Design and implementation (in Verilog) of an SRAM driving the DAC along with a Microblaze embedded processor
Report and Signoff due Week 10 (November 14)

This project involves a digital system design in Verilog targeting the Spartan 6 on the Nexys3 board. It involves interfacing to external devices (SRAM, DAC, and VGA monitor) and also includes a Microblaze embedded processor programmed in C. It shows how an existing IP written in VHDL (the VGA controller) can be integrated into the rest of the Verilog design.

Part a (background):
- Convert the seven_segment display you used in project 1 from VHDL to Verilog
- Important: this should be designed and implemented as a stand-alone module.
- Test this out by using the slide-switches to enter various numbers.

Part b (background):
- Read the DS865 data sheet
- Complete the Microblaze MCS tutorial including the extra steps to add switches and leds.

Part 1:
- Add a DCM to create 100MHz for the Microblaze and 25MHz for the VGA controller (and other logic in your design)
- Create a VGA display to display a small white block (16 pixels wide by 16 pixels high) that moves from the top left corner to the top right corner then moves down one block row, and repeats. The block should move at 2Hz.
  - Instantiate the Digilent VGA Controller as a VHDL module (do not modify any of the code or convert to Verilog)
  - Show the x and y coordinates of the block on the seven segment displays as it moves

Part 2:
- Use the SRAM to load 8 bytes that drive the DAC to produce a repeating sawtooth pattern (from 0x00 to 0xFF)

Part 2a (DAC data load phase):
- Use the slider switches to load 8 bytes into SRAM for the DAC data bytes. Load the value on the slider switch and then press a button to load the value into the first location of the SRAM. Load the second value on the slider switches and press the button again to load the next value. Repeat for all eight bytes. (Note – you will need to debounce the button presses.)
- Use the seven-segment display to show the byte being loaded (00 to FF) and the address location (0 to 7).
- Also display this same data and address information on the PC HyperTerminal window (with suitable text e.g. “loading 0x33 to address location 2”)

1
Part 2b (DAC pattern generation phase):
- Once all the eight bytes have been loaded press a button to start the sawtooth pattern from the DAC
- Capture a ‘scope picture of the DAC waveform and integrate with the rest of your report.

Part 3:
- Modify the Microblaze to add a second 16-bit port and connect to the seven segment display. Show an incrementing decimal number running at 10Hz. When a push button is pressed increment the number and when another push button is pressed decrement the number.
  - Do not modify the seven segment display module. Change the values you send to it.

Part 4: Simulation and Testing
- Create a test bench to show a write and read of two values from the SRAM (use the simple SRAM model provided)

As usual, this is not a complete description – make whatever additions you think are necessary.

Prepare a sign-off sheet and demo your system (part1, 2, and 3) along with your Verilog source files, and write your report before the deadline.

Reference Material

Make sure you read the Memory section of the Nexys3 Reference Manual.

The external RAM is a 128Mbit Micron device connected to a 16-bit bus. We will only use the bottom eight 8-bit locations to store the DAC data (you will need to connect unused address lines to logic ‘0’ and configure the device for just byte-wide access). The 16-bit data bus is shared with the on-board FLASH device so make sure you disable this device by using its chip select line.

Note: You should be able to verify the contents of the SRAM by setting a RAM address on the DIP switches and pressing a read button. Use the seven segment displays to show the contents of the SRAM.
Grading Guidelines

• [40 pts] Implementation
  o [40 pts] Design works on board and meets requirements

• [25 pts] Source Code – Verilog and C program in Appendix
  o Code style and comments (well-commented and tab-indented code!)
  o Use of case vs. if, spaghetti code vs. structured, etc.
  o Recognizable implementation of "standard" elements (state machines, counters, clock dividers, decoders)
  o Sensible use of modularity
  o No latches or other synthesis problems
  o Test bench code

• [35 pts] Lab Report
  o [5 pts] Brief Introduction / Problem Statement
  o [10 pts] General Overview of approach to solution and description (include Block and State Diagrams with descriptions). Include picture of VGA display showing the block and HyperTerminal window showing text.
  o [5 pts] FPGA Resource usage (# flip-flops with explanation) and listing and explanation of warning messages (don’t copy all the Xilinx reports – just the relevant sections)
  o [10 pts] Test bench description and simulation waveforms (annotate and explain well)
  o [5 pts] Conclusions
    • Problems faced in implementation
    • Solutions used to solve problems
    • Lessons learned from the project
    • Suggestions for further improvements and extensions

• [10 pts] Extra points
  o Possible extra points for good additional features or capabilities (need to demo on board and include description in report)