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 4, 2013

Contents

List of Figures

List of Tables

Listings

1 Pinout Diagram

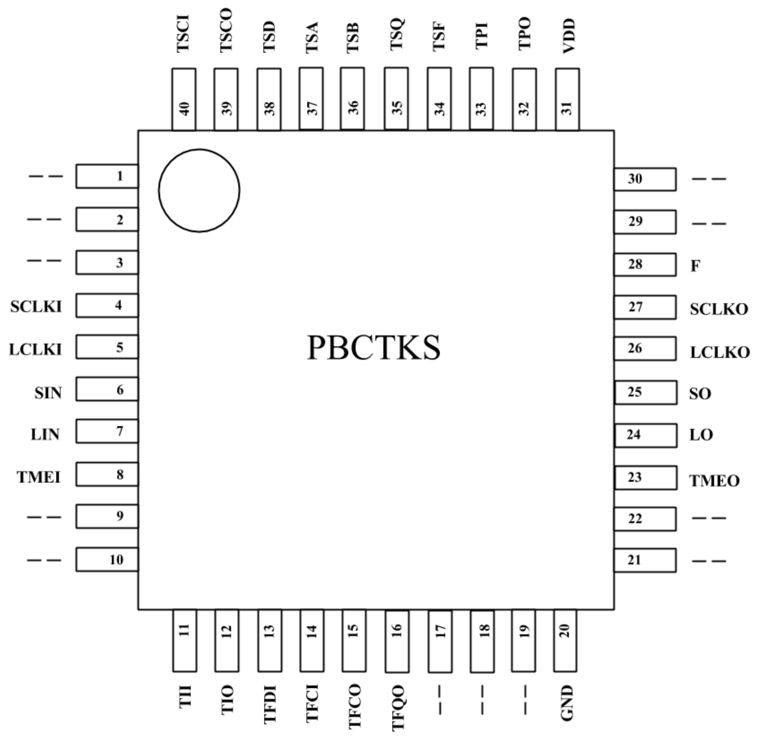

Figure 1: Pinout Diagram of the PBTCKS Chip

1.1 Pinout Description

Function	Pin #	I/O	Description
–	1	–	–
–	2	–	–
–	3	–	–
SCLKI	4	I	Shift Register Clock Input
LCLKI	5	I	LUT Shift Register Clock Input
SIN	6	I	Shift Register Input (P 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
TFQO	16	O	Test Flip-Flop Q Output
–	17	–	–
–	18	–	–
–	19	–	–
GND	20	–	Ground Reference for I/O Pins
–	21	–	–
–	22	–	–
TMEO	23	O	Test Mode Enabled Output
LO	24	O	Shift Register Output of P input
SO	25	O	Shift Register Output of LUT
LCLKO	26	O	LUT Shift Register Clock Output
SCLKO	27	O	Shift Register Clock Output
F	28	O	Output of Computation of LUT
–	29	–	–
–	30	–	–
VDD	31	I	Test LUT Slice B Input
TPO	32	O	Test P Slice Output
TPI	33	I	TesT P Slice Input
TSF	34	O	Test LUT Slice Mux Output
TSQ	35	O	Test LUT Slice Shift Register Output
TSB	36	I	Test LUT Slice B Address Input
TSA	37	I	Test LUT Slice A Address Input
TSD	38	I	Test LUT Slice Shift Register Input
TSCO	39	O	Test LUT Slice Clock Output
TSCI	40	I	Test LUT 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 O. 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, O. 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 LUT, 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 work, they use LUT 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

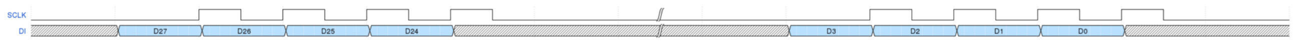

Figure 2: Cycle Timing Diagram for LUT Shift Register

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 revers order. For example, if we want to perform an AND operation, we have to feed the data bits in like so: 1000.

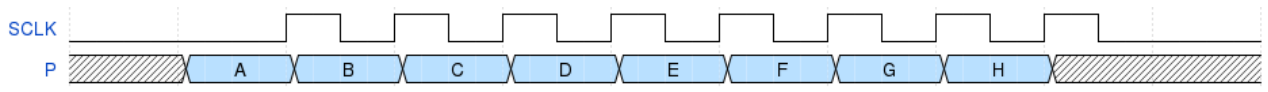

Figure 3: Cycle Timing Diagram for P Input Shift Register

As described from above the user has to just feed in input P, but in reverse 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

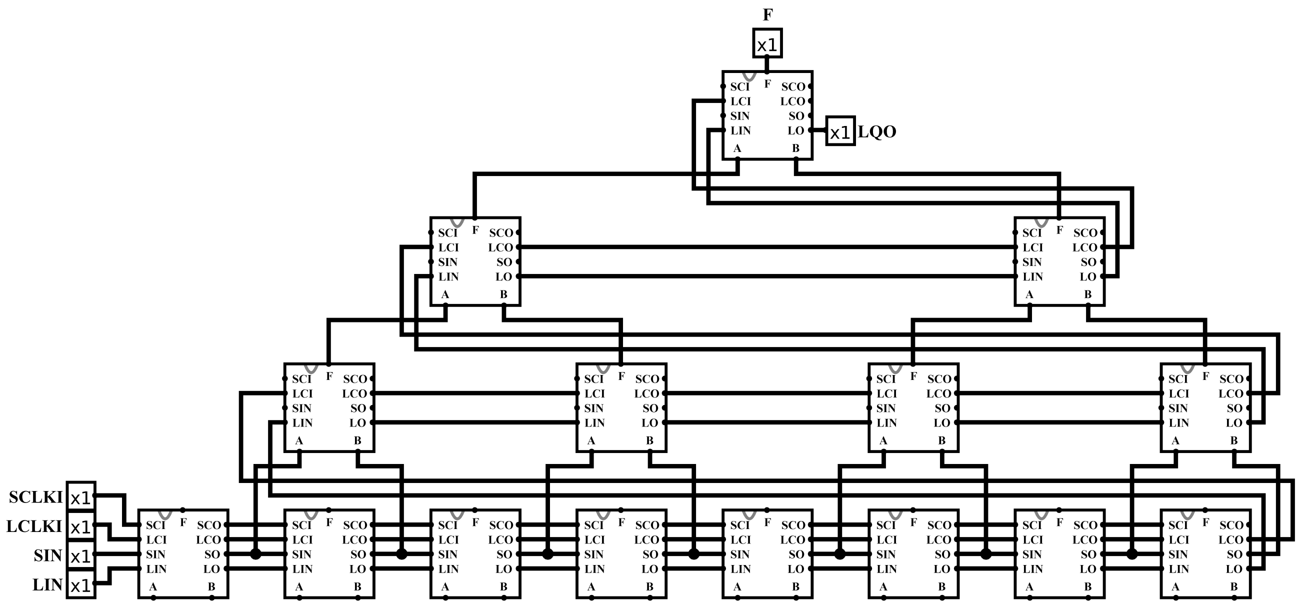

Figure 4: Hierarchical Design in Logisim

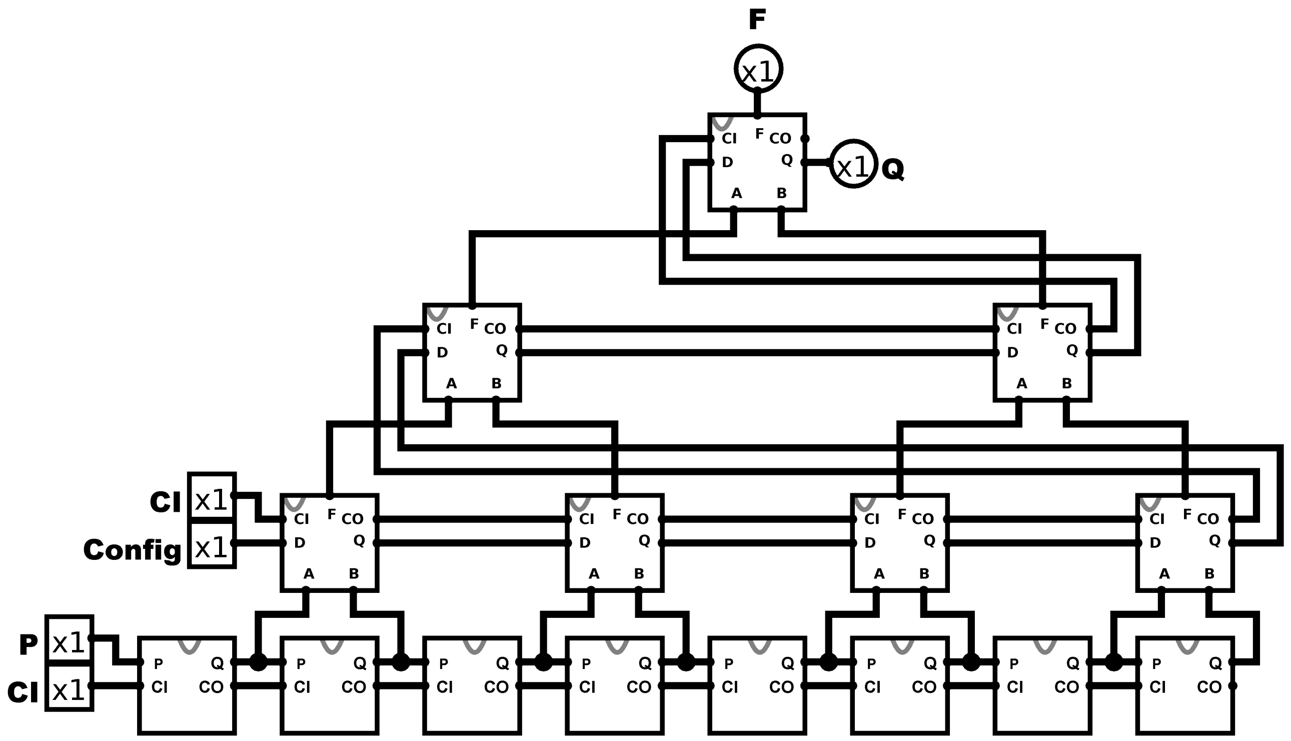

Figure 5: Top Level Diagram (8 Inputs, 28-bit slice)

We can see that the slice design of a 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 vector of 28 bits wide.

5.2 Bit Slice Design Scheme

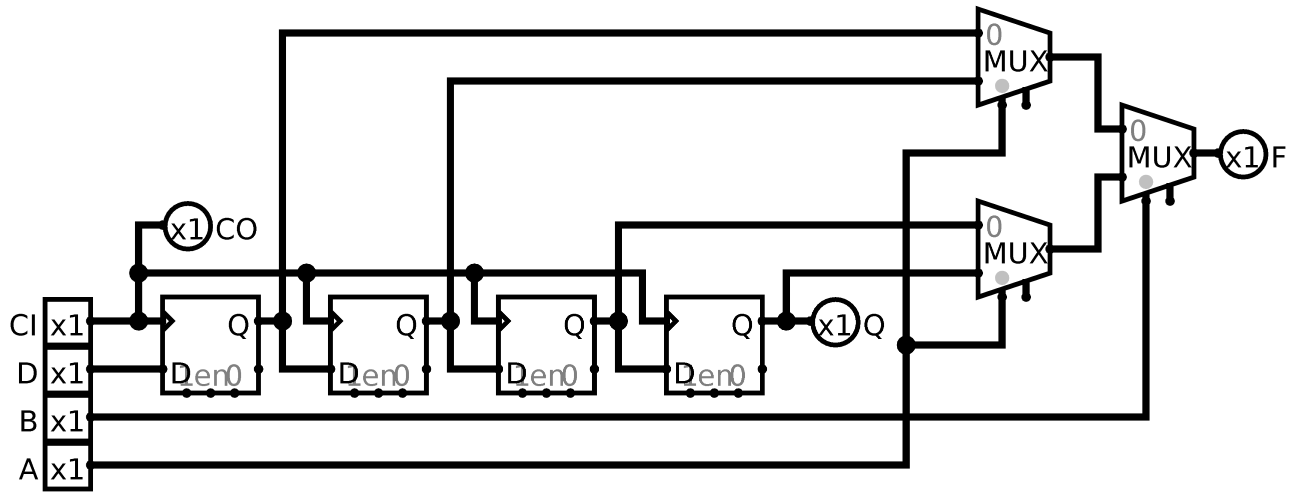

Figure 6: LUT Slice in Logisim

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.

5.3 Top Level Test Mode Diagram

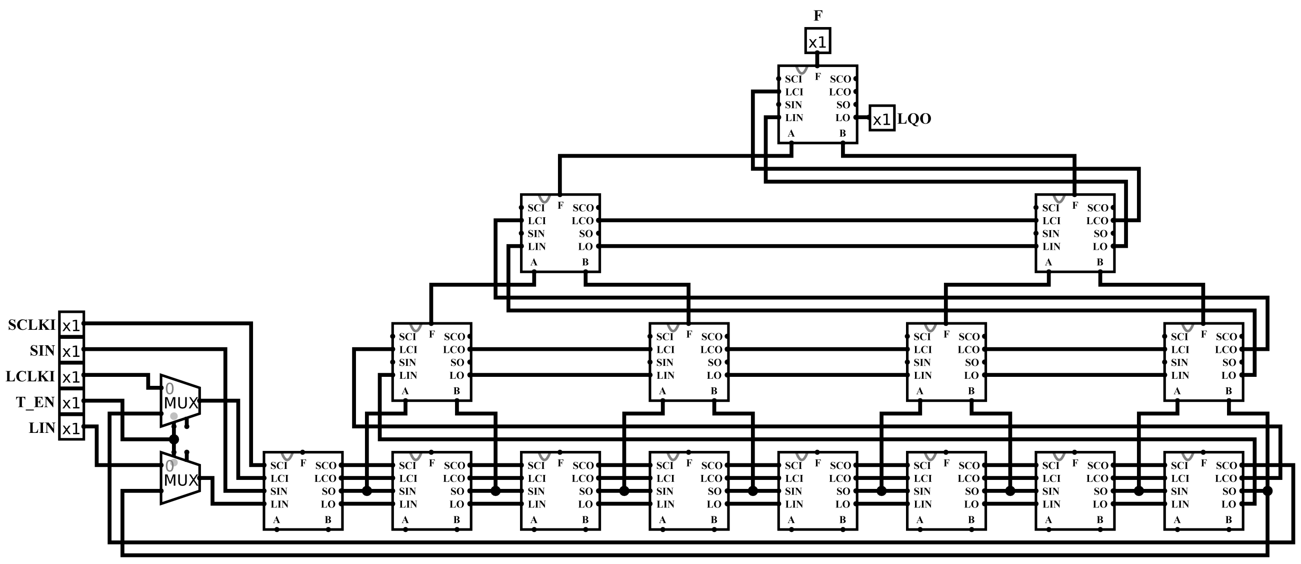

Figure 7: Hierarchical Test Design in Logisim

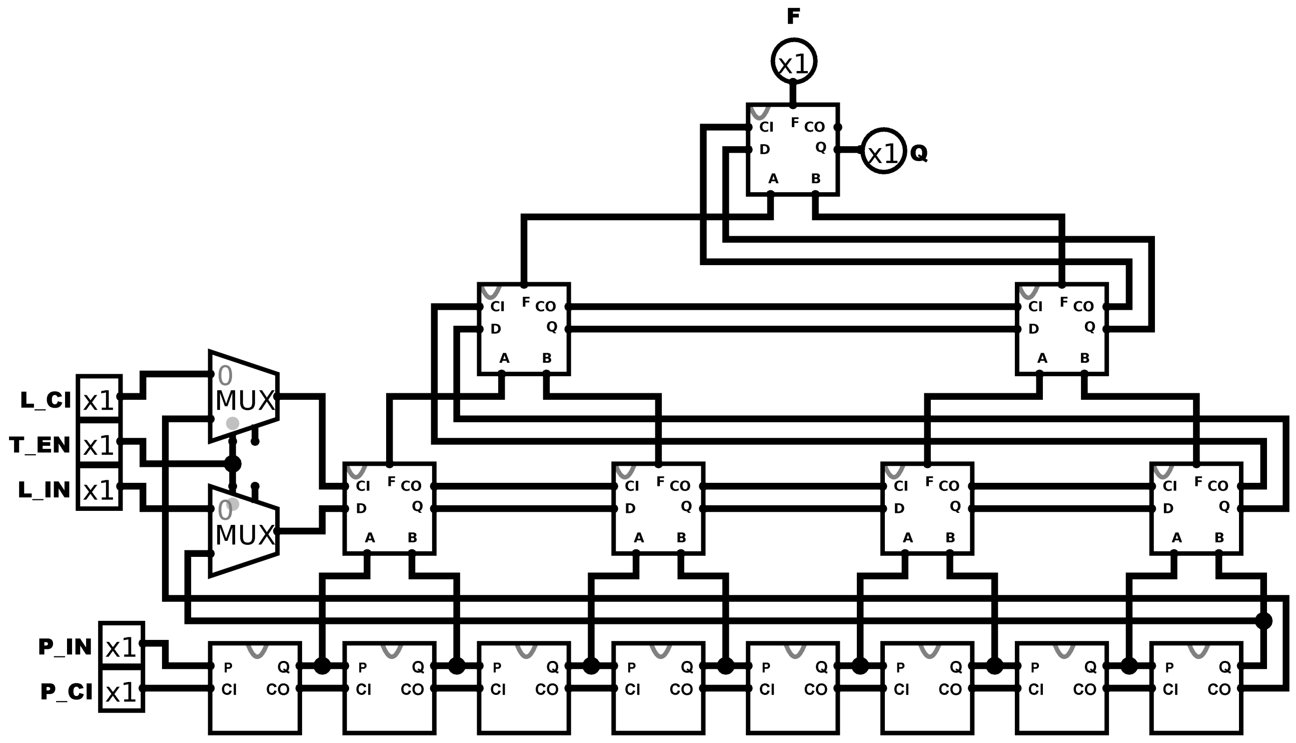

Figure 8: Top Level Test Mode Diagram (8 Inputs, 28-bit slice)

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

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

entity top 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;

architecture rtl of top 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 port map(l_clk, p_clk, t_en, shift_clki);
    t_mux_2 : entity work.mux2x1 port map(l_in,  p_out, t_en, l_shf_ini);

    t_inv_1 : entity work.invx1  port map(shift_clki, shift_clk);
    t_inv_2 : entity work.invx1  port map(l_shf_ini, l_shf_in);

    lut_1 : entity work.lut
    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 1: Top Module

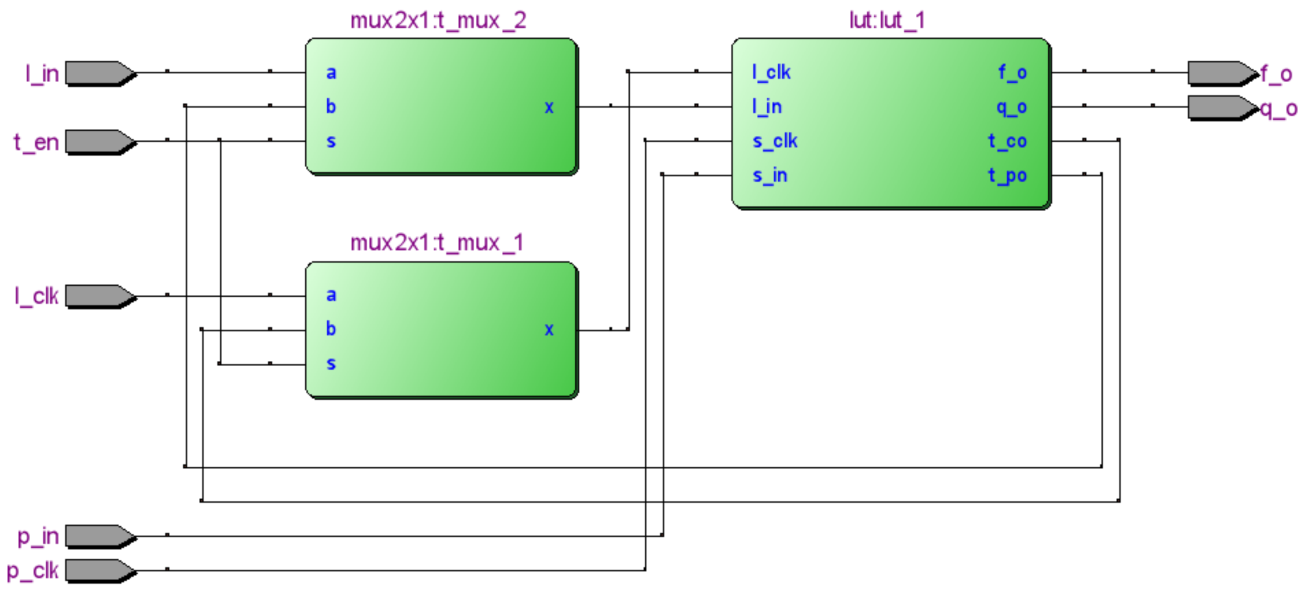

Figure 9: RTL Design of Top Level

6.2 Slice Modules

6.2.1 LUT

```vhdl
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;

    -- lut shift out is output from prev ff
    q <= ff_o(4);
    ff_o(0) <= d;

    -- select first two outputs of LUT
    -- on sel line A
    mux1 : mux2x1 port map(ff_o(1),  ff_o(2), a, mux_o(0));
    inv1 : invx1  port map(mux_o(0), mux_fo(0));

    -- select last two outputs of LUT
    -- on sel line A
    mux2 : mux2x1 port map(ff_o(3),  ff_o(4), a, mux_o(1));
    inv2 : invx1  port map(mux_o(1), mux_fo(1));

    -- select the outputs from
    -- each mux on sel line B
    mux3 : mux2x1 port map(mux_fo(0), mux_fo(1), b, f_muxo);

    -- invert mux due to func of mux: y=!(S?(A:B))
    inv3 : invx1 port map(f_muxo, f);

end rtl;
```

Listing 2: Lookup Table Slice Module

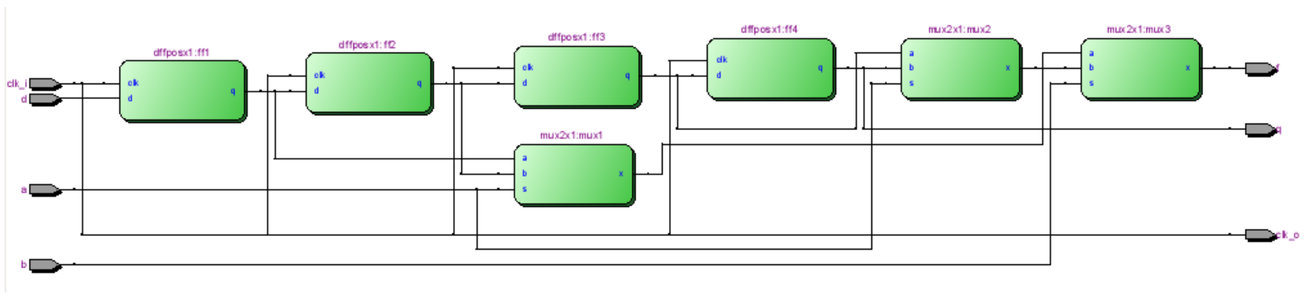

Figure 10: RTL Design of LUT Slice

6.2.2 Shift Register

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

entity shift_slice is
    port(
        clk_i   : in std_logic;
        p       : in std_logic;
        clk_o   : out std_logic;
        q       : out std_logic
    );
end shift_slice;

architecture rtl of shift_slice is

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

begin
```

```vhdl

    ff_p1 : dffposx1
    port map(
        clk => clk_i,
        d   => p,
        q   => q
    );

    clk_o <= clk_i;

end rtl;
```

Listing 3: Lookup Table Slice Module

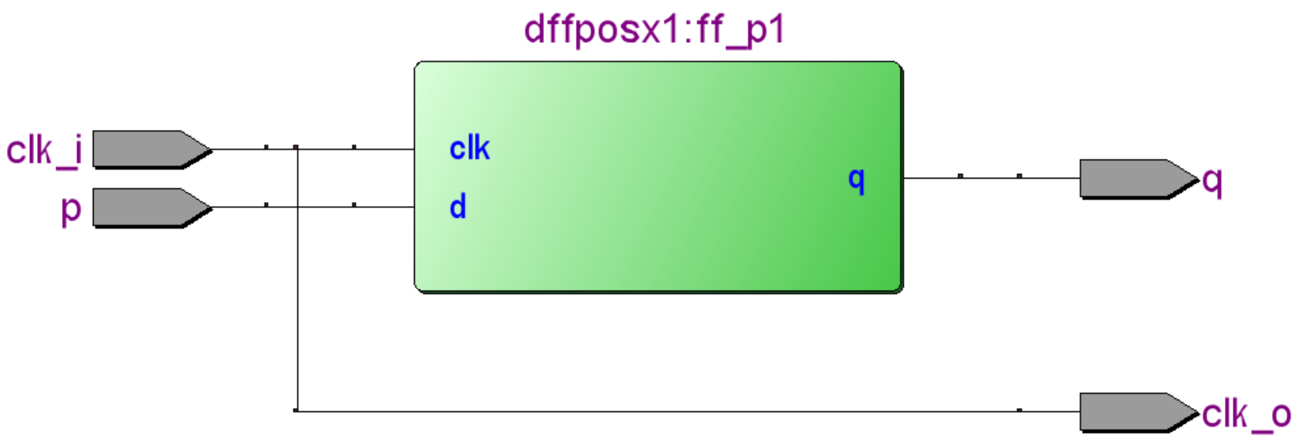

Figure 11: RTL Design of Shift Register Slice

6.3 Gates

6.3.1 D-Flip Flop

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

entity dffposx1 is
    generic(
        delay : time := 0 ps
    );
    port(
        clk : in std_logic;
        d   : in std_logic;
        q   : out std_logic
    );
end dffposx1;

architecture rtl of dffposx1 is begin
    process(clk) begin
        if rising_edge(clk) then
            q <= d after delay;
        end if;
    end process;
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,
        f       => f
    );

    process

        procedure clock is begin
            clk <= '1';
            wait for 10 ns;
            clk <= '0';
            wait for 10 ns;
        end procedure clock;

    begin

        wait for 10 ns;
        a   <= '1';
        b   <= '1';

        -- lut for AND gate
        di <= '1';
        wait for 10 ns;
        clock;

        di  <= '0';
        wait for 10 ns;
        clock;

        di  <= '0';
        wait for 10 ns;
        clock;

        di  <= '0';
        wait for 10 ns;
        clock;

        wait;

    end process;

end behavior;
```

Listing 6: LUT Slice Test Bench Module

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.txt_util.all;

entity top_tb is
    generic(
        --stim_file : string := "test_tree.sim"
        stim_file : string := "two_tree.sim"
    );
end top_tb;

architecture behavior of top_tb is

    constant n : integer := 2;

    signal p_clk : std_logic := '0';
    signal l_clk : std_logic := '0';
    signal p     : std_logic := '0';
    signal l_in  : std_logic := '0';
    signal f_o    : std_logic;
    signal q_o    : std_logic;
    signal t_en   : std_logic := '0';

    signal p_in_vector  : std_logic_vector((2**n)-1          downto 0);
    signal lut_vector   : std_logic_vector((((2**n)-1)*4)-1 downto 0);

    file stimulus : TEXT open read_mode is stim_file;

    component top
        generic(
           n       : integer := 3    -- number of levels in tree
        );
        port(
            p_clk  : in std_logic;   -- p 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   -- lut shift register output
        );
    end component;

begin

    dut : top
    generic map(
        n      => n
    )
    port map(
        p_clk => p_clk,
        l_clk => l_clk,
        p_in  => p,
        l_in  => l_in,
        t_en  => t_en,
        f_o   => f_o,
        q_o   => q_o
    );

    process

        procedure clk_p_in is begin
            p_clk <= '1';
            wait for 10 ns;
            p_clk <= '0';
            wait for 10 ns;
        end procedure clk_p_in;

        procedure clk_lut_in is begin
            l_clk <= '1';
            wait for 10 ns;
            l_clk <= '0';
            wait for 10 ns;
        end procedure clk_lut_in;

        variable l: line;
        variable p_in_str  : string(1 to 2**n);
        variable l_shf_str: string(1 to ((2**n)-1)*4);

    begin

        while not endfile(stimulus) loop

```

```
            -- load stimulus for this test
            readline(stimulus, l); read(l, p_in_str);
            p_in_vector <= to_std_logic_vector(p_in_str);

            readline(stimulus, l); read(l, l_shf_str);
            lut_vector <= to_std_logic_vector(l_shf_str);

            wait for 50 ns;

            -- clock in the P input
            for i in 0 to (2**n)-1 loop
                p <= p_in_vector(i);
                wait for 10 ns;
                clk_p_in;
            end loop;

            -- clock in the "program" (lut functions)
            for i in 0 to (((2**n)-1)*4)-1 loop
                l_in <= lut_vector(i);
                wait for 10 ns;
                clk_lut_in;
            end loop;

        end loop;

        report "Test Complete" severity note;
        wait;

    end process;

end behavior;
```

Listing 7: Top Level Test Bench Module

7 VHDL Waveform Plots and Results

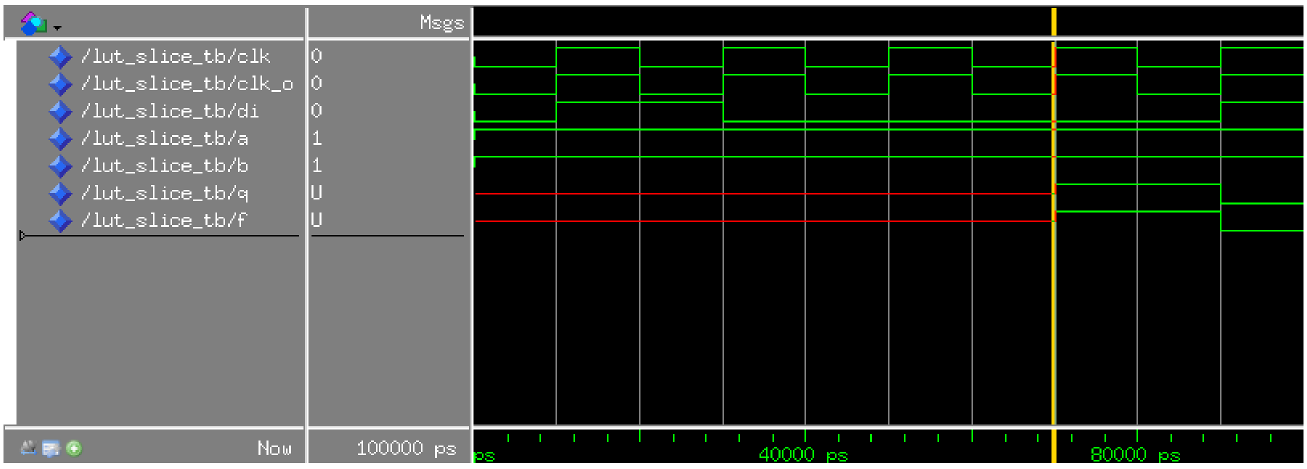

Figure 12: AND Gate of LUT Slice

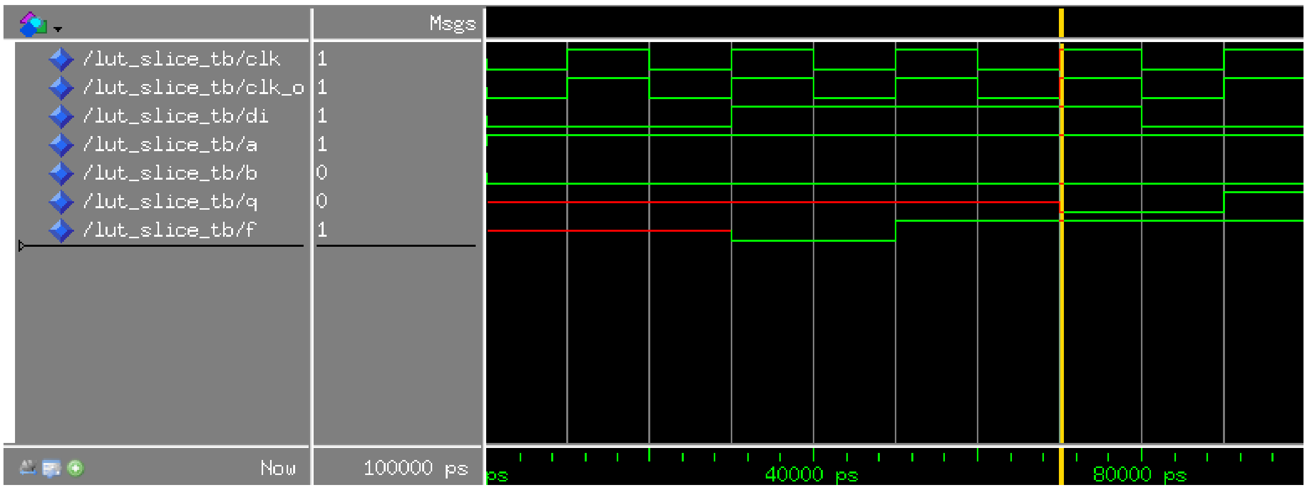

Figure 13: NAND Gate of LUT Slice

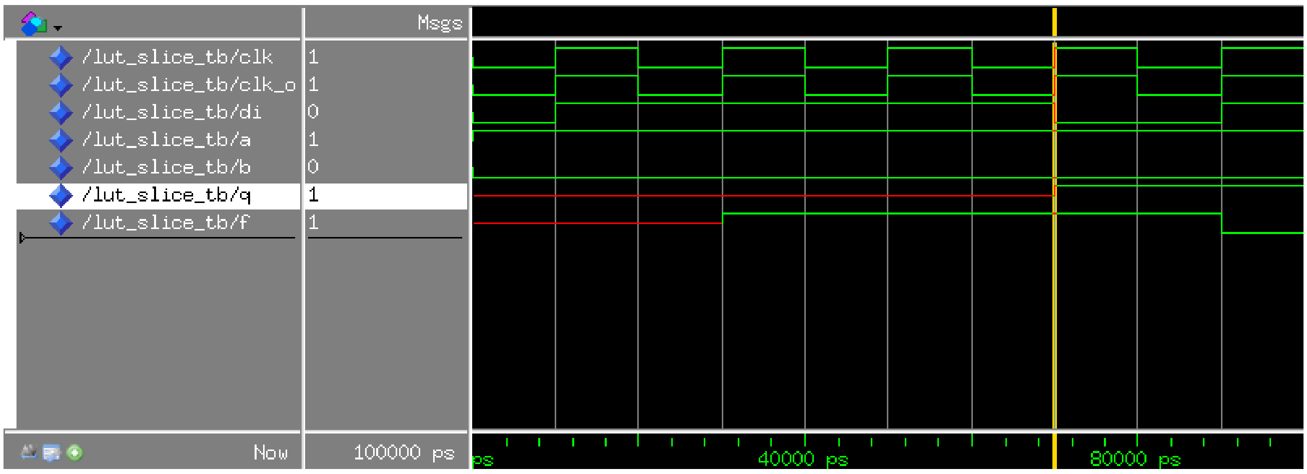

Figure 14: OR Gate of LUT Slice

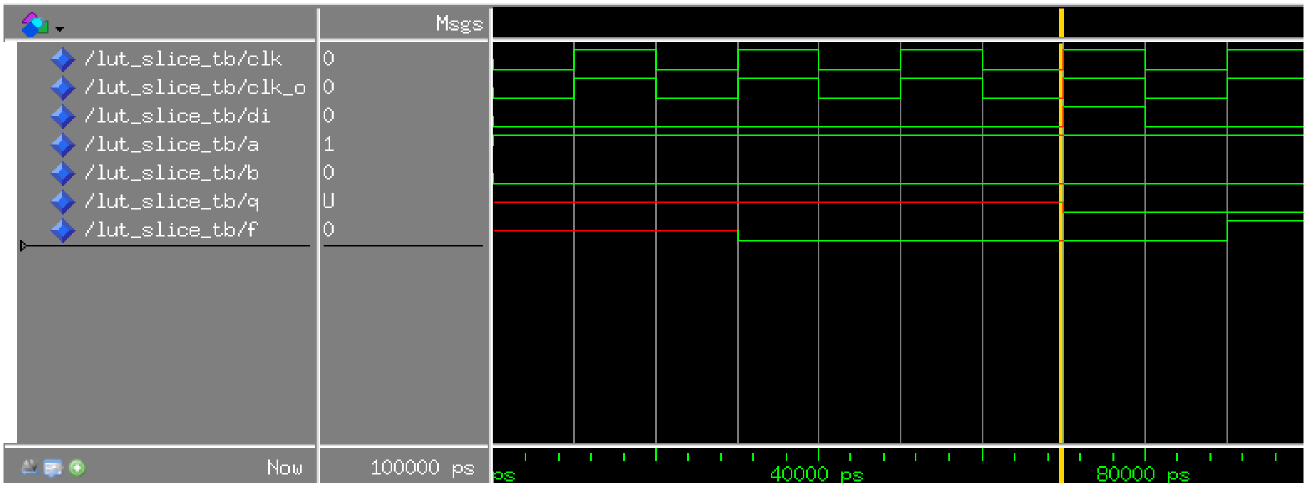

Figure 15: NOR Gate of LUT Slice

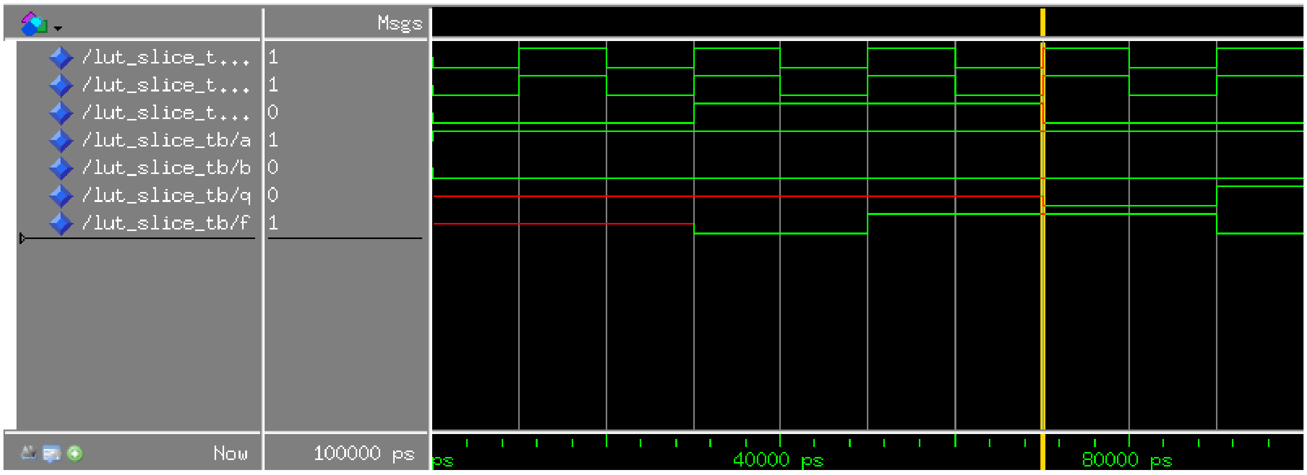

Figure 16: XOR Gate of LUT Slice

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.

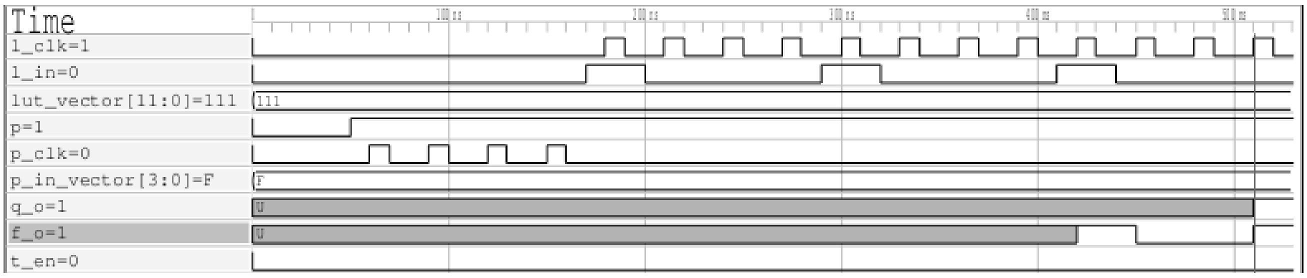

Figure 17: Waveform for Top Test Bench

Since we are testing with 8 input values, there are 2**28 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).

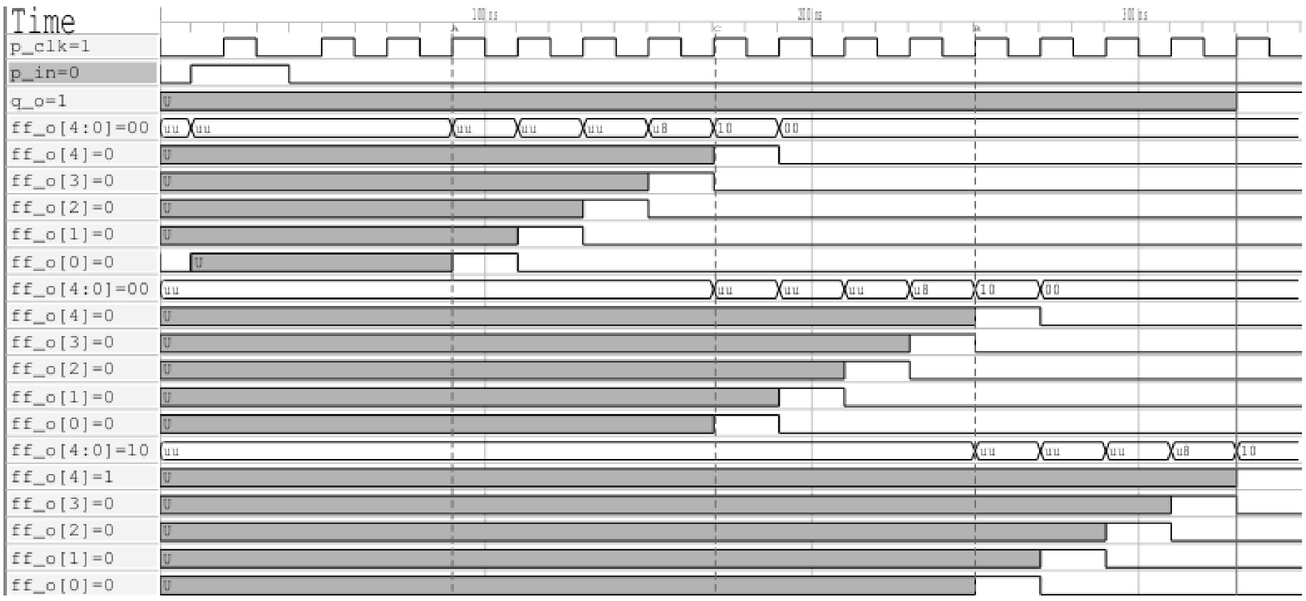

Figure 18: Waveform for Top Test Bench for Test Enabled

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

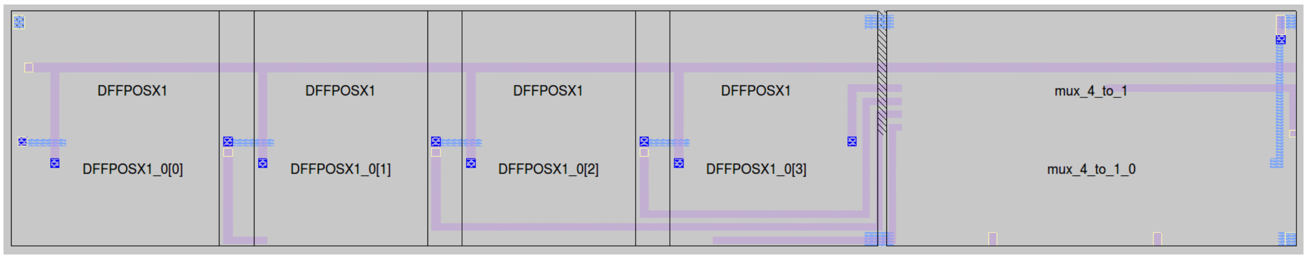

Figure 19: LUT Slice Magic Layout

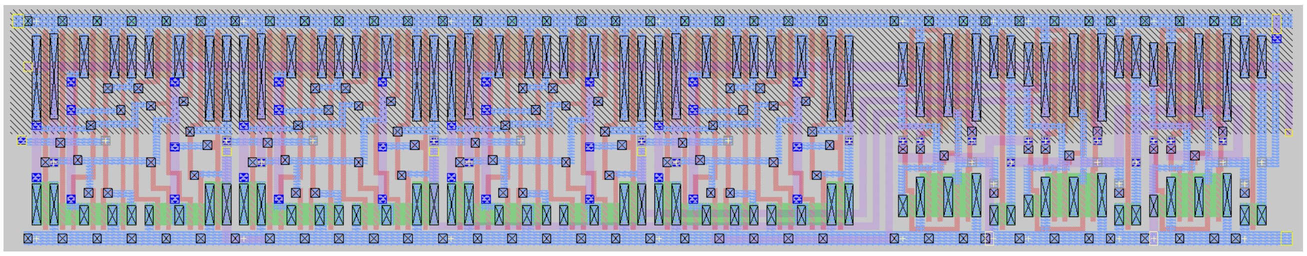

Figure 20: LUT Slice Internal Magic Layout

9.2 Shift Slice Layout

VDD
DFFPOSX1
DFFPOSX1_0

Figure 21: Shift Slice Magic Layout

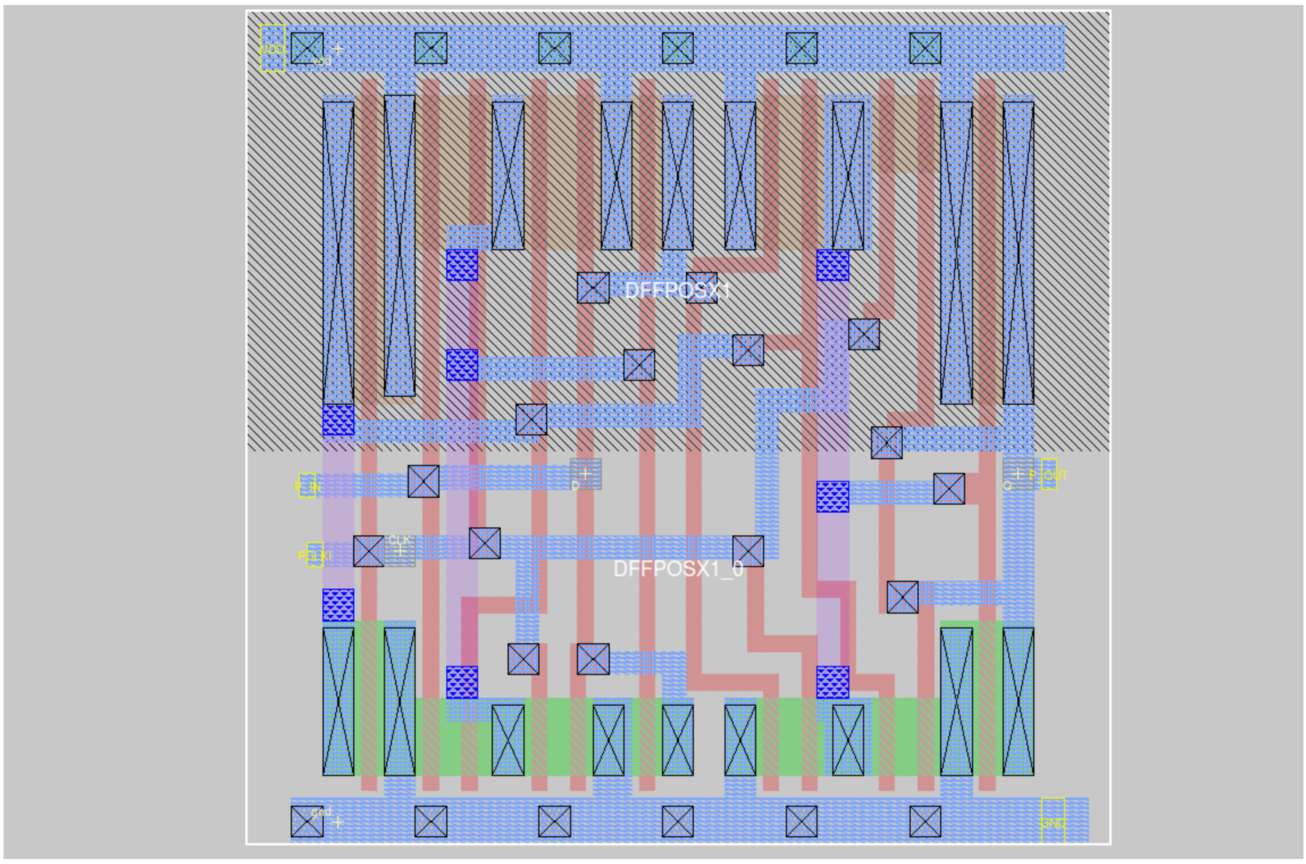

Figure 22: Shift Slice Internal Magic Layout

10 IRSIM Simulations

10.1 Bit Slice Simulations

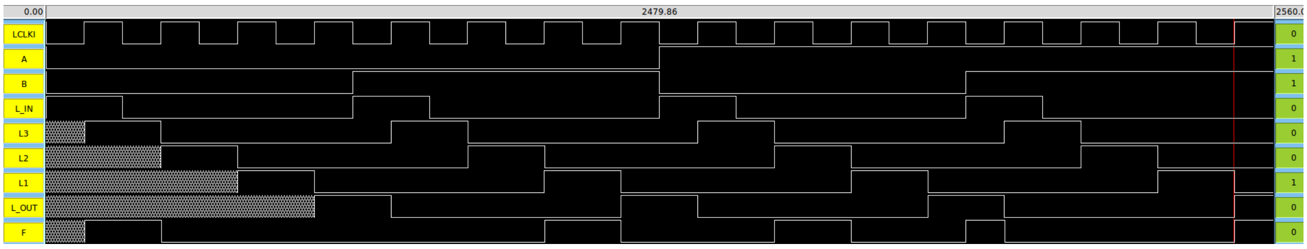

Figure 23: LUT Slice with AND gate 'programmed'

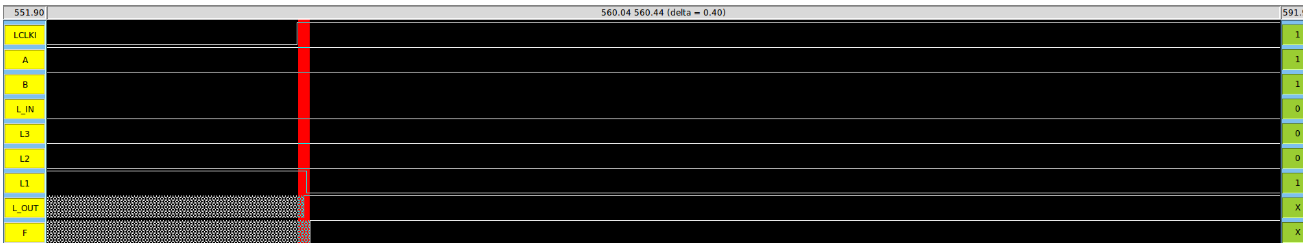

Figure 24: Rising Edge of F output of LUT Slice

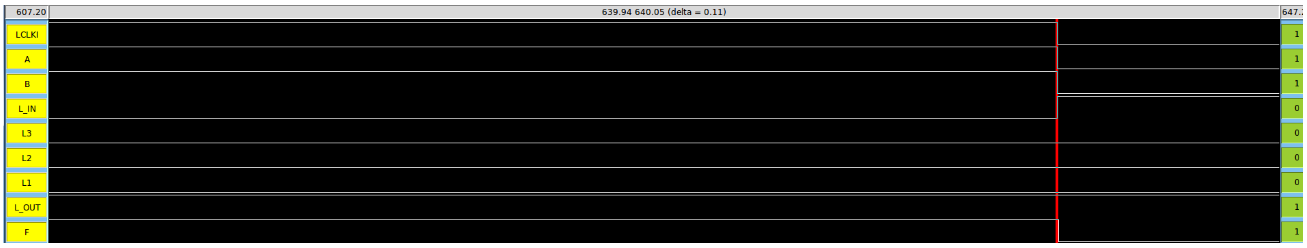

Figure 25: Falling Edge of F output of LUT Slice

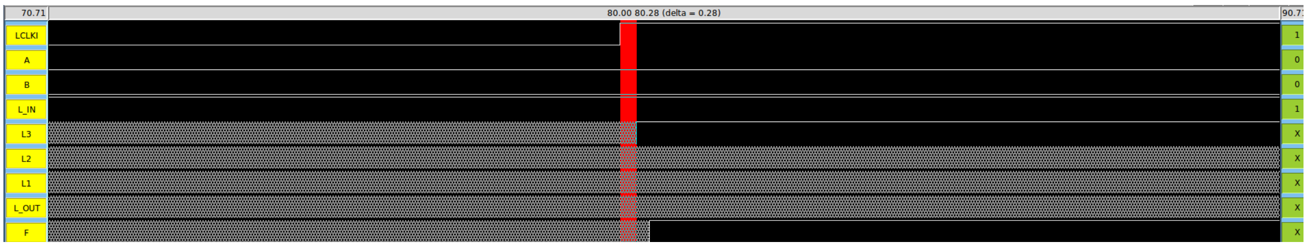

Figure 26: Rising Edge of L3 DFF of LUT Slice

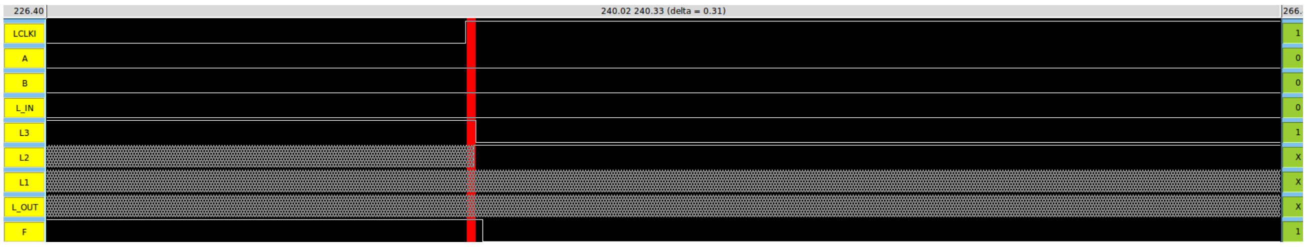

Figure 27: Falling Edge of L3 DFF of LUT Slice

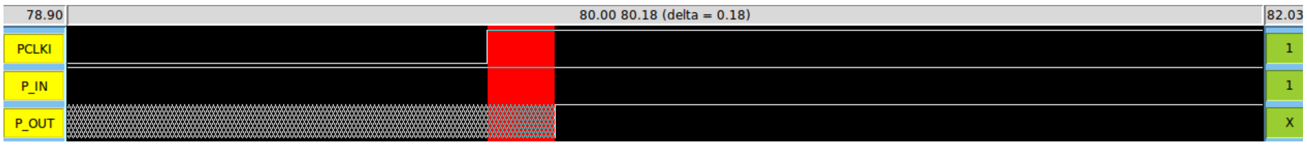

Figure 28: Rising Edge of DFF of Shift Slice

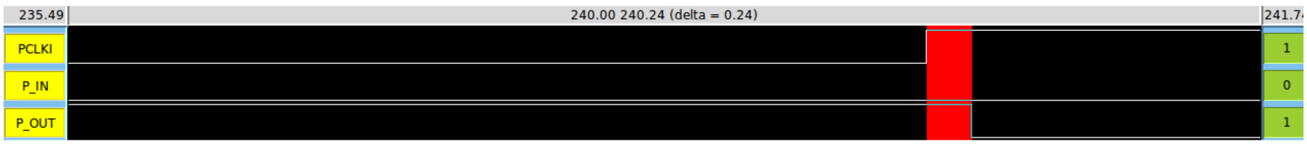

Figure 29: Falling Edge of DFF of Shift Slice

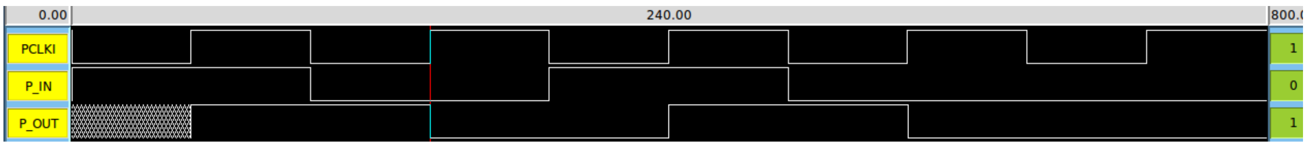

Figure 30: Waveform of DFF of Shift Slice

11 Gate Level Simulations

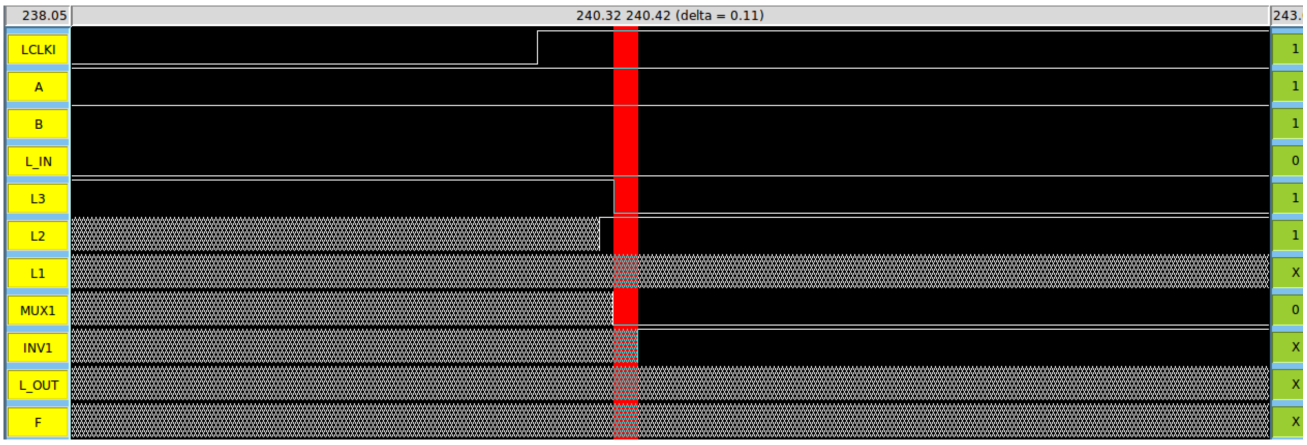

Figure 31: Rising Edge of Inverter

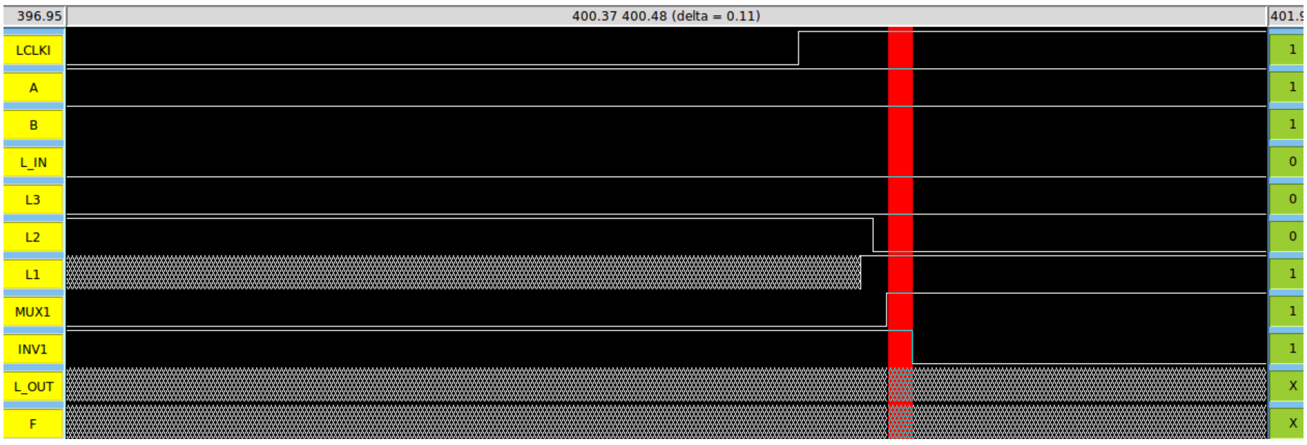

Figure 32: Falling Edge of Inverter

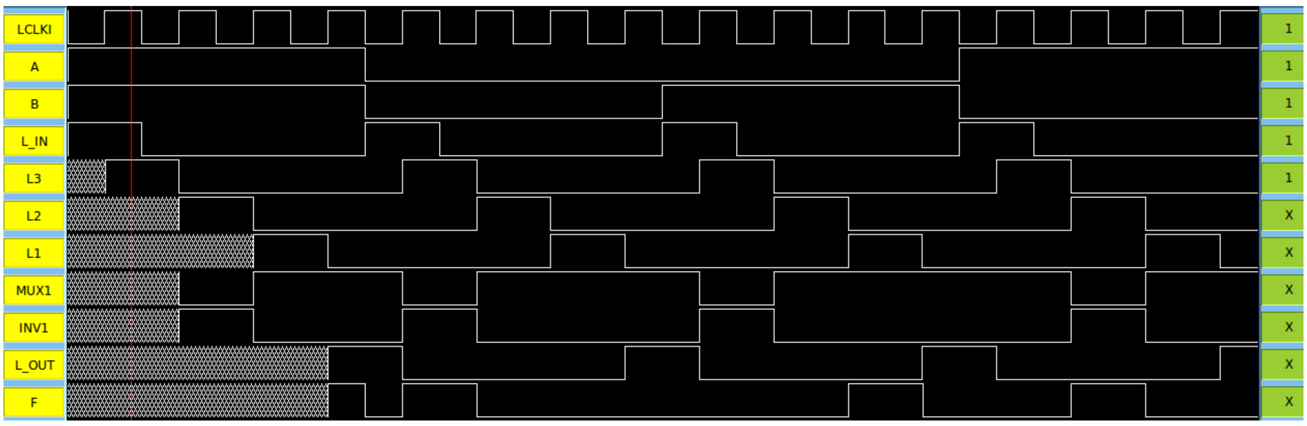

Figure 33: Waveform of LUT Slice (Includes nodes at each flip flop)

12 HSpice Simulations

12.1 Bit Slice Simulations

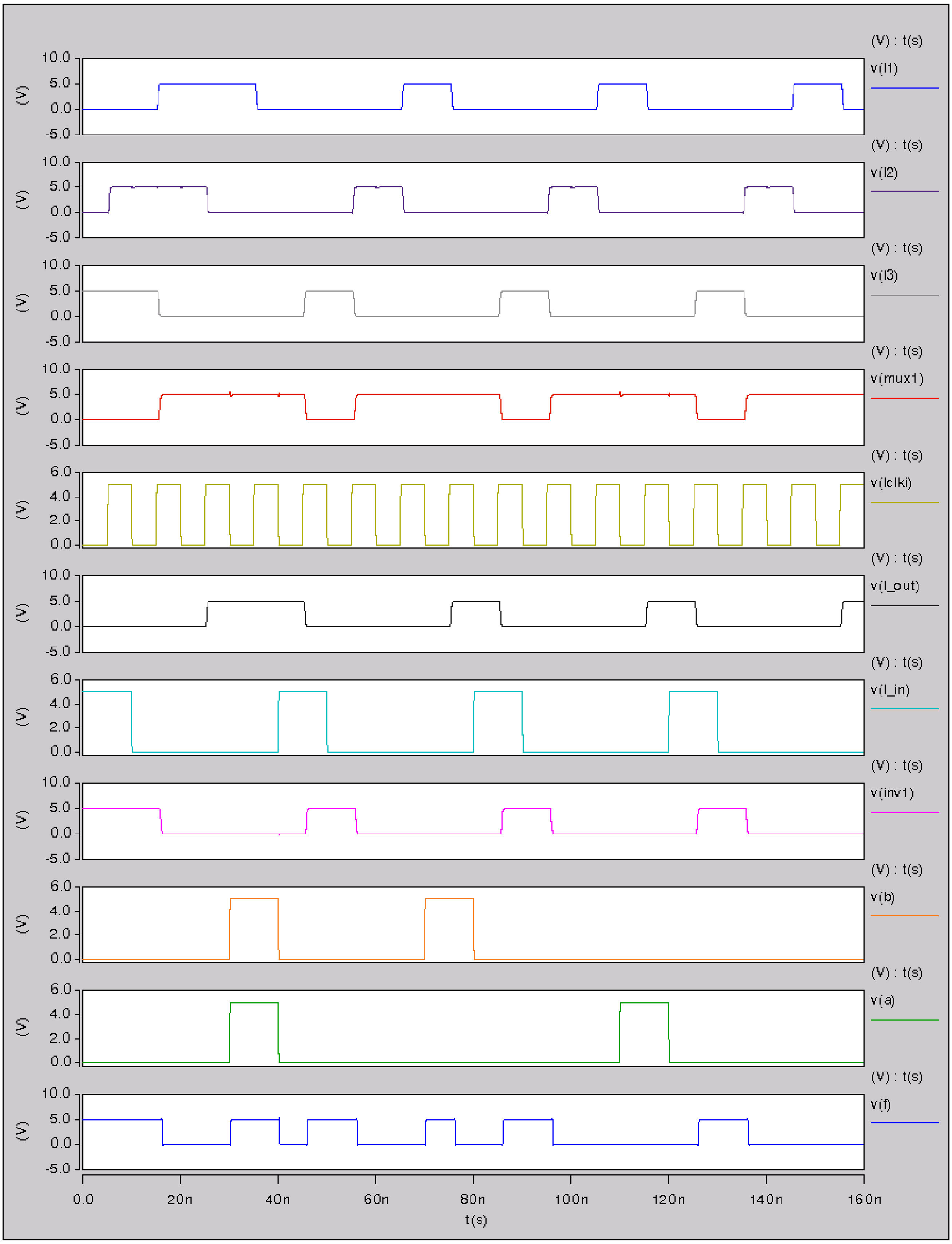

Figure 34: Hspice LUT Slice Wavefrom

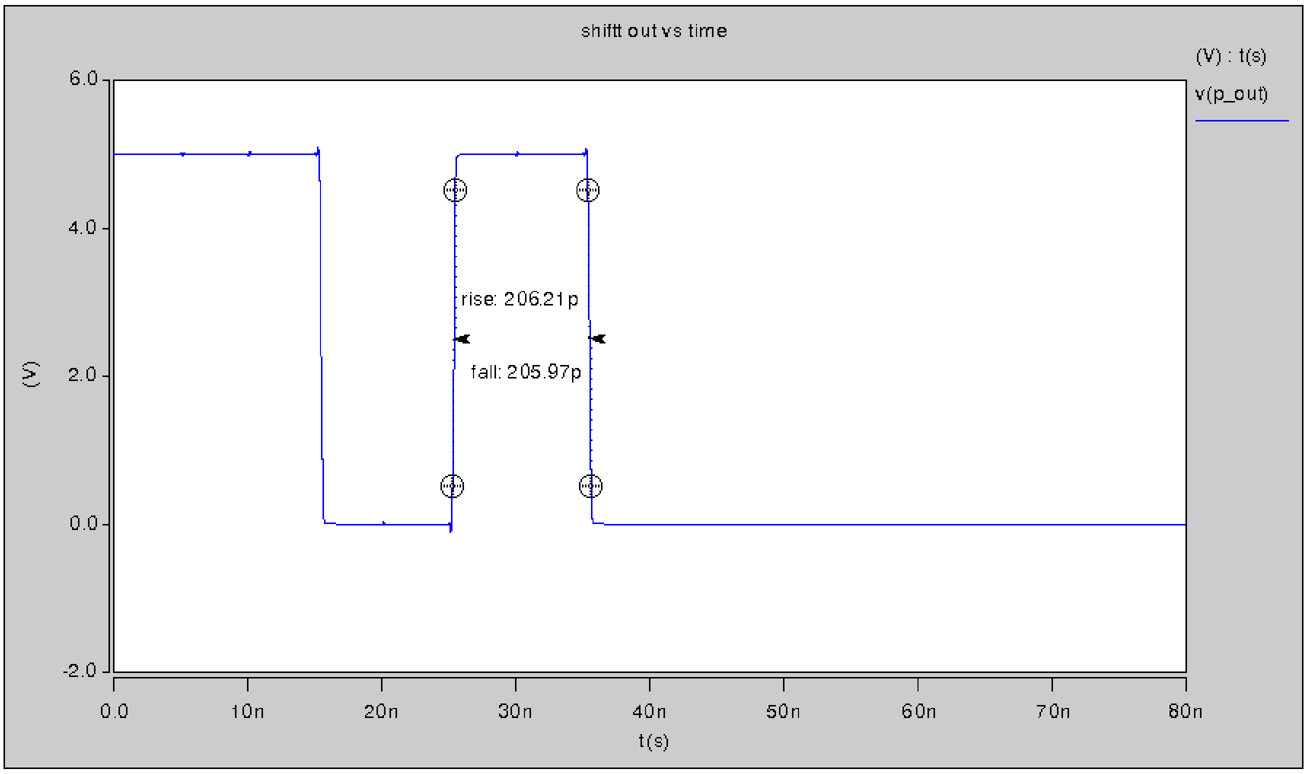

Figure 35: Hspice Delay of Shift Register Slice

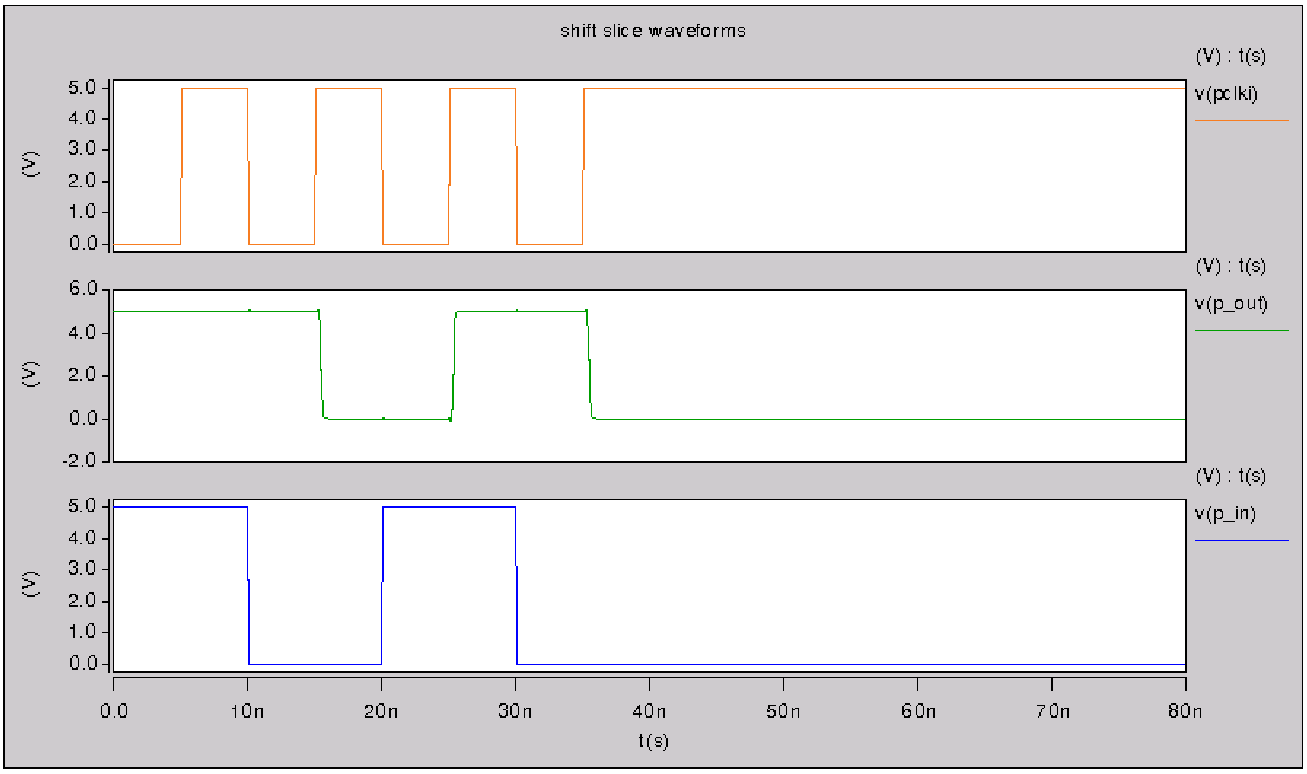

Figure 36: Hspice Shift Register Slice Waveform

12.2 Gate Level Simulations

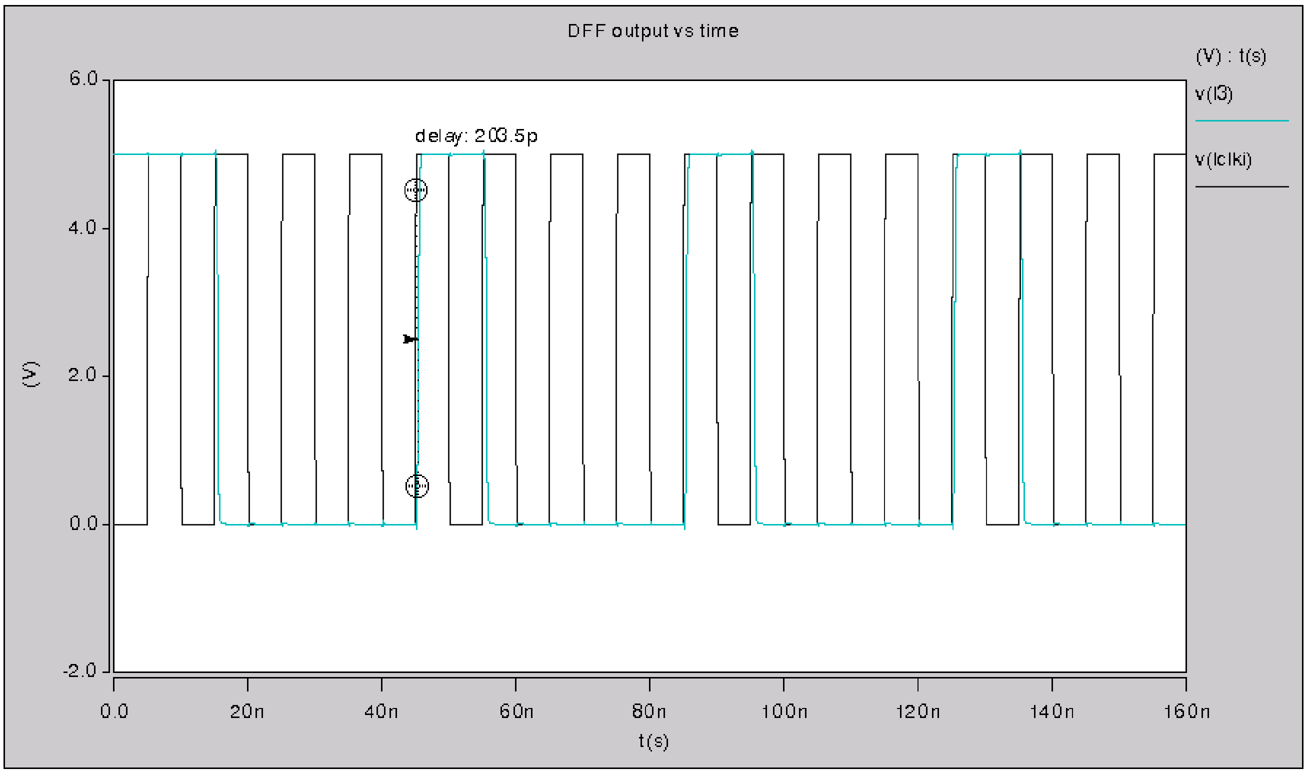

Figure 37: Hspice Delay of DFF

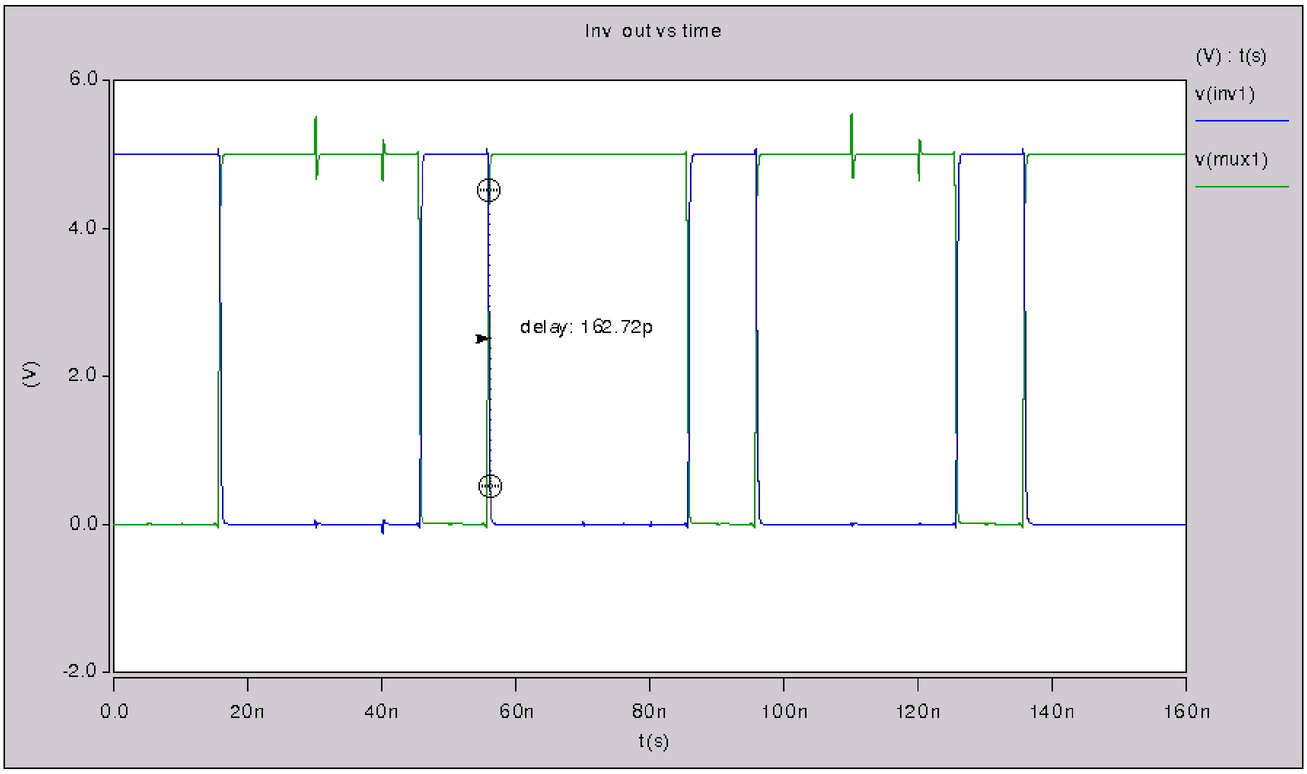

Figure 38: Hspice Delay of Inveter

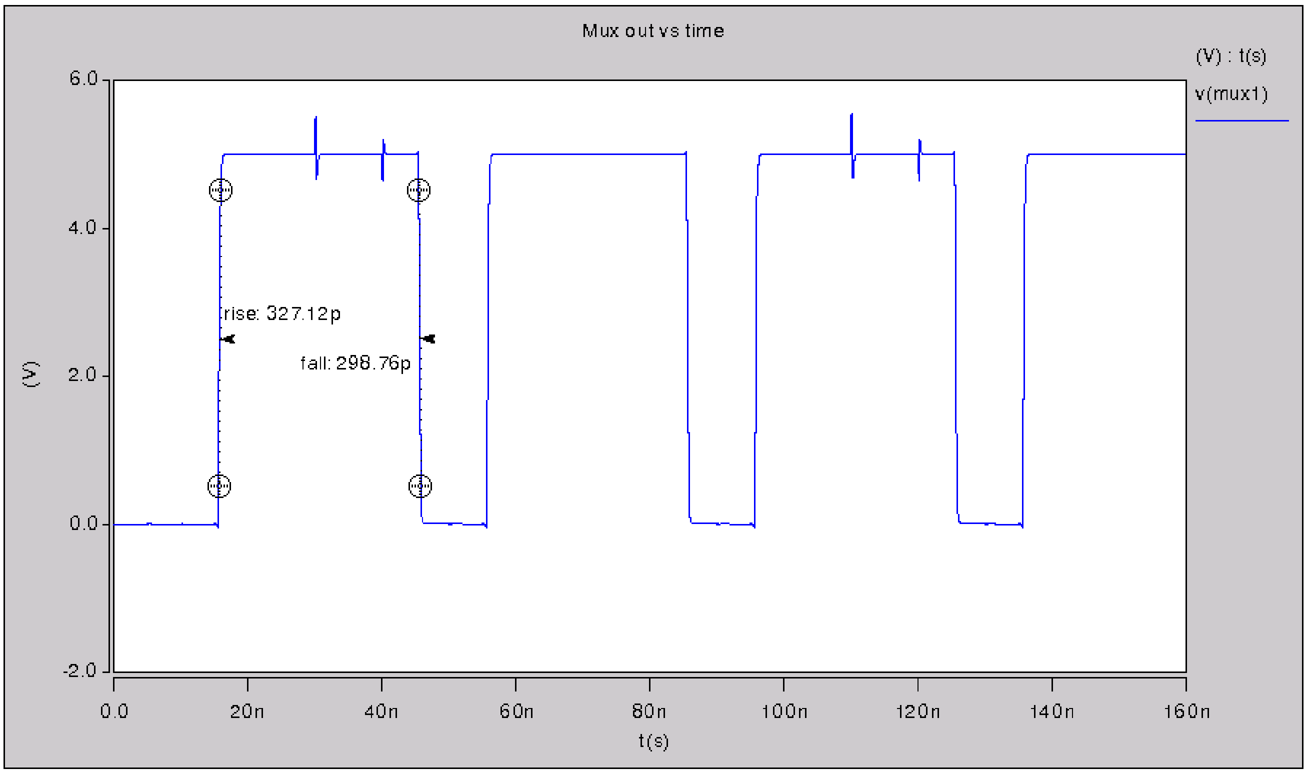

Figure 39: Hspice Delay of Mux

	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: Delay Times

13 VHDL Modules with Delay

```vhdl
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
    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

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

entity lut_del is
    generic(
        n       : integer := 2    -- number of levels in tree
    );
    port(
        s_clk   : in std_logic;   -- shift register clock
        l_clk   : in std_logic;   -- lut shift register clock
        s_in    : in std_logic;   -- shift register input (P)
        l_in    : in std_logic;   -- lut shift register input
        t_po    : out std_logic;  -- p_out for test mode
        f_o     : out std_logic;  -- final output of computation
        q_o     : out std_logic   -- final lut shift register output
    );
end lut_del;

architecture rtl of lut_del is

    -- tree diagram for n = 3
    -- row 0 : A
    -- row 1 : BC
    -- row 2 : DEFG
    -- row 3 : shifter

    -- lut connection diagram
    -- l_in -> D -> E -> F -> G -> B -> C -> A

    component dffposx1_del is
        port(
            clk  : in std_logic;
            d    : in std_logic;
            q    : out std_logic
        );
    end component;

    component lut_slice_del 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 component;

    -- carray_array(row, col)
    type carry_array is array (0 to n, 0 to 2**n) of std_logic;
    signal clk_c : carry_array;

    -- carries select outputs of one
    -- row to select inputs of next one
    signal r_c   : carry_array;

    -- carry output of each lut shift
    -- register
    signal l_c   : carry_array;

    -- input shift register carries
    signal s_c   : std_logic_vector(2**n downto 0) := (others => '0');

    --  input shift regstier clock carries
    signal p_clk : std_logic_vector(2**n downto 0) := (others => '0');

begin

    -- generate the input shift register
    shift_gen : for i in 0 to (2**n)-1 generate
        ff_i : dffposx1_del
        port map(
            clk => s_clk,
            d   => s_c(i),
                q    => s_c(i+1)
            );
    end generate shift_gen;

    -- generate the tree
    level_gen : for level in 0 to n-1 generate
        lut_gen : for i in 0 to (2**level)-1 generate
            lut_i : lut_slice_del
            port map(
                clk_i => l_clk,
                d     => l_c(level, i),
                q     => l_c(level, i+1),
                a     => r_c(level+1, i*2),
                b     => r_c(level+1, i*2+1),
                f     => r_c(level, i)
             );
        end generate;
    end generate;

    -- output the value of our function
    q_o <= l_c(0, 1);
    f_o <= r_c(0, 0);

    -- connect each row of LUTs together
    -- first slice in row i connects to
    -- last slice in row i+1
    lut_connect : for level in 0 to n-2 generate
        l_c(level, 0)   <= l_c(level+1, (2**level+1));
    end generate;

    -- connect input shift register to bottom row of slices
    shift_connect : for i in 0 to (2**n)-1 generate
        r_c(n, i) <= s_c(i+1);
    end generate;

    -- connect input to shift register
    s_c(0) <= s_in;

    -- connect input to first lut (bottom row)
    l_c(n-1, 0) <= l_in;

    -- use last slice of P shift register
    -- feed it into LUT input for test mode
    t_po <= s_c(2**n);

end rtl;
```

Listing 9: Look Up Table Module Delay

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

entity lut_slice_del 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_del;
```

```

architecture rtl of lut_slice_del is

    component dffposx1_del is
        port(
            clk  : in std_logic;
            d    : in std_logic;
            q    : out std_logic
        );
    end component;

    component invx1_del is
        port(
            a    : in std_logic;
            x    : out std_logic
        );
    end component;

    component mux2x1_del is
        port(
            b : in std_logic;
            a : in std_logic;
            s : 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_del
        port map(
            clk => clk_i,
            d   => ff_o(i),
            q   => ff_o(i+1)
        );
    end generate;

    -- lut shift out is output from prev ff
    q <= ff_o(4);
    ff_o(0) <= d;

    -- select first two outputs of LUT
    -- on sel line A
    mux1 : mux2x1_del port map(ff_o(1),  ff_o(2), a, mux_o(0));
    inv1 : invx1_del  port map(mux_o(0), mux_fo(0));

    -- select last two outputs of LUT
    -- on sel line A
    mux2 : mux2x1_del port map(ff_o(3),  ff_o(4), a, mux_o(1));
    inv2 : invx1_del  port map(mux_o(1), mux_fo(1));

    -- select the outputs from
    -- each mux on sel line B
    mux3 : mux2x1_del port map(mux_fo(0), mux_fo(1), b, f_muxo);

    -- invert mux due to func of mux: y = !(S?(A:B))
    inv3 : invx1_del port map(f_muxo, f);

end rtl;
```

Listing 10: Look Up Table Slice Module Delay

13.1 VHDL Test Bench Modules with Delay

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.txt_util.all;

entity top_test_del_tb is
end top_test_del_tb;

architecture behavior of top_test_del_tb is

    constant n : integer := 2;

    component top_del
        generic(
            n       : integer := 3     -- number of levels in tree
        );
        port(
            p_clk  : in std_logic;    -- p 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    -- lut shift register output
        );
    end component;

    signal p_clk  : std_logic := '0';
    signal l_clk  : std_logic := '0';
    signal p      : std_logic := '0';
    signal l_in   : std_logic := '0';
    signal f_o    : std_logic;
    signal q_o    : std_logic;
    signal t_en   : std_logic := '0';

begin

    dut : top_del
    generic map(
        n      => n
    )
    port map(
        p_clk => p_clk,
        l_clk => l_clk,
        p_in  => p,
        l_in  => l_in,
        t_en  => t_en,
        f_o   => f_o,
        q_o   => q_o
    );

    process

        procedure clk is begin
```

```
            p_clk <= '1';
            wait for 10 ns;
            p_clk <= '0';
            wait for 10 ns;
        end procedure clk;

    begin

        wait for 10 ns;
        -- enable test mode
        t_en <= '1';

        -- clock in p data
        p <= '1';
        wait for 10 ns;
        clk;

        -- clock in p data
        p <= '0';
        wait for 10 ns;
        clk;

        -- # of FF's per LUT: 4
        -- # of LUT FF's    : (((2**n)-1)*4)
        -- # of P FF's      : 2**n
        -- subtract 2 since we just loaded in two inputs
        for i in 0 to ((((2**n)-1)*4)+2**n)-2-1 loop
            clk;
        end loop;

        wait;

    end process;

end behavior;
```

Listing 11: Test Mode Enabled Test Bench

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.txt_util.all;

entity top_del_tb is
    generic(
        --stim_file : string := "test_tree.sim"
        stim_file : string := "two_tree.sim"
    );
end top_del_tb;

architecture behavior of top_del_tb is

    constant n : integer := 2;

    signal p_clk : std_logic := '0';
    signal l_clk : std_logic := '0';
    signal p     : std_logic := '0';
    signal l_in  : std_logic := '0';
    signal f_o   : std_logic;
    signal q_o   : std_logic;
    signal t_en  : std_logic := '0';

    signal p_in_vector : std_logic_vector((2**n)-1        downto 0);
    signal lut_vector  : std_logic_vector((((2**n)-1)*4)-1 downto 0);

    file stimulus : TEXT open read_mode is stim_file;

    component top_del
        generic(
           n      : integer := 3    -- number of levels in tree
       );
        port(
            p_clk  : in std_logic;   -- p 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   -- lut shift register output
        );
    end component;

begin

    dut : top_del
    generic map(
        n     => n
    )
    port map(
        p_clk  => p_clk,
        l_clk  => l_clk,
        p_in   => p,
        l_in   => l_in,
        t_en   => t_en,
        f_o    => f_o,
        q_o    => q_o
    );

    process

        procedure clk_p_in is begin
            p_clk <= '1';
            wait for 10 ns;
            p_clk <= '0';
            wait for 10 ns;
        end procedure clk_p_in;

        procedure clk_lut_in is begin
            l_clk <= '1';
            wait for 10 ns;
            l_clk <= '0';
            wait for 10 ns;
        end procedure clk_lut_in;

        variable l: line;
        variable p_in_str : string(1 to 2**n);
        variable l_shf_str: string(1 to ((2**n)-1)*4);

    begin

        while not endfile(stimulus) loop

            -- load stimulus for this test
            readline(stimulus, l); read(l, p_in_str);
            p_in_vector <= to_std_logic_vector(p_in_str);

            readline(stimulus, l); read(l, l_shf_str);
```

```vhdl
            lut_vector <= to_std_logic_vector(l_shf_str);

            wait for 50 ns;

            -- clock in the P input
            for i in 0 to (2**n)-1 loop
                p <= p_in_vector(i);
                wait for 10 ns;
                clk_p_in;
            end loop;

            -- clock in the "program" (lut functions)
            for i in 0 to (((2**n)-1)*4)-1 loop
                l_in <= lut_vector(i);
                wait for 10 ns;
                clk_lut_in;
            end loop;

        end loop;

        report "Test Complete" severity note;
        wait;

    end process;

end behavior;
```

Listing 12: Top Module Test Bench

13.2 VHDL Gate Modules with Delay

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

entity dffposx1_del is
    generic(
        delay : time := 203.5 ps
    );
    port(
        clk  : in std_logic;
        d    : in std_logic;
        q    : out std_logic
    );
end dffposx1_del;

architecture rtl of dffposx1_del is begin
    process(clk) begin
        if rising_edge(clk) then
            q <= d after delay;
        end if;
    end process;
end rtl;
```

Listing 13: DFF Module Delay

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

entity invx1_del is
    generic(
        delay : time := 162.7 ps
    );
    port(
        a : in std_logic;
        x : out std_logic
    );
end invx1_del;

architecture rtl of invx1_del is begin
    x <= not a after delay;
end rtl;
```

Listing 14: Inverter Module Delay

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

entity mux2x1_del is
    generic(
        delay : time := 315 ps
    );
    port(
        b : in std_logic;
        a : in std_logic;
        s : in std_logic;
        x : out std_logic
    );
end mux2x1_del;

architecture rtl of mux2x1_del is begin

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

end rtl;
```

Listing 15: Inverter Module Delay

14 VHDL Modules Delay Waveforms

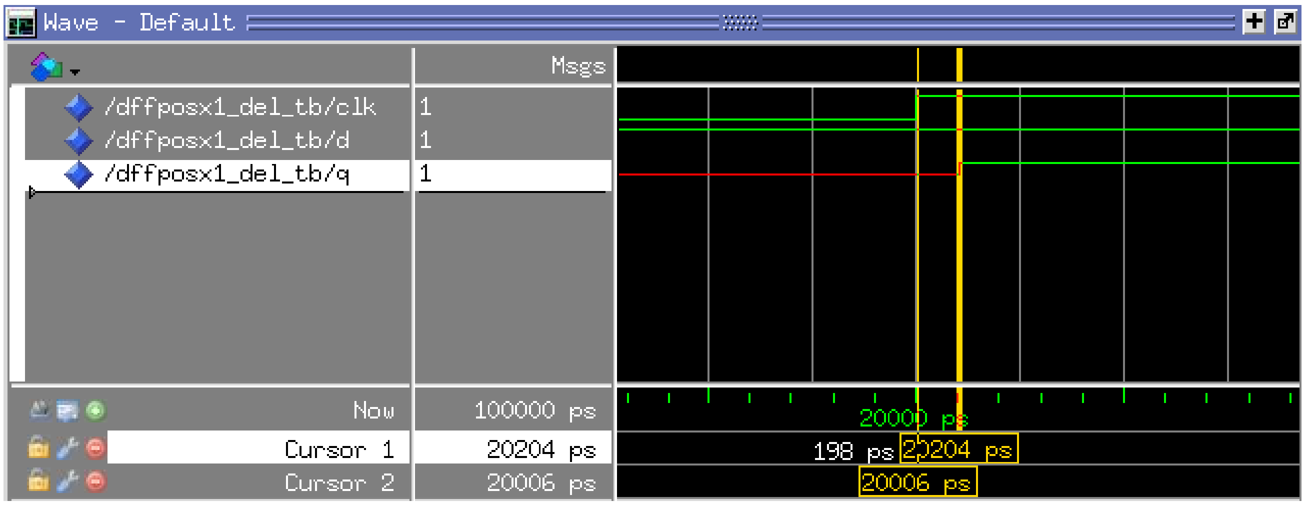

Figure 40: Delay Waveform of DFF

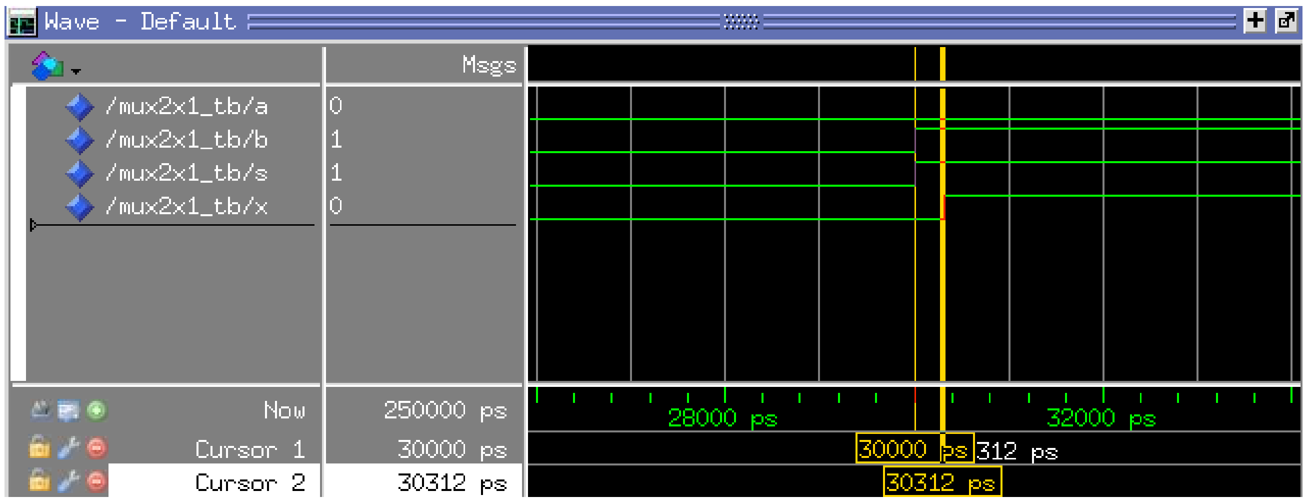

Figure 41: Delay Waveform of MUX

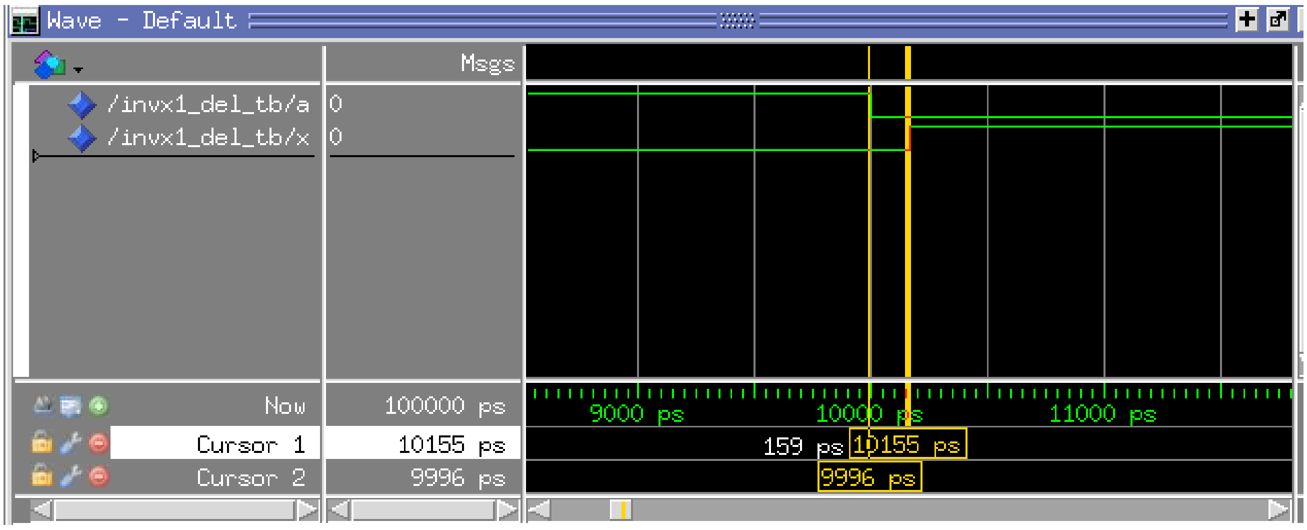

Figure 42: Delay Waveform of Inverter

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.

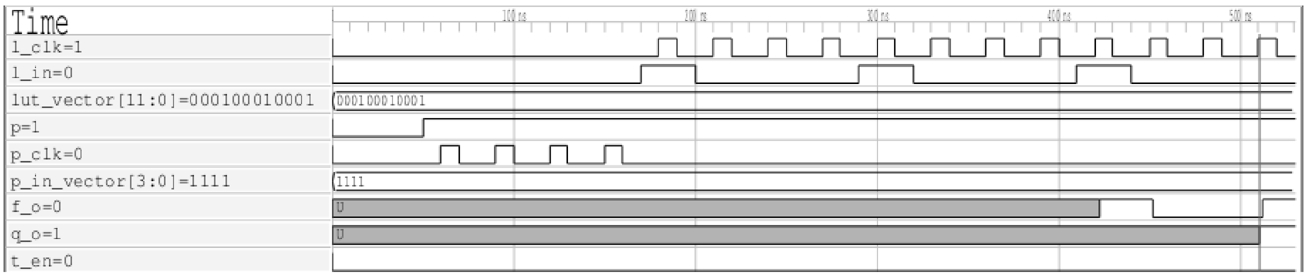

Figure 43: Top Level Delay Waveform

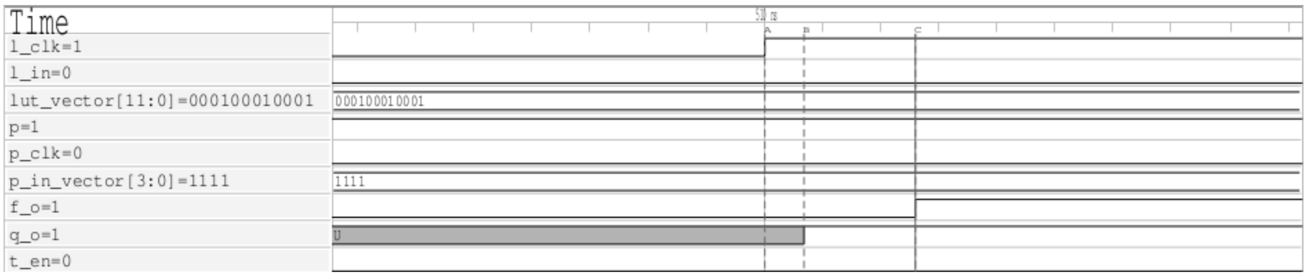

Figure 44: Top Level Delay Waveform Zoomed In

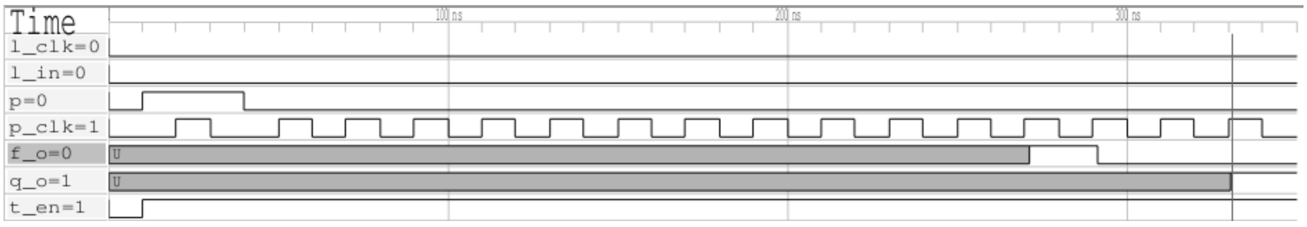

Figure 45: Top Level Test Mode Enabled Delay Waveform

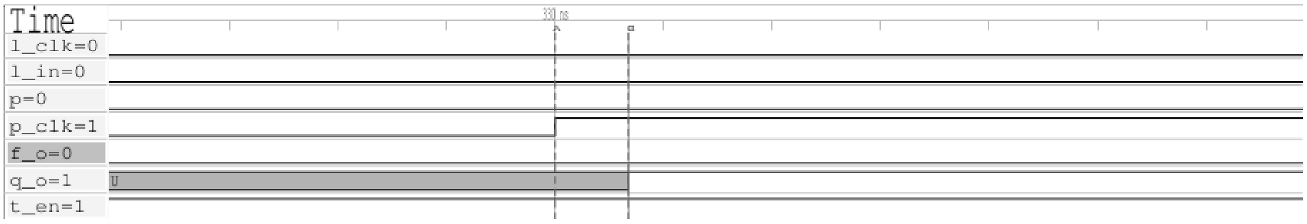

Figure 46: Top Level Test Mode Enabled Delay Waveform Zoomed In

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.

15 Floor Plan

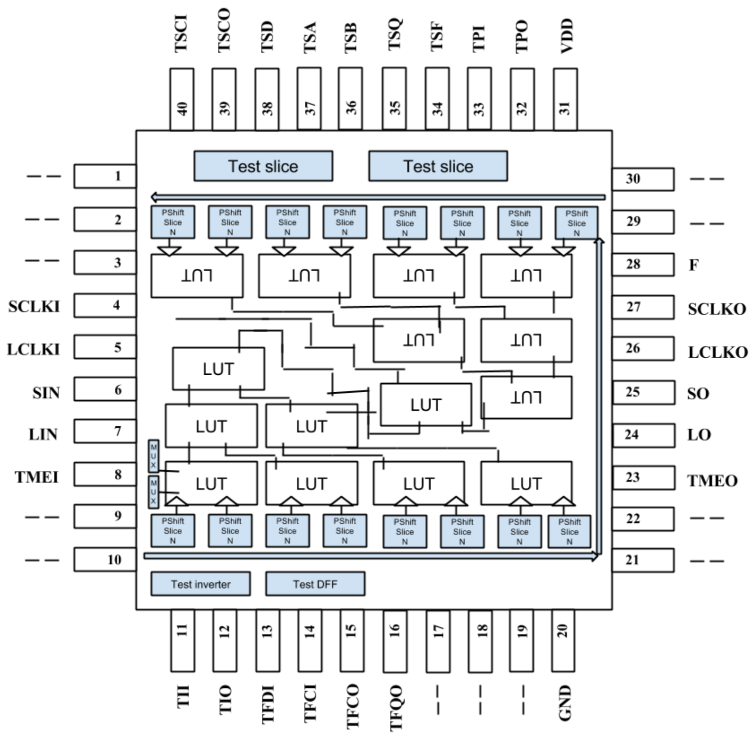

Figure 47: Floor Plan Design

This is the initial floor plan of our design. This is what we have come up with. It made it much easier for us to first design this floor plan in Magic and see how compact we can get it. The figure below shows the initial Magic Layout of the Chip. Below we can see that this is as compact as we can get it while at the same time we are able to fit 32 inputs for the shift register and 31 Look Up Tables.

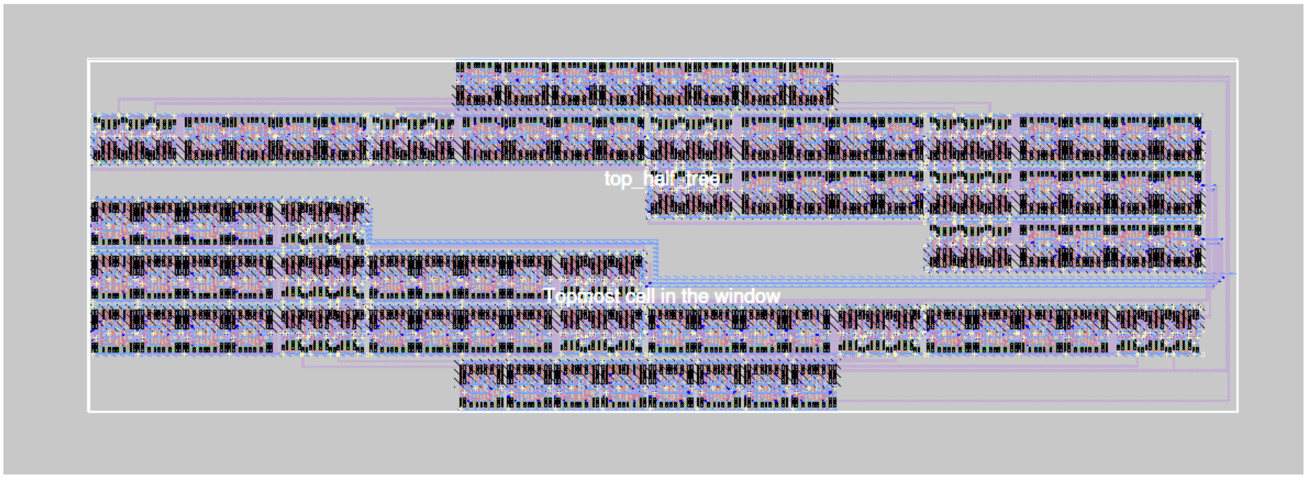

Figure 48: Initial Magic Layout Design

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. Although we are not able to utilize all of the area provided, we came up with the most efficient way by stacking the LUTs together. At first, we thought we could fold the LUTs against each corner of the frame, but doing so, let to much wasted space in the middle. Another method we had in mind was to have all of the P shift registers to be stacked all the way around the edges, and have it go around in a spiral, but this was inefficient and made it difficult to connect. Therefore as you can see, the method we have come up with is shown in the figure below. This method makes it a bit more efficient and utilizes more space than what we have previously come up with.

17 Work Division

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

Table 5: Task Assignment