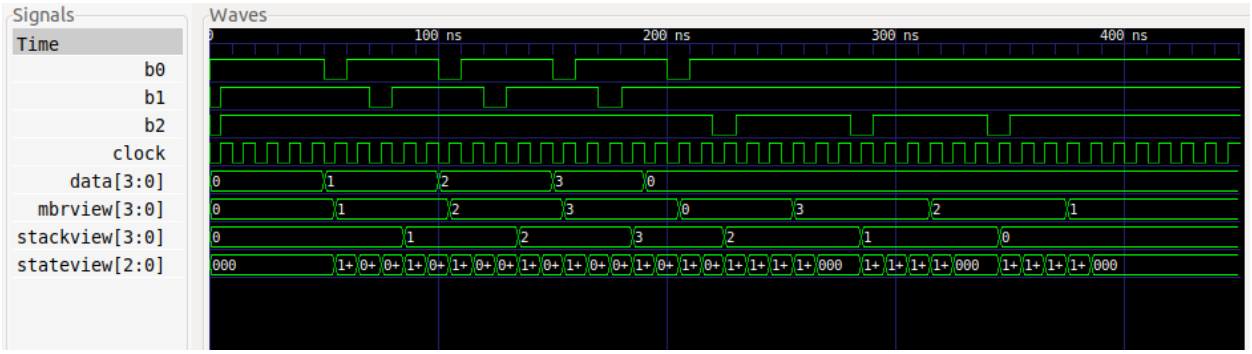
Lucy Barest
Prof. Li
Prof. Taylor
CS232

Project 6: Stack-based calculator

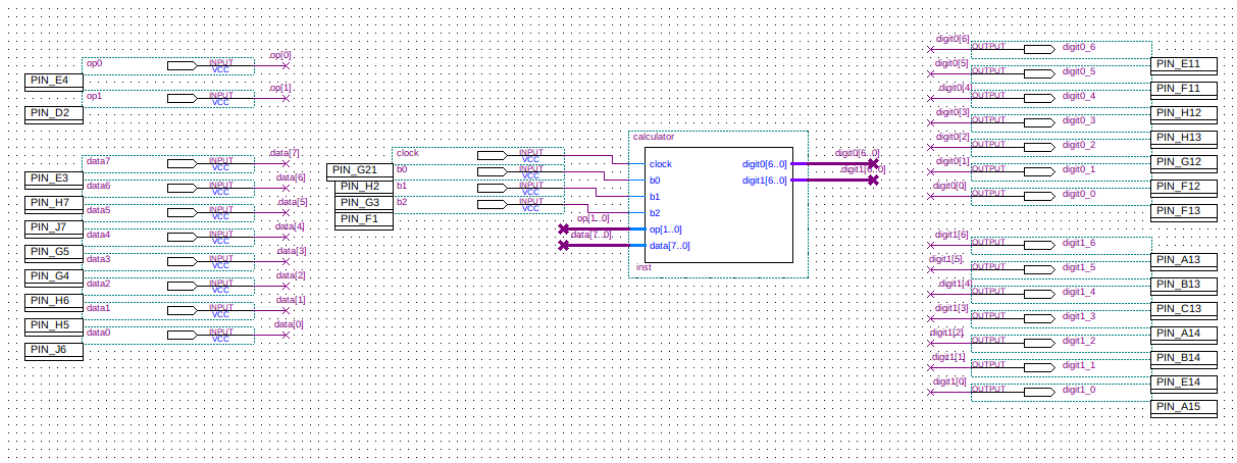
Task: The goal of this project was to develop a stack-based four-function calculator utilizing RAM as a fundamental component. To operate the calculator, we employed three buttons to trigger functions, two switches to designate mathematical operations (+, -, *, /), and eight switches to define numerical values. Button 0 (capture) transfers the current switch settings to the MBR (Memory Buffer Register). Button 1 (enter) pushes the content of the MBR onto the stack. Button 2, (action) carries out the operation indicated by the two operation switches, utilizing both the MBR and the initial stack value, and updating the MBR with the result.

Image 1: gtkwave from Lab



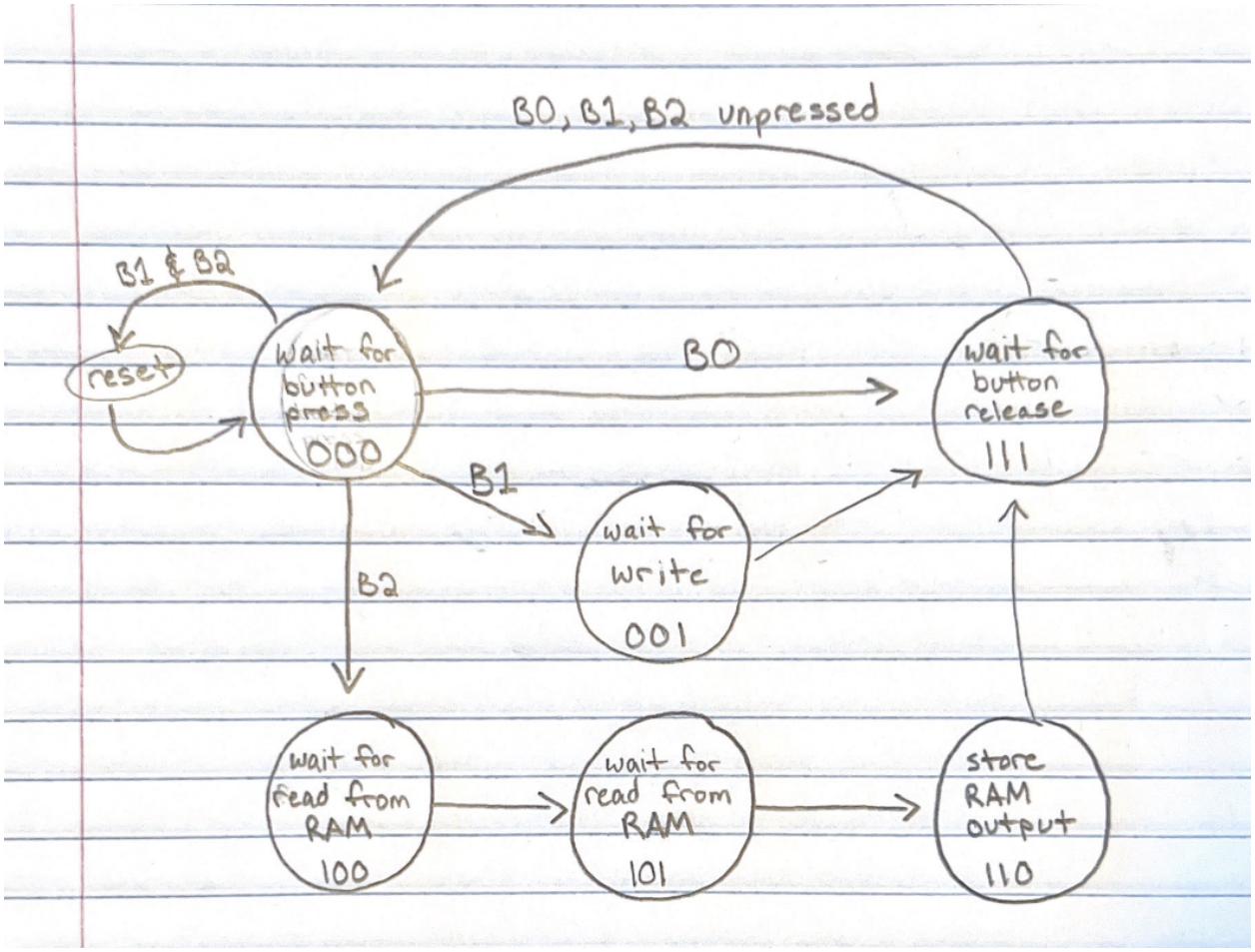
The screenshot above is the GHDL output of my stacker.vhdl file from the lab. All of the outputs correspond to the given GHDL file, so I know that my stacker works.

Image 2: Top level entity (calc_circuit.bdf)



This circuit connects all of the inputs and outputs to form the finished calculator by using a block of calculator.vhd, which uses RAM to implement a stack-based four-function calculator.

Image 3: State machine diagram



The main concept behind this state machine lies in its initial waiting state (000). Depending on the button pressed, it transitions to different states. If Button 0 is activated, it captures the current switch values and stores them in the MBR, proceeding to the alternate waiting state (111), where it remains until all buttons are released, before returning to the original waiting state (000).

While in state 000, pressing Button 1 results in the MBR's current content being pushed onto the stack, and the "write enable" signal is activated to permit RAM writing. It then transitions to state 001, waits for the write process, and subsequently moves to state 111 and back to 000.

When in state 000, pressing Button 2 shifts the stack pointer backward by one position. It then advances to states 100, 101, and 110. The inclusion of states 100 and 101 allows for a 2-clock cycle delay to ensure that all elements update before operations can occur. In state 110, the circuit executes operations between RAM_output and the MBR, storing the result in the MBR, and subsequently transitions to state 111 and back to 000.

If, at any point, both B1 and B2 are simultaneously pressed, the stack pointer, MBR, RAM_input, and RAM_we are all reset to 0, and the state is reverted to 000.

Circuit in action: (videos in zip file)

Video	Add_Sub.mov	Mult_Div.mov	complex.mov
Expression	8 + 4 = 12 8 - 2 = 6	2 * 3 = 6 8 / 4 = 2	3 + (6 / (3 * 1)) = 5
Operations	mbr = 8 push MBR mbr = 4 pop 8 → 8+4 = 12 B1&B2 = reset mbr = 8 push MBR mbr = 2 Pop 8 → 8-MBR = 6	mbr = 2 push MBR mbr = 3 pop 2 → 2*3 = 6 B1&B2 = reset mbr = 8 push MBR mbr = 4 Pop 8 → 8/MBR = 2	mbr = 3 push MBR mbr = 6 push MBR mbr = 3 push MBR mbr = 1 pop 3 → MBR = 3*MBR = 3*1 = 3 pop 6 → MBR = 6/MBR = 6/3 = 2 pop 3 → MBR = 3+MBR = 3+2 = 5

The video ‘Add_Sub’ shows the result of adding 8 + 4 = 12, resetting the circuit by pressing B1 and B2, and then subtracting 6-2=4.

The video ‘Mult_Div’ shows the result of multiplying 2 * 3 = 6, resetting the circuit by pressing B1 and B2, and then dividing 8 / 4 = 2.

The video ‘compex’ shows the result of the complex expression: 3 + (6 / (3 * 1)) = 5

Acknowledgements:
Shelby Roman
Maddie Puzon
Alex Solano
Desmond Frimpong
Samuel Atilano