VLSI

PROGRAMMABLE BINARY TREE COMPUTATION

CHIP NAME: PBTCKS

Samir Silbak[1] and Manasa Kasula[2]

[1]silbaksr@mail.uc.edu
[2]kasulama@mail.uc.edu

[1](513) 207-0687
[2](847) 612-7364

November 12, 2013

Contents

List of Figures

List of Tables

Listings

1 Pinout Diagram

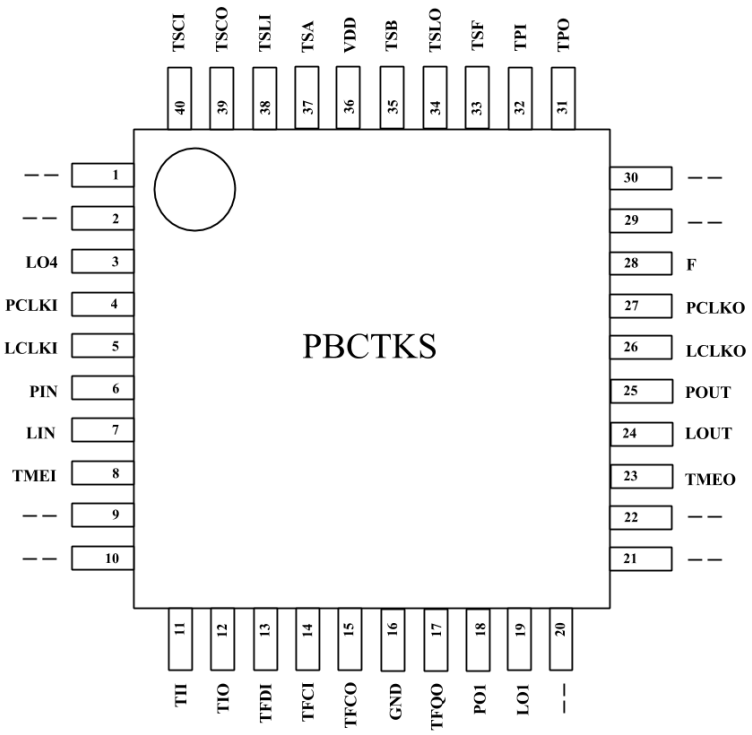

Figure 1: Pinout Diagram of the PBTCKS Chip

1.1 Pinout Description

Function	Pin #	I/O	Description
–	1	–	–
–	2	–	–
LO4	3	–	Last 16 Array Bit LUT Ouptut
PCLKI	4	I	P Shift Register Clock Input
LCLKI	5	I	LUT Shift Register Clock Input
PIN	6	I	P Shift Register Input
LIN	7	I	LUT Shift Register Input
TMEI	8	I	Test Mode Enabled Input
–	9	–	–
–	10	–	–
TII	11	I	Test Inverter Input
TIO	12	O	Test Inverter Output
TFDI	13	I	Test Flip-Flop D Input
TFCI	14	I	Test Flip-Flop Clock Input
TFCO	15	O	Test Flip-Flop Clock Output
GND	16	–	Ground Reference for I/O Pins
TFQO	17	O	Test Flip-Flop Q Output
PO1	18	–	First 8 Bit P Array Output
LO1	19	–	First 16 Bit LUT Array Output
–	20	–	–
–	21	–	–
–	22	–	–
TMEO	23	O	Test Mode Enabled Output
LOUT	24	O	P Shift Register Output of P input
POUT	25	O	P Shift Register Output of LUT
PCLKO	26	O	LUT Shift Register Clock Output
PCLKO	27	O	P Shift Register Clock Output
F	28	O	Output of Computation of LUT
–	29	–	–
–	30	–	–
TPO	31	O	Test P Slice Output
TPI	32	I	TesT P Slice Input
TSF	33	O	Test LUT Slice Mux Output
TSLO	34	O	Test LUT Slice Shift Register Output
TSB	35	I	Test LUT Slice B Address Input
VDD	36	I	Test LUT Slice B Input
TSA	37	I	Test LUT Slice A Address Input
TSLI	38	I	Test LUT Slice Shift Register Input
TSCO	39	O	Test LUT Slice Clock Output
TSCI	40	I	Test LUT and P SLice Clock Input

Table 1: Pinout Description

2 Explaination of Chip Function

The main functionality of the chip is to be able to take N-bit inputs and compute the combinational logic among all N-bit inputs. Each node in the binary tree is what constitutes the combinational function of two inputs. Each node is described to be a 4-bit Look Up Table (LUT). The LUT has the function of any combinational function the user wants to perform. For example, for an AND gate, the user must shift in "0001" into the LUT. Each LUT of all the nodes together create the "program" which are all cascaded together to perform a shift register. The binary tree accepts the input P and produces only a single bit output F. For example, if the user has 8 different inputs, an example of the function can be described as so: (A or B) or (C or D) or (E or F) or (G or H). Here we can see that we have 8 different inputs performing 7 different functions (in this case each function is the same) and only one single output, F. P is defined to be twice the number of leaf nodes in the binary tree, and the input is shifted in serially using a shift register. In order to "program" the LUTs with the specific functions, these also must be shifted in serially using a shift register. Therefore, going back to the case where we have 8 different inputs and 7 different LUTs, we can see that we will have to shift in 28 (7*4) bits into the LUT shift register. Each set of LUT outputs are connected to a 2:1 multiplexer to choose the output of the function. What this means is that the inputs can be thought of the address line into the LUT–what ever value happens to be stored is the result of that computation. For example, the figure below shows the truth table for an AND gate, we see inputs A and B, these are the address lines into the LUT. Therefore, we see that the output value of the LUT can only be 1 in the last row of the truth table, this method is very efficient and in fact this is how FPGAs/ASICs work, they use LUTs to perform these combinational functions.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: AND Gate Truth Table

2.1 Configuration of Chip

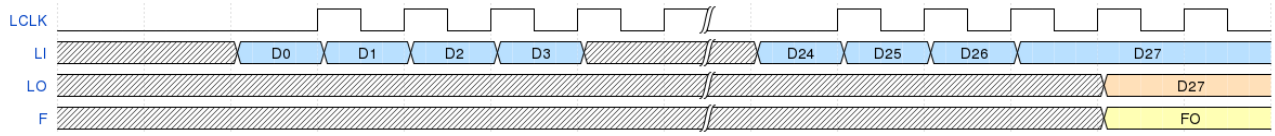

Figure 2: Cycle Timing Diagram for LUT Shift Register

Here we can see that the only thing the user needs to do is shift in the "function" wanted among the inputs. Since the data bits get loaded in from MSB to LSB, the user must input the function in reversed order. For example, if we want to perform an AND operation, we have to feed the data bits in like so: 1000.

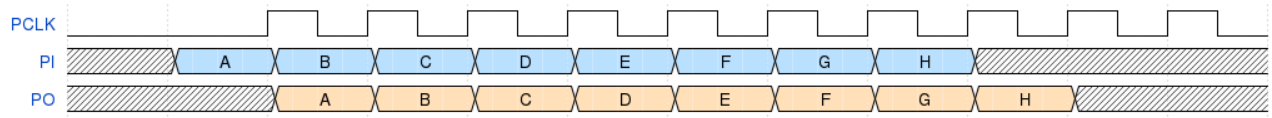

Figure 3: Cycle Timing Diagram for P Input Shift Register

As described from above the user has to just feed in input P, but in reversed order. Note that in test mode, the user has to just toggle the TMEI high.

3 Inclusion and Explanation of the Test Mode

In order to enable the test mode, the user must set the TMEI pin high. In doing so, we will bypass the last output of P and feed it into the shift register input of the LUT. This will connect all the flip flops together, and we can perform our scan chain to make sure the inputs are being shifted the way they are supposed to be. We will be able to monitor the output on the TMEO pin. Since only one clock line is required, we bypass the LUT clock line by just hooking up the clock line of the P shift register. If test mode is disabled however, then the circuit will perform back to its original function.

4 Major Design Decisions

The first thing that needed to be accomplished was to assemble the bit-slice design with minimal hardware and hardware that was supported by the library given to us. For example, our LUT bit slice uses three 2:1 Multiplexers, we could have used a 4:1 multiplexer, but having done so, our design would have been more complex when designing our Magic layouts.

We also wanted to make sure that the wiring would not be too complex between each LUT slice. Since this is a binary tree computation, we made sure to hook up the hardware in that fashion as it made it much easier to visualize how this needed to be connected while keeping in mind that we must connect each one to achieve the desirable 50MHz clock frequency.

5 Block Diagrams

5.1 Top Level Diagram

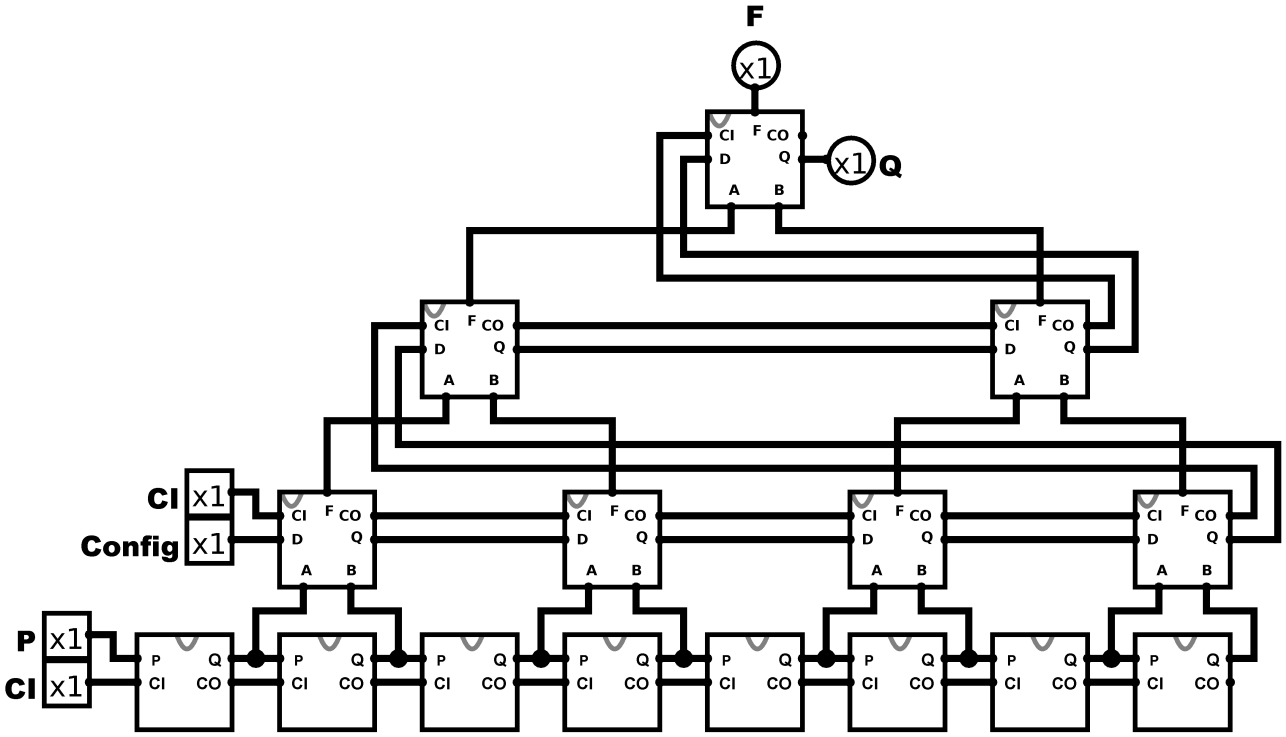

Figure 4: First Top Level Diagram (8 Inputs, 28-bit slice)

5.2 Top Level Diagram: LUT Slice and Shift Slice

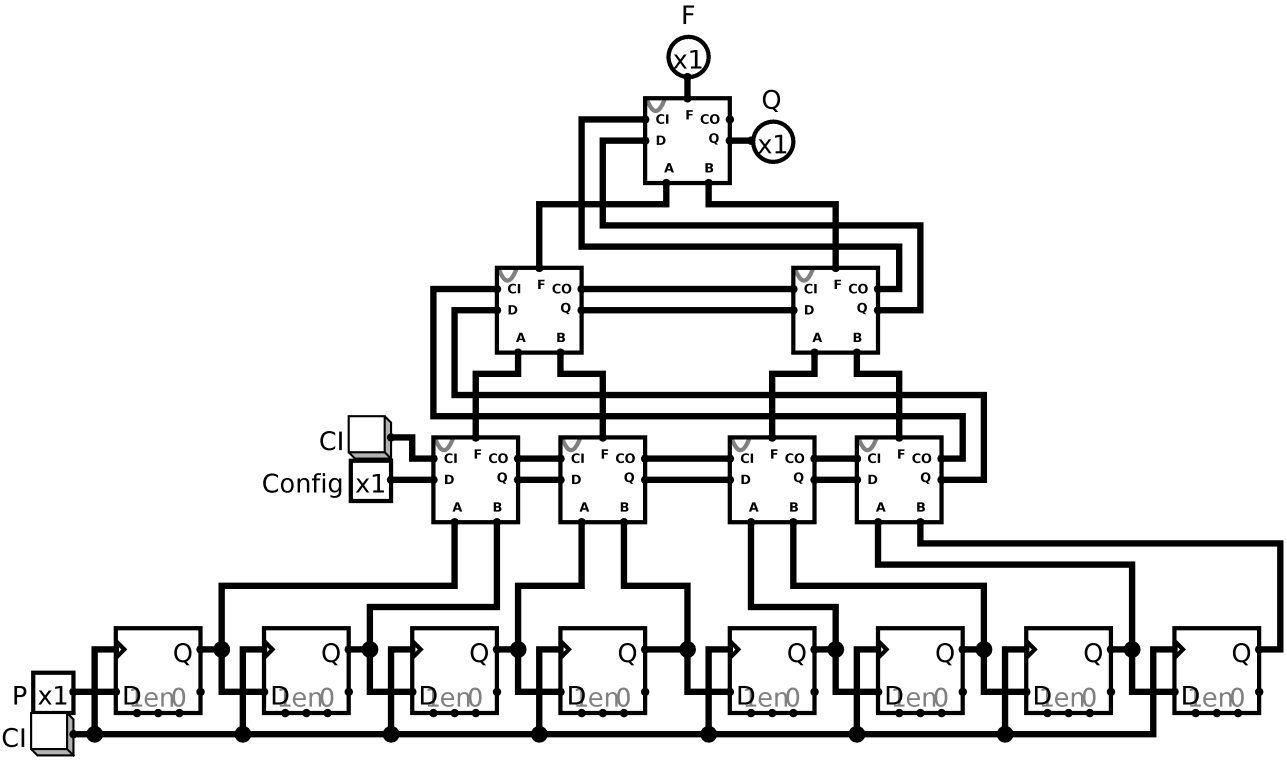

Figure 5: Top Level Diagram (8 Inputs, 28-bit slice)

We can see that the slice design of the P shift register is just made up of D-flip flops as shown in the above top level diagram for input P. Here we can see that we have P being twice the leaf nodes (8 in this case) and the LUTs are all cascaded together giving us a total width vector of 28 bits.

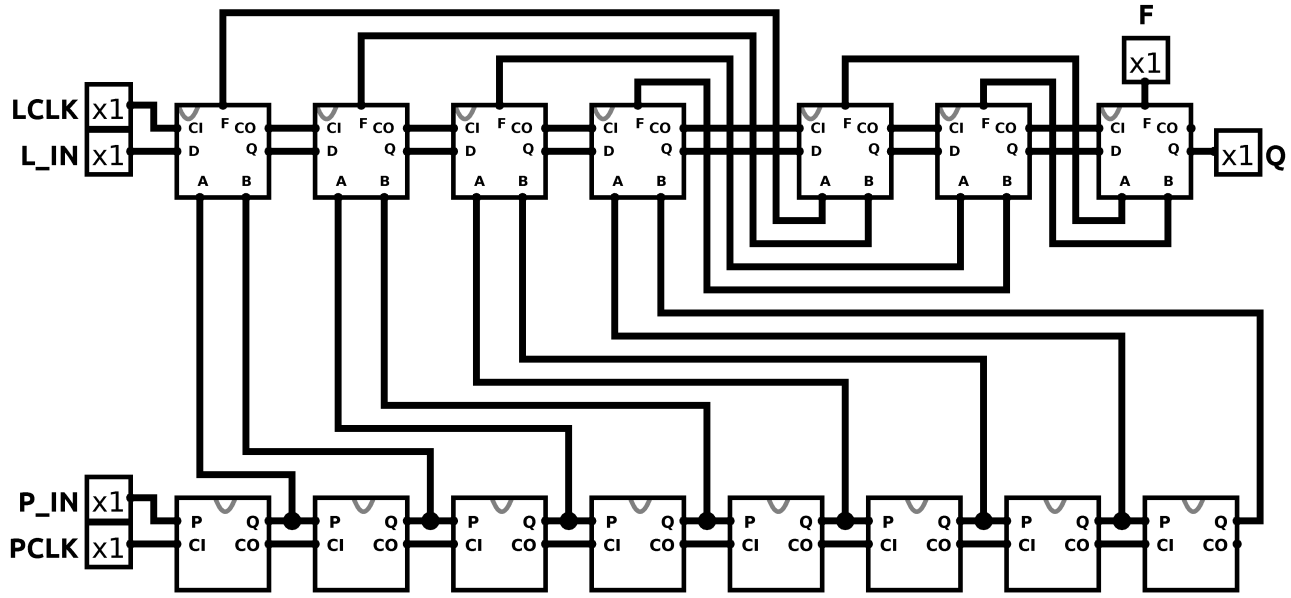

Figure 6: New Designed Top Level Diagram (8 Inputs, 28-bit slice)

We came up with a new design of how we will lay the chip out. This new design (more explained in major design decisions) takes full advantage of the total area given to us.

5.3 Bit Slice Design Scheme

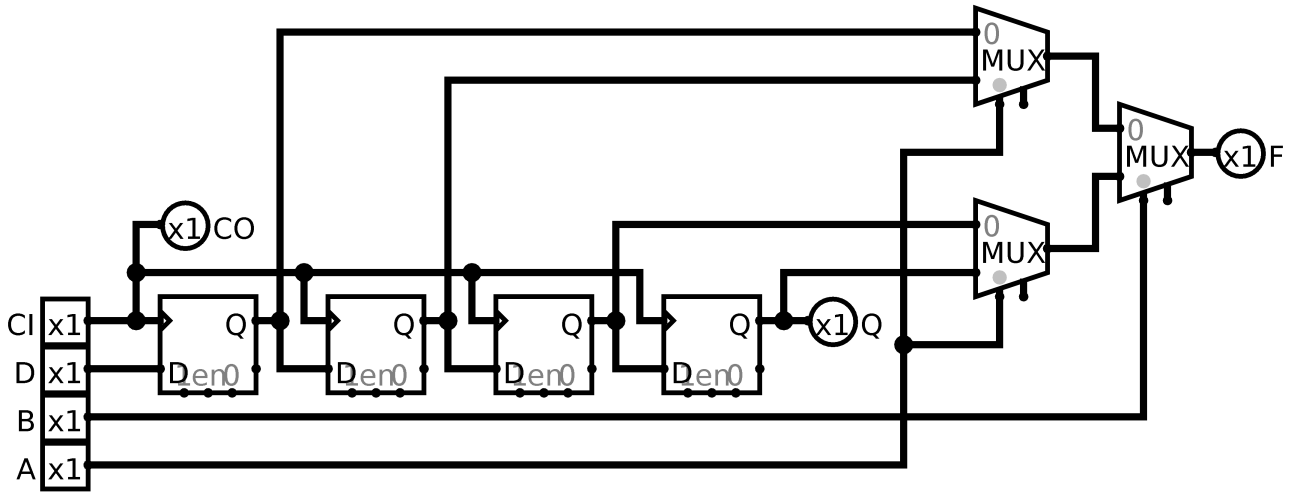

Figure 7: LUT Slice in Logisim

Here we can see we have three 2:1 multiplexers, and 4 D-flip flops. We have the input of the LUT getting shifted through the d-flip flops. Each set of two outputs from the D-flip flops are connected to the inputs of the mux on select line A. Then the output from each mux will be fed into the final mux performing the computation on select line B.

5.4 Top Level Test Mode Diagram

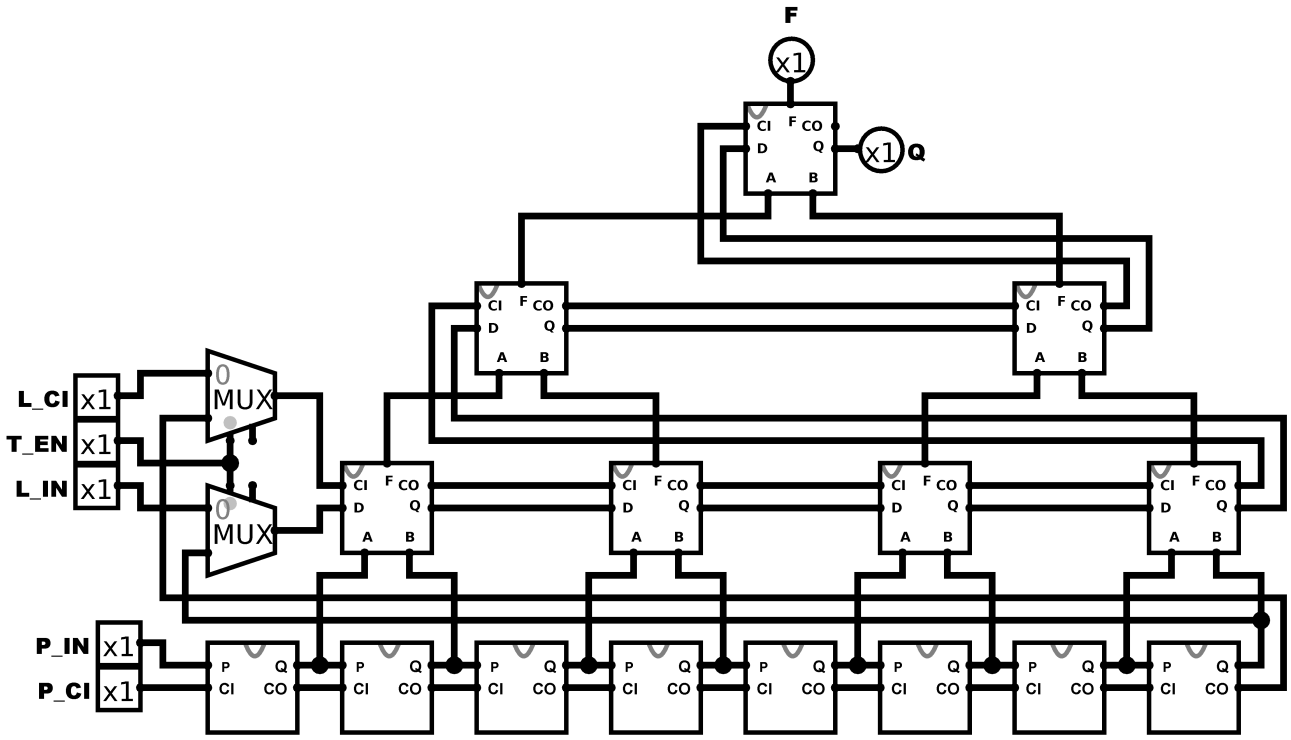

Figure 8: Top Level Test Mode Diagram (8 Inputs, 28-bit slice)

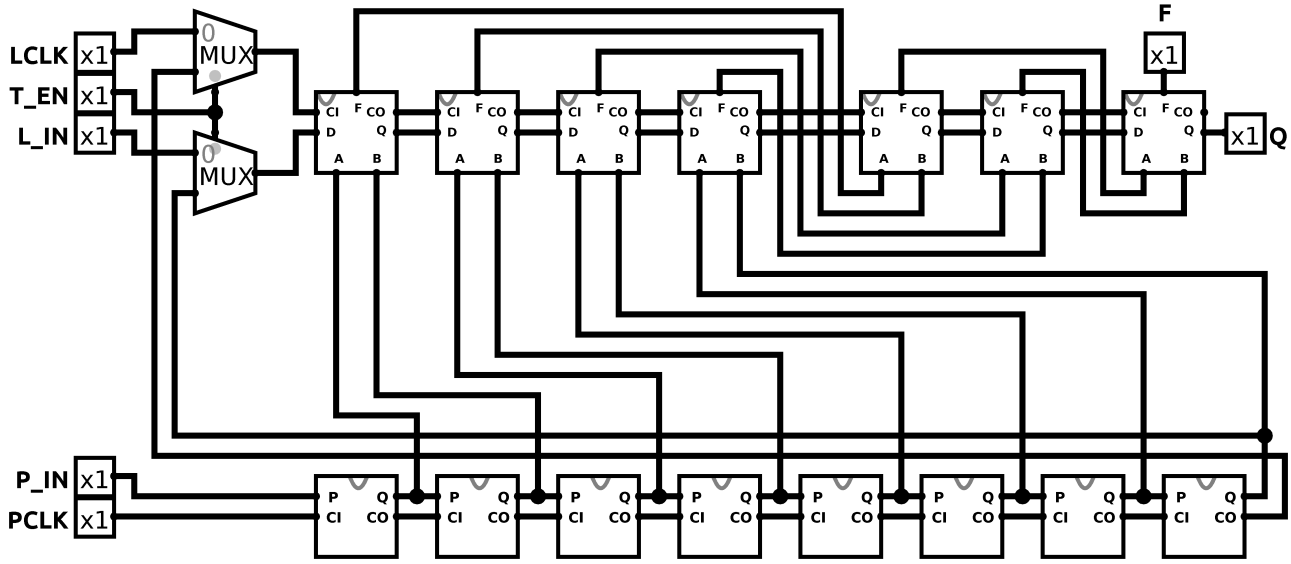

Figure 9: New Designed Top Level Test Mode Diagram (8 Inputs, 28-bit slice)

This test mode is configured with two 2:1 multiplexers. Each multiplexer will select the input line whether we are in test or normal mode. In test mode enabled, we see that we need to feed in the last output of the shift register into the input of L_IN, where L_IN is the input data into the LUT. Also, the clock used to shift the P data in, must now be the same clock input for the LUT. If test mode is disabled, we can see that the multiplexer will select the L_IN to be the data configured by the user (not the data outputted from P).

6 VHDL Models with Test Mode

6.1 Top Level Module

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity top is
5      generic(
6         n      : integer := 2    -- number of levels in tree
7     );
8      port(
9          p_clk : in std_logic;   -- shift register clock
10         l_clk : in std_logic;   -- lut shift register clock
11         p_in  : in std_logic;   -- shift register input (P)
12         l_in  : in std_logic;   -- lut shift register input
13         t_en  : in std_logic;   -- test enalbe input
14         f_o   : out std_logic;  -- final output of computation
15         q_o   : out std_logic   -- final lut shift register output
16     );
17 end top;
18
19 architecture rtl of top is
20
21     signal shift_clki   : std_logic := '0';
22     signal shift_clk    : std_logic := '0';
23     signal l_shf_ini    : std_logic := '0';
24     signal l_shf_in     : std_logic := '0';
25     signal p_out        : std_logic := '0';
26
27 begin
28
29     -- test mux connects output of P into input of LUT and use same clock line
30     t_mux_1 : entity work.mux2x1 port map(l_clk, p_clk, t_en, shift_clki);
31     t_mux_2 : entity work.mux2x1 port map(l_in,  p_out, t_en, l_shf_ini);
32
33     t_inv_1 : entity work.invx1  port map(shift_clki, shift_clk);
34     t_inv_2 : entity work.invx1  port map(l_shf_ini, l_shf_in);
35
36     lut_1 : entity work.lut
37     generic map(
38         n       => n
39     )
40     port map(
41         s_clk   => p_clk,
42         l_clk   => shift_clk,
43         s_in    => p_in,
44         l_in    => l_shf_in,
45         t_po    => p_out,
46         f_o     => f_o,
47         q_o     => q_o
48     );
49
50 end rtl;
```

Listing 1: Top Module

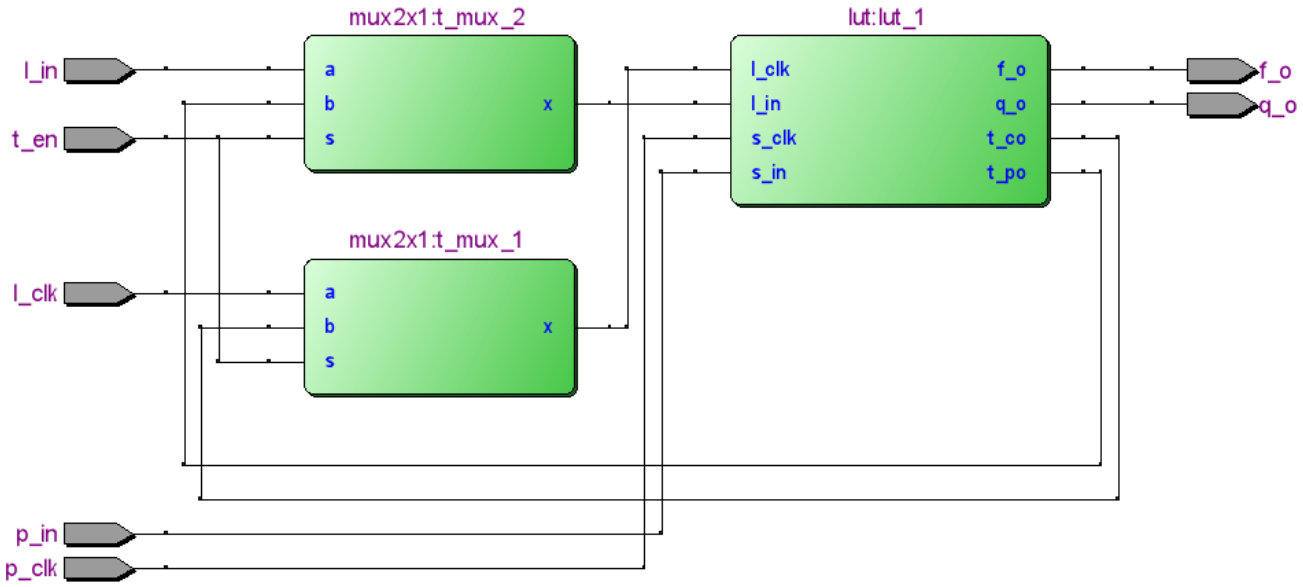

Figure 10: RTL Design of Top Level

6.2 Slice Modules

6.2.1 LUT

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lut_slice is
    port(
        clk_i   : in std_logic;
        d       : in std_logic;
        a       : in std_logic;
        b       : in std_logic;
        q       : out std_logic;
        f       : out std_logic
    );
end lut_slice;

architecture rtl of lut_slice is

    component dffposx1 is
        port(
            clk : in std_logic;
            d   : in std_logic;
            q   : out std_logic
        );
    end component;

    component mux2x1 is
        port(
            b : in std_logic;
            a : in std_logic;
            s : in std_logic;
            x : out std_logic
        );
    end component;

    component invx1 is
        port(
            a : in std_logic;
            x : out std_logic
        );
    end component;

    -- flip flop and mux outputs
    signal ff_o     : std_logic_vector(4 downto 0) := (others => '0');
    signal mux_o    : std_logic_vector(1 downto 0) := (others => '0');
    signal mux_fo   : std_logic_vector(1 downto 0) := (others => '0');
    signal f_muxo   : std_logic := '0';

begin

    -- shifting in from LSB to MSB
    shift_gen_lut : for i in 0 to 3 generate
        ff_lut_i : dffposx1
        port map(
            clk => clk_i,
            d   => ff_o(i),
            q   => ff_o(i+1)
        );
    end generate;

```

```
60      -- lut shift out is output from prev ff
61      q <= ff_o(4);
62      ff_o(0) <= d;
63
64      -- select first two outputs of LUT
65      -- on sel line A
66      mux1 : mux2x1 port map(ff_o(1),  ff_o(2), a, mux_o(0));
67      inv1 : invx1  port map(mux_o(0), mux_fo(0));
68
69      -- select last two outputs of LUT
70      -- on sel line A
71      mux2 : mux2x1 port map(ff_o(3),  ff_o(4), a, mux_o(1));
72      inv2 : invx1  port map(mux_o(1), mux_fo(1));
73
74      -- select the outputs from
75      -- each mux on sel line B
76      mux3 : mux2x1 port map(mux_fo(0), mux_fo(1), b, f_muxo);
77
78      -- invert mux due to func of mux: y=!(S?(A:B))
79      inv3 : invx1 port map(f_muxo, f);
80
81  end rtl;
```

Listing 2: Lookup Table Slice Module

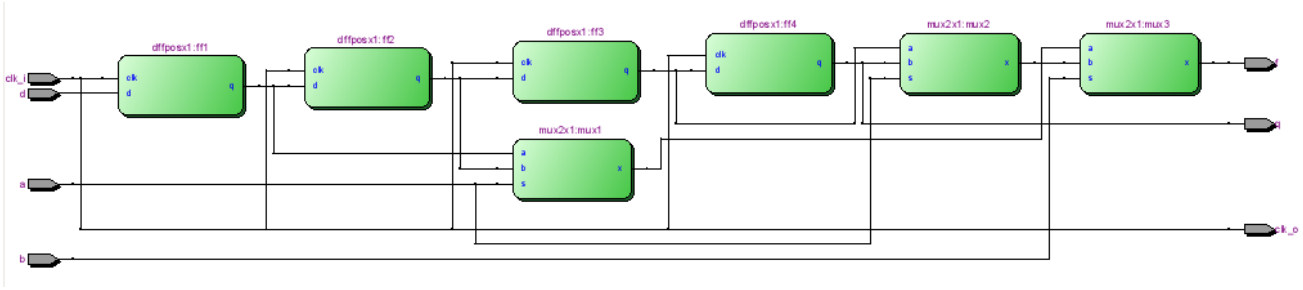

Figure 11: RTL Design of LUT Slice

6.2.2 Shift Register

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity shift_slice is
5       port(
6           clk_i   : in std_logic;
7           p       : in std_logic;
8           clk_o   : out std_logic;
9           q       : out std_logic
10      );
11  end shift_slice;
12
13  architecture rtl of shift_slice is
14
15      component dffposx1
16          port(
17              clk : in std_logic;
18              d   : in std_logic;
19              q   : out std_logic
20          );
21      end component;
22
23  begin
```

```
24
25      ff_p1 : dffposx1
26      port map(
27          clk => clk_i,
28          d   => p,
29          q   => q
30      );
31
32      clk_o <= clk_i;
33
34  end rtl;
```

Listing 3: Lookup Table Slice Module

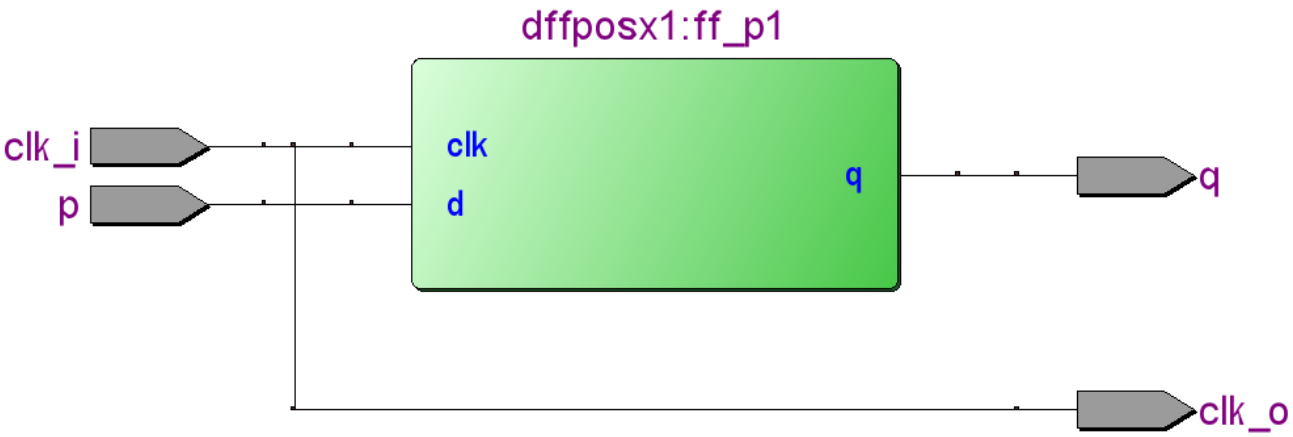

Figure 12: RTL Design of Shift Register Slice

6.3 Gates

6.3.1 D-Flip Flop

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity dffposx1 is
5       generic(
6           delay : time := 0 ps
7       );
8       port(
9           clk : in std_logic;
10          d   : in std_logic;
11          q   : out std_logic
12      );
13  end dffposx1;
14
15  architecture rtl of dffposx1 is begin
16      process(clk) begin
17          if rising_edge(clk) then
18              q <= d after delay;
19          end if;
20      end process;
21  end rtl;
```

Listing 4: D-Flip Flop Module

6.3.2 2:1 Multiplexer

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mux2x1 is
    generic(
        delay : time := 0 ps
    );
    port(
        b : in std_logic;
        a : in std_logic;
        s : in std_logic;
        x : out std_logic
    );
end mux2x1;

architecture rtl of mux2x1 is begin

    x <= not(b) after delay when (s = '0') else
         not(a) after delay when (s = '1');

end rtl;
```

Listing 5: 2:1 Multiplexer Module

6.4 VHDL Test Benches

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lut_slice_tb is
end lut_slice_tb;

architecture behavior of lut_slice_tb is

    signal clk      : std_logic := '0';
    signal di       : std_logic := '0';
    signal a        : std_logic := '0';
    signal b        : std_logic := '0';
    signal q        : std_logic;
    signal f        : std_logic;

    component lut_slice
        port(
            clk_i   : in std_logic;
            d       : in std_logic;
            a       : in std_logic;
            b       : in std_logic;
            q       : out std_logic;
            f       : out std_logic
        );
    end component;

begin

    dut : lut_slice
    port map(
        clk_i   => clk,
        d       => di,
        a       => a,
        b       => b,
        q       => q,
```

```
37          f       => f
38      );
39
40      process
41
42          procedure clock is begin
43              clk <= '1';
44              wait for 10 ns;
45              clk <= '0';
46              wait for 10 ns;
47          end procedure clock;
48
49      begin
50
51          wait for 10 ns;
52          a   <= '1';
53          b   <= '1';
54
55          -- lut for AND gate
56          di <= '1';
57          wait for 10 ns;
58          clock;
59
60          di  <= '0';
61          wait for 10 ns;
62          clock;
63
64          di  <= '0';
65          wait for 10 ns;
66          clock;
67
68          di  <= '0';
69          wait for 10 ns;
70          clock;
71
72          wait;
73
74      end process;
75
76  end behavior;
```

Listing 6: LUT Slice Test Bench Module

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use std.textio.all;
4   use work.txt_util.all;
5
6   entity top_tb is
7       generic(
8           --stim_file : string := "test_tree.sim"
9           --stim_file : string := "two_tree.sim"
10          stim_file : string := "top_32.sim"
11      );
12  end top_tb;
13
14  architecture behavior of top_tb is
15
16      constant n : integer := 5;
17
18      signal p_clk : std_logic := '0';
19      signal l_clk : std_logic := '0';
20      signal p     : std_logic := '0';
```

```
21      signal l_in   : std_logic := '0';
22      signal f_o    : std_logic;
23      signal q_o    : std_logic;
24      signal t_en   : std_logic := '0';
25
26      signal p_in_vector : std_logic_vector((2**n)-1         downto 0);
27      signal lut_vector  : std_logic_vector((((2**n)-1)*4)-1 downto 0);
28
29      file stimulus : TEXT open read_mode is stim_file;
30
31      component top
32          generic(
33             n      : integer := 3    -- number of levels in tree
34          );
35          port(
36              p_clk : in std_logic;   -- p shift register clock
37              l_clk : in std_logic;   -- lut shift register clock
38              p_in  : in std_logic;   -- shift register input (P)
39              l_in  : in std_logic;   -- lut shift register input
40              t_en  : in std_logic;   -- test enalbe input
41              f_o   : out std_logic;  -- final output of computation
42              q_o   : out std_logic   -- lut shift register output
43          );
44      end component;
45
46  begin
47
48      dut : top
49      generic map(
50          n     => n
51      )
52      port map(
53          p_clk => p_clk,
54          l_clk => l_clk,
55          p_in  => p,
56          l_in  => l_in,
57          t_en  => t_en,
58          f_o   => f_o,
59          q_o   => q_o
60      );
61
62      process
63
64          procedure clk_p_in is begin
65              p_clk <= '1';
66              wait for 10 ns;
67              p_clk <= '0';
68              wait for 10 ns;
69          end procedure clk_p_in;
70
71          procedure clk_lut_in is begin
72              l_clk <= '1';
73              wait for 10 ns;
74              l_clk <= '0';
75              wait for 10 ns;
76          end procedure clk_lut_in;
77
78          variable l: line;
79          variable p_in_str : string(1 to 2**n);
80          variable l_shf_str: string(1 to ((2**n)-1)*4);
81
82      begin
83
84          while not endfile(stimulus) loop
```

```
85
86              -- load stimulus for this test
87              readline(stimulus, l); read(l, p_in_str);
88              p_in_vector <= to_std_logic_vector(p_in_str);
89
90              readline(stimulus, l); read(l, l_shf_str);
91              lut_vector <= to_std_logic_vector(l_shf_str);
92
93              wait for 50 ns;
94
95              -- clock in the P input
96              for i in 0 to (2**n)-1 loop
97                  p <= p_in_vector(i);
98                  wait for 10 ns;
99                  clk_p_in;
100             end loop;
101
102             -- clock in the "program" (lut functions)
103             for i in 0 to (((2**n)-1)*4)-1 loop
104                 l_in <= lut_vector(i);
105                 wait for 10 ns;
106                 clk_lut_in;
107             end loop;
108
109         end loop;
110
111         report "Test Complete" severity note;
112         wait;
113
114     end process;
115
116 end behavior;
```

Listing 7: Top Level Test Bench Module

7 VHDL Waveform Plots and Results

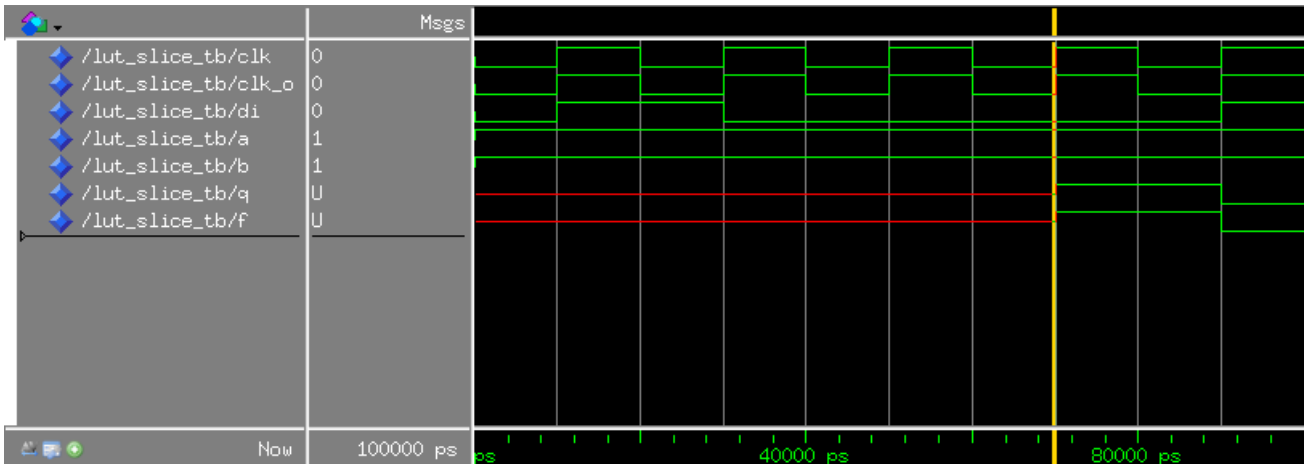

Figure 13: AND Gate of LUT Slice

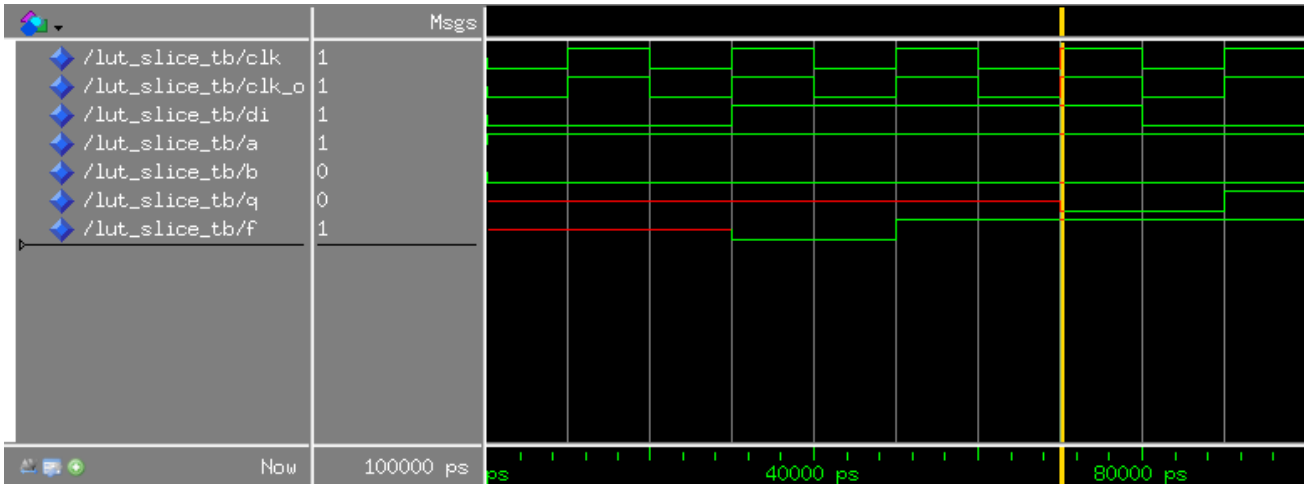

Figure 14: NAND Gate of LUT Slice

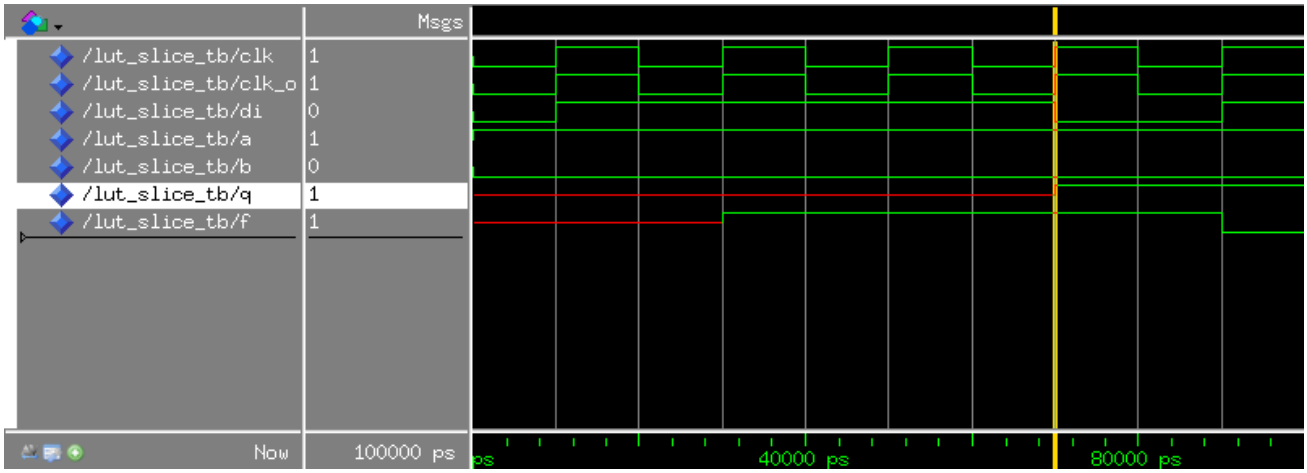

Figure 15: OR Gate of LUT Slice

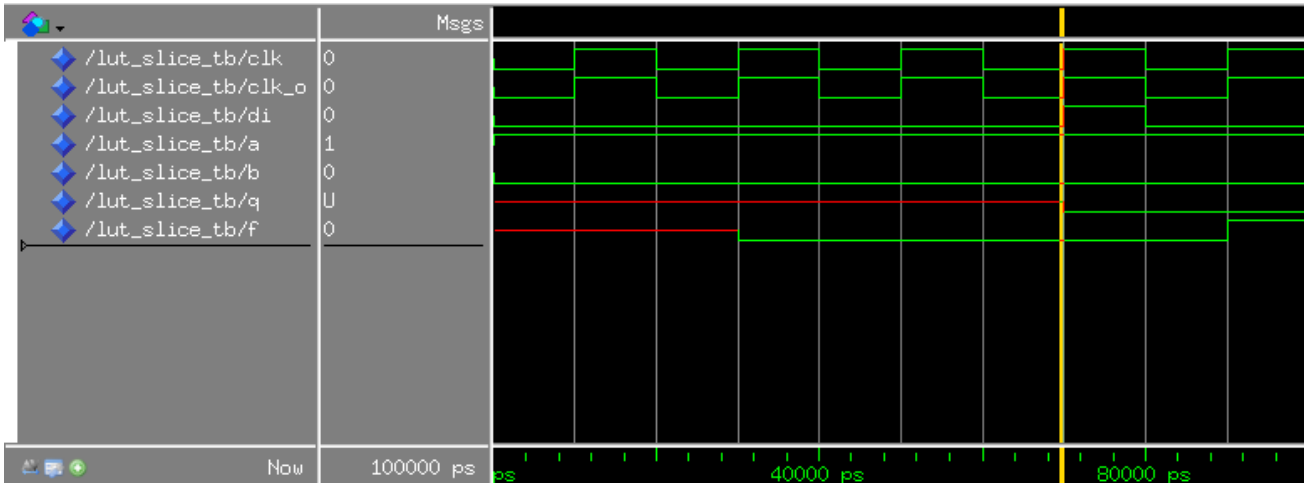

Figure 16: NOR Gate of LUT Slice

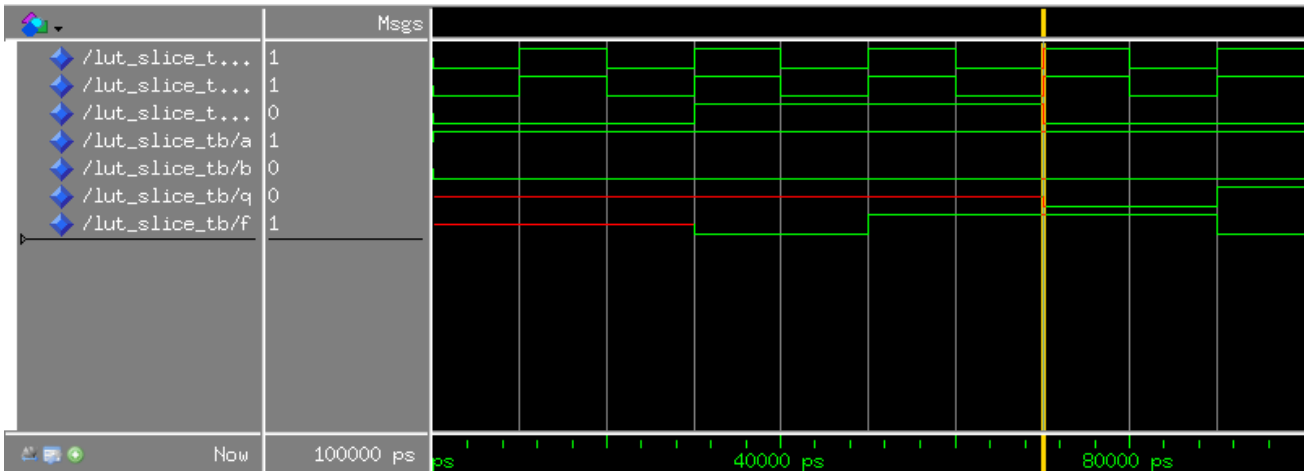

Figure 17: XOR Gate of LUT Slice

Before we could go on with designing the top level in VHDL, we had to make sure that the slice itself worked first. Here we are just showing just a few waveforms from each function. As we can see here, each gate that was tested performed as expected.

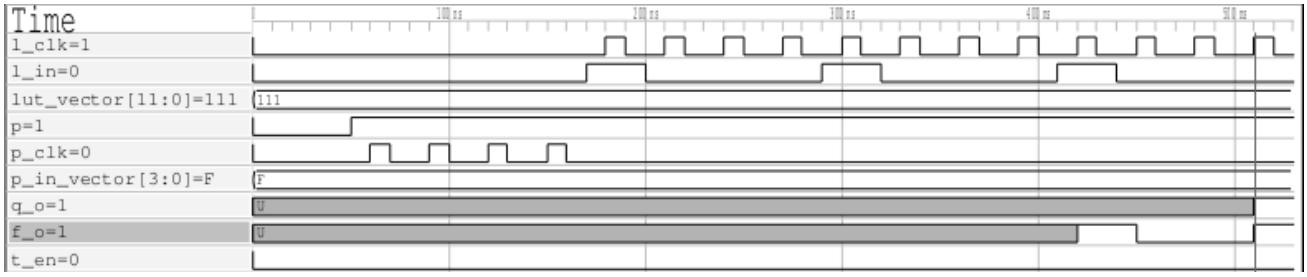

Figure 18: Waveform for Top Test Bench

Since we are testing with 8 input values, there are 2**(28/4) combinations for the combinational functions. Since that is way more than we can test, we selected a few to test. We can see the results in the above waveform (Test Mode Disabled).

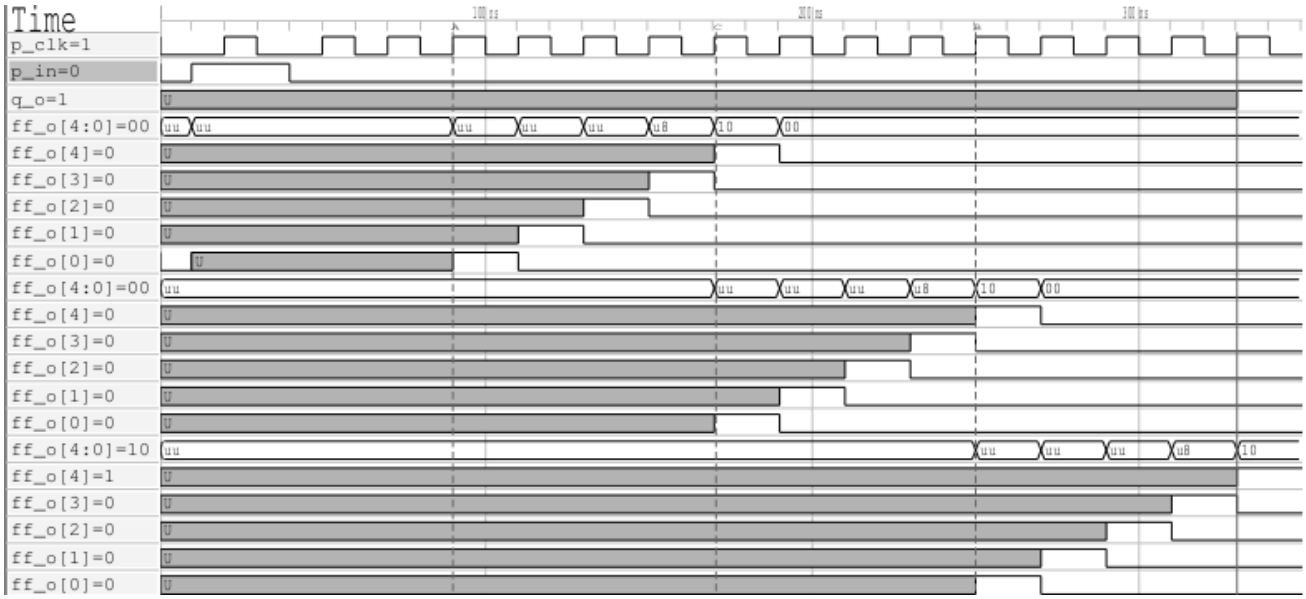

Figure 19: Waveform for Top Test Bench for Test Enabled

This is with Test Enabled for N=2. Here we can see the output of F at the very last clock cycle. In this case we tested the inputs 1111, and made sure our AND gate worked properly, and surely enough it works. We can see

that F is high at the end of the simulation. We tested other functions and inputs as well, but we are just showing one waveform to keep things compact.

8 Work Division

Student	Task
Both	Pin-out Diagram.
Both	Explanation of how the chip works.
Both	Description of the major design decisions made.
Both	Inclusion and explanation of the test mode.
Silbak	VHDL LUT slice Module
Kasula	VHDL LUT slice Test Bench Module
Silbak	VHDL Top Level Module
Both	VHDL Top Level Test Bench Module
Silbak	LUT Slice Block Diagram
Kasula	LUT Slice Top Level Diagram

Table 3: Task Assignment

9 Magic Layouts

9.1 LUT Slice Layout

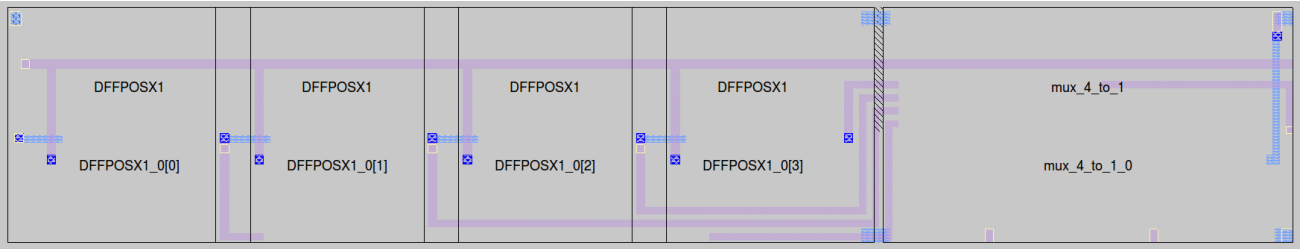

Figure 20: LUT Slice Magic Layout

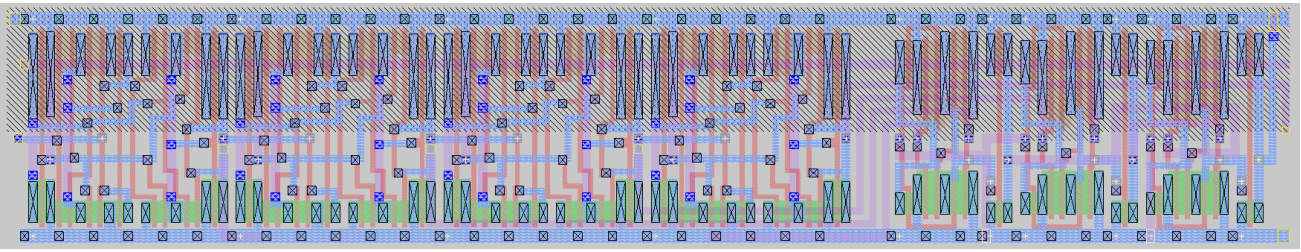

Figure 21: LUT Slice Internal Magic Layout

9.2 Shift Slice Layout

Figure 22: Shift Slice Magic Layout

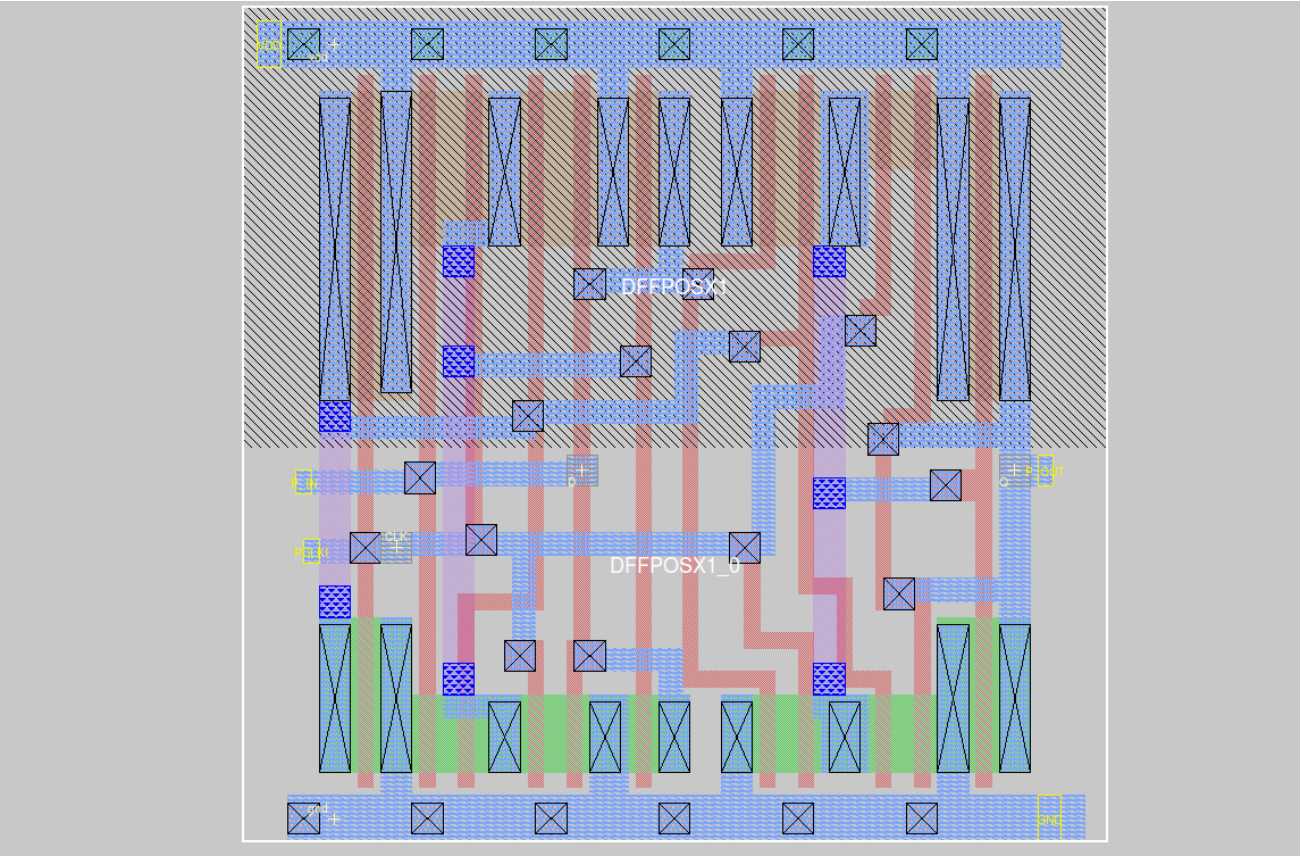

Figure 23: Shift Slice Internal Magic Layout

10 IRSIM Simulations

10.1 Bit Slice Simulations

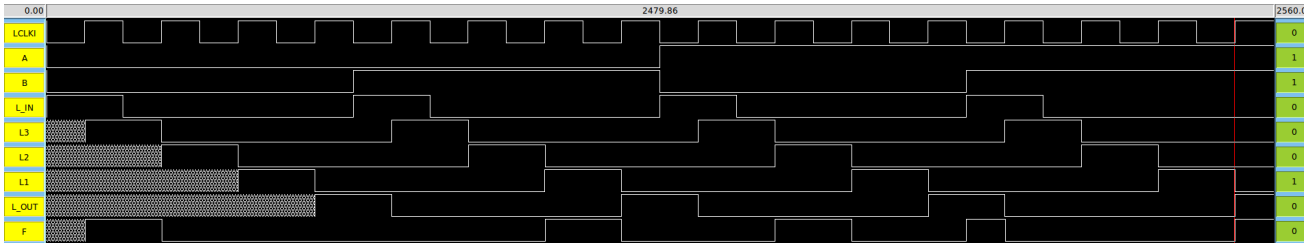

Figure 24: LUT Slice with AND gate 'programmed'

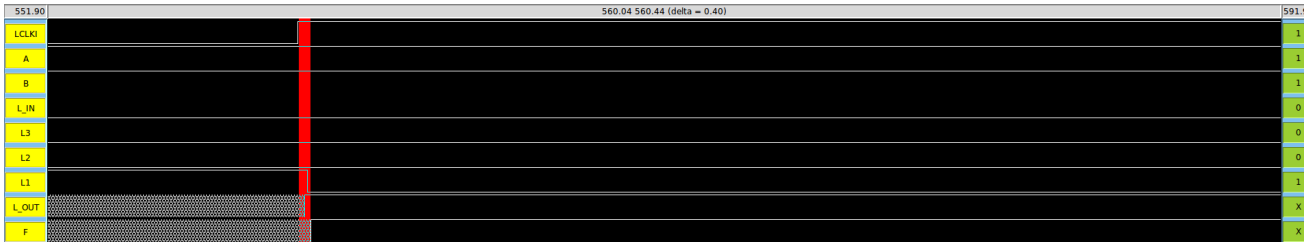

Figure 25: Rising Edge of F output of LUT Slice

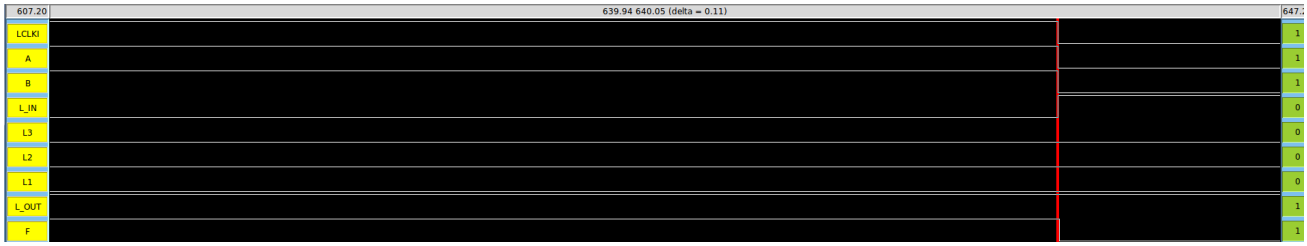

Figure 26: Falling Edge of F output of LUT Slice

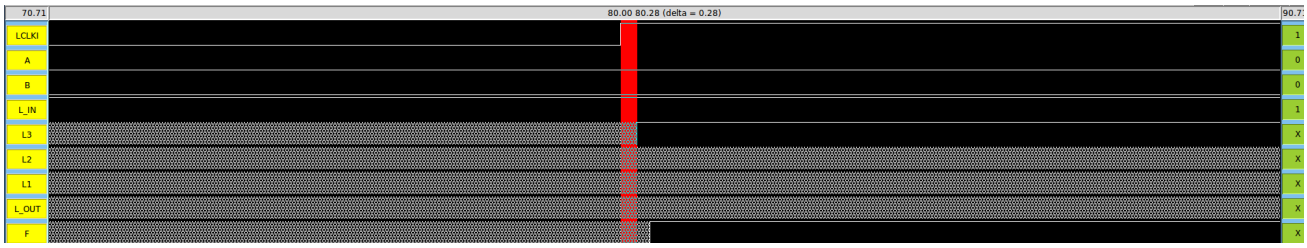

Figure 27: Rising Edge of L3 DFF of LUT Slice

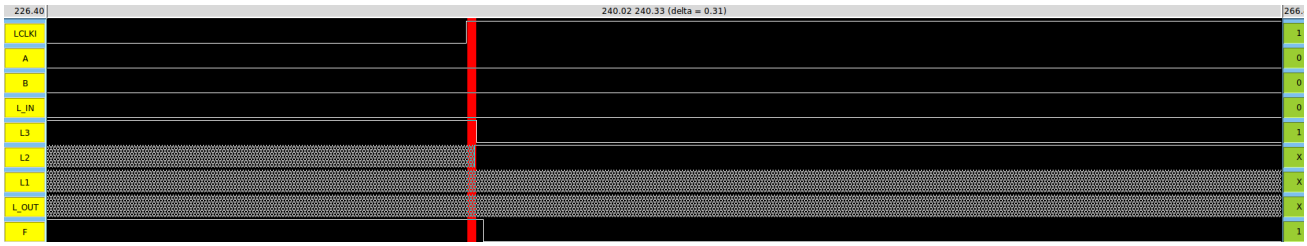

Figure 28: Falling Edge of L3 DFF of LUT Slice

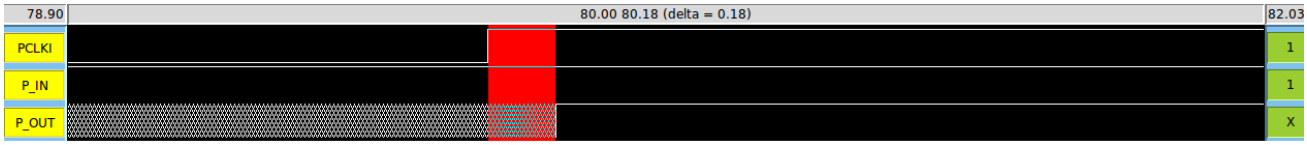

Figure 29: Rising Edge of DFF of Shift Slice

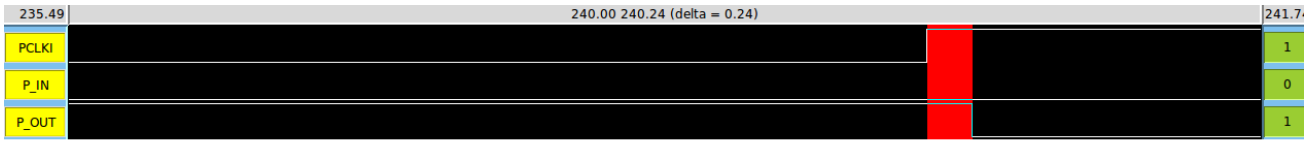

Figure 30: Falling Edge of DFF of Shift Slice

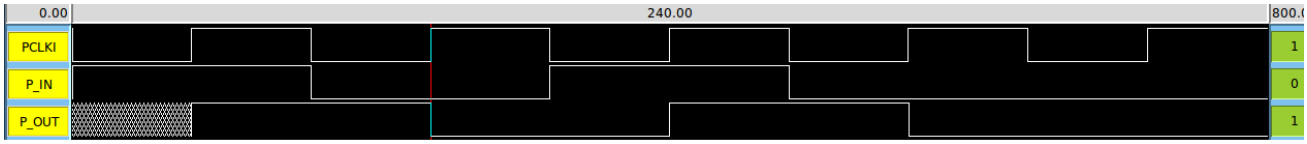

Figure 31: Waveform of DFF of Shift Slice

11 Gate Level Simulations

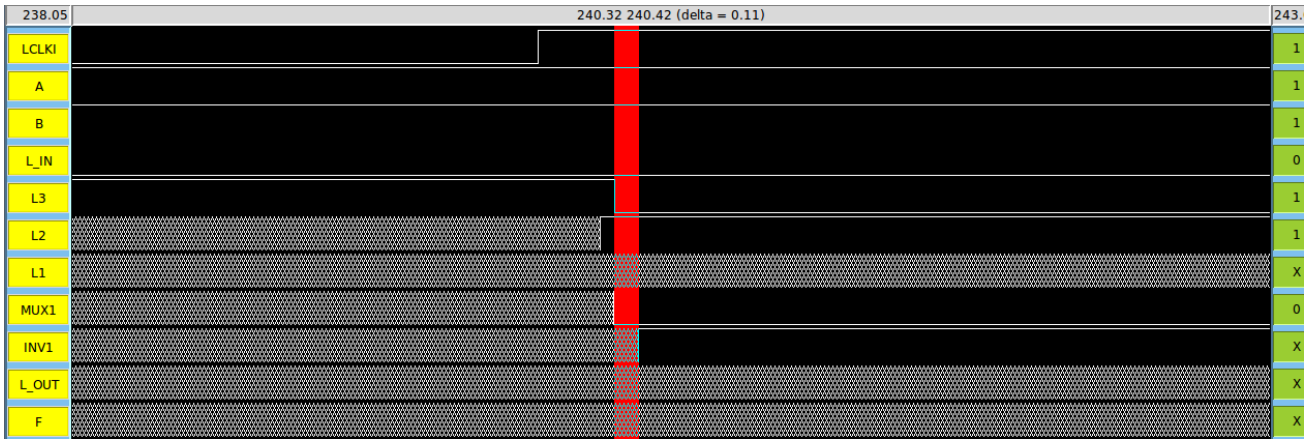

Figure 32: Rising Edge of Inverter

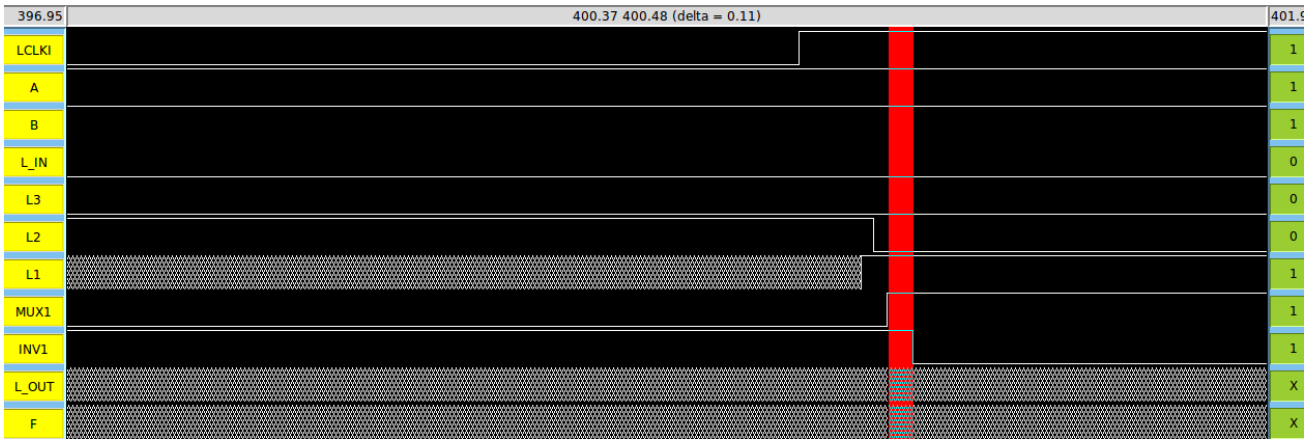

Figure 33: Falling Edge of Inverter

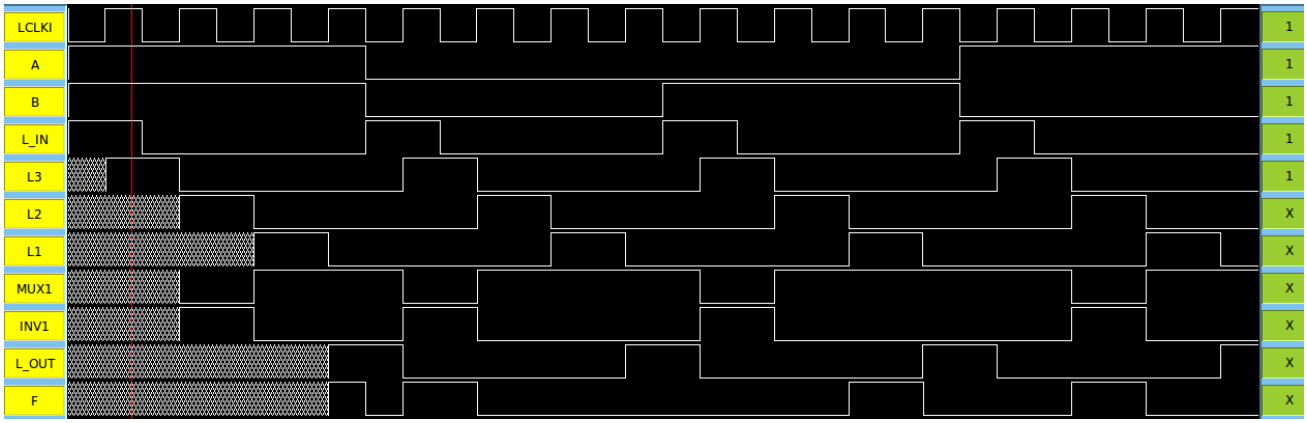

Figure 34: Waveform of LUT Slice (Includes nodes at each flip flop)

12 HSpice Simulations

12.1 Bit Slice Simulations

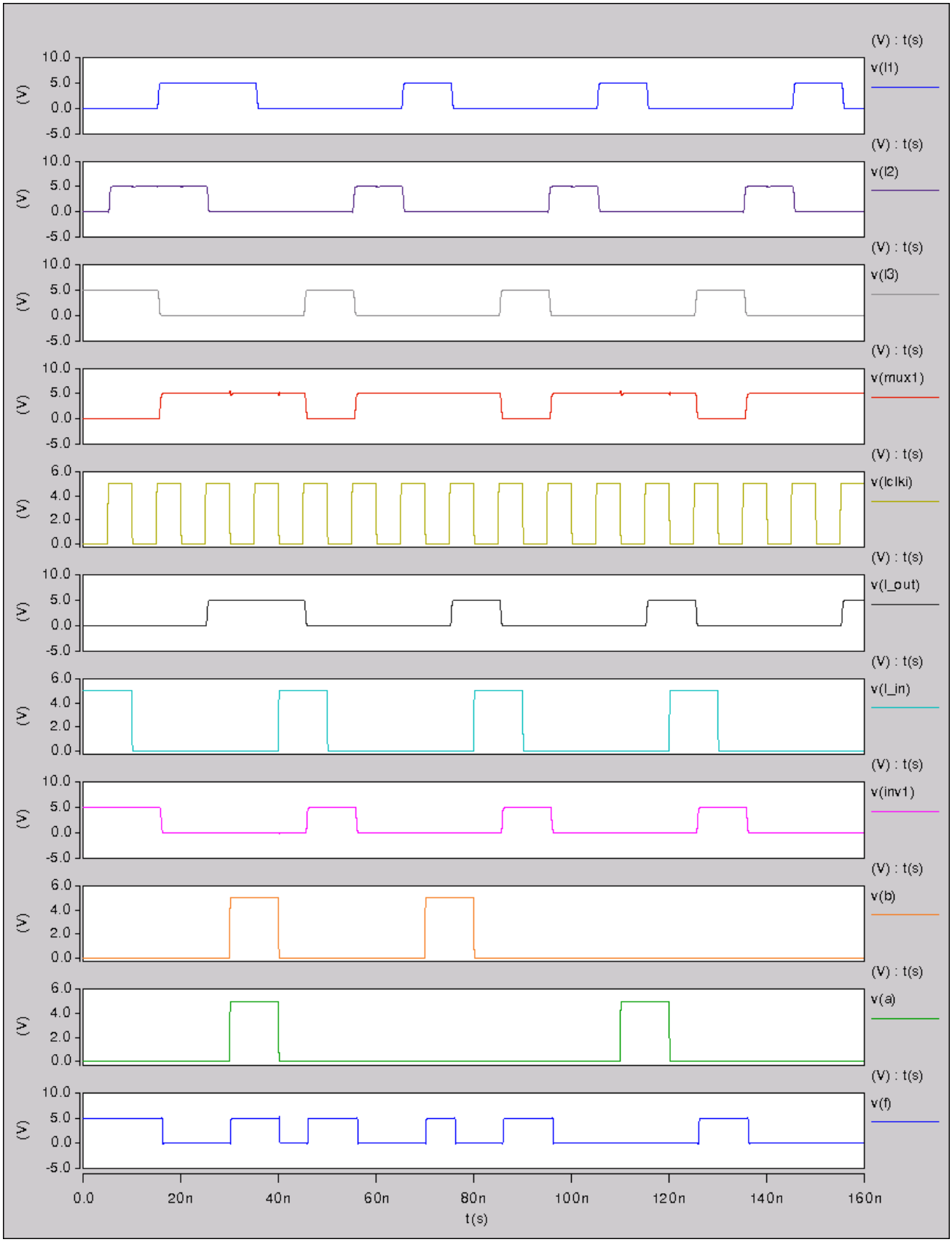

Figure 35: Hspice LUT Slice Wavefrom

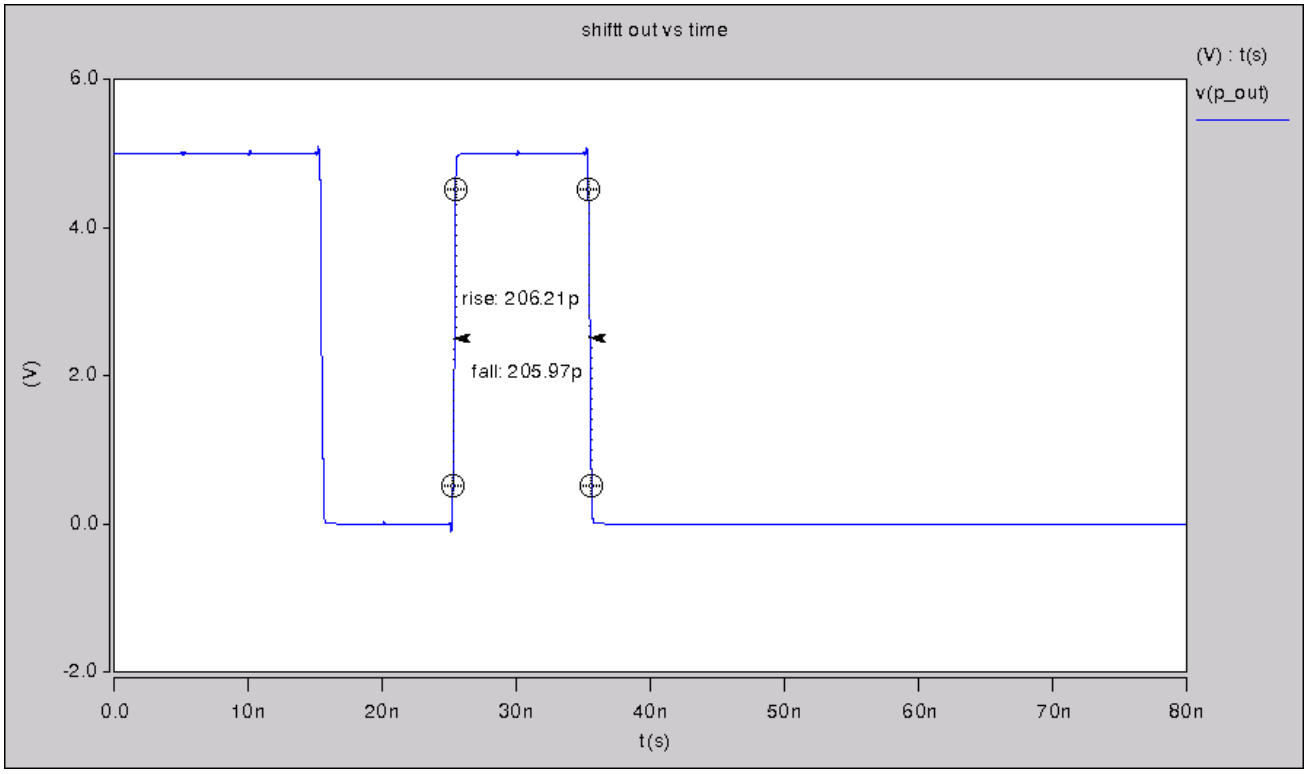

Figure 36: Hspice Delay of Shift Register Slice

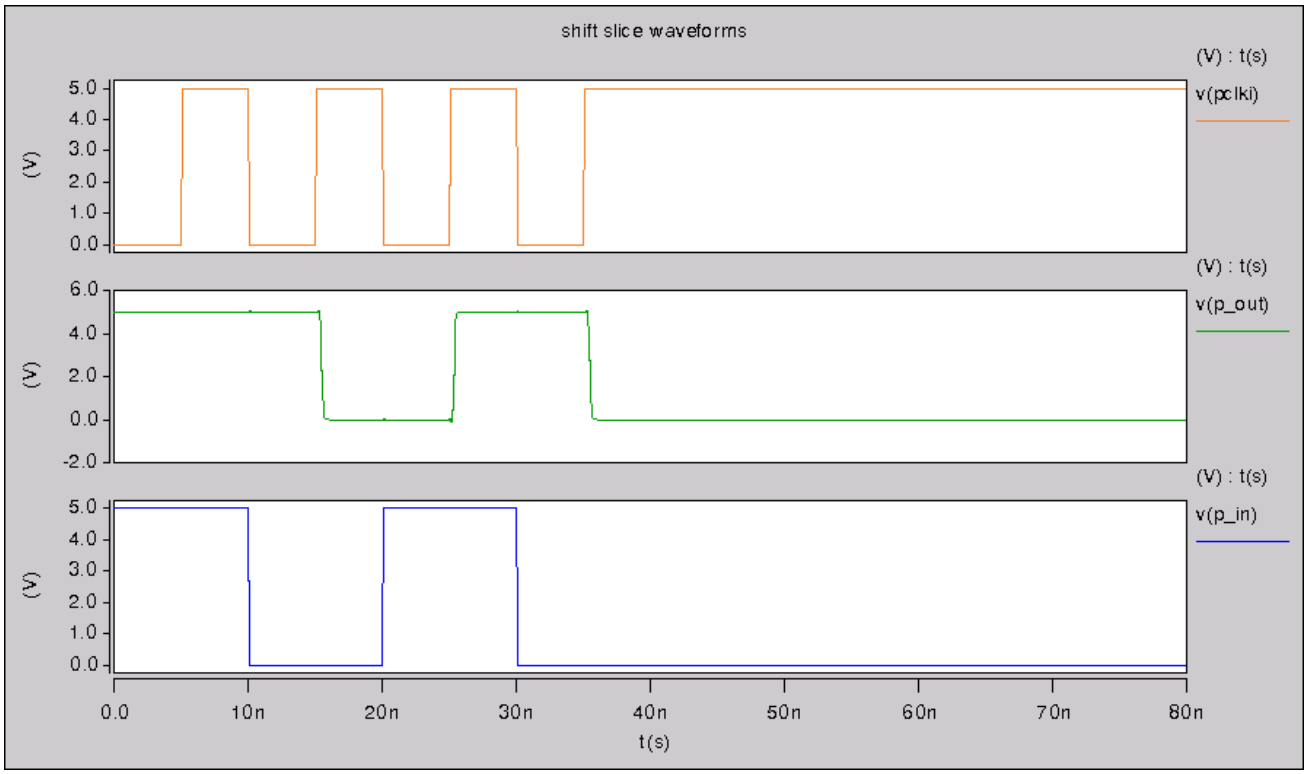

Figure 37: Hspice Shift Register Slice Waveform

12.2 Gate Level Simulations

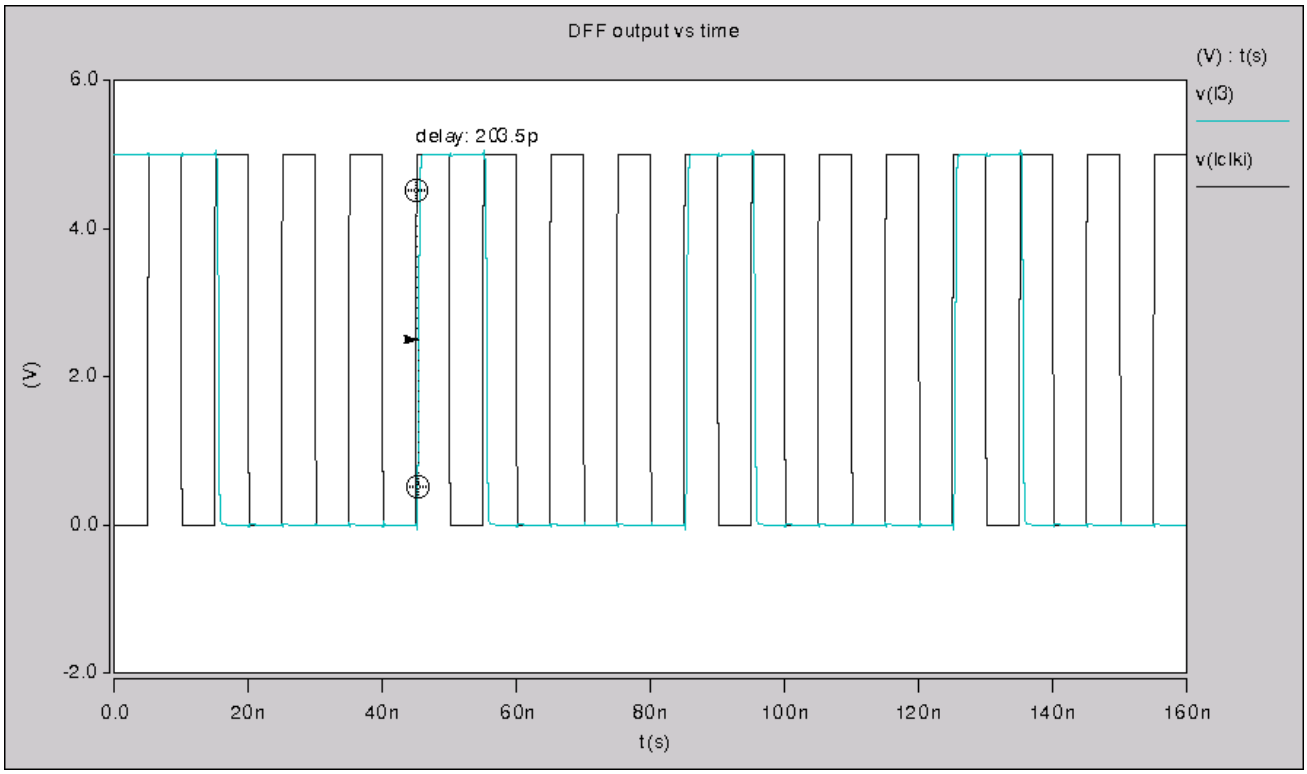

Figure 38: Hspice Delay of DFF

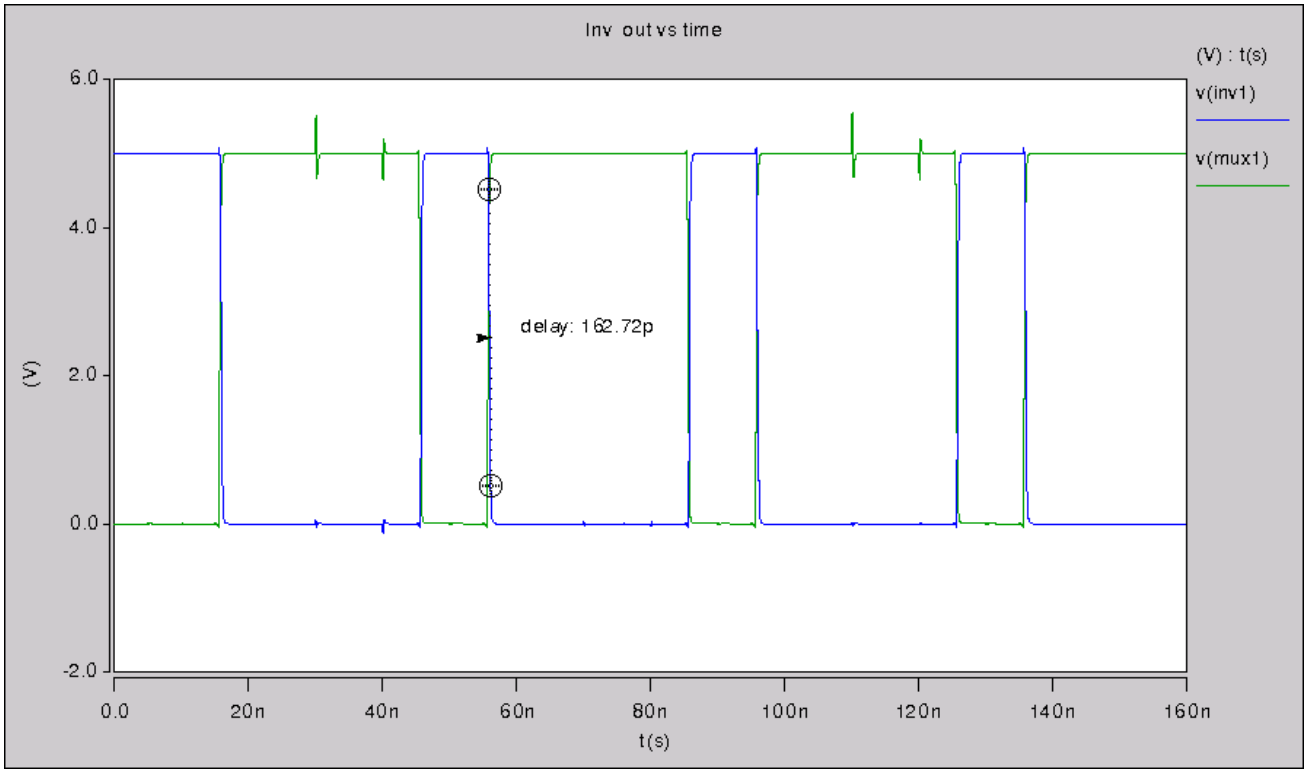

Figure 39: Hspice Delay of Inveter

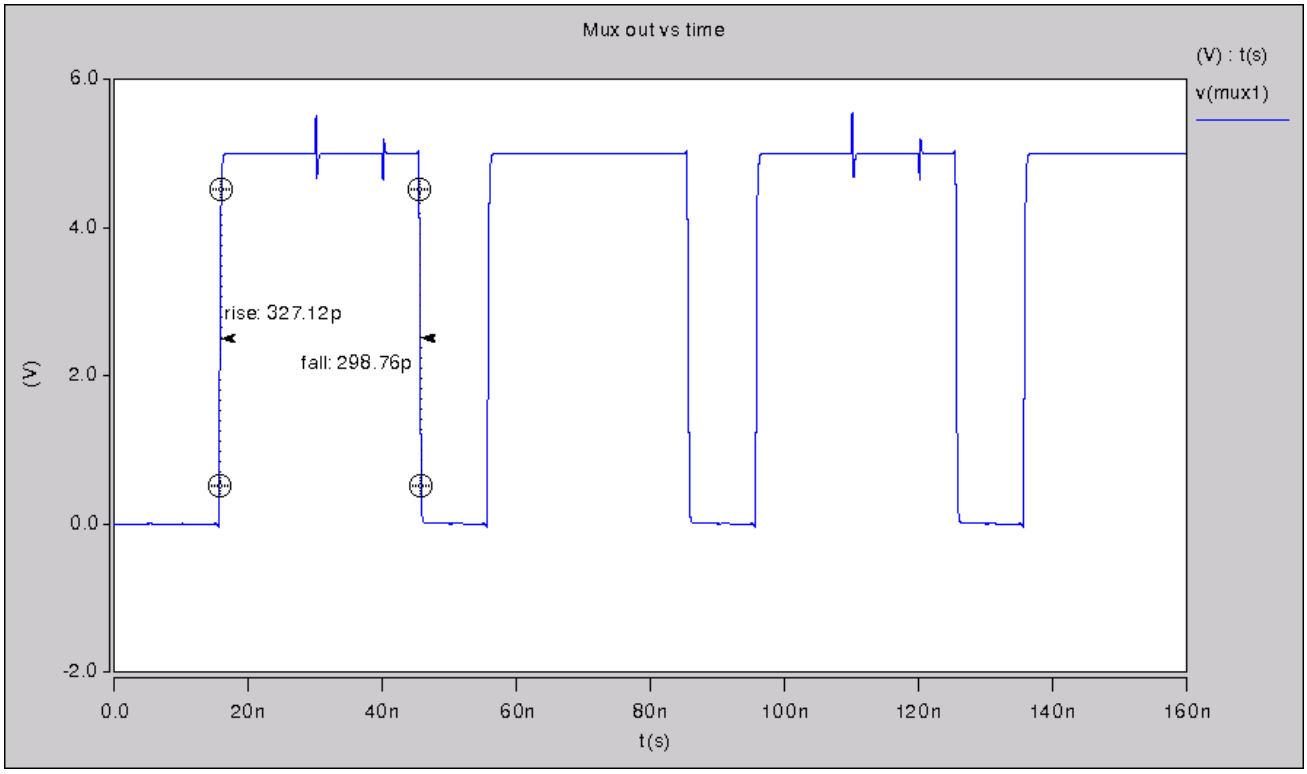

Figure 40: Hspice Delay of Mux

	IRSIM	VHDL	Hspice
Inverter	0.11 ns	0.159 ns	0.1627 ns
Mux	0.315 ns	0.312 ns	0.31214 ns
DFF	0.255 ns	0.198 ns	0.20305 ns

Table 4: Leaf Level Component Delay Times

13 VHDL Modules with Delay

```
library ieee;
use ieee.std_logic_1164.all;

entity top_del is
    generic(
       n      : integer := 2    -- number of levels in tree
   );
    port(
        p_clk : in std_logic;   -- shift register clock
        l_clk : in std_logic;   -- lut shift register clock
        p_in  : in std_logic;   -- shift register input (P)
        l_in  : in std_logic;   -- lut shift register input
        t_en  : in std_logic;   -- test enalbe input
        f_o   : out std_logic;  -- final output of computation
        q_o   : out std_logic   -- final lut shift register output
    );
end top_del;

architecture rtl of top_del is

    signal shift_clki   : std_logic := '0';
    signal shift_clk    : std_logic := '0';
    signal l_shf_ini    : std_logic := '0';
    signal l_shf_in     : std_logic := '0';
    signal p_out        : std_logic := '0';

begin

    -- test mux connects output of P into input of LUT and use same clock line
    t_mux_1 : entity work.mux2x1_del port map(l_clk, p_clk, t_en, shift_clki);
    t_mux_2 : entity work.mux2x1_del port map(l_in,  p_out, t_en, l_shf_ini);

    t_inv_1 : entity work.invx1_del  port map(shift_clki, shift_clk);
    t_inv_2 : entity work.invx1_del  port map(l_shf_ini, l_shf_in);

    lut_1 : entity work.lut_del
    generic map(
        n       => n
    )
    port map(
        s_clk   => p_clk,
        l_clk   => shift_clk,
        s_in    => p_in,
        l_in    => l_shf_in,
        t_po    => p_out,
        f_o     => f_o,
        q_o     => q_o
    );

end rtl;
```

Listing 8: Top Module Delay

```
library ieee;
use ieee.std_logic_1164.all;

entity lut_del is
    generic(
        n       : integer := 2    -- number of levels in tree
   );
```

```
8       port(
9           s_clk    : in std_logic;   -- shift register clock
10          l_clk    : in std_logic;   -- lut shift register clock
11          s_in     : in std_logic;   -- shift register input (P)
12          l_in     : in std_logic;   -- lut shift register input
13          t_po     : out std_logic;  -- p_out for test mode
14          f_o      : out std_logic;  -- final output of computation
15          q_o      : out std_logic   -- final lut shift register output
16      );
17  end lut_del;
18
19  architecture rtl of lut_del is
20
21      -- tree diagram for n = 3
22      -- row 0 : A
23      -- row 1 : BC
24      -- row 2 : DEFG
25      -- row 3 : shifter
26
27      -- lut connection diagram
28      -- l_in -> D -> E -> F -> G -> B -> C -> A
29
30      component dffposx1_del is
31          port(
32              clk : in std_logic;
33              d   : in std_logic;
34              q   : out std_logic
35          );
36      end component;
37
38      component lut_slice_del is
39          port(
40              clk_i   : in std_logic;
41              d       : in std_logic;
42              a       : in std_logic;
43              b       : in std_logic;
44              q       : out std_logic;
45              f       : out std_logic
46          );
47      end component;
48
49      -- carray_array(row, col)
50      type carry_array is array (0 to n, 0 to 2**n) of std_logic;
51      signal clk_c : carry_array;
52
53      -- carries select outputs of one
54      -- row to select inputs of next one
55      signal r_c   : carry_array;
56
57      -- carry output of each lut shift
58      -- register
59      signal l_c   : carry_array;
60
61      -- input shift register carries
62      signal s_c   : std_logic_vector(2**n downto 0) := (others => '0');
63
64      --  input shift regstier clock carries
65      signal p_clk : std_logic_vector(2**n downto 0) := (others => '0');
66
67  begin
68
69      -- generate the input shift register
70      shift_gen : for i in 0 to (2**n)-1 generate
71          ff_i : dffposx1_del
```

```
72          port map(
73              clk => s_clk,
74              d   => s_c(i),
75              q   => s_c(i+1)
76          );
77      end generate shift_gen;
78
79      -- generate the tree
80      level_gen : for level in 0 to n-1 generate
81          lut_gen : for i in 0 to (2**level)-1 generate
82              lut_i : lut_slice_del
83              port map(
84                  clk_i => l_clk,
85                  d     => l_c(level, i),
86                  q     => l_c(level, i+1),
87                  a     => r_c(level+1, i*2),
88                  b     => r_c(level+1, i*2+1),
89                  f     => r_c(level, i)
90               );
91          end generate;
92      end generate;
93
94      -- output the value of our function
95      q_o <= l_c(0, 1);
96      f_o <= r_c(0, 0);
97
98      -- connect each row of LUTs together
99      -- first slice in row i connects to
100     -- last slice in row i+1
101     lut_connect : for level in 0 to n-2 generate
102         l_c(level, 0) <= l_c(level+1, (2**(level+1)));
103     end generate;
104
105     -- connect input shift register to bottom row of slices
106     shift_connect : for i in 0 to (2**n)-1 generate
107         r_c(n, i) <= s_c(i+1);
108     end generate;
109
110     -- connect input to shift register
111     s_c(0) <= s_in;
112
113     -- connect input to first lut (bottom row)
114     l_c(n-1, 0) <= l_in;
115
116     -- use last slice of P shift register
117     -- feed it into LUT input for test mode
118     t_po <= s_c(2**n);
119
120 end rtl;
```

Listing 9: Look Up Table Module Delay

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity lut_slice_del is
6      port(
7          clk_i   : in std_logic;
8          d       : in std_logic;
9          a       : in std_logic;
10         b       : in std_logic;
11         q       : out std_logic;
```

```
12          f       : out std_logic
13      );
14  end lut_slice_del;
15
16  architecture rtl of lut_slice_del is
17
18      component dffposx1_del is
19          port(
20              clk : in std_logic;
21              d   : in std_logic;
22              q   : out std_logic
23          );
24      end component;
25
26      component invx1_del is
27          port(
28              a   : in std_logic;
29              x   : out std_logic
30          );
31      end component;
32
33      component mux2x1_del is
34          port(
35              b : in std_logic;
36              a : in std_logic;
37              s : in std_logic;
38              x : out std_logic
39          );
40      end component;
41
42      -- flip flop and mux outputs
43      signal ff_o     : std_logic_vector(4 downto 0) := (others => '0');
44      signal mux_o    : std_logic_vector(1 downto 0) := (others => '0');
45      signal mux_fo   : std_logic_vector(1 downto 0) := (others => '0');
46      signal f_muxo   : std_logic := '0';
47
48  begin
49
50      -- shifting in from LSB to MSB
51      shift_gen_lut : for i in 0 to 3 generate
52          ff_lut_i : dffposx1_del
53          port map(
54              clk => clk_i,
55              d   => ff_o(i),
56              q   => ff_o(i+1)
57          );
58      end generate;
59
60      -- lut shift out is output from prev ff
61      q <= ff_o(4);
62      ff_o(0) <= d;
63
64      -- select first two outputs of LUT
65      -- on sel line A
66      mux1 : mux2x1_del port map(ff_o(1),  ff_o(2), a, mux_o(0));
67      inv1 : invx1_del  port map(mux_o(0), mux_fo(0));
68
69      -- select last two outputs of LUT
70      -- on sel line A
71      mux2 : mux2x1_del port map(ff_o(3),  ff_o(4), a, mux_o(1));
72      inv2 : invx1_del  port map(mux_o(1), mux_fo(1));
73
74      -- select the outputs from
75      -- each mux on sel line B
```

```
76      mux3 : mux2x1_del port map(mux_fo(0), mux_fo(1), b, f_muxo);
77
78      -- invert mux due to func of mux: y = !(S?(A:B))
79      inv3 : invx1_del port map(f_muxo, f);
80
81  end rtl;
```

Listing 10: Look Up Table Slice Module Delay

13.1 VHDL Test Bench Modules with Delay

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use std.textio.all;
4   use work.txt_util.all;
5
6   entity top_test_del_tb is
7   end top_test_del_tb;
8
9   architecture behavior of top_test_del_tb is
10
11      constant n : integer := 5;
12
13      component top_del
14          generic(
15              n       : integer := 3    -- number of levels in tree
16          );
17          port(
18              p_clk : in std_logic;   -- p shift register clock
19              l_clk : in std_logic;   -- lut shift register clock
20              p_in  : in std_logic;   -- shift register input (P)
21              l_in  : in std_logic;   -- lut shift register input
22              t_en  : in std_logic;   -- test enalbe input
23              f_o   : out std_logic;  -- final output of computation
24              q_o   : out std_logic   -- lut shift register output
25          );
26      end component;
27
28      signal p_clk : std_logic := '0';
29      signal l_clk : std_logic := '0';
30      signal p     : std_logic := '0';
31      signal l_in  : std_logic := '0';
32      signal f_o   : std_logic;
33      signal q_o   : std_logic;
34      signal t_en  : std_logic := '0';
35
36  begin
37
38      dut : top_del
39      generic map(
40          n     => n
41      )
42      port map(
43          p_clk => p_clk,
44          l_clk => l_clk,
45          p_in  => p,
46          l_in  => l_in,
47          t_en  => t_en,
48          f_o   => f_o,
49          q_o   => q_o
50      );
51
```

```
52      process
53
54          procedure clk is begin
55              p_clk <= '1';
56              wait for 10 ns;
57              p_clk <= '0';
58              wait for 10 ns;
59          end procedure clk;
60
61      begin
62
63          wait for 10 ns;
64          -- enable test mode
65          t_en <= '1';
66
67          -- clock in p data
68          p <= '1';
69          wait for 10 ns;
70          clk;
71
72          -- clock in p data
73          p <= '0';
74          wait for 10 ns;
75          clk;
76
77          -- # of FF's per LUT: 4
78          -- # of LUT FF's     : (((2**n)-1)*4)
79          -- # of P FF's       : 2**n
80          -- subtract 2 since we just loaded in two inputs
81          for i in 0 to ((((2**n)-1)*4)+2**n)-2-1 loop
82              clk;
83          end loop;
84
85          wait;
86
87      end process;
88
89  end behavior;
```

Listing 11: Test Mode Enabled Test Bench

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use std.textio.all;
4   use work.txt_util.all;
5
6   entity top_del_tb is
7       generic(
8           --stim_file : string := "test_tree.sim"
9           --stim_file : string := "two_tree.sim"
10          stim_file : string := "top_32.sim"
11      );
12  end top_del_tb;
13
14  architecture behavior of top_del_tb is
15
16      constant n : integer := 5;
17
18      signal p_clk : std_logic := '0';
19      signal l_clk : std_logic := '0';
20      signal p     : std_logic := '0';
21      signal l_in  : std_logic := '0';
22      signal f_o   : std_logic;
```

```
23      signal q_o   : std_logic;
24      signal t_en  : std_logic := '0';
25
26      signal p_in_vector : std_logic_vector((2**n)-1         downto 0);
27      signal lut_vector  : std_logic_vector((((2**n)-1)*4)-1 downto 0);
28
29      file stimulus : TEXT open read_mode is stim_file;
30
31      component top_del
32          generic(
33             n      : integer := 3    -- number of levels in tree
34          );
35          port(
36              p_clk : in std_logic;   -- p shift register clock
37              l_clk : in std_logic;   -- lut shift register clock
38              p_in  : in std_logic;   -- shift register input (P)
39              l_in  : in std_logic;   -- lut shift register input
40              t_en  : in std_logic;   -- test enalbe input
41              f_o   : out std_logic;  -- final output of computation
42              q_o   : out std_logic   -- lut shift register output
43          );
44      end component;
45
46  begin
47
48      dut : top_del
49      generic map(
50          n     => n
51      )
52      port map(
53          p_clk => p_clk,
54          l_clk => l_clk,
55          p_in  => p,
56          l_in  => l_in,
57          t_en  => t_en,
58          f_o   => f_o,
59          q_o   => q_o
60      );
61
62      process
63
64          procedure clk_p_in is begin
65              p_clk <= '1';
66              wait for 10 ns;
67              p_clk <= '0';
68              wait for 10 ns;
69          end procedure clk_p_in;
70
71          procedure clk_lut_in is begin
72              l_clk <= '1';
73              wait for 10 ns;
74              l_clk <= '0';
75              wait for 10 ns;
76          end procedure clk_lut_in;
77
78          variable l: line;
79          variable p_in_str : string(1 to 2**n);
80          variable l_shf_str: string(1 to ((2**n)-1)*4);
81
82      begin
83
84          while not endfile(stimulus) loop
85
86              -- load stimulus for this test
```

```
87              readline(stimulus, l); read(l, p_in_str);
88              p_in_vector <= to_std_logic_vector(p_in_str);
89
90              readline(stimulus, l); read(l, l_shf_str);
91              lut_vector <= to_std_logic_vector(l_shf_str);
92
93              wait for 50 ns;
94
95              -- clock in the P input
96              for i in 0 to (2**n)-1 loop
97                  p <= p_in_vector(i);
98                  wait for 10 ns;
99                  clk_p_in;
100             end loop;
101
102             -- clock in the "program" (lut functions)
103             for i in 0 to (((2**n)-1)*4)-1 loop
104                 l_in <= lut_vector(i);
105                 wait for 10 ns;
106                 clk_lut_in;
107             end loop;
108
109         end loop;
110
111         report "Test Complete" severity note;
112         wait;
113
114     end process;
115
116 end behavior;
```

Listing 12: Top Module Test Bench

13.2 VHDL Gate Modules with Delay

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity dffposx1_del is
5      generic(
6          delay : time := 203.5 ps
7      );
8      port(
9          clk : in std_logic;
10         d   : in std_logic;
11         q   : out std_logic
12     );
13 end dffposx1_del;
14
15 architecture rtl of dffposx1_del is begin
16     process(clk) begin
17         if rising_edge(clk) then
18             q <= d after delay;
19         end if;
20     end process;
21 end rtl;
```

Listing 13: DFF Module Delay

```
1  library ieee;
2  use ieee.std_logic_1164.all;
```

```
3
4  entity invx1_del is
5      generic(
6          delay : time := 162.7 ps
7      );
8      port(
9          a : in std_logic;
10         x : out std_logic
11     );
12 end invx1_del;
13
14 architecture rtl of invx1_del is begin
15     x <= not a after delay;
16 end rtl;
```

Listing 14: Inverter Module Delay

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux2x1_del is
5      generic(
6          delay : time := 315 ps
7      );
8      port(
9          b : in std_logic;
10         a : in std_logic;
11         s : in std_logic;
12         x : out std_logic
13     );
14 end mux2x1_del;
15
16 architecture rtl of mux2x1_del is begin
17
18     x <= not(b) after delay when (s = '0') else
19          not(a) after delay when (s = '1');
20
21 end rtl;
```

Listing 15: Inverter Module Delay

14 VHDL Modules Delay Waveforms

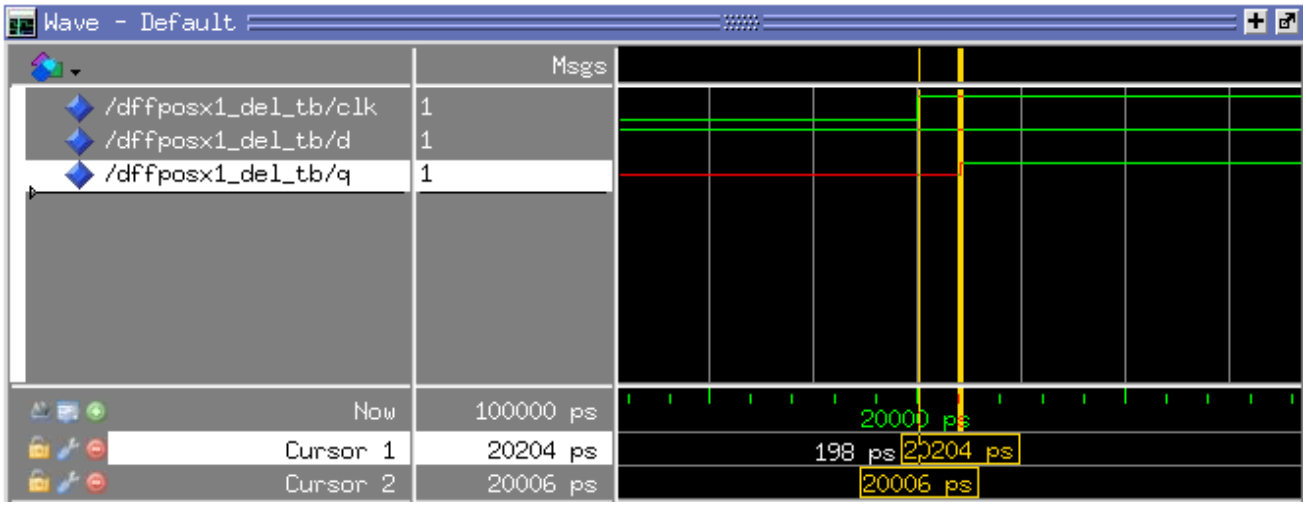

Figure 41: Delay Waveform of DFF

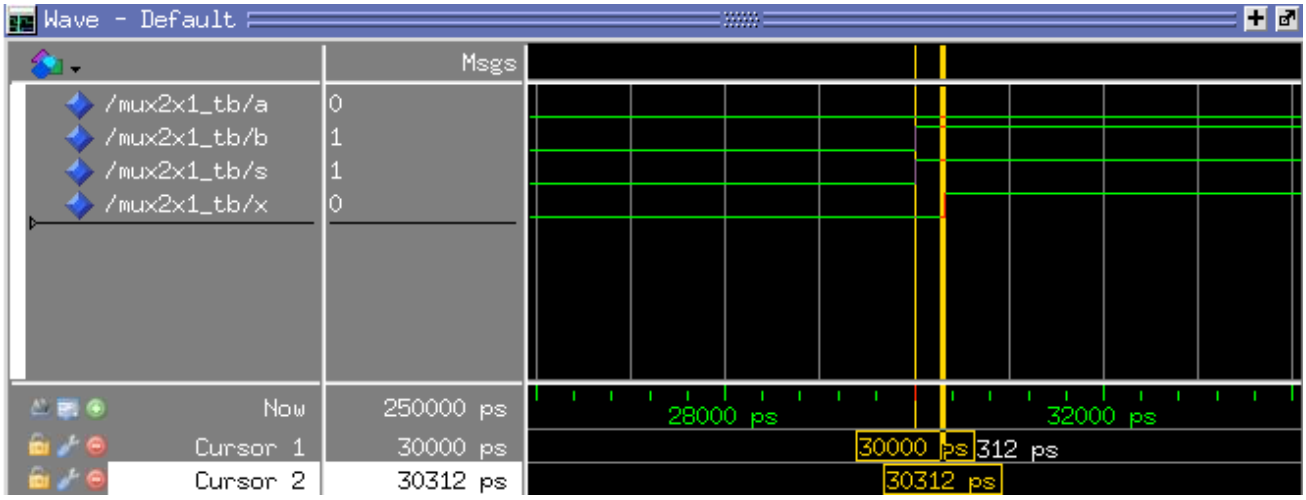

Figure 42: Delay Waveform of MUX

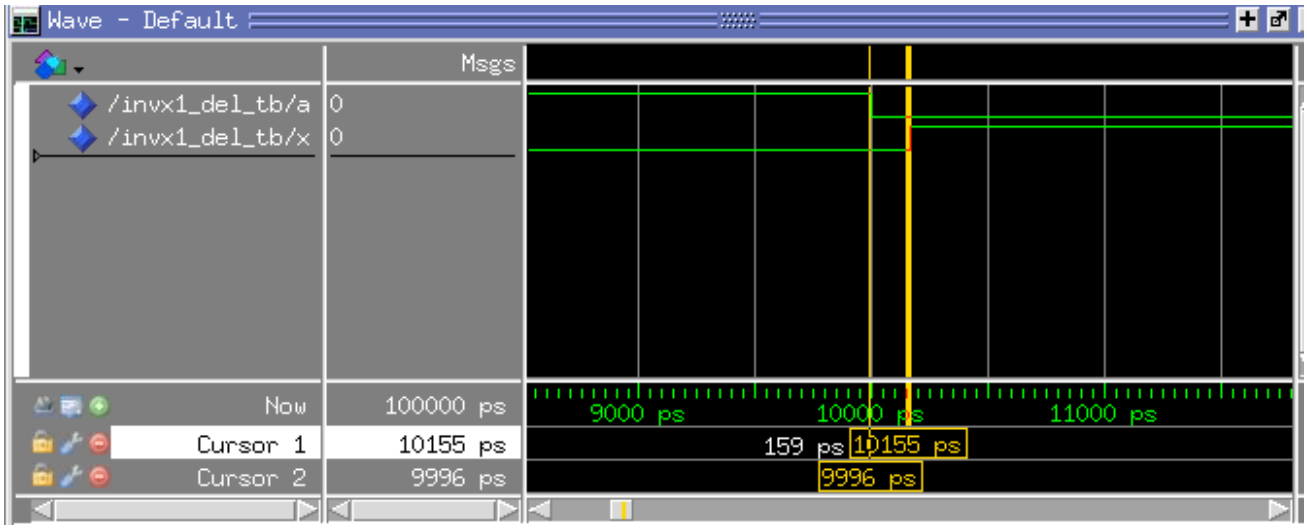

Figure 43: Delay Waveform of Inverter

In table 4, we compare the delay results for each leaf level gate among the three design simulations. Please look at the figures listed above. We can see that these compare very well with each other. Also, note that in order to calculate the total delay time of the LUT slice, we must add up the delay values found from four D flip flops, three muxes, and three inverters. After adding them up, our LUT slice circuit gets a delay time of approximately ~2 ns. Also, note that in order to get maximum clock rate, we just take the worst case delay of our circuit, and use this as a maximum clock rate. The table and figures above give us a worst case delay which is above the 50Mhz desirable signal.

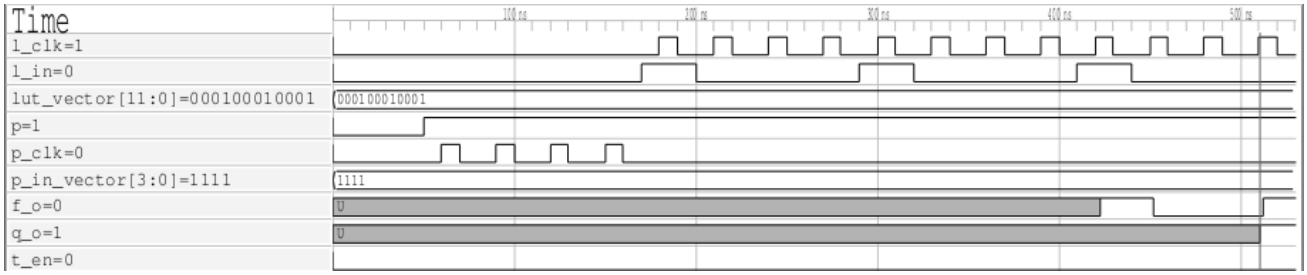

Figure 44: Top Level Delay Waveform

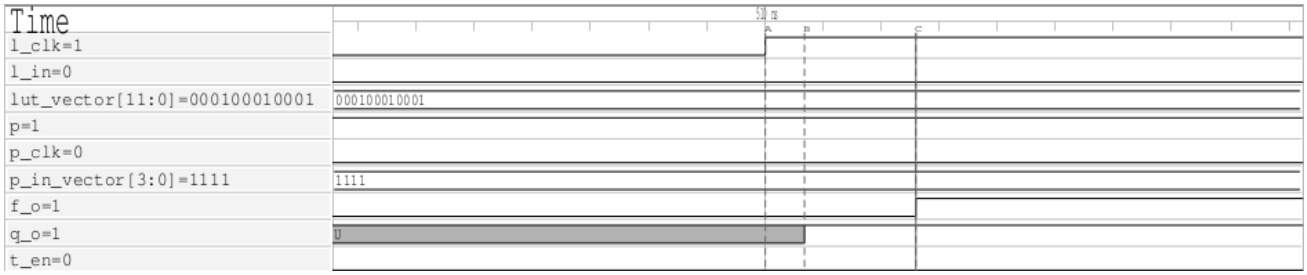

Figure 45: Top Level Delay Waveform Zoomed In

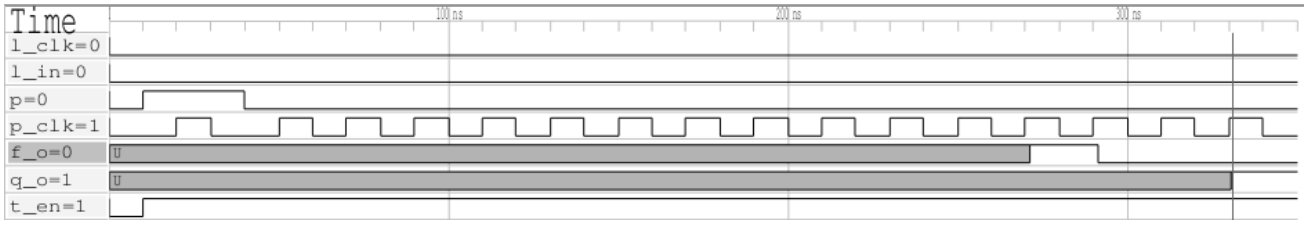

Figure 46: Top Level Test Mode Enabled Delay Waveform

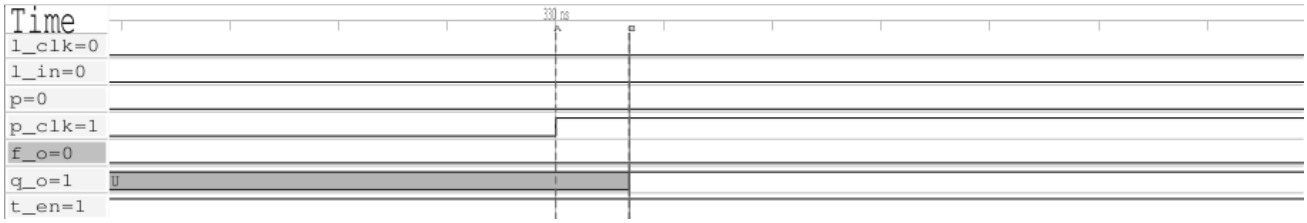

Figure 47: Top Level Test Mode Enabled Delay Waveform Zoomed In

As we can see here, the circuit still functions correctly given the delay times extracted from the simulations we have exhaustively gone through. This compares well with the previous simulation because as I have mentioned the functionality of the circuit still works properly. Also, we went ahead and measured the delay time for both the final Q and F output. In the above figure, the total delay time from A to B (Q output) was found to be 0.690 ns and from A to C (F output) was found to be ~ 2.6 ns (worst case delay). This tells us that maximum clock speed achieved can be 1/2.6ns ~384.6153846 MHz.

	IRSIM	**VHDL**	**Hspice**
LUT Slice	2.295 ns	2.6 ns	2.2375 ns
P Slice	0.255 ns	0.198 ns	0.20305 ns

Table 5: Bit Slice Worst Case Delay Times

15 Floor Plan

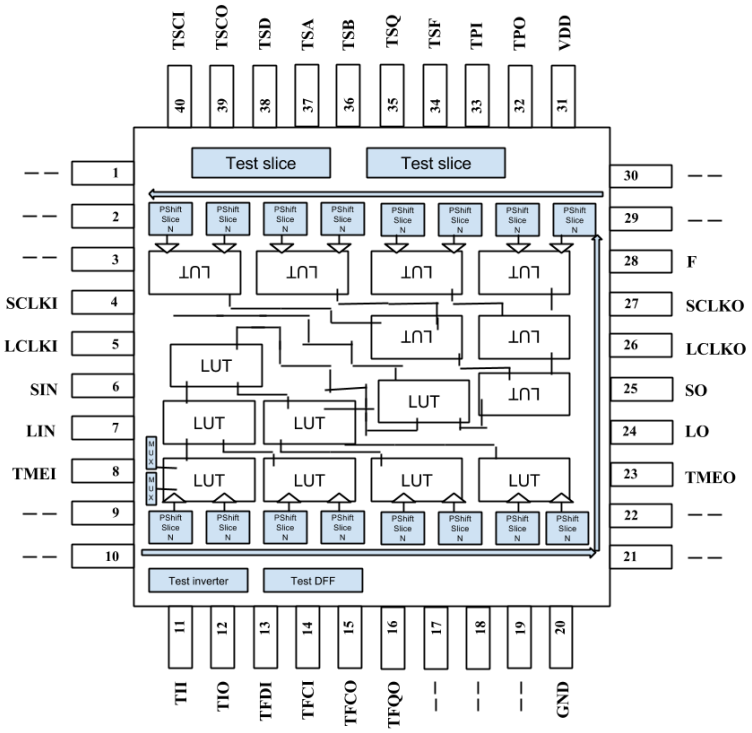

Figure 48: 32 Bit Floor Plan Design

This is the initial floor plan we designed. Note that in this design, we see a maximum of 16 inputs, this was just to show how it is to be designed in magic. In fact our actual design will consist of a maximum 32 inputs. Therefore, our design will have a total of 156 (32 + 31*4) D flip flops.

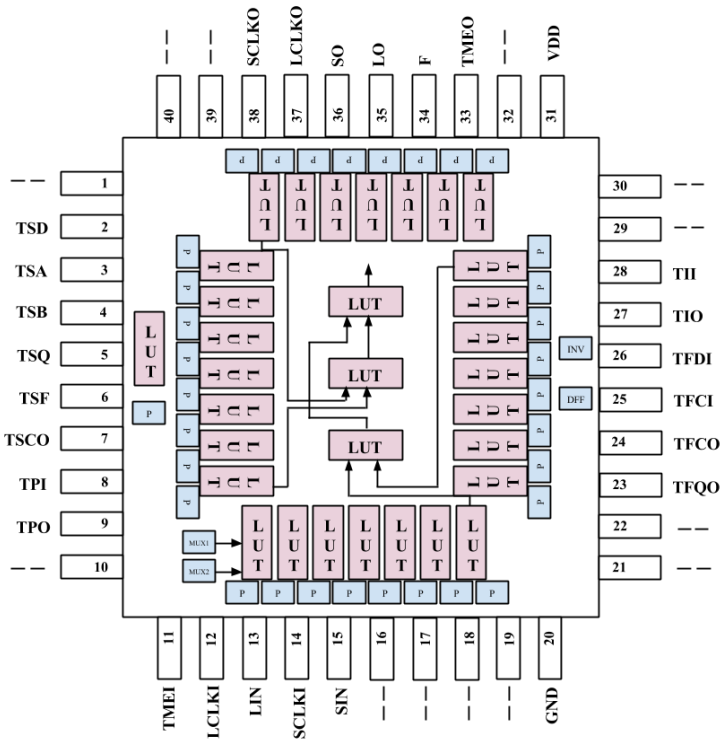

Figure 49: 64 Bit Floor Plan Design

This is the initial floor plan we designed for the 64 bit layout. Note that in this design, we see a maximum of 32 inputs, this was just to show how it is to be designed in magic. In fact our actual design will consist of a maximum 64 inputs. Therefore, our design will have a total of 316 (64 + 63*4) D flip flops. Not only does our design officially maximize N, but maximizes the total area used as well.

16 Major Design Decisions

We had to think of how to design our layout before we could come up with a floor plan. Since we have a binary tree, it was quite difficult to come up with an efficient way to utilize the whole area available given to us. At first, we thought we could fold the LUTs against each corner of the frame, but in doing so, led to a lot of wasted space in the middle. Therefore, we decided to go back and redesign the top level diagram of our circuit since constructing the layout in a binary tree manner would be very inefficient. The new top level design can be seen in Figure 4. The slices themselves are not changed, only the way these slices are arranged (layed out) have changed.

17 Work Division

Student	Task
Both	Modified Pin-out Diagram
Both	Magic Layout
Both	HSPICE
Both	IRSIM
Both	VHDL
Both	Floor Plan

Table 6: Task Assignment

18 Chip Level Layout

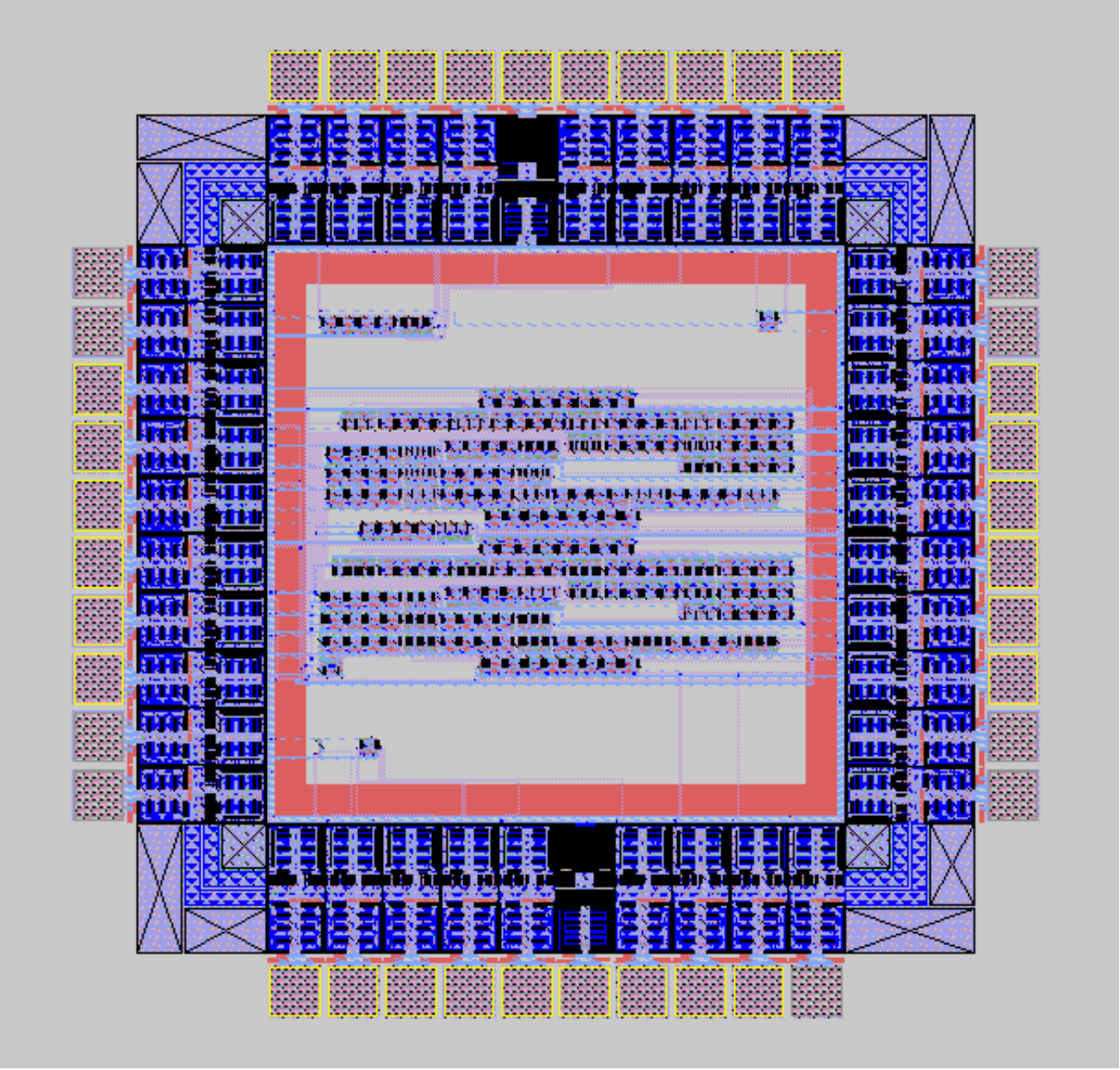

Figure 50: 32 Bit Layout - Magic

19 Users Guide

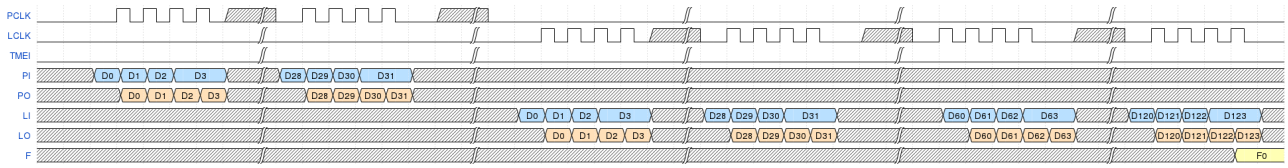

Figure 51: Cycle Timing Diagram for Normal Mode

In normal mode, the user will need to leave the TMEI signal low. The user will then need to give data inputs to both PI and LI. Since, LI has 4 times the number of bits that P has, LCLK should then be triggered for 4 times longer than PCLK. The final computation is shown to be on the very last clock cycle. Also, it does not matter whether the P input or LUT inputs are loaded in first, the functionality of the circuit will still remain working properly. Note to the user, in this timing diagram, we are showing that P input is loaded in first.

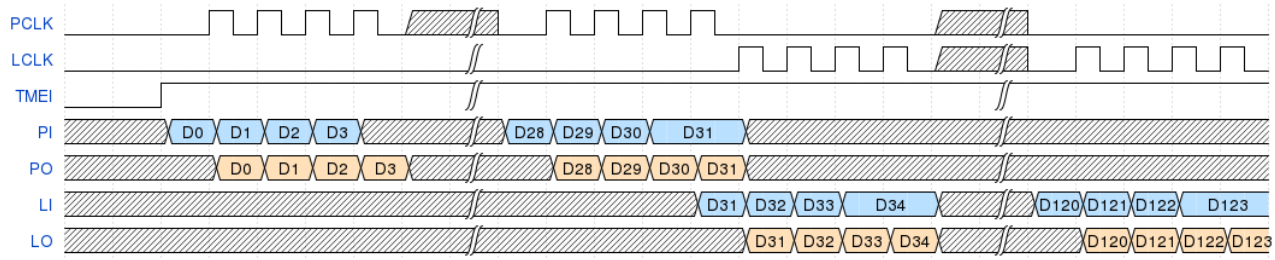

Figure 52: Cycle Timing Diagram for Test Mode Enable

In test mode, we are connecting all the D-Flip Flops together to make sure all the flip flops are shifting properly. This scheme is known as a scan chain. Here we can see that when TMEI is high, we are in test mode. The output of the last clock signal and data output of P will be connected to the input of the LCLK and LI, respectively. Since the architecture contains 32 bits for the P input and 124 (31*4) bits for the LUT, we will have to shift in a vector of 156 bits (124+32) in order to see the output of the scan chain. Only then can the user monitor the output to verify that all the shift registers are working as intended. Note that F (the final computation) is not shown in this timing diagram, because the user is not concerned with the functionality of the LUTs in this mode–only monitoring the shift registers.

The user will need to set TMEI high, then he/she will have to give the data inputs to PI and make sure to have a total of 156 clock cycles for PCLK, the rest is taken care of by the hardware (the user will not have to give any inputs to the LUT).

20 Test Strategy

We have a few methods for testing to ensure that the chip is working correctly. The first method we use for testing purposes is to connect all the D-Flip Flops together to create the initial scan chain. This will allow the user to make sure all of the shift registers are working properly. Another method, is to have individual bit slice designs completely independent of the circuit itself. Therefore, we have the Look Up Table Slice and the P-shift register slice completely independent of the circuit. The third method consists of having separate standard slices, both an inverter and a D-Flip Flop completely independent of the circuit as well. The final method consists of monitoring certain signals within each cell. Other testing strategies include looking at the clock output signals and making sure that the signal follows the input clock signals. If the user ensures these signals are not lagging, then it's safe to conclude that the signal strength provided by the buffers are sufficient enough to overcome the capacitance.

In the first method, the user will only need to pull the TMEI pin high.

In the second method, the user will need to feed the individual bit slice inputs as noted:

LUT slice:

TSCI (test slice clock input) is the input to the clock line.
TSD (test slice D input) is the input to the LUT, the user will shift in the combinational function he/she desires to test.
TSA and TSB (test slice A and B) are inputs and act as the address lines into the LUT, the user should feed these inputs after "programming" the LUT.
TSQ is the output of the LUT, the user can probe this pin to make sure everything that was shifted in is correct.
F is the output of the final computation the LUT slice performs. The user can can probe this pin to monitor that the final computation is working as intended.
TSCO (test slice clock output) is the output to the clock line.

P Shift register Slice:

TSCI (test slice clock input) is the input to the clock line.
TSPI (test slice p input) is the input to the signal P.
TSPO (test slice p output) is the output signal of P.

The third method consists of two separate standard cells, the inverter and the DFF:

Inverter:

TII (test inverter input) is the input to the inverter, the user will give this pin any value.
TIO (test inverter output) is the output to the inverter, the user will probe this pin to monitor the output.

D-Flip Flop:

TFCI (test flip flop clock input) is the input to the test clock line, the user will give this pin the clock signal.
TFDI (test flip flop D input) is the input to the D-Flip Flop, the user will give this pin an input value to be monitored.
TFQO (test flip flop Q output) is the output of the D-Flip Flop, the user will use this pin to monitor the output.
TFCO (test flip flop clock output) is the output of the clock line.

For the last and final method, we have certain pins used to monitor certain various areas within the circuit. The user should use the Pin description to see each output pin needed to test to ensure the functionality of the circuit is performing correctly.

21 Descrption of Chip Architecture

The PBCTKS architecture uses a 32-bit wide P register and a 124-bit wide LUT register for programming mode. This allows the user to program up to 2**32 different combinations for given inputs and 2**(124/4) different combinations to program the functionality of the chip. This chip consists of ~31 I/O pins which include test pins for testing purposes, and includes a pin to set the chip in normal or testing mode.

22 Major Design Decisions

We layed everything out in magic, and noticed that we would only be using about 60% of chip area. This was just unacceptable, so we needed to come up with a different layout design where we would be able utilize more of the chip area. We decided to design how we needed to rearrange the slices in a more compact fashion. In Figure 49, we can see the method of the layout that would have been used in Magic.

Unfortunately, we were very constrained with time, and connecting the VDD and GND lines became exponentially complicated that we had to stop and revert back to the 32 bit design. The incomplete 64 bit design can be shown below.

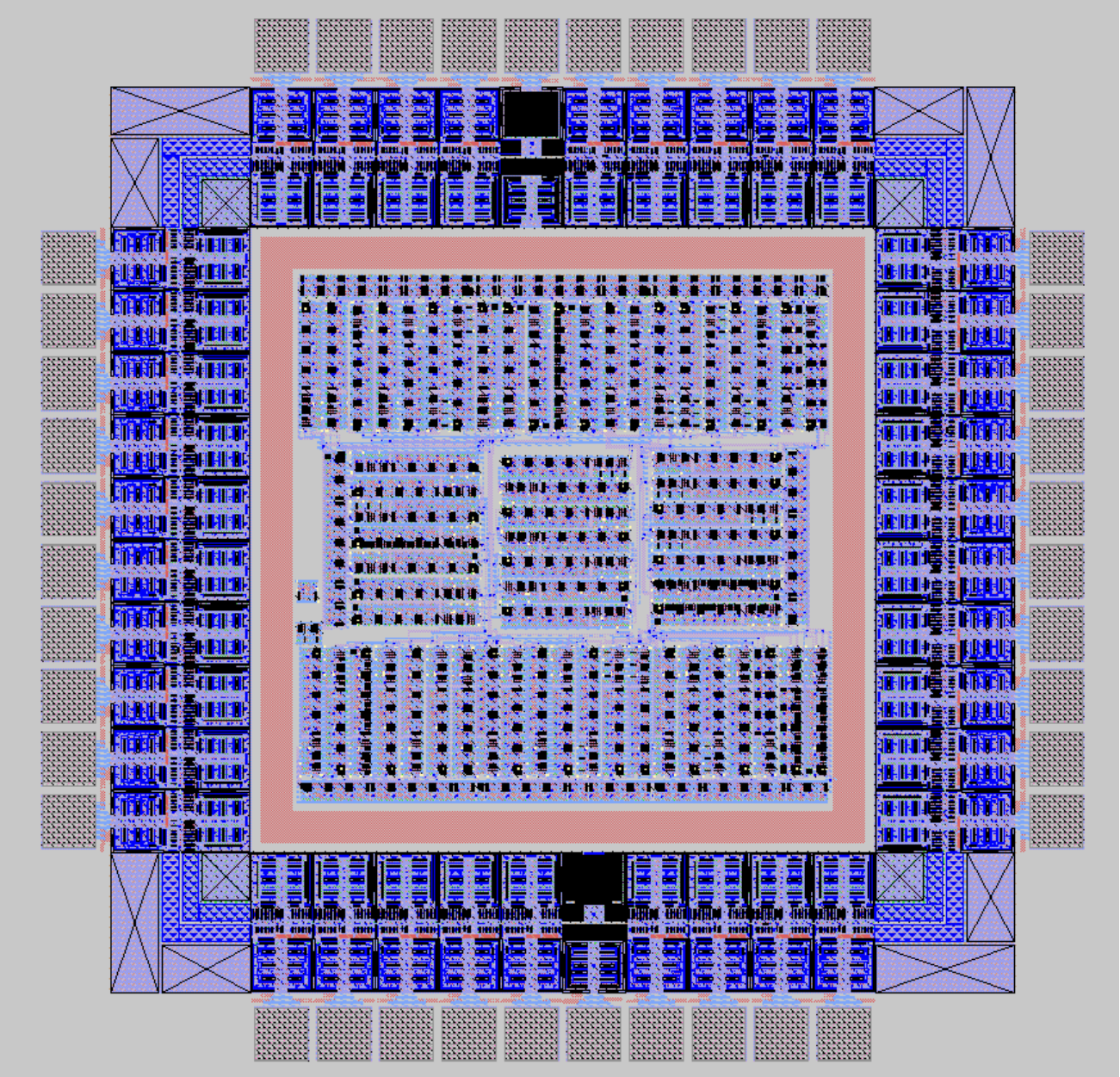

Figure 53: 64 Bit Layout in Magic

Here we can see that we maximized not only N, but the total area that was given to us. But as I have mentioned previously, due to wiring complexity and timing constraints, we were not able to complete this on time and had to revert back to the original 32 bit design.

DFFPOSX1	Capacitance (pF)
CLK	~0.05
D	~0.0157

Table 7: Capacitance Values for D-Flip-Flop

BUFFX4	Capacitance (pF)	Cap Output Drive Strength (pF)
A	~0.0322	1.85
Y	~0.0322	

Table 8: Capacitance Value and Output Drive Strength for Buffer

Since the clock signal was global and driving all of the slices, we needed to accommodate for the capacitance. In the table above we can see that the capacitance of the D-Flip Flop was found to be 0.05pF. Therefore, we

knew that this capacitance would add up over all of the bit slices, so we needed to make sure to add buffers when needed. The table above shows the output drive strength for the given buffer. We then used this to calculate the maximum buffers that could be used: 1.85pF / 0.05pF ~37. Therefore, we ended up just putting a buffer every 32 DFFs (leaving 5 for error).

23 IRSIM Simulations

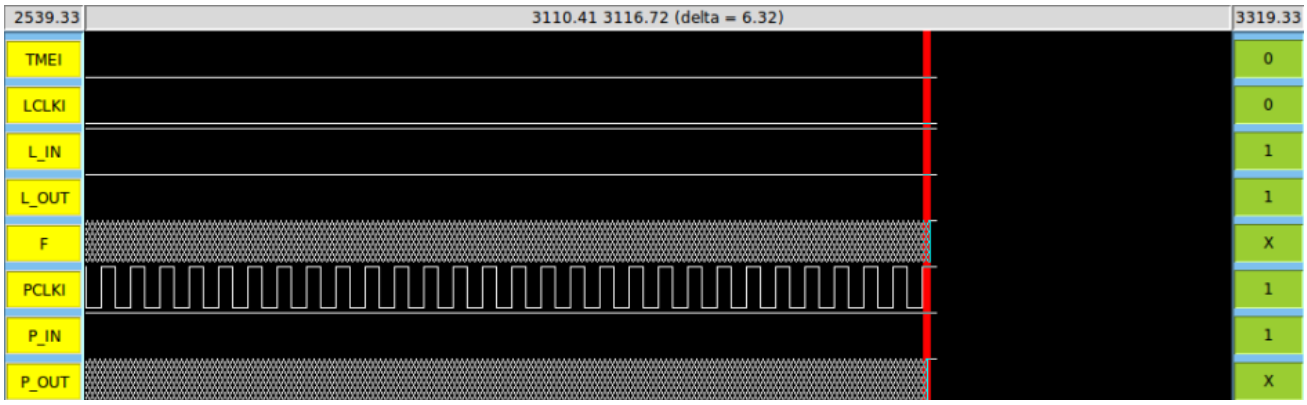

Figure 54: Top Delay Normal Mode

We can see the worst case delay time found in IRSIM shown in the above picture. This is very consistent with what we have seen in our VHDL design.

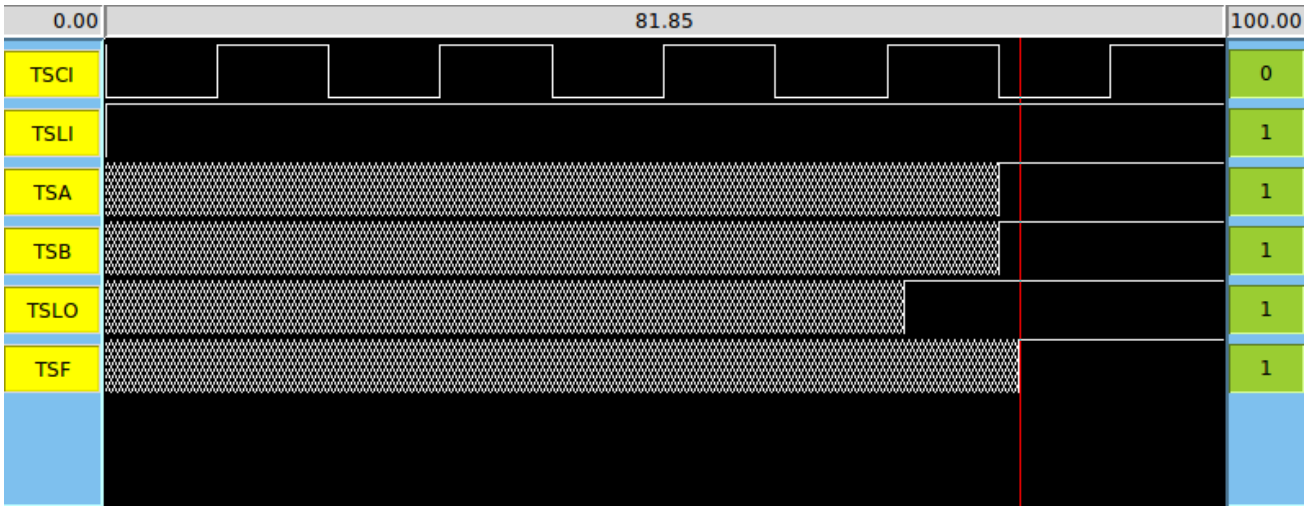

Figure 55: Test LUT Slice

We can see that the independent LUT slice works correctly.

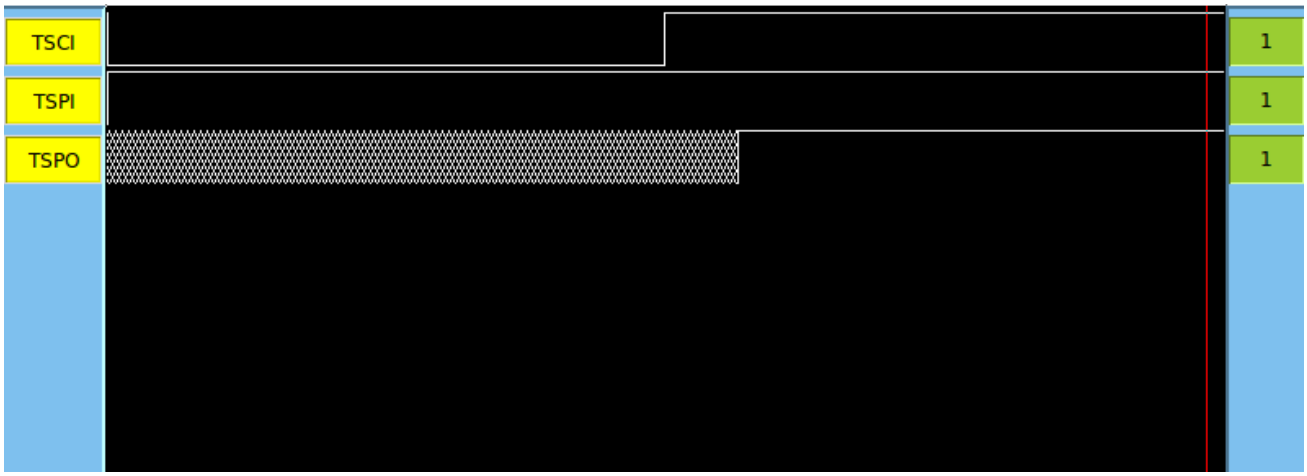

Figure 56: Test P Shift Register Slice

We can see that the independent P Shift Register slice works correctly.

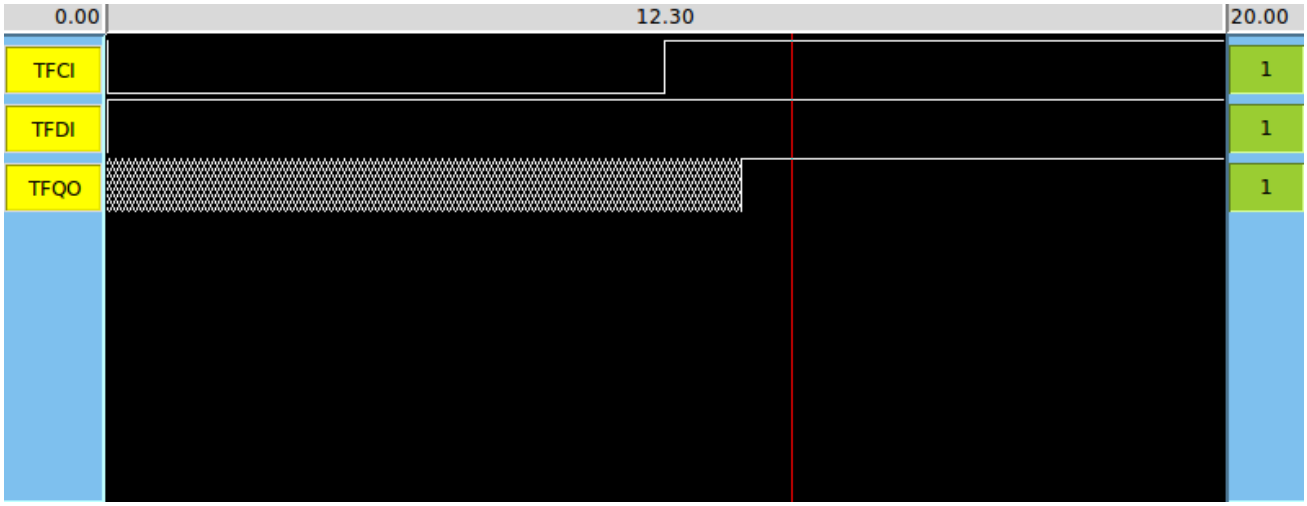

Figure 57: Test DFF Slice

We can see that the independent DFF slice works correctly.

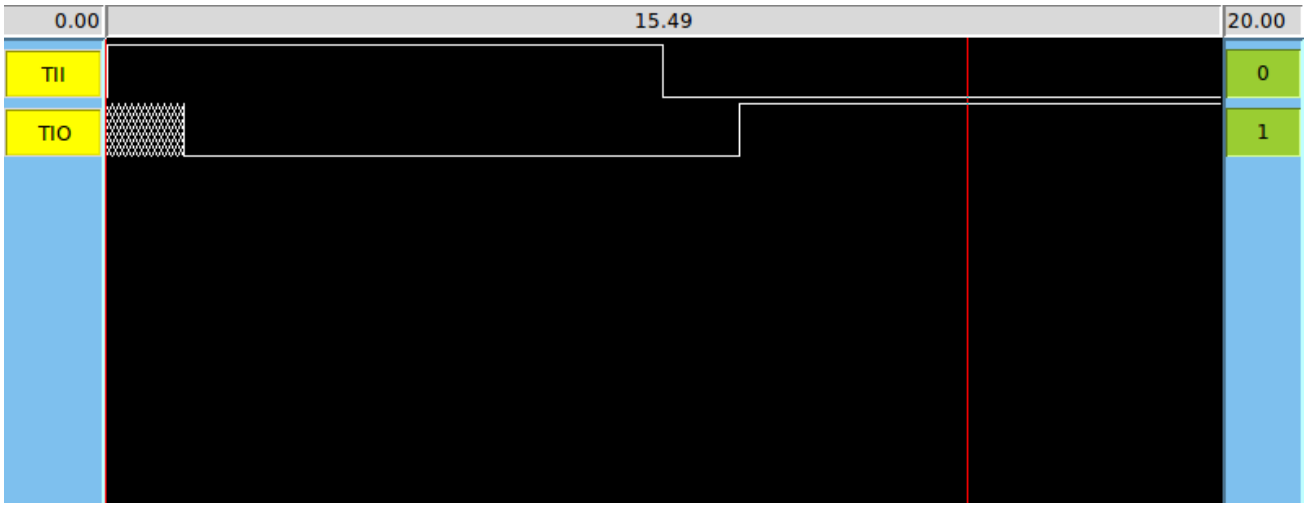

Figure 58: Inverter

We can see that the independent Inverter works correctly.

24 VHDL Simulations

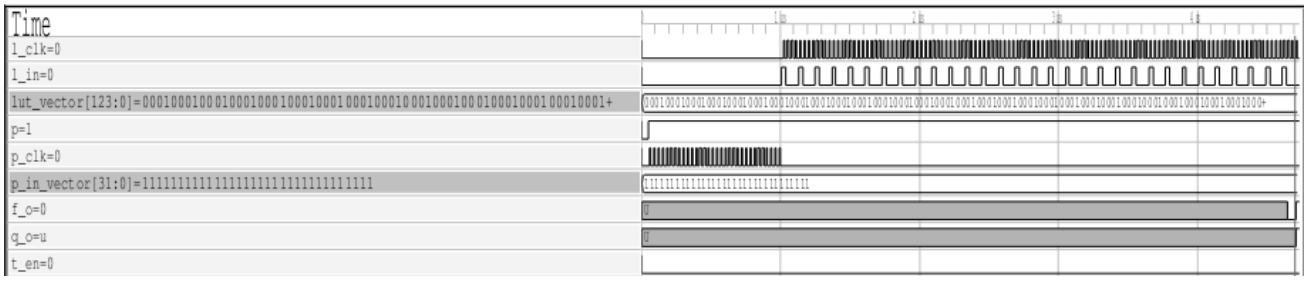

Figure 59: Top 32-bit Normal Mode

Figure 60: Top 32-bit Test Mode

Figure 61: Top Delay 32-bit Normal Mode

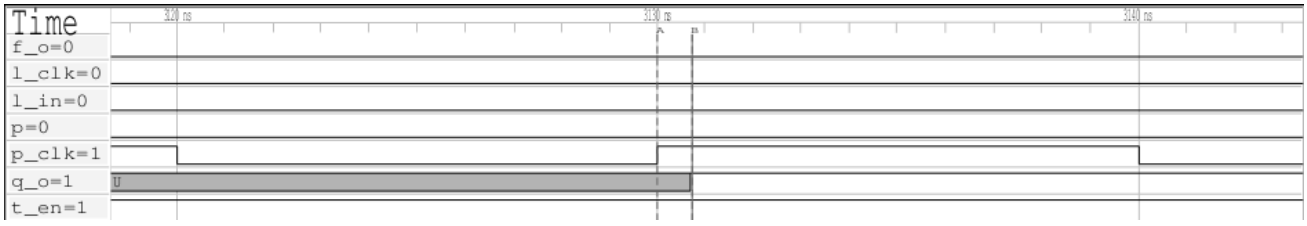

Figure 62: Top Delay 32-bit Test Mode

	VHDL	IRSIM
F	5.470 ns	6.32 ns

Table 9: Total Delay Times for Normal Mode between VHDL and IRSIM

F is the final output computation for the Look Up Table (this was our worst case delay). We can see here that the times are fairly close to one another. Note this is the total delay, but keep in mind that the delay per slice for worst case delay was found to be 2.6ns. So our Maximum clock frequency is found to be 1/2.6ns, approximately 385 MHz.

25 Simulation Strategy

It was much easier to use Python to write to the cmd file, and then check the output log file if anything failed. It would be too difficult to give a 124 bit vector manually, so we used Python to feed the signals for both in VHDL and IRSIM. Also, since we want to make sure things works as intended, the best method was to just shift in all 1's for all the DFF, and monitor the outputs accordingly. This gave us an idea if something was being shorted out

or something was not shifting out properly. Then we monitored each node for debugging purposes to ensure that each bit was being shifted out from one flip flop to the next. A figure below shows the method used.

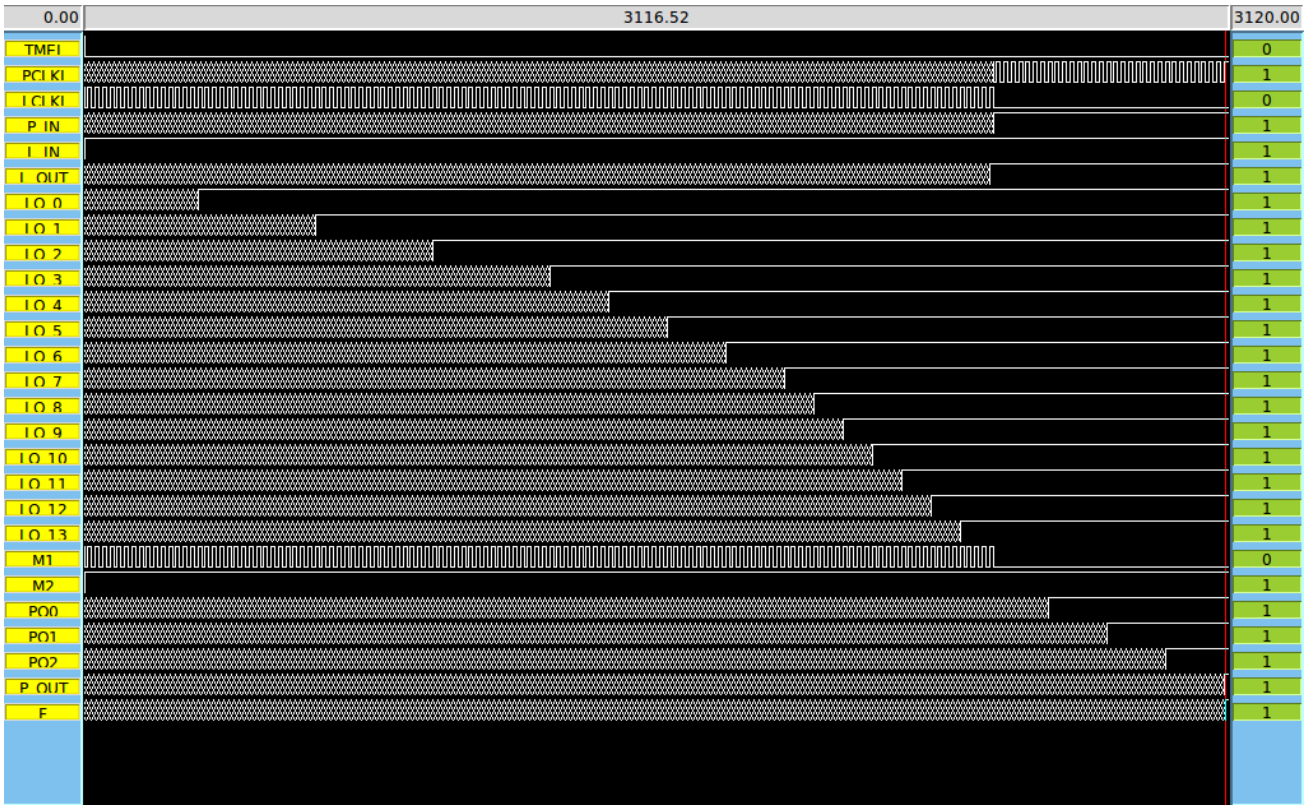

Figure 63: Debug Mode in IRSIM

26 Work Division

Student	Task
Both	Modified Pin-out Diagram
Both	Magic Layout
Both	IRSIM
Both	VHDL
Both	Modified Floor Plan

Table 10: Task Assignment