CS223: Hardware Lab

Tutorial 4
HDL

A. Sahu

Dept of Comp. Sc. & Engg.

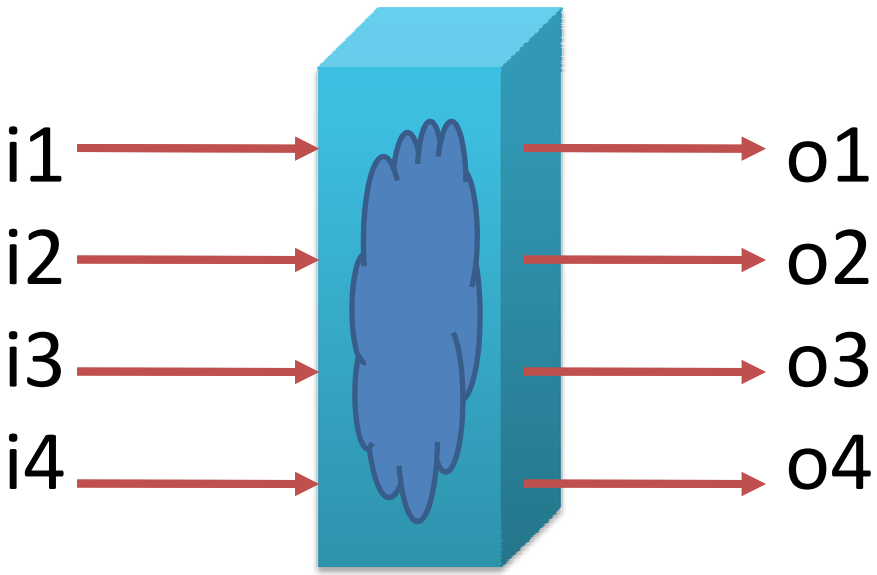
Indian Institute of Technology Guwahati

Outline

- **VHDL basic language concepts**
- basic design methodology
- Examples

Modeling Combinational Logic

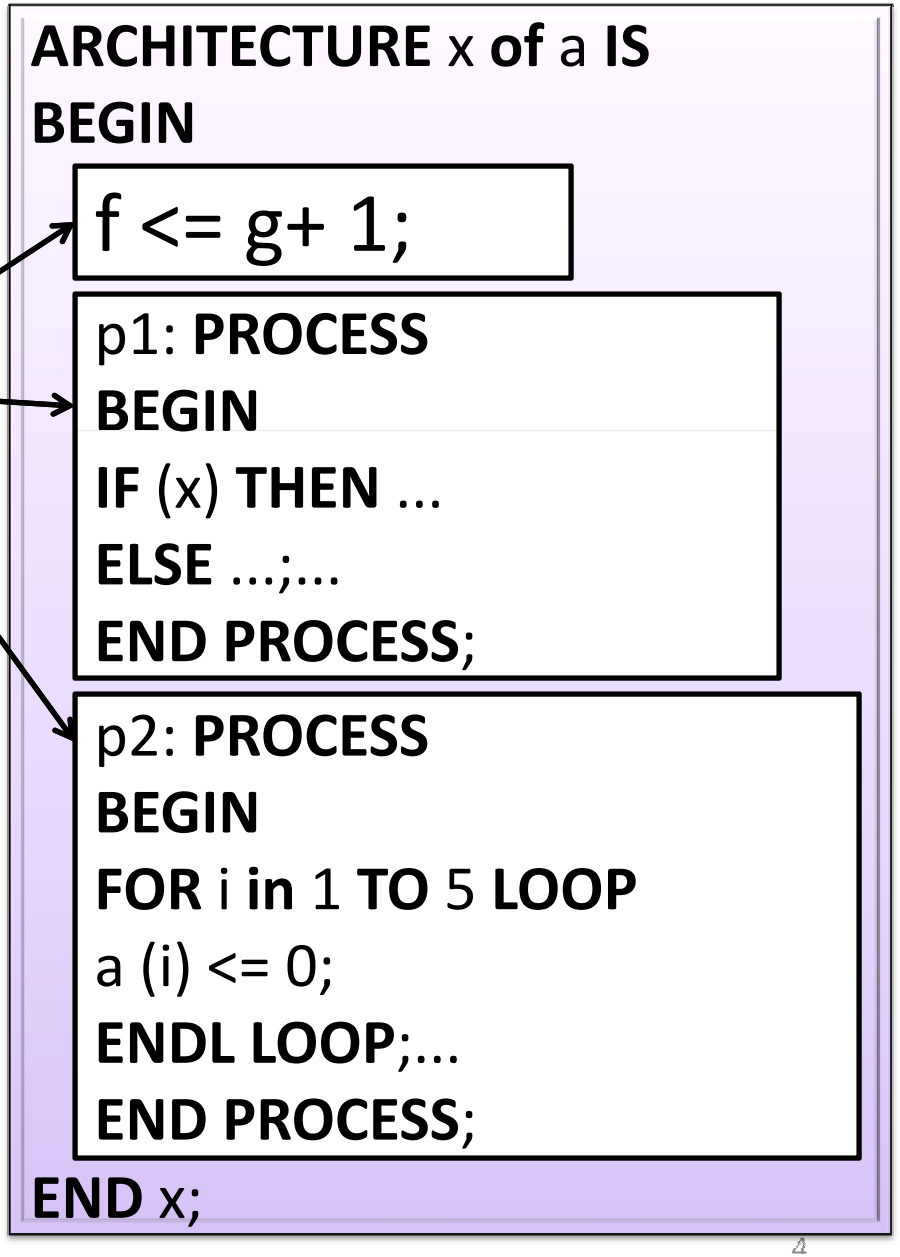
- One concurrent assignment for each output




```
ARCHITECTURE data_flow
OF comb_logic IS
BEGIN
        o1 <= i1 and i2;
        o2 <= (i2 or i3) xor (i1 and i4);
        o3 <= ...;
        o4 <= ...;
END data_flow;
```

VHDL PROCESS

- PROCESS is sequential
- Processes are concurrent w.r.t each other
- Signal assignment is a simple special case
- Architecture consists of a set of Processes (and signal assignments) at top level
- Processes communicate using signals

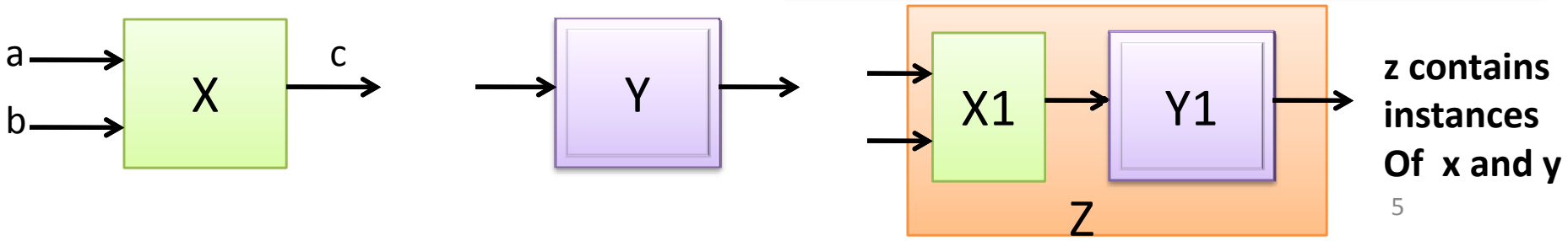


Structural Description

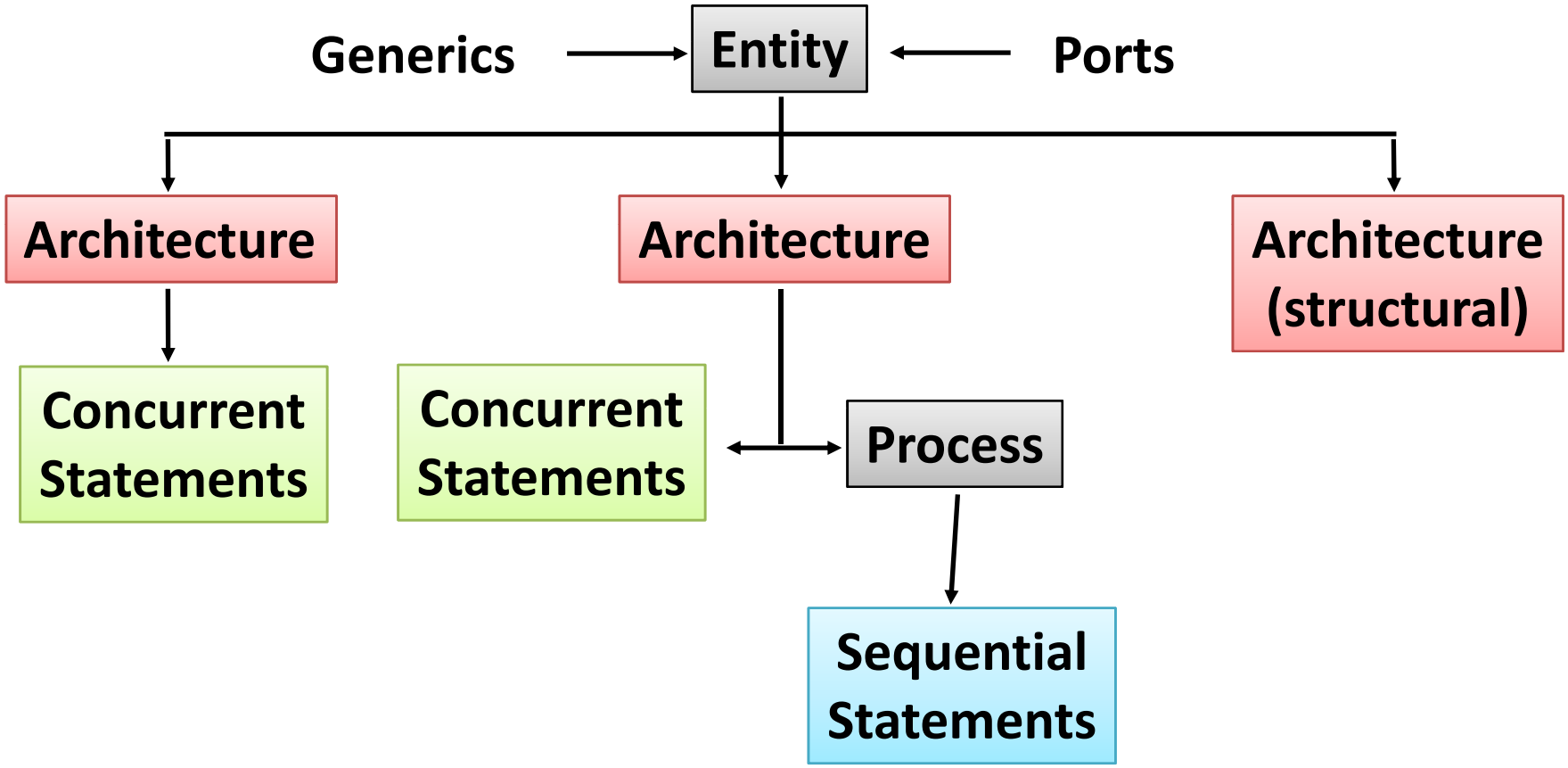
- Instantiation and Interconnection
- **Hierarchy**

```
ENTITY x IS
PORT (a, b: IN BIT, c: OUT BIT);
END x;

ARCHITECTURE xa OF x IS
BEGIN
        c <= a AND b;
END xa;
```

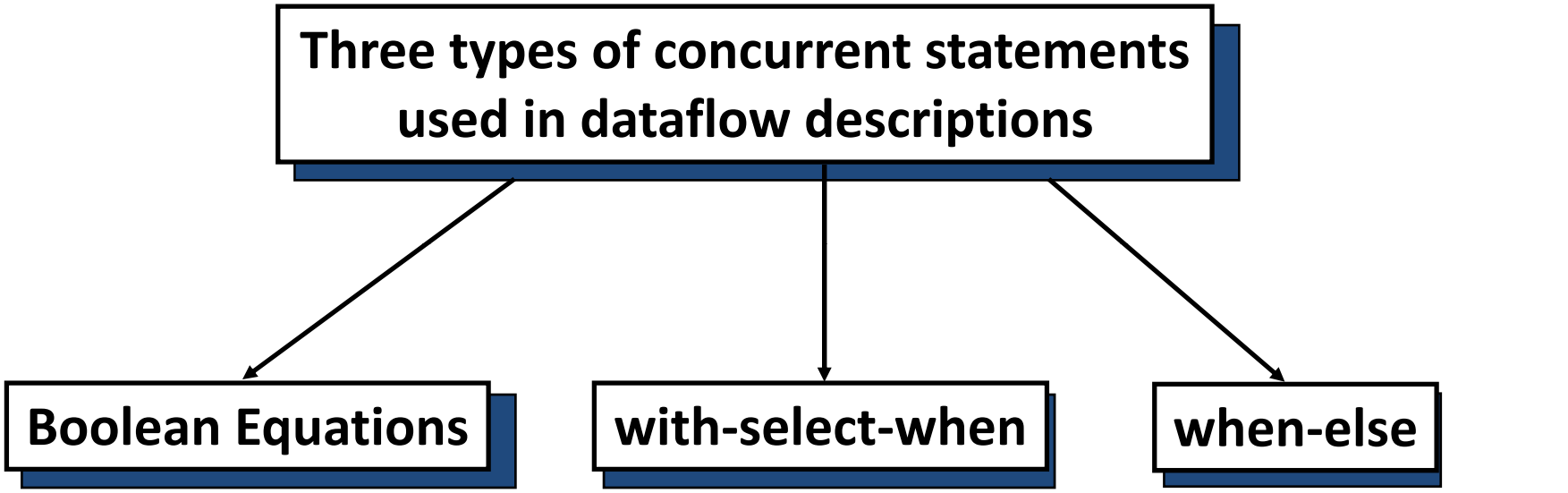
```
ENTITY y IS
PORT (a : IN BIT, b: OUT BIT);
END y;

ARCHITECTURE ya OF y IS
BEGIN
b <= NOT a;
END xa;
```



VHDL Models


Generics
Entity
Ports
Architecture
Architecture
Architecture (structural)
Concurrent Statements
Concurrent Statements
Process
Sequential Statements

Concurrent Statements



For concurrent signal assignments

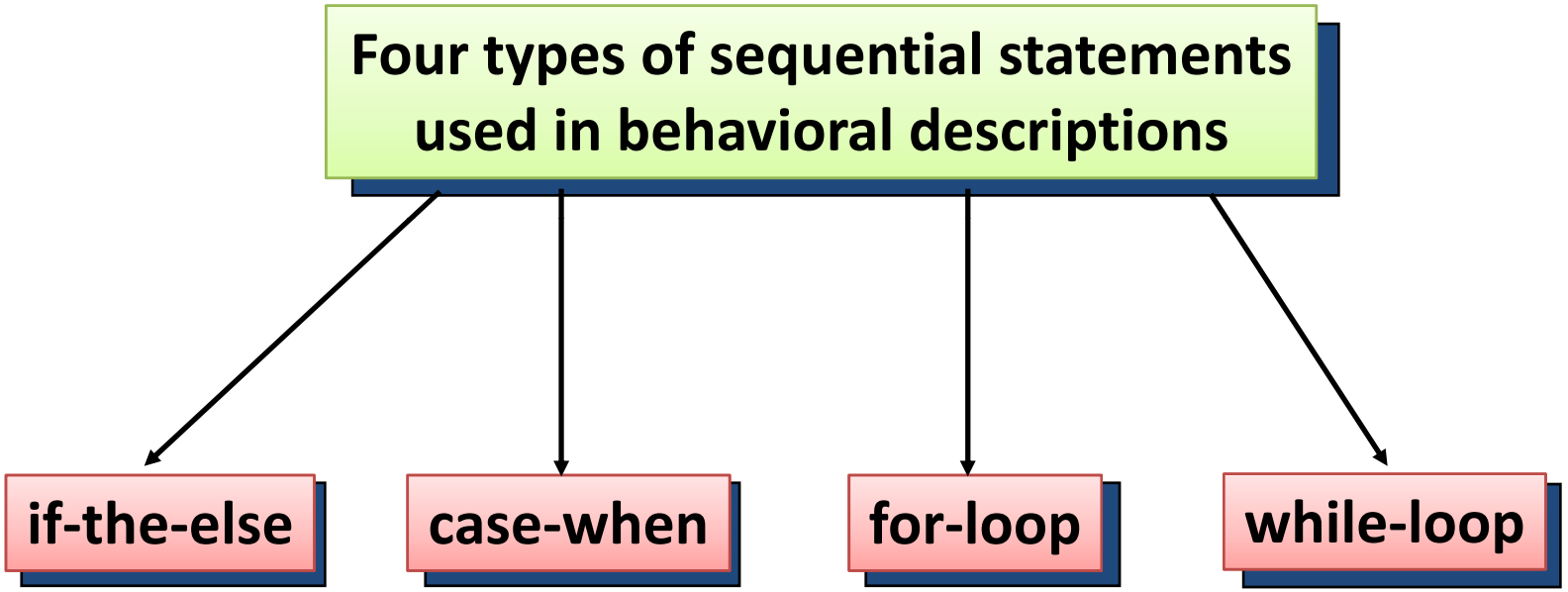
For selective signal assignments

For conditional signal assignments

Sequential Statements

- Sequential statements are contained in a *process*, *function*, or *procedure*.
- Inside a process signal assignment is sequential from a simulation point of view.
- The order in which signal assignments are listed *does affect* the result.

Sequential Statements

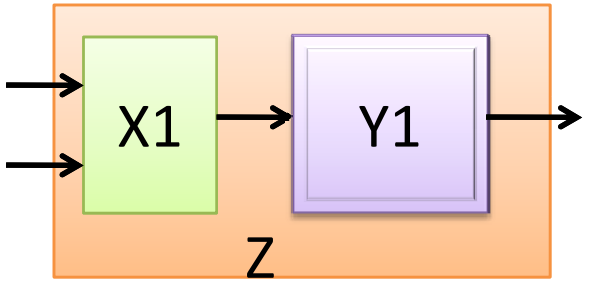


4 ways to DO IT -- the VHDL way

- Dataflow when-else, when
- RTL Clocked, Transfers
- Behavioral Process, For loop, While loop, if then else
- Structural Using Components
- Mixed :

Instantiation and Interconnection - 1

```
ENTITY z IS
        PORT (p, q: IN BIT, r: OUT BIT);
END x;

ARCHITECTURE structural OF z IS
COMPONENT xc
        PORT (a, b: IN BIT; c: OUT BIT);
END COMPONENT;
COMPONENT yc
        PORT (a, b: IN BIT; c: OUT BIT);
END COMPONENT;
FOR ALL: xc USE WORK.x (xa);
FOR ALL: yc USE WORK.y (ya);
SIGNAL t: BIT;
BEGIN
x1: xc PORT MAP (p, q, t);
y1: yc PORT MAP (t, r);
END structural;
```



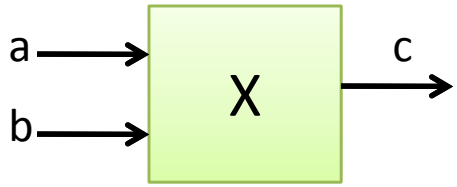
Component declaration

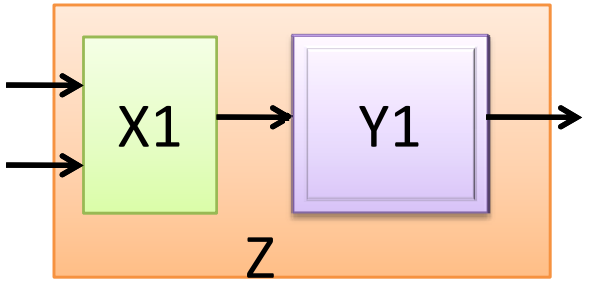
Configuration specification
(which architecture?)

Temporary signal

Instantiation

Instantiation and Interconnection - 2





Instance name

Component name

```
x1:     xc PORT MAP (p, q, t);
y1:     yc PORT MAP (t, r);
```

Same name implies connection

Port association list:
order of names determines connectivity:
a - p
b - q
c – t

Port Mapping

```
COMPONENT xc
PORT (a, b: IN BIT; c: OUT BIT);
END COMPONENT;
```

Mapping by position: preferred for short port lists

```
x1: xc PORT MAP (p, q, t);
```

Mapping by name: preferred for long port lists

```
x1: xc PORT MAP (b => q, a => p, c => t);
```

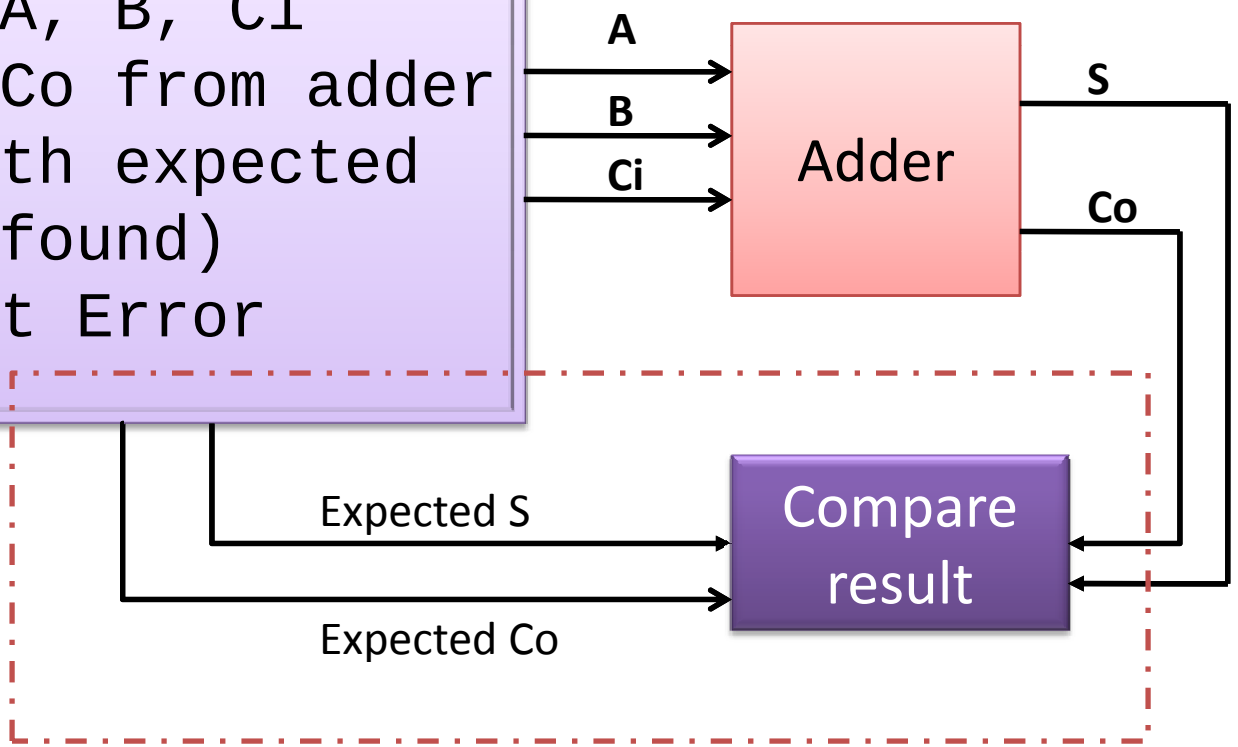
In both cases, complete port mapping should be specified

Test Benches

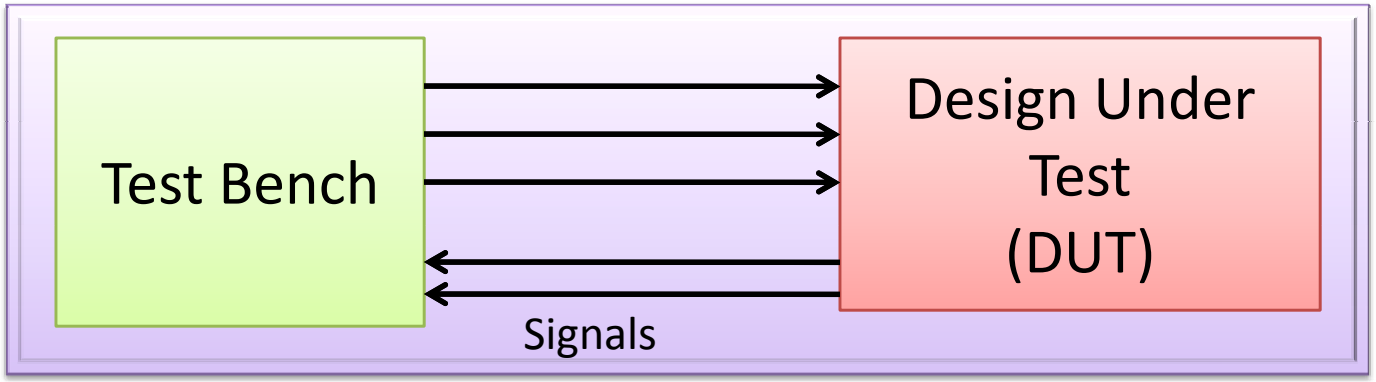
- Purpose - test correctness of Design Under Test (DUT)
  - provide input stimulus
  - observe outputs
  - compare against expected outputs
- Test Bench is also a VHDL model

Test bench for Adder

```
Test bench
For (i=0;i<8;i++){
     wait 10ns
     Read (Truth Table)
     Send read A, B, Ci
     Receive S,Co from adder
     Compare with expected
     if (error found)
          Report Error
}
```



Test Bench Modelling



- Test bench a separate VHDL entity
- Ports are connected to DUT's ports
  - i/p port corresponding to DUT's o/p port
  - o/p port corresponding to DUT's i/p port
- Test bench instantiates the DUT
- Stimulus generation and output monitoring in a separate VHDL process
- Signals are connected to DUT's ports

Libraries and Packages

- PACKAGE - collection of
  - Components
  - data types
  - functions/procedures
- LIBRARY - collection of PACKAGEs

Packages

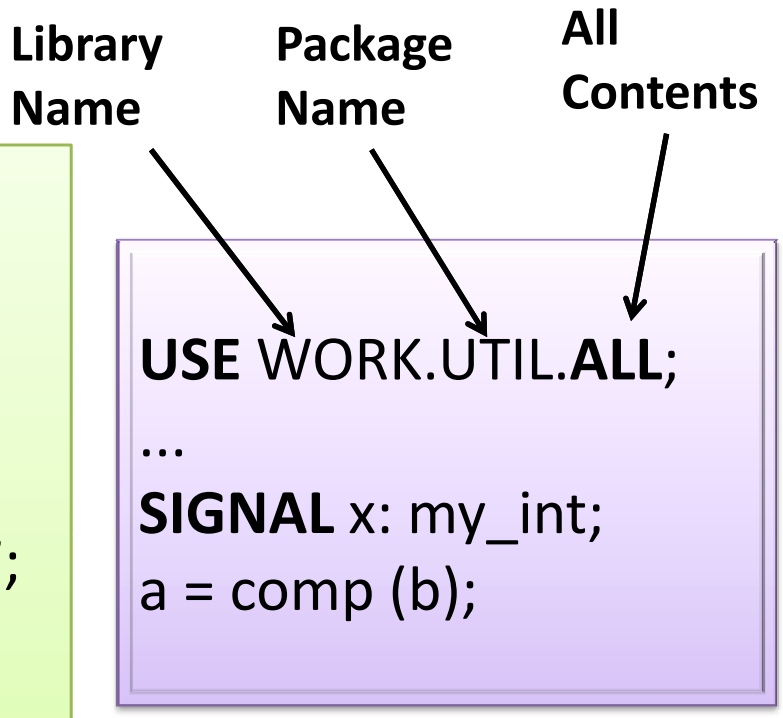
```
PACKAGE util IS
        COMPONENT c IS
                PORT (a: IN BIT, b: OUT BIT);
        END COMPONENT
        TYPE my_int IS INTEGER RANGE -7 TO 7;
        FUNCTION comp (a: BIT_VECTOR)
        RETURN BIT_VECTOR;
END util;

PACKAGE BODY util IS
FUNCTION comp (a: BIT_VECTOR)  RETURN
BIT_VECTOR IS
BEGIN
        RETURN NOT a;
END comp;
END util;
```

Package Declaration

Package Body

Using a Package

```
PACKAGE util IS
COMPONENT c IS
        PORT (a: IN BIT, b: OUT BIT);
        END COMPONENT
TYPE my_int IS INTEGER RANGE -7 TO 7;
FUNCTION comp (a: BIT_VECTOR)
RETURN BIT_VECTOR;
END util;
```



Libraries

- STD
  - STANDARD : types/utilities (BIT, TIME, INTEGER,...)
- TEXTIO
  - interface to text files
- WORK
  - default library for storing user designs
- STD_LOGIC_1164
  - multi-valued logic

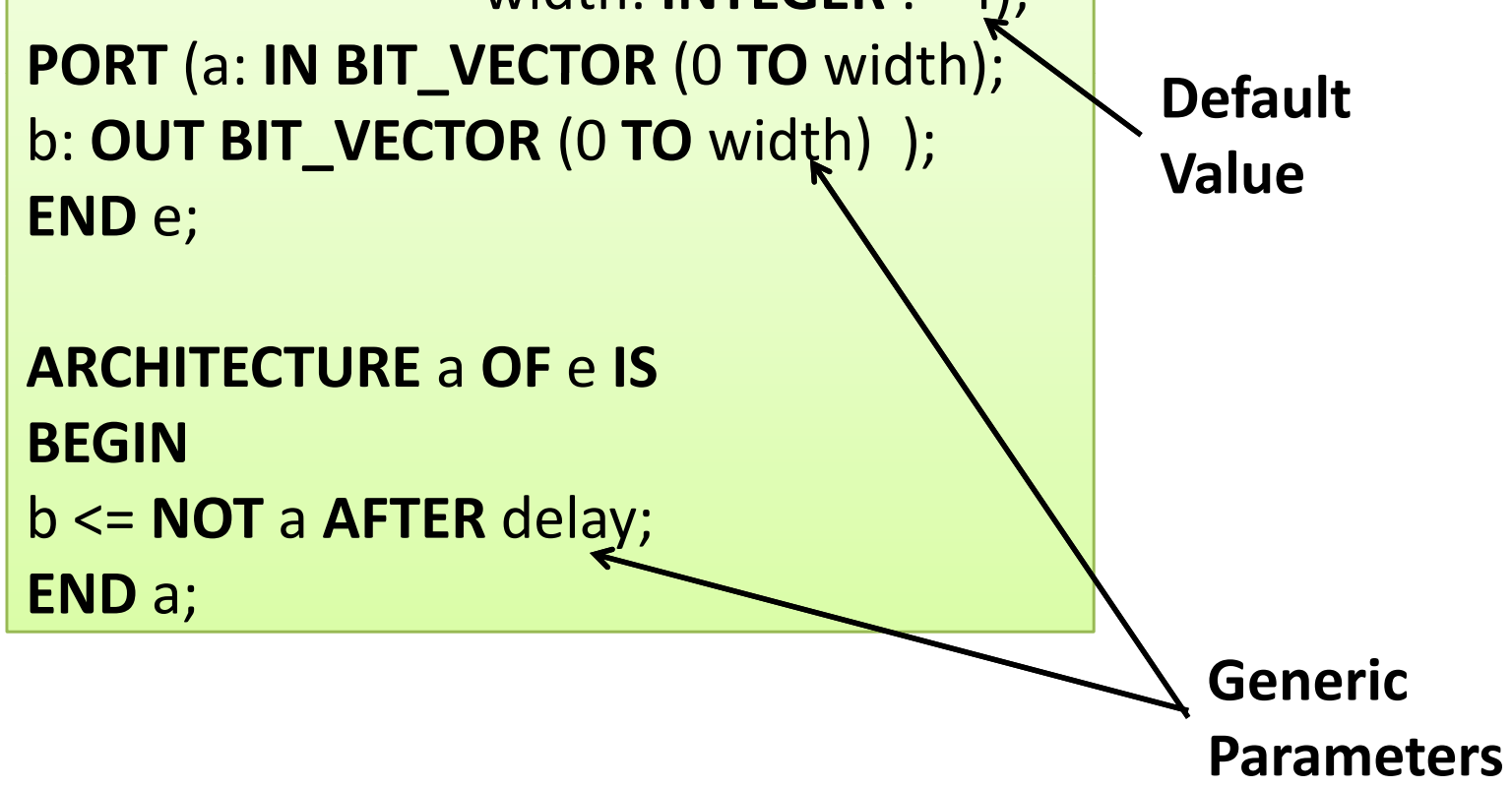
TEXTIO Package

- Data types and functions for
  - reading from text files
  - writing out text files

```
FILE    f: TEXT IS "file_name";
VARIABLE    one_line: line;
VARIABLE    str: STRING;
...
READLINE (f, one_line);    -- read one line from file
READ (str, one_line);      -- read a word from line
WRITELINE (g, one_line); -- write one line to file
WRITE (str, one_line);     -- write a word into line
```

Design Parameterization: GENERIC & GENERATE

```
ENTITY e IS
GENERIC (delay: TIME := 2 NS;
                       width: INTEGER := 4);
PORT (a: IN BIT_VECTOR (0 TO width);
b: OUT BIT_VECTOR (0 TO width)  );
END e;

ARCHITECTURE a OF e IS
BEGIN
b <= NOT a AFTER delay;
END a;
```



Passing GENERIC Parameters

```
ENTITY c IS
GENERIC (delay: TIME := 4 ns); PORT (a: IN BIT; b: OUT BIT);
END c;
```

```
ARCHITECTURE a OF e IS
COMPONENT c
GENERIC (t: TIME:= 4 NS);
PORT (a: IN BIT, b: OUT BIT);
END COMPONENT;
SIGNAL x, y: BIT;
FOR ALL: c USE work.c (arc);

BEGIN
c1: c GENERIC MAP (3 ns)
PORT MAP (x, y);
END a;
```

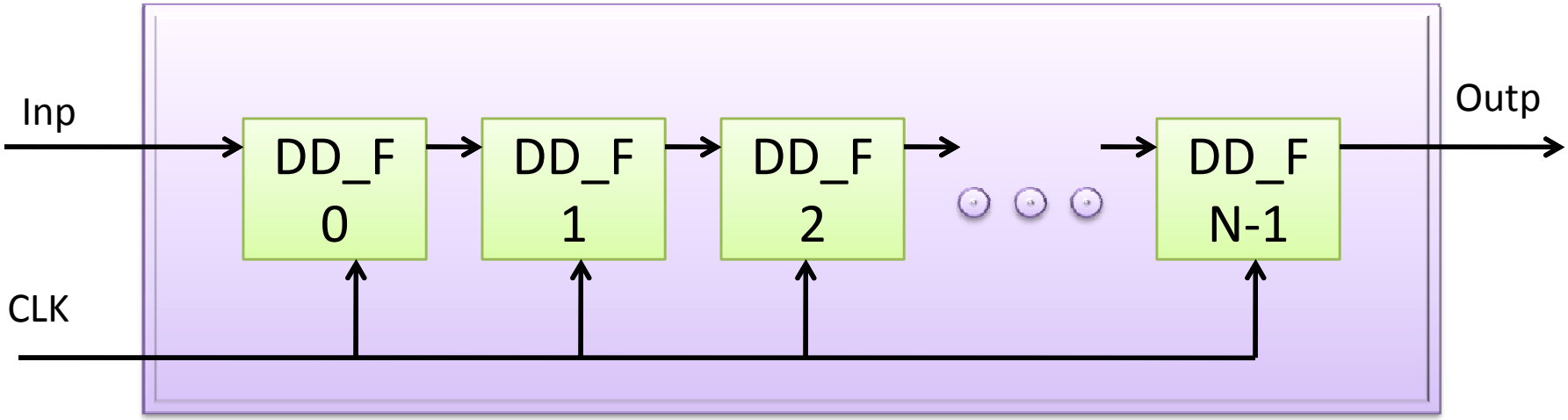
Delay Parameter 3ns

```
ARCHITECTURE def OF e IS
COMPONENT c
GENERIC (t: TIME:= 4 NS);
PORT (a: IN BIT, b: OUT BIT);
END COMPONENT;
SIGNAL x, y: BIT;
FOR ALL: c USE work.c (arc);

BEGIN
c1: c PORT MAP (x, y);
END def;
```

Delay Default Value 4ns

GENERATE: Conditional and Looped Instantiation

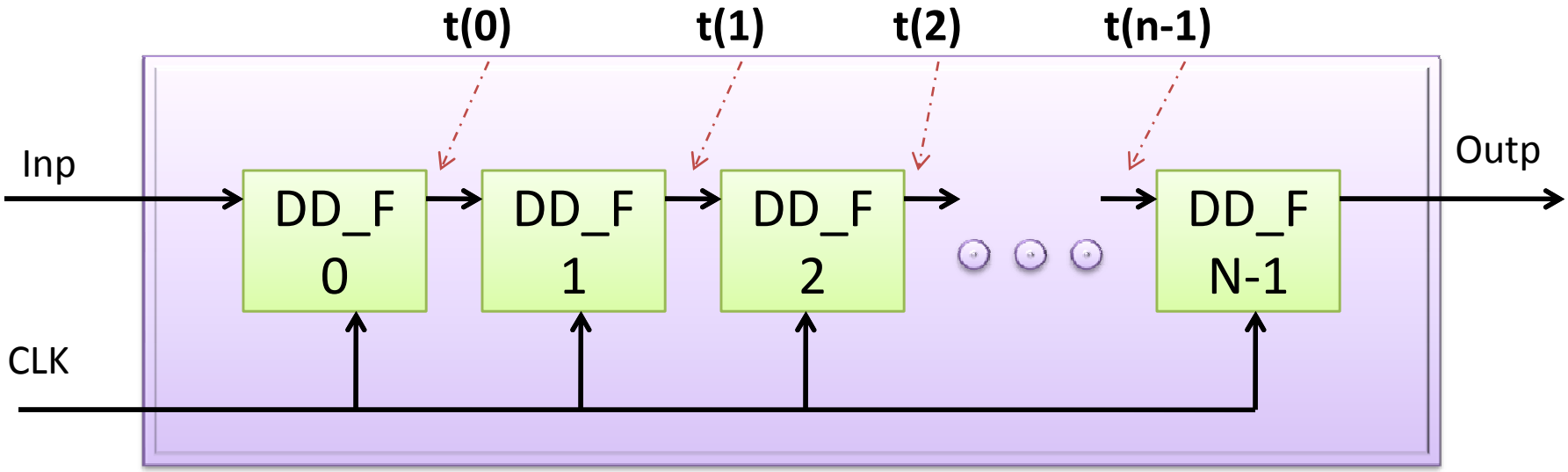
- Number of instances of DFF determined by Generic Parameter n



GENERATE: Conditional and Looped Instantiation

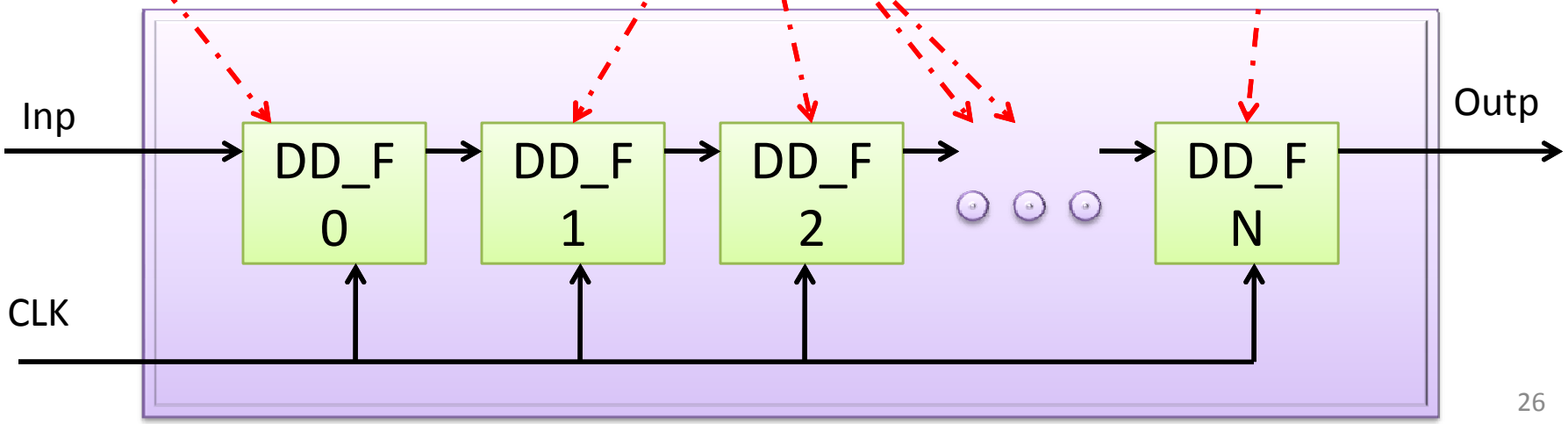
```
GENERIC (n: INTEGER)...
...
SIGNAL t: BIT_VECTOR (0 TO n-1);
```

Need intermediate signal t (0 to n-1)



GENERATE Statement

```
SIGNAL t: BIT_VECTOR (0 TO n-1);
...
dff_0: DFF PORT MAP (Inp, Clk, t (0));
dff_n: DFF PORT MAP (t (n-1), Clk, Outp);
FOR i IN 1 TO n-1 GENERATE
        dff_i: DFF PORT MAP ( t (i-1), Clk, t (i) );
END GENERATE;
```



N Bit Ripple Carry Adder: FA

```
ENTITY full_adder IS
    PORT (a, b, c_in: IN std_logic;
    Sum, Carry: OUT std_logic );
END full_adder;

ARCHITECTURE full_adder_arch_1 OF  full_adder  IS
SIGNAL S1, S2, S3: std_logic;
BEGIN
          s3     <= ( a AND b )       after 5 ns;
          s2     <= ( c_in AND s1 ) after 5 ns;
          s1     <= ( a XOR b )       after 15 ns;
          Carry <= ( s2 OR s3 )      after 5 ns;
          Sum  <= ( s1 XOR c_in ) after 15 ns;
END full_adder_arch_1;
```

N Bit Ripple Carry Adder

```
ENTITY adder_bits_n IS

GENERIC(n: INTEGER  :=  2);

  PORT (
        Cin: IN     std_logic;
        a, b: IN     std_logic_vector(n-1 downto 0);
        S: OUT std_logic_vector(n-1 downto 0);
        Cout: OUT std_logic
         );

END;
```

N Bit Ripple Carry Adder

```
ARCHITECTURE ripple_n_arch OF adder_bits_n IS

   COMPONENT full_adder
         PORT (x, y, z: IN std_logic; Sum, Carry: OUT std_logic);
   END COMPONENT;

   SIGNAL      t: std_logic_vector(n downto 0);

BEGIN
   t(0)   <= Cin; Cout <= t(n);
   FA: FOR i in 0 to n-1 GENERATE
         FA_i: full_adder PORT MAP (t(i), a(i), b(i), S(i), t(i+1));
   end generate;
END;
```

Test benches for 4 bit adder: Stimulus only

```
ARCHITECTURE tb OF tb_adder_4 IS

  COMPONENT adder_bits_n  GENERIC(n: INTEGER := 2);
   PORT ( Cin: IN    std_logic;   a, b: IN    std_logic_vector(n-1 downto 0);
   S: OUT std_logic_vector(n-1 downto 0);   Cout: OUT std_logic);
   END COMPONENT;

   SIGNAL      x, y, Sum:     std_logic_vector(n downto 0);
   SIGNAL      c, Cout:        std_logic;
BEGIN
    x <= "0000", "0001" after 200 ns, "0101", after 400 ns;
    y <= "0010", "0011" after 200 ns, "1010", after 400 ns;
    c <= '1', '0' after 200 ns;
    UUT_ADDER_4: adder_bits_n GENERIC MAP(4)
                              PORT MAP (c, x, y, Sum, Cout);
END
```

Data Flow: Full Adder

```
ENTITY full_adder IS
    PORT (a, b, c_in: IN std_logic;
    Sum, Carry: OUT std_logic );
END full_adder;

ARCHITECTURE Data_Flow OF  full_adder  IS
SIGNAL S1, S2, S3: std_logic;
BEGIN
        s3     <= ( a AND b )       after 5 ns;
        s2     <= ( c_in AND s1 ) after 5 ns;
        s1     <= ( a XOR b )       after 15 ns;
        Carry <= ( s2 OR s3 )      after 5 ns;
        Sum  <= ( s1 XOR c_in ) after 15 ns;
END full_adder_arch_1;
```

Components : FA

```
Architecture structural of Full_adder is
   component  XOR_GATE is
         port(   X, Y :  in std_logic; F2: out std_logic  );
   end component;
   component  AND_GATE is
         port(   X, Y :  in std_logic; F2: out std_logic  );
   end component;
   component  OR_GATE is ……..
   signal s1, s2.s3: std_logic;          -- signal just like wire
Begin
          XOR1:          XOR_GATE port map (a, b, s1);
         AND1:           AND_GATE port map  (a, b , s3);
          AND2 :         AND_GATE port map(c_in, s1, s2);
          OR1 :          OR_GATE port map (s2,s3, Carry);
         XOR2:           XOR_GATE port map (s1,c_in, Sum);
end structural;
```

Behavioral Model

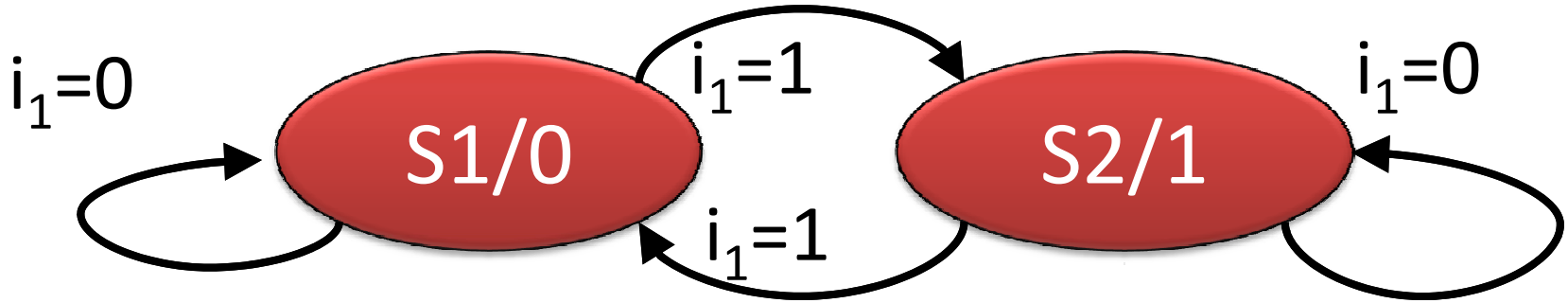
```
architecture BEHAV_FA of FULL_ADDER is
signal int1, int2, int3: std_logic;
Begin   -- Process P1 that defines the first half adder
P1: process (A, B)
    begin
        int1<= A xor B;
        int2<= A and B;
    end process;
-- Process P2 that defines the second half adder and the OR -- gate
P2: process (int1, int2, Cin)
    begin
        Sum <= int1 xor Cin;
        int3 <= int1 and Cin;
        Cout <= int2 or int3;
    end process;
end BEHAV_FA;
```

Resister

```
entity regis is
   port(       rst, clk, load: in std_logic;
               input: in std_logic_vector( 3 downto 0 );
               output: out std_logic_vector( 3 downto 0 )
   );
end regis;
architecture regis_arc of regis is
begin
   process( rst, clk, load, input )
   begin
      if( rst = '1' ) then       output <= "0000";
      elsif( clk'event and clk = '1') then
         if( load = '1' ) then   output <= input;
         end if;     end if;
  end process;
end regis_arc;
```

Simple FSM Design Example in VHDL

Design a FSM that output 1 iff the number of input sequence is odd




```
ENTITY FSM_Parity IS
        PORT (i1:       IN      std_logic;
              o1:       OUT     std_logic;
              CLK:      IN      std_logic; --Clock
              RST:      IN      std_logic  --Reset
); END;
```

Alternative: Less coding

```
ARCHITECTURE FSM_Parity_arch OF FSM_Parity IS
  TYPE      FSMStates IS (s1, s2);
  SIGNAL  State, NextState: FSMStates;
BEGIN

  PROCESS (State, i1) BEGIN
    CASE State IS
          WHEN s1 =>    if  i1='1' then    NextState <= s2;
                              else    NextState <= s1;
                        end if;
                        o1 <=  '0';
          WHEN s2 =>    if  i1='1' then    NextState <= s1;
                              else    NextState <= s2;
                        end if;
                        o1 <=  '1';
          WHEN OTHERS =>
                        o1 <=  '1';  NextState <= NextState;
    END CASE;
  END PROCESS;
```

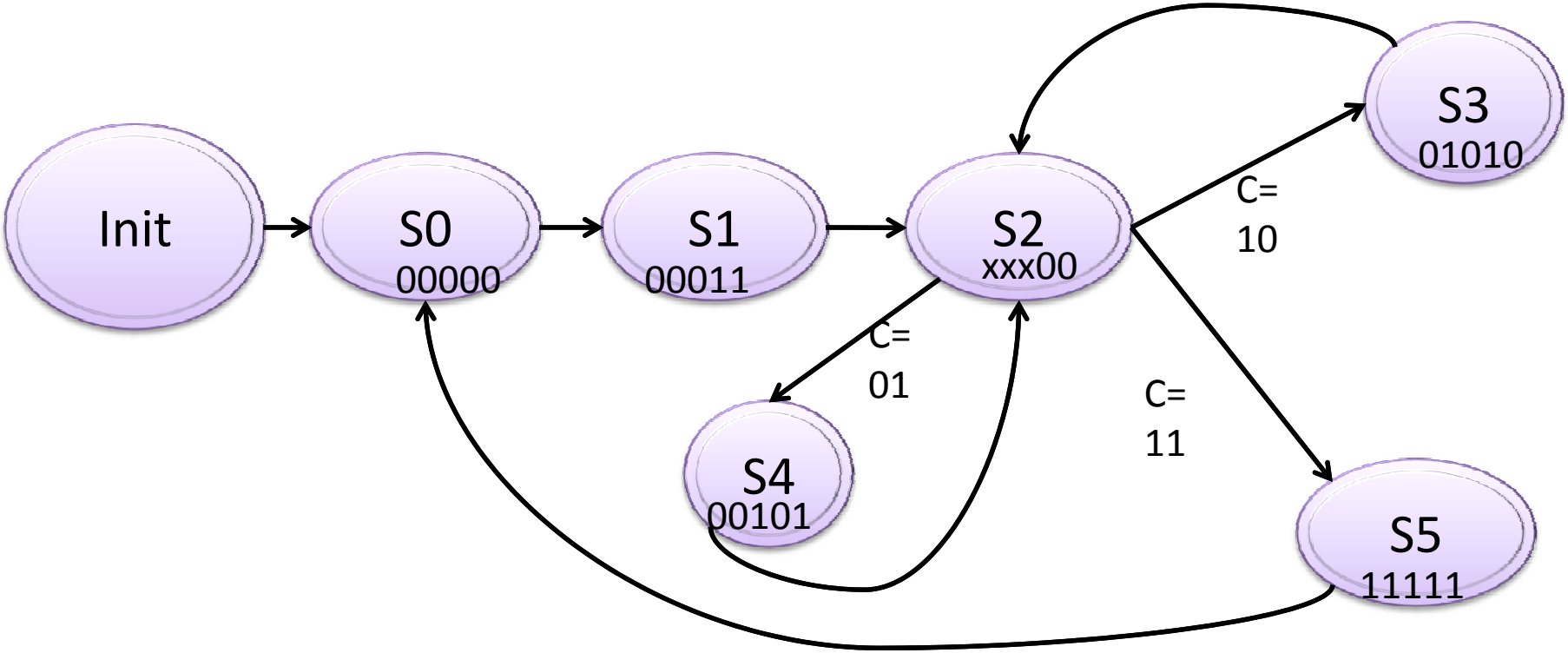
Important Note: every input to the state machine must be in the PROCESS sensitivity list

Important Note: every WHEN must assign the same set of signals: i.e. NextState and o1. if you miss one assignment latches will show up!

FSM controller: NextState Process

```
PROCESS (CLK, RST) BEGIN
        IF RST='1' THEN -- Asynchronous Reset
                State <= s1;

        ELSIF rising_edge(CLK) THEN
                State <= NextState;
        END IF;
END PROCESS;

END ARCHITECTURE;
```

How to Write FSM is VHDL




```
ENTITY fsm IS
   PORT(   rst, clk, proceed : IN std_logic;
           comparison: IN std_logic_vector( 1 DOWNTO 0 );
           enable, xsel, ysel, xld, yld: OUT std_logic
   );
END fsm;
```

FSM Architecture

```
ARCHITECTURE fsm_arc OF fsm IS
   TYPE states IS ( init, s0, s1, s2, s3, s4, s5 );
   SIGNAL nState, cState: states;
   BEGIN
   Process1: PROCESS( rst, clk )
             BEGIN
                IF( rst = '1' ) THEN
                   cState <= init;
                ELSIF( clk'event and clk = '1' ) THEN
                   cState <= nState;
                END IF;
             END PROCESS;
```

FSM Architecture: Continued…

```
Process2: process( proceed, comparison, cState )
  begin
    variable :  OP : std_loic_vector (4 downto 0);
    case cState is
    when init =>   if( proceed = '0‘)  then      nState <= init;
                          else          nState <= s0;         end if;
    when s0 =>     OP <=  “ 00000 “ ; nState <= s1;
    when s1 =>     OP <=  “ 00001”;  nState <= s2;
    when s2 =>     OP<=  “ XXX01” ;
          if( comparison = "10" ) then       nState <= s3;
               elsif( comparison = "01" ) then        nState <= s4;
               elsif( comparison = "11" ) then       nState <= s5;          end if;
    when s3 =>     OP <= “ 01010”  nState <= s2;
    when s4 =>     OP <=  “ 00101” ; nState <= s2;
    when s5 =>     OP <=  “11111” ;  nState <= s0;
    when others =>  nState <= s0;
    end case;
    enable <= OP(4);  xsel <= OP(3);ysel <= OP(2);xld <= OP(1);yld <= OP(0);
  end process;
end fsm_arc;
```

Advanced topic related to FPGA

- **How to use inbuilt components of FPGA?**
  - Block RAM, DSP Slices
  - Vivado/ISE use by default based on your HDL coding sytle
- **How to use PC to FPGA Communication : Via USB?**
  - So that, you can use the FPGA board as application accelerator
- **How to use other part of FPGA Board?**
  - Network, HDMI, AV Port and Codecs

Block RAM inferable code: RAM Example

```
ENTITY ram_example IS
   PORT (Clk : in std_logic;
         address : in integer;
         we : in std_logic;
         data_i : in BIT(7 DOWNTO 0);
         data_o : out BIT(7 DOWNTO 0)
         );
END ram_example;
```

Block RAM inferable code: RAM Example

```
ARCHITECTURE Behavioral OF ram_example IS
TYPE ram_t IS array (0 to 255) OF BIT(7 DOWNTO 0);
signal ram : ram_t := (others => (others => '0'));
   BEGIN
   PROCESS(Clk)
   BEGIN
      if(rising_edge(Clk)) then
         if(we='1') then  ram(address) <= data_i;
         end if;
         data_o <= ram(address);
      end if;
   END PROCESS;
   END Behavioral;
```

DSP Slices inferable code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

ENTITY FunctionExample IS
    Port (  x1,x2 : IN BIT(7 downto 0);
            C : OUT BIT (9 downto 0));
END FunctionExample;

ARCHITECTURE Behavioral OF FunctionExample IS
    constant A: integer:= 4;
    constant B: integer:= 2;
BEGIN
        C <=TO_SIGNED(A*x1 + B*x2,10);
END Behavioral;
```

VHDL Tutorial

- Forwarded By Frank Vahid: Digital Design

  http://esd.cs.ucr.edu/labs/tutorial/

- Google search “VHDL Tutorial: Learn by Example”

Thanks