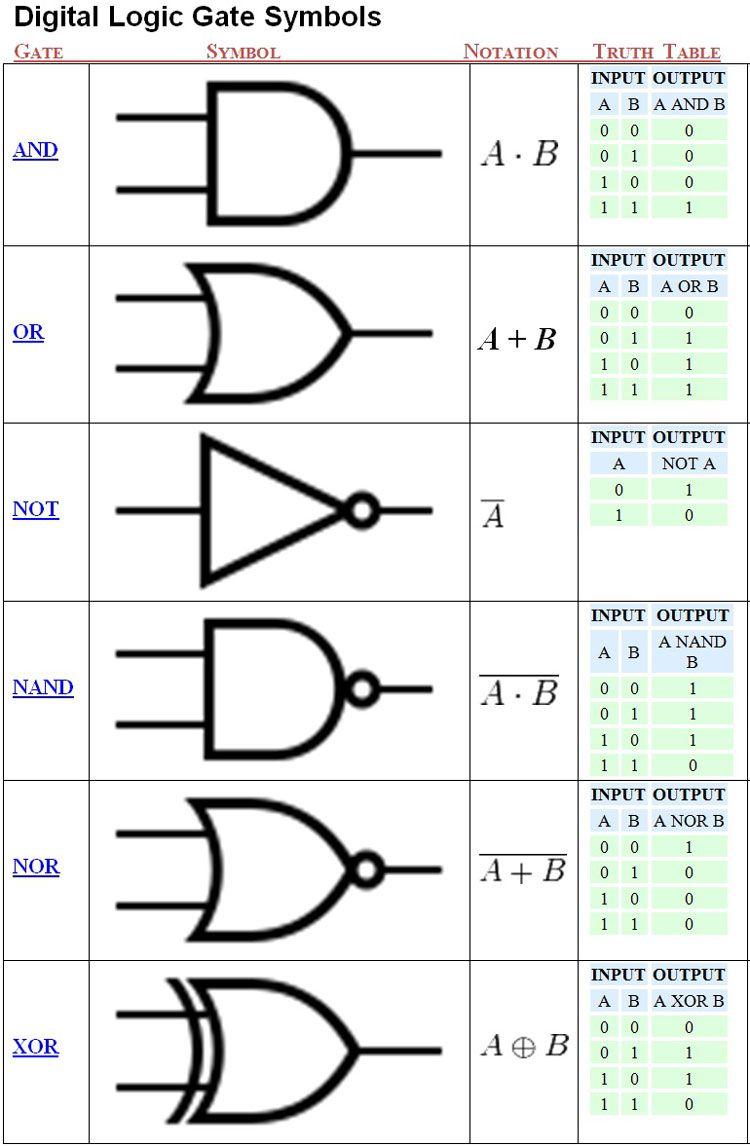
CPE 166 Midterm 2 Review

Picture of Logic Gates

VHDL Syntax 

Assign: <=

Compare: =

Numbers: need to be in ‘’ or “”

Logical Operators

And: c <= a and b;

Or: c <= a or b;

Nand: c < a nand b;

Nor: c < a nor b;

Xor: c <= a xor b;

Xnor: c <= a xnor b;

Not: c <= not c;

Relational Operators

= Equal

/= Not Equal

< Less Than

<= Less Than or Equal To

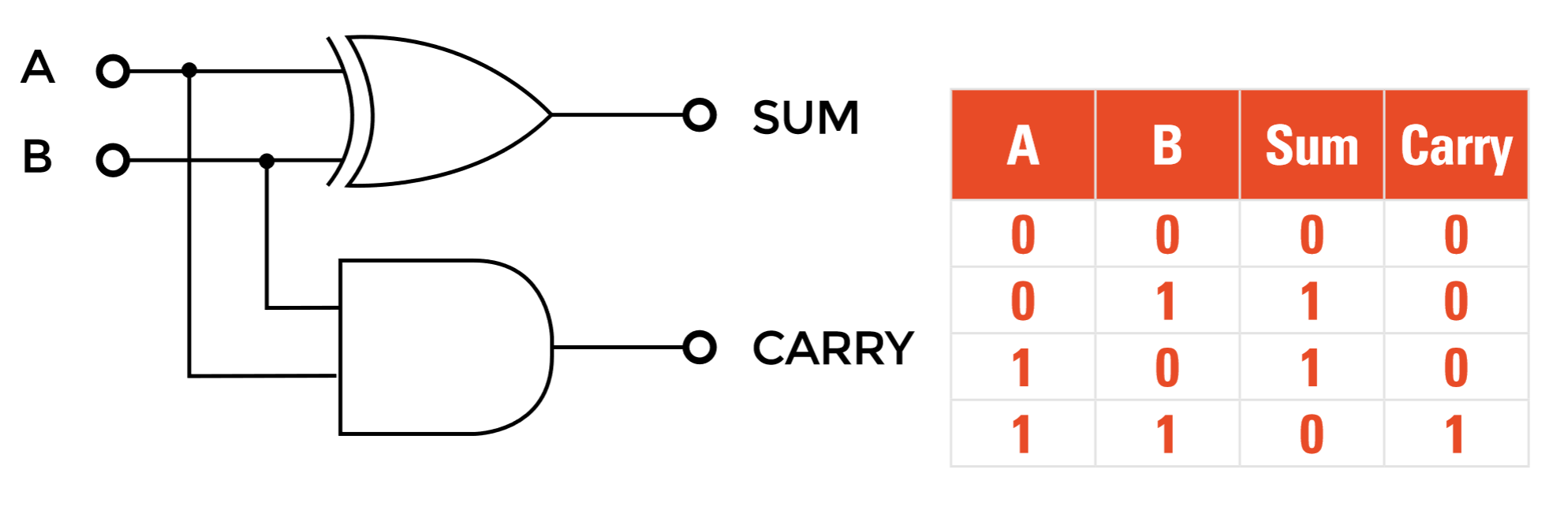
> Greater Than

>= Greater Than or Equal To

Combinational Logic Designs in VHDL

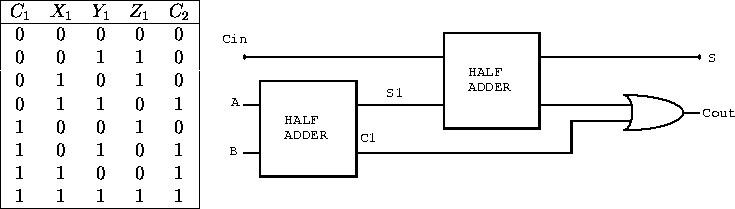
Half adder

Adder without carry in, sum is 1 when there is a single 1, 0 if two 0’s or 2 1’s. When there are two ones, carry out. Therefore, xor & and gate for HA



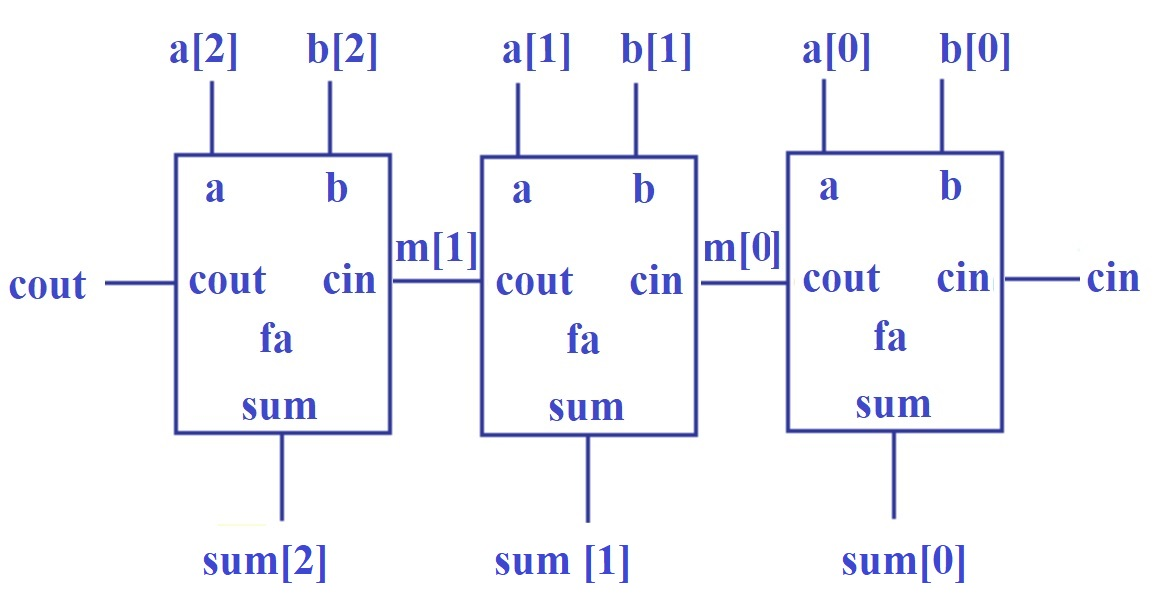
|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; use ieee.numeric\_std.all;   entity half\_adder is  port (  A, B: in std\_logic;  --  o\_sum : out std\_logic;  o\_carry : out std\_logic  ); end half\_adder;   architecture rtl of half\_adder is begin  o\_sum <= A xor B;  o\_carry <= A and B; end rtl; |

Full Bit Adder

  
Adds carry in which changes the combinational logic  
Two Half adders create a full adder

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; use ieee.numeric\_std.all;   entity full\_adder is  port (  A,B : in std\_logic;  i\_carry : in std\_logic;  --  o\_sum : out std\_logic;  o\_carry : out std\_logic  ); end full\_adder;     architecture rtl of full\_adder is  signal w\_WIRE\_1 : std\_logic;  signal w\_WIRE\_2 : std\_logic;  signal w\_WIRE\_3 : std\_logic; begin  w\_WIRE\_1 <= A xor B;  w\_WIRE\_2 <= w\_WIRE\_1 and i\_carry;  w\_WIRE\_3 <= A and B;  o\_sum <= w\_WIRE\_1 xor i\_carry;  o\_carry <= w\_WIRE\_2 or w\_WIRE\_3; end rtl; |

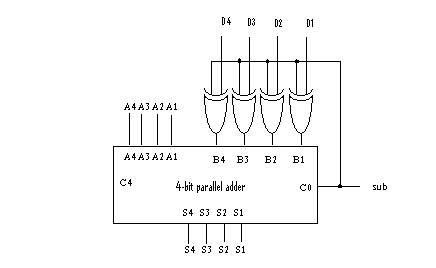
Ripple Carry Adder comprises of Full Adders that go into each other



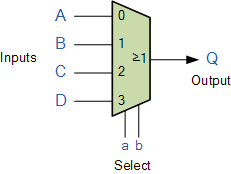
|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;   entity ripple\_carry\_adder is  generic (  g\_WIDTH : natural := 2  );  port (  i\_add\_term1 : in std\_logic\_vector(g\_WIDTH-1 downto 0);  i\_add\_term2 : in std\_logic\_vector(g\_WIDTH-1 downto 0);  --  o\_result : out std\_logic\_vector(g\_WIDTH downto 0)  ); end ripple\_carry\_adder;     architecture rtl of ripple\_carry\_adder is    component full\_adder is  port (  A,B : in std\_logic;  i\_carry : in std\_logic;  o\_sum, o\_carry : out std\_logic;  end component full\_adder;  signal w\_CARRY : std\_logic\_vector(g\_WIDTH downto 0);  signal w\_SUM : std\_logic\_vector(g\_WIDTH-1 downto 0); begin  w\_CARRY(0) <= '0';   -- no carry input on first full adder    SET\_WIDTH : for ii in 0 to g\_WIDTH-1 generate  i\_FULL\_ADDER\_INST : full\_adder  port map (  A => i\_add\_term1(ii),  B => i\_add\_term2(ii),  i\_carry => w\_CARRY(ii),  o\_sum => w\_SUM(ii),  o\_carry => w\_CARRY(ii+1)  );  end generate SET\_WIDTH;    o\_result <= w\_CARRY(g\_WIDTH) & w\_SUM; -- VHDL Concatenation   end rtl; |

subtractor

Subtractor is just an adder but with B xor’d



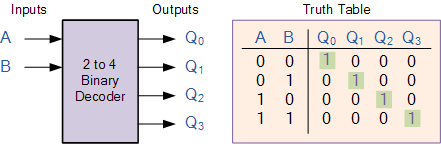
1-b. Multiplexer



Device that selects different inputs and has a single output. Code:

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; entity mux is  port (  a, b, c, d : in std\_logic;  s : in std\_logic\_vector (1 downto 0);  o : out std\_logic  ); end mux;  architecture archi of mux is  begin  process (a, b, c, d, s)  begin  case s is  when "00" => o <= a;  when "01" => o <= b;  when "10" => o <= c;  when others => o <= d;  end case; end process; end archi; |

Decoder



Decoder takes n input and expands to 2^n output. 2to4: 00= 0000, 01 = 0010, 10 = 0100, 11 - 1000 Code:

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; entity dec is port ( sel: in std\_logic\_vector (2 downto 0); res: out std\_logic\_vector (7 downto 0) ); end dec; architecture archi of dec is begin with sel select res <= "00000001" when "000", "00000010" when "001", "00000100" when "010", "00001000" when "011", "00010000" when "100", "00100000" when "101", "01000000" when "110" "10000000" when others; end archi; |

Comparators

VHDL: (=, /=,<, <=, >, >=)

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_unsigned.all; entity compar is  port(  A, B : in std\_logic\_vector(7 downto 0);  CMP : out std\_logic  ); end compar; architecture archi of compar is  begin  CMP <= '1' when A >= B else '0'; end archi; |

Sequential circuits using VHDL

D Flip-flops

Flip-flop with Positive-Edge Clock and Synchronous Set

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; entity flop is  port(  C, CE, PRE : in std\_logic;  D: in std\_logic\_vector(3 downto 0);  Q : out std\_logic\_vector (3 downto 0)  ); end flop;  architecture archi of flop is  begin  process (C, PRE)  begin  if (PRE='1') then  Q <= "1111";  elsif (C'event and C='1')then  if (CE='1') then  Q <= D;  end if;  end if; end process; end archi; |

Shift-registers

8-bit Shift-Left Register with Positive-Edge Clock, Serial In, and Serial Out

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_unsigned.all; use ieee.std\_logic\_arith.all;  entity shift is  port(  C, SI : in std\_logic;  SO : out std\_logic  ); end shift;  architecture archi of shift is signal tmp: std\_logic\_vector(7 downto 0);  begin  process (C)  begin  if (C'event and C='1') then  for i in 0 to 6 loop  tmp(i+1) <= tmp(i);  end loop;  tmp(0) <= SI;  end if;  end process;  SO <= tmp(7); end archi; |

Counters .4-bit up/down counter with an asynchronous clear

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_unsigned.all; use ieee.std\_logic\_arith.all; entity counter is port( C, CLR, UP\_DOWN : in std\_logic; Q: out std\_logic\_vector(3 downto 0) ); end counter; architecture archi of counter is signal tmp: std\_logic\_vector(3 downto 0); begin process (C, CLR) begin if (CLR='1') then tmp <= "0000"; elsif (C'event and C='1') then if (UP\_DOWN='1') then tmp <= tmp + 1; else tmp <= tmp - 1; end if; end if; end process; Q <= tmp; end archi; |

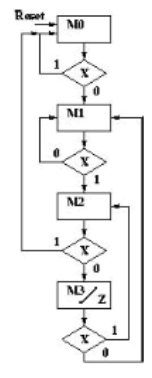
Clock-dividers

Create a new clock that is a division of the original clock. Code below divides by 100, if you want to change the amount divide, change counter. Counter is number you want to divide -1, elsif counter is half of the number you want to divide by -1

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;    entity module\_new\_clock is  port(  clk: in std\_logic;  reset: in std\_logic;  clkout: out std\_logic  );  end module\_new\_clock;    architecture beh of module\_new\_clock is  signal cnt: std\_logic;  signal counter: std\_logic\_vector(7 downto 0);  begin  clkout <= cnt;   process(clk,reset)   begin   if (reset='1') then   counter<=(others=>'0');   cnt<='0';  elsif( rising\_edge(clk) ) then   if (counter = 99 ) then   counter <= ( others => '0' );   cnt <= '1';   elsif (counter < 499) then   counter <= counter + 1;   cnt <= '1';   else   counter <= counter + 1;   cnt <= '0';   end if;   end if;   end process;  end beh; |

Be able to develop hierarchical design using VHDL

Finite State Machine

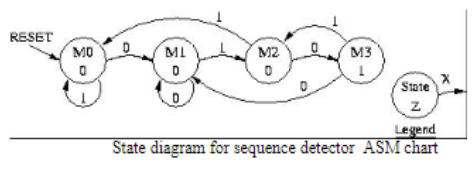
Be able to Draw ASM chart, as well as Finite state machine / ASM Chart  
simulation waveforms: Be able to draw current state, next state and also moore outputs, mealy outputs waveforms. 

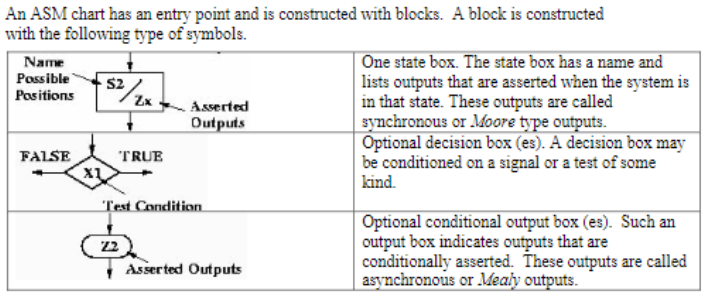
ASM Diagram Block

ASM Rules:

•The entrance paths to an ASM block lead to only one state box

•Of 'N' possible exit paths, for each possible valid input combination, only one exit path can be followed, that is there is only one valid next state.





Waveforms:

8. Be able to draw synthesized circuits with given VHDL codes.

Even parity and odd parity

Even parity is 1 when there is an odd number of 1s

Odd Parity is 1 when when there is an even number of 1s

Even Parity and Odd Parity Examples

ab parity (f\_even)

00\_0, 01\_1, 10\_1, 11\_0

f\_even <= a xor b;

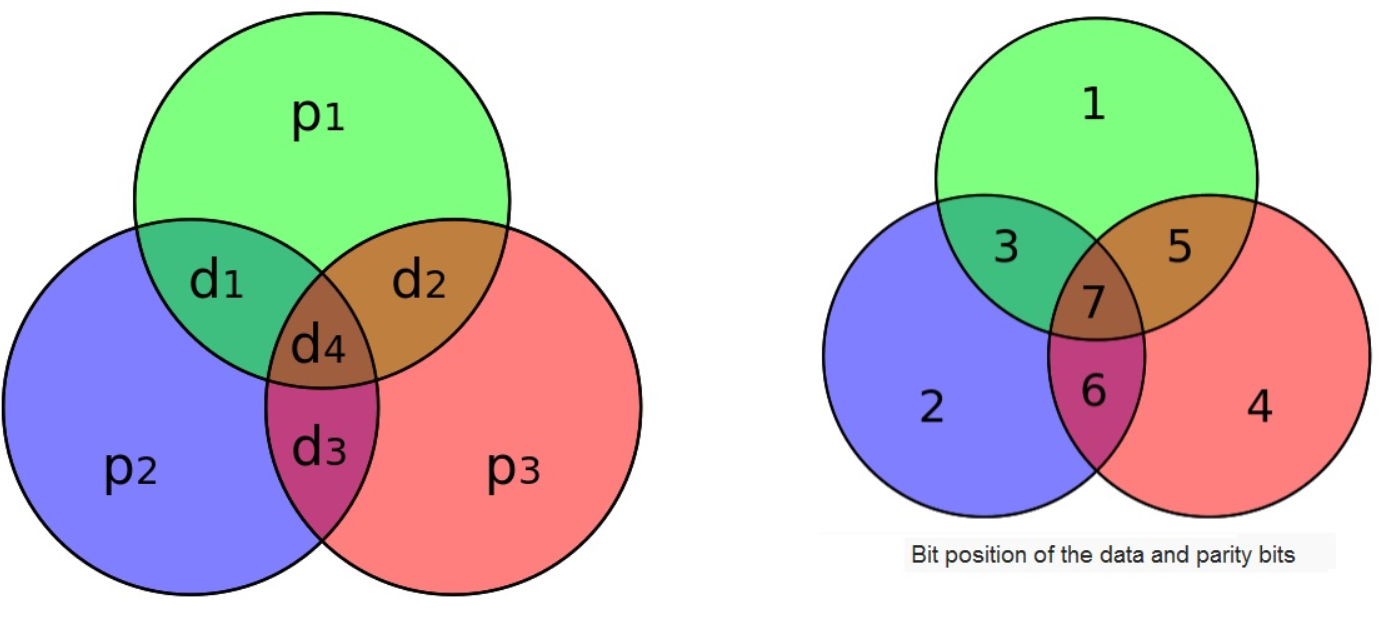
ab parity (f\_odd)

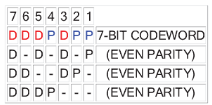
00\_1, 01\_0, 10\_0,11\_1

f\_odd <= not ( a xor b);

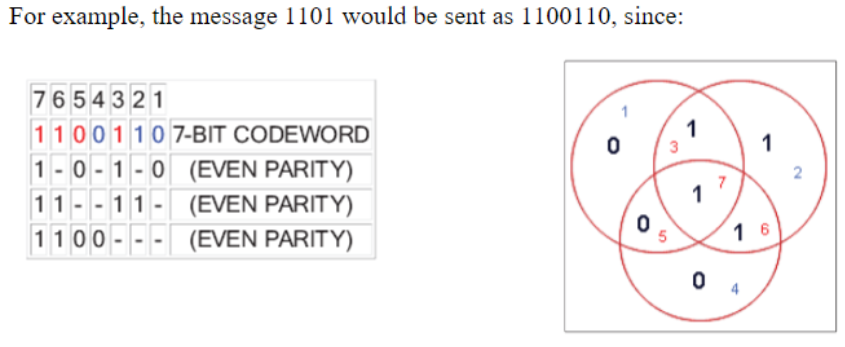
(7,4) Hamming codes

D is the input for hamming, P is parity bit





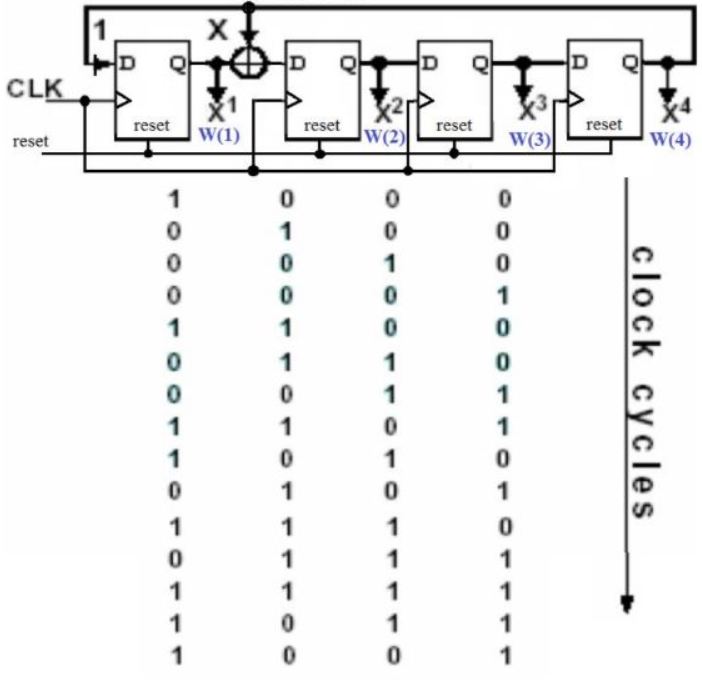
|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity module\_haming is  port(s0,s1,s2: in std\_logic; d: in std\_logic\_vector(4 downto 1);  h: out std\_logic\_vector(7 downto 1) );  signal p: std\_logic\_vector(7 downto 1);  end module\_haming;  architecture beh of module\_haming is  begin  process(d,s0,s1,s2)  begin   if (s0='1') then  h<=("0000000");  else p(1)<=d(4) xor d(2) xor d(1);  p(2)<=d(4) xor d(3) xor d(1);  p(3)<=d(1);  p(4)<=d(4) xor d(3) xor d(2);  p(5)<=d(2);  p(6)<=d(3);  p(7)<=d(4);  end if;  if (s2='1') then  h<=p;  elsif(s1='1') then  h<=('0'&'0'&'0'&d);  end if;  end process;  end beh; |



Can check for a single bit error by checking parity

LFSR

Linear Feedback Shift Register generates inputs, can be represented as 1+ x+x^4



|  |
| --- |
| LIBRARY IEEE; USE IEEE.STD\_LOGIC\_1164.ALL; ENTITY lfsr IS PORT ( reset, clk: IN STD\_LOGIC; Q : OUT STD\_LOGIC\_VECTOR(4 downto 1) ); END lfsr; ARCHITECTURE beh OF lfsr IS signal W: std\_logic\_vector(4 downto 1); BEGIN process( clk, reset ) begin if (reset='1') then W <= ( 1=>'1', others => '0' ); elsif (rising\_edge (clk)) then W <= W(3 downto 2) & ( W(1) xor W(4) ) & W(4); end if; end process; Q <= W; END beh; ------------------------------------ LIBRARY IEEE; USE IEEE.STD\_LOGIC\_1164.ALL;  -- if using more libraries add more here ENTITY lfsr\_tb is End lfsr\_tb;  ARCHITECTURE beh OF lfsr\_tb IS  -- all of the stuff needed to test goes into a component Component lfsr PORT (  reset, clk: IN STD\_LOGIC;  Q : OUT STD\_LOGIC\_VECTOR(4 downto 1) ); END Component;  signal reset, clk: std\_logic; signal Q: std\_logic\_vector(4 downto 1); BEGIN  -- Instantiate the Unit Under Test (UUT)  uut: lfsr port map(  reset=>reset,  clk=>clk,  Q=>Q  );  -- Clock process definitions Process Begin  Clk <= '0';  Wait for 10 ns;  Clk <= '1';  Wait for 10 ns; End process;  -- Stimulus process Process Begin  reset <='1';  Wait for 6 ns;  reset <='0';  Wait for 40 ns;  Wait; End process;  END beh; |

Be able to analyze LFSR outputs clock cycle by clock cycle.

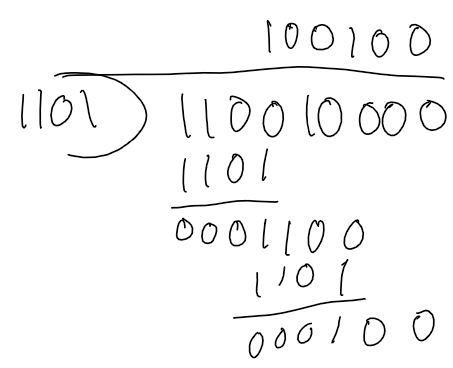
CRC Computation

message represented by some polynomial G(x), and a generating polynomial P(x)

1 - Multiply the message G(x) by x^3, where 3 is the number of bits in the CRC.Add 3 three zeros to the binary G(x).

2 - Divide the product x3 [G(x)] by the generating polynomial P(x). (when dividing, xor the two numbers)

We wish to find "the remainder, modulo P(x)"CRC Error

3 - Disregard the quotient and add the remainder C(x)to the product x^3 [G(x)] to yield the codemessage polynomial F(x), which is represented as:

F(x) = x3 [G(x)] + C(x)

ex) G(x) = 110010, P(x) = 1001(x^3 + x^2 + 1)

G(x)x3 = 110010000

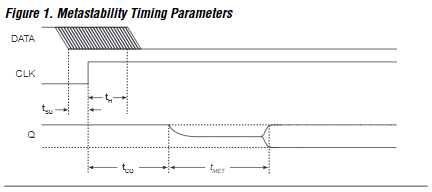
Cx = 100

CRC Detection

Upon reception, the entire received F(x) = "message + crc" can be checked simply by dividing F(x)/P(x) using the same generating polynomial. If the remainder after division equals zero, then no error was found.

Metastability

Metastability is when a system persists for an unbounded time in an unstable equilibrium or metastable state. When a digital signal is not 1 or 0 for an example, circuit can act in unpredictable ways.



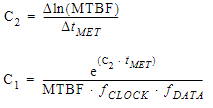
Tmet = settling time

Tco- clock to output delay(minimum output

transition time)

Th- minimum hold time

Tsu = minimum setup time

MTBF provides an estimate of the mean time between the probable occurrence of two successive metastable events. 

You can avoid metastability’s negative effects by using the output of the synchronizing flip flop rather than the asynchronous signal or by adding the Tmet calculated for a specific MTBF to the worst-case timing delay calculations, giving the output of the flipflops time to settle

Hazard

- A glitch is an unwanted pulse at the output of a combinational logic network – a momentary change in an output that should not have changed

-A circuit with the potential for a glitch is said to have a hazard.In other words a hazard is something intrinsic about a circuit; a circuit with hazard may or may not have a glitch depending on input patterns and the electric characteristics of the circuit

(static 1 hazard, static 0 hazard, dynamic hazard) 