CPE/EE 422/522
Advanced Logic Design
L05

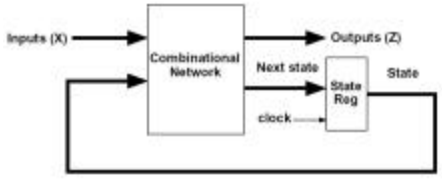
Electrical and Computer Engineering
University of Alabama in Huntsville

Outline

- What we know
  - Combinational Networks
  - Sequential Networks:
    - Basic Building Blocks, Mealy & Moore Machines, Max Frequency, Setup & Hold Times, Synchronous Design
- What we do not know
  - Equivalent states and reduction of state tables
  - Hardware Description Languages

Review: Mealy Sequential Networks

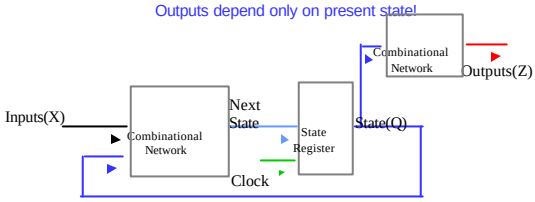
General model of Mealy Sequential Network



(1) X inputs are changed to a new value
(2) After a delay, the Z outputs and next state appear at the output of CM
(3) The next state is clocked into the state register and the state changes

Review: General Model of Moore Sequential Machine

Outputs depend only on present state!



$X = x_1\ x_2 \ldots\ x_n$
$Q = Q_1\ Q_2 \ldots\ Q_k$
$Z = z_1\ z_2 \ldots\ z_m$

$Q(t^+) = G(X(t), Q(t))$
$Z(t) = F(Q(t))$

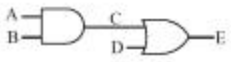
Intro to VHDL

- Technology trends
  - 1 billion transistor chip running at 20 GHz in 2007
- Need for Hardware Description Languages
  - Systems become more complex
  - Design at the gate and flip-flop level becomes very tedious and time consuming
- HDLs allow
  - Design and debugging at a higher level before conversion to the gate and flip-flop level
  - Tools for synthesis do the conversion
- VHDL, Verilog
- VHDL – VHSIC Hardware Description Language

Intro to VHDL

- Developed originally by DARPA
  - for specifying digital systems
- International IEEE standard (IEEE 1076-1993)
- Hardware Description, Simulation, Synthesis
- Provides a mechanism for digital design and reusable design documentation
- Support different description levels
  - Structural (specifying interconnections of the gates),
  - Dataflow (specifying logic equations), and
  - Behavioral (specifying behavior)
- Top-down, Technology Dependent

VHDL Description of Combinational Networks



Concurrent Statements

```
C <= A and B  after 5 ns;
E <= C or D   after 5 ns;
```

If delay is not specified, "delta" delay is assumed

```
C <= A and B;
E <= C or D;
```

Order of concurrent statements is not important

```
E <= C or D;
C <= A and B;
```

This statement executes repeatedly

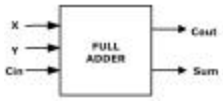
```
CLK <= not CLK after 10 ns;
```

This statement causes a simulation error

```
CLK <= not CLK;
```

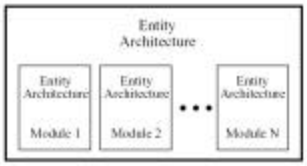

Entity-Architecture Pair

Full Adder Example



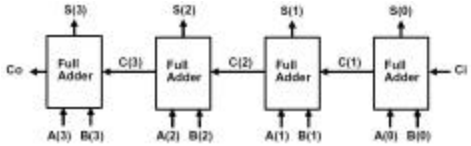

```
entity FullAdder is
  port (X, Y, Cin: in bit;    -- Inputs
    Cout, Sum: out bit);      -- Outputs
end FullAdder;

architecture Equations of FullAdder is
begin                         -- Concurrent Assignments
  Sum  <= X xor Y xor Cin after 10 ns;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
```


VHDL Program Structure




```
entity entity-name is
  [port(interface-signal-declaration);]
end [entity] [entity-name];

architecture architecture-name of entity-name is
  [declarations]
begin
  architecture body
end [architecture] [architecture-name];
```


4-bit Adder




```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;     -- Inputs
        S: out bit_vector(3 downto 0); Co: out bit);     -- Outputs
end Adder4;
```


4-bit Adder (cont'd)

```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;     -- Inputs
        S: out bit_vector(3 downto 0); Co: out bit);     -- Outputs
end Adder4;

architecture Structure of Adder4 is
component FullAdder
   port (X, Y, Cin: in bit;          -- Inputs
         Cout, Sum: out bit);        -- Outputs
end component;
signal C: bit_vector(3 downto 1);
begin    --instantiate four copies of the FullAdder
   FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
   FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
   FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
   FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```


4-bit Adder - Simulation

```
list A B Co C Ci S  -- put these signals on the output list
force A 1111        -- set the A inputs to 1111
force B 0001        -- set the B inputs to 0001
force Ci 1          -- set the Ci to 1
run 50              -- run the simulation for 50 ns
force Ci 0
force A 0101
force B 1110
run 50

ns  delta     a     b    co  c  ci   s
 0     +0   0000  0000   0 000  0 0000
 0     +1   1111  0001   0 000  1 0000
10     +0   1111  0001   0 001  1 1111
20     +0   1111  0001   0 011  1 1101
30     +0   1111  0001   0 111  1 1001
40     +0   1111  0001   1 111  1 0001
50     +0   0101  1110   1 111  0 0001
60     +0   0101  1110   1 110  0 0101
70     +0   0101  1110   1 100  0 0111
80     +0   0101  1110   1 100  0 0011
```

Modeling Flip-Flops Using VHDL Processes

General form of process

```
process(sensitivity-list)
  begin
    sequential-statements
  end process;
```

- Whenever one of the signals in the sensitivity list changes, the sequential statements are executed in sequence one time

Concurrent Statements vs. Process

A, B, C, D are integers
A=1, B=2, C=3, D=0
D changes to 4 at time 10

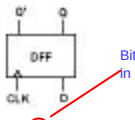
```
A <= B;   -- statement 1
B <= C;   -- statement 2
C <= D;   -- statement 3
```

```
process (B, C, D)
begin
  A <= B;   -- statement 1
  B <= C;   -- statement 2
  C <= D;   -- statement 3
end process;
```

Simulation Results

time	delta	A	B	C	D	
0	+0	0	1	2	0	
10	+0	1	2	3	4	(stat. 3 exe.)
10	+1	1	2	4	4	(stat. 2 exe.)
10	+2	1	4	4	4	(stat. 1 exe.)
10	+3	4	4	4	4	(no exec.)

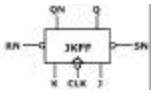
D Flip-flop Model



Bit values are enclosed in single quotes

```
entity DFF is
  port (D, CLK: in bit;
        Q: out bit; QN: out bit := '1');
-- initialize QN to '1' since bit signals are initialized to '0' by default
end DFF;

architecture SIMPLE of DFF is
begin
  process (CLK)                    -- process is executed when CLK changes
  begin
    if CLK = '1' then              -- rising edge of clock
      Q  <= D after 10 ns;
      QN <= not D after 10 ns;
    end if;
  end process;
end SIMPLE;
```

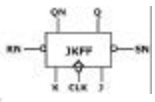

JK Flip-Flop Model



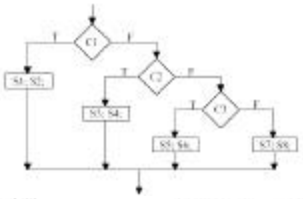

```
entity JKFF is
  port (SN, RN, J, K, CLK: in bit;          -- inputs
        Q: inout bit; QN: out bit := '1');  -- see Note 1
end JKFF;

architecture JKFF1 of JKFF is
begin
  process (SN, RN, CLK)                     -- see Note 2
  begin
    if RN = '0' then Q<= '0' after 10 ns;                 -- RN=0 will clear the FF
    elsif SN = '0' then Q<= '1' after 10 ns;              -- SN=0 will set the FF
    elsif CLK = '0' and CLK'event then                    -- see Note 3
      Q <= (J and not Q) or (not K and Q) after 10 ns;    -- see Note 4
    end if;
  end process;
  QN <= not Q;                              -- see Note 5
end JKFF1;
```


JK Flip-Flop Model

Note 1: Q is declared as inout (rather than out) because it appears on both the left and right sides of an assignment within the architecture.
Note 2: The flip-flop can change state in response to changes in SN, RN, and CLK, so these 3 signals are in the sensitivity list.
Note 3: The condition (CLK = '0' and CLK'event) is TRUE only if CLK has just changed from '1' to '0'.
Note 4: Characteristic equation which describes behavior of J-K flip-flop.
Note 5: Every time Q changes, QN will be updated. If the statement were placed within the process, the old value of Q would be used instead of the new value.



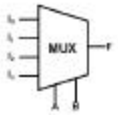
Using Nested IFs and ELSEIFs




```
if (C1) then S1; S2;
  else if (C2) then S3; S4;
         else if (C3) then S5; S6;
                else S7; S8;
         end if;
  end if;
end if;
```

```
if (C1) then S1; S2;
  elsif (C2) then S3; S4;
  elsif (C3) then S5; S6;
  else S7; S8;
end if;
```

VHDL Models for a MUX




```
F <= (not A and not B and I0) or
     (not A and B and I1) or
     (A and not B and I2) or
     (A and B and I3);
```

MUX model using a conditional signal assignment statement:

```
F <= I0 when Sel = 0
     else I1 when Sel = 1
     else I2 when Sel = 2
     else I3;
```

Sel represents the integer equivalent of a 2-bit binary number with bits A and B

If a MUX model is used inside a process, the MUX can be modeled using a CASE statement (cannot use a concurrent statement):

```
case Sel is
    when 0 => F <= I0;
    when 1 => F <= I1;
    when 2 => F <= I2;
    when 3 => F <= I3;
end case;
```

The case statement has the general form:

```
case expression is
    when choice1 => sequential statements1
    when choice2 => sequential statements2
    ...
    [when others => sequential statements]
end case;
```


MUX Models (1)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity SELECTOR is
 port (
   A  : in std_logic_vector(15 downto 0);
   SEL : in std_logic_vector( 3 downto 0);
   Y  : out std_logic);
end SELECTOR;
```

```
architecture RTL1 of SELECTOR is
begin
 p0 : process (A, SEL)
 begin
  if (SEL = "0000") then    Y <= A(0);
  elsif (SEL = "0001") then Y <= A(1);
  elsif (SEL = "0010") then Y <= A(2);
  elsif (SEL = "0011") then Y <= A(3);
  elsif (SEL = "0100") then Y <= A(4);
  elsif (SEL = "0101") then Y <= A(5);
  elsif (SEL = "0110") then Y <= A(6);
  elsif (SEL = "0111") then Y <= A(7);
  elsif (SEL = "1000") then Y <= A(8);
  elsif (SEL = "1001") then Y <= A(9);
  elsif (SEL = "1010") then Y <= A(10);
  elsif (SEL = "1011") then Y <= A(11);
  elsif (SEL = "1100") then Y <= A(12);
  elsif (SEL = "1101") then Y <= A(13);
  elsif (SEL = "1110") then Y <= A(14);
  else    Y <= A(15);
  end if;
 end process;
endRTL1;
```


MUX Models (2)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity SELECTOR is
 port (
   A  : in std_logic_vector(15 downto 0);
   SEL : in std_logic_vector( 3 downto 0);
   Y  : out std_logic);
end SELECTOR;
```

```
architecture RTL3 of SELECTOR is
begin
 with SEL select
   Y <= A(0)  when "0000",
        A(1)  when "0001",
        A(2)  when "0010",
        A(3)  when "0011",
        A(4)  when "0100",
        A(5)  when "0101",
        A(6)  when "0110",
        A(7)  when "0111",
        A(8)  when "1000",
        A(9)  when "1001",
        A(10) when "1010",
        A(11) when "1011",
        A(12) when "1100",
        A(13) when "1101",
        A(14) when "1110",
        A(15) when others;
end RTL3;
```


MUX Models (3)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity SELECTOR is
 port (
   A  : in std_logic_vector(15 downto 0);
   SEL : in std_logic_vector( 3 downto 0);
   Y  : out std_logic);
end SELECTOR;
```

```
architecture RTL2 of SELECTOR is
begin
 p1 : process (A, SEL)
 begin
  case SEL is
    when "0000" => Y <= A(0);
    when "0001" => Y <= A(1);
    when "0010" => Y <= A(2);
    when "0011" => Y <= A(3);
    when "0100" => Y <= A(4);
    when "0101" => Y <= A(5);
    when "0110" => Y <= A(6);
    when "0111" => Y <= A(7);
    when "1000" => Y <= A(8);
    when "1001" => Y <= A(9);
    when "1010" => Y <= A(10);
    when "1011" => Y <= A(11);
    when "1100" => Y <= A(12);
    when "1101" => Y <= A(13);
    when "1110" => Y <= A(14);
    when others => Y <= A(15);
  end case;
 end process;
endRTL2;
```

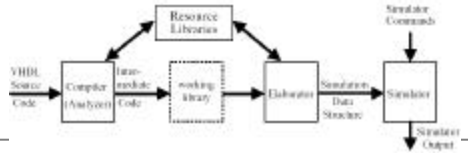

MUX Models (4)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity SELECTOR is
 port (
   A  : in std_logic_vector(15 downto 0);
   SEL : in std_logic_vector( 3 downto 0);
   Y  : out std_logic);
end SELECTOR;
```

```
architecture RTL4 of SELECTOR is
begin
 Y <= A( conv_integer(SEL));
end RTL4;
```

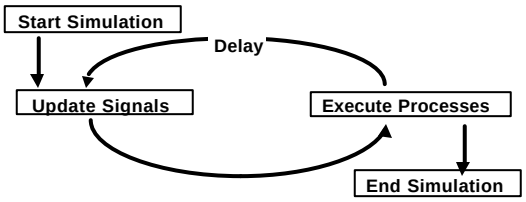

Compilation and Simulation of VHDL Code

- Compiler (Analyzer) – checks the VHDL source code
  - does it conforms with VHDL syntax and semantic rules
  - are references to libraries correct
- Intermediate form used by a simulator or by a synthesizer
- Elaboration
  - create ports, allocate memory storage, create interconnections, ...
  - establish mechanism for executing of VHDL processes



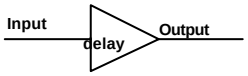
Timing Model

- VHDL uses the following simulation cycle to model the stimulus and response nature of digital hardware



Delay Types

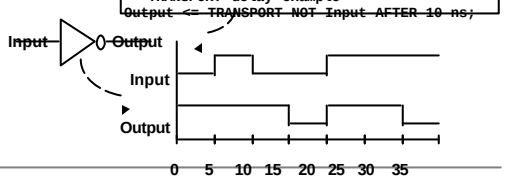
- All VHDL signal assignment statements prescribe an amount of time that must transpire before the signal assumes its new value
- This prescribed delay can be in one of three forms:
  - Transport – prescribes propagation delay only
  - Inertial – prescribes propagation delay and minimum input pulse width
  - Delta – the default if no delay time is explicitly specified



Transport Delay

- Transport delay must be explicitly specified
  - I.e. keyword "TRANSPORT" must be used
- Signal will assume its new value after specified delay

```
-- TRANSPORT delay example
Output <= TRANSPORT NOT Input AFTER 10 ns;
```



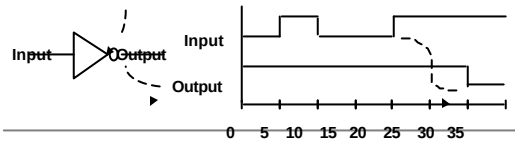
Inertial Delay

- Provides for specification propagation delay and input pulse width, i.e. 'inertia' of output:

```
target <= [REJECT time_expression] INERTIAL waveform;
```

- Inertial delay is default and REJECT is optional:

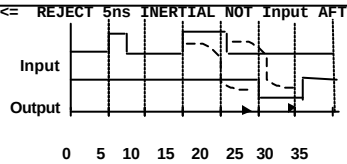
```
Output <= NOT Input AFTER 10 ns;
  Propagation delay and minimum pulse width are 10ns
```



Inertial Delay (cont.)

- Example of gate with 'inertia' smaller than propagation delay
  - e.g. Inverter with propagation delay of 10ns which suppresses pulses shorter than 5ns

```
Output <=  REJECT 5ns INERTIAL NOT Input AFTER 10ns;
```



- Note: the REJECT feature is new to VHDL 1076-1993

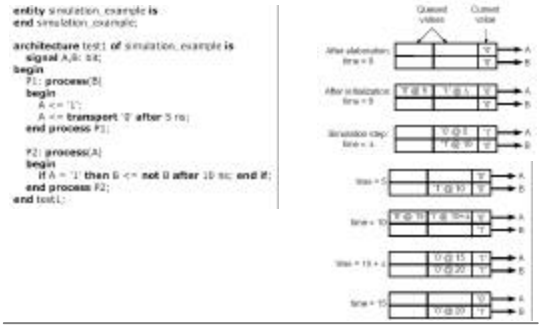
Delta Delay

- Default signal assignment propagation delay if no delay is explicitly prescribed
  - VHDL signal assignments do not take place immediately
  - Delta is an infinitesimal VHDL time unit so that all signal assignments can result in signals assuming their values at a future time
  - E.g.

```
Output <= NOT Input;
-- Output assumes new value in one delta cycle
```

- Supports a model of concurrent VHDL process execution
  - Order in which processes are executed by simulator does not affect simulation output

Simulation Example



Problem #1

- Using the labels, list the order in which the following signal assignments are evaluated if in2 changes from a '0' to a '1'. Assume in1 has been a '1' and in2 has been a '0' for a long time, and then at time *t* in2 changes from a '0' to a '1'.

```
entity not_another_prob is
        port (in1, in2: in bit;
        a: out bit);
end not_another_prob;

architecture oh_behave of not_another_prob is
        signal b, c, d, e, f: bit;
begin
        L1:  d <= not(in1);
        L2:  c<= not(in2);
        L3:  f <= (d and in2) ;
        L4:  e <= (c and in1) ;
        L5:  a <= not b;
        L6:  b <= e or f;
end oh_behave;
```

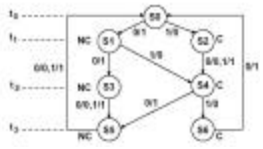

Problem #2

- Under what conditions do the two assignments below result in the same behavior? Different behavior? Draw waveforms to support your answers.

```
out <= reject 5 ns inertial (not a) after 20 ns;
out <= transport (not a) after 20 ns;
```


Modeling a Sequential Machine

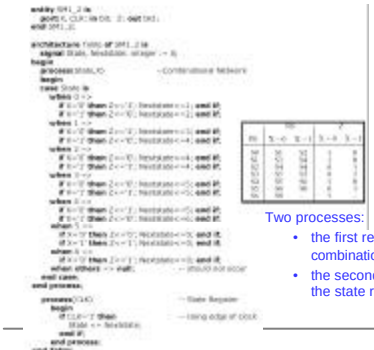
Mealy Machine for
8421 BCD to 8421 BCD + 3 bit serial converter



	NS		Z	
PS	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	–	1	–

How to model this in VHDL?

Behavioral VHDL Model



Two processes:

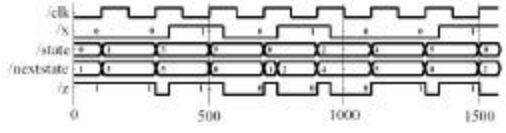
- the first represents the combinational network;
- the second represents the state register

Simulation of the VHDL Model

Simulation command file:

```
wave CLK X State NextState Z
force CLK 0 0, 1 100 -repeat 200
force X 0 0, 1 350, 0 550, 1 750, 0 950, 1 1350
run 1600
```

Waveforms:



Dataflow VHDL Model

```
-- The following is a description of the sequential machine of
-- Figure 1-17 in terms of its next state equations.
-- The following state assignment was used:
-- S0-->0; S1-->4; S2-->5; S3-->7; S4-->6; S5-->3; S6-->2

entity SM1_2 is
   port(X,CLK: in bit;
        Z: out bit);
end SM1_2;

architecture Equations1_4 of SM1_2 is
   signal Q1,Q2,Q3: bit;
begin
   process(CLK)
   begin
      if CLK='1' then          -- rising edge of clock
         Q1<=not Q2 after 10 ns;
         Q2<=Q1 after 10 ns;
         Q3<=(Q1 and Q2 and Q3) or (not X and Q1 and not Q3) or
             (X and not Q1 and not Q2) after 10 ns;
      end if;
   end process;
   Z<=(not X and not Q3) or (X and Q3) after 20 ns;
end Equations1_4;
```

$$Q_1(t^+) = Q_2$$

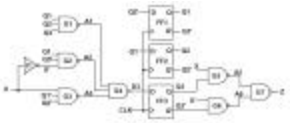
$$Q_2(t^+) = Q_1$$

$$Q_3(t^+) = Q_1 Q_2 Q_3 + X\,'Q_1 Q\,'_3 + X\,'Q\,'_1 Q\,'_2$$

$$Z = X\,'Q\,'_3 + XQ_3$$

Structural Model

```
library BITLIB;
use BITLIB.bit_pack.all;

entity SM1_2 is
   port(X,CLK: in bit;
        Z: out bit);
end SM1_2;

architecture Structure of SM1_2 is
   signal A1,A2,A3,A5,A6,D3: bit:='0';
   signal Q1,Q2,Q3: bit:='0';
   signal Q1N,Q2N,Q3N, XN: bit:='1';
begin
   I1: Inverter port map (X,XN);
   G1: Nand3 port map (Q1,Q2,Q3,A1);
   G2: Nand3 port map (Q1,Q3N,XN,A2);
   G3: Nand3 port map (X,Q1N,Q2N,A3);
   G4: Nand3 port map (A1,A2,A3,D3);
   FF1: DFF port map (Q2N,CLK,Q1,Q1N);
   FF2: DFF port map (Q1,CLK,Q2,Q2N);
   FF3: DFF port map (D3,CLK,Q3,Q3N);
   G5: Nand2 port map (X,Q3,A5);
   G6: Nand2 port map (XN,Q3N,A6);
   G7: Nand2 port map (A5,A6,Z);
end Structure
```



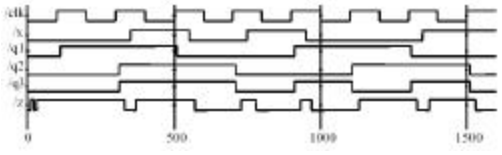
Package bit_pack is a part of library BITLIB – includes gates, flip-flops, counters
(See Appendix B for details)

Simulation of the Structural Model

Simulation command file:

```
wave CLK X Q1 Q2 Q3 Z
force CLK 0 0, 1 100 -repeat 200
force X 0 0, 1 350, 0 550, 1 750, 0 950, 1 1350
run 1600
```

Waveforms:



Wait Statements

- ... an alternative to a sensitivity list
  - Note: a process cannot have both wait statement(s) and a sensitivity list
- Generic form of a process with wait statement(s)

```
process
begin
   sequential-statements
   wait statement
   sequential-statements
   wait-statement
   ...
end process;
```

How wait statements work?

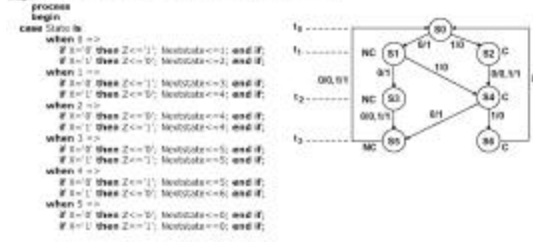
- Execute seq. statement until a wait statement is encountered.
- Wait until the specified condition is satisfied.
- Then execute the next set of sequential statements until the next wait statement is encountered.
- ...
- When the end of the process is reached start over again at the beginning.

Forms of Wait Statements

```
wait on sensitivity-list;
wait for time-expression;
wait until boolean-expression;
```

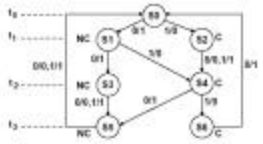
- Wait on
  - until one of the signals in the sensitivity list changes
- Wait for
  - waits until the time specified by the time expression has elapsed
  - What is this:
    `wait for 0 ns;`
- Wait until
  - the boolean expression is evaluated whenever one of the signals in the expression changes, and the process continues execution when the expression evaluates to TRUE

Using Wait Statements (1)



Using Wait Statements (2)

```
        when S =>
            if X='0' then Z<='1'; Nextstate<=0; end if;
            when others => null;        -- should not occur
    end case;

    wait on CLK, X;
    if rising_edge(CLK) then            -- rising_edge function is in BITLIB *
        State <= Nextstate;
        wait for 0 ns;                  -- wait for State to be updated
    end if;
    end process;
end table;
```



To Do

- Read
  - Textbook chapters 2.1, 2.2