

MSCE6323/ENCS5399: Verification and Validation of Hardware

Assignment#2 Reference Model by using System Verilog

**Prepared by:**

Zainab Shaabneh\_1182820

**Instructor**:

Dr. Ayman Hroub.

**Date:**

19 \ 1 \ 2023

* Create a new System Verilog module and name it appropriately "CCLU"
* Declare the input and output ports as described in the CCLU specifications.

input loop;

input [31:0] address;

input [31:0] target;

input [31:0] counter;

input reset;

input clk;

output valid;

output isFull;

output error;

output [31:0] target;

* **Create a stack data structure to store the CCL instructions. This can be done using an array of a struct, with each struct containing the fields for the CCL's address and counter**.

typedef struct {

logic [31:0] address;

logic [31:0] counter;

} ccl\_t;

ccl\_t stack[10]; // stack size is 10

logic [31:0] top; // index of the top of the stack

* **then use this stack to push and pop CCL instructions, as well as retrieve the top element of the stack. For example, to push a new CCL instruction onto the stack, you can use the following code:**

// assume address and counter are input values

stack[top] = {address, counter};

top++;

* **To pop the top element of the stack:**

top--;

* **And to retrieve the top element of the stack**

ccl\_t top\_ccl = stack[top];

* **Implement the logic for handling the different input scenarios and conditions. This can be done using a combination of always blocks and if-else statements**

always\_comb begin

if(reset) begin // to empty the stack

top = 0;

for(int i=0; i<10; i++) begin

stack[i].address = 0;

stack[i].counter = 0;

end

end

else if(loop) begin //instruction that already exists on the stack

if(stack[top].address == address) begin

if (stack[top].counter == 1) begin

top--; // pop the top element of the stack

end

else begin

stack[top].counter--; // decrement the counter of the top element of the stack

end

target <= stack[top].address; // output the address of the top element of the stack

valid <= 1'b1;

end

else begin

if (counter==0) begin.

error <= 1'b1;

end else if (counter==1) begin

target <= target;

valid <= 1'b1;

end else if (top==9) begin

error <= 1'b1;

end else begin

top++;

stack[top].address <= address;

stack[top].counter <= counter-1;

target <= target;

valid <= 1'b1; end

end

end

* **Create an always block to handle the clock signal and update the stack and output signals accordingly.**

always\_ff @(posedge clk) begin

// code to different input scenarios and conditions

end