

**B V V SANGHA’S**

BASAVESHWAR ENGINEERING COLLEGE (AUTONOMOUS)

[An Institute under Technical Education Quality Improvement Programme (TEQIP) of world bank, Accredited by National Board of Accreditation of AICTE, Government Aided Institution and permanently Affiliated to Visvesvaraya Technological University, Belagavi]

# S. Nijalingappa Vidyanagar, BAGALKOTE-587102 Karnataka, India

**Department of Electronics and Communication Engineering**

**Academic Year 2022-23**

# Verilog project report on “Stack or LIFO using Verilog code”

**Under the Guidance of Prof. Mallikarjun.C. A**

**Project associates**

**Pooja M Babaleshwar 2BA20EC049**

**Sapna Yaranal 2BA20EC076**

**Department of Electronics and Communication Engineering**

This is to certify that this project phase report entitled **“Stack or LIFO using Verilog Code ”** submitted to Electronics and Communication Engineering Department**, B**asaveshwar Engineering College, Bagalkot, is a bonafide record of work done by, Pooja M Babaleshwar (2BA20EC049), Sapna Yaranal(2BA20EC076)

studying in V semester during the year 2021-2022under my supervision.

GUIDE:

Prof. M.C.Aralimarad

Assistant professor

BEC, Bagalkot

Place : Bagalkot

Date : 12/01/2023

### Stack or LIFO Verilog Code

The Last In First Out (LIFO) or Stack is a data arrangement structure in which the data that enters the last is the one that is removed first. Let us see how to implement the concept of Stack using Verilog.  
  
  
For a normal memory, we provide the address for reads and writes. But here we know where to write and what to read for stack operation. So address is not required to access a stack location.  
  
Procedure to implement stack:

1. Create a normal memory in Verilog.
2. When the data and push signal is given, write to the memory starting from first address.
3. When pop signal is given, read from the memory from last written address.
4. When stack becomes empty, assert empty and if it becomes full, assert full signal.

We just require one index variable (internal variable) to control a stack because we are writing to and reading from the same address. However this is not the case for [FIFO](https://electrobinary.blogspot.com/2020/05/synchronous-fifo.html).

**Verilog Code Logic:**

1. Use a variable index to point to the memory location to read or write.
2. During push signal, write to the memory and increment index.
3. During pop, read from the memory from last written memory index and decrement index.
4. Empty occurs when index equals 0.
5. Full occurs when index equals depth.

For empty and full, I have used  a shortcut logic to perform comparison as == operator will realize more hardware.

**Verilog Code:**

**module** Stack( clk,rstn, pop, push, empty,full,din,dout);

**parameter** **WIDTH** = **8**;

**parameter** **DEPTH** = **8**;

**input** clk;

**input** rstn;

**input** pop;

**input** push;

**input** [**WIDTH**-**1**:**0**]din;

**output** [**WIDTH**-**1**:**0**]dout;

**output** empty;

**output** full;

**reg** [**WIDTH**-**1**:**0**]stack[**DEPTH**-**1**:**0**]; //memory

**reg** [**WIDTH**-**1**:**0**]index, next\_index;

**reg** [**WIDTH**-**1**:**0**]dout, next\_dout;

**wire** empty, full;

**always** @ (**posedge** clk) //Sequential block

**begin**

**if**(!rstn)

**begin**

dout <= **8'd0**;

index <= **1'b0**;

**end**

**else**

**begin**

dout <= next\_dout;

index <= next\_index;

**end**

**end**

**assign** empty = !(|index);

**assign** full = !(|(index ^ **DEPTH**));

**always** @ (**\***) //Combinational Block

**begin**

**if**(push) //write

**begin**

stack[index] = din;

next\_index = index+**1'b1**;

**end**

**else** **if**(pop) //read

**begin**

next\_dout = stack[index-**1'b1**];

next\_index = index-**1'b1**;

**end**

**else**

**begin**

next\_dout = dout;

next\_index = index;

**end**

**end**

**endmodule**

Testbench:

**module** Stack\_tb;

// Inputs

**reg** clk;

**reg** rstn;

**reg** pop;

**reg** push;

**reg** [**7**:**0**] din;

// Outputs

**wire** empty;

**wire** full;

**wire** [**7**:**0**] dout;

// Instantiate the Unit Under Test (UUT)

Stack uut (

.clk(clk),

.rstn(rstn),

.pop(pop),

.push(push),

.empty(empty),

.full(full),

.din(din),

.dout(dout)

);

**always** #**5** clk = ~clk;

**task** reset(); //reset task

**begin**

clk = **1'b1**;

rstn = **1'b0**;

pop = **1'b0**;

push = **1'b0**;

din = **8'd0**;

#**30** rstn = **1'b1**;

**end**

**endtask**

**task** read\_stack(); //read task

**begin**

pop = **1'b1**;

#**10**

pop = **1'b0**;

**end**

**endtask**

**task** write\_stack([**7**:**0**]din\_tb); //write task

**begin**

push = **1'b1**;

din = din\_tb;

#**10** push = **1'b0**;

**end**

**endtask**

// Main code

**initial**

**begin**

reset();

#**10**;

**repeat**(**2**)

**begin**

write\_stack(**8'h11**);

#**10**;

write\_stack(**8'h22**);

#**10**;

write\_stack(**8'h33**);

#**10**;

write\_stack(**8'h44**);

#**40**;

**end**

read\_stack();

#**20**;

read\_stack();

#**20**;

write\_stack(**8'hAA**);

#**10**;

write\_stack(**8'hBB**);

#**40**;

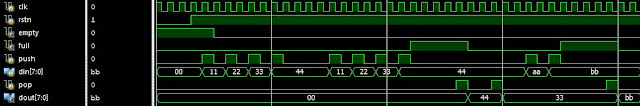
read\_stack();

#**20**;

$finish;

**end**

**endmodule**

**Simulation Result:**  
[](https://1.bp.blogspot.com/-nUTAoQlpKe0/XqxKPTHDW1I/AAAAAAAABfk/lX2D7-mliIcGJKYnX5f5mHSMnZVwYhHbgCLcBGAsYHQ/s1600/Stack_sim.JPG)  
  
  
  
  
  
From the simulation result,

1. We see that the Stack is initially empty as there is no data.
2. Data is pushed into it until it becomes full.
3. Two data are popped and then two data are pushed again to make it full again

Thus we have verified the basic operation of a stack.