LAB 8 – Full System Integration

SESSION II

SPEED UP THE SNAKE

In this part of the exercise, we will try to control the snake's speed. The idea is to use two switches on the starter kit to control four different speed values. The tasks are the following:

1. Extend the top-level hierarchy to accept inputs from switches and transfer the values to registers.
2. Understand the provided assembly program *snake_patterns.asm* and modify it to accept a 2-bit input value from the switches. Based on the input, increase or decrease the speed of the snake.
3. *Optionally, you have two challenge tasks to complete.*

Extending the I/O functionalities

For the crawling snake exercise in the previous part, we only had to store the value to drive the 7-segment display appropriately. In this part, we need to read the values from the switches on the starter kit. For this purpose, we add an additional I/O register to the MIPS processor that is 2 bits wide (1 bit for each switch) and assign an address. The table of I/O registers is as follows:

Address	Direction	Width	Description
`0x00007FF0`	`out`	`28 bits`	Value to be sent to the 7-segment display.
`0x00007FF4`	`in`	`2 bits`	Speed step 0, 1, 2 or 3.

In the top.v file, create an input signal for the switches. Depending on the value of `IOAddr,` set the `IOReadData` signal that connects to the MIPS processor. Modify the XDC file so that the input signal is connected to the correct FPGA pins. Synthesize the project to check for any errors. Note: `IOReadData` is a 32-bit signal; however, we use only the 2 LSBs to read the status of the switches.

Pin	Label on board	Description
V16	SW1	Slide switch, MSB of speed step amount.
V17	SW0	Slide switch, LSB of speed step amount.

Modifying the Assembly program

The assembly code from Part 1 simply loops continuously with no controls. Therefore, we need to modify the code to accept the input from the switches and accordingly control the speed of the crawling snake.

Using the MARS environment, modify the assembly program *snake_patterns.asm* so that it will read in the 2-bit switch value using memory-mapped I/O. Then, depending on the input, modify the delay loop such that the snake moves faster or slower.

Hint: You can use the MARS environment to check your modified code. Just remember to test it with a smaller loop counter value.

Your modified program needs to be copied into the text file *insmem_h.txt*. To do this, you can use the Memory Dump option within the MARS assembler. By selecting the "Memory Segment" as ".text" and "Dump Format" as "Hexadecimal Text", you are able to generate the required file. All you have to do is use a text editor and make sure that the file has exactly 64 lines. All lines after your real code have to be filled with zeroes.

Since we use memory-based variables in the data memory, the ".data" segment has to be exported to *datamem_h.txt* the same way as the ".text" segment. Notice that, after exporting, the *datamem_h.txt* file contains many lines. Make sure that the file has 64 lines, and delete all lines after that.

Generate the programming file in Xilinx Vivado, program the FPGA and control the speed of the crawling snake using the switches. Fix any problems you might have and show your results to an assistant.

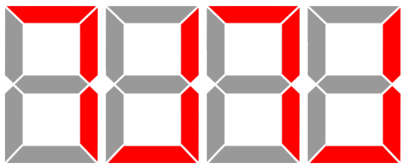
Summary of the Design Flow

The following figure summarizes the steps to follow while building the system. The workflow can be divided into hardware and software sections. The software section consists of the assembly program that the MIPS processor executes. The hardware section comprises the MIPS processor and the peripheral system itself. The assembly code is written and validated using the MARS simulator environment. After checking if the code produces the desired output, the data and instruction memories are dumped as hexadecimal values. The two memory dumps represent the data and opcodes that are used by the MIPS processor. The hardware architecture is constructed in Verilog using Xilinx Vivado. The *DataMemory.v* and *InstructionMemory.v* files use the files dumped by MARS to instantiate the memory blocks. The XDC file is modified to map the circuit to the FPGA board. Finally, the bit file used to program the FPGA is generated. Remember to follow the steps when you make any changes to the files.

The Challenge

We now have a complete system consisting of a MIPS processor that can run our own programs, accept inputs from the onboard switches, and display results on the 7-segment display. Here are some ideas for the challenge seekers.

Challenge 1: Change the snake crawling pattern. For example, modify the assembly code to crawl the snake in a zig-zag pattern, as shown in the figure below.



Challenge 2: Toggle the direction of the snake. Add an additional switch input to control the direction of the snake's movement. For example, the snake moves forward (you can choose to either loop or zig-zag) if the switch is 0 and backward when the switch is 1. This is more challenging because of the limited number of instructions that can be executed in our MIPS processor. You will also need to modify the top-level hierarchy to add a register to transfer the value of this new switch.

Last Words

In this exercise, we have realized a complete and working microprocessor that is able to execute a small subset of the MIPS instruction set. It doesn't seem to be that complex, does it? There are two main reasons why writing the code of the processor was (relatively) easy.

1. First, MIPS uses a simple architecture, and one of its priorities is a simple design implementation.
2. Second, we started from a well-documented block diagram (taken from the book) that had already decided how the processor was partitioned, what signals were used to connect them, and how the signals were defined. All that was left was to convert the block diagram into Verilog syntax.

Additionally, we added the capability to interface with the rest of the world so that we can transfer the processor to the FPGA board and actually see our programs execute.

In the following lab, we will improve the current processor by adding new instructions and speeding up calculations.