

INTRANEX

INTRANEX is a **programmable interconnect network** that accepts a N bit input W and produces a N bit output Z. The interconnect can be programmed to realize any mapping from W to Z.

University of Cincinnati - EECE6080
FALL 2013

Max Thrun
973 919 6593
max.thrun@gmail.com
(Coordinator)

Xiaohu Qi
513 652 2075
qixiaohuihaha@gmail.com

Contents

1	Pinout Diagram	4
2	Chip Functionality	6
2.1	Configuring the Programmable Interconnect Network	6
2.2	Loading and reading a value	6
2.3	Test Mode	7
3	Design Decisions	7
4	Block Diagrams	8
4.1	Top Level	8
4.2	Top Level With Test Mode	8
4.3	Top Level Bit Sliced	9
4.4	Parallel Load Shift Register	10
4.4.1	Bit-slicing Scheme	10
4.4.2	Bit-Slice	10
4.5	Programmable Interconnect Network	11
4.5.1	Bit-slicing Scheme	11
4.5.2	Bit-Slice	11
5	VHDL Models	12
5.1	Top Level	12
5.2	PIN	13
5.3	PIN Slice	14
5.4	Shifter	15
5.5	Shifter Slice	16
5.6	Gates	17
6	VHDL Test Benches	18
6.1	Top Level Functional	18
6.2	Top Level Test Mode	20
6.3	PIN Slice	21
6.4	Shifter Slice	22
7	VHDL Test Bench Results	23
7.1	Top Level Functional	23
7.2	Top Level Test Mode	23
8	Work Division	23

List of Figures

1	Pinout Diagram	4
2	PIN Configuration	6
3	Loading a value	6
4	Loading a value and reading the result	6
5	Enabling test mode and loading all DFFs	7
6	3 INTRANEX Chain	7
7	Top Level Block Diagram (3-Bit Configuration)	8
8	Top Level Block Diagram Showing Test Mode Logic (3-Bit Configuration)	8
9	Top Level Bit Sliced Block Diagram (3-Bit Configuration)	9
10	Parallel Load Bit-Sliced Shifter Register (3-Bit Configuration)	10
11	Parallel Load Shifter Register Bit-Slice	10
12	Bit-Sliced Programmable Interconnect Network (3-Bit Configuration)	11
13	Programmable Interconnect Network Bit-Slice	11
14	Top Level Generated RTL Diagram	12
15	Pin Generated RTL Diagram	13
16	Pin Slice Generated RTL Diagram	14
17	Shifter Generated RTL Diagram	15
18	Shifter Slice Generated RTL Diagram	16
19	Top Level Functional Test Bench Waveform	23
20	Top Level Test Mode Test Bench Waveform	23

List of Tables

1	Pin Descriptions	5
2	Task Assignment	23

Listings

1	Top Level VHDL Module	12
2	PIN VHDL Module	13
3	PIN Slice VHDL Module	14
4	Parallel Load Shifter VHDL Module	15
5	Parallel Load Shifter Slice VHDL Module	16
6	AOI21X1 VHDL Module	17
7	DFFPOSX1 VHDL Module	17
8	INVX1 VHDL Module	17
9	MUX2X1 VHDL Module	17
10	Top Level VHDL Test Bench	18
11	Python Vector Generator	19
12	Top Level Test Mode VHDL Test Bench	20
13	PIN Slice VHDL Test Bench	21
14	Parallel Load Shifter Slice VHDL Test Bench	22

1 Pinout Diagram

The pinout diagram for INTRANEX is shown below in Figure 1. Pins that are currently unutilized will be assigned to various internal logic signals once the floorplan is finalized. Note the symmetry of the core functionality. This was done so that multiple INTRANEX chip can be directly chained together with minimal routing effort during PCB layout.

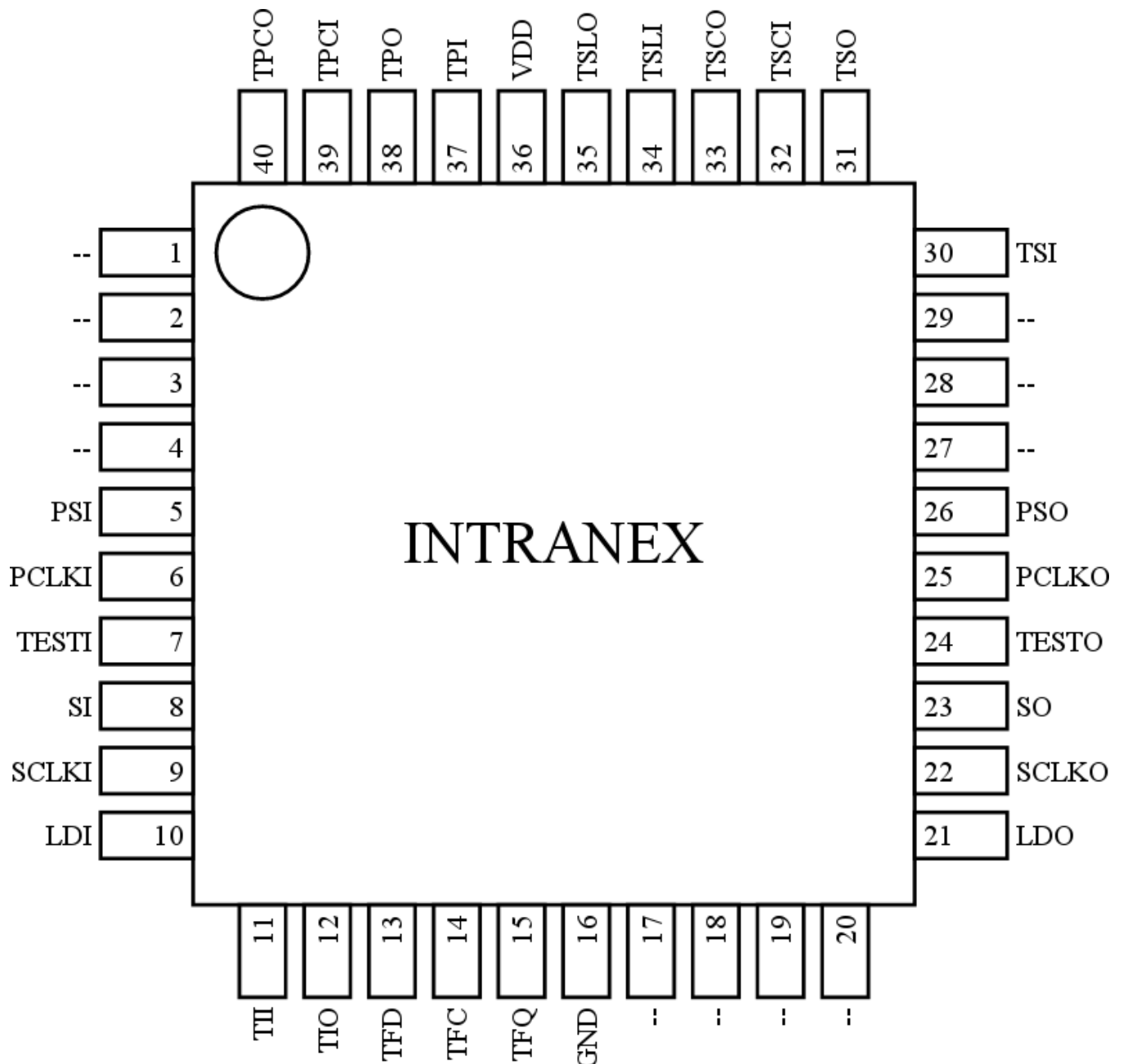


Figure 1: Pinout Diagram

The table below shows each pin and its corresponding name, type, and a brief description of its functionality. Type is of either I (Input), O (Output), or P (Power).

Pin #	Name	Type	Description
1	–	–	–
2	–	–	–
3	–	–	–
4	–	–	–
5	PSI	I	PIN serial input
6	PCLKI	I	PIN clock input
7	TESTI	I	Test Mode enable input
8	SI	I	Serial input
9	SCLKI	I	Serial clock input
10	LDI	I	Parallel load input
11	TII	I	Test inverter input
12	TIO	O	Test interter output
13	TFD	I	Test flip-flop D input
14	TFC	I	Test flip-flop clock input
15	TFQ	O	Test flop-flop Q output
16	GND	P	–
17	–	–	–
18	–	–	–
19	–	–	–
20	–	–	–
21	LDO	O	Parallel load output
22	SCLKO	O	Serial clock output
23	SO	O	Serial output
24	TESTO	O	Test Mode enable output
25	PCLKO	O	PIN clock output
26	PSO	O	PIN serial output
27	–	–	–
28	–	–	–
29	–	–	–
30	TSI	I	Test shift slice serial input
31	TSO	O	Test shift slice serial output
32	TSCI	I	Test shift slice clock input
33	TSCO	O	Test shift slice clock output
34	TSLI	I	Test shift slice load input
35	TSLO	O	Test shift slice load output
36	VDD	P	–
37	TPI	I	Test pin slice serial input
38	TPO	O	Test pin slice serial output
39	TPCI	I	Test pin slice clock input
40	TPCO	O	Test pin slice clock output

Table 1: Pin Descriptions

2 Chip Functionality

The major function of this chip is to take an N bit input and translate any bit position to any other position. This allows for commonly desired functionality such as bit reversing or nibble swapping. To accomplish this we use a N by N bit interconnect network known as the PIN (Programmable Interconnect Network). The PIN is configured to perform the desired bit mappings by clocking in the mappings using the PSI (PIN Shift Input) and PCLKI (PIN Clock Input) pins. The value to be manipulated, called the Input Value, Shift Value or Shifter Value, is then clocked in serially using the SI (Shifter Input) and SCLKI (Shifter Clock Input) pins. To obtain the result the LDI (Load Input) pin is pulled high and the SCLKI pin is pulsed to latch the result in to the shift register. Once the result is latched the LDI pin is de-asserted and the result can be clocked out of the SO (Shifter Output) pin. Note that the input value is clocked in MSB first and the output value is clocked out MSB first as well.

2.1 Configuring the Programmable Interconnect Network

A timing diagram illustrating the PIN configuration process for a 3-Bit INTRANEX is shown below. For a 3-Bit input value a 3x3 grid is required resulting in a PIN configuration vector of 9 Bits. The mapping for each of these bits is also labeled and will be explained further in later sections.

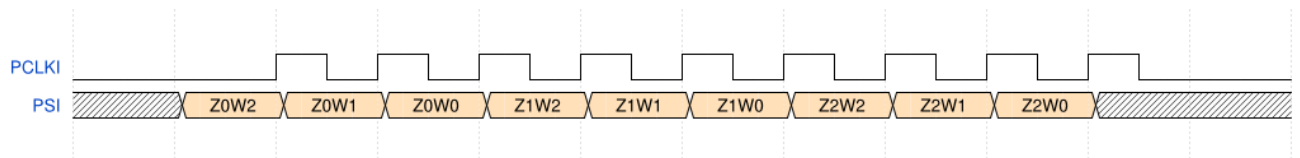


Figure 2: PIN Configuration

2.2 Loading and reading a value

Loading an input value is achieved by clocking the value in on the SI pin using the SCLKI pin. The LDI pin must be held low during this operation. The diagram below illustrates this process and shows the bit definitions of the value being clocked in.

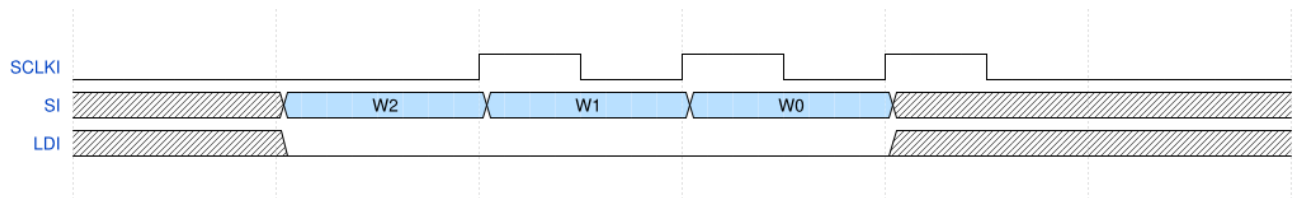


Figure 3: Loading a value

After the value has been loaded in the result is clocked out in a similar fashion. To first latch the result the LDI pin needs to be held high and the SCLKI pin pulsed. The MSB of the result is now available on the SO pin. The LDI pin should now be held low while clocking out the remaining result bits.

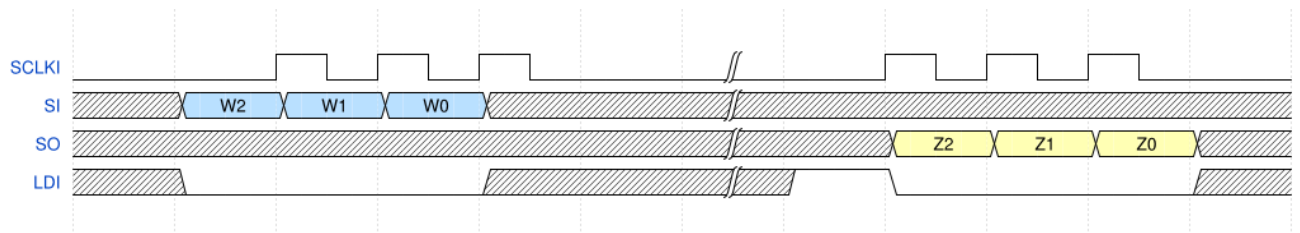


Figure 4: Loading a value and reading the result

2.3 Test Mode

Test Mode is enabled by pulling the **TESTI** pin high. When this occurs the output of the internal input value shift register is rerouted to connect to the input of the PIN network bypassing its normal **PSI** input. Additionally the **SCLK0** signal is also routed to the PIN bypassing its normal **PCLKI** signal. Finally the **PS0** signal is routed to the **S0** pin. This allows values that are clocked in via the **SI** pin to propagate through the shifter and then through the PIN and then out the **S0** pin. The fact that the values come out the **S0** pin allows multiple INTRANEX chips to be directly chained and tested in circuit using only the **SI** and **SCLKI** pins of the first chip in the chain. Note that the **LDI** pin must be held low during this entire operation in order to ensure proper shifting through the input value shift register.

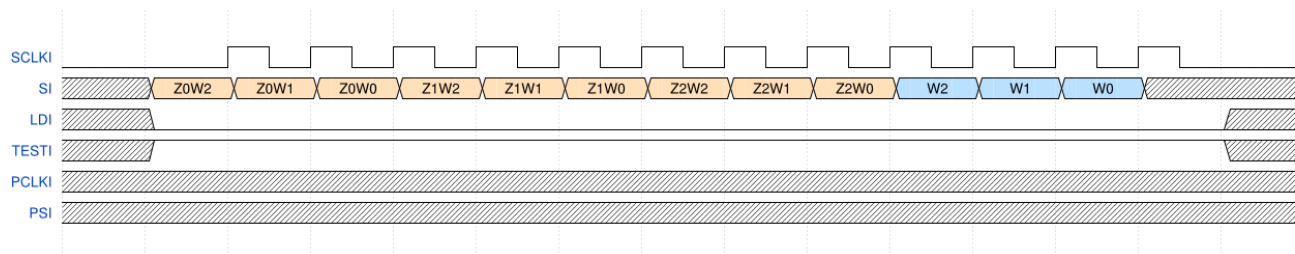


Figure 5: Enabling test mode and loading all DFFs

3 Design Decisions

When evaluating design concepts and possible solutions we prioritized a few key factors that we wanted to achieve. The first is a fully bit-sliced solution where each slice can directly connect to the next with minimal wiring overhead and zero additional logic. This will allow us to utilize Magics `Array` functionality to quickly build up our chip and allow us to easily scale to any desired size. As we see in later sections we were able to achieve a fully bit-sliced design with zero logic overhead.

In order to achieve totally minimized wiring overhead it would be necessary to design two different slice layouts, one of which is mirrored and flipped. This would allow each slice row in the PIN to share a power rail with the rows above and below it and also minimize the length of the row-to-row wiring. This design however greatly increases the complexity of the VHDL design as wiring the rows together becomes trickier. Additionally we would have to maintain two different versions of the PIN slices. We decided to instead go with a design where all slices are exactly identical and the interconnect between them is linear. This allows for easier calculation of PIN configuration values as every row has the same index order. The only real disadvantage to this design is that we will require long interconnects between slices. We are assuming for now that even with the added capacitance of these long interconnects we will still be able to achieve max clock speeds of greater than 50Mhz. By progress report 2 we will have layout simulation results to confirm this.

As stated earlier an important goal for us was to be able to directly chain multiple INTRANEX chips together. Our current design achieves this and an example chain showing 3 INTRANEXs chained together is shown below. Note that the pin layout in this diagram matches that of the actual layout we plan on implementing.

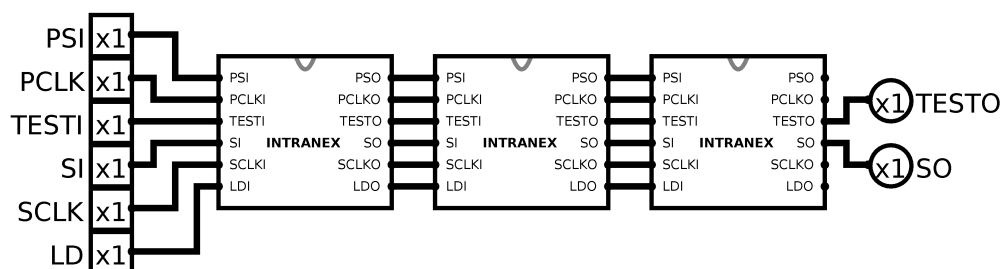


Figure 6: 3 INTRANEX Chain

4 Block Diagrams

4.1 Top Level

A top level block diagram for a 3-bit INTRANEX is shown below. The top module is the PIN and the bottom module is the parallel load shift register. Test mode logic has been excluded to more clearly illustrate the core functionality.

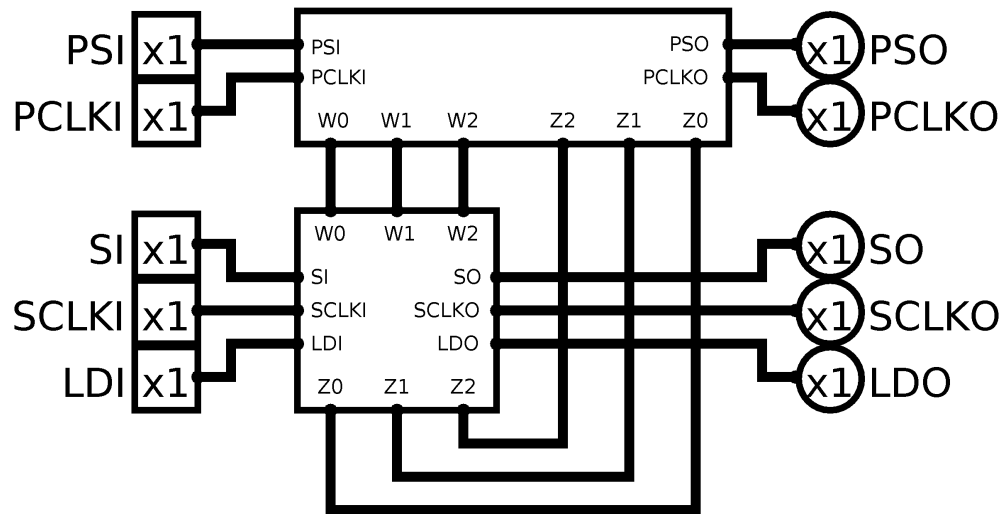


Figure 7: Top Level Block Diagram (3-Bit Configuration)

4.2 Top Level With Test Mode

The same top level diagram is shown with the addition of the test mode logic. The test mode logic simply consists of 3 2:1 multiplexers that redirect the output of the shift register to the input of the PIN and the output of the PIN to what is normally the output of the shift register. In other words, it wires in the PIN between the shifter and the shifters normal output pins.

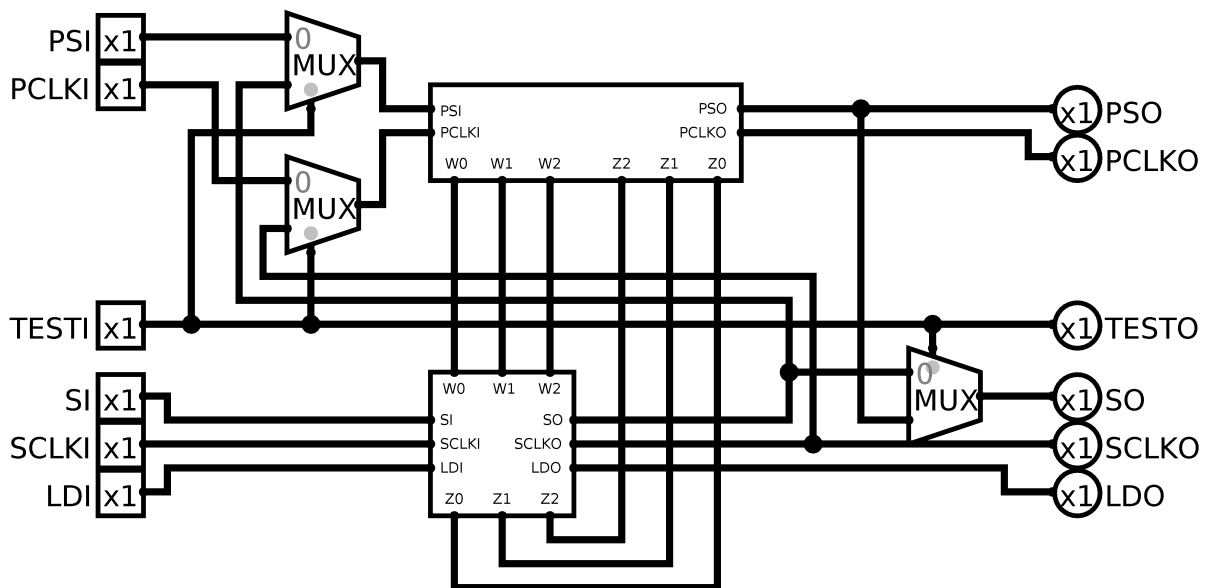


Figure 8: Top Level Block Diagram Showing Test Mode Logic (3-Bit Configuration)

4.3 Top Level Bit Sliced

The diagram below shows a bit sliced version of the top level diagram shown in Figure 7. We can see how each slice is directly connected together with zero interfacing logic as well as the long row-to-row connections as discussed earlier.

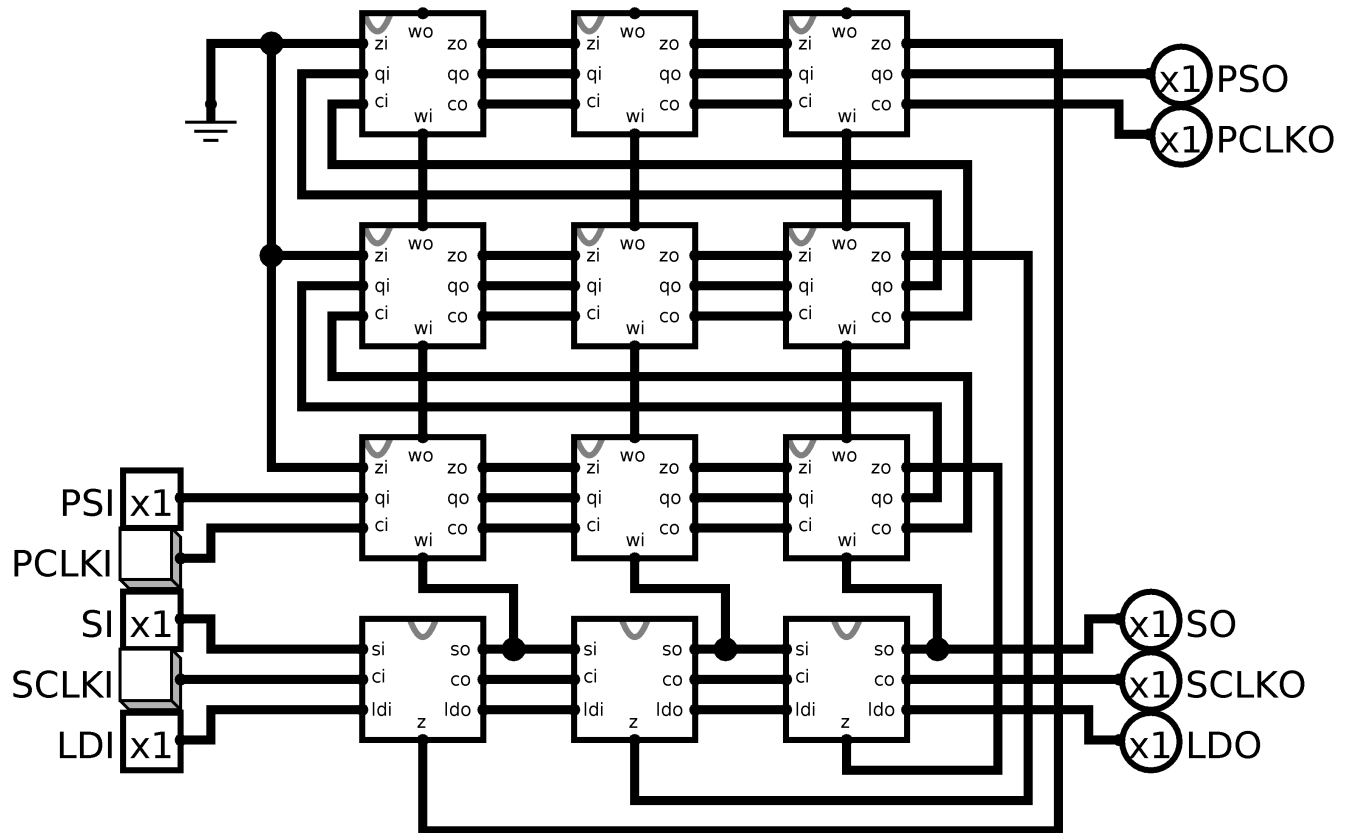


Figure 9: Top Level Bit Sliced Block Diagram (3-Bit Configuration)

4.4 Parallel Load Shift Register

4.4.1 Bit-slicing Scheme

Looking at just the shift register we can see that it is a parallel load parallel output shifter that is easily extendable by simply tacking on additional slices.

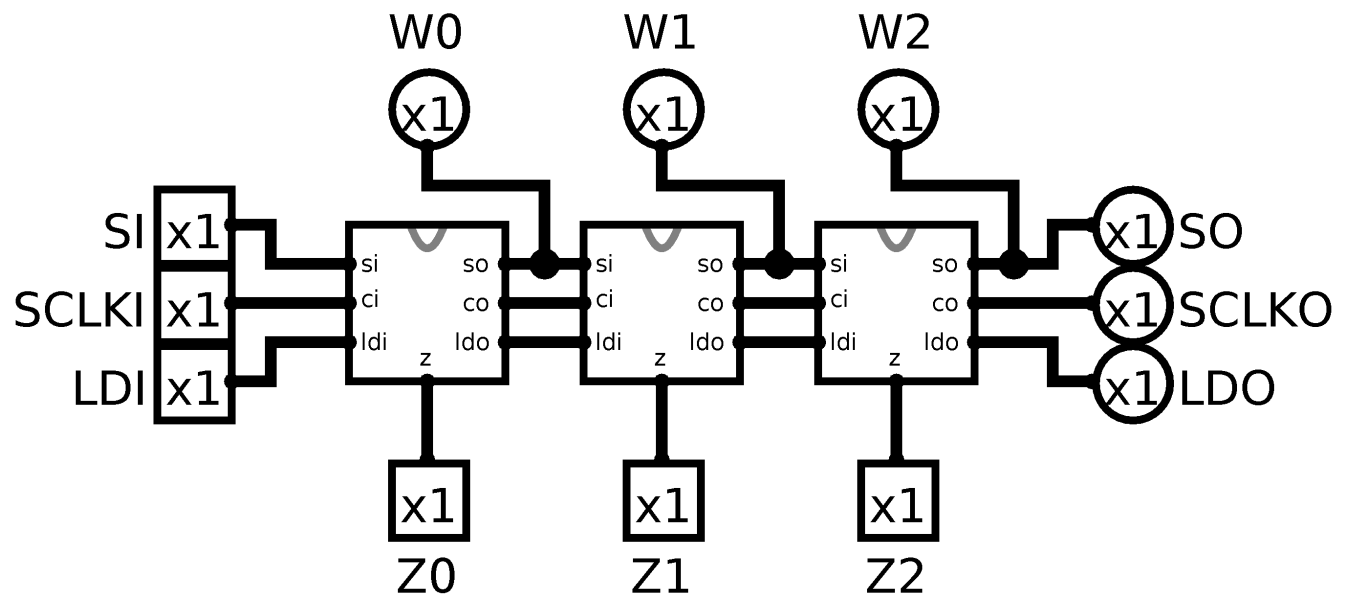


Figure 10: Parallel Load Bit-Sliced Shifter Register (3-Bit Configuration)

4.4.2 Bit-Slice

Looking at the internals of a single shift slice we can see that is is just a 2:1 multiplexer and a D Flip Flop. The multiplexer determines if the slice should load either the value from the previous slice (SI) or the parallel input (Z). When LDI is 0 it uses the value of the previous slice and when it is a 1 it uses the parallel load value.

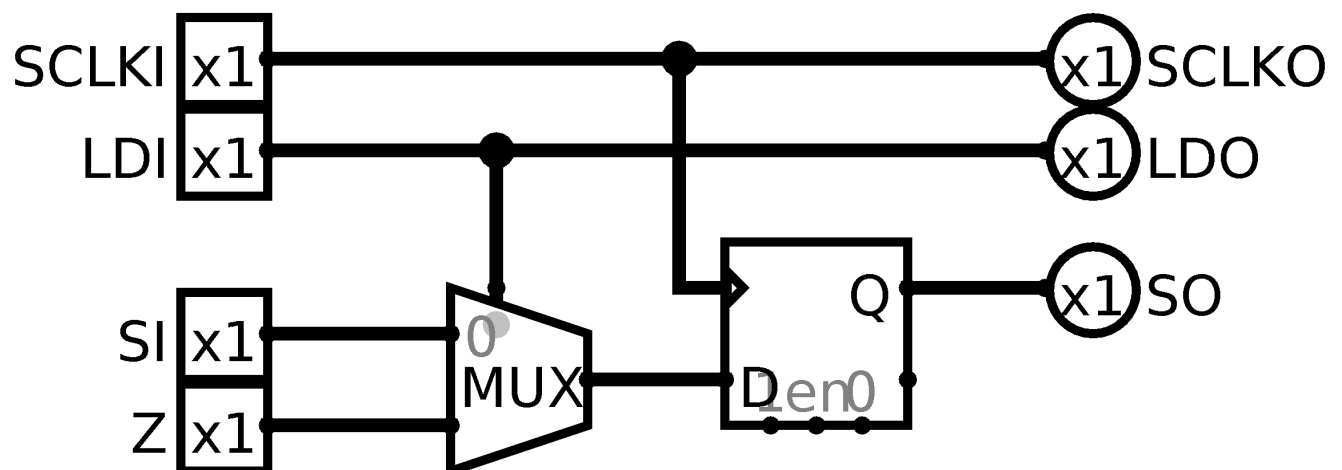


Figure 11: Parallel Load Shifter Register Bit-Slice

4.5 Programmable Interconnect Network

4.5.1 Bit-slicing Scheme

The diagram below shows just the PIN in bit-slice form. One of the design decisions made while determining the slice interconnects was to also pass the PCLK from slice to slice. The alternative was to simply connect each slices PCLKI to the main PCLKI pin at a higher level. We wanted to avoid as much manual layout as possible so it determined to be easier and cleaner to route the clock in such a way that it would be automatically connected when we layout the slice array.

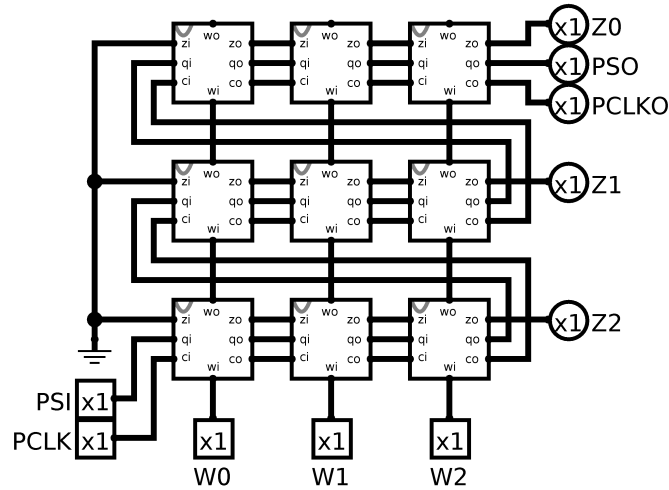


Figure 12: Bit-Sliced Programmable Interconnect Network (3-Bit Configuration)

4.5.2 Bit-Slice

The PIN bit-slices, one of which is shown below, is what drive the whole functionality of our chip. WI is the input values bit for the current column. If that bit is set and this slice is configured as ‘connected’ we want to output a logic high on the Z bus simultaneously. We cannot, however, just simply AND these two values together and attach it to the bus as this would allow for multiple slices to drive or sink the bus. To avoid this we use an OR gate to determine if the slice behind us is outputting a 1. If so we just pass it along. If we want to output a 1 it is also no problem as the OR will accommodate us as well.

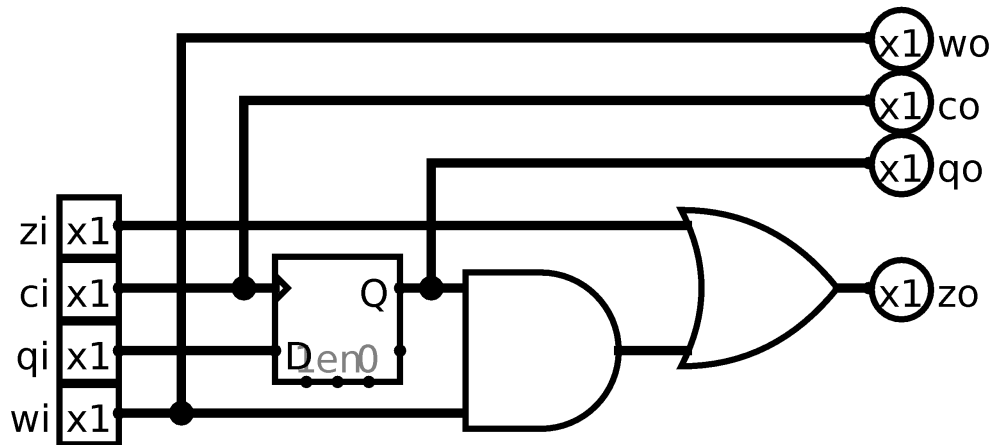


Figure 13: Programmable Interconnect Network Bit-Slice

5 VHDL Models

5.1 Top Level

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity top is
5      generic(
6          n : integer := 3
7      );
8      port(
9          psi : in  std_logic;
10         pso : out std_logic;
11         pclk : in  std_logic;
12         si : in  std_logic;
13         so : out std_logic;
14         sclk : in  std_logic;
15         ld : in  std_logic;
16         test : in  std_logic;
17     );
18 end top;
19
20 architecture rtl of top is
21
22     -- output of pin
23     signal z : std_logic_vector((n-1) downto 0) := (others => '0');
24     -- parallel output of shifter
25     signal w : std_logic_vector((n-1) downto 0) := (others => '0');
26
27     signal pin_clk : std_logic;
28     signal pin_psi : std_logic;
29     signal pin_pso : std_logic;
30     signal shift_out : std_logic;
31
32 begin
33
34     -- test mode mux connects shifter and pin together
35     test_mux_1 : entity work.mux2x1 port map(pclk, sclk, test, pin_clk);
36     test_mux_2 : entity work.mux2x1 port map(psi, shift_out, test, pin_psi);
37     test_mux_3 : entity work.mux2x1 port map(shift_out, pin_pso, test, so);
38
39     pin : entity work.pin
40     generic map(
41         n => n
42     )
43     port map(
44         clk => pin_clk,
45         psi => pin_psi,
46         pso => pin_pso,
47         z => z,
48         w => w
49     );
50
51     pso <= pin_pso;
52
53     shifter : entity work.shift
54     generic map(
55         n => n
56     )
57     port map(
58         clk => sclk,
59         si => si,
60         so => shift_out,
61         ld => ld,
62         z => z,
63         w => w
64     );
65
66 end rtl;

```

Listing 1: Top Level VHDL Module

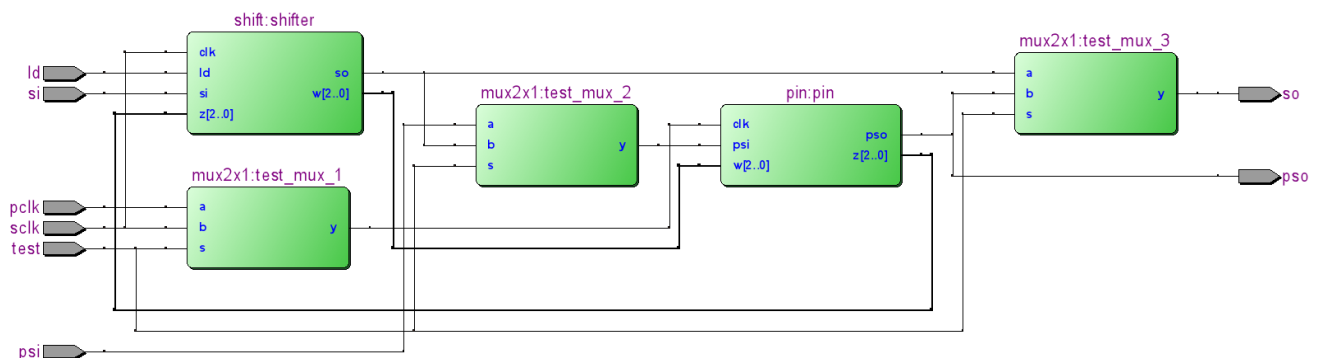


Figure 14: Top Level Generated RTL Diagram

5.2 PIN

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity pin is
5      generic(
6          n : integer := 3
7      );
8      port(
9          clk : in  std_logic;
10         psi : in  std_logic;
11         pso : out std_logic;
12         z : out std_logic_vector((n-1) downto 0);
13         w : in  std_logic_vector((n-1) downto 0)
14     );
15 end pin;
16
17 architecture rtl of pin is
18
19     component pin_slice is
20         port(
21             zi : in  std_logic;
22             qi : in  std_logic;
23             wi : in  std_logic;
24             ci : in  std_logic;
25             zo : out std_logic;
26             qo : out std_logic;
27             wo : out std_logic;
28             co : out std_logic
29         );
30     end component;
31
32     -- carry.array(row, col)
33     type carry_array is array (0 to n, 0 to n) of std_logic;
34     signal zc : carry_array;
35     signal cc : carry_array;
36     signal wc : carry_array;
37     signal qc : carry_array;
38
39     begin
40
41         -- setup first and last inputs for each row
42         z_connect : for i in 0 to n-1 generate
43             zc(i, 0) <= '0';
44             z(i) <= zc(i, n);
45         end generate;
46
47         -- setup first inputs for each column
48         w_connect : for i in 0 to n-1 generate
49             wc(0, i) <= w(i);
50         end generate;
51
52         -- setup row transfer
53         -- (last output of row to first input of next row)
54         r_connect : for i in 0 to n-2 generate
55             qc(i, 0) <= qc(i+1, n);
56             cc(i, 0) <= cc(i+1, n);
57         end generate;
58
59         -- connect external inputs
60         qc(n-1, 0) <= psi;
61         cc(n-1, 0) <= clk;
62         pso <= qc(0, n);
63
64         -- generate the grid of slices
65         pin_z_gen : for zz in 0 to n-1 generate
66             pin_w_gen : for ww in 0 to n-1 generate
67                 pin_i : pin_slice port map(
68                     zi => zc(zz, ww),
69                     qi => qc(zz, ww),
70                     wi => wc(zz, ww),
71                     ci => cc(zz, ww),
72                     zo => zc(zz, ww+1),
73                     qo => qc(zz, ww+1),
74                     wo => wc(zz+1, ww),
75                     co => cc(zz, ww+1)
76                 );
77             end generate;
78         end generate;
79
80     end rtl;

```

Listing 2: PIN VHDL Module

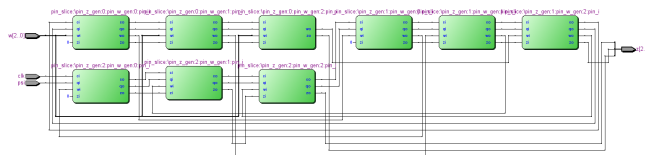


Figure 15: Pin Generated RTL Diagram

5.3 PIN Slice

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity pin_slice is
5      port(
6          zi : in std_logic;
7          qi : in std_logic;
8          wi : in std_logic;
9          ci : in std_logic;
10         zo : out std_logic;
11         qo : out std_logic;
12         wo : out std_logic;
13         co : out std_logic;
14     );
15 end pin_slice;
16
17 architecture rtl of pin_slice is
18
19     signal g1_o : std_logic := '0';
20     signal g2_o : std_logic := '0';
21
22 begin
23
24     g1 : entity work.dffposx1 port map(ci, qi, g1_o);
25     g2 : entity work.aoi21x1 port map(wi, g1_o, zi, g2_o);
26     g3 : entity work.invx1 port map(g2_o, zo);
27
28     -- pass through
29     co <= ci;
30     wo <= wi;
31     qo <= g1_o;
32
33 end rtl;

```

Listing 3: PIN Slice VHDL Module

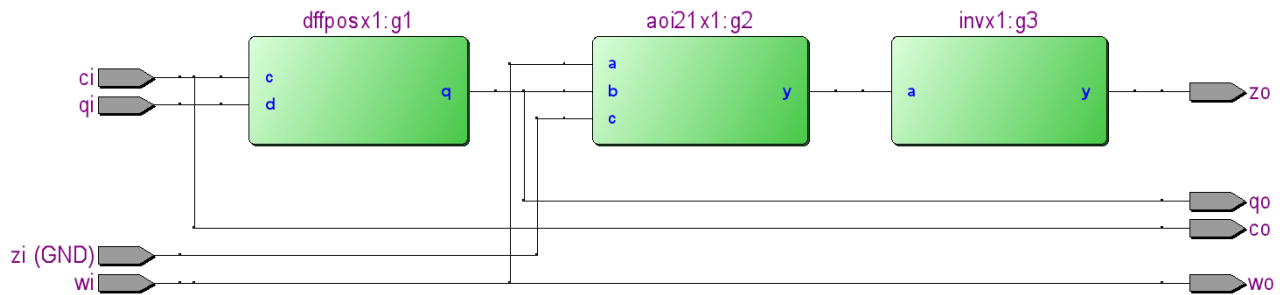


Figure 16: Pin Slice Generated RTL Diagram

5.4 Shifter

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity shift is
5      generic(
6          n : integer := 3
7      );
8      port(
9          clk : in  std_logic;
10         ld  : in  std_logic;
11         si  : in  std_logic;
12         so  : out std_logic;
13         z   : in  std_logic_vector((n-1) downto 0);
14         w   : out std_logic_vector((n-1) downto 0)
15     );
16 end shift;
17
18 architecture rtl of shift is
19
20     component shift_slice
21     port(
22         clki : in  std_logic;
23         clk0 : out std_logic;
24         ldi  : in  std_logic;
25         ldo  : out std_logic;
26         si   : in  std_logic;
27         so   : out std_logic;
28         z    : in  std_logic
29     );
30     end component;
31
32     -- vector to hold values between slices
33     signal c_so : std_logic_vector(n downto 0) := (others => '0');
34     signal c_clk : std_logic_vector(n downto 0) := (others => '0');
35     signal c_ld : std_logic_vector(n downto 0) := (others => '0');
36
37     begin
38         -- input of slice 0 comes from module input
39         c_so(0) <= si;
40         c_ld(0) <= ld;
41         c_clk(0) <= clk;
42
43         -- final shift output comes from output of last slice
44         so <= c_so(n);
45
46         -- generate N slices
47         shift_gen : for i in 0 to n-1 generate
48             shift_i : shift_slice port map(
49                 clki => c_clk(i),
50                 clk0 => c_clk(i+1),
51                 ldi  => c_ld(i),
52                 ldo  => c_ld(i+1),
53                 si   => c_so(i),
54                 so   => c_so(i+1),
55                 z    => z(i)
56             );
57         end generate;
58
59         -- connect the output of each slice to parallel output vector
60         connect : for i in 0 to n-1 generate
61             w(i) <= c_so(i+1);
62         end generate;
63     end rtl;
64
65 end rtl;

```

Listing 4: Parallel Load Shifter VHDL Module

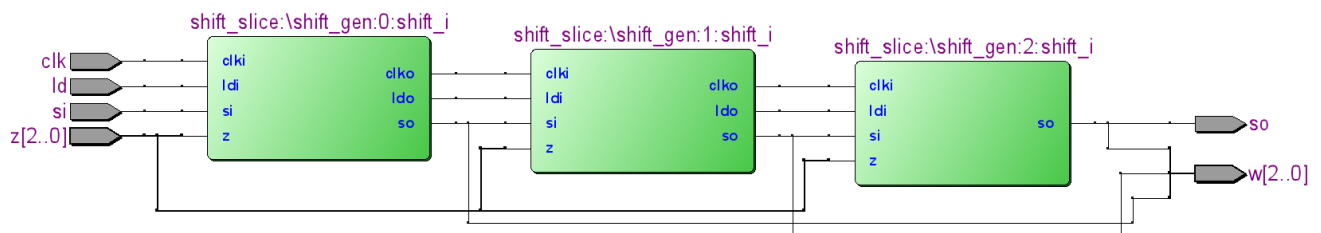


Figure 17: Shifter Generated RTL Diagram

5.5 Shifter Slice

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity shift_slice is
5      port(
6          clki : in  std_logic;
7          clko : out std_logic;
8          ldi  : in  std_logic;
9          ldo  : out std_logic;
10         si   : in  std_logic;
11         so   : out std_logic;
12         z    : in  std_logic
13     );
14 end shift_slice;
15
16 architecture rtl of shift_slice is
17     signal g1_o : std_logic := '0';
18
19 begin
20
21     g1 : entity work.mux2x1 port map(si, z, ldi, g1_o);
22     g2 : entity work.dffposx1 port map(clki, g1_o, so);
23
24     -- pass through
25     clko <= clki;
26     ldo <= ldi;
27
28 end rtl;

```

Listing 5: Parallel Load Shifter Slice VHDL Module

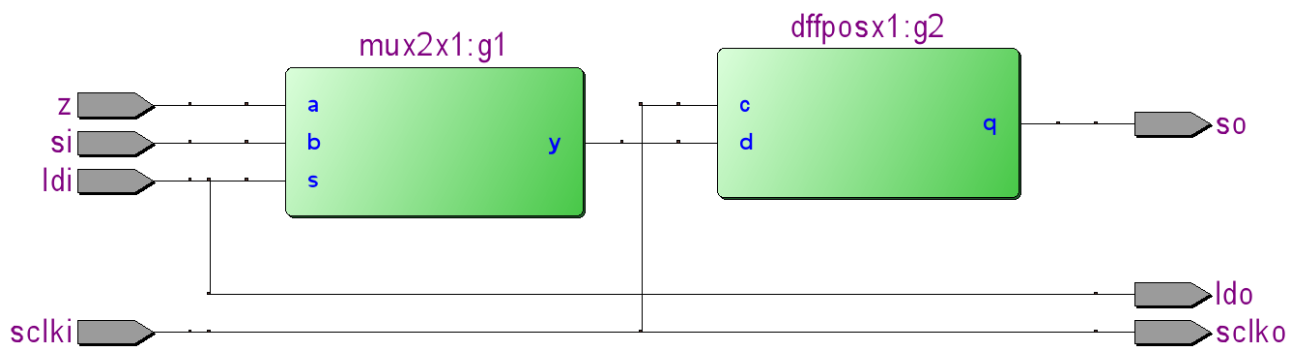


Figure 18: Shifter Slice Generated RTL Diagram

5.6 Gates

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity aoi21x1 is
5      generic(delay : time := 0 ps);
6      port(
7          a : in std_logic;
8          b : in std_logic;
9          c : in std_logic;
10         y : out std_logic
11     );
12 end aoi21x1;
13
14 architecture rtl of aoi21x1 is begin
15     process(a, b, c) begin
16         y <= not ((a and b) or c) after delay;
17     end process;
18 end rtl;

```

Listing 6: AOI21X1 VHDL Module

```

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

```

Listing 7: DFFPOSX1 VHDL Module

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity invx1 is
5      generic(delay : time := 0 ps);
6      port(
7          a : in std_logic;
8          y : out std_logic
9     );
10 end invx1;
11
12 architecture rtl of invx1 is begin
13     y <= not a after delay;
14 end rtl;

```

Listing 8: INVX1 VHDL Module

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux2x1 is
5      generic(delay : time := 0 ps);
6      port(
7          a : in std_logic;
8          b : in std_logic;
9          s : in std_logic;
10         y : out std_logic
11     );
12 end mux2x1;
13
14 architecture rtl of mux2x1 is begin
15     process(a, b, s) begin
16         if (s = '1') then
17             y <= b after delay;
18         else
19             y <= a after delay;
20         end if;
21     end process;
22 end rtl;

```

Listing 9: MUX2X1 VHDL Module

6 VHDL Test Benches

6.1 Top Level Functional

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use std.textio.all;
4  use work.txt_util.all;
5
6  entity top_tb is
7      generic(
8          stim_file : string := "vectors.3.bit.sim"
9      );
10 end top_tb;
11
12 architecture tb_rtl of top_tb is
13
14     constant n : integer := 3;
15
16     signal psi : std_logic := '0';
17     signal pso : std_logic;
18     signal pclk : std_logic := '0';
19     signal si : std_logic := '0';
20     signal so : std_logic;
21     signal sclk : std_logic := '0';
22     signal ld : std_logic := '0';
23     signal test : std_logic := '0';
24
25     component top
26     generic(
27         n : integer := 3
28     );
29     port(
30         psi : in std_logic;
31         pso : out std_logic;
32         pclk : in std_logic;
33         si : in std_logic;
34         so : out std_logic;
35         sclk : in std_logic;
36         ld : in std_logic;
37         test : in std_logic
38     );
39 end component;
40
41 signal pin_vector : std_logic_vector((n*n)-1 downto 0);
42 signal shift_vector : std_logic_vector(n-1 downto 0);
43 signal result_vector : std_logic_vector(n-1 downto 0);
44
45 file stimulus : TEXT open read_mode is stim_file;
46
47 begin
48
49     uut : top
50     generic map(
51         n => n
52     )
53     port map(
54         psi => psi,
55         pso => pso,
56         pclk => pclk,
57         si => si,
58         so => so,
59         sclk => sclk,
60         ld => ld,
61         test => test
62     );
63
64     process
65
66         procedure clock_shifter is begin
67             sclk <= '1';
68             wait for 20 ns;
69             sclk <= '0';
70             wait for 20 ns;
71         end procedure clock_shifter;
72
73         procedure clock_pin is begin
74             pclk <= '1';
75             wait for 20 ns;
76             pclk <= '0';
77             wait for 20 ns;
78         end procedure clock_pin;
79
80         variable l : line;
81         variable pin_str : string(1 to n*n);
82         variable shf_str : string(1 to n);
83
84     begin
85
86         while not endfile(stimulus) loop
87
88             -- load stimulus for this test
89             readline(stimulus, l); read(l, pin_str);
90             pin_vector <= to_std_logic_vector(pin_str);
91
92             readline(stimulus, l); read(l, shf_str);
93             shift_vector <= to_std_logic_vector(shf_str);
94
95             readline(stimulus, l); read(l, shf_str);
96             result_vector <= to_std_logic_vector(shf_str);
97
98             wait for 100 ns;
99

```

```

100      -- clock in the pin
101      for i in 0 to (n*n)-1 loop
102          psi <= pin_vector(i);
103          wait for 20 ns;
104          clock_pin;
105      end loop;
106
107      -- clock in the value
108      for i in 0 to n-1 loop
109          si <= shift_vector(i);
110          wait for 20 ns;
111          clock_shifter;
112      end loop;
113
114      -- pull latch high so the first result
115      -- loop will trigger the latch
116      ld <= '1';
117      wait for 20 ns;
118
119      -- clock out result and check it
120      for i in 0 to n-1 loop
121          clock_shifter;
122          assert so = result_vector(i) report "Test Failed!";
123          ld <= '0';
124          wait for 20 ns;
125      end loop;
126
127      end loop;
128
129      report "Test Complete" severity note;
130      wait;
131
132      end process;
133
134      end tb_rtl;

```

Listing 10: Top Level VHDL Test Bench

We decided to write a small Python script to generate the expected output vector for all possible PIN configurations and input values. Our test bench then runs through all of these vectors and checks if the output vector from our VHDL design matches the known output.

```

1  N = 3
2
3  out = open("vectors.%d.bit.sim"%N, "w")
4
5  # loop through all possible pins and shift values
6  for pin in range(2**(N*N)):
7      for shift in range(2*N):
8
9          # get bit strings
10         pin_bits = bin(pin).replace("0b", "").zfill(N*N)
11         shift_bits = bin(shift).replace("0b", "").zfill(N)
12         result_bits = ["0"]*N
13
14         # calculate the expected result
15         for z in range(N):
16             for w in range(N):
17                 if pin_bits[(N-1-z)*N+w] == "1" and shift_bits[w] == "1":
18                     result_bits[z] = "1"
19
20         # writeout results
21         out.write("%s\n" % pin_bits)
22         out.write("%s\n" % shift_bits)
23         out.write("%s\n" % "".join(result_bits))
24
25  out.close()

```

Listing 11: Python Vector Generator

6.2 Top Level Test Mode

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity top_test_tb is
5  end top_test_tb;
6
7  architecture tb_rtl of top_test_tb is
8
9      constant n : integer := 3;
10
11     signal psi : std_logic := '0';
12     signal pso : std_logic;
13     signal pclk : std_logic := '0';
14     signal si : std_logic := '0';
15     signal so : std_logic;
16     signal sclk : std_logic := '0';
17     signal ld : std_logic := '0';
18     signal test : std_logic := '0';
19
20     component top
21     generic(
22         n : integer := n
23     );
24     port(
25         psi : in std_logic;
26         pso : out std_logic;
27         pclk : in std_logic;
28         si : in std_logic;
29         so : out std_logic;
30         sclk : in std_logic;
31         ld : in std_logic;
32         test : in std_logic
33     );
34     end component;
35
36 begin
37
38     uut : top
39     generic map(
40         n => n
41     )
42     port map(
43         psi => psi,
44         pso => pso,
45         pclk => pclk,
46         si => si,
47         so => so,
48         sclk => sclk,
49         ld => ld,
50         test => test
51     );
52
53     process
54     procedure clock is begin
55         sclk <= '1';
56         wait for 20 ns;
57         sclk <= '0';
58         wait for 20 ns;
59     end procedure clock;
60     begin
61
62         wait for 20 ns;
63
64         -- pull test line high to enable test mode
65         test <= '1';
66
67         -- clock in a '1'
68         si <= '1';
69         wait for 20 ns;
70         clock;
71
72         -- clock in a '0'
73         si <= '0';
74         wait for 20 ns;
75         clock;
76
77         -- push the pulse through till just before the last FF
78         -- (n*n)+n == number of flip flops
79         -- 2 == we already did two clocks
80         -- 1 == we want to stop before before final output
81         for i in 1 to (n*n)+n-2-1 loop
82             clock;
83         end loop;
84
85         -- check to make sure the bit in front of the pulse is 0
86         assert so = '0' report "Bit leading pulse not 0";
87         clock;
88         -- check to make sure the pulse is 1
89         assert so = '1' report "Pulse is not 1";
90         clock;
91         -- check to make sure the bit behind pulse is 0
92         assert so = '0' report "Bit trailing pulse not 0";
93         clock;
94
95         report "Test Complete" severity note;
96         wait;
97     end process;
98
99 end tb_rtl;

```

Listing 12: Top Level Test Mode VHDL Test Bench

6.3 PIN Slice

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity pin_slice_tb is
5  end pin_slice_tb;
6
7  architecture tb_rtl of pin_slice_tb is
8
9      signal zi : std_logic := '0';
10     signal qi : std_logic := '0';
11     signal wi : std_logic := '0';
12     signal ci : std_logic := '0';
13     signal zo : std_logic;
14     signal qo : std_logic;
15     signal wo : std_logic;
16     signal co : std_logic;
17
18     component pin_slice
19     port(
20         zi : in std_logic;
21         qi : in std_logic;
22         wi : in std_logic;
23         ci : in std_logic;
24         zo : out std_logic;
25         qo : out std_logic;
26         wo : out std_logic;
27         co : out std_logic
28     );
29     end component;
30
31 begin
32
33     uut : pin_slice
34     port map(
35         zi => zi,
36         qi => qi,
37         wi => wi,
38         ci => ci,
39         zo => zo,
40         qo => qo,
41         wo => wo,
42         co => co
43     );
44
45     process
46         type pattern_type is record
47             -- inputs
48             zi, qi, wi : std_logic;
49             -- output
50             zo : std_logic;
51         end record;
52
53         type pattern_array is array (natural range <>) of pattern_type;
54         constant patterns : pattern_array :=
55             -- zi qi wi zo
56             (( '0', '0', '0', '0' ),
57              ( '0', '0', '1', '0' ),
58              ( '0', '1', '0', '0' ),
59              ( '0', '1', '1', '1' ),
60              ( '1', '0', '0', '1' ),
61              ( '1', '0', '1', '1' ),
62              ( '1', '1', '0', '1' ),
63              ( '1', '1', '1', '1' ));
64
65     begin
66         -- check each pattern
67         for i in patterns'range loop
68
69             -- set the inputs
70             zi <= patterns(i).zi;
71             qi <= patterns(i).qi;
72             wi <= patterns(i).wi;
73             wait for 10 ns;
74
75             -- pulse the clock and check clock passthrough
76             ci <= '1';
77             wait for 10 ns;
78             assert co = '1' report "CO does not equal 1" severity error;
79             ci <= '0';
80             wait for 10 ns;
81             assert co = '0' report "CO does not equal 0" severity error;
82
83             -- check the outputs
84             assert qo = patterns(i).qi report "QI not equal QO" severity error;
85             assert zo = patterns(i).zo report "ZO does not match pattern" severity error;
86
87         end loop;
88
89         report "Test Complete" severity note;
90         wait;
91
92     end process;
93
94 end tb_rtl;

```

Listing 13: PIN Slice VHDL Test Bench

6.4 Shifter Slice

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity shift_slice_tb is
5  end shift_slice_tb;
6
7  architecture tb_rtl of shift_slice_tb is
8
9      signal sclki : std_logic := '0';
10     signal sclko : std_logic;
11     signal ldi : std_logic := '0';
12     signal ldo : std_logic;
13     signal si : std_logic := '0';
14     signal so : std_logic;
15     signal z : std_logic := '0';
16
17     component shift_slice is
18     port(
19         sclki : in std_logic;
20         sclko : out std_logic;
21         ldi : in std_logic;
22         ldo : out std_logic;
23         si : in std_logic;
24         so : out std_logic;
25         z : in std_logic
26     );
27 end component;
28
29 begin
30
31     uut : shift_slice
32     port map(
33         sclki => sclki,
34         sclko => sclko,
35         ldi => ldi,
36         ldo => ldo,
37         si => si,
38         so => so,
39         z => z
40     );
41
42     process
43         type pattern_type is record
44             -- inputs
45             ldi, z, si : std_logic;
46             -- output
47             so : std_logic;
48         end record;
49
50         type pattern_array is array (natural range <>) of pattern_type;
51         constant patterns : pattern_array :=
52             -- ldi, z, si, so
53             ((('0','0','0','0'), ('0')),
54              (('0','0','1','1'), ('1')),
55              (('0','1','0','0'), ('0')),
56              (('0','1','1','1'), ('1')),
57              (('1','0','0','0'), ('0')),
58              (('1','0','1','0'), ('0')),
59              (('1','1','0','1'), ('1')),
60              (('1','1','1','1'), ('1')));
61
62     begin
63         -- check each pattern
64         for i in patterns'range loop
65
66             -- set the inputs
67             ldi <= patterns(i).ldi;
68             z <= patterns(i).z;
69             si <= patterns(i).si;
70             wait for 10 ns;
71
72             -- pulse the clock and check clock passthrough
73             sclki <= '1';
74             wait for 10 ns;
75             assert sclko = '1' report "SCLKO does not equal 1" severity error;
76             assert ldo = patterns(i).ldi report "SCLKO does not equal 1" severity error;
77             sclki <= '0';
78             wait for 10 ns;
79             assert sclko = '0' report "SCLKO does not equal 0" severity error;
80             assert ldo = patterns(i).ldi report "SCLKO does not equal 1" severity error;
81
82             -- check the output
83             assert so = patterns(i).so report "SO is incorrect" severity error;
84
85         end loop;
86
87         report "Test Complete" severity note;
88         wait;
89
90     end process;
91
92 end tb_rtl;

```

Listing 14: Parallel Load Shifter Slice VHDL Test Bench

7 VHDL Test Bench Results

7.1 Top Level Functional

While our top level functional testbench is completely automated and does an exhaustive test on all possible inputs an example waveform is shown below. We can first see that we clock in a PIN configuration vector of 000001000 which enables slice Z1W2. We then shift in a value of 001. Given these input vectors we expect the output vector to be 010. We can see from the waveform below that we achieve the expected result.

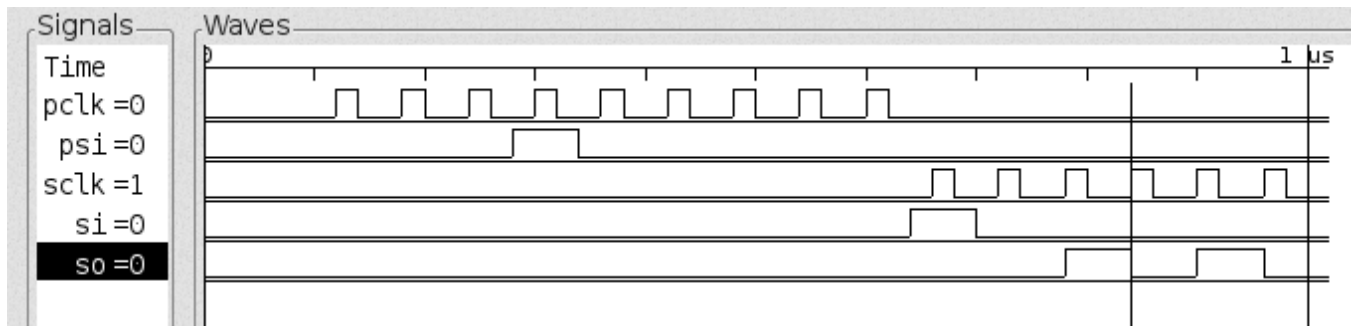


Figure 19: Top Level Functional Test Bench Waveform

7.2 Top Level Test Mode

Our top level test mode testbench is also completely automated. For this test we simply send a pulse through all the flip flops and count the number of clock cycles it takes for the pulse to come out the other end. For a 3-Bit configuration there are $3 \times 3 + 3$ flip flops so we expect the pulse to appear at the output after 12 clock pulses. We can see from the waveform below that we achieve the expected output.

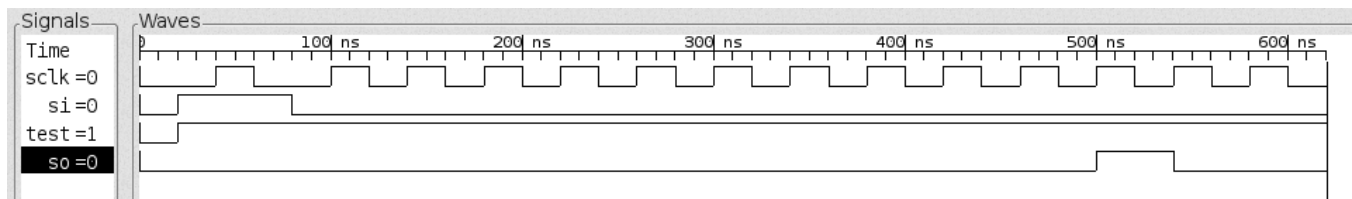


Figure 20: Top Level Test Mode Test Bench Waveform

8 Work Division

Task	Person
Pinout Diagram	Both
Explanation of Functionality	Both
Design Decisions	Both
Top Level Block Diagrams	Both
Shifter Block Diagrams	Qi
PIN Block Diagrams	Thrun
VHDL Shifter+TB	Qi
VHDL PIN+TB	Qi
VHDL Top+TB	Thrun
VHDL Top Test Mode TB	Thrun

Table 2: Task Assignment