## Problem Definition

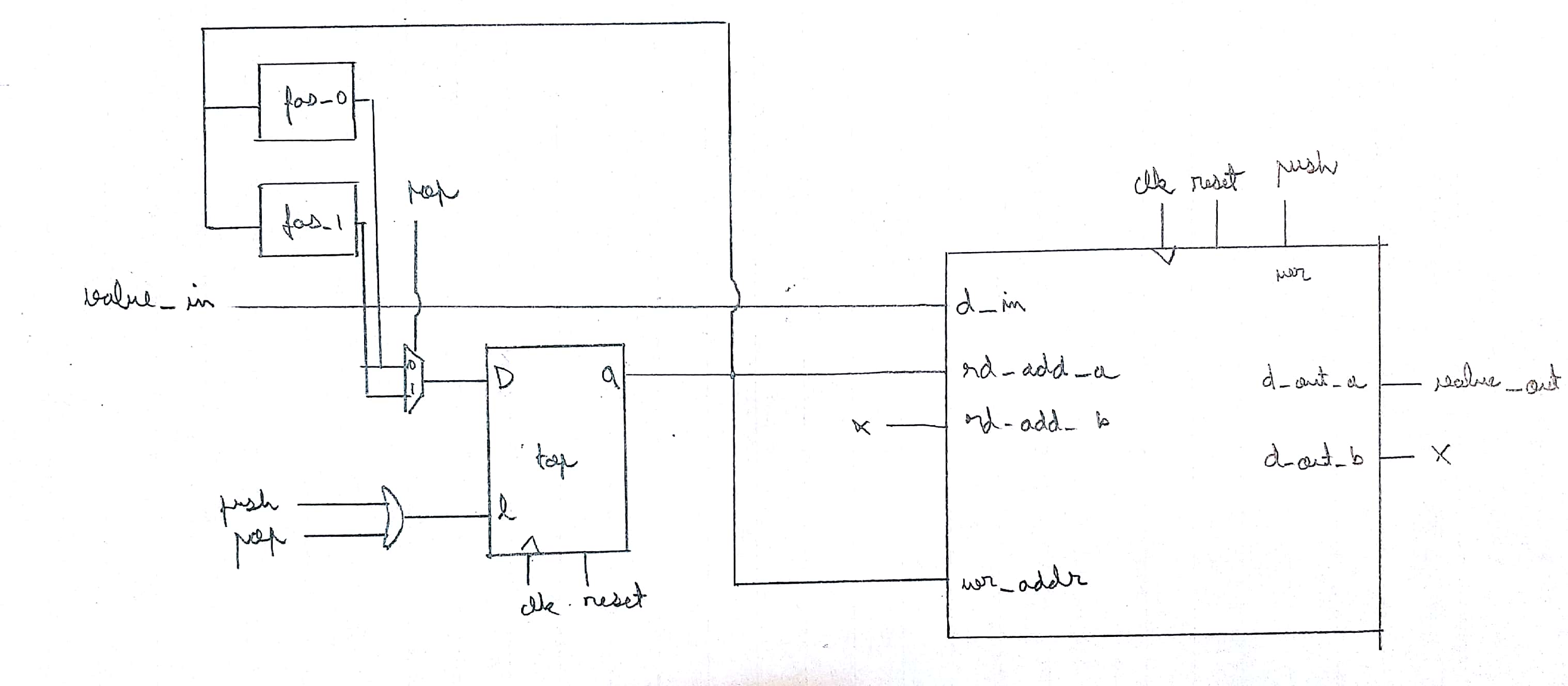
Implement a stack using Verilog. Inputs as push, pop, value\_in and outputs as underflow, overflow and value\_out.

## Implementation Details

The stack has been achieved using a register file of depth 8 and width 16 to store 16 bit numbers, a register of 3 bits constructed using 3 dfrls, a full adder/subtractor of 3 bits for the address and wires.

The abstract solution is to generate both, incremented and decremented address based on the current top pointer address and then assigning it based on a push or pop operation. The register file is instantiated once with all necessary values. The output 16 bits is the number stored at the top address.

## Circuit Diagram



## Modules used from Library File

* or2: Generate write enable signal for register file.
* fa: Used to construct a 3 bit full adder/subtractor.
* mux2: Used to select which 3 bit address (increment or decrement) to choose for instantiation of the register file.
* mux8: In the register file, it is used to select the address from which the read needs to be performed.
* dfrl: Register and register file constructed using dfrls.

## Modules defined in Project File

* reg\_3: Used to store the 3 bit address of top
* mux2\_3: Used to select the 3 bit address amongst 2 addresses using one select line to write to another 3 bit address
* fas3: Used to generate the incremented and decremented address
* reg\_file: Used to store 8, 16 bit numbers as part of the stack

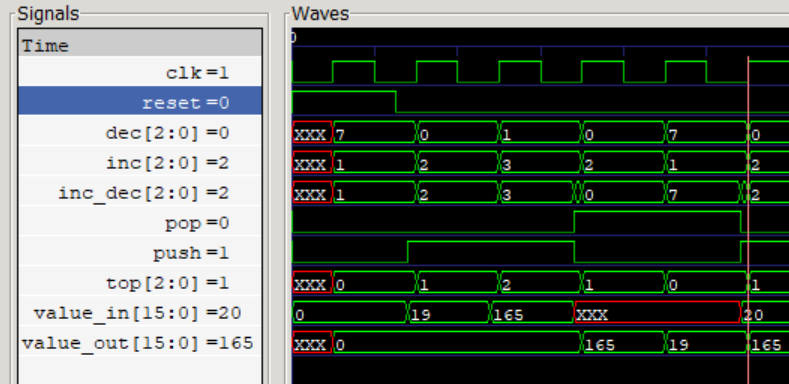
## Progress

The entire implementation of the stack is completed. Following is a snippet of the waveforms obtained on 2 push operations followed by 2 pop operations. OVERFLOW conditions are not handled as of yet.

The logic is as follows.

1. Generate write enable if pop or push is high. This is because these are the conditions under which you write to the register file.
2. Generate the incremented and decremented addresses using a full adder/subtractor.
3. Choose the address to be finally updated onto the register holding top based on the values of push and pop.
4. Write this chosen value to the said register storing top.
5. Instantiate the register file with one read operation of the top address (push this output to value\_out) and one write operation with the value\_out.

See issues for more details.



## Code Snippets

Following is the main stack instantiation with the inputs and outputs.

module stack(input wire clk, reset, push, pop, input wire [15:0] value\_in, output wire [15:0] value\_out);

wire [2:0] top, inc\_temp, dec\_temp, inc\_dec, inc, dec;

wire carry\_a, carry\_s, is7, is0;

wire push\_or\_pop;

wire [15:0] d\_care;

or2 o0(push, pop, push\_or\_pop);

fas3 a(top, 3'b001, 1'b0, inc, carry\_a);

fas3 b(top, 3'b001, 1'b1, dec, carry\_s);

mux2\_3 m23\_2(pop, inc, dec, inc\_dec);

reg\_3 r3(clk, reset, push\_or\_pop, inc\_dec, top);

reg\_file rf(clk, reset, push, top, 3'bxxx, top, value\_in, value\_out, d\_care);

endmodule

## Issues

UNDERFLOW conditions are implicitly handled by the current definition. OVERFLOW conditions need to be implemented. The plan is to check if the incremented address is 3’b111 and then fail to reset it to 3’b000 by using a mux. The additions that must be implemented in the current code are the generation of underflow and overflow signals.