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

October 30, 2013

Contents

List of Figures

List of Tables

Listings

1 Pinout Diagram

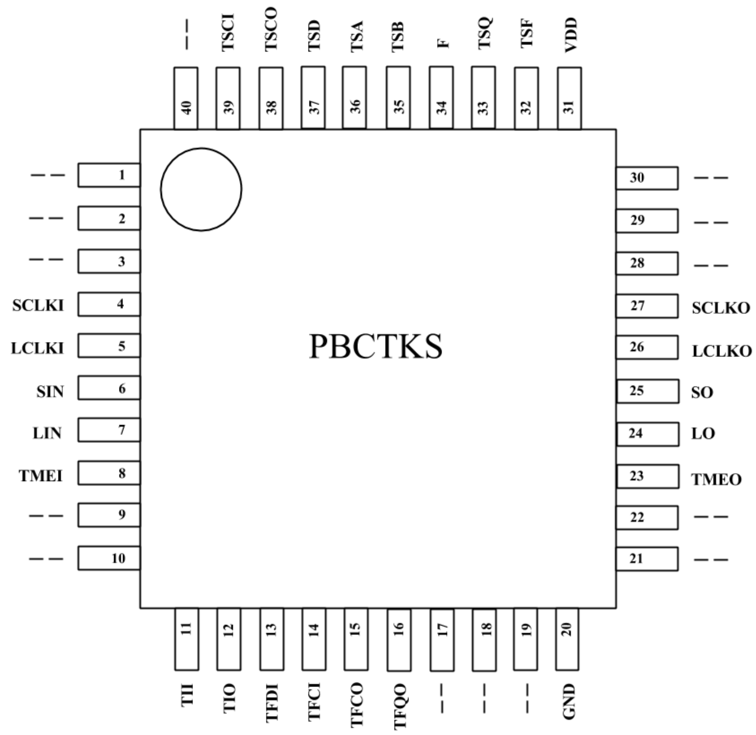

Figure 1: Pinout Diagram of the PBTCKS Chip

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
–	28	–	–
–	29	–	–
–	30	–	–
VDD	31	I	Test LUT Slice B Input
TSF	32	O	Test LUT Slice Mux Output
TSQ	33	O	Test LUT Slice Shift Register Output
FO	34	O	Output of Computation of LUT
TSB	35	I	Test LUT Slice B Address Input
TSA	36	I	Test LUT Slice A Address Input
TSD	37	I	Test LUT Slice Shift Register Input
TSCO	38	O	Test LUT Slice Clock Output
TSCI	39	I	Test LUT SLice Clock Input
–	40	–	–

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

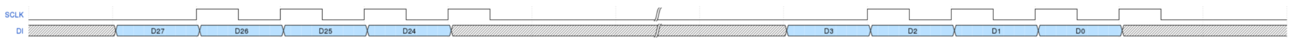

Figure 2: Cycle Timing Diagram for LUT Shift Register

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.

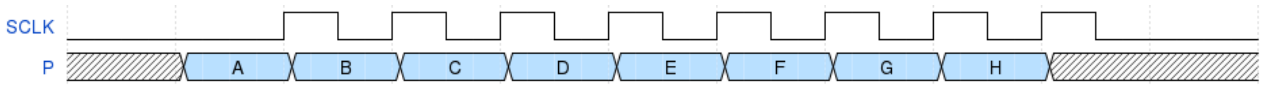

Figure 3: Cycle Timing Diagram for P Input Shift Register

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

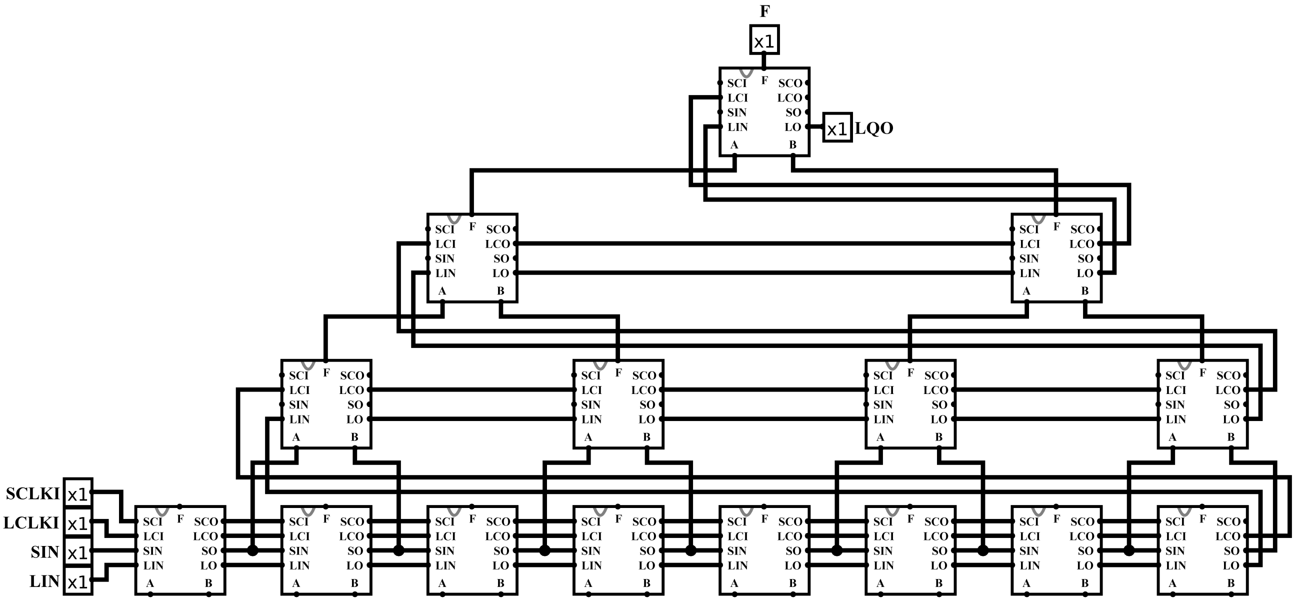

Figure 4: Hierarchical Design in Logisim

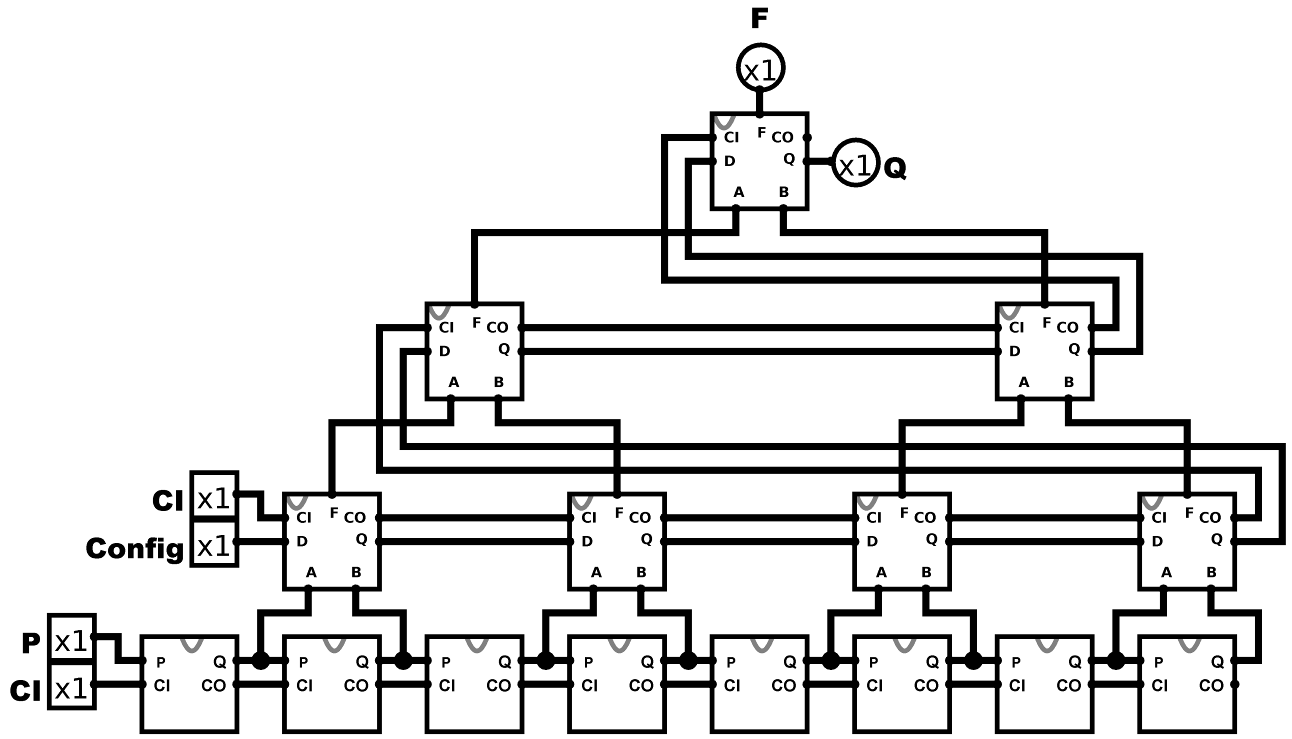

Figure 5: Top Level Diagram (8 Inputs, 28-bit slice)

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

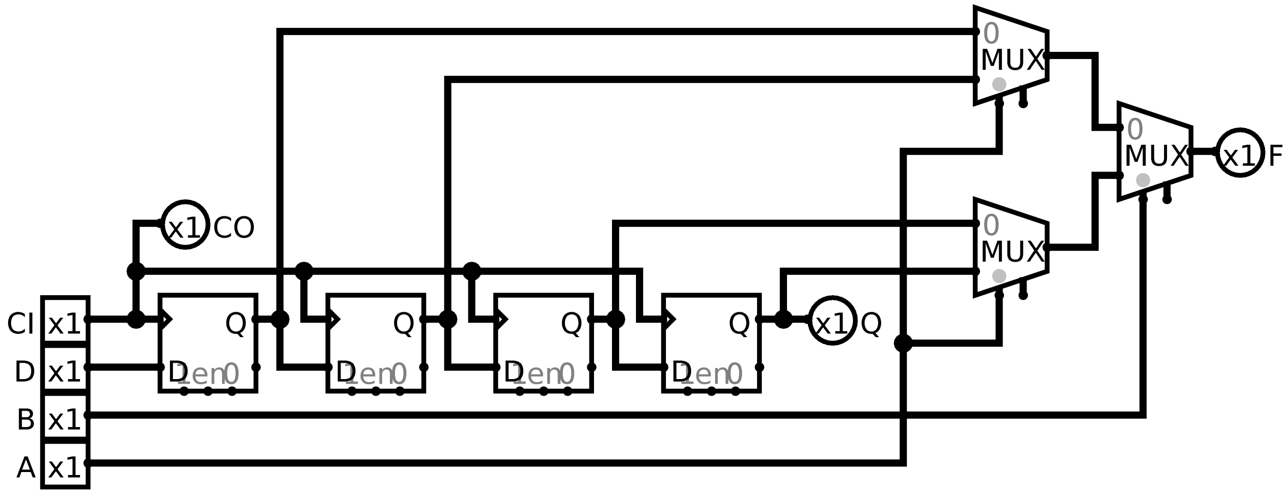

Figure 6: LUT Slice in Logisim

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

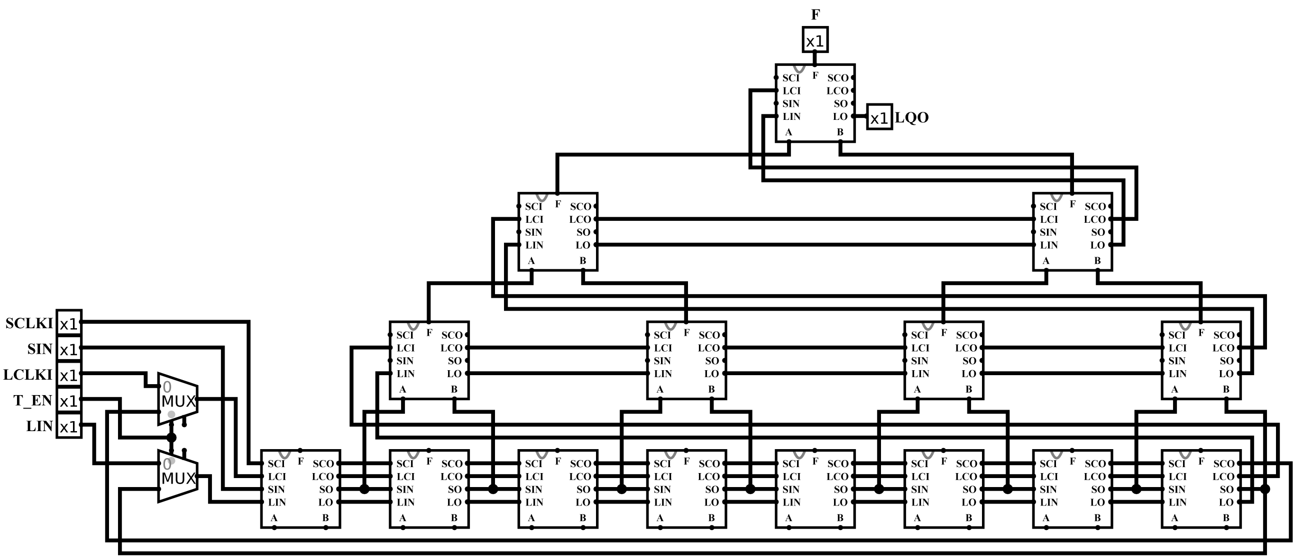

Figure 7: Hierarchical Test Design in Logisim

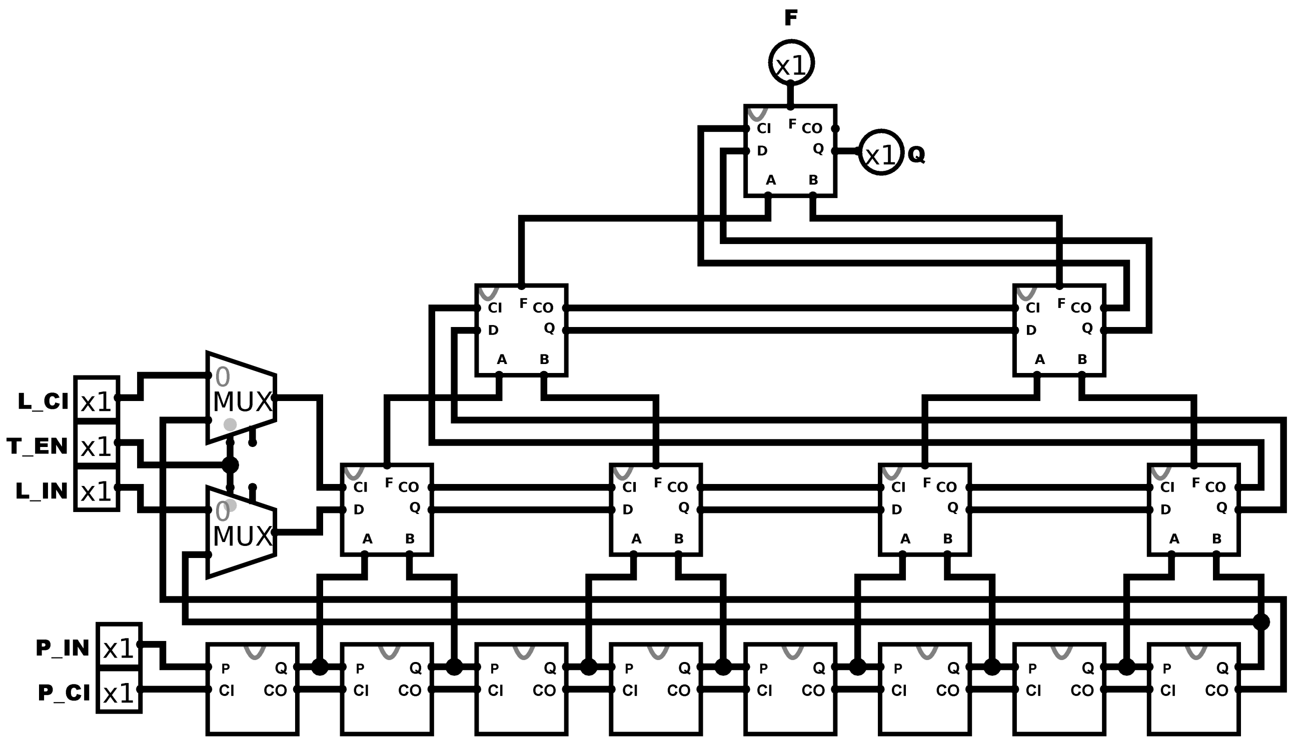

Figure 8: Top Level Test Mode Diagram (8 Inputs, 28-bit slice)

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

```
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

    component lut is
        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
            t_co   : out std_logic;   -- p_clk for test mode
            f_o    : out std_logic;   -- final output of computation
            q_o    : out std_logic    -- final lut shift register output
        );
    end component;

    signal shift_clk  : 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_clk);
    t_mux_2 : entity work.mux2x1 port map(l_in,  p_out, t_en, 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

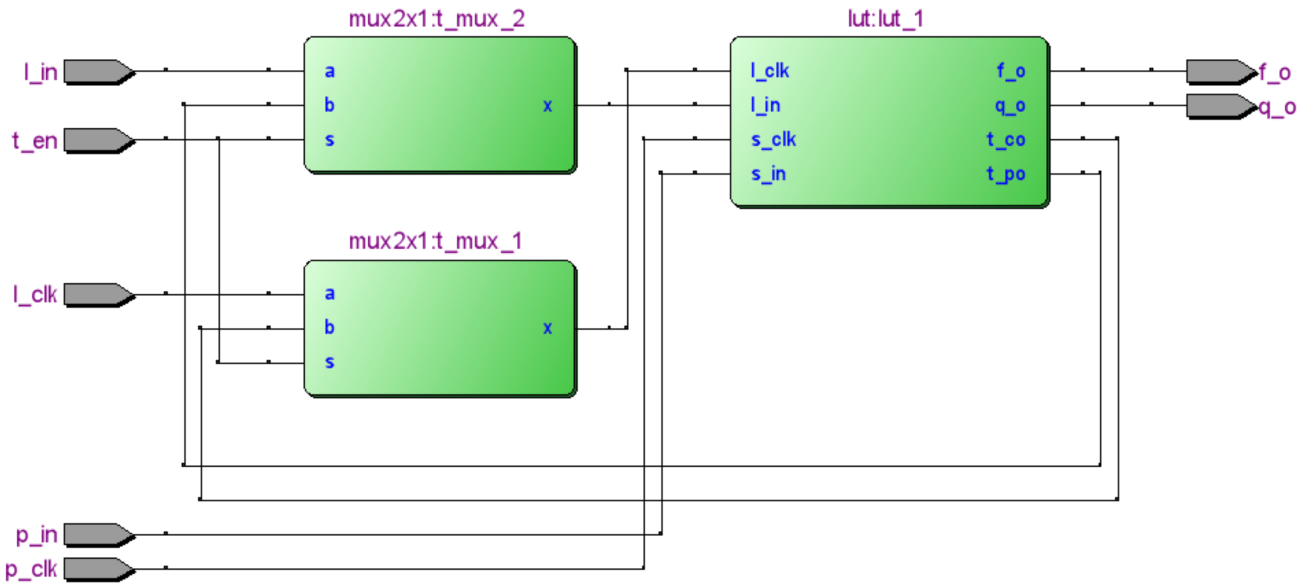

Figure 9: RTL Design of Top Level

6.2 Slice Modules

6.2.1 LUT

```
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(
            a : in std_logic;
            b : 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');

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));

    -- select last two outputs of LUT
```

```
    -- on sel line A
    mux2 : mux2x1 port map(ff_o(3), ff_o(4), a, mux_o(1));

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

end rtl;
```

Listing 2: Lookup Table Slice Module

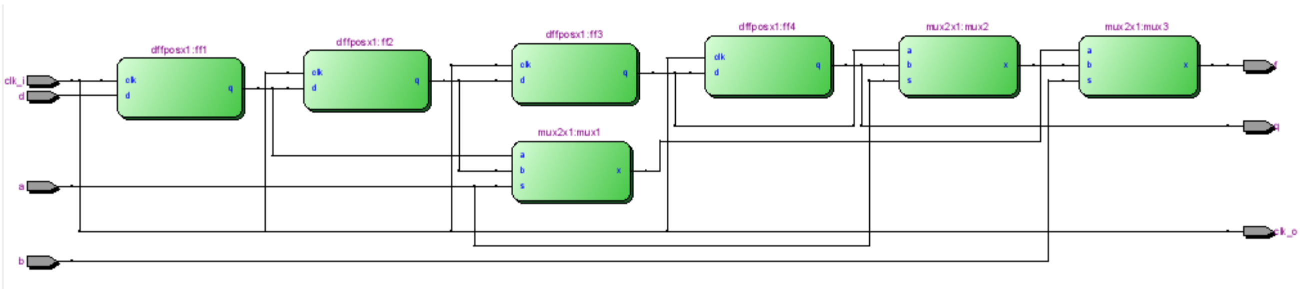

Figure 10: RTL Design of LUT Slice

6.2.2 Shift Register

```
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

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

    clk_o <= clk_i;

end rtl;
```

Listing 3: Lookup Table Slice Module

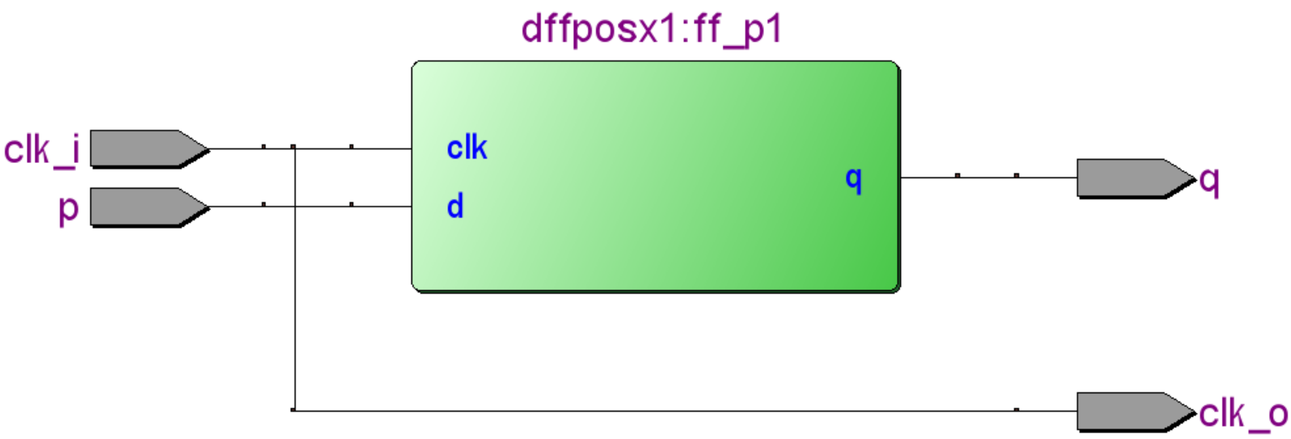

Figure 11: RTL Design of Shift Register Slice

6.3 Gates

6.3.1 D-Flip Flop

```
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

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

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

```
architecture rtl of mux2x1 is begin

    x <= a after delay when (s = '0') else
         b after delay when (s = '1');

end rtl;
```

Listing 5: 2:1 Multiplexer Module

6.4 VHDL Test Benches

```
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 clk_o    : std_logic;
    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;
            --clk_o   : out 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,
        --clk_o   => clk_o,
        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

```
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

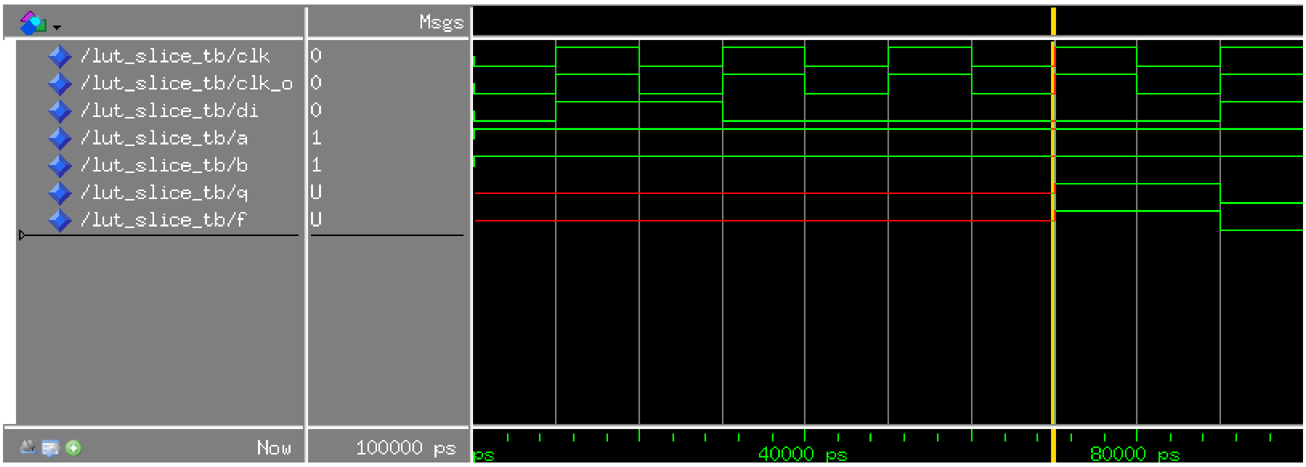

Figure 12: AND Gate of LUT Slice

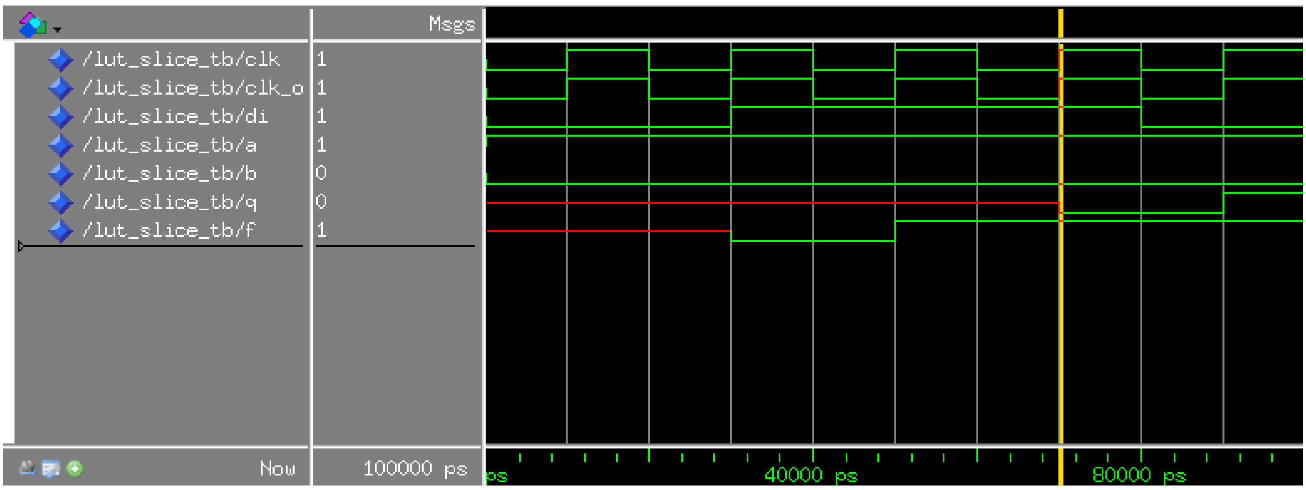

Figure 13: NAND Gate of LUT Slice

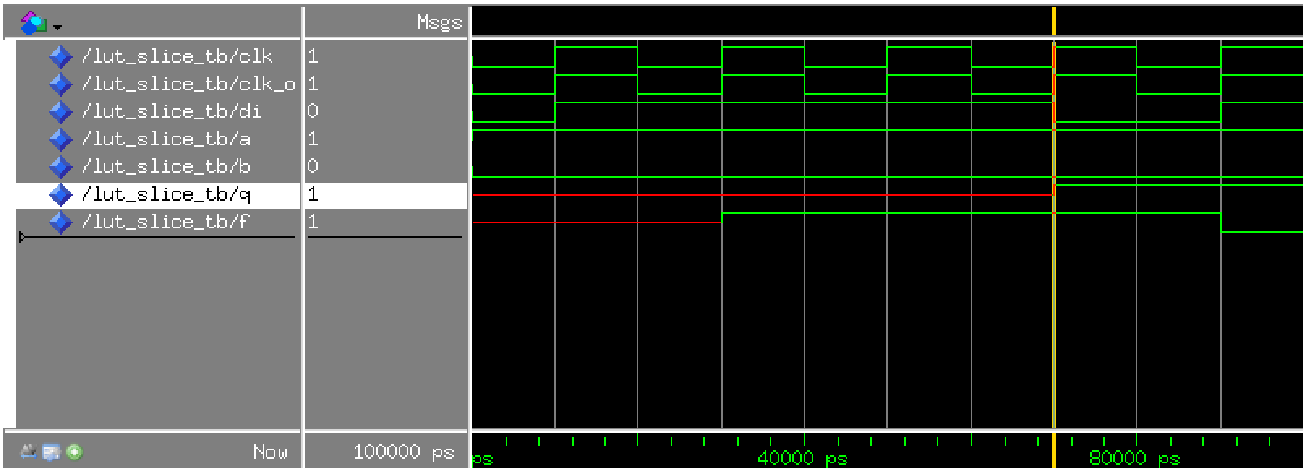

Figure 14: OR Gate of LUT Slice

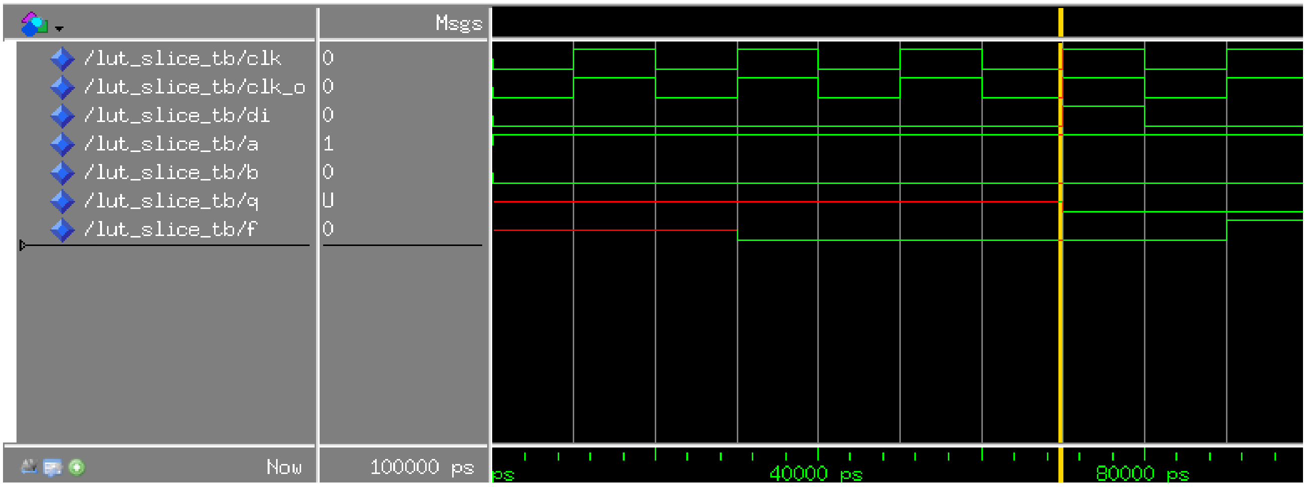

Figure 15: NOR Gate of LUT Slice

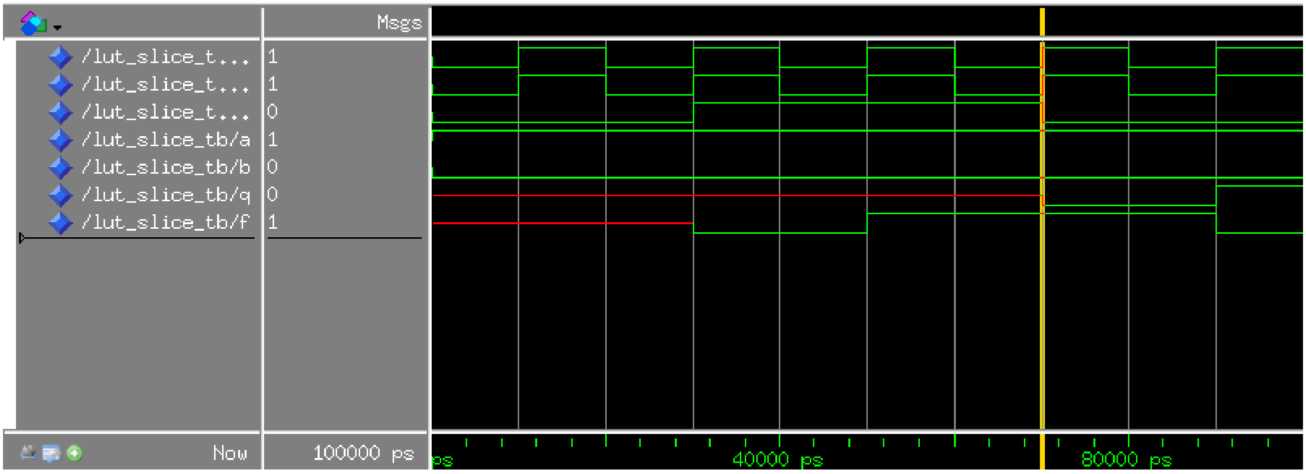

Figure 16: XOR Gate of LUT Slice

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.

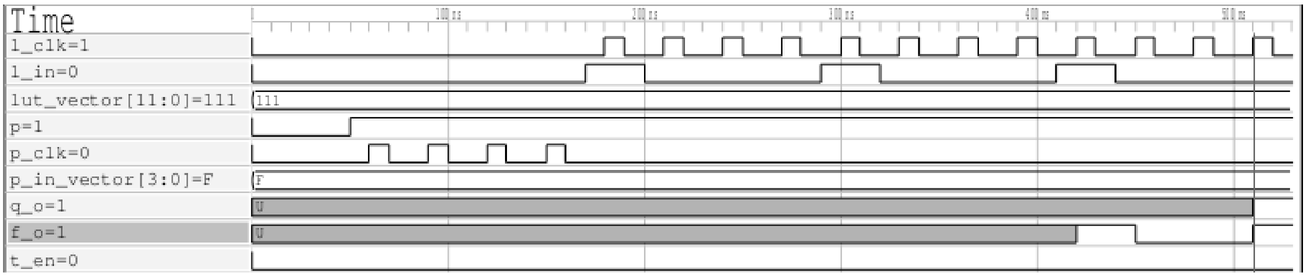

Figure 17: Waveform for Top Test Bench

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).

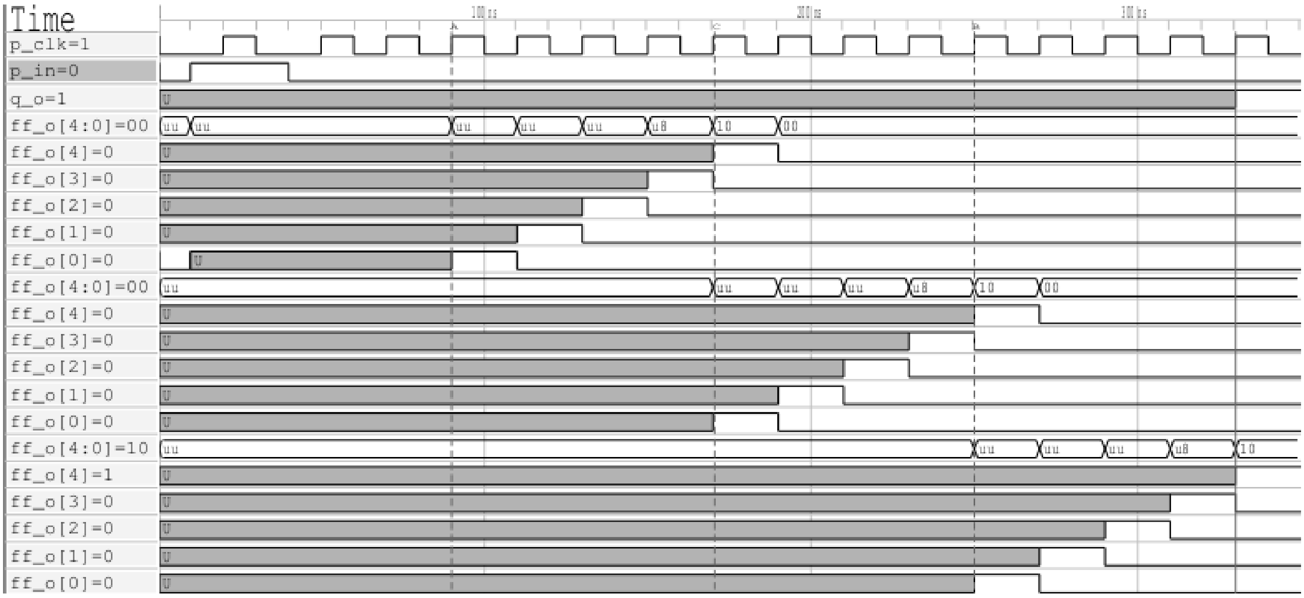

Figure 18: Waveform for Top Test Bench for Test Enabled

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