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

 $^{1}(513) 207-0687$ $^{2}(847) 612-7364$

Contents

1	Pinout Diagram 1.1 Pinout Description	6 7
2	Explaination of Chip Function	8
	2.1 Configuration of Chip	8
3	Inclusion and Explanation of the Test Mode	9
4	Major Design Decisions	9
5	Block Diagrams	10
	5.1 Top Level Diagram	10
	5.2 Bit Slice Design Scheme	11
	5.3 Top Level Test Mode Diagram	11
6	VHDL Models with Test Mode	13
	6.1 Top Level Module	13
	6.2 Slice Modules	15
	6.2.1 LUT	15
	6.2.2 Shift Register	16
	6.3 Gates	17
	6.3.1 D-Flip Flop	17
	6.3.2 2:1 Multiplexer	17
	6.4 VHDL Test Benches	18
7	VHDL Waveform Plots and Results	21
8	Work Division	24
9	Magic Layouts	25
	9.1 LUT Slice Layout	25
	9.2 Shift Slice Layout	25
10	IRSIM Simulations	26
10	10.1 Bit Slice Simulations	26
11	Gate Level Simulations	28
1 ก	HSpice Simulations	29
12	12.1 Bit Slice Simulations	29 29
	12.2 Gate Level Simulations	
13	3 VHDL Modules with Delay	33
	13.1 VHDL Test Bench Modules with Delay	37
	13.2 VHDL Gate Modules with Delay	40
14	VHDL Modules Delay Waveforms	41
15	Floor Plan	44
16	6 Major Design Decisions	44

17 Work Division 45

List of Figures

1	Pinout Diagram of the PBTCKS Chip	6
2	Cycle Timing Diagram for LUT Shift Register	8
3	Cycle Timing Diagram for P Input Shift Register	8
4	Hierarchical Design in Logisim	10
5	Top Level Diagram (8 Inputs, 28-bit slice)	10
6	LUT Slice in Logisim	. 11
7	Hierarchical Test Design in Logisim	. 11
8	Top Level Test Mode Diagram (8 Inputs, 28-bit slice)	12
9	RTL Design of Top Level	14
10	RTL Design of LUT Slice	16
11	RTL Design of Shift Register Slice	17
12	AND Gate of LUT Slice	. 21
13	NAND Gate of LUT Slice	22
14	OR Gate of LUT Slice	22
15	NOR Gate of LUT Slice	22
16	XOR Gate of LUT Slice	23
17	Waveform for Top Test Bench	23
18	Waveform for Top Test Bench for Test Enabled	23
19	LUT Slice Magic Layout	25
20	LUT Slice Internal Magic Layout	25
21	Shift Slice Magic Layout	$\frac{25}{25}$
22	Shift Slice Internal Magic Layout	26
23	LUT Slice with AND gate 'programmed'	26
24	Rising Edge of F output of LUT Slice	26
25	Falling Edge of F output of LUT Slice	27
26	Rising Edge of L3 DFF of LUT Slice	27
20 27	Falling Edge of L3 DFF of LUT Slice	27
28	Rising Edge of DFF of Shift Slice	27
29	Falling Edge of DFF of Shift Slice	27
30	Waveform of DFF of Shift Slice	28
31	Rising Edge of Inverter	28
32	Falling Edge of Inverter	28
33	Waveform of LUT Slice (Includes nodes at each flip flop)	28
34	Hspice LUT Slice Wavefrom	29
35	Hspice Delay of Shift Register Slice	30
36	Hspice Shift Register Slice Waveform	30
37	Hspice Delay of DFF	
38	Hspice Delay of Inveter	
39	Hspice Delay of Mux	32
40	Delay Waveform of DFF	
41	Delay Waveform of MUX	42
42	Delay Waveform of Inverter	42
43	Top Level Delay Waveform	42
44	Top Level Delay Waveform Zoomed In	43
45	Top Level Test Mode Enabled Delay Waveform	43
46	Top Level Test Mode Enabled Delay Waveform Zoomed In	43
47	Floor Plan Design	44
48	Initial Magic Layout Design	44

List of Tables

1	Pinout Description	7
2	AND Gate Truth Table	8
3	Task Assignment	24
4	Delay Times	32
5	Task Assignment	45
Listing	gs	
1	Top Module	13
2	Lookup Table Slice Module	15
3	Lookup Table Slice Module	16
4	D-Flip Flop Module	17
5	2:1 Multiplexer Module	18
6	LUT Slice Test Bench Module	18
7	Top Level Test Bench Module	19
8	Top Module Delay	33
9	Look Up Table Module Delay	33
10	Look Up Table Slice Module Delay	35
11	Test Mode Enabled Test Bench	37
12	Top Module Test Bench	38
13	DFF Module Delay	40
14	Inverter Module Delay	40
15	Inventor Module Delay	11

1 Pinout Diagram

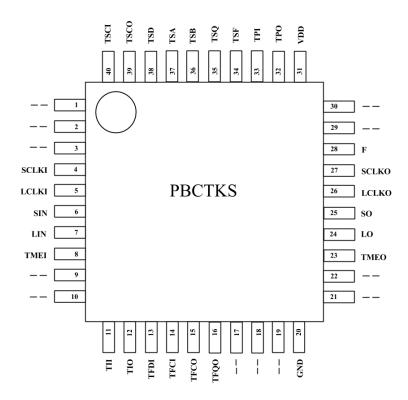


Figure 1: Pinout Diagram of the PBTCKS Chip

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	О	Test Inverter Output
TFDI	13	I	Test Flip-Flop D Input
TFCI	14	I	Test Flip-Flop Clock Input
TFCO	15	О	Test Flip-Flop Clock Output
TFQO	16	О	Test Flip-Flop Q Output
_	17	_	-
_	18	_	_
_	19	_	-
GND	20	_	Ground Reference for I/O Pins
_	21	_	_
_	22	_	_
TMEO	23	О	Test Mode Enabled Output
LO	24	О	Shift Register Output of P input
SO	25	О	Shift Register Output of LUT
LCLKO	26	О	LUT Shift Register Clock Output
SCLKO	27	О	Shift Register Clock Output
F	28	О	Output of Computation of LUT
_	29	_	_
_	30	_	_
VDD	31	I	Test LUT Slice B Input
TPO	32	О	Test P Slice Output
TPI	33	I	TesT P Slice Input
TSF	34	О	Test LUT Slice Mux Output
TSQ	35	О	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	О	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.

\mathbf{A}	\mathbf{B}	\mathbf{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

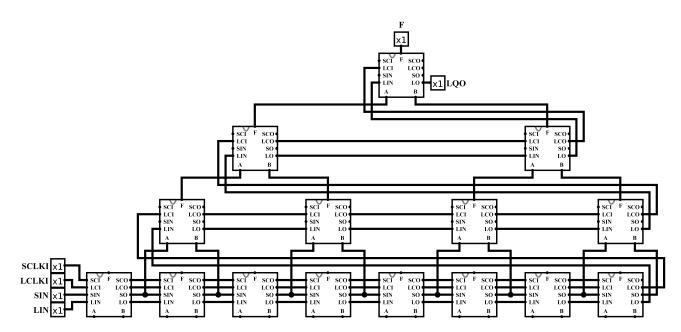


Figure 4: Hierarchical Design in Logisim

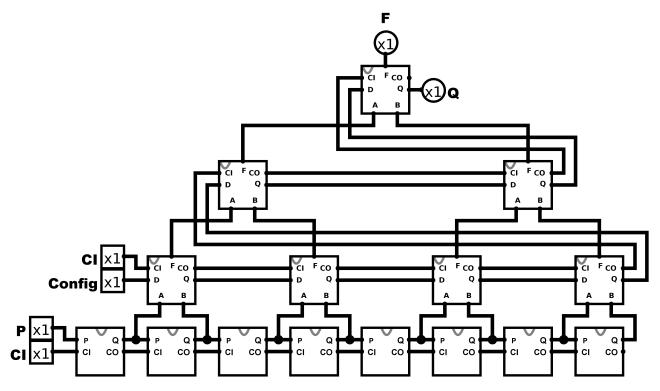


Figure 5: Top Level Diagram (8 Inputs, 28-bit slice)

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

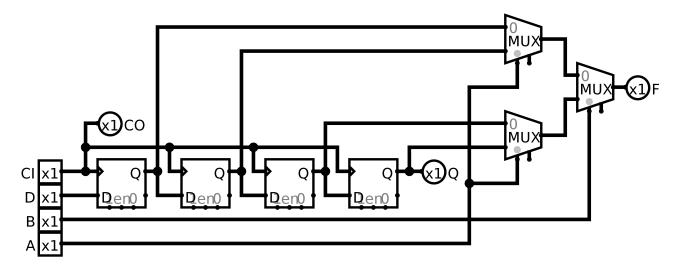


Figure 6: LUT Slice in Logisim

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

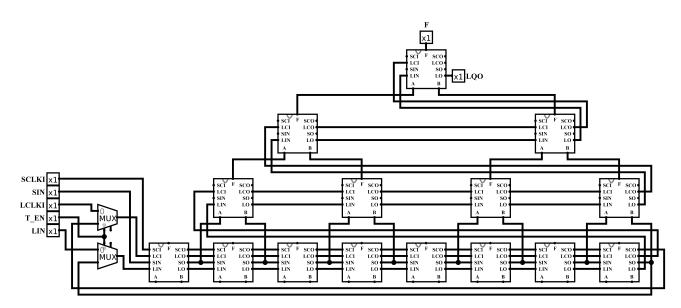


Figure 7: Hierarchical Test Design in Logisim

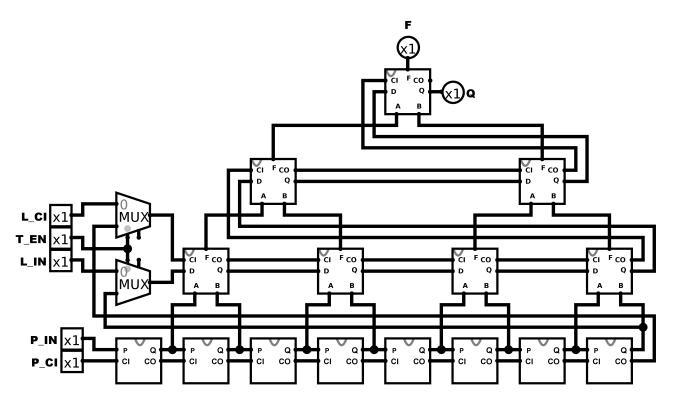


Figure 8: Top Level Test Mode Diagram (8 Inputs, 28-bit slice)

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 (
5
                                      -- number of levels in tree
6
          n
                   : integer := 2
      );
7
       port (
            p_clk : in std_logic;
                                      -- shift register clock
9
            l_clk : in std_logic;
                                      — lut shift register clock
10
                  : in std_logic;
            p_in
                                      — shift register input (P)
11
                  : in std_logic;
            l_in
                                      -- lut shift register input
12
                                      -- test enalbe input
            t₋en
                  : in std_logic;
13
                  : out std_logic; — final output of computation
14
                  : out std_logic
                                      -- final lut shift register output
15
16
   end top;
17
   architecture rtl of top is
19
20
        signal shift_clki
                             : std_logic := '0';
21
        signal shift_clk
                             : std_logic := '0';
22
                            : std_logic := '0';
        signal l_shf_ini
23
                             : std_logic := '0';
        signal l_shf_in
24
        signal p_out
                             : std_logic := '0';
25
26
27
   begin
28
       — test mux connects output of P into input of LUT and use same clock line
29
       t_mux_1 : entity work.mux2x1 port map(|_c|k , p_c|k , t_en , shift_c|ki );
30
       t_mux_2 : entity work.mux2x1 port map(l_in , p_out , t_en , l_shf_ini);
31
32
        t_inv_1 : entity work.invx1 port map(shift_clki, shift_clk);
33
        t_inv_2 : entity work.invx1 port map(l_shf_ini, l_shf_in);
34
35
        lut_1 : entity work.lut
36
        port map(
            s_clk
                    \Rightarrow p_clk,
                    => shift_clk,
            l_clk
            s_in
                    \Rightarrow p_in,
40
                    \Rightarrow I_shf_in ,
            l_in
41
                    \Rightarrow p_out,
42
            t_po
            f _o
                    \Rightarrow f_o,
43
                    \Rightarrow q_o
44
            q_0
        );
45
46
   end rtl;
```

Listing 1: Top Module

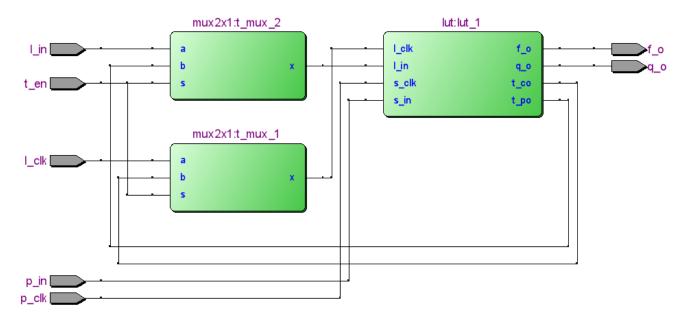


Figure 9: RTL Design of Top Level

6.2 Slice Modules

6.2.1 LUT

59

```
library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;
3
   entity lut_slice is
5
        port (
6
            clk_i
                     : in std_logic;
7
            d
                      : in std_logic;
8
                      : in std_logic;
9
                      : in std_logic;
            b
10
                      : out std_logic;
11
            q
                      : out std_logic
12
13
   end lut_slice;
14
15
   architecture rtl of lut_slice is
16
17
        component dffposx1 is
18
            port (
19
                 clk : in std_logic;
20
                    : in std_logic;
21
                      : out std_logic
22
            );
23
        end component;
24
25
        component mux2x1 is
26
            port (
27
                 b : in std_logic;
28
                 a : in std_logic;
29
                 s : in std_logic;
30
31
                 x : out std_logic
             );
        end component;
33
34
        component invx1 is
35
            port (
36
                 a : in std_logic;
37
                 x : out std_logic
38
            );
39
        end component;
40
41
42
        — flip flop and mux outputs
        signal ff_0 : std_logic_vector(4 downto 0) := (others \Rightarrow '0');
43
                         : std_logic_vector(1 downto 0) := (others \Rightarrow '0');
44
        signal mux_o
                        : std_logic_vector(1 downto 0) := (others \Rightarrow '0');
        signal mux_fo
45
        signal f_muxo
                        : std_logic := '0';
46
47
   begin
48
49
        — shifting in from LSB to MSB
50
        shift_gen_lut : for i in 0 to 3 generate
51
            ff_lut_i : dffposx1
52
            port map(
                 clk \Rightarrow clk_i,
54
                 d \Rightarrow ff_o(i),
55
                     \Rightarrow ff_o(i+1)
56
            );
57
        end generate;
58
```

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

Listing 2: Lookup Table Slice Module

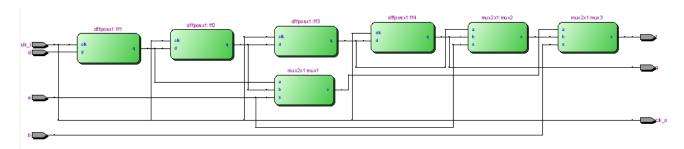


Figure 10: RTL Design of LUT Slice

6.2.2 Shift Register

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

```
24
           ff_-p1\ :\ dffposx1
25
           port map(
26
                clk \Rightarrow clk_i ,
27
                    \Rightarrow p,
                d
28
                      \Rightarrow q
                q
29
           );
30
31
32
           clk_o \le clk_i;
    end rtl;
```

Listing 3: Lookup Table Slice Module

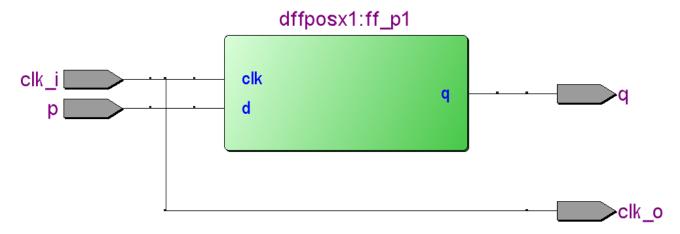


Figure 11: RTL Design of Shift Register Slice

6.3 Gates

6.3.1 D-Flip Flop

```
library ieee;
   use ieee.std_logic_1164.all;
   entity dffposx1 is
        generic (
5
            delay \ : \ time \ := \ 0 \ ps
6
        );
        port (
8
            clk : in std_logic;
9
            d : in std_logic;
10
11
                 : out std_logic
12
   end dffposx1;
   architecture rtl of dffposx1 is begin
15
        process(clk) begin
16
            if rising_edge(clk) then
17
                 q \ll d after delay;
18
            end if;
19
        end process;
20
   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
6
        );
       port (
            b : in std_logic;
9
            a : in std_logic;
10
            s : in std_logic;
11
            x : out std_logic
12
        );
13
   end mux2x1;
14
15
   architecture rtl of mux2x1 is begin
16
17
       x \le not(b) after delay when (s = '0') else
18
             not(a) after delay when (s = '1');
19
20
   end rtl;
21
```

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
5
   end lut_slice_tb;
6
   architecture behavior of lut_slice_tb is
8
9
        signal clk
                           : std_logic := '0';
10
                           : std_logic := '0';
        signal di
11
                           : std_logic := '0';
12
        signal a
        signal b
                           : std_logic := '0';
13
                           : std_logic;
14
        signal q
        signal f
                           : std_logic;
15
16
        component lut_slice
17
             port (
18
                         : in std_logic;
                  clk_i
19
                           : in std_logic;
20
                           : in std_logic;
21
                           : in std_logic;
22
                           : out std_logic;
                  q
24
                           : out std_logic
             );
25
        end component;
26
27
   begin
28
29
        dut : lut_slice
30
        port map(
31
             clk_i
                      \Rightarrow clk,
32
             d
                      \Rightarrow di,
33
             а
                      \Rightarrow a,
34
             b
                      \Rightarrow b,
             q
                      \Rightarrow q,
```

```
f
37
                      => f
        );
38
39
        process
40
41
             procedure clock is begin
42
                  clk \ll '1';
43
                  wait for 10 ns;
44
45
                  clk \ll '0';
                  wait for 10 ns;
             end procedure clock;
47
        begin
49
50
             wait for 10 ns;
51
                <= '1';
52
                 <= '1';
53
54
             -- lut for AND gate
55
             d\,i\;<=\;\,{}^{'}1\;{}^{'};
             wait for 10 ns;
             clock;
59
             di \ll 0;
60
             wait for 10 ns;
61
             clock;
62
63
             di \ll 0;
64
             wait for 10 ns;
65
             clock;
66
             di \ll 0;
             wait for 10 ns;
69
             clock;
70
71
             wait;
72
73
        end process;
74
75
   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
6
       generic (
7
           --- stim_file : string := "test_tree.sim"
8
           stim_file : string := "two_tree.sim"
9
       );
10
   end top_tb;
11
12
   architecture behavior of top_tb is
13
14
       constant n : integer := 2;
15
16
       signal p_clk : std_logic := '0';
17
       signal l_clk : std_logic := '0';
18
                  : std_logic := '0';
       signal p
19
       signal l_in : std_logic := '0';
20
```

```
21
        signal f_o
                     : std_logic;
        signal q_o
                     : std_logic;
22
        signal t_en : std_logic := '0';
23
24
                                                                       downto 0);
        signal p_{in\_vector} : std\_logic\_vector((2**n)-1)
25
        signal [ut\_vector : std\_logic\_vector((((2**n)-1)*4)-1 downto 0);
26
27
        file stimulus : TEXT open read_mode is stim_file;
28
29
        component top
             generic (
                                            -- number of levels in tree
                n
                        : integer := 3
           );
33
            port (
34
                                            -- p shift register clock
                 p_clk : in std_logic;
35
                                            -- lut shift register clock
                 l_clk : in std_logic;
36
                       : in std_logic;
                                            — shift register input (P)
                 p_in
37
                 l_in
                       : in std_logic;
                                            -- lut shift register input
38
                 t_en : in std_logic;
                                            -- test enalbe input
39
                 f_o
                       : out std_logic;
                                            — final output of computation
                      : out std_logic
                                            — lut shift register output
41
42
             );
43
        end component;
44
45
   begin
46
        dut : top
47
        generic map(
48
            n
                   => n
49
50
        port map(
             p_clk \Rightarrow p_clk,
             l_clk \Rightarrow l_clk,
             p_i n \Rightarrow p,
54
                  => l_in ,
             l_in
55
             t_en \Rightarrow t_en
56
             f_o
                   \Rightarrow f_o,
57
                   => q_o
             \mathsf{o}_-\mathsf{p}
58
        );
59
60
        process
             procedure clk_p_in is begin
63
                 p_clk \ll '1';
64
                 wait for 10 ns;
65
                 p_clk \ll '0';
66
                 wait for 10 ns;
67
            end procedure clk_p_in;
68
69
             procedure clk_lut_in is begin
70
                 I_clk \ll '1';
71
                 wait for 10 ns;
                 I_-cIk <= '0';
                 wait for 10 ns;
74
            end procedure clk_lut_in;
75
76
             variable |: line;
77
             variable p_in_str : string(1 to 2**n);
78
             variable l_shf_str: string(1 to ((2**n)-1)*4);
79
80
        begin
81
82
             while not endfile (stimulus) loop
83
```

```
-- load stimulus for this test
                  readline(stimulus, I); read(I, p_in_str);
86
                  p_in_vector <= to_std_logic_vector(p_in_str);</pre>
87
88
                  readline(stimulus \ , \ l\ ); \ read(l\ , \ l\_shf\_str\ );
89
                  lut_vector <= to_std_logic_vector(l_shf_str);</pre>
90
91
                  wait for 50 ns;
92
93
                  — clock in the P input
                  for i in 0 to (2**n)-1 loop
                      p \ll p_i n_v ector(i);
                      wait for 10 ns;
97
                      clk_p_in;
98
                  end loop;
99
100
                  — clock in the "program" (lut functions)
101
                  for i in 0 to (((2**n)-1)*4)-1 loop
102
                      l_in <= lut_vector(i);</pre>
103
                      wait for 10 ns;
                      clk_lut_in;
105
                  end loop;
107
             end loop;
108
109
             report "Test Complete" severity note;
110
             wait;
111
112
        end process;
113
114
    end behavior;
```

Listing 7: Top Level Test Bench Module

7 VHDL Waveform Plots and Results

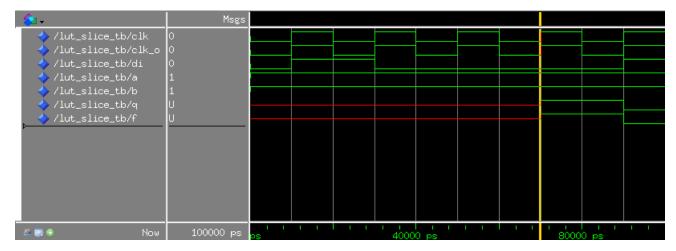


Figure 12: AND Gate of LUT Slice

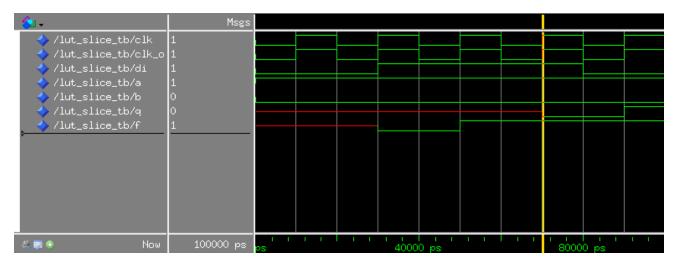


Figure 13: NAND Gate of LUT Slice

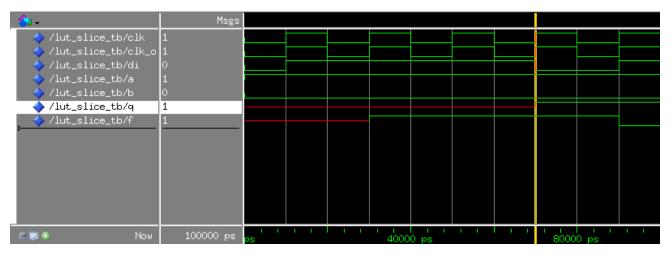


Figure 14: OR Gate of LUT Slice

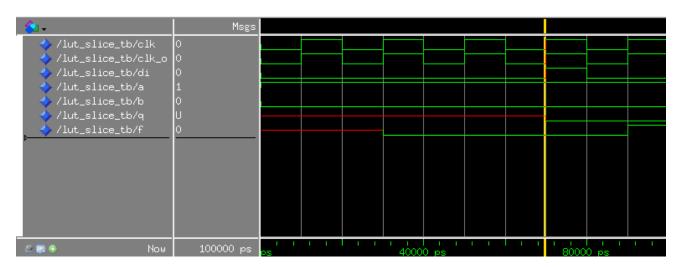


Figure 15: NOR Gate of LUT Slice

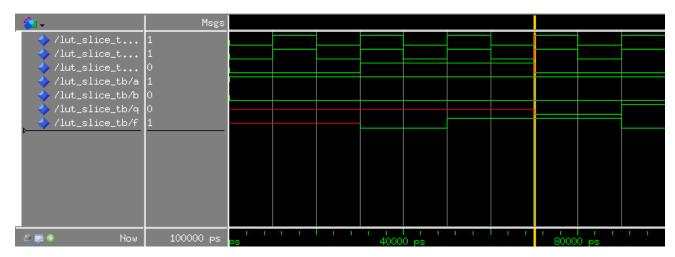


Figure 16: XOR Gate of LUT Slice

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.

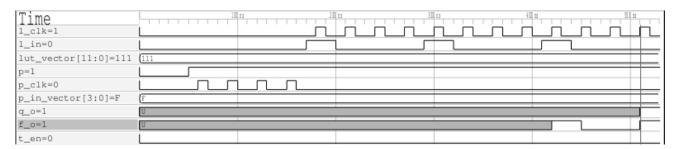


Figure 17: Waveform for Top Test Bench

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).

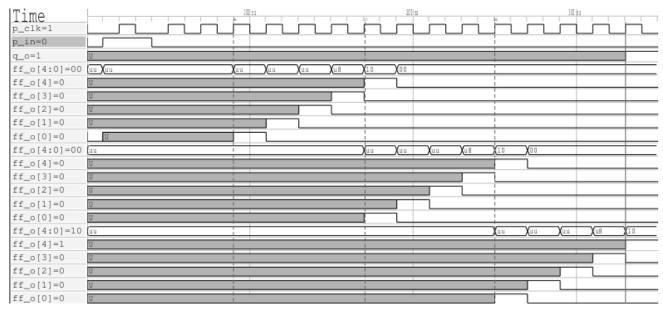


Figure 18: Waveform for Top Test Bench for Test Enabled

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

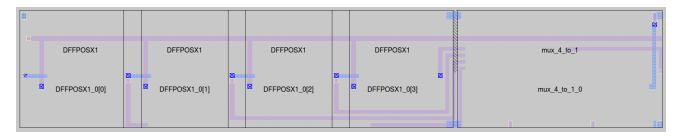


Figure 19: LUT Slice Magic Layout

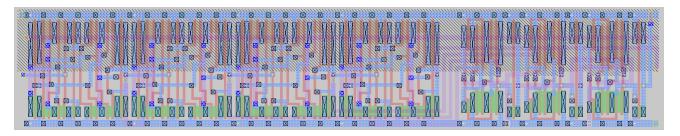


Figure 20: LUT Slice Internal Magic Layout

9.2 Shift Slice Layout



Figure 21: Shift Slice Magic Layout

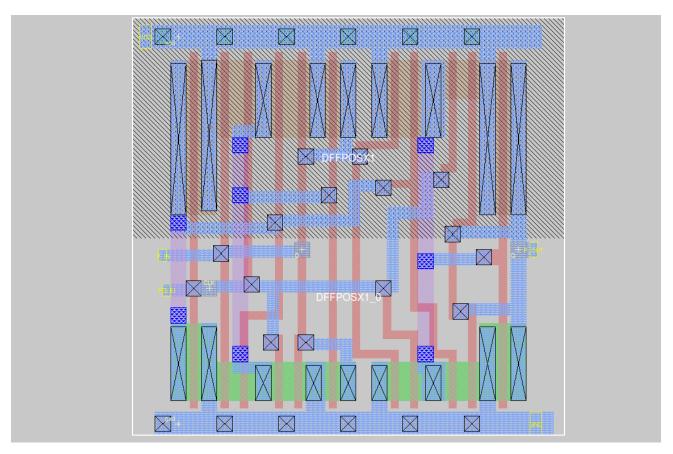


Figure 22: Shift Slice Internal Magic Layout

10 IRSIM Simulations

10.1 Bit Slice Simulations

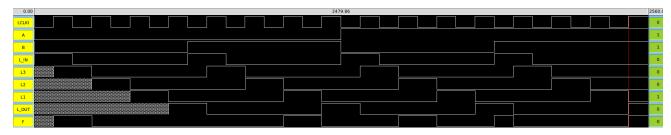


Figure 23: LUT Slice with AND gate 'programmed'

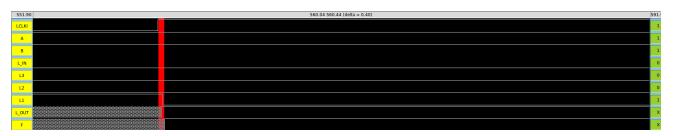


Figure 24: Rising Edge of F output of LUT Slice

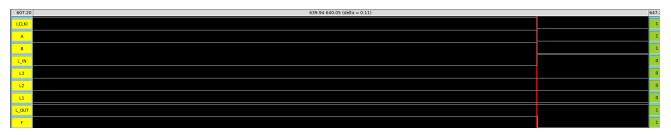


Figure 25: Falling Edge of F output of LUT Slice

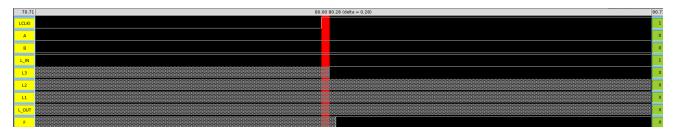


Figure 26: Rising Edge of L3 DFF of LUT Slice

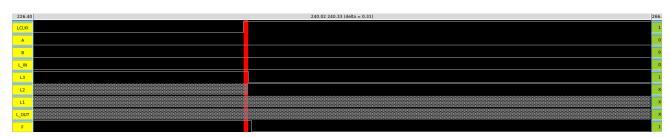


Figure 27: Falling Edge of L3 DFF of LUT Slice

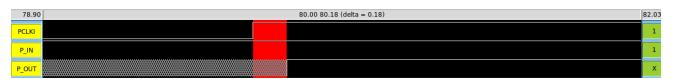


Figure 28: Rising Edge of DFF of Shift Slice

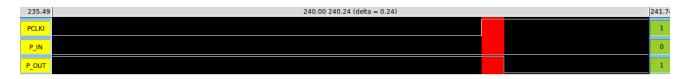


Figure 29: Falling Edge of DFF of Shift Slice

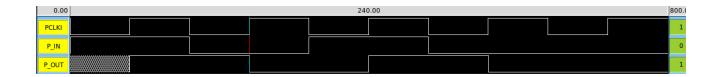


Figure 30: Waveform of DFF of Shift Slice

11 Gate Level Simulations

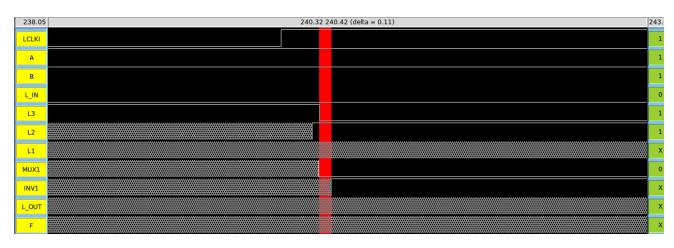


Figure 31: Rising Edge of Inverter

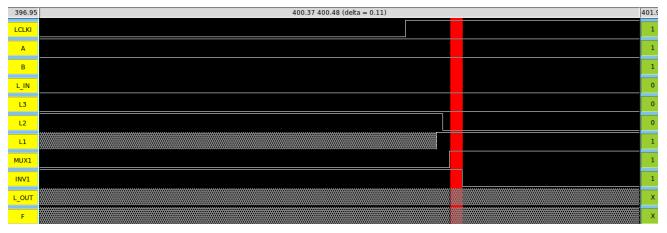


Figure 32: Falling Edge of Inverter

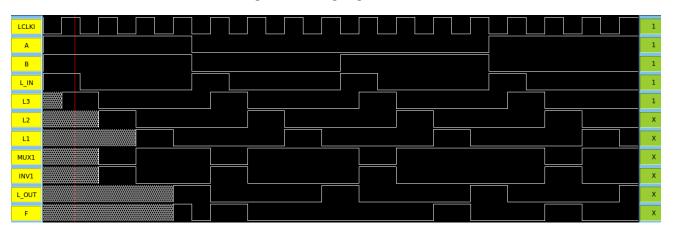


Figure 33: Waveform of LUT Slice (Includes nodes at each flip flop)

12 HSpice Simulations

12.1 Bit Slice Simulations

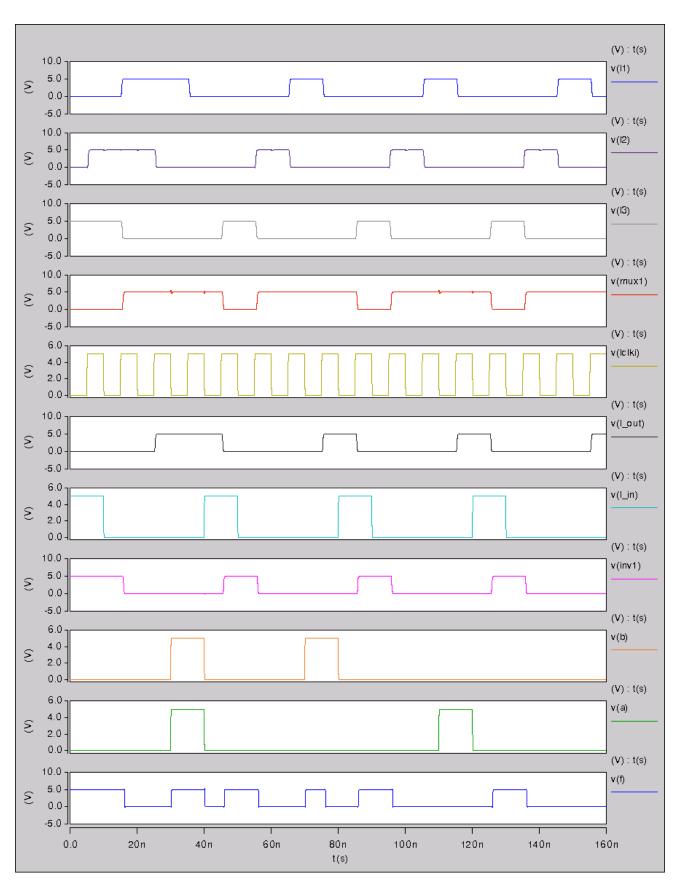


Figure 34: Hspice LUT Slice Wavefrom

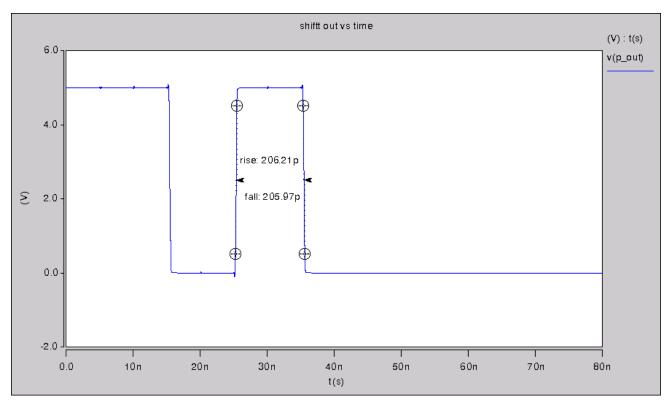


Figure 35: Hspice Delay of Shift Register Slice

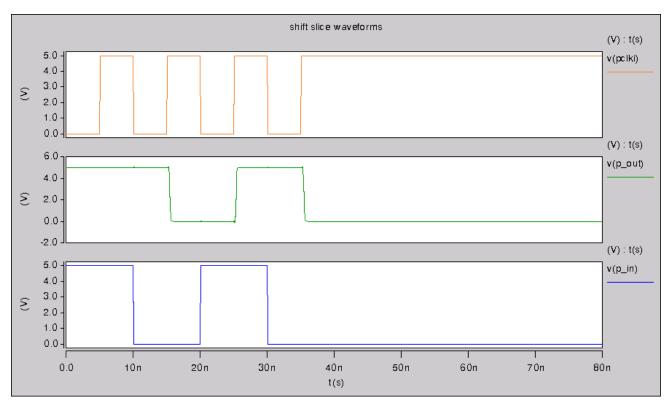


Figure 36: Hspice Shift Register Slice Waveform

12.2 Gate Level Simulations

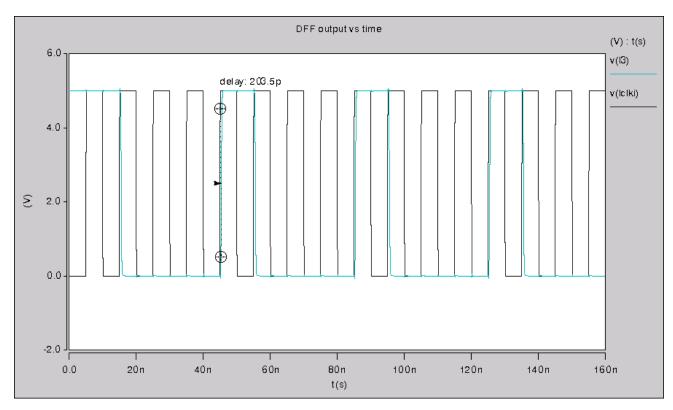


Figure 37: Hspice Delay of DFF

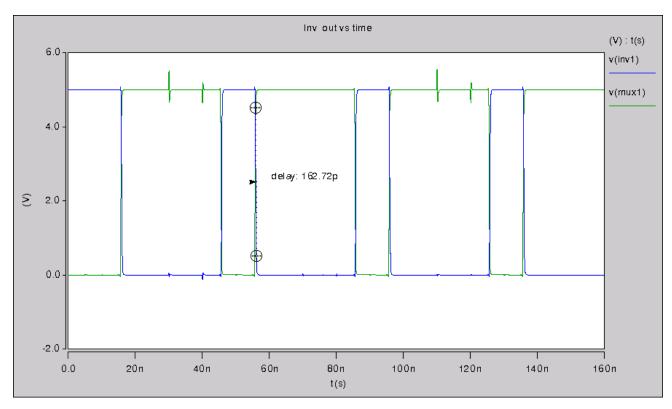


Figure 38: Hspice Delay of Inveter

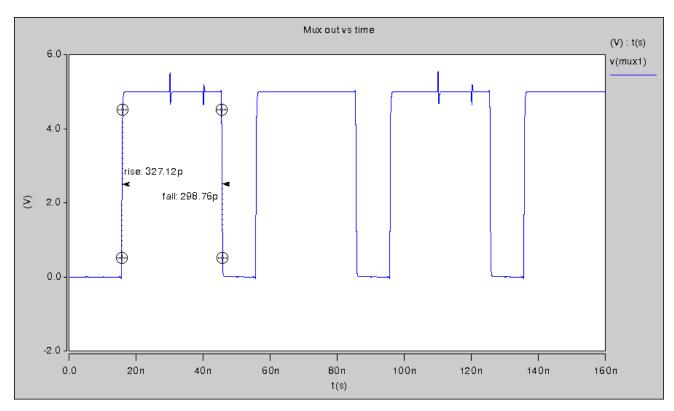


Figure 39: Hspice Delay of Mux

	IRSIM	VHDL	Hspice
Inverter			0.1627 ns
Mux			0.31214 ns
DFF	0.255 ns	$0.198 \; \mathrm{ns}$	0.20305 ns

Table 4: Delay Times

13 VHDL Modules with Delay

```
library ieee;
   use ieee.std_logic_1164.all;
   entity top_del is
4
        generic (
5
                   : integer := 2
                                       -- number of levels in tree
           n
6
7
        port (
8
            p_clk : in std_logic;
                                       -- shift register clock
9
            l_clk : in std_logic;
                                       -- lut shift register clock
10
            p_in : in std_logic;
                                       — shift register input (P)
11
                   : in std_logic;
                                       -- lut shift register input
            l_in
12
            t_en : in std_logic;
                                       -- test enalbe input
13
            f_{-}o
                   : out std_logic; — final output of computation
14
                   : out std_logic
                                       -- final lut shift register output
15
            \mathsf{o}_{-}\mathsf{p}
        ):
16
   end top_del;
17
18
   architecture rtl of top_del is
19
20
                              : std_logic := '0';
        signal shift_clki
21
        signal shift_clk
                              : std_logic := '0';
22
                             : std_logic := '0';
        signal l_shf_ini
23
        signal l_shf_in
                              : std_logic := '0';
24
        signal p_out
                              : std_logic := '0';
25
26
   begin
27
28
        — test mux connects output of P into input of LUT and use same clock line
29
        t_mux_1 : entity work.mux2x1_del port map(|_clk , p_clk , t_en , shift_clki );
30
        t_mux_2 : entity work.mux2x1_del port map(l_in , p_out , t_en , l_shf_ini);
31
32
        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);
34
35
        lut_1 : entity work.lut_del
36
        port map(
37
            s_clk
                     \Rightarrow p_clk,
38
                     => shift_clk,
            I_clk
39
            s_in
                     \Rightarrow p_in,
40
            l_in
                     \Rightarrow l_shf_in,
41
            t_{-}po
                     \Rightarrow p_out,
42
            f_o
                     \Rightarrow f_o,
43
44
            q_{-}o
                     \Rightarrow q_o
45
        );
46
   end rtl;
47
```

Listing 8: Top Module Delay

```
library ieee;
   use ieee.std_logic_1164.all;
2
   entity lut_del is
4
       generic (
5
                    : integer := 2
                                      -- number of levels in tree
6
           n
7
       port (
                    : in std_logic;
                                      — shift register clock
           s_clk
                    : in std_logic;
           l_clk
                                      — lut shift register clock
10
```

```
11
            s_in
                     : in std_logic;
                                       — shift register input (P)
            l_in
                     : in std_logic;
                                       — lut shift register input
12
            t_{\,-}p\,o
                     : out std_logic; — p_out for test mode
13
            f_o
                     : out std_logic; — final output of computation
14
            q_0
                     : out std_logic
                                       — final lut shift register output
15
        ):
16
   end lut_del;
17
18
   architecture rtl of lut_del is
19
20
       -- tree diagram for n=3
21
       -- row 0 : A
22
       -- row 1 : BC
23
       -- row 2 : DEFG
24
       -- row 3 : shifter
25
26
       -- lut connection diagram
27
        -- I_{-}in -> D -> E -> F -> G -> B -> C -> A
28
29
        \color{red} \textbf{component} \hspace{0.2cm} \textbf{dffpos} \\ \textbf{x1\_del} \hspace{0.2cm} \textbf{is} \\
            port (
                 clk: in std_logic;
32
                d : in std_logic;
33
                     : out std_logic
34
35
            );
        end component;
36
37
        component lut_slice_del is
38
            port (
39
                 clk_i
                         : in std_logic;
40
                         : in std_logic;
                d
                         : in std_logic;
                а
42
                         : in std_logic;
                b
43
                         : out std_logic;
44
                q
                         : out std_logic
45
            );
46
        end component;
47
48
       -- carray_array(row, col)
49
        type carry_array is array (0 to n, 0 to 2**n) of std_logic;
50
        signal clk_c : carry_array;
51
       — carries select outputs of one
53
        — row to select inputs of next one
54
        signal r_c : carry_array;
55
56
        — carry output of each lut shift
57
        -- register
58
        signal l_c
                    : carry_array;
59
60
        — input shift register carries
61
        63
       -- input shift regstier clock carries
64
        signal p_clk : std_logic_vector(2**n downto 0) := (others \Rightarrow '0');
65
66
   begin
67
68
         - generate the input shift register
69
        shift_gen : for i in 0 to (2**n)-1 generate
70
71
            ff_i : dffposx1_del
72
            port map(
                clk \implies s_-clk,
                d \Rightarrow s_c(i),
```

```
75
                       \Rightarrow s<sub>-</sub>c(i+1)
              );
76
         end generate shift_gen;
77
78
         -- generate the tree
79
         \label{level_gen} \mbox{level\_gen} \ : \ \mbox{for level in 0 to } \ n{-}1 \ \mbox{generate}
80
              lut_gen : for i in 0 to (2**level)-1 generate
81
                   lut_i : lut_slice_del
82
                   port map(
83
                        clk_i \Rightarrow l_clk,
                               \Rightarrow l_c(level, i),
                        d
                               \Rightarrow l_c(level, i+1),
                        q
                               \Rightarrow r_c(level+1, i*2),
87
                        а
                        b
                               \Rightarrow r_c(level+1, i*2+1),
88
                        f
                               \Rightarrow r_c(level, i)
89
                    );
90
              end generate;
91
         end generate;
92
93
         — output the value of our function
         q_{-o} \ll I_{-c}(0, 1);
         f_{-o} \ll r_{-c}(0, 0);
96
97
         -- connect each row of LUTs together
98
         — first slice in row i connects to
99
          -- last slice in row i\!+\!1
100
         lut_connect : for level in 0 to n-2 generate
101
              I_c(level, 0) <= I_c(level+1, (2**level+1));
102
         end generate;
103
104
         — connect input shift register to bottom row of slices
105
         shift\_connect : for i in 0 to (2**n)-1 generate
106
              r_{-}c(n, i) \le s_{-}c(i+1);
107
         end generate;
108
109
         -- connect input to shift register
110
         s_c(0) \ll s_{in};
111
112
         — connect input to first lut (bottom row)
113
         I_{-c}(n-1, 0) <= I_{-in};
114
         — use last slice of P shift register
         — feed it into LUT input for test mode
117
         t_po <= s_c(2**n);
118
119
    end rtl;
120
```

Listing 9: Look Up Table Module Delay

```
library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;
3
   entity lut_slice_del is
5
6
       port (
            clk_i
                     : in std_logic;
7
                     : in std_logic;
            d
                     : in std_logic;
9
            а
                     : in std_logic;
            b
10
                     : out std_logic;
11
            q
                     : out std_logic
12
        ):
13
   end lut_slice_del;
```

```
15
   architecture rtl of lut_slice_del is
16
17
        component dffposx1_del is
18
            port (
19
                 clk : in std_logic;
20
                     : in std_logic;
21
                     : out std_logic
22
23
        end component;
25
        component invx1_del is
26
27
            port (
                     : in std_logic;
28
                а
                     : out std_logic
29
                 Х
            );
30
        end component;
31
32
        component mux2x1_del is
33
            port (
                 b : in std_logic;
                 a : in std_logic;
36
37
                 s : in std_logic;
                x : out std_logic
38
39
            );
        end component;
40
41
        — flip flop and mux outputs
42
        signal ff_o
                       : std_logic_vector(4 downto 0) := (others => '0');
43
                         : std_logic_vector(1 downto 0) := (others => '0');
        signal mux_o
44
                         : std_logic_vector(1 downto 0) := (others \Rightarrow '0');
        signal mux_fo
        signal f_muxo
                        : std_logic := '0';
46
47
   begin
48
49
        — shifting in from LSB to MSB
50
        shift_gen_lut : for i in 0 to 3 generate
51
            ff_lut_i : dffposx1_del
52
            port map(
53
                 clk \Rightarrow clk_i
54
                 d \Rightarrow ff_o(i),
                     \Rightarrow ff<sub>-</sub>o(i+1)
            );
        end generate;
58
59
       -- lut shift out is output from prev ff
60
       q \ll ff_o(4);
61
        ff_-o(0) \ll d;
62
63
        -- select first two outputs of LUT
64
        -- on sel line A
65
                                                 ff_{-o}(2), a, mux_{-o}(0);
        mux1 : mux2x1_del port map(ff_o(1),
        inv1 : invx1_del  port map(mux_o(0), mux_fo(0));
67
68
       — select last two outputs of LUT
69
       -- on sel line A
70
       mux2 : mux2x1_del port map(ff_o(3), ff_o(4), a, mux_o(1));
71
        inv2 : invx1_del  port map(mux_o(1), mux_fo(1));
72
73
       -- select the outputs from
74
       -- each mux on sel line B
75
       mux3 : mux2x1_del port map(mux_fo(0), mux_fo(1), b, f_muxo);
77
       -- invert mux due to func of mux: y = !(S?(A:B))
```

```
inv3 : invx1_del port map(f_muxo, f);
so 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
6
   end top_test_del_tb;
   architecture behavior of top_test_del_tb is
9
10
        constant n : integer := 2;
11
12
        component top_del
13
            generic (
14
                                            -- number of levels in tree
                        : integer := 3
15
                n
           );
16
            port (
17
                 p_clk : in std_logic;
                                            -- p shift register clock
18
                 l_clk : in std_logic;
                                            -- lut shift register clock
19
                 p_in
                       : in std_logic;
                                            — shift register input (P)
20
                 l_in
                       : in std_logic;
                                            -- lut shift register input
21
22
                 t_en
                       : in std_logic;
                                            -- test enalbe input
                                           -- final output of computation
23
                 f_o
                        : out std_logic;
                                            -- lut shift register output
                        : out std_logic
24
                 q_0
            );
25
        end component;
26
27
                                        '0':
        signal p_clk : std_logic :=
28
                                        '0';
        signal l_clk : std_logic :=
29
        signal p
                      : std_logic :=
30
        signal l_in
                     : std_logic :=
31
        signal f_o
                      : std_logic;
32
        signal q_o
                      : std_logic;
33
        signal t_en : std_logic := '0';
34
35
36
   begin
37
        dut : top_del
38
        generic map(
39
            n
                   => n
40
41
        port map(
42
            p_clk \Rightarrow p_clk,
43
            l_clk \Rightarrow l_clk,
44
45
            p_in \Rightarrow p,
                   => l_in ,
46
            l_{-in}
                  \Longrightarrow t_en,
47
            t_en
                   \Rightarrow f_o,
            f_o
48
                   => q_o
49
            o_-\,p
        );
50
51
        process
52
53
            procedure clk is begin
54
```

```
p_clk \ll '1';
                 wait for 10 ns;
56
                 p_clk \le '0';
57
                 wait for 10 ns;
58
            end procedure clk;
59
60
        begin
61
62
63
            wait for 10 ns;
            -- enable test mode
            t_-en <= '1';
            -- clock in p data
67
            p \ll 1';
68
            wait for 10 ns;
69
            clk;
70
71
            -- clock in p data
72
            p \ll '0';
73
            wait for 10 ns;
            clk;
76
            -- # of FF's per LUT: 4
77
            -- \# \text{ of LUT FF's} : (((2**n)-1)*4)
78
            -- # of P FF's
                                  : 2**n
79
            -- subtract 2 since we just loaded in two inputs
80
            for i in 0 to ((((2**n)-1)*4)+2**n)-2-1 loop
81
82
            end loop;
83
84
            wait;
        end process;
87
   end behavior;
89
```

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
6
        generic (
            --- stim_file : string := "test_tree.sim"
            \mathsf{stim\_file} \; : \; \mathsf{string} \; := \; " \, \mathsf{two\_tree.sim}"
        );
10
   end top_del_tb;
11
12
   architecture behavior of top_del_tb is
13
14
        constant n : integer := 2;
15
16
        signal p_clk : std_logic := '0';
17
        signal l_clk : std_logic := '0';
18
                      : std_logic := '0';
        signal p
        signal l_in : std_logic := '0';
20
        signal f_o
                      : std_logic;
21
        signal q_o
                      : std_logic;
22
        signal t_en : std_logic := '0';
23
24
        signal p_in_vector : std_logic_vector((2**n)-1)
                                                                       downto 0);
```

```
26
        signal lut_vector : std_logic_vector((((2**n)-1)*4)-1 downto 0);
27
        file stimulus : TEXT open read_mode is stim_file;
28
29
        component top_del
30
            generic (
31
                        : integer := 3
                                            -- number of levels in tree
32
           );
33
            port (
34
                 p_clk : in std_logic;
                                            -- p shift register clock
                                            -- lut shift register clock
                 l_clk : in std_logic;
                       : in std_logic;
                                            — shift register input (P)
                 p_in
                 l_{-in}
                       : in std_logic;
                                            — lut shift register input
38
                       : in std_logic;
                                            -- test enalbe input
39
                 t₋en
                 f_o
                        : out std_logic;
                                           — final output of computation
40
                        : out std_logic
                                            -- lut shift register output
41
                 q_0
            );
42
        end component;
43
44
   begin
45
46
47
        dut : top_del
48
        generic map(
49
            n
                   => n
50
        port map(
51
            p_clk \Rightarrow p_clk,
52
             l_clk \Rightarrow l_clk,
53
                  \Rightarrow p,
            p_in
54
             l_{-}in
                   ⇒ l_in ,
55
            t_en
                  \Rightarrow t_en,
            f_o
                   \Rightarrow f_o,
             o_{-}p
                   => q_o
        );
59
60
        process
61
62
            procedure clk_p_in is begin
63
                 p_clk \ll '1';
64
                 wait for 10 ns;
65
                 p_clk \le '0';
                 wait for 10 ns;
            end procedure clk_p_in;
69
            procedure clk_lut_in is begin
70
                 I_clk \ll '1';
71
                 wait for 10 ns;
72
                 l_clk \ll '0';
73
                 wait for 10 ns;
74
            end procedure clk_lut_in;
75
76
             variable |: line;
             variable p_in_str : string(1 to 2**n);
             variable l_shf_str: string(1 to ((2**n)-1)*4);
79
        begin
81
82
            while not endfile(stimulus) loop
83
84
                 -- load stimulus for this test
85
                 readline(stimulus, I); read(I, p_in_str);
86
                 p_in_vector <= to_std_logic_vector(p_in_str);</pre>
88
                 readline(stimulus, I); read(I, I_shf_str);
```

```
lut_vector <= to_std_logic_vector(l_shf_str);</pre>
90
91
                  wait for 50 ns;
92
93
                    – clock in the Pinput
94
                  for i in 0 to (2**n)-1 loop
95
                       p <= p_in_vector(i);</pre>
96
                       wait for 10 ns;
97
                       clk_p_in;
98
                  end loop;
                  -- clock in the "program" (lut functions)
101
                  for i in 0 to (((2**n)-1)*4)-1 loop
102
                       l_in <= lut_vector(i);</pre>
103
                       wait for 10 ns;
104
                       clk_lut_in;
105
                  end loop;
106
107
             end loop;
108
109
             report "Test Complete" severity note;
110
111
             wait;
112
         end process;
113
114
    end behavior;
115
```

Listing 12: Top Module Test Bench

13.2 VHDL Gate Modules with Delay

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

Listing 13: DFF Module Delay

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

entity invx1_del is
generic(
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;</pre>
```

Listing 14: Inverter Module Delay

```
library ieee;
   use ieee.std_logic_1164.all;
    entity mu \times 2 \times 1_- del is
         generic (
5
              \mathsf{delay} \;:\; \mathsf{time} \;:=\; 315 \;\; \mathsf{ps}
6
         );
7
         port (
8
              b : in std_logic;
9
             a : in std_logic;
10
             s : in std_logic;
             x : out std_logic
13
   end mux2x1_del;
14
15
   architecture rtl of mux2x1_del is begin
16
17
         x \le not(b) after delay when (s = '0') else
18
               not(a) after delay when (s = '1');
19
20
   end rtl;
21
```

Listing 15: Inverter Module Delay

14 VHDL Modules Delay Waveforms



Figure 40: Delay Waveform of DFF

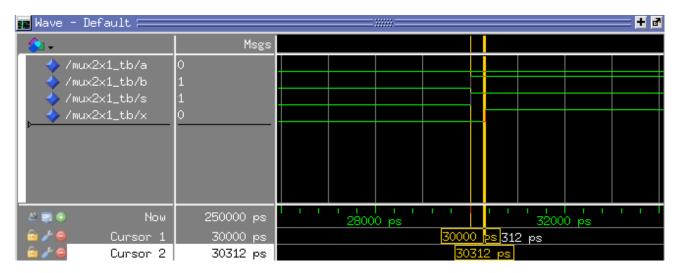


Figure 41: Delay Waveform of MUX



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.

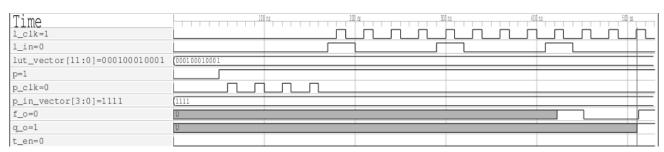


Figure 43: Top Level Delay Waveform

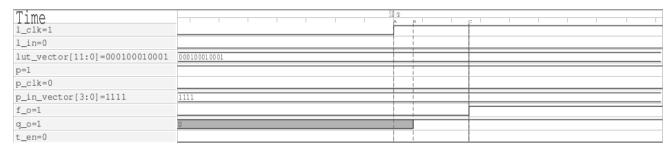


Figure 44: Top Level Delay Waveform Zoomed In

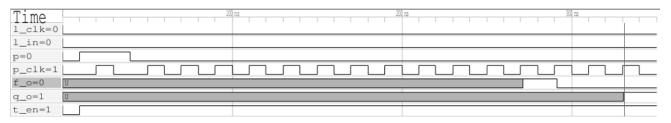


Figure 45: Top Level Test Mode Enabled Delay Waveform

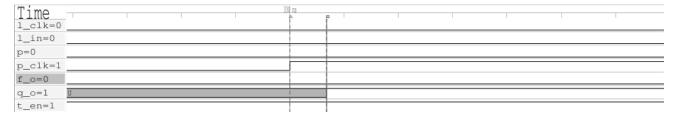


Figure 46: Top Level Test Mode Enabled Delay Waveform Zoomed In

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

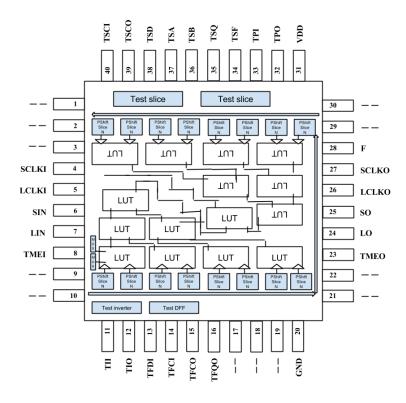


Figure 47: Floor Plan Design

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.

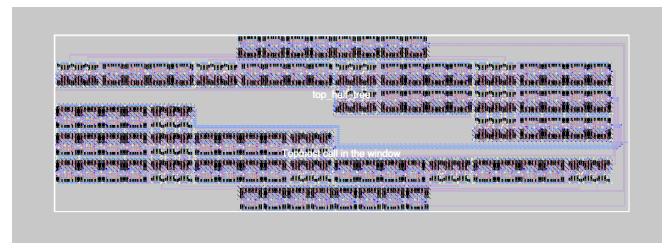


Figure 48: Initial Magic Layout Design

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