Design and implementation (in Verilog) of a Microblaze embedded processor connected to a DAC
Report and Signoff due Week 10 (November 8th)

This project involves the design and implementation of a digital system design using Verilog. It involves interfacing to external devices (DAC) and also includes a Microblaze embedded processor programmed in C. It shows how an existing IP (the seven-segment display) can be integrated into the rest of the Verilog design, and how software can be used to control custom logic.

Part a (background):
- Convert the seven_segment display you designed 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: Microblaze (hardware connections and software additions)
- Add an MMCM to create 100MHz for the Microblaze and 10MHz for the DAC serial clock and other logic in your design.
  - (Derive other slower clocks from the 10MHz signal)
- Connect 4 push buttons to a 4-bit GPI port on the Microblaze so you can read their value in software.
- Connect the 16 LEDs to a 16-bit GPO port so you can control them in software.
- Connect the 8-bits of your seven segment display to a new 8-bit GPO port on the Microblaze so that it can be controlled by software.
- In response to a push-button press, prompt the user to enter a two digit number on the PC keyboard (e.g. ‘12’) and display this number on the bottom two seven segment displays.
- If a push-button is pressed display a message on the PC with the name of this project.
- If another push-button is pressed display a message on the PC showing your name
- If another push-button is pressed turn on discrete LED (0). Each time the same push-button is pressed turn on the next led one place to the left until all the LEDs are on.
- Your software should allow for the previous described actions to be carried out in any order.

Part 2: DAC – waveform generation
- Read the AD703 Serial Input, Dual 8-bit DAC data sheet
- Use the PMOD DA1 Module provided in class
  - Just use the first AD7303 and only DAC A (ignore the other 3 DACs)
  - Implement the DAC SPI interface with a state machine and shift register.
• Use a 10MHz SPI clock
• Use 100KHz for the DAC update (sync pulses)
  ▪ Update the DAC every 100KHz with new 16-bit data values
• Verify the DAC interface is working by first using a constant value to drive the DAC (use the slider switches).
  ▪ You should see a constant voltage out of the DAC (in the range 0 to 3.3V) that changes as you modify the slider switches
  ▪ Display the 8-bit data value you send to the DAC on two of the seven-segment displays
• Create a sinewave waveform with a frequency of 6.25KHz using 16 constant values per cycle (calculate the necessary constant values)
• Use an oscilloscope to capture pictures of the sine waveform and the SPI interface
  ▪ For all ‘scope pictures, preferably take a screen capture with a USB flash drive rather than a camera picture. You should be able to clearly see all the signals and the timebase.

Part 3: Simulation and Testing
• Use an oscilloscope to
  ▪ Capture a complete 16-bit DAC transfer using (show the SCLK, SYNC, and DIN lines on the same scope capture) – describe this picture in your report.
  ▪ Verify the sine and triangle waveforms (take a ‘scope screen capture and include in your report)
• Create Verilog test fixtures to show the DAC logic interface working (show your state machine states and the transfer of sine wave data to the shift register and the SPI interface)

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

Prepare a sign-off sheet and demo your system (parts 1 to 2 combined) along with your Verilog source files, C program, and write your report before the deadline.

Reference Material

Read the DAC module data sheets
Sine wave generation

6.25KHz Sine Wave with 100KHz DAC Sync pulses

100KHz (10us) DAC updates – new analog voltage output after each update
Grading Guidelines

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

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

- [35 pts] Lab Report
  - [5 pts] Brief Introduction / Problem Statement
  - [10 pts] General Overview of approach to solution and description (include Block and State Diagrams with descriptions). Include oscilloscope pictures and describe.
  - [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)
  - [10 pts] Test bench description and simulation waveforms (annotate and explain well)
  - [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
  - Possible extra points for good additional features or capabilities (need to demo on board and include description in report)