Lab Assignment #6

Guideline

This lab can be done with a partner. Required starter code is provided on Canvas.

Objectives

- To implement a stack calculator.
- To get more familiar with block RAMs on an FPGA and understand memory interfacing
- To understand how to model buses in Verilog.

Description

In this lab, you will write Verilog code to implement a stack calculator using a memory module (block RAM on the FPGA) and the board I/O.

A stack calculator stores its operands in a stack structure and performs operations (e.g. addition, subtraction, etc) on the top two values of the stack. The operands are popped off the stack and the result is pushed back on the top of the stack, so the stack is one element less than it was before the operation. The output of the calculator is always the value at the top of the stack. The stack may contain more than two operands at any time, but operations are only performed on the top two values. This is similar to the Reverse Polish Notation (RPN) used by old TI calculators.

You will implement a simple stack calculator using Xilinx BlockRAM as the storage for the stack. The memory supplied has 128 locations by 8 bits (1 byte) wide. Please check the synthesis report (not the utilization report) to make sure that Vivado is synthesizing the memory module using BlockRAM and not distributed LUT-RAM. If it is not BlockRAM, then you can add the following line to your XDC file:

set_property RAM_STYLE BLOCK [get_cells *]

We will also provide you the code for top block which integrates the controller, memory along with the data bus. You will need to modify the supplied code to implement the bus interface to memory and the controller. This will involve using tri-state buffers. Through tri-state buffers, we will be able to ensure that only one driver drives the data bus at a time..

You will also need to implement a master controller for the calculator. This controller contains three registers, a stack pointer (SPR), a display address register (DAR), and a display value register (DVR). The SPR will contain the address of the next free address past the top of the stack. The DAR will hold the address of the data that should be displayed on the output. Whenever the SPR is updated, the DAR should be updated to SPR plus 1. The DVR contains the value that should be displayed on the output. The content of the DVR is the value stored at the memory location pointed to by the DAR. This should be updated every time the DAR changes by reading from the memory location contained in the DAR. The memory will be 128 bytes total giving a 7-bit address. So the SPR and DAR will both be 7 bits wide and the DVR will be 8 bits wide. The master controller will be responsible for taking in the inputs, updating its registers accordingly, as well as performing the operations of the calculator and displaying the outputs. Before use, a reset/clear operation should be used to initialize the calculator. The SPR should be initialized to 0x7F, the DAR to 0x00, and the DVR to 0x00 (don't read from memory this time). As data is pushed on to the stack, the SPR will decrement. In other words, the stack will grow towards decreasing addresses.

The calculator will use all of the inputs/outputs on the Xilinx board. The seven segment displays will show the contents of the DVR in hex (only two of them will be used because the data size is 8 bits). Values will be entered, 8 bits at a time, using SW7-0 on the board (SW0 maps to the LSB). LED[6:0] show the contents of the DAR bits 6 down to 0 and LED[7] will map to an 'EMPTY' flag. If the stack is empty (the SPR contains 0x7F), the EMPTY flag (LED[7]) will be set to '1'.

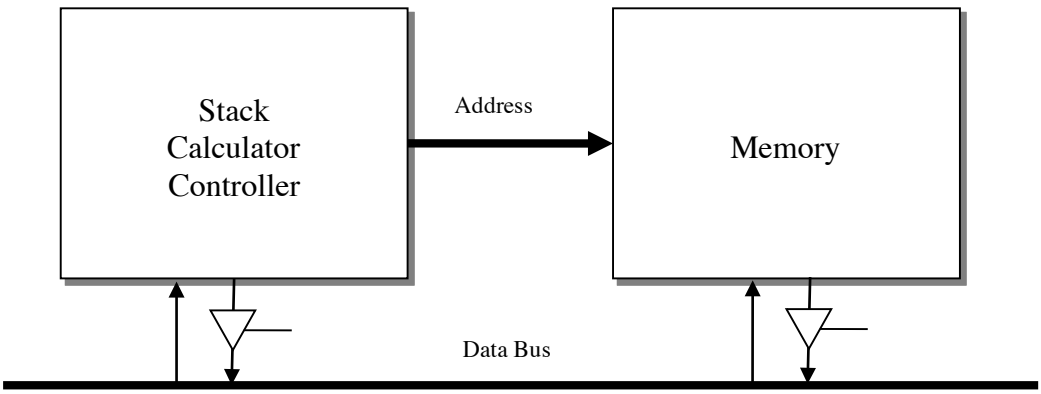
The buttons/switches will provide the operational inputs to the controller. Each button will implement a function as defined in the table below:

Mode	**SW15**	**SW14**	**BTNL**	**BTNR**
Push/Pop	0	0	Delete/Pop	Enter/Push
Add/Subtract	0	1	Subtract	Add
Clear/Top	1	0	Clear/RST	Top
Dec/Inc	1	1	Dec Addr	Inc Addr

- **Enter/Push**: Reads the value from SW7-0 on the board and pushes it on the top of the stack. To do this, keep SW15 and SW14 at 0 and press BTNR
- **Delete/Pop**: Pops and discards the 8-bit value on the top of the stack. To do this, keep SW15 and SW14 at 0 (ie. down) and press BTNL.
- **Add**: Pops the top two 8-bit values on the stack, adds them, and pushes the 8-bit result on the top of the stack, discarding the carry bit. To do this, keep SW15 at 0 and SW14 at 1 and press BTNR
- **Subtract**: Pops the top two 8-bit values on the stack, subtracts them, and pushes the 8-bit result on the top of the stack, discarding the borrow bit (**High Addr minus Low Addr**). To do this, keep SW15 at 0 and SW14 at 1 and press BTNL
- **Clear/RST**: Resets the SPR to 0x7F, the DAR to 0x00. The stack should be empty now (EMPTY flag should be set to 1). This instruction does not access memory. To do this, keep SW15 at 1 and SW14 at 0 and press BTNL.
- **Top**: Sets the DAR to the top of the stack (SPR+1; will cause the DVR to update). To do this, keep SW15 at 1 and SW14 at 0 and press BTNR
- **Dec Addr**: Decrements the DAR by 1. To do this, keep SW15 at 1 and SW14 at 1 and press BTNL
- **Inc Addr**: Increments the DAR by 1. To do this, keep SW15 at 1 and SW14 at 1 and press BTNR

The following figure shows the block diagram of your design:

Top-Level Design



Example

Suppose you want to add 0x92 (binary form: 10010010) and 0x25 (binary form: 00100101). You should first push these two numbers on the top of the stack (the stack at this time can contain other numbers). Perform the following sequence to enter 0x92 and then 0x25 into the stack:

0. Reset the calculator, switch SW15 to 1, SW14 to 0 and push BTNL. At this point, LED[7:0]=1_0000000 and 7-seg are 00
1. Set the switches (SW7 down to SW0) to 10010010.
2. Set SW15 back to 0. Push BTNR. At this point, LED[7:0] are 0_1111111 and the 7-segs are 92.
3. Set the switches (SW7 down to SW0) to 00100101.
4. Push BTNR. At this point, LED[7:0] are 0_1111110 and the 7-segs are 25.
5. Set SW14 to 1 and push BTNR. At this point, LED[7:0] are 0_1111111 and the 7-segs are B7.
6. After steps 2 or 4, you can pop (delete) the numbers you have entered by setting SW15 and SW14 to 0 pressing BTNL.

Useful Information

1. Note that the overflow, underflow, and pushing both BTNL and BTNR at the same time are not considered in this lab. In general you can assume the calculator will be used as described, i.e. you do not have to worry about the error conditions like POPing without having pushed anything, decrementing DAR beyond the lowest address. Keep it simple.
2. All data on the stack should be considered unsigned.
3. You must model the tristate buffers in the format provided in the top module starter file. **There are two assign statements to *data_bus* and you can only modify the right hand side of these two assignments.**
4. Also note that the INC and DEC commands affect the DAR and DVR. They don't modify SPR. INC and DEC are just to be able to see the contents of various locations on the stack. Similarly, the POP/PUSH/ADD/SUBTRACT will use the SPR (however, they will update the DAR and DVR as well).
5. Simulation is NOT a requirement to get full credit if your design works perfectly on the board. If it does not work on the board, please have simulation ready for partial credit.
6. The simplest way to approach the design of the controller is to use a large state machine. The first state will be state will waits for inputs from the user. You will jump from this state to others depending on the inputs.
7. The memory works on 100MHz, but has single cycle latency. This means that when reading from the memory, if you make WE 0 in one clock cycle (in other words, in one state of the controller), you should read data from the data bus in the next clock cycle (in other words, in the next state of the controller). Similarly, when writing to the memory, you should make WE 1 in one clock cycle and wait for one clock cycle to let the memory write the data.
8. The controller can use as many cycles (states) as it wishes to perform the tasks (like POP, PUSH, ADD, etc). Since the clock frequency is 100MHz, even if the controller takes 10 cycles (say) to perform an operation, the user won't be able to see the lag with the naked eye.
9. If DAR deviates from **SPR+1** because of successive **Dec/Inc Addr** operations, it must be re-updated to **SPR + 1** after an operation that is not **Dec/Inc Addr**.
10. If you need, you can modify the ports of the modules given to you **except the top or memory module.**

Submission Details

Submit the following files through Canvas. No hard copy submission is required.

- Typed Verilog Code (.v files)
- [filename].bit file
- [filename].xdc file
- Synthesis report showing that your final design does not contain any latches and that block RAM has been used in the design

Checkout Details

Demonstrate the calculator working on the board to the TA during checkout.