flip flops that I could probably just put out in the open. The comparator is just a block from the Xilinx libraries.

3. Interconnections

When my main control sends a RD signal to the SRAM, the RD signal remains high the entire time. The same is true for a WR signal. I found it convenient that, when I delayed the write signal for one clock cycle using a flip-flop, it could be used as the tri state enable. the Pass and Error signals are only active in their respective states and somehow I managed to flip them around so that a '1' response is a pass and a '2' response is an error. I had the unfortunate problem of having the Done signal from the SRAM controller return after the CS and OE signals went high. This meant that the BORG could never do the comparison because the equal search came after the read signal finished. In the end, I just delayed the EQ signal with a flip-flop.

4. Conclusion

This project was very educational and fun. It took a bit too much time though. I have run the tester and it gives and error on all the correct inputs and passes correctly as well.