FLORIDA INTERNATIONAL UNIVERSITY

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EEL-5725 Digital Systems Engineering I
And Special Topics Assignment

VHDL implementation of several devices

Submitted to: Dr. Subbarao Wunnava

Submitted by: Pablo Gomez (FEEDS Ph.D student)

Original: Fall 2004/2005 Revised Summer 2006

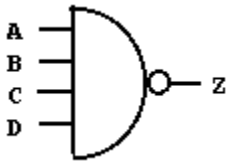
INTRODUCTION

The purpose of this assignment is to implement several combinational and register based devices using VHDL code (behavioral, data flow and Boolean) and compare the performance when using two different clock rates: 100KHz and 10MHz.

The following devices were designed, implemented and tested using Warp and ModelSim software and the Cypress CY374i CPLD:

- 4 and 8 input NAND gates
- 4 and 8 input NOR gates
- 4 and 8 input multiplexers
- 3/8 decoder
- 4-bit up/down binary counters
- 4 and 8 bit left/right serial shift registers

4-INPUT NAND GATE



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity NAND4 is port(
A,B,C,D : in std_logic;
Z : out std_logic);
end NAND4;
 	
architecture archNAND4 of NAND4 is
begin
process (A,B,C,D) begin
if (A = '1' and B = '1' and C = '1' and D = '1') then
Z <= '0';
else
Z <= '1';
end if;
end process;
end archNAND4;
	library ieee;
	use ieee.std_logic_1164.all;
	entity NAND4 is port(
	A,B,C,D : in std_logic;
	Z : out std_logic);
	end NAND4;
	
	architecture archNAND4 of NAND4 is
	begin
	Z <= '0' when (A = '1' and B = '1' and C = '1' and D = '1')
	else '1';
	end archNAND4;
		library ieee;
		use ieee.std_logic_1164.all;
		entity NAND4 is port(
		A,B,C,D : in std_logic;
		Z : out std_logic);
		end NAND4;
		
		architecture archNAND4 of NAND4 is
		begin
		Z <= not (A and B and C and D);
		end archNAND4;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
/z = a * b * c * d	/z = a * b * c * d	/z = a * b * c * d

The generated equations in all cases are the same. The following figures show the verification and timing analysis at 100KHz and 10MHz. The device performs correctly at both frequency rates.

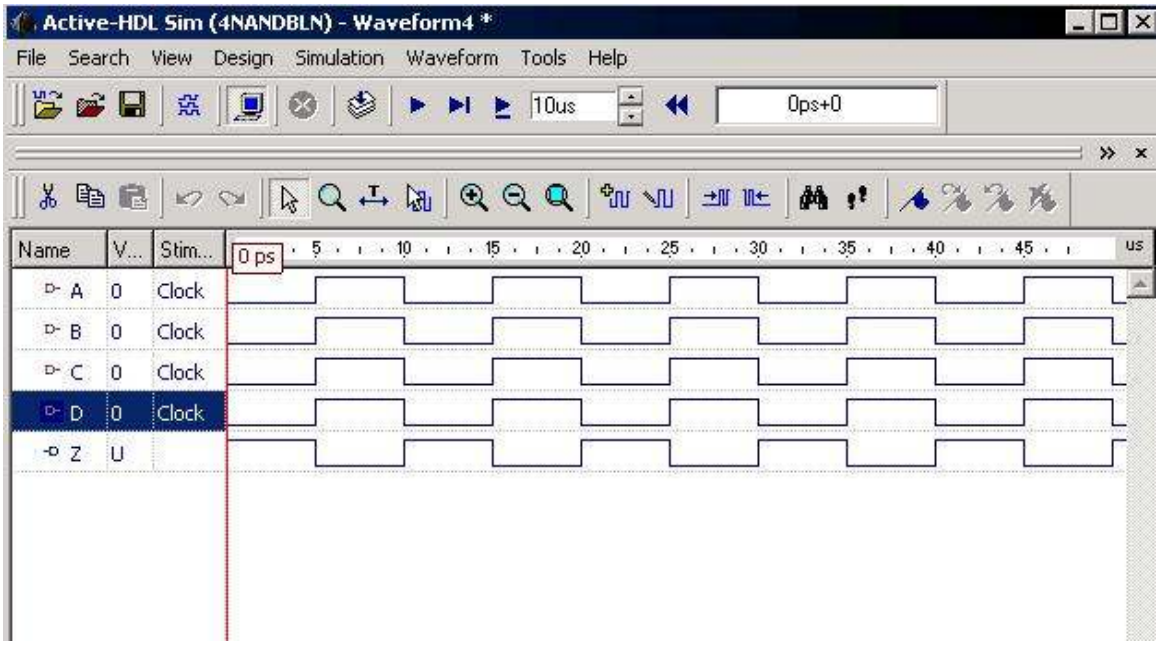


Figure 1. 4-Input NAND at 100KHz

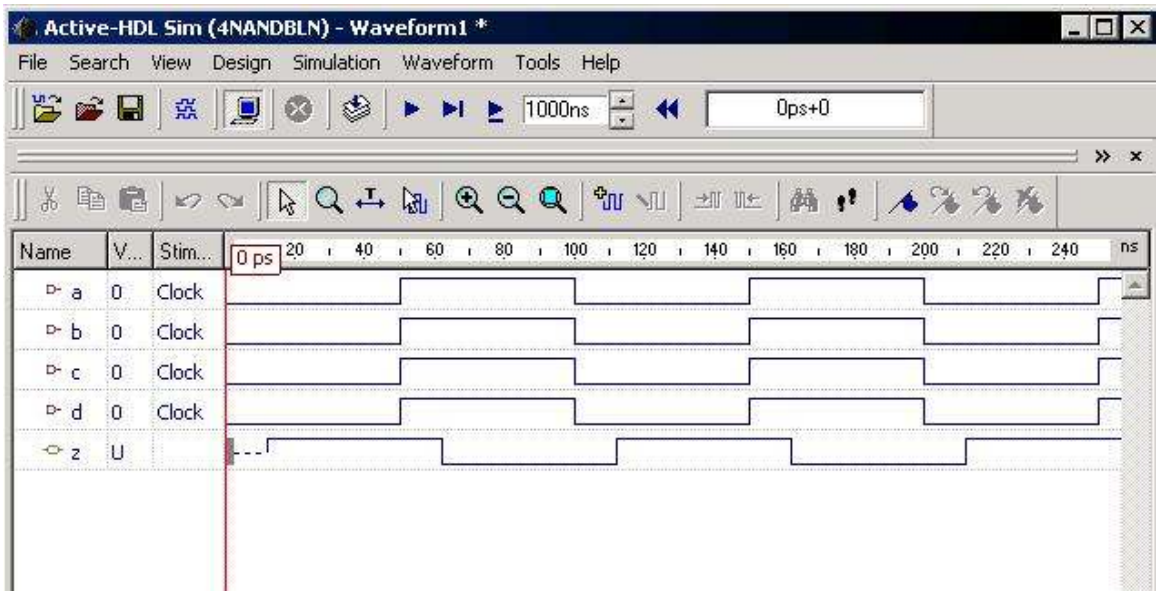
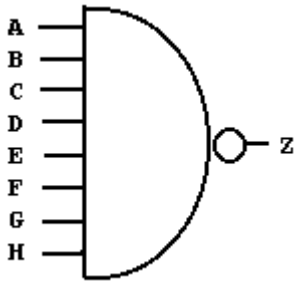


Figure 2. 4-Input NAND at 10MHz

8-INPUT NAND GATE



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity NAND8 is port(
A,B,C,D,E,F,G,H : in std_logic;
Z : out std_logic);
end NAND8;
 	
architecture archNAND8 of NAND8 is
begin
process(A,B,C,D,E,F,G,H)
begin
if (A='1' and B='1' and
C='1' and D='1' and
E='1' and F='1' and
G='1' and H='1') then
Z <= '0';
else
Z <= '1';
end if;
end process;
end archNAND8;
	library ieee;
	use ieee.std_logic_1164.all;
	entity NAND8 is port(
	A : in std_logic_vector(0 to 7);
	Z : out std_logic);
	end NAND8;
	
	architecture archNAND8 of NAND8 is
	begin
	with A select
	Z <= '0' when "11111111",
	'1' when others;
	end archNAND8;
		library ieee;
		use ieee.std_logic_1164.all;
		entity NAND8 is port(
		A : in std_logic_vector(0 to 7);
		Z : out std_logic);
		end NAND8;
		
		architecture archNAND8 of NAND8 is
		begin
		Z <= not (A(0) and A(1) and
		A(2) and A(3) and
		A(4) and A(5) and
		A(6) and A(7));
		end archNAND8;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
/z = a * b * c * d * e * f * g * h	/z = a_0 * a_1 * a_2 * a_3 * a_4 * a_5 * a_6 * a_7	/z = a_0 * a_1 * a_2 * a_3 * a_4 * a_5 * a_6 * a_7

The generated equations in all cases are the same. The following figures show the verification and timing analysis at 100KHz and 10MHz. The device performs correctly at both frequency rates.

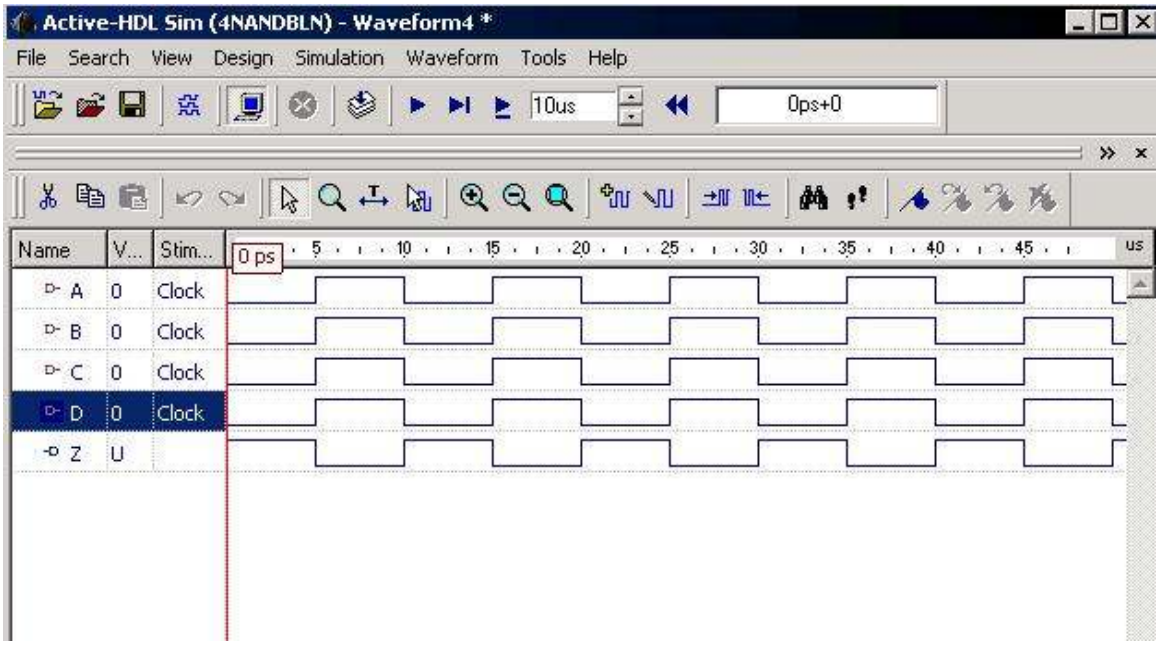


Figure 3. 8-Input NAND at 100KHz

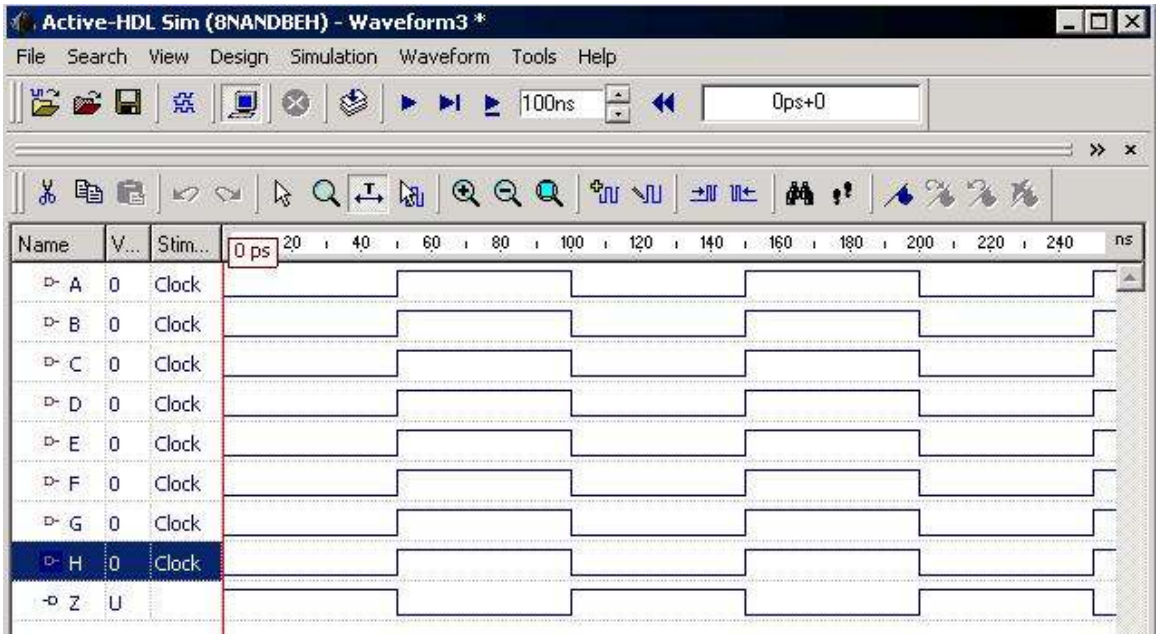
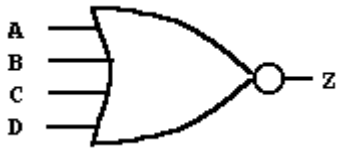


Figure 4. 8-Input NAND at 10MHz

4-INPUT NOR GATE



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity NOR4 is port(
A,B,C,D : in std_logic;
Z : out std_logic);
end NOR4;
 	
architecture archNOR4 of NOR4 is
begin
process (A,B,C,D) begin
if (A = '1' or B = '1' or
C = '1' or D = '1') then
Z <= '0';
else
Z <= '1';
end if;
end process;
end archNOR4;
	library ieee;
	use ieee.std_logic_1164.all;
	entity NOR4 is port(
	A,B,C,D : in std_logic;
	Z : out std_logic);
	end NOR4;
	
	architecture archNOR4 of NOR4 is
	begin
	Z <= '0' when (A = '1' or B = '1'
	or C = '1' or D = '1')
	else '1';
	end archNOR4;
		library ieee;
		use ieee.std_logic_1164.all;
		entity NOR4 is port(
		A,B,C,D : in std_logic;
		Z : out std_logic);
		end NOR4;
		
		architecture archNOR4 of NOR4 is
		begin
		Z <= not (A or B or C or D);
		end archNOR4;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
z = /a * /b * /c * /d	z = /a * /b * /c * /d	z = /a * /b * /c * /d

The generated equations in all cases are the same. The following figures show the verification and timing analysis at 100KHz

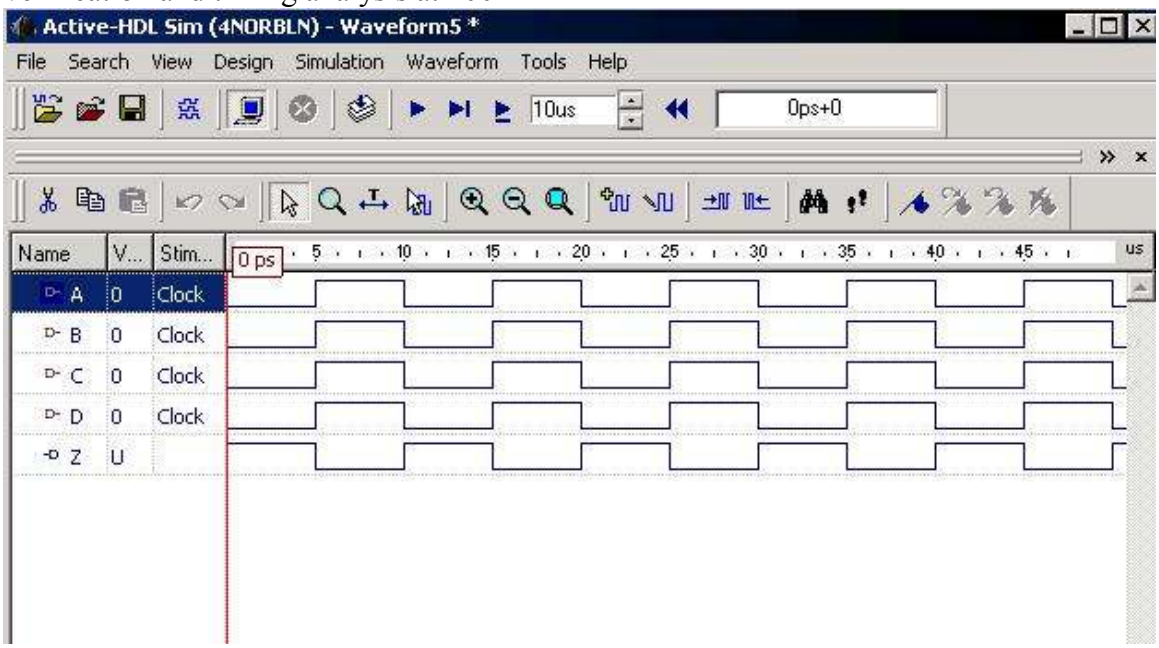
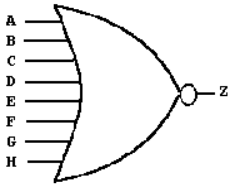


Figure 5. 4-Input NOR at 100KHz

8-INPUT NOR GATE



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity NOR8 is port(
A,B,C,D,E,F,G,H : in std_logic;
Z : out std_logic);
end NOR8;
architecture archNOR8 of NOR8 is
begin
process(A,B,C,D,E,F,G,H)
begin
if (A='1' or B='1' or C='1' or D='1' or E='1' or F='1' or G='1' or H='1') then
Z <= '0';
else
Z <= '1';
end if;
end process;
end archNOR8;
	library ieee;
	use ieee.std_logic_1164.all;
	entity NOR8 is port(
	A : in std_logic_vector(0 to 7);
	Z : out std_logic);
	end NOR8;
	architecture archNOR8 of NOR8 is
	begin
	with A select
	Z <= '1' when "00000000",
	'0' when others;
	end archNOR8;
		library ieee;
		use ieee.std_logic_1164.all;
		entity NOR8 is port(
		A : in std_logic_vector(0 to 7);
		Z : out std_logic);
		end NOR8;
		architecture archNOR8 of NOR8 is
		begin
		Z <= not (A(0) or A(1) or A(2) or A(3) or A(4) or A(5) or A(6) or A(7));
		end archNOR8;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
z = /a * /b * /c * /d * /e * /f * /g * /h	z = /a_0 * /a_1 * /a_2 * /a_3 * /a_4 * /a_5 * /a_6 * /a_7	z = /a_0 * /a_1 * /a_2 * /a_3 * /a_4 * /a_5 * /a_6 * /a_7

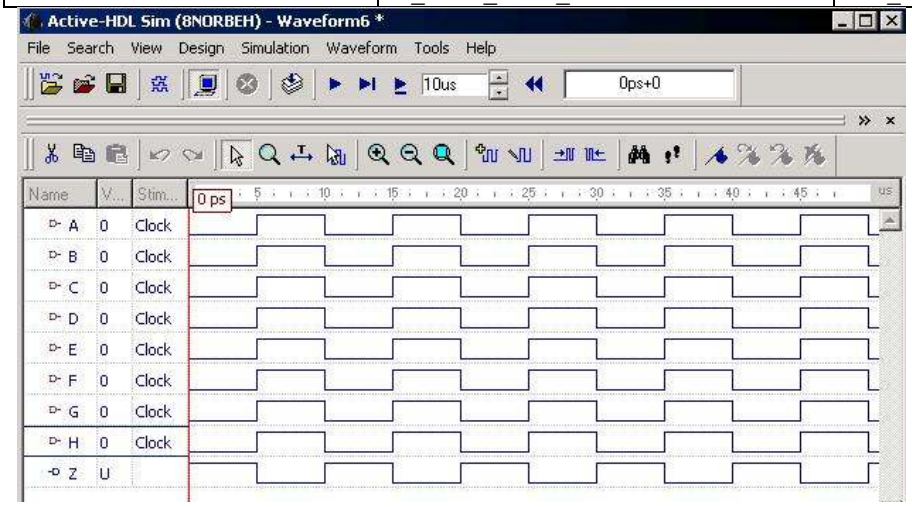
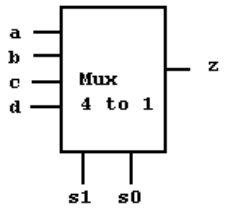


Figure 6. 8-Input NOR at 100KHz

.

4 to 1 MULTIPLEXER



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity mux41 is
port (a, b, c, d : in std_logic;
s : in std_logic_vector (1 downto 0);
z : out std_logic);
end mux41;
architecture archi of mux41 is
begin
process (a, b, c, d, s)
begin
if (s = "00") then z <= a;
elsif (s = "01") then z <= b;
elsif (s = "10") then z <= c;
else z <= d;
end if;
end process;
end archi;
	library ieee;
	use ieee.std_logic_1164.all;
	entity mux41 is
	port (a, b, c, d : in std_logic;
	s : in std_logic_vector (1 downto 0);
	z : out std_logic);
	end mux41;
	
	architecture archi of mux41 is
	begin
	with s select
	z <= a when "00",
	b when "01",
	c when "10",
	d when others;
	end archi;
		library ieee;
		use ieee.std_logic_1164.all;
		entity mux41 is
		port (a,b,c,d : in std_logic;
		s : in std_logic_vector (1 downto 0);
		z : out std_logic);
		end mux41;
		
		architecture archi of mux41 is
		begin
		z <= (a and (not s(0) and not s(1))) or
		(b and (not s(0) and s(1))) or
		(c and (s(0) and not s(1))) or
		(d and (s(0) and s(1)));
		end archi;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
z = a * /s_1 * /s_0 + c * s_1 * /s_0 + b * /s_1 * s_0 + d * s_1 * s_	z = a * /s_1 * /s_0 + c * s_1 * /s_0 + b * /s_1 * s_0 + d * s_1 * s_0	z = a * /s_1 * /s_0 + b * s_1 * /s_0 + c * /s_1 * s_0 + d * s_1 * s_0

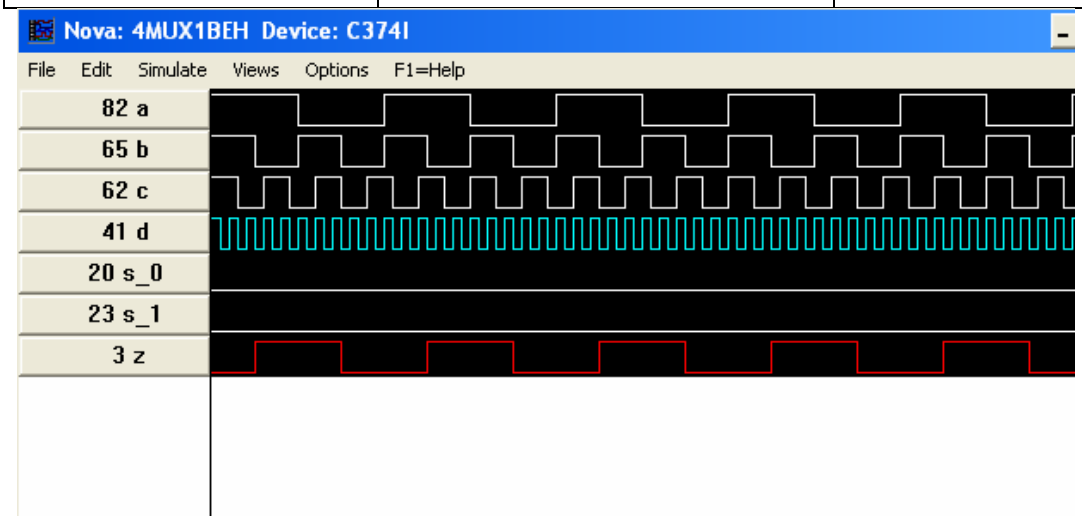


Figure 7. Mux 4/1 Functionality (A selected)

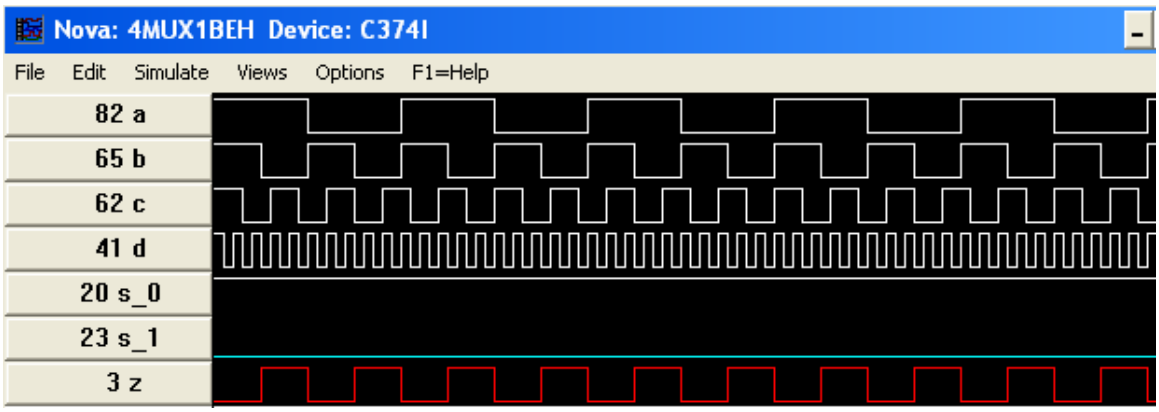


Figure 8. Mux 4/1 Functionality (B selected)

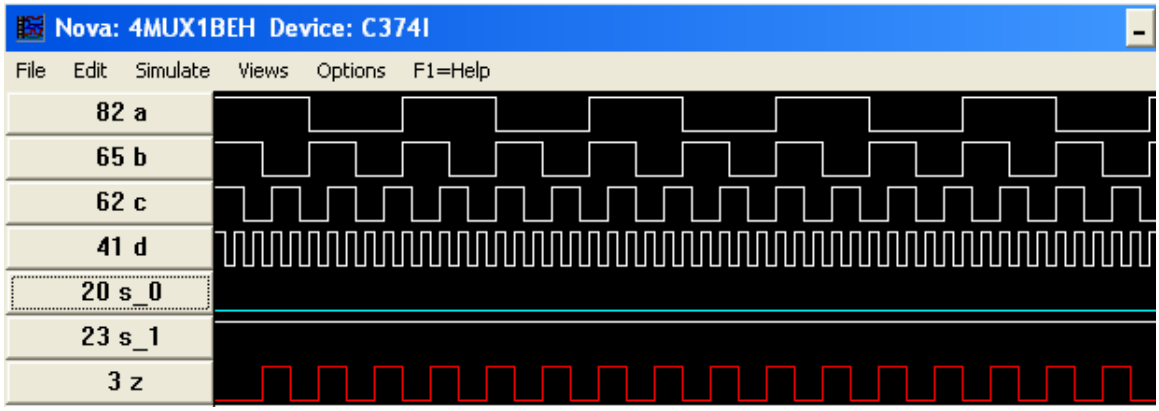


Figure 9. Mux 4/1 Functionality (C selected)

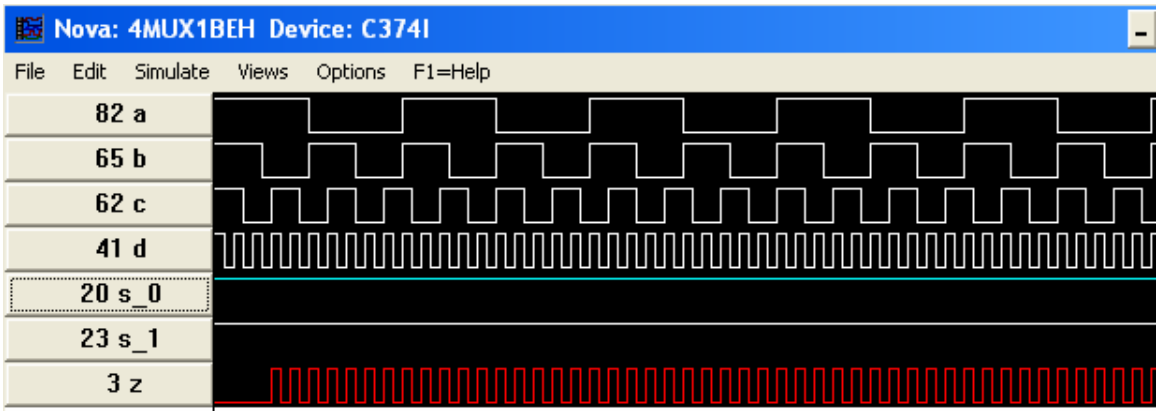


Figure 10. Mux 4/1 Functionality (D selected)

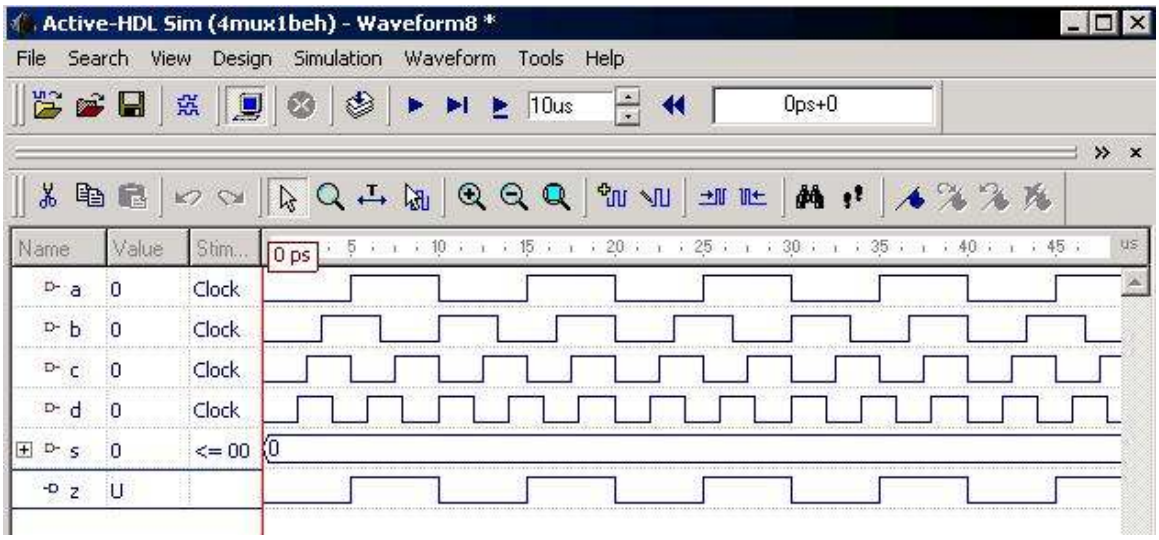


Figure 11. Mux 4/1 clocked at 100KHz

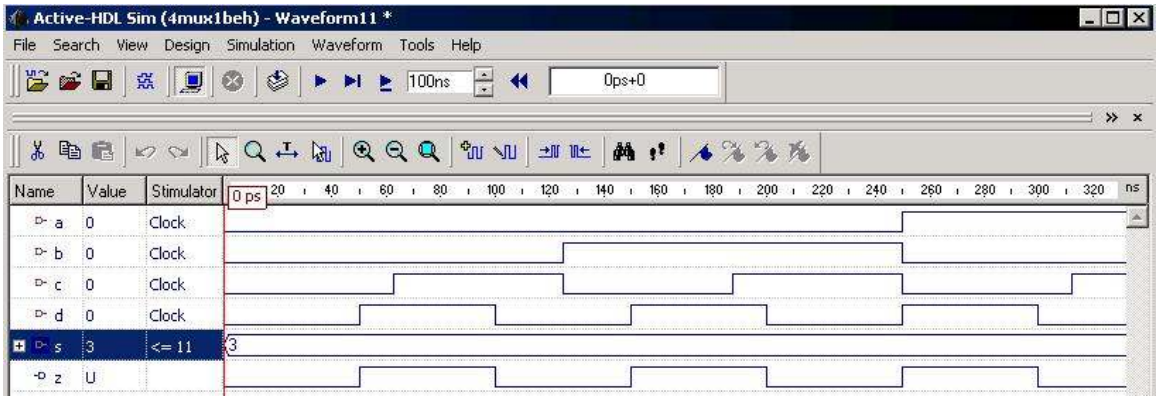
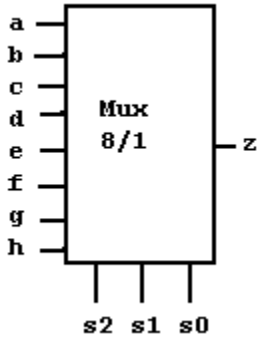


Figure 12. Mux 4/1 clocked at 10MHz

8 to 1 MULTIPLEXER



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity mux81 is
port (a, b, c, d, e, f, g, h : in std_logic;
s : in std_logic_vector (2 downto 0);
z : out std_logic);
end mux81;
 	
architecture archi of mux81 is
begin
process (a, b, c, d, e, f, g, h, s)
begin
if (s = "000") then z <= a;
elsif (s = "001") then z <= b;
elsif (s = "010") then z <= c;
elsif (s = "011") then z <= d;
elsif (s = "100") then z <= e;
elsif (s = "101") then z <= f;
elsif (s = "110") then z <= g;
else z <= h;
end if;
end process;
end archi;
	library ieee;
	use ieee.std_logic_1164.all;
	entity mux81 is
	port (a,b,c,d,e,f,g,h : in std_logic;
	s : in std_logic_vector (2 downto 0);
	z : out std_logic);
	end mux81;
	
	architecture archi of mux81 is
	begin
	with s select
	z <= a when "000",
	b when "001",
	c when "010",
	d when "011",
	e when "100",
	f when "101",
	g when "110",
	h when others;
	end archi;
		library ieee;
		use ieee.std_logic_1164.all;
		entity mux81 is
		port (a,b,c,d,e,f,g,h : in std_logic;
		s : in std_logic_vector (2 downto 0);
		z : out std_logic);
		end mux81;
		
		architecture archi of mux81 is
		begin
		z <= (a and (not s(0) and not s(1) and not s(2))) or
		(b and (not s(0) and not s(1) and s(2))) or
		(c and (not s(0) and s(1) and not s(2))) or
		(d and (not s(0) and s(1) and s(2))) or
		(e and (s(0) and not s(1) and not s(2))) or
		(f and (s(0) and not s(1) and s(2))) or
		(g and (s(0) and s(1) and not s(2))) or
		(h and (s(0) and s(1) and s(2)));
		end archi;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
z = a * /s_2 * /s_1 * /s_0 + e * s_2 * /s_1 * /s_0 + c * /s_2 * s_1 * /s_0 + g * s_2 * s_1 * /s_0 + b * /s_2 * /s_1 * s_0 + f * s_2 * /s_1 * s_0 + d * /s_2 * s_1 * s_0 + h * s_2 * s_1 * s_0	z = a * /s_2 * /s_1 * /s_0 + e * s_2 * /s_1 * /s_0 + c * /s_2 * s_1 * /s_0 + g * s_2 * s_1 * /s_0 + b * /s_2 * /s_1 * s_0 + f * s_2 * /s_1 * s_0 + d * /s_2 * s_1 * s_0 + h * s_2 * s_1 * s_0	z = a * /s_2 * /s_1 * /s_0 + b * s_2 * /s_1 * /s_0 + c * /s_2 * s_1 * /s_0 + d * s_2 * s_1 * /s_0 + e * /s_2 * /s_1 * s_0 + f * s_2 * /s_1 * s_0 + g * /s_2 * s_1 * s_0 + h * s_2 * s_1 * s_0

The generated equations in all cases are the same. The following figures show the verification and timing analysis at 100KHz and 10MHz. The device performs correctly at both frequency rates.

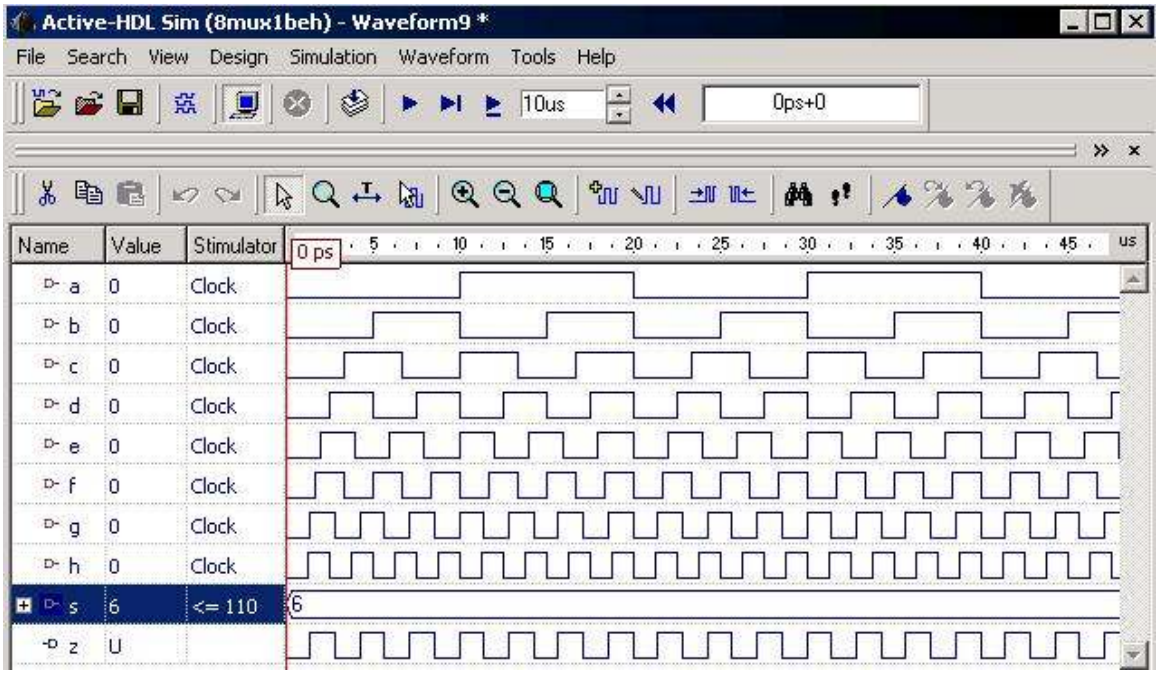


Figure 13. Mux 8/1 clocked at 100KHz

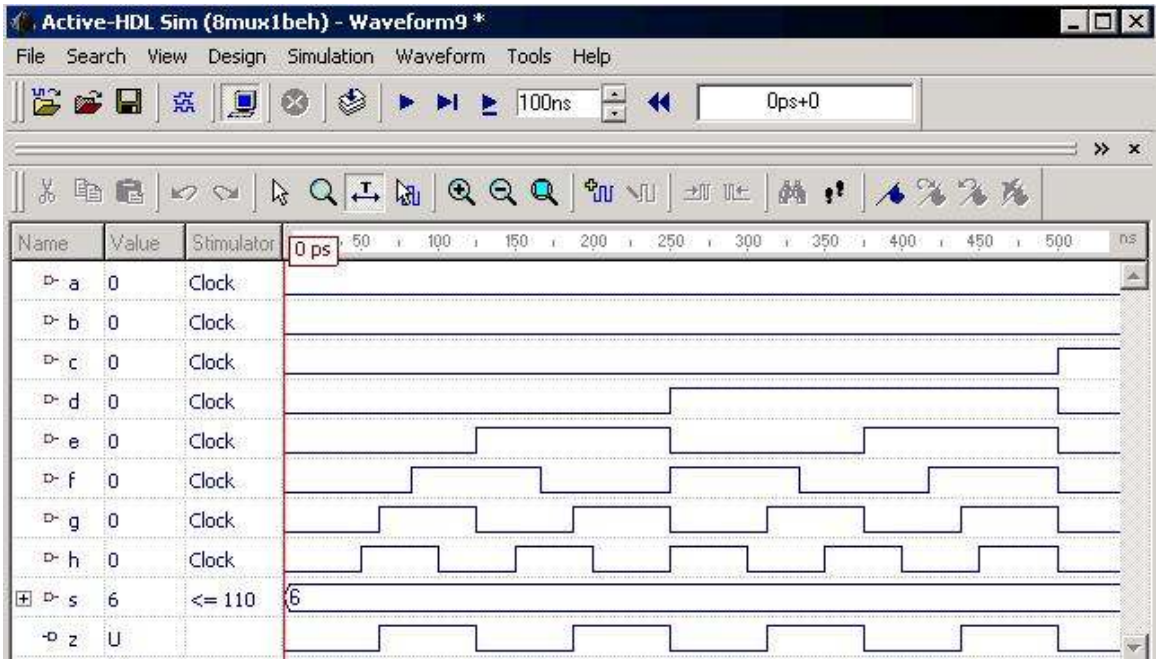
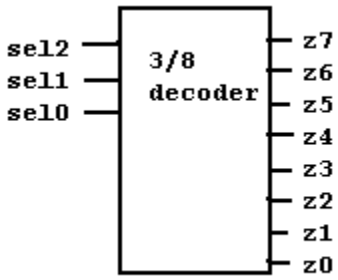


Figure 14. Mux 8/1 clocked at 10MHz

3 to 8 DECODER



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity decoder is
port (sel: in std_logic_vector (2 downto 0);
z: out std_logic_vector (7 downto 0));
end decoder;
architecture archi of decoder is
begin
process(sel) begin
if (sel = "000") then z <= "00000001";
elsif (sel = "001") then z <= "00000010";
elsif (sel = "010") then z <= "00000100";
elsif (sel = "011") then z <= "00001000";
elsif (sel = "100") then z <= "00010000";
elsif (sel = "101") then z <= "00100000";
elsif (sel = "110") then z <= "01000000";
else z <= "10000000";
end if;
end process;
end archi;
	library ieee;
	use ieee.std_logic_1164.all;
	entity decoder is
	port (sel: in std_logic_vector (2 downto 0);
	z: out std_logic_vector (7 downto 0));
	end decoder;
	architecture archi of decoder is
	begin
	z <= "00000001" when sel = "000" else
	"00000010" when sel = "001" else
	"00000100" when sel = "010" else
	"00001000" when sel = "011" else
	"00010000" when sel = "100" else
	"00100000" when sel = "101" else
	"01000000" when sel = "110" else
	"10000000";
	end archi;
		library ieee;
		use ieee.std_logic_1164.all;
		entity decoder is
		port (sel: in std_logic_vector (2 downto 0);
		z: out std_logic_vector (7 downto 0));
		end decoder;
		architecture archi of decoder is
		begin
		z(7) <= sel(2) and sel(1) and sel(0);
		z(6) <= sel(2) and sel(1) and not sel(0);
		z(5) <= sel(2) and not sel(1) and sel(0);
		z(4) <= sel(2) and not sel(1) and not sel(0);
		z(3) <= not sel(2) and sel(1) and sel(0);
		z(2) <= not sel(2) and sel(1) and not sel(0);
		z(1) <= not sel(2) and not sel(1) and sel(0);
		z(0) <= not sel(2) and not sel(1) and not sel(0);
		end archi;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
z_7 = sel_2 * sel_1 * sel_0 z_6 = sel_2 * sel_1 * /sel_0 z_5 = sel_2 * /sel_1 * sel_0 z_4 = sel_2 * /sel_1 * /sel_0 z_3 = /sel_2 * sel_1 * sel_0 z_2 = /sel_2 * sel_1 * /sel_0 z_1 = /sel_2 * /sel_1 * sel_0 z_0 = /sel_2 * /sel_1 * sel_0	z_7 = sel_2 * sel_1 * sel_0 z_6 = sel_2 * sel_1 * /sel_0 z_5 = sel_2 * /sel_1 * sel_0 z_4 = sel_2 * /sel_1 * /sel_0 z_3 = /sel_2 * sel_1 * sel_0 z_2 = /sel_2 * sel_1 * /sel_0 z_1 = /sel_2 * /sel_1 * sel_0 z_0 = /sel_2 * /sel_1 * /sel_0	z_7 = sel_2 * sel_1 * sel_0 z_6 = sel_2 * sel_1 * /sel_0 z_5 = sel_2 * /sel_1 * sel_0 z_4 = sel_2 * /sel_1 * /sel_0 z_3 = /sel_2 * sel_1 * sel_0 z_2 = /sel_2 * sel_1 * /sel_0 z_1 = /sel_2 * /sel_1 * sel_0 z_0 = /sel_2 * /sel_1 * /sel_0

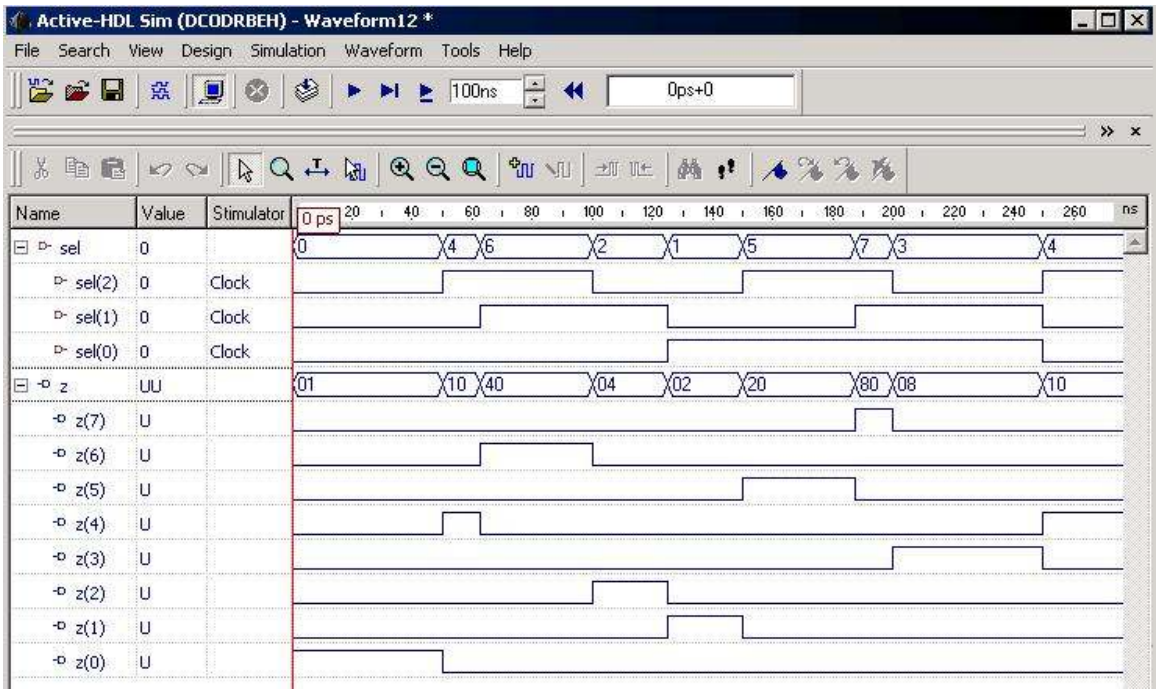
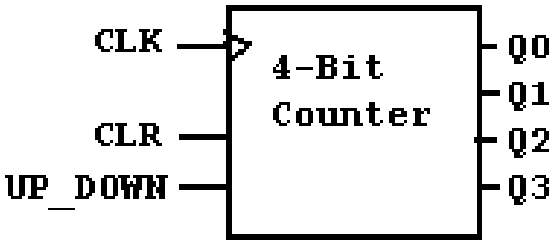


Figure 15. Decoder 3/8 clocked at 10MHz

4-BIT UP/DOWN COUNTER W/CLEAR



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
--use
ieee.std_logic_unsigned.all;
use work.std_arith.all;
entity counter is
  port(CLK, CLR, UP_DOWN :
in std_logic;
  Q : out std_logic_vector(3
downto 0));
end counter;
architecture archi of counter
is
  signal buff: std_logic_vector(3
downto 0);
  begin
  process (CLK, CLR)
  begin
  if (CLR='1') then
  buff <= "0000";
  elsif (CLK'event and
CLK='1') then
  if (UP_DOWN='1') then
  buff <= buff + 1;
  else
  buff <= buff - 1;
  end if;
  end if;
  end process;
  Q <= buff;
end archi;
	library ieee;
	use ieee.std_logic_1164.all;
	use work.std_arith.all;
	entity counter is
	  port(CLK, CLR, UP_DOWN : in
	std_logic;
	  Q : out std_logic_vector(3 downto
	0));
	end counter;
	
	architecture archi of counter is
	  signal buff: std_logic_vector(3
	downto 0);
	  begin
	  process (CLK, CLR,UP_DOWN)
	  begin
	  case CLR is
	  when '1' => buff <= "0000";
	  when others =>
	  if (CLK'event and CLK='1')
	then
	  case UP_DOWN is
	  when '1' => buff <= buff +
	1;
	  when others => buff <=
	buff - 1;
	  end case;
	  end if;
	  end case;
	  Q <= buff;
	  end process;
	end archi;	

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
q_3.T = /up_down * /q_2.Q * /q_1.Q * /q_0.Q + up_down * q_2.Q * q_1.Q * q_0.Q q_3.AP = GND q_3.AR = clr q_3.C = clk q_2.T = /up_down * /q_1.Q * /q_0.Q + up_down * q_1.Q * q_0.Q q_1.T = /up_down * /q_0.Q + up_down * q_0.Q q_0.D = /q_0.Q	q_3.T = /up_down * /q_2.Q * /q_1.Q * /q_0.Q + up_down * q_2.Q * q_1.Q * q_0.Q q_3.AP = GND q_3.AR = clr q_3.C = clk q_2.T = /up_down * /q_1.Q * /q_0.Q + up_down * q_1.Q * q_0.Q q_1.T = /up_down * /q_0.Q + up_down * q_0.Q q_0.D = /q_0.Q	

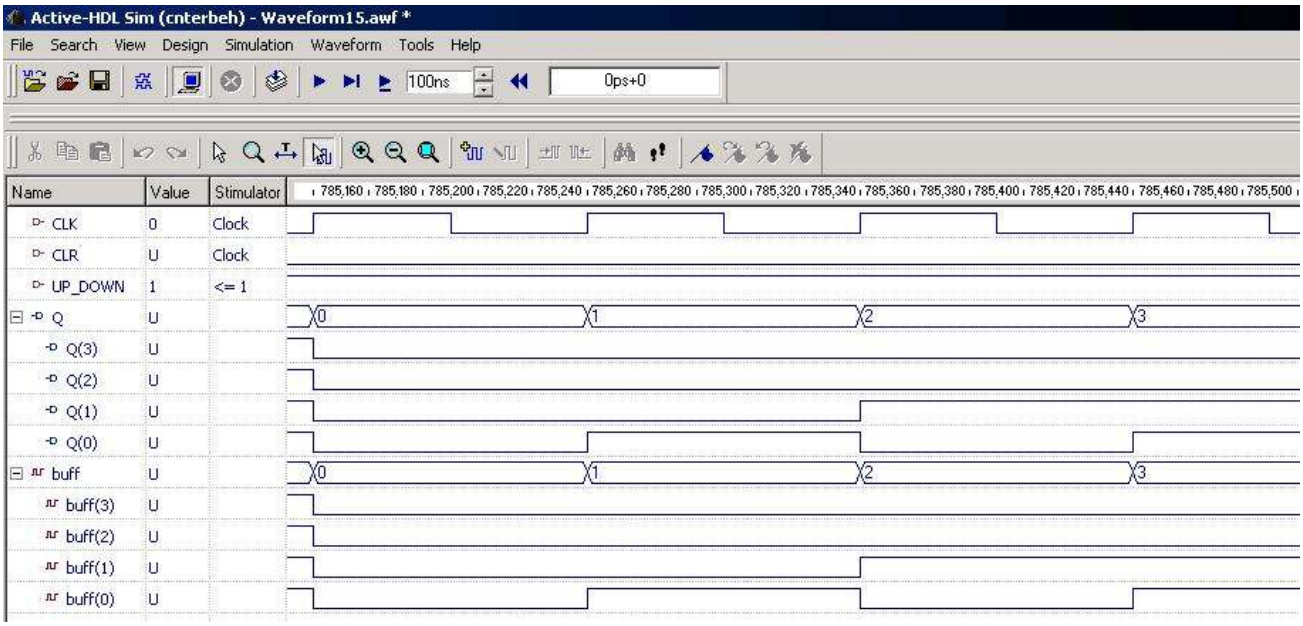
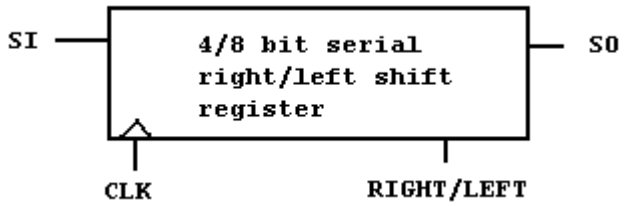


Figure 16. 4-Bit Counter clocked at 10MHz

4-BIT LEFT/RIGHT SHIFT REGISTER



Behavioral VHDL	Data Flow VHDL	Boolean VHDL

library ieee;
use ieee.std_logic_1164.all;
entity SRLSHIFT is
port(CLK : in std_logic;
UPDWN : in std_logic;
SI : in std_logic;
SO : out std_logic);
end SRLSHIFT;
 	
architecture ARCHI of SRLSHIFT is
signal BUFF: std_logic_vector(3 downto 0);
begin
process (CLK, UPDWN)
begin
if (CLK'event and CLK='1') then
if (UPDWN = '1') then
-- Shift Right ---------
for i in 0 to 2 loop
BUFF(i+1) <= BUFF(i);
end loop;
BUFF(0) <= SI;
SO <= BUFF(3);
else
-- Shift Left ----------
for i in 3 downto 1 loop
BUFF(i-1) <= BUFF(i);
end loop;
BUFF(3) <= SI;
SO <= BUFF(0);
end if;
end if;
end process;
end ARCHI;
	library ieee;
	use ieee.std_logic_1164.all;
	entity SRLSHIFT is
	port(CLK : in std_logic;
	UPDWN : in std_logic;
	SI : in std_logic;
	SO : out std_logic);
	end SRLSHIFT;
	
	architecture ARCHI of SRLSHIFT is
	signal BUFF: std_logic_vector(3 downto 0);
	begin
	process (CLK, UPDWN)
	begin
	if (CLK'event and CLK='1') then
	CASE UPDWN IS
	WHEN '1' =>
	-- Shift Right ---------
	for i in 0 to 2 loop
	BUFF(i+1) <= BUFF(i);
	end loop;
	BUFF(0) <= SI;
	SO <= BUFF(3);
	WHEN OTHERS =>
	-- Shift Left ----------
	for i in 3 downto 1 loop
	BUFF(i-1) <= BUFF(i);
	end loop;
	BUFF(3) <= SI;
	SO <= BUFF(0);
	END CASE;
	end if;
	end process;
	end ARCHI;
		library ieee;
		use ieee.std_logic_1164.all;
		entity SRLSHIFT is
		port(CLK : in std_logic;
		UPDWN : in std_logic;
		SI : in std_logic;
		SO : out std_logic);
		end SRLSHIFT;
		
		architecture ARCHI of SRLSHIFT is
		signal BUFF: std_logic_vector(3 downto 0);
		begin
		SO <= (UPDWN and BUFF(3)) or (not UPDWN and BUFF(0));
		BUFF(1) <= (not UPDWN and BUFF(2)) or (UPDWN and BUFF(0));
		BUFF(0) <= (not UPDWN and BUFF(1)) or (UPDWN and SI);
		BUFF(2) <= (UPDWN and BUFF(1)) or (not UPDWN and BUFF(3));
		end ARCHI;

Generated Logic Equations	**Generated Logic Equations**	**Generated Logic Equations**
so.D = updwn * buff_3.Q + /updwn * buff_0.Q buff_1.D = /updwn * buff_2.Q + updwn * buff_0.Q buff_0.D = /updwn * buff_1.Q + updwn * si buff_2.D = updwn * buff_1.Q + /updwn * buff_3.Q buff_3.D = /updwn * si + updwn * buff_2.Q	so.D = updwn * buff_3.Q + /updwn * buff_0.Q buff_1.D = /updwn * buff_2.Q + updwn * buff_0.Q buff_0.D = /updwn * buff_1.Q + updwn * si buff_2.D = updwn * buff_1.Q + /updwn * buff_3.Q buff_3.D = /updwn * si + updwn * buff_2.Q	so.D = updwn * buff_3.Q + /updwn * buff_0.Q buff_1.D = /updwn * buff_2.Q + updwn * buff_0.Q buff_0.D = /updwn * buff_1.Q + updwn * si buff_2.D = updwn * buff_1.Q + /updwn * buff_3.Q

8-BIT LEFT/RIGHT SHIFT REGISTER

Behavioral VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity SrlShift is
  port(CLK  : in std_logic;
       UPDWN : in std_logic;
       SI    : in std_logic;
       SO    : out std_logic);
end SrlShift;

architecture Archi of SrlShift
is
  signal buff: std_logic_vector(7
downto 0);
  begin
    process (CLK,UPDWN)
      begin
        if (CLK'event and
CLK='1') then
          if (UPDWN = '1') then
            -- Shift Right ---------
            for i in 0 to 6 loop
               buff(i+1) <= buff(i);
            end loop;
            buff(0) <= SI;
            SO <= buff(7);
          else
            -- Shift Left ----------
            for i in 7 downto 1
loop
               buff(i-1) <= buff(i);
            end loop;
            buff(7) <= SI;
            SO <= buff(0);
          end if;
        end if;
    end process;
end Archi;
```

Generated Logic Equations

```
so.D =  updwn * buff_7.Q
      + /updwn * buff_0.Q
so.AP = GND
so.AR = GND
so.C =   clk

buff_0.D =  /updwn * buff_1.Q
      + updwn * si
```

Data Flow VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity SrlShift is
  port(CLK  : in std_logic;
       UPDWN : in std_logic;
       SI    : in std_logic;
       SO    : out std_logic);
end SrlShift;

architecture Archi of SrlShift is
  signal buff: std_logic_vector(7
downto 0);
  begin
    process (CLK,UPDWN)
      begin
        if (CLK'event and CLK='1') then
          CASE UPDWN IS
            WHEN '1' =>
            -- Shift Right ---------
            for i in 0 to 6 loop
               buff(i+1) <= buff(i);
            end loop;
            buff(0) <= SI;
            SO <= buff(7);
            WHEN OTHERS =>
            -- Shift Left ----------
            for i in 7 downto 1 loop
               buff(i-1) <= buff(i);
            end loop;
            buff(7) <= SI;
            SO <= buff(0);
          END CASE;
        end if;
    end process;
end Archi;
```

Generated Logic Equations

```
so.D =  updwn * buff_7.Q
      + /updwn * buff_0.Q
so.AP = GND
so.AR = GND
so.C =   clk

buff_0.D =  /updwn * buff_1.Q
      + updwn * si
```

Boolean VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity SrlShift is
  port(CLK  : in std_logic;
       UPDWN : in std_logic;
       SI    : in std_logic;
       SO    : out std_logic);
end SrlShift;

architecture Archi of
SrlShift is
  signal BUFF:
std_logic_vector(7 downto
0);
  begin
    SO <= (UPDWN and
BUFF(7)) or (not UPDWN
and BUFF(0));
    BUFF(0) <= (not UPDWN
and BUFF(1)) or (UPDWN
and SI);
    BUFF(1) <= (not UPDWN
and BUFF(2)) or (UPDWN
and BUFF(0));
    BUFF(2) <= (not UPDWN
and BUFF(3)) or (UPDWN
and BUFF(1));
    BUFF(3) <= (not UPDWN
and BUFF(4)) or (UPDWN
and BUFF(2));
    BUFF(4) <= (not UPDWN
and BUFF(5)) or (UPDWN
and BUFF(3));
    BUFF(5) <= (not UPDWN
and BUFF(6)) or (UPDWN
and BUFF(5));
    BUFF(6) <= (not UPDWN
and BUFF(5)) or (UPDWN
and BUFF(7));
    BUFF(7) <= (UPDWN and
BUFF(6)) or (not UPDWN
and SI);
end Archi;
```

Generated Logic Equations

```
so.D =  updwn * buff_7.Q
      + /updwn * buff_0.Q
so.AP = GND
so.AR = GND
so.C =   clk

buff_0.D =  /updwn *
buff_1.Q
```

```
buff_1.D =   /updwn * buff_2.Q
      + updwn * buff_0.Q

buff_2.D =   /updwn * buff_3.Q
      + updwn * buff_1.Q

buff_3.D =   /updwn * buff_4.Q
      + updwn * buff_2.Q

buff_4.D =  /updwn * buff_5.Q
      + updwn * buff_3.Q

buff_5.D =  /updwn * buff_6.Q
      + updwn * buff_4.Q

buff_6.D =   updwn * buff_5.Q
      + /updwn * buff_7.Q

buff_7.D =  updwn * buff_6.Q
      + /updwn * si
```

```
buff_1.D =   /updwn * buff_2.Q
      + updwn * buff_0.Q

buff_2.D =   /updwn * buff_3.Q
      + updwn * buff_1.Q

buff_3.D =   /updwn * buff_4.Q
      + updwn * buff_2.Q

buff_4.D =  /updwn * buff_5.Q
      + updwn * buff_3.Q

buff_5.D =  /updwn * buff_6.Q
      + updwn * buff_4.Q

buff_6.D =   updwn * buff_5.Q
      + /updwn * buff_7.Q

buff_7.D =  updwn * buff_6.Q
      + /updwn * si
```

```
      + updwn * si

buff_1.D =   /updwn *
buff_2.Q
      + updwn * buff_0.Q

buff_2.D =   /updwn *
buff_3.Q
      + updwn * buff_1.Q

buff_3.D =   /updwn *
buff_4.Q
      + updwn * buff_2.Q

buff_4.D =  /updwn *
buff_5.Q
      + updwn * buff_3.Q

buff_5.D =  /updwn *
buff_6.Q
      + updwn * buff_4.Q

buff_6.D =   updwn *
buff_5.Q
      + /updwn * buff_7.Q

buff_7.D =  updwn *
buff_6.Q
      + /updwn * si
```

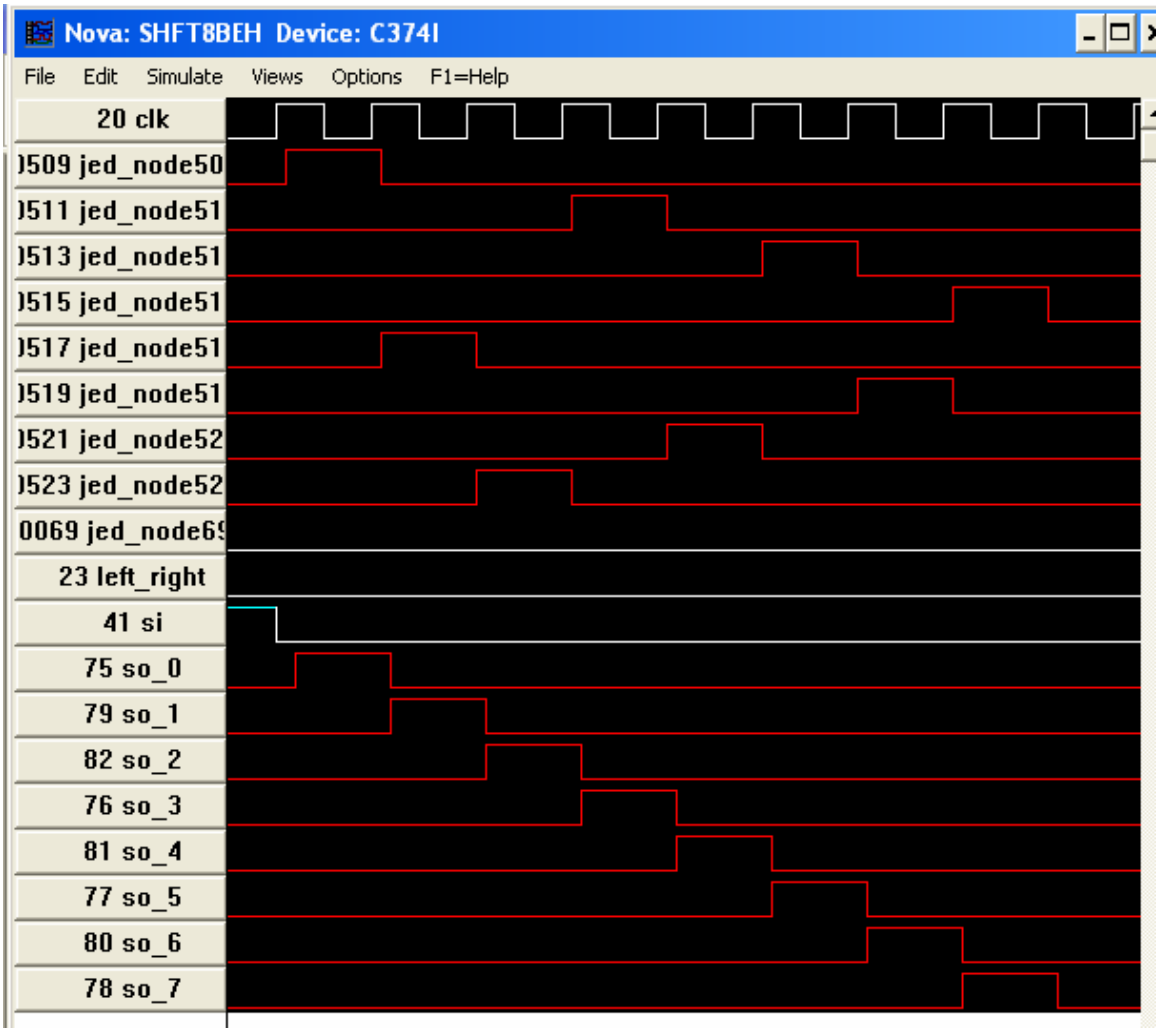


Figure 17. 8-BIT Serial Right Shift Register

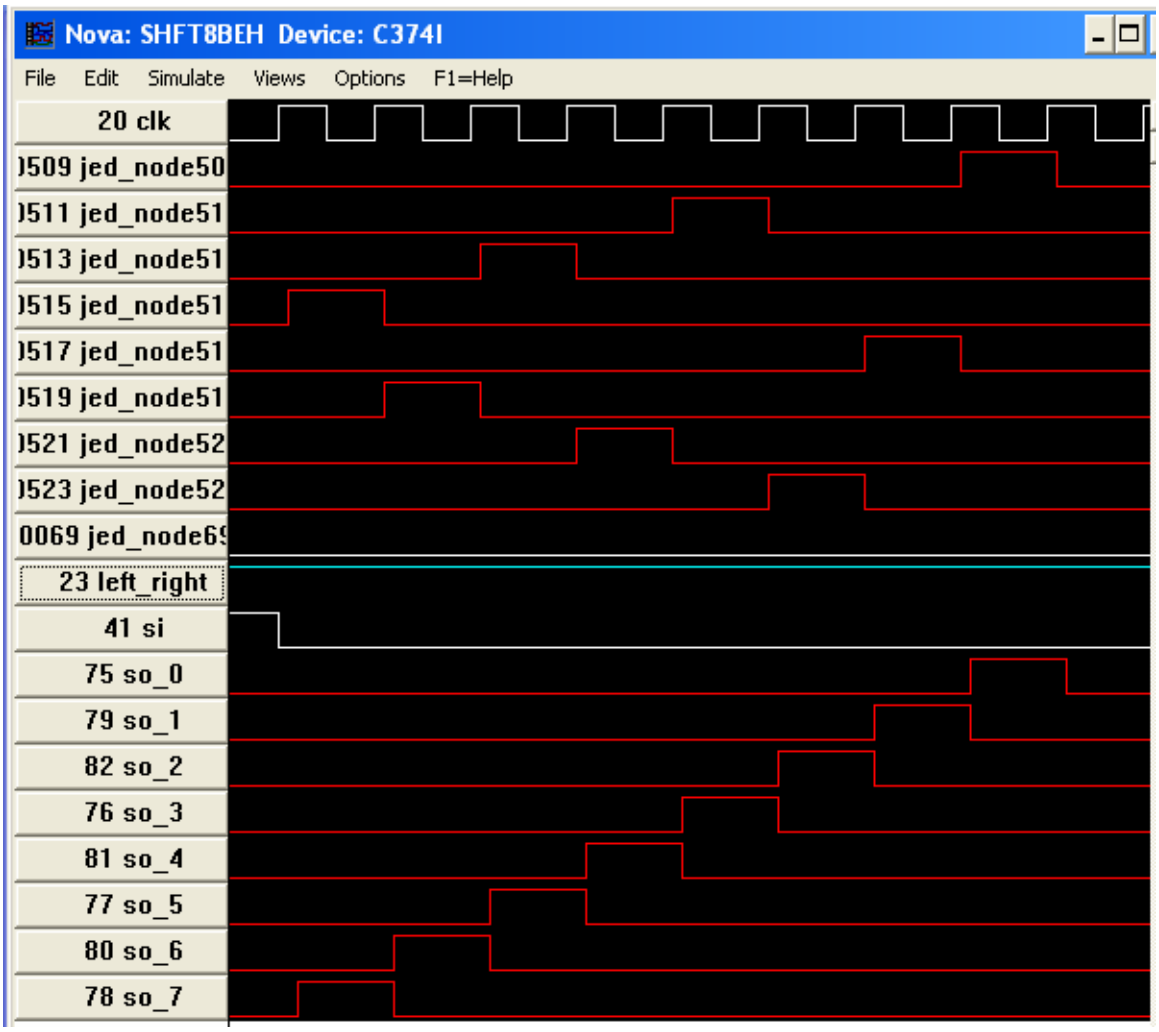


Figure 18. 8-BIT Serial Left Shift Register

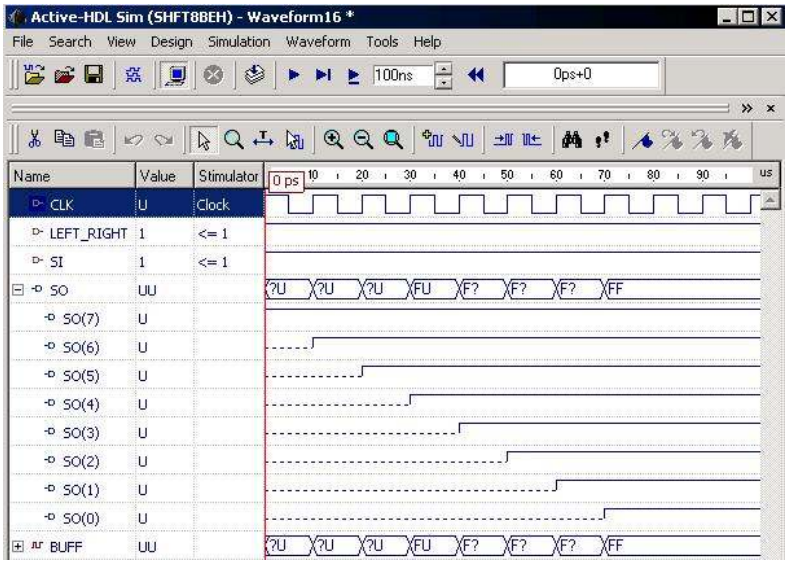


Figure 19. 8-BIT Serial Right Shift Register clocked at 100KHz

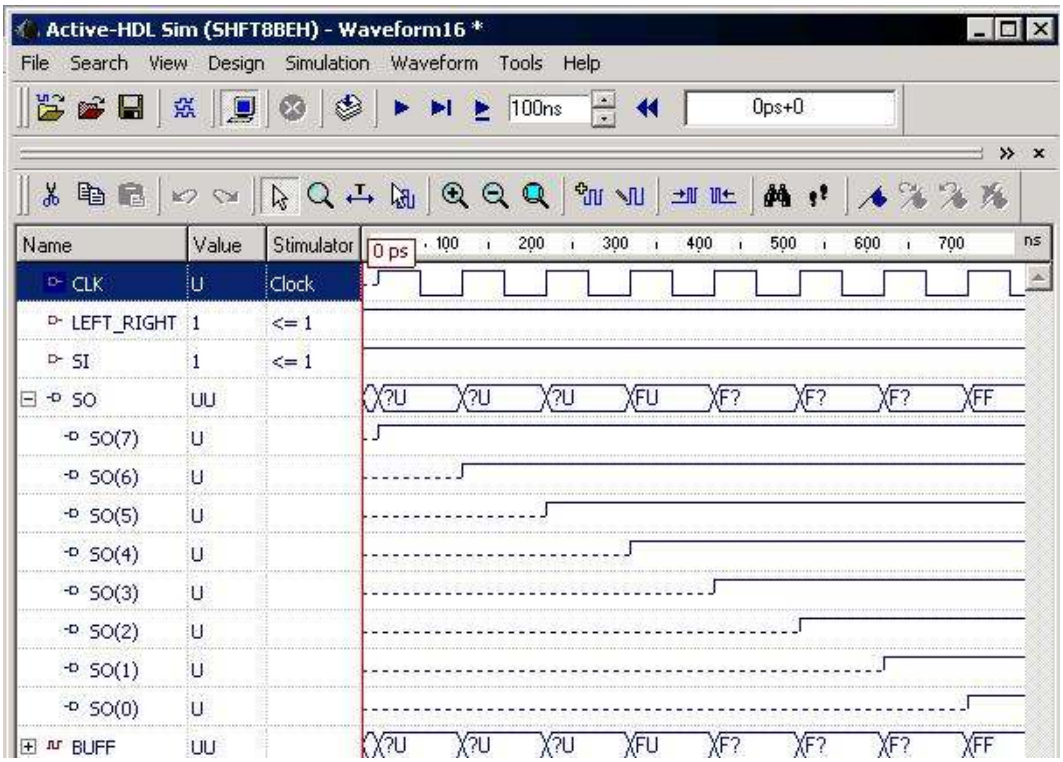


Figure 20. 8-BIT Serial Right Shift Register clocked at 10MHz

CONCLUSIONS

Several combinational and register based devices were successfully implemented using three types of VHDL code: behavioral, data flow and Boolean. The Cypress CPLD CY374I performs correctly at clock rates of 100KHz and 10MHz. No difference in functionality was found simulating the devices at these two rates.

The VHDL coding can now be coupled to C++ and Java and can be made truly interactive with the user groups, stationed at different locations and organizations. Also, as the FPGA costs are coming down as of 2006, it may be instructive to redo the assignment with the FPGA platforms from Cypress or Xilinx or Altera, and compare the results with the CPLD in terms of power and resource management.

Pablo Gomez

Subbarao V. Wunnava

FIU Electrical and Computer Engineering

May 15, 2006