ECSE 323 – Digital System Design
52-stack Card Deck
g39_lab3

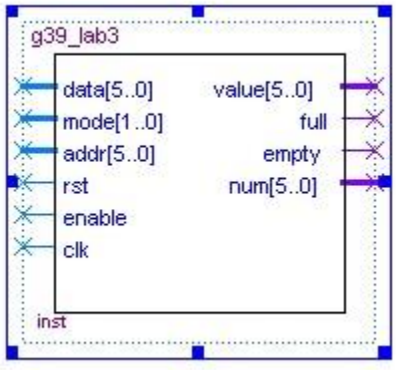
Description:

The stack52 circuit represents a card deck to be used in the card game described in Lab 5. The circuit has four modes: No Operation “NOP”, “PUSH”, “POP” and Initialize “INIT”. These modes, along with a reset control input “rst”, resemble a real card deck; starting with a new deck (INIT), starting with an empty hand (rst), drawing a card (POP) and adding a card to a player’s hand (PUSH).

The circuit has the following inputs/outputs:

data: 6-bit input representing a card to push
mode: 2-bit input representing the operation desired
addr: 6-bit input representing a location in the stack
enable: 1-bit input to control the circuit (enable/disable operations)
rst: 1-bit input to create an empty hand
clk: 1-bit input

value: 6-bit output representing the value at the input address
num: 6-bit output representing the number of cards
empty: 1-bit output a boolean value to indicate an empty stack
full: 1-bit output a boolean value to indicate a full stack



Implementation Details:

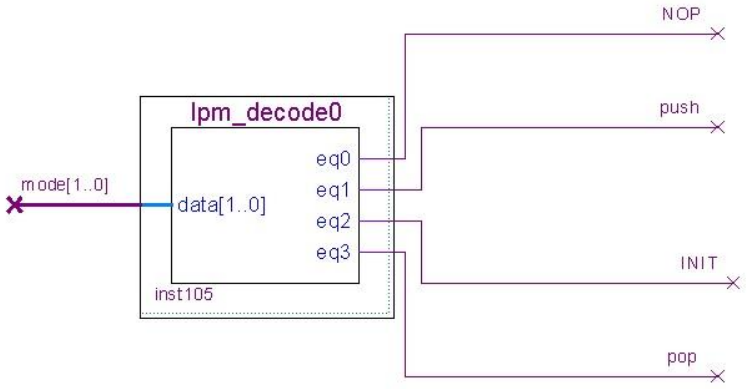
Sub-circuits:

I. Mode selection:

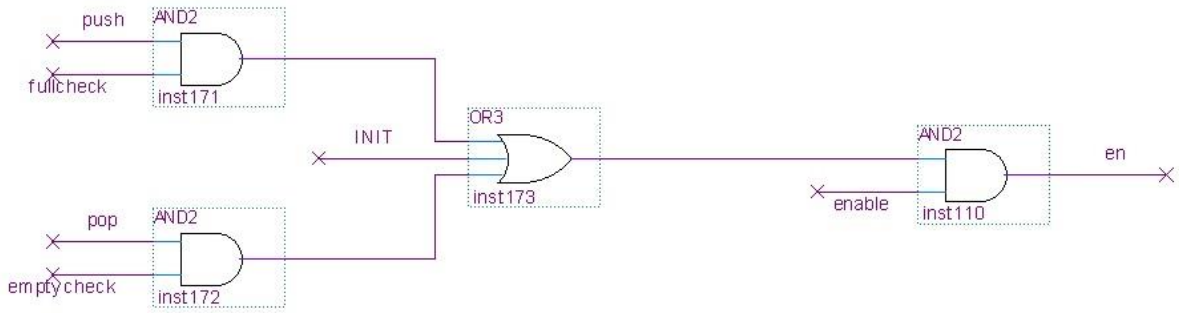
A decoder is used to select the mode.

mode[1]	mode[0]	NOP	PUSH	INIT	POP
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

As seen above, once an operation is selected, all other operations are ‘0’.



II. Enable sub-circuit:

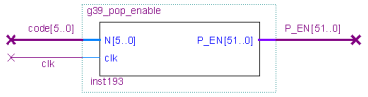


This enable circuit, together with the pop enable circuit, controls the enable input to the LPM_FFs. Inputs to this sub-circuit are 3 modes, the enable input, as well as 2 inputs known as fullcheck and emptycheck. The input fullcheck is equal to NOT full, while emptycheck is equal to NOT empty. All in all, what this circuit basically does is enable the circuit in 3 conditions:

1. Pushing (Push is high) while not full (full = 0) and the enable input is high
2. Popping (Pop is high) while not empty (empty = 0) and the enable input is high
3. Initializing the stack (INIT is high)

III. POP_EN sub-circuit:

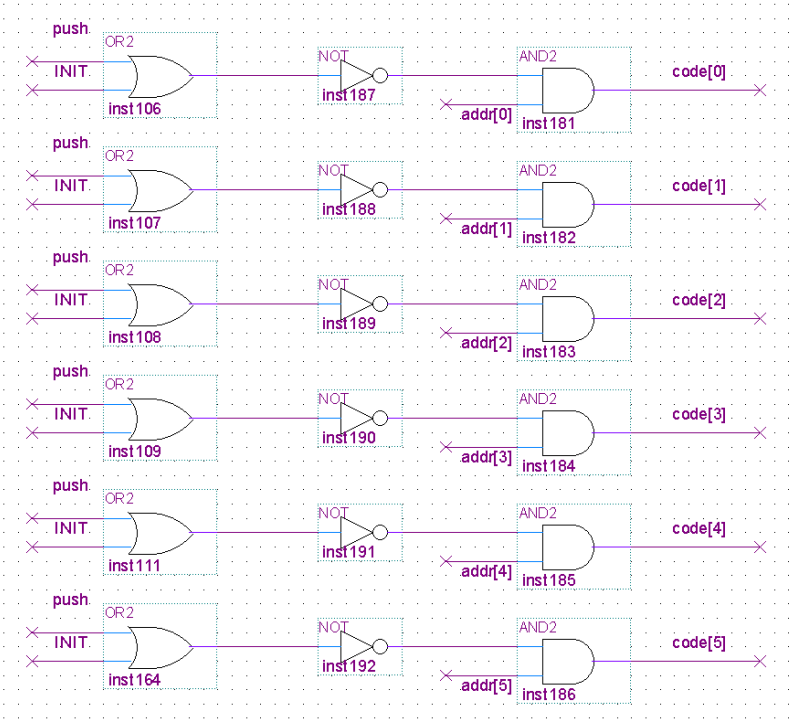
```
-- this circuit implements pop enable circuit
--
-- entity name: g39_RANDU
--
-- Copyright (C) 2017 Saleh Bakhit, Adham El-Khouly
-- Version 1.0
-- Author: Saleh Bakhit (saleh.bakhit@mail.mcgill.ca), Adham El-Khouly (adham.el-khouly@mail.mcgill.ca)
-- Date: 13/02/2017

LIBRARY lpm;
library ieee;
USE lpm.lpm_components.all;
USE ieee.std_logic_1164.all;



entity g39_pop_enable is
port ( N : in std_logic_vector(5 downto 0);
P_EN : out std_logic_vector(51 downto 0);
clk : in std_logic);
end g39_pop_enable;

architecture behavior_pop_enable of g39_pop_enable is


Begin
crc_table : lpm_rom -- use the altera rom library macrocell
GENERIC MAP(
lpm_widthad => 6, -- sets the width of the ROM address bus
lpm_numwords => 64, -- sets the number of words stored in the ROM
lpm_outdata => "UNREGISTERED", -- no register on the output
lpm_address_control => "REGISTERED", -- register on the input
lpm_file => "g39_LUTcontent.mif", -- the ascii file containing the ROM data
lpm_width => 52) -- the width of the word stored in each ROM location
PORT MAP(inclock => clk, address => N, q => P_EN);


end behavior_pop_enable;
```




```
-- Quartus II generated Memory Initialization File (.mif)

WIDTH=52;
DEPTH=64;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
        0   :   1111111111111111111111111111111111111111111111111111;
        1   :   1111111111111111111111111111111111111111111111111110;
        2   :   1111111111111111111111111111111111111111111111111100;
        3   :   1111111111111111111111111111111111111111111111111000;
        4   :   1111111111111111111111111111111111111111111111110000;
        5   :   1111111111111111111111111111111111111111111111100000;
        6   :   1111111111111111111111111111111111111111111111000000;
        7   :   1111111111111111111111111111111111111111111110000000;
        8   :   1111111111111111111111111111111111111111111100000000;
        9   :   1111111111111111111111111111111111111111111000000000;
        10  :   1111111111111111111111111111111111111111110000000000;
        11  :   1111111111111111111111111111111111111111100000000000;
        12  :   1111111111111111111111111111111111111111000000000000;
        13  :   1111111111111111111111111111111111111110000000000000;
        14  :   1111111111111111111111111111111111111100000000000000;
        15  :   1111111111111111111111111111111111111000000000000000;
        16  :   1111111111111111111111111111111111110000000000000000;
        17  :   1111111111111111111111111111111111100000000000000000;
        18  :   1111111111111111111111111111111111000000000000000000;
        19  :   1111111111111111111111111111111110000000000000000000;
        20  :   1111111111111111111111111111111100000000000000000000;
        21  :   1111111111111111111111111111111000000000000000000000;
        22  :   1111111111111111111111111111110000000000000000000000;
        23  :   1111111111111111111111111111100000000000000000000000;
        24  :   1111111111111111111111111111000000000000000000000000;
        25  :   1111111111111111111111111110000000000000000000000000;
        26  :   1111111111111111111111111100000000000000000000000000;
        27  :   1111111111111111111111111000000000000000000000000000;
        28  :   1111111111111111111111110000000000000000000000000000;
        29  :   1111111111111111111111100000000000000000000000000000;
        30  :   1111111111111111111111000000000000000000000000000000;
        31  :   1111111111111111111110000000000000000000000000000000;
        32  :   1111111111111111111100000000000000000000000000000000;
        33  :   1111111111111111111000000000000000000000000000000000;
        34  :   1111111111111111110000000000000000000000000000000000;
        35  :   1111111111111111100000000000000000000000000000000000;
        36  :   1111111111111111000000000000000000000000000000000000;
        37  :   1111111111111110000000000000000000000000000000000000;
        38  :   1111111111111100000000000000000000000000000000000000;
        39  :   1111111111111000000000000000000000000000000000000000;
        40  :   1111111111110000000000000000000000000000000000000000;
        41  :   1111111111100000000000000000000000000000000000000000;
        42  :   1111111111000000000000000000000000000000000000000000;
        43  :   1111111110000000000000000000000000000000000000000000;
        44  :   1111111100000000000000000000000000000000000000000000;
        45  :   1111111000000000000000000000000000000000000000000000;
        46  :   1111110000000000000000000000000000000000000000000000;
        47  :   1111100000000000000000000000000000000000000000000000;
        48  :   1111000000000000000000000000000000000000000000000000;
        49  :   1110000000000000000000000000000000000000000000000000;
        50  :   1100000000000000000000000000000000000000000000000000;
        51  :   1000000000000000000000000000000000000000000000000000;
        [52..63]  :   0000000000000000000000000000000000000000000000000000;
END;
```

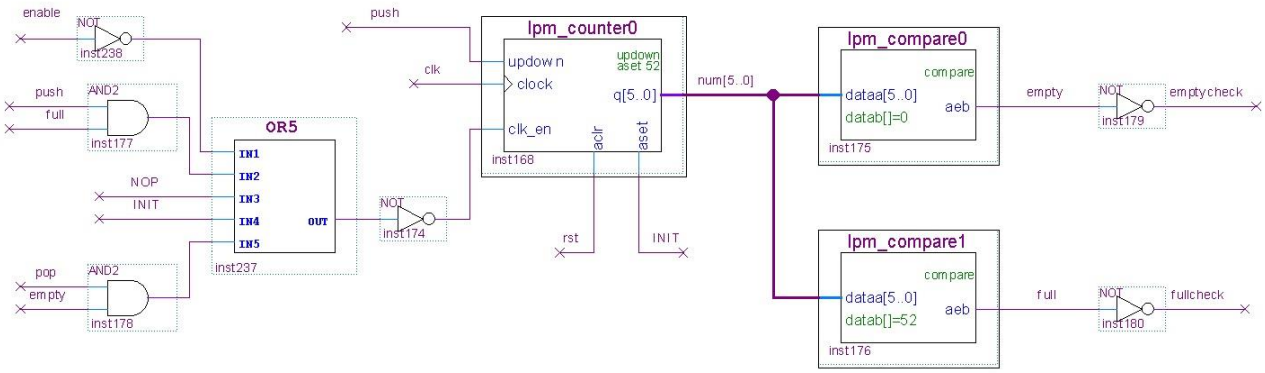
This pop enable circuit, together with the enable circuit explained above control the enable input to the LPM_FFs. The circuit is implemented using a ROM. It takes in a 6 bit address and output a 52-bit output. The output bits are set such that for a given address (addr[5..0]), P_EN(0) through P_EN(addr-1) are zero and P_EN(addr) through P_EN(51) are one. For addresses, greater than 51, all output bits are zero. This is shown in the .mif file provided above. For example, when the addr = 3, the output of the pop enable circuit is ‘0’ for the 3 least significant bits and ‘1’ for everything else. The output of the Pop Enable circuit and the Enable circuit are ANDed at each enable LPM_FF and will be shown in further screenshots.

IV. Modified address sub-circuit (for Pop Enable circuit):



This circuit's main function is to provide the input to the Pop Enable circuit. Whenever the user is not pushing (push = 0) nor initializing (INIT =0) the code, which is the output of this sub-circuit, is equal to the address. However, when pushing (push = 1), or initializing (INIT =1) the code is equal to "000000". This is done so that during the push operation, the input: data[5..0] would be pushed at the top of the stack. While when initializing, this ensures that all the LPM_FFs are enabled as well. The output is then used as an input to the Pop Enable circuit, and will be explained further when explaining that circuit.

V. Number of Cards and status of stack sub-circuit:



- When mode is INIT, “aset” input to the LPM_counter0 is 1 and so it sets num to 52.
- When rst is ‘1’, “aset” input to the LPM_counter0 is 1 and so it sets num to 0.
- Otherwise:
  1. If PUSH and NOT full, increment num.
  2. If POP and NOT empty, decrement num.
- Lastly, there are have 5 conditions where the incrementing/decrementing should be disabled:
  1. PUSH and full
  2. POP and empty
  3. enable = ‘0’
  4. NOP
  5. INIT

  If any of these is 1, output of OR5 is 1 and clk_en is 0.

  A better way of implementing clk_en would be to set it to ‘1’ when we want to enable the counter instead of setting it to ‘0’ when disabling it.

  So clk_en = PUSH*(NOT)full + POP*(NOT)empty is more efficient than our implementation.

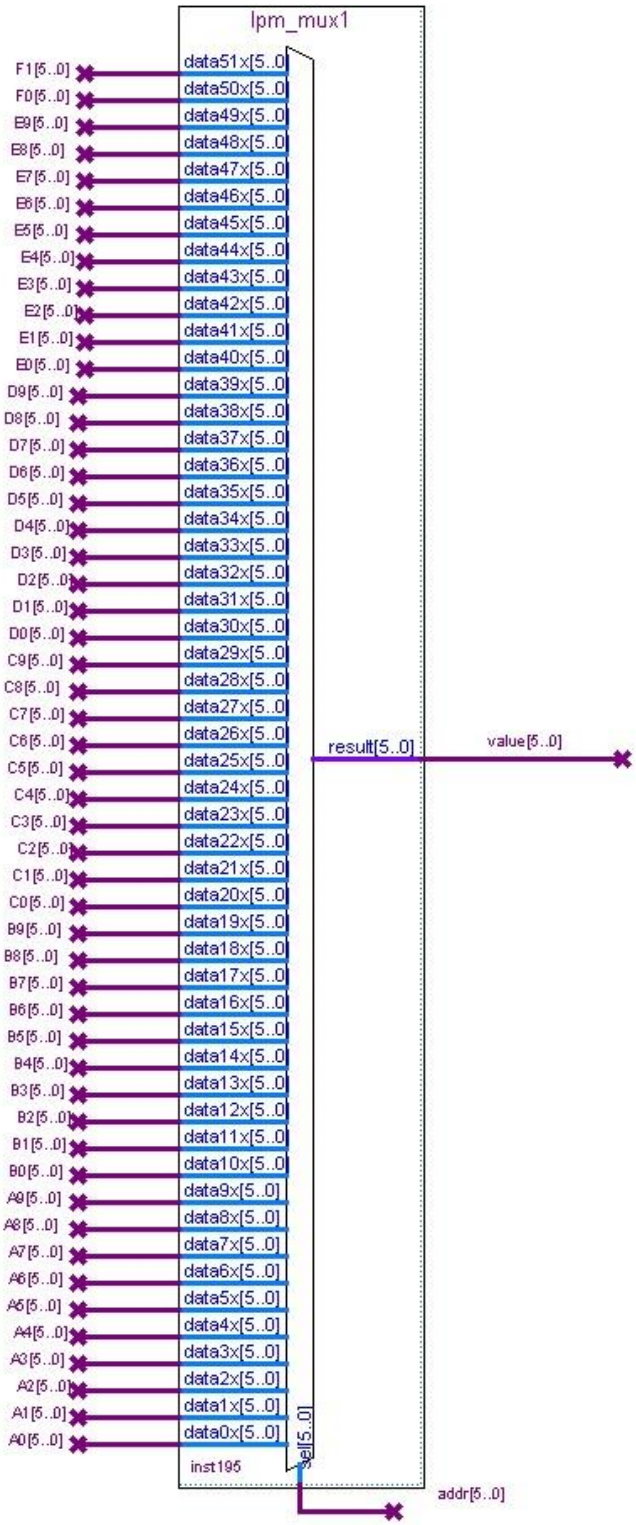
To determine the status of the stack num is compared with 0 and 52.

- If num = 0, empty = 1
- If num = 52, full = 1

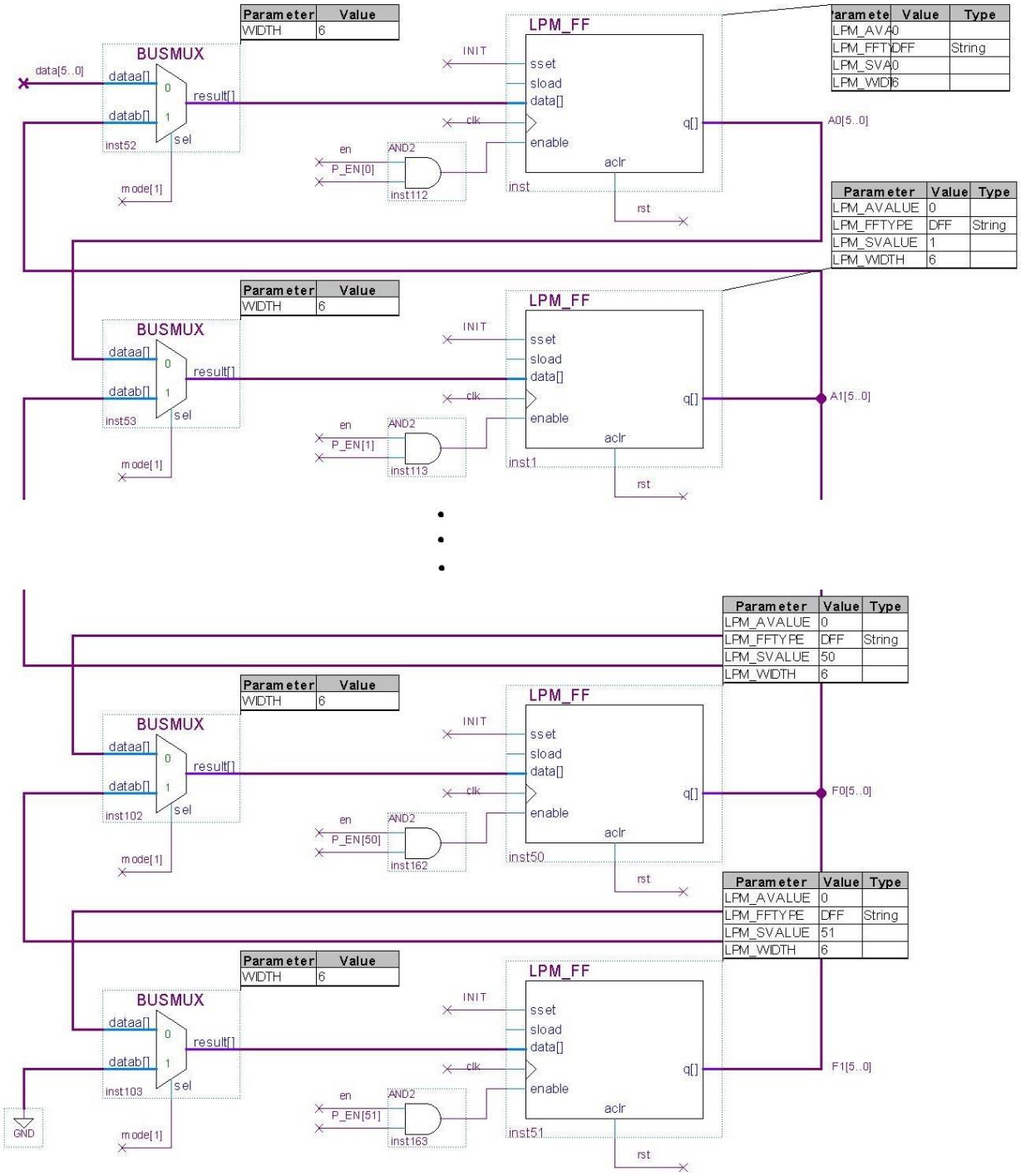
emptycheck = (NOT)empty and fullcheck = (NOT)full are added for convenience only.

VI. Outputting value:

An LPM_MUX with 52 data inputs - 6-bits wide – is used to display the value at the input address. So, select is takes the address as an input to output the corresponding value.



Main circuit:



This is the main circuit of this design. It utilizes 52 BUSMUXs and 52 LPM_FFs. The selection line for the BUSMUX is mode[1]. And as explained in the decoder, this would mean that it would select dataa[] in case of NOP & push, and datab[] in case of INIT and pop. However, it should be mentioned that in case of NOP the LPM_FF is NOT enabled, and in the case of INIT, sset has higher priority than holding the input. Therefore, what is selected only matters in case of Push and Pop. Explained below is what happens when pushing (assuming stack is not full):

- Mode[1] is '0', therefore data[5..0] is selected and put on the line
- en = '1' and P_EN = "1111…11", meaning all the LPM_FFs are enabled
- Therefore data[5..0] is held at A0[5..0] and everything else is pushed downwards

In case of Pop:

- Mode[1] is '1', and datab[] is selected
- en = '1' and P_EN generates the output depending on the address given as explained in the pop enable description. For the sake of this example, assume address is "000000". This would set P_EN = "111…11", meaning all the LPM_FFs are enabled
- A0[5..0] becomes A1[5..0], and so forth. That means that everything is shifted upwards

In case of INIT:

- sset = '1'
- en = '1' and P_EN = "111…11", meaning all the LPM_FFs are enabled
- A0[5..0] would take the LPM_SVALUE = "000000", A1[5..0] would take the LPM_SVALUE = "000001"

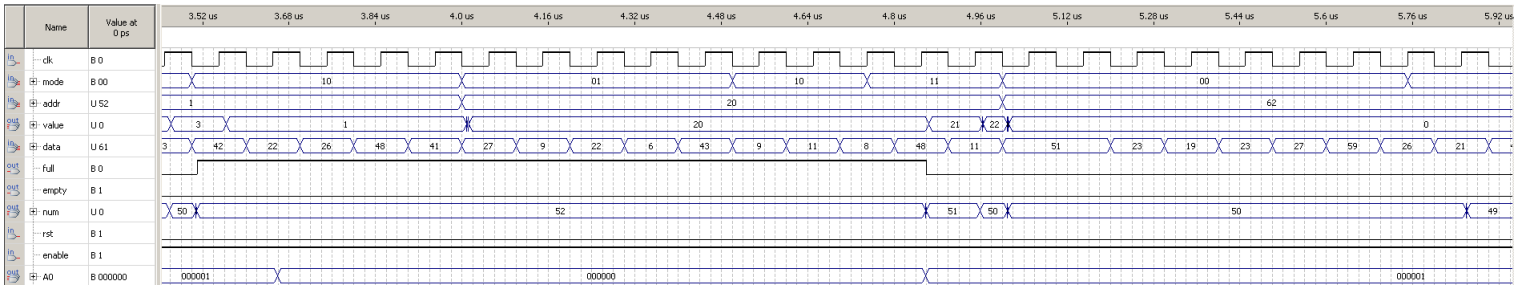
In case of NOP:

- en = '0'
- Therefore nothing happens

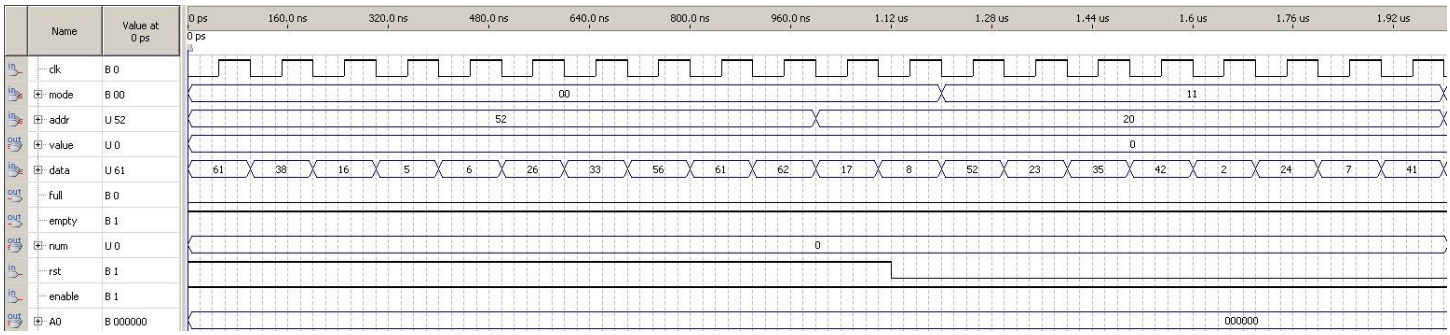
It is also worth mentioning that when rst is high, it would reset all the LPM_FFs to "000000".

Testing:

The simulation below showcases the following: INIT, PUSH*full and POP



The simulation below showcases POP*empty



The simulation below showcases PUSH