****

2017-18

**DCS-2**

**ECE-407**

**AMITESH K SINGH**

|  |  |  |
| --- | --- | --- |
| S.NO | EXPERIMENT | REMARKS |
|  |  |  |
|  |  |  |

SWITCH DEBOUNCING[2 type]

library IEEE;

use ieee.std\_logic\_unsigned.all;

use IEEE.STD\_LOGIC\_1164.ALL;

entity debounce2 is

Port ( button : in STD\_LOGIC;

clk : in STD\_LOGIC;

res : out STD\_LOGIC);

end debounce2;

architecture Behavioral of debounce2 is

signal ff:std\_logic\_vector(1 downto 0);

signal count\_out:std\_logic\_vector(2 downto 0);

signal count\_clr:std\_logic;

begin

count\_clr<=ff(1) xor ff(0);

process(button,clk)

begin

if(rising\_edge(clk))

then

ff(0)<=button;

ff(1)<=ff(0);

if(count\_clr='1')

then

count\_out<=(others=>'0');

elsif (count\_out(2)='0')

then

count\_out<=count\_out + '1';

else

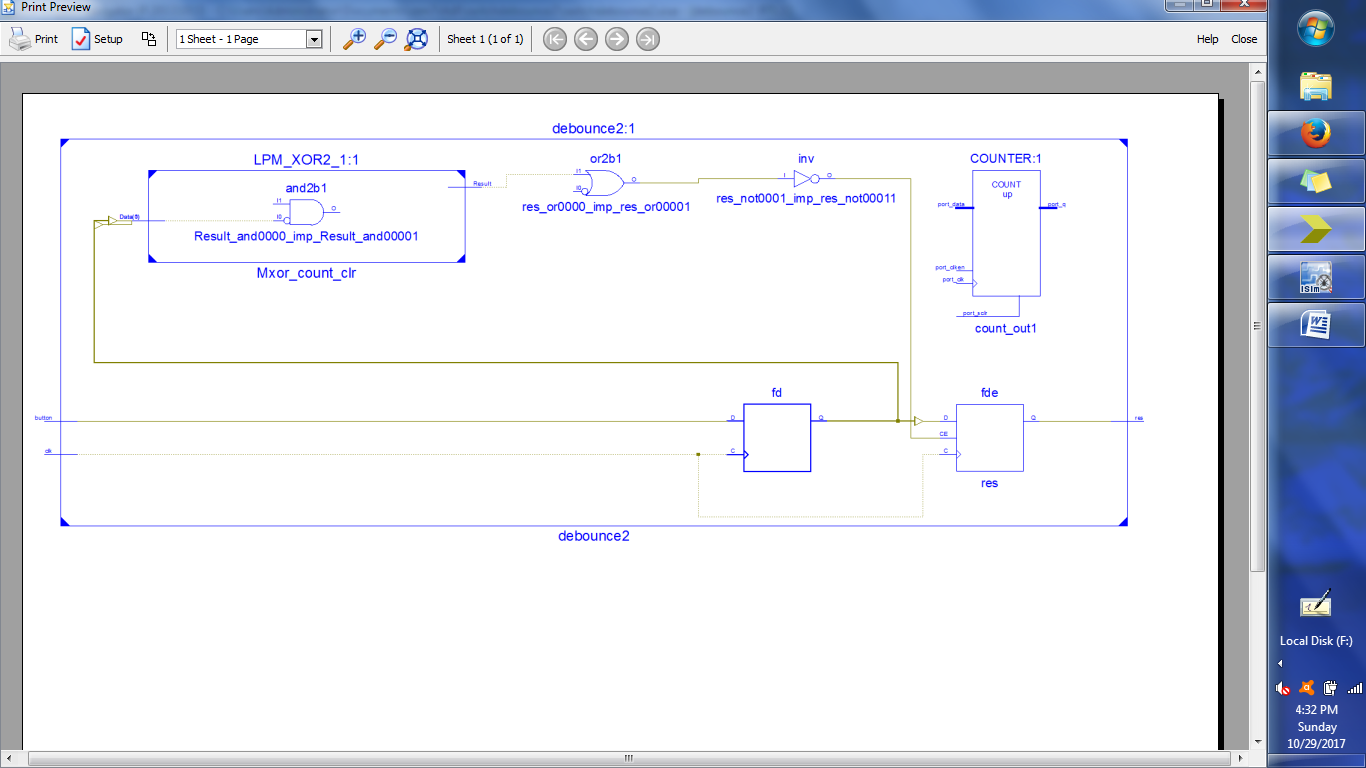
res<=ff(1);

end if;

end if;

end process;

end Behavioral;



# TestBench:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY debouncetb IS

END debouncetb;

ARCHITECTURE behavior OF debouncetb IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT debounce2

PORT(

button : IN std\_logic;

clk : IN std\_logic;

res : OUT std\_logic

);

END COMPONENT;

--Inputs

signal button : std\_logic := '0';

signal clk : std\_logic := '0';

--Outputs

signal res : std\_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: debounce2 PORT MAP (

button => button,

clk => clk,

res => res

);

-- Clock process definitions

clk\_process :process

begin

clk <= '0';

wait for 10 ns;

clk <= '1';

wait for 10 ns;

end process;

-- Stimulus process

stim\_proc: process

begin

-- hold reset state for 100 ns.

--wait for 100 ns;

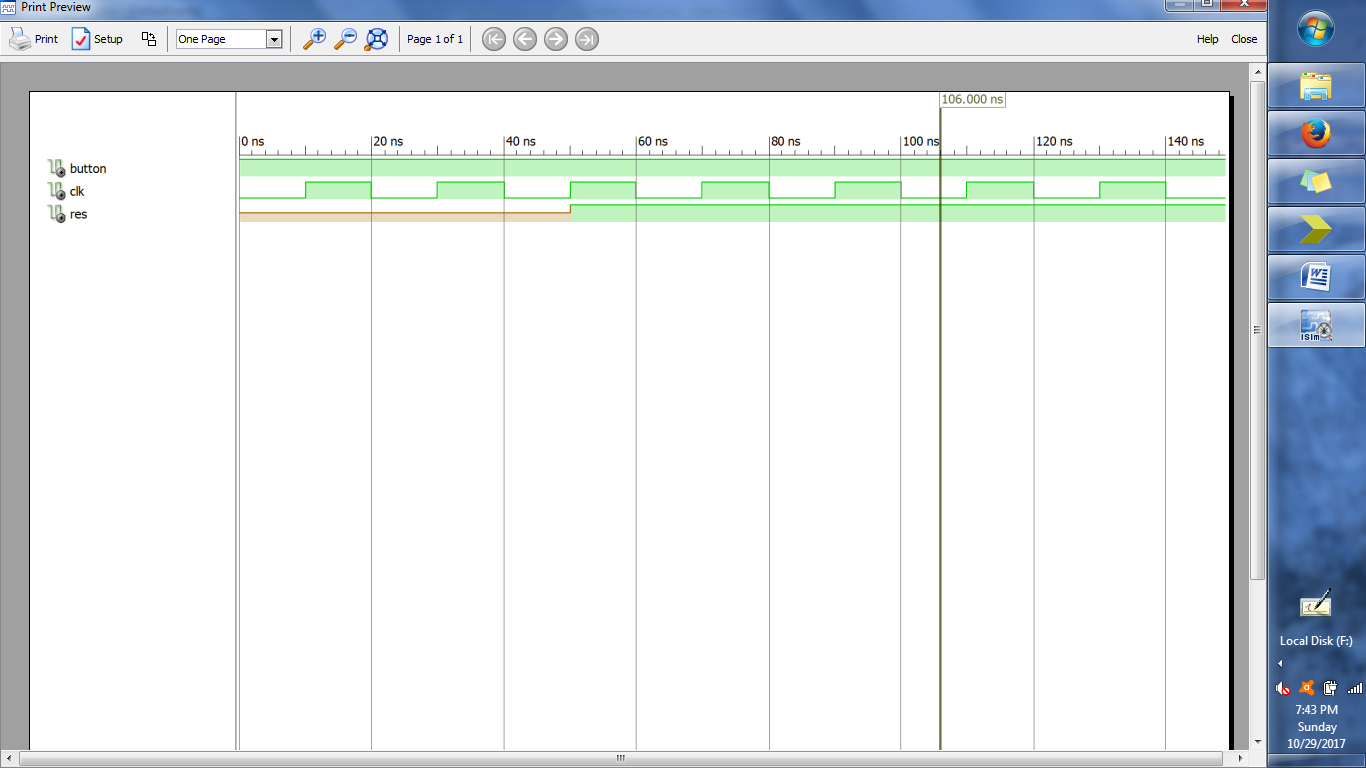
button<='1';

wait for 100 ns;

wait;

end process;

END;



RANDOM NO. GENERATOR USING LFSR

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity prng is

Port ( clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

rnum : out STD\_LOGIC\_vector(7 downto 0));

end prng;

architecture Behavioral of prng is

signal ff:std\_logic\_vector(7 downto 0);

signal temp:std\_logic:='0';

begin

temp <= ff(4)xor ff(3) xor ff(2) xor ff(0);

process(clk,rst)

begin

if(rst='1')

then

ff<="00000001";

elsif(rising\_edge(clk))then

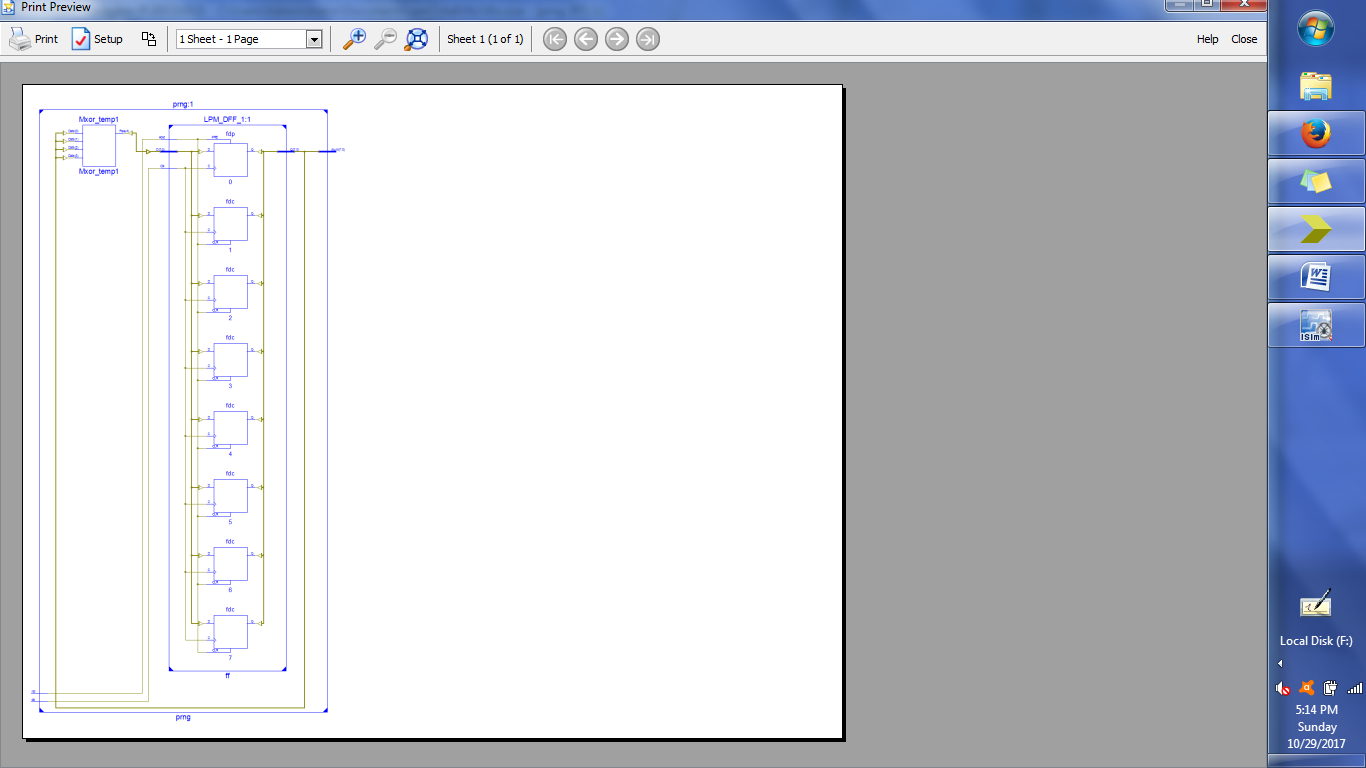
ff<=temp & ff(7 downto 1);

end if;

end process;

rnum<=ff;

end Behavioral;



## TESTBENCH:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY tblfsr IS

END tblfsr;

ARCHITECTURE behavior OF tblfsr IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT prng

PORT(

clk : IN std\_logic;

rst : IN std\_logic;

rnum : OUT std\_logic\_vector(7 downto 0)

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0';

signal rst : std\_logic := '0';

--Outputs

signal rnum : std\_logic\_vector(7 downto 0);

-- Clock period definitions

-- constant clk\_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: prng PORT MAP (

clk => clk,

rst => rst,

rnum => rnum

);

clk\_process :process

begin

clk <= '0';

wait for 10 ns;

clk <= '1';

wait for 10 ns;

end process;

-- Stimulus process

stim\_proc: process

begin

-- hold reset state for 100 ns.

wait for 10 ns;

rst<='1' ;

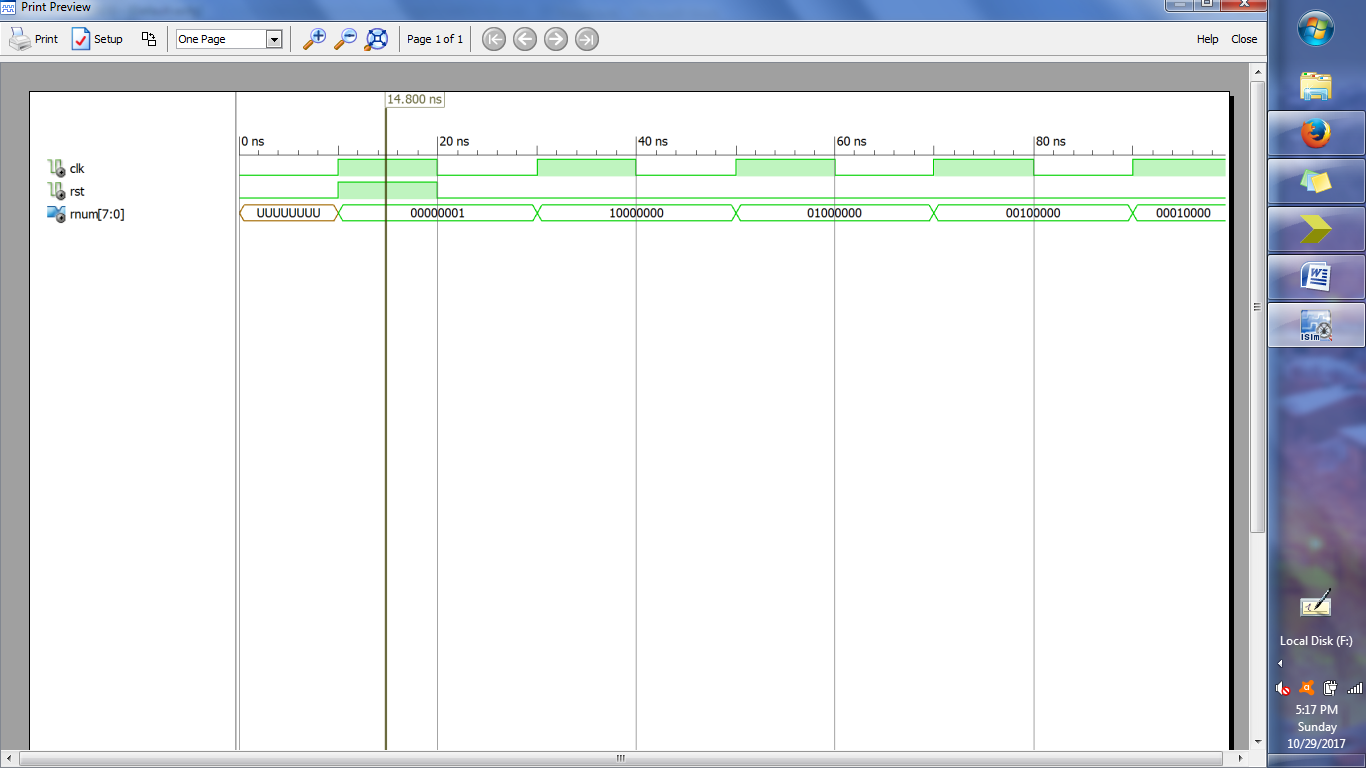
wait for 10 ns;

rst<='0';

wait;

end process;

END;



DICE GAME

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_unsigned.ALL;

entity game1 is

Port ( sw : in STD\_LOGIC;

clk : in STD\_LOGIC;

clk\_new : in STD\_LOGIC;

rst : in STD\_LOGIC;

led\_w : out STD\_LOGIC;

led\_l : out STD\_LOGIC;

led\_pa : out STD\_LOGIC;

ssd : out STD\_LOGIC\_vector(8 downto 0));

end game1;

architecture Behavioral of game1 is

signal p\_state,n\_state: integer range 0 to 3;

signal roll\_value,roll\_store:integer range 0 to 6;

begin

process(p\_state,sw,clk\_new)

begin

case p\_state is

when 0=>

if(sw='0')then

n\_state<=0;led\_w<='0';led\_l<='0';led\_pa<='0';ssd<="011110111";

roll\_value<=1;roll\_store<=0;

else

n\_state<=1;led\_w<='0';led\_l<='0';led\_pa<='0';ssd<="011110111";

roll\_value<=1;roll\_store<=0;

end if;

when 1=> if(sw='1')then

n\_state<=1;

if(rising\_edge(clk\_new))then

if(roll\_value=6)then

roll\_value<=1;

else

roll\_value<=roll\_value +1;

end if;

end if;

else

n\_state<=2;led\_pa<='1';roll\_store<=roll\_value;--storing 1 value

end if;

when 2=> --ssd o/p

if(roll\_value=1)then ssd<="011111001";

elsif(roll\_value=2)then ssd<="010100100";

elsif(roll\_value=3)then ssd<="010110000";

elsif(roll\_value=4)then ssd<="011100110";

elsif(roll\_value=5)then ssd<="011101101";

else ssd<="010000010";

end if;

if(sw='0')then--keep the display on

n\_state<=2;

else

n\_state<=3;--sw=1

end if;

when 3=> if(sw='1')then

n\_state<=3;

if(rising\_edge(clk\_new))then

if(roll\_value=6)then

roll\_value<=1;

else

roll\_value<=roll\_value+1;

end if;

end if;

else --

n\_state<=0;led\_pa<='0';

if(roll\_store=roll\_value)then

led\_w<='1';led\_l<='0';

else

led\_w<='0';led\_l<='1';

end if;

end if;

end case;

end process;

process(clk,rst)

begin

if(rst='1')then

p\_state<=0;

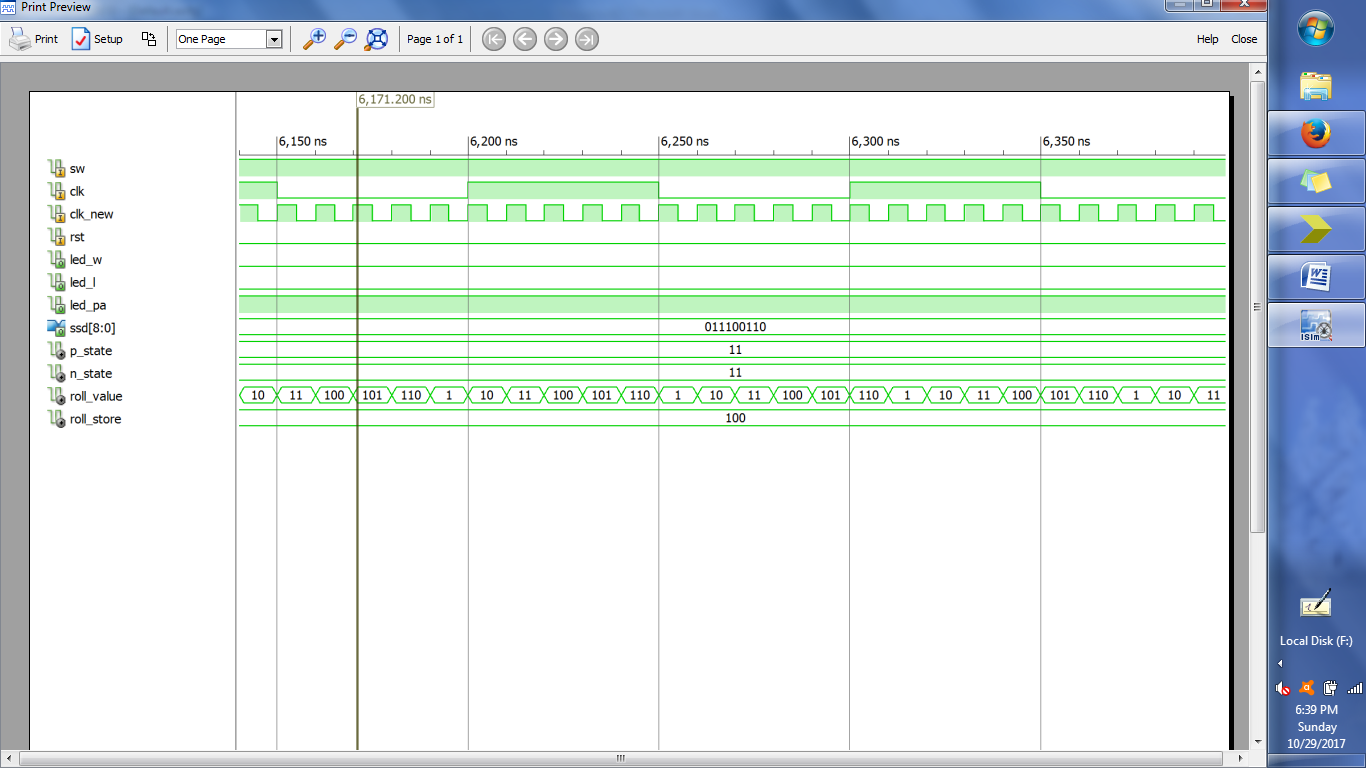
elsif(rising\_edge(clk))then

p\_state<=n\_state;

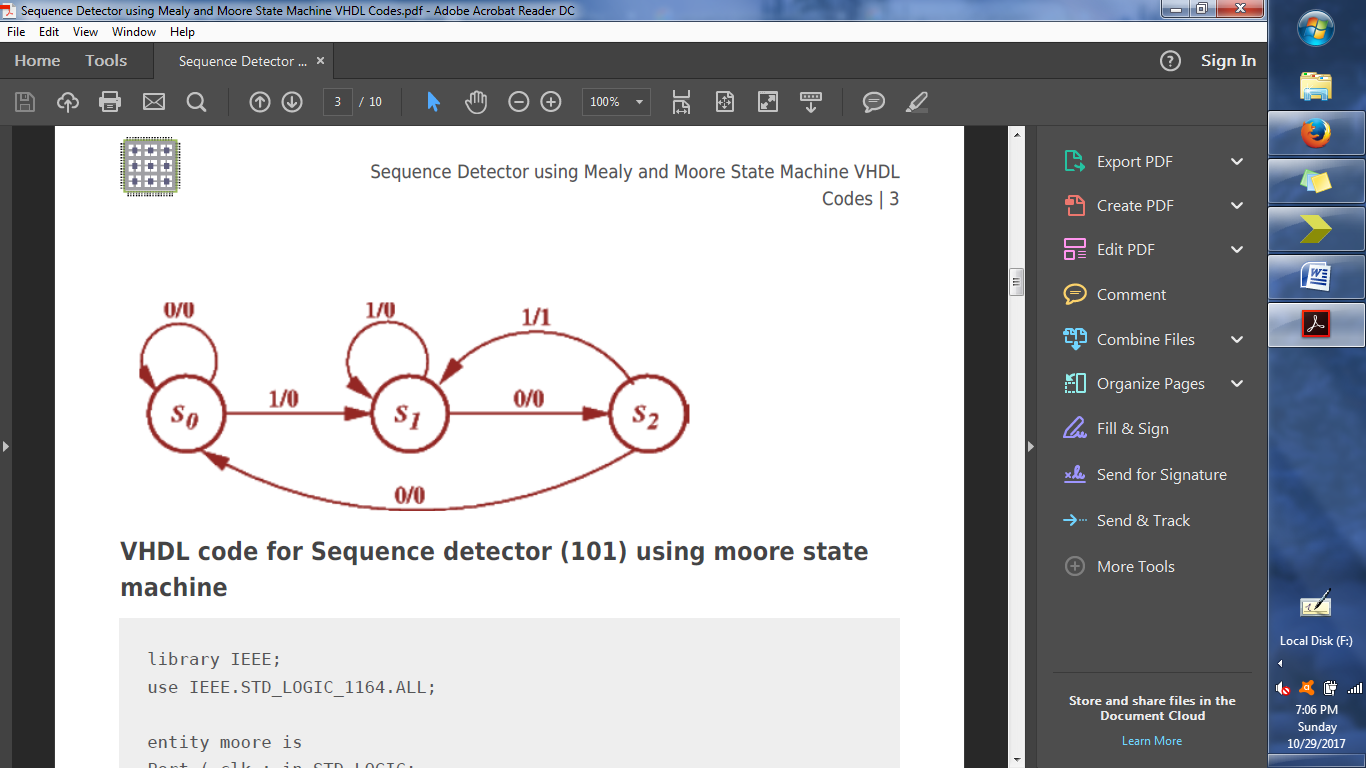
end if;

end process;

end Behavioral;



SEQUENCE DETECTOR[101]



library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity seq is

Port ( clk : in STD\_LOGIC;

din : in STD\_LOGIC;

rst : in STD\_LOGIC;

dout : out STD\_LOGIC);

end seq;

architecture Behavioral of seq is

type state is (st0, st1, st2, st3);

signal present\_state, next\_state : state;

begin

syncronous\_process : process (clk)

begin

if rising\_edge(clk) then

if (rst = '1') then

present\_state <= st0;

else

present\_state <= next\_state;

end if;

end if;

end process;

next\_state\_and\_output\_decoder : process(present\_state, din)

begin

dout <= '0';

case (present\_state) is

when st0 =>

if (din = '1') then

next\_state <= st1;

dout <= '0';

else

next\_state <= st0;

dout <= '0';

end if;

when St1 =>

if (din = '1') then

next\_state <= st1;

dout <= '0';

else

next\_state <= st2;

dout <= '0';

end if;

when St2 =>

if (din = '1') then

next\_state <= st1;

dout <= '1';

else

next\_state <= st0;

dout <= '0';

end if;

when others =>

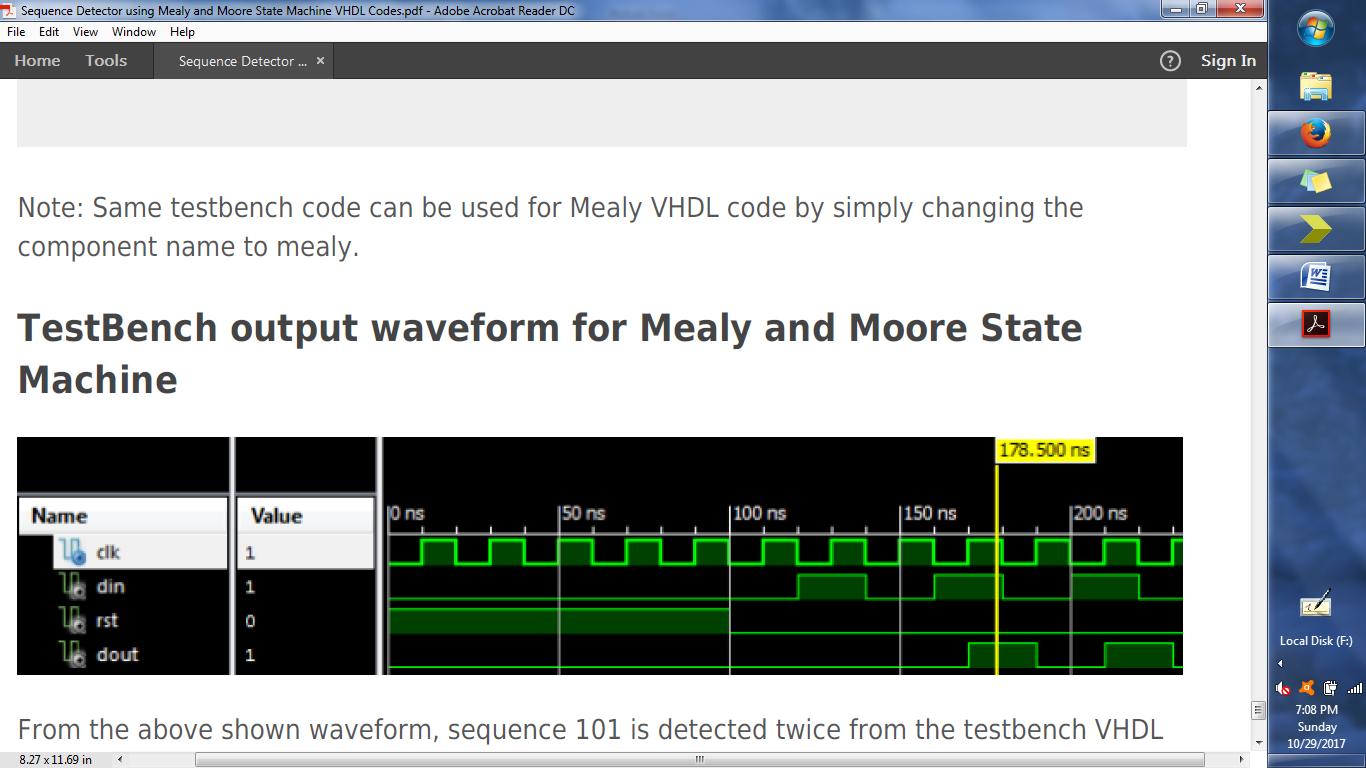
next\_state <= st0;

dout <= '0';

end case;

end process;

end Behavioral;



DICE GAME 2

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity datapath is

port(

button : in std\_logic ;

led : out std\_logic\_vector(3 downto 0 ) ;

clk : in std\_logic ;

display1 : out std\_LOGIC\_VECTOR(6 downto 0 ) ;

display2: out std\_logic\_vector(6 downto 0) );

end datapath;

architecture Behavioral of datapath is

signal c2\_en, c1\_en, l1\_en, l2\_en,switch: std\_logic ;

signal count1, count2,lat1\_count2, lat2\_count1, lat1\_count1,lat2\_count2 : std\_logic\_vector(3 downto 0 ) ;

signal sum1, sum2 : std\_logic\_vector( 3 downto 0 ) ;

signal eq : std\_logic\_vector(1 downto 0 ) ;

component counter\_en port (

en: in std\_logic ;

clk : in std\_logic ;

count: out std\_logic\_vector(3 downto 0)

);

end component ;

component latch\_en port( D : in STD\_LOGIC\_vector(3 downto 0 );

Q : out STD\_LOGIC\_vector(3 downto 0 ) ;

en : in STD\_LOGIC);

end component ;

component summer port(

ch1: in std\_logic\_vector(3 downto 0 ) ;

ch2 : in std\_logic\_vector( 3 downto 0 ) ;

sum : out std\_logic\_vector( 3 downto 0 ) );

end component ;

component comparator port(

sum1 : in std\_logic\_vector(3 downto 0 ) ;

sum2 : in std\_logic\_vector( 3 downto 0 ) ;

eq : out std\_logic\_vector( 1 downto 0 ) );

end component;

component decoder\_bcd\_7seg port (count : in STD\_LOGIC\_VECTOR (3 downto 0);

display : out STD\_LOGIC\_VECTOR (6 downto 0));

end component ;

component debounce port(clk : IN STD\_LOGIC ;

button : IN STD\_LOGIC;

result : OUT STD\_LOGIC);

end component ;

component dicegame port (

sw : in std\_logic ;

eq : in std\_logic\_vector(1 downto 0 ) ;

clk : in std\_logic ;

output: out std\_logic\_vector(7 downto 0) );

end component ;

begin

A1: counter\_en port map ( c1\_en, clk, count1) ;

A2: counter\_en port map ( c2\_en , clk , count2) ;

A3: latch\_en port map ( count1, lat1\_count1 ,l1\_en) ;

A4: latch\_en port map ( count2, lat1\_count2 ,l1\_en) ;

A5: latch\_en port map ( count1, lat2\_count1,l2\_en) ;

A6: latch\_en port map ( count2, lat2\_count2,l2\_en) ;

A7: decoder\_bcd\_7seg port map ( lat1\_count1, display1 );

A8: decoder\_bcd\_7seg port map ( lat1\_count2, display2 );

A9: summer port map ( lat1\_count1, lat1\_count2, sum1) ;

A10: summer port map ( lat2\_count1, lat2\_count2, sum2) ;

A11: comparator port map ( sum1, sum2, eq) ;

A12: debounce port map ( clk, button, switch) ;

A13: dicegame port map (

sw => switch,

eq => eq,

clk => clk ,

output(3 downto 0) => led(3 downto 0),

output(4) => l1\_en,

output(5) => l2\_en,

output(6) => c1\_en,

output(7) => c2\_en );

end Behavioral;

entity dicegame is

port (

sw : in std\_logic ;

eq : in std\_logic\_vector(1 downto 0) ;

clk : in std\_logic ;

output: out std\_logic\_vector(7 downto 0)

) ;

end dicegame;

architecture Behavioral of dicegame is

type state\_type is (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11);

signal state, next\_state : state\_type ;

begin

SYNC\_PROC : process(clk)

begin

if rising\_edge(clk) then

state <= next\_state ;

else

state <= state ;

end if ;

end process ;

NEXT\_STATE\_DECODE : process(state,sw,eq)

begin

next\_state <= S0 ;

case(state) is

when S0 =>

if(sw = '0' ) then

next\_state <= S1 ;

else

next\_state <= S0 ;

end if ;

when S1 =>

if(sw = '1') then

next\_state <= S2 ;

else

next\_state <= S1 ;

end if ;

when S2 =>

next\_state <= S3 ;

when S3 =>

if(eq = "00" ) then

next\_state <= S4 ;

elsif(eq = "01") then

next\_state <= S5 ;

else

next\_state <= S6 ;

end if ;

when S4 =>

if(sw ='0') then

next\_state <= S0 ;

else

next\_state <= S4 ;

end if ;

when S5 =>

if(sw ='0') then

next\_state <= S0 ;

else

next\_state <= S5 ;

end if ;

when S6 =>

if(sw ='0') then

next\_state <= S7 ;

else

next\_state <= S6 ;

end if ;

when S7 =>

if(sw ='0') then

next\_state <= S7 ;

else

next\_state <= S8 ;

end if ;

when S8 =>

next\_state <= S9 ;

when S9 =>

if(eq ="00") then

next\_state <= S10 ;

elsif( eq = "10") then

next\_state <= S11 ;

else

next\_state <= S6 ;

end if ;

when S10 =>

if( sw ='0') then

next\_state <= S0 ;

else

next\_state <= S10 ;

end if ;

when S11 =>

if( sw ='1') then

next\_state <= S11 ;

else

next\_state <= S0 ;

end if ;

when others =>

next\_state <= S0 ;

end case ;

end process ;

OUTPUT\_DECODE : process (state,sw,eq)

begin

case(state) is

when S0 =>

output <= X"88" ;

when S1 =>

output <= X"C0" ;

when S2 =>

output <= X"00" ;

when S3 =>

output <= X"30" ;

when S4 =>

output <= X"8A" ;

when S5=>

output <= X"89" ;

when S6 =>

output <= X"84" ;

when S7 =>

output <= X"C0" ;

when S8 =>

output <= X"00" ;

when S9 =>

output <= X"10" ;

when S10 =>

output <= X"89" ;

when S11 =>

output <= X"8A" ;

when others =>

output <= X"88" ;

end case ;

end process ;

end Behavioral;

entity comparator is

port(

sum1 : in std\_logic\_vector(3 downto 0 ) ;

sum2 : in std\_logic\_vector( 3 downto 0 ) ;

eq : out std\_logic\_vector( 1 downto 0 ) );

end comparator;

architecture Behavioral of comparator is

begin

eq <= "00" when sum1 = "0100" or sum1 = "1000" or sum1 = "1100" else

"01" when sum1 = "0011" or sum1 = "0110" or sum1 = "1001" else

"10" when sum1 = sum2 else

"11" ;

end Behavioral;

entity summer is

port(

ch1: in std\_logic\_vector(3 downto 0 ) ;

ch2 : in std\_logic\_vector( 3 downto 0 ) ;

sum : out std\_logic\_vector( 3 downto 0 ) );

end summer;

architecture Behavioral of summer is

begin

sum <= ch1+ ch2 ;

end Behavioral;

### TESTBENCH:

ENTITY dicegame\_tb IS

END dicegame\_tb;

ARCHITECTURE behavior OF dicegame\_tb IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT datapath

PORT(

button : IN std\_logic;

led : OUT std\_logic\_vector(3 downto 0);

clk : IN std\_logic;

display1 : OUT std\_logic\_vector(6 downto 0);

display2 : OUT std\_logic\_vector(6 downto 0)

);

END COMPONENT;

--Inputs

signal button : std\_logic := '0';

signal clk : std\_logic := '0';

--Outputs

signal led : std\_logic\_vector(3 downto 0);

signal display1 : std\_logic\_vector(6 downto 0);

signal display2 : std\_logic\_vector(6 downto 0);

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: datapath PORT MAP (

button => button,

led => led,

clk => clk,

display1 => display1,

display2 => display2

);

-- Stimulus process

stim\_proc: process

begin

-- hold reset state for 100 ns.

wait for 1 ns;

clk <= not clk ;

end process;

B : process

begin

wait for 56 ns ;

button <= '1' ;

wait for 93 ns ;

button <= '0' ;

wait for 97 ns ;

button <= '1' ;

wait for 91 ns ;

button <= '0';

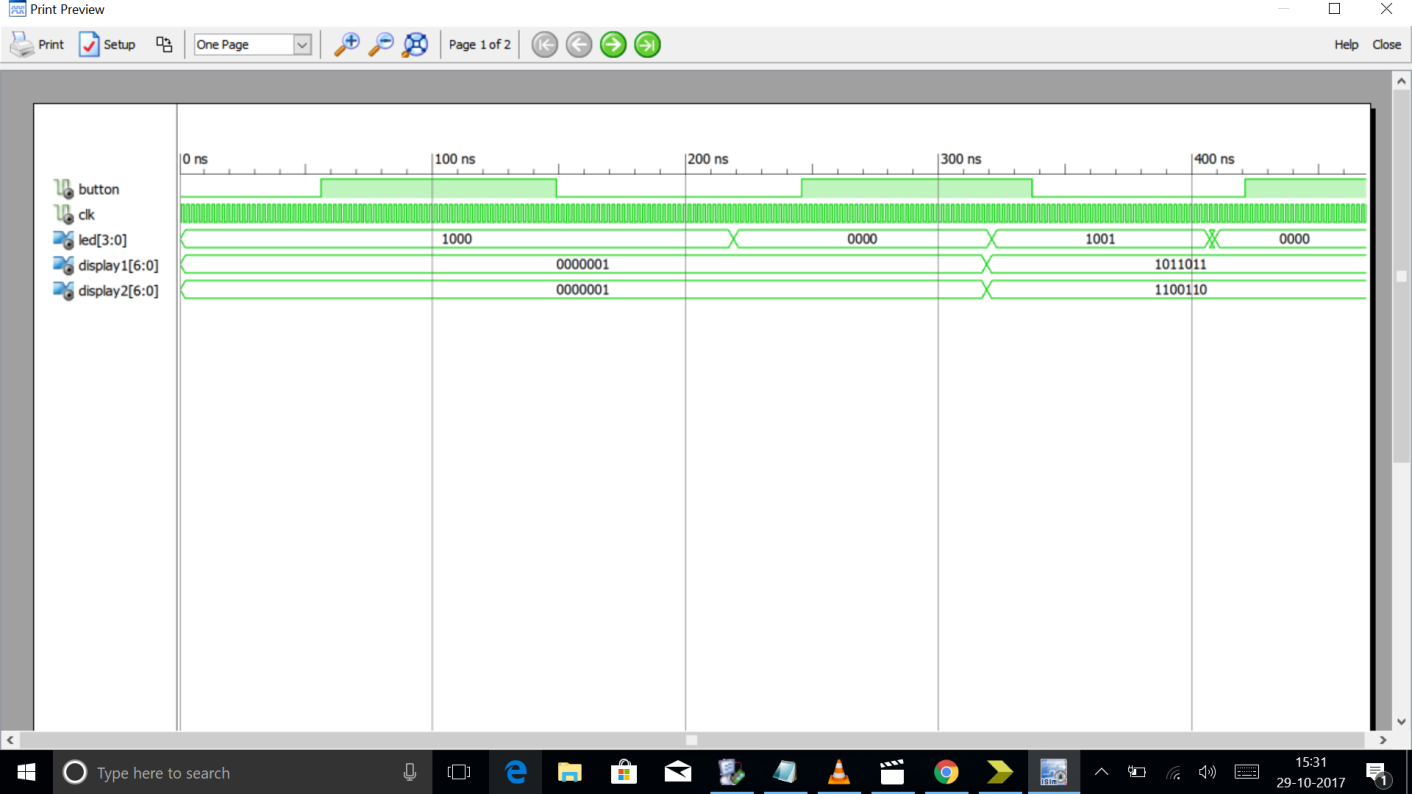
wait for 84 ns ;

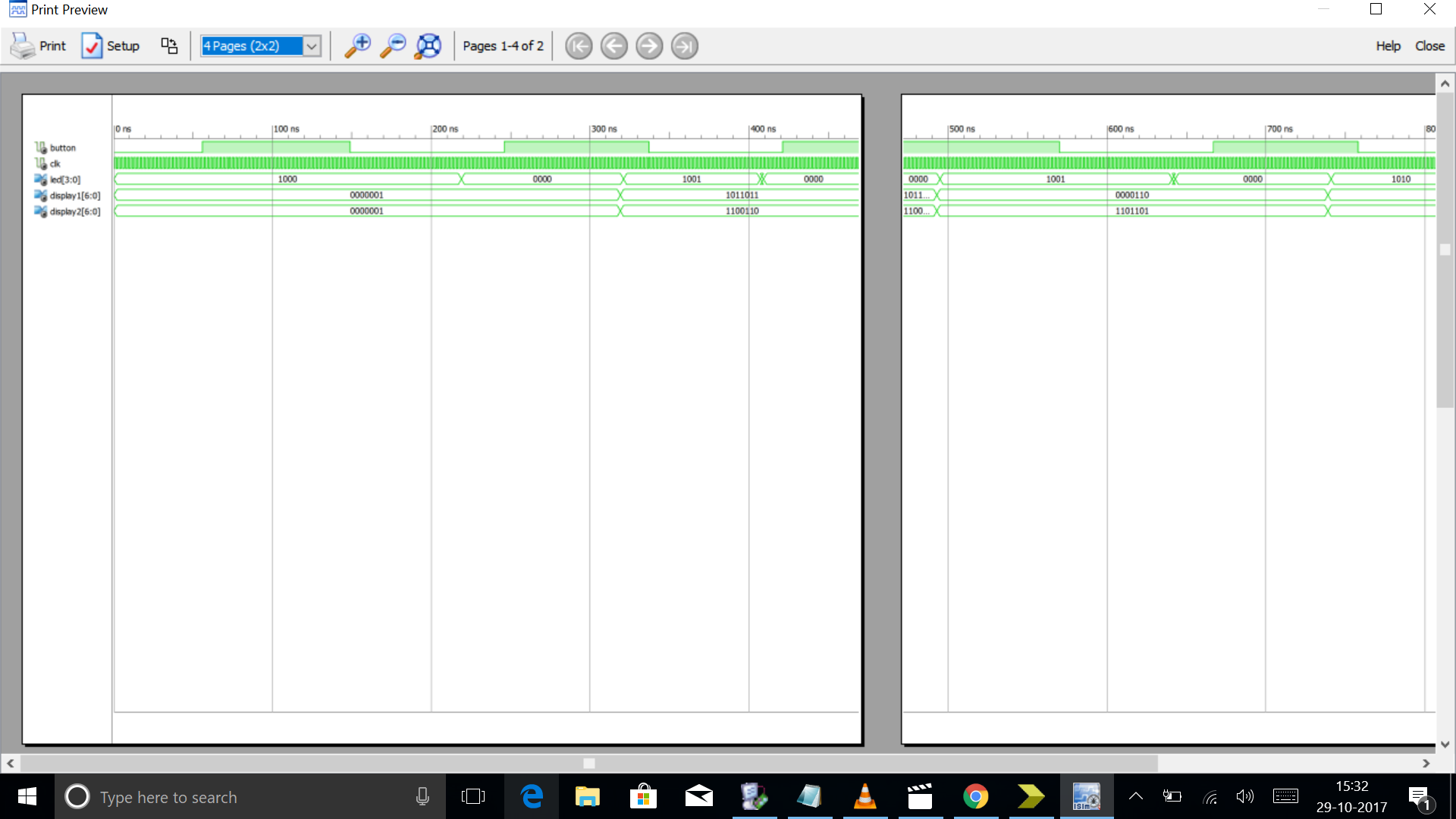
button <= '1' ;

end process ;

END;

## OUTPUT:





UART TX

-- This file contains the UART Transmitter. This transmitter is able

-- to transmit 8 bits of serial data, one start bit, one stop bit,

-- and no parity bit. When transmit is complete o\_TX\_Done will be

-- driven high for one clock cycle.

--

-- Set Generic g\_CLKS\_PER\_BIT as follows:

-- g\_CLKS\_PER\_BIT = (Frequency of i\_Clk)/(Frequency of UART)

-- Example: 10 MHz Clock, 115200 baud UART

-- (10000000)/(115200) = 87

--

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity UART\_TX is

generic (

g\_CLKS\_PER\_BIT : integer := 115 -- Needs to be set correctly

);

port (

i\_Clk : in std\_logic;

i\_TX\_DV : in std\_logic;

i\_TX\_Byte : in std\_logic\_vector(7 downto 0);

o\_TX\_Active : out std\_logic;

o\_TX\_Serial : out std\_logic;

o\_TX\_Done : out std\_logic

);

end UART\_TX;

architecture RTL of UART\_TX is

type t\_SM\_Main is (s\_Idle, s\_TX\_Start\_Bit, s\_TX\_Data\_Bits,

s\_TX\_Stop\_Bit, s\_Cleanup);

signal r\_SM\_Main : t\_SM\_Main := s\_Idle;

signal r\_Clk\_Count : integer range 0 to g\_CLKS\_PER\_BIT-1 := 0;

signal r\_Bit\_Index : integer range 0 to 7 := 0; -- 8 Bits Total

signal r\_TX\_Data : std\_logic\_vector(7 downto 0) := (others => '0');

signal r\_TX\_Done : std\_logic := '0';

begin

p\_UART\_TX : process (i\_Clk)

begin

if rising\_edge(i\_Clk) then

case r\_SM\_Main is

when s\_Idle =>

o\_TX\_Active <= '0';

o\_TX\_Serial <= '1'; -- Drive Line High for Idle

r\_TX\_Done <= '0';

r\_Clk\_Count <= 0;

r\_Bit\_Index <= 0;

if i\_TX\_DV = '1' then

r\_TX\_Data <= i\_TX\_Byte;

r\_SM\_Main <= s\_TX\_Start\_Bit;

else

r\_SM\_Main <= s\_Idle;

end if;

-- Send out Start Bit. Start bit = 0

when s\_TX\_Start\_Bit =>

o\_TX\_Active <= '1';

o\_TX\_Serial <= '0';

-- Wait g\_CLKS\_PER\_BIT-1 clock cycles for start bit to finish

if r\_Clk\_Count < g\_CLKS\_PER\_BIT-1 then

r\_Clk\_Count <= r\_Clk\_Count + 1;

r\_SM\_Main <= s\_TX\_Start\_Bit;

else

r\_Clk\_Count <= 0;

r\_SM\_Main <= s\_TX\_Data\_Bits;

end if;

-- Wait g\_CLKS\_PER\_BIT-1 clock cycles for data bits to finish

when s\_TX\_Data\_Bits =>

o\_TX\_Serial <= r\_TX\_Data(r\_Bit\_Index);

if r\_Clk\_Count < g\_CLKS\_PER\_BIT-1 then

r\_Clk\_Count <= r\_Clk\_Count + 1;

r\_SM\_Main <= s\_TX\_Data\_Bits;

else

r\_Clk\_Count <= 0;

-- Check if we have sent out all bits

if r\_Bit\_Index < 7 then

r\_Bit\_Index <= r\_Bit\_Index + 1;

r\_SM\_Main <= s\_TX\_Data\_Bits;

else

r\_Bit\_Index <= 0;

r\_SM\_Main <= s\_TX\_Stop\_Bit;

end if;

end if;

-- Send out Stop bit. Stop bit = 1

when s\_TX\_Stop\_Bit =>

o\_TX\_Serial <= '1';

-- Wait g\_CLKS\_PER\_BIT-1 clock cycles for Stop bit to finish

if r\_Clk\_Count < g\_CLKS\_PER\_BIT-1 then

r\_Clk\_Count <= r\_Clk\_Count + 1;

r\_SM\_Main <= s\_TX\_Stop\_Bit;

else

r\_TX\_Done <= '1';

r\_Clk\_Count <= 0;

r\_SM\_Main <= s\_Cleanup;

end if;

-- Stay here 1 clock

when s\_Cleanup =>

o\_TX\_Active <= '0';

r\_TX\_Done <= '1';

r\_SM\_Main <= s\_Idle;

when others =>

r\_SM\_Main <= s\_Idle;

end case;

end if;

end process p\_UART\_TX;

o\_TX\_Done <= r\_TX\_Done;

end RTL;

UART RX

-- This file contains the UART Receiver. This receiver is able to

-- receive 8 bits of serial data, one start bit, one stop bit,

-- and no parity bit. When receive is complete o\_rx\_dv will be

-- driven high for one clock cycle.

--

-- Set Generic g\_CLKS\_PER\_BIT as follows:

-- g\_CLKS\_PER\_BIT = (Frequency of i\_Clk)/(Frequency of UART)

-- Example: 10 MHz Clock, 115200 baud UART

-- (10000000)/(115200) = 87

--

library ieee;

use ieee.std\_logic\_1164.ALL;

use ieee.numeric\_std.all;

entity UART\_RX is

generic (

g\_CLKS\_PER\_BIT : integer := 115 -- Needs to be set correctly

);

port (

i\_Clk : in std\_logic;

i\_RX\_Serial : in std\_logic;

o\_RX\_DV : out std\_logic;

o\_RX\_Byte : out std\_logic\_vector(7 downto 0)

);

end UART\_RX;

architecture rtl of UART\_RX is

type t\_SM\_Main is (s\_Idle, s\_RX\_Start\_Bit, s\_RX\_Data\_Bits,

s\_RX\_Stop\_Bit, s\_Cleanup);

signal r\_SM\_Main : t\_SM\_Main := s\_Idle;

signal r\_RX\_Data\_R : std\_logic := '0';

signal r\_RX\_Data : std\_logic := '0';

signal r\_Clk\_Count : integer range 0 to g\_CLKS\_PER\_BIT-1 := 0;

signal r\_Bit\_Index : integer range 0 to 7 := 0; -- 8 Bits Total

signal r\_RX\_Byte : std\_logic\_vector(7 downto 0) := (others => '0');

signal r\_RX\_DV : std\_logic := '0';

begin

-- Purpose: Double-register the incoming data.

-- This allows it to be used in the UART RX Clock Domain.

-- (It removes problems caused by metastabiliy)

p\_SAMPLE : process (i\_Clk)

begin

if rising\_edge(i\_Clk) then

r\_RX\_Data\_R <= i\_RX\_Serial;

r\_RX\_Data <= r\_RX\_Data\_R;

end if;

end process p\_SAMPLE;

-- Purpose: Control RX state machine

p\_UART\_RX : process (i\_Clk)

begin

if rising\_edge(i\_Clk) then

case r\_SM\_Main is

when s\_Idle =>

r\_RX\_DV <= '0';

r\_Clk\_Count <= 0;

r\_Bit\_Index <= 0;

if r\_RX\_Data = '0' then -- Start bit detected

r\_SM\_Main <= s\_RX\_Start\_Bit;

else

r\_SM\_Main <= s\_Idle;

end if;

-- Check middle of start bit to make sure it's still low

when s\_RX\_Start\_Bit =>

if r\_Clk\_Count = (g\_CLKS\_PER\_BIT-1)/2 then

if r\_RX\_Data = '0' then

r\_Clk\_Count <= 0; -- reset counter since we found the middle

r\_SM\_Main <= s\_RX\_Data\_Bits;

else

r\_SM\_Main <= s\_Idle;

end if;

else

r\_Clk\_Count <= r\_Clk\_Count + 1;

r\_SM\_Main <= s\_RX\_Start\_Bit;

end if;

-- Wait g\_CLKS\_PER\_BIT-1 clock cycles to sample serial data

when s\_RX\_Data\_Bits =>

if r\_Clk\_Count < g\_CLKS\_PER\_BIT-1 then

r\_Clk\_Count <= r\_Clk\_Count + 1;

r\_SM\_Main <= s\_RX\_Data\_Bits;

else

r\_Clk\_Count <= 0;

r\_RX\_Byte(r\_Bit\_Index) <= r\_RX\_Data;

-- Check if we have sent out all bits

if r\_Bit\_Index < 7 then

r\_Bit\_Index <= r\_Bit\_Index + 1;

r\_SM\_Main <= s\_RX\_Data\_Bits;

else

r\_Bit\_Index <= 0;

r\_SM\_Main <= s\_RX\_Stop\_Bit;

end if;

end if;

-- Receive Stop bit. Stop bit = 1

when s\_RX\_Stop\_Bit =>

-- Wait g\_CLKS\_PER\_BIT-1 clock cycles for Stop bit to finish

if r\_Clk\_Count < g\_CLKS\_PER\_BIT-1 then

r\_Clk\_Count <= r\_Clk\_Count + 1;

r\_SM\_Main <= s\_RX\_Stop\_Bit;

else

r\_RX\_DV <= '1';

r\_Clk\_Count <= 0;

r\_SM\_Main <= s\_Cleanup;

end if;

-- Stay here 1 clock

when s\_Cleanup =>

r\_SM\_Main <= s\_Idle;

r\_RX\_DV <= '0';

when others =>

r\_SM\_Main <= s\_Idle;

end case;

end if;

end process p\_UART\_RX;

o\_RX\_DV <= r\_RX\_DV;

o\_RX\_Byte <= r\_RX\_Byte;

end rtl;

# TESTBENCH

library ieee;

use ieee.std\_logic\_1164.ALL;

use ieee.numeric\_std.all;

entity uart\_tb is

end uart\_tb;

architecture behave of uart\_tb is

component uart\_tx is

generic (

g\_CLKS\_PER\_BIT : integer := 115 -- Needs to be set correctly

);

port (

i\_clk : in std\_logic;

i\_tx\_dv : in std\_logic;

i\_tx\_byte : in std\_logic\_vector(7 downto 0);

o\_tx\_active : out std\_logic;

o\_tx\_serial : out std\_logic;

o\_tx\_done : out std\_logic

);

end component uart\_tx;

component uart\_rx is

generic (

g\_CLKS\_PER\_BIT : integer := 115 -- Needs to be set correctly

);

port (

i\_clk : in std\_logic;

i\_rx\_serial : in std\_logic;

o\_rx\_dv : out std\_logic;

o\_rx\_byte : out std\_logic\_vector(7 downto 0)

);

end component uart\_rx;

-- Test Bench uses a 10 MHz Clock

-- Want to interface to 115200 baud UART

-- 10000000 / 115200 = 87 Clocks Per Bit.

constant c\_CLKS\_PER\_BIT : integer := 87;

constant c\_BIT\_PERIOD : time := 8680 ns;

signal r\_CLOCK : std\_logic := '0';

signal r\_TX\_DV : std\_logic := '0';

signal r\_TX\_BYTE : std\_logic\_vector(7 downto 0) := (others => '0');

signal w\_TX\_SERIAL : std\_logic;

signal w\_TX\_DONE : std\_logic;

signal w\_RX\_DV : std\_logic;

signal w\_RX\_BYTE : std\_logic\_vector(7 downto 0);

signal r\_RX\_SERIAL : std\_logic := '1';

-- Low-level byte-write

procedure UART\_WRITE\_BYTE (

i\_data\_in : in std\_logic\_vector(7 downto 0);

signal o\_serial : out std\_logic) is

begin

-- Send Start Bit

o\_serial <= '0';

wait for c\_BIT\_PERIOD;

-- Send Data Byte

for ii in 0 to 7 loop

o\_serial <= i\_data\_in(ii);

wait for c\_BIT\_PERIOD;

end loop; -- ii

-- Send Stop Bit

o\_serial <= '1';

wait for c\_BIT\_PERIOD;

end UART\_WRITE\_BYTE;

begin

-- Instantiate UART transmitter

UART\_TX\_INST : uart\_tx

generic map (

g\_CLKS\_PER\_BIT => c\_CLKS\_PER\_BIT

)

port map (

i\_clk => r\_CLOCK,

i\_tx\_dv => r\_TX\_DV,

i\_tx\_byte => r\_TX\_BYTE,

o\_tx\_active => open,

o\_tx\_serial => w\_TX\_SERIAL,

o\_tx\_done => w\_TX\_DONE

);

-- Instantiate UART Receiver

UART\_RX\_INST : uart\_rx

generic map (

g\_CLKS\_PER\_BIT => c\_CLKS\_PER\_BIT

)

port map (

i\_clk => r\_CLOCK,

i\_rx\_serial => r\_RX\_SERIAL,

o\_rx\_dv => w\_RX\_DV,

o\_rx\_byte => w\_RX\_BYTE

);

r\_CLOCK <= not r\_CLOCK after 50 ns;

process is

begin

-- Tell the UART to send a command.

wait until rising\_edge(r\_CLOCK);

wait until rising\_edge(r\_CLOCK);

r\_TX\_DV <= '1';

r\_TX\_BYTE <= X"AB";

wait until rising\_edge(r\_CLOCK);

r\_TX\_DV <= '0';

wait until w\_TX\_DONE = '1';

-- Send a command to the UART

wait until rising\_edge(r\_CLOCK);

UART\_WRITE\_BYTE(X"3F", r\_RX\_SERIAL);

wait until rising\_edge(r\_CLOCK);

-- Check that the correct command was received

if w\_RX\_BYTE = X"3F" then

report "Test Passed - Correct Byte Received" severity note;

else

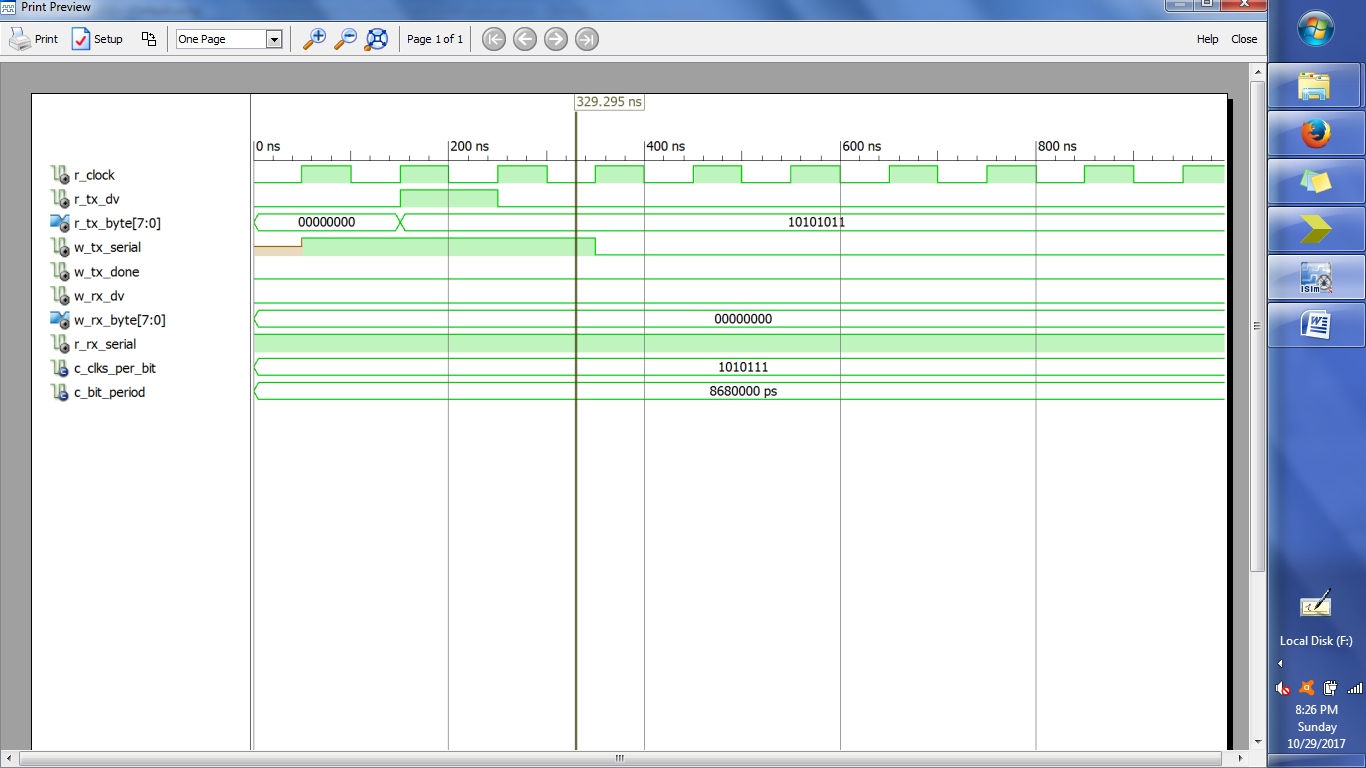
report "Test Failed - Incorrect Byte Received" severity note;

end if;

assert false report "Tests Complete" severity failure;

end process;

end behave;



KEYBOARD SCANNER

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity scanner is

port (R0, R1,R2, R3, CLK: in std\_logic ;

C0,A1,C2: inout std\_logic;

N0,N1,N2,N3, V: out std\_logic) ;

end scanner;

architecture behaviour of scanner is

signal QA, K,Kd:std\_logic;

signal state ,nextstate: integer range 0 to 5;

signal temp1,temp2:std\_logic:='0';

begin

K<=(R0 or R1 or R3); --this is the decoder section

N3 <=((R2 and (not(C0))) or (R3 and (not(C1))));

N2 <= (R1 or R2)and C0;

temp1<=not(R1)and(not(R0)and C0);

temp2<=(not(R2)and C2)or temp1;

N1 <= (R0 and (not (C0)))or temp2;

N0 <= ((R1 and C1) or ((not R1) and C2) or ((not R3) and (not R1) and (not C1)));

process (state , R0,R1,R2,R3,C0,C1,C2,K,Kd,QA)

begin

C0 <= '0' ; C1 <= '0' ; C2 <= '0' ; V <= '0';

case state is

when 0 => nextstate <= 1;

when 1 => C0 <= '1' ; C1 <='1' C2 <='1' ;

if ((Kd and K) ='1' )then

nextstate <=2;

else nextstate <=1;

end if ;

when 2 => C0<= '1'

if (Kd and K) ='1' then V <='1' nextstate<=5;

elsif K = '0' then nextstate <= 3;

else nextstate <=2;

end if ;

when 3 => C1 <= '1';

if (Kd and K) = '1' then V <= '1' next state <=5;

elsif K ='0' then nextstate <=4;

else nextstate <=3;

end if;

when 4 => C2 ,='1'

if (Kd and K) ='1' then V <='1'; nextstate <= 5;

else nextstate <=4;

end if;

when 5 => C0<= '1'; C1 <= '1'; C2 <= '1';

if Kd = '0' then nextstate <= 1;

else nextstate <=5;

end if ;

end case;

end process;

process (CLK)

begin

if (CLk = '1' and ClK'EVENT) then

state <= nextstate;

QA <= K;

Kd <= QA;

end if;

end process;

end behaviour;

OUTPUT:

