CSCE 587 Microprocessor Design and Synthesis

Homework 5 – Racing Car Game with CPU

Assignment (300):

Perform the following tasks and submit homework to doug@sidechannel.net. Zip all files into a single file named <lastname>-hw05.zip. Strive to keep final zip file as small as possible. (Do not send multi-MB pics – shrink/reduce size to something reasonable.)

1. Read chapter 28 on the Racing game with CPU and recreate the game on your FPGA board. Instead of using hpaddle and vpaddle inputs to update the car/sprite position, initialize the position of car in the middle of the screen and use two push buttons to move right or left.

Notes:

- The push buttons will most like drive a counter that determines horizontal position, the car does not move vertically.
- The provided FEMTO-8 assembler compiles code to binary values; use it generate the information needed to create a "hardcoded ROM" which stores the program. (I would recommend testing any assembly code changes in the online simulator first.) The assembler is not tolerant of spaces between operands. For example "mov A, [B]" will not assemble correctly, but "mov A,[B]" will – test code in browser as much as possible to identify errors.
- Feel free to delete various "_top" modules scattered throughout original source, to reduce overall complexity and remove dependence on hvsync_generator module.
- You can work in teams of up to three people to complete this project (two is probably optimal). If a team develops a solution, make one submission and make sure to indicate who was on it. There is plenty of work for team members - especially if work is divided between hardware and software.
- Consider using the online simulator as a useful feedback mechanism for changes, especially changes associated with new registers in a more extensive memory map. If this approach is taken to implement a working racing car game, there is probably little use for dual-ported RAM as all contention for position information should be resolved. This includes prototyping software.

Assembler Usage:

- Using the assembler is easy; just supply the code to compile in a text file as the first argument. Output will be displayed indicating binary values needed to initialize the ROM, plus a string of hexadecimal opcode / operands that can be cut and paste into a text document to be read in as a "memory" file.

Hints:

- As discussed in class, the decision logic to trigger the sprite renderers needs access to player and enemy position information. If this information is stored in RAM (as it is now), accessing it will

be problematic as only specific values (given an address) can be read and evaluated on each clock cycle. To avoid creating a potentially complicated state machine to generate all the signals required to drive the renderers, it might be easier to “map” in a few new special purpose registers to store player and enemy position information directly.