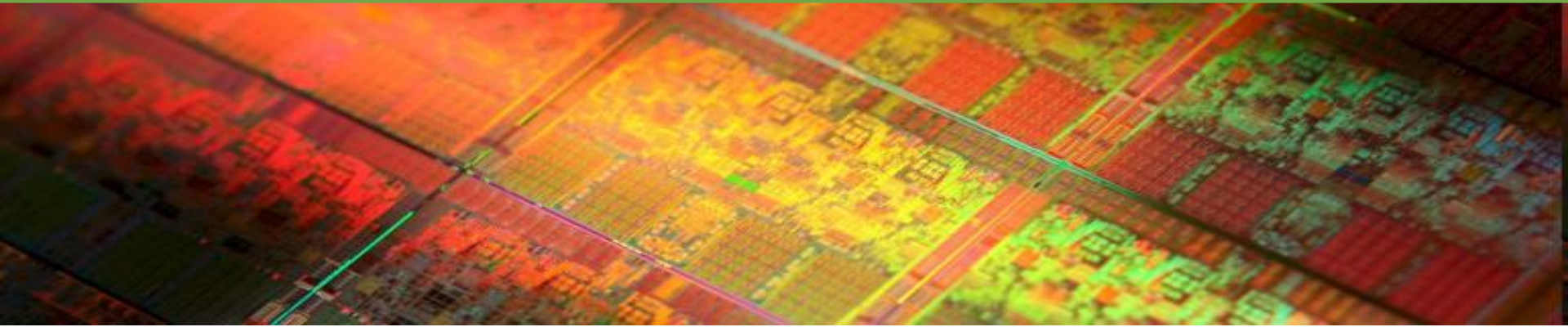


EE 2000

Logic Circuit Design (Fall 2018)



Lecture 2: Introduction to Hardware Description Language

Instructor: Dr. Ray C.C. Cheung

Outline

- Administrative issues
 - 3-week (VHDL + FPGA), 3-week (Sequential Logic by Dr. Lau), 4-week (VHDL + FPGA)
 - Lab 1, 2, 3, 4, 5 - Week 6, 8, 11, 12, 13
- Logic Design → Programmable Logic Design
- Tutorial Next week - Priority Encoder
 - Reset every Wednesday
- Make-up Lecture
 - Dec 4, 2018 (Tue)
 - 2-5pm
 - LT1, YEUNG

Outline

- *Introduction*
- VHDL Module
 - Entity & Architecture
 - Signal
- VHDL Data Type
- VHDL Operator
- VHDL Flow Control
- More VHDL Examples
- Summary

Introduction

- VHDL is a hardware description language (HDL).
 - We learn how to describe a hardware design.
 - Many other methods - Schematic
 - Why not schematic? Real life circuits are way too complex
- VHDL
 - VHSIC Hardware Description Language (VHDL)
 - VHSIC → Very High Speed Integrated Circuit
 - Widely used in industry, and University education
 - Developed in the 1980s under DoD sponsorship
 - Standardized by IEEE
 - VHDL leads naturally to a top-down design methodology.

Introduction

- VHDL is very similar to ADA programming language
 - Verilog is very similar to C
- VHDL Program is a collection of module blocks
 - Using Hierarchical design flow
 - Top-down approach
 - Modular-based approach
 - Team-work approach
 - Divide-and-conquer approach

Introduction

- With the allowance of more and more components on a chip, digital systems have continued to grow in complexity.
- Technology improvements have advanced the very large scale integration (VLSI) field. Despite changes in integration ability, the term VLSI remains popular.
- Early integrated circuits belonged to small-scale integration (SSI), medium-scale integration (MSI), or large-scale integration (LSI) groups depending on the density of integration.

Scale of Integration	Number of Transistors
SSI	$2 \approx 64$
MSI	$64 \approx 2000$
LSI	2000 - 64,000
VLSI	64,000-2,000,000
ULSI	2,000,000 - No limit

VHDL Basics

- Software Program - always being executed sequentially
 - C, Java, Python, Fortran, ...
- Hardware Program - Events offer concurrently
 - VHDL, Verilog, ...
- Efforts between the two ends
 - Handel-C, ...
 - Think of a software when design as a hardware
- VHDL is a strongly typed language
 - Case insensitive

Behavioral and Structural VHDL

- 3 Models:
 - **Behavioral:**
 - Specifies only the behavior at a higher level of abstraction.
 - Does not imply any particular structure or technology.
 - **Dataflow (Register Transfer Language [RTL]):**
 - Data path and control signals are specified.
 - System is described in terms of the data transfer between registers.
 - At an intermediate level of abstraction.
 - **Structural:**
 - Specifies more details.
 - Components used and the structure of the interconnection between the components are clearly specified.
 - At a low level of abstraction.

Methods To Describe Logic Circuits

- We have learned how to represent each of the basic logic functions using:
 - Logical statements in our own language
 - Truth tables
 - Traditional graphic logic symbols
 - Boolean algebra expressions
 - Timing diagrams

Description vs. Programming Languages

- It is important to distinguish between hardware description languages & programming languages
 - In both, a language is used to program a device.
- Computers operate by following a list of tasks, each of which must be done in sequential order.
 - Speed of operation is determined by how fast the computer can execute each instruction.
- A digital logic circuit is limited in speed only by how quickly the circuitry can change outputs in response to changes in the inputs.
 - It is monitoring all in-puts concurrently & responding to any changes.

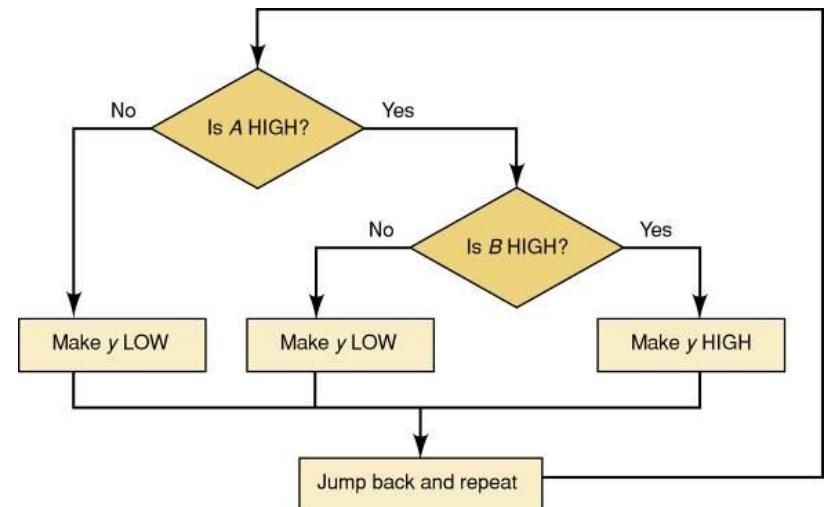
Description vs. Programming Languages

- Comparing operation of a computer and a logic circuit in performing the logical operation of $y = AB$.

Each shape in the flowchart represents one instruction.

If each takes 20 ns, it will take a minimum of two or three instructions (40-60 ns) to respond to changes in the inputs.

The computer must run a program of instructions that makes decisions.



VHDL Modules

- The general structure of a VHDL module is an **entity** description and an **architecture** description.
- **Entity** description declares the input and output signals.
- **Port** declaration specifies the inputs and outputs to the module.
- **buffer**: similar to **inout**, can be read and written into the entity, helpful if the signal is an output.
- **linkage**: useful when VHDL entities are connected to non-VHDL entities.

VHDL Modules

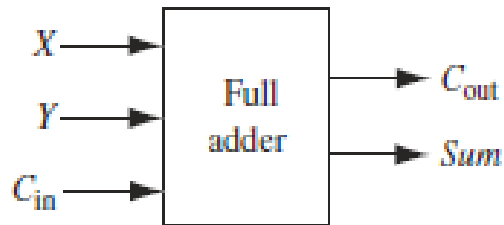
- The list-of-interface-signal-declaration has the form:

```
list-of-interface-signals: mode type [:= initial-value]  
{; list-of-interface-signals: mode type [:= initial-value]};
```

- The curly brackets indicate zero or more repetitions of the enclosed clause.
- **Mode** indicates the direction of information
- Use the **in**, **out**, and **inout** keywords to specify the direction of port signals.
- **Type** specifies the data type that can be communicated

VHDL Modules

- Example for a Full Adder:

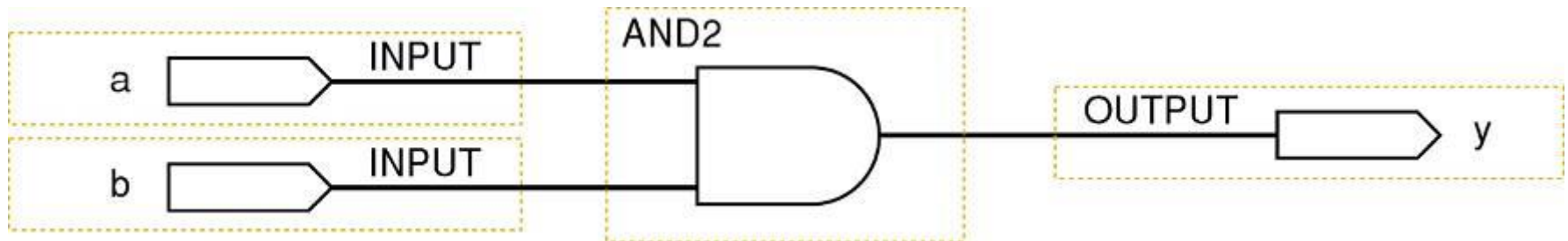


```
entity FullAdder is
  port(X, Y, Cin: in bit;      --Inputs
        Cout, Sum: out bit);  --Outputs
end FullAdder;
```

- In this example, the VHDL assignment statements for Sum and C_{out} represent the logic equations for the full adder.

HDL Format and Syntax

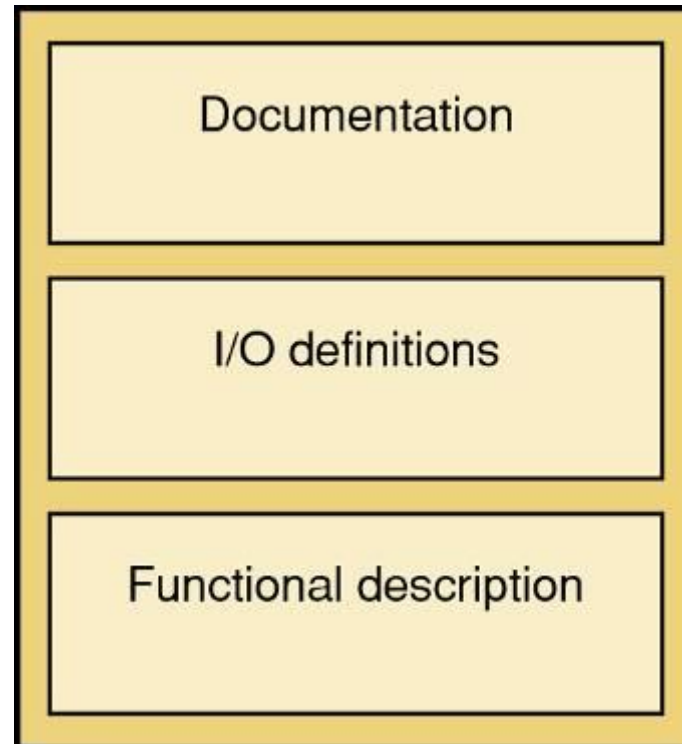
- Languages that are interpreted by computers must follow strict rules of syntax.
 - Syntax refers to the order of elements.
- On the left side of the diagram is the set of inputs, and on the right is the set of outputs.
- The symbols in the middle define its operation.



HDL Format and Syntax

- Format refers to a definition of inputs, outputs & how the output responds to the input (operation).

Format of HDL
files.



HDL Format and Syntax

- In a text-based language, the circuit described must be given a name.
- The definition of the operation is contained in a set of statements that follow the input/output (I/O) definition.
- Inputs & outputs (ports) must be assigned names and defined according to the nature of the port.
- The mode defines whether it is input, output, or both.

VHDL Description of Combinational Circuits

- VHDL models combinational circuits using concurrent statements.
- Concurrent statements:
 - Statements that are always ready to execute.
 - Evaluated any time and every time a signal on the right side of the statement changes.
 - Execute repeatedly as if they were in a loop.



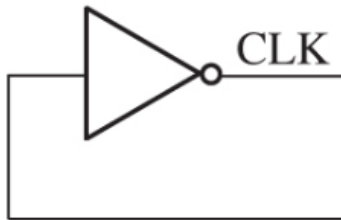
```
C <= A and B after 5 ns;  
E <= C or D after 5 ns;
```

VHDL Description of Combinational Circuits

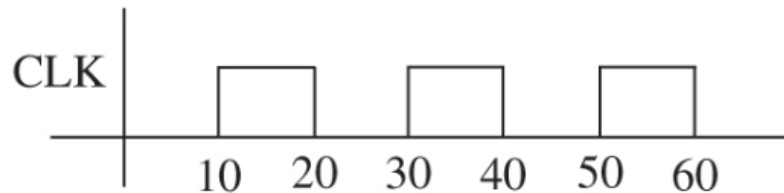
- A signal assignment statement has the form:
 - `signal_name <= expression [after delay];`
 - Expression is evaluated when the statement is executed
 - Signal on the left changes after `delay`
 - Square brackets indicate that **after** `delay` is optional
- The hardware description language defines the connections to be made.
 - It is loaded into the device after translation by a compiler.
- The higher-level hardware description language, makes programming the PLDs much easier than trying to use Boolean algebra, schematic drawings, or truth tables.

Inverter with Feedback

- This is how we define a clock signal in VHDL.

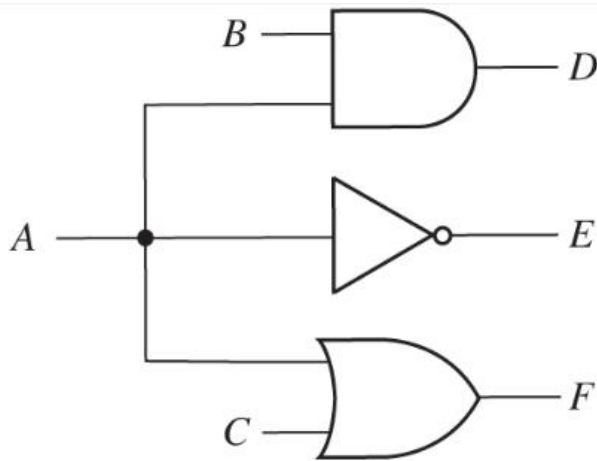


```
CLK <= not CLK after 10 ns;
```



Common Input for 3 Gates

- Concurrent statements all execute at the same time.



```
-- when A changes, these concurrent  
-- statements all execute at the  
-- same time  
D <= A and B after 2 ns;  
E <= not A after 1 ns;  
F <= A or C after 3 ns;
```

bit_vector(3 downto 0)

- B: in bit_vector(3 downto 0);
- B <= "1100"
- Assign '1' to B(3)
- Assign '1' to B(2)
- Assign '0' to B(1)
- Assign '0' to B(0)

Array of AND Gates

- Two different ways, which one you prefer?



-- the hard way

```
C(3) <= A(3) and B(3);
```

```
C(2) <= A(2) and B(2);
```

```
C(1) <= A(1) and B(1);
```

```
C(0) <= A(0) and B(0);
```

-- the easy way assuming C, A and
-- B are 4-bit bit-vectors

```
C <= A and B;
```

Outline

- Introduction
- ***VHDL Module***
 - Entity & Architecture
 - Signal
- VHDL Data Type
- VHDL Operator
- VHDL Flow Control
- More VHDL Examples

Design Entity

- Provide a Block Box view of a module

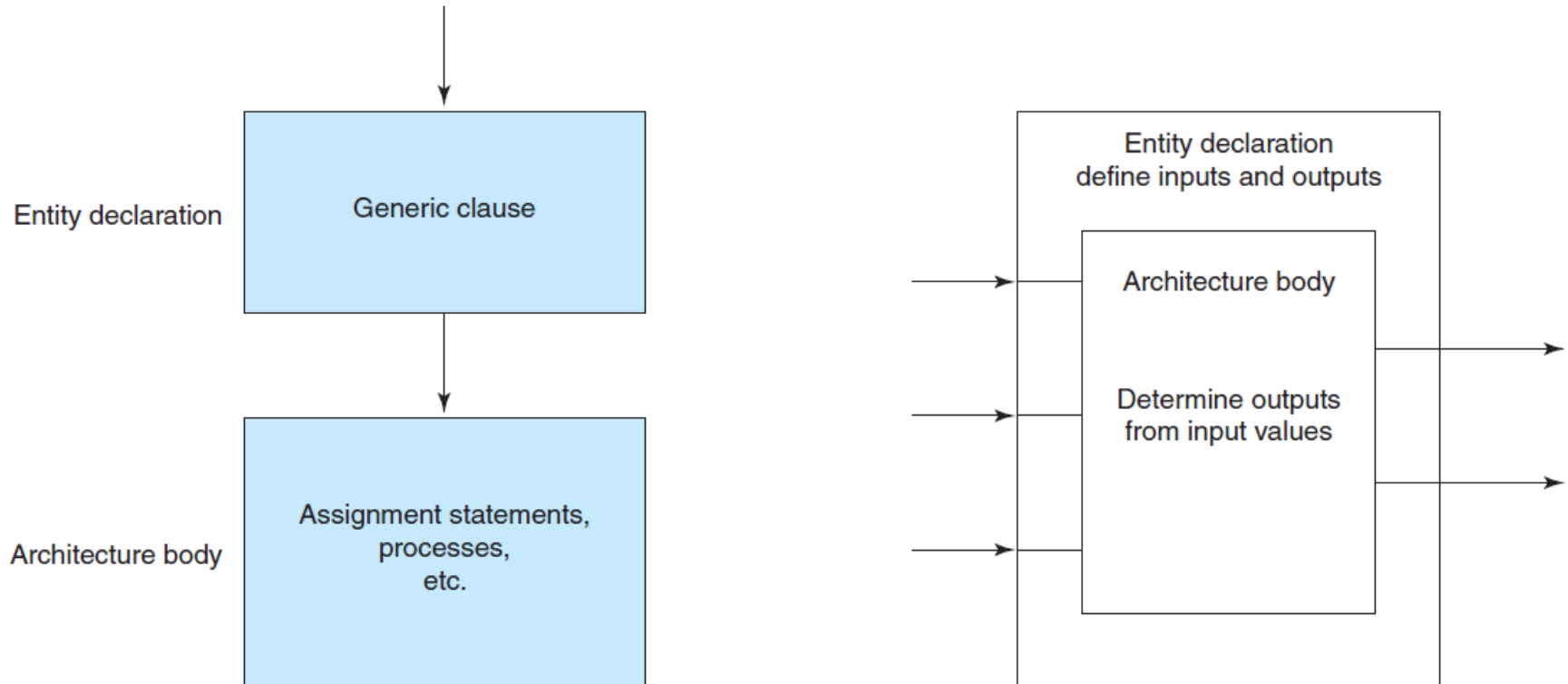


Figure shows two ways to graphically portray the relationship between the entity declaration and the architecture body of a design entity.

Entity Declaration

- Entity declaration structure
 - First line contains identifier
 - Followed by port clause and sometimes a generic clause
 - Ends by repeating name of entity

entity exp01 is

Port (

sw: IN BIT;
led: OUT BIT

);

end exp01;

Entity Declaration

- Port clause
 - Declares:
 - Port names
 - Port modes
 - Types of data ports will pass
 - Does not declare pin numbers

entity exp01 is

Port (

sw: IN BIT;
led: OUT BIT

);

end exp01;

HDL Format and Syntax - VHDL

- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase

```
ENTITY and_gate IS
PORT (    a, b    :IN BIT;
        y        :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
    y <= a AND b;
END ckt;
```

The keyword PORT tells the compiler that we are defining inputs and outputs to this circuit block.

HDL Format and Syntax - VHDL

- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase.

```
ENTITY and_gate IS
PORT (   a, b   :IN BIT;
        y       :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
    y <= a AND b;
END ckt;
```

The BIT
description tells
the compiler that
each variable in
the list is a single
bit.

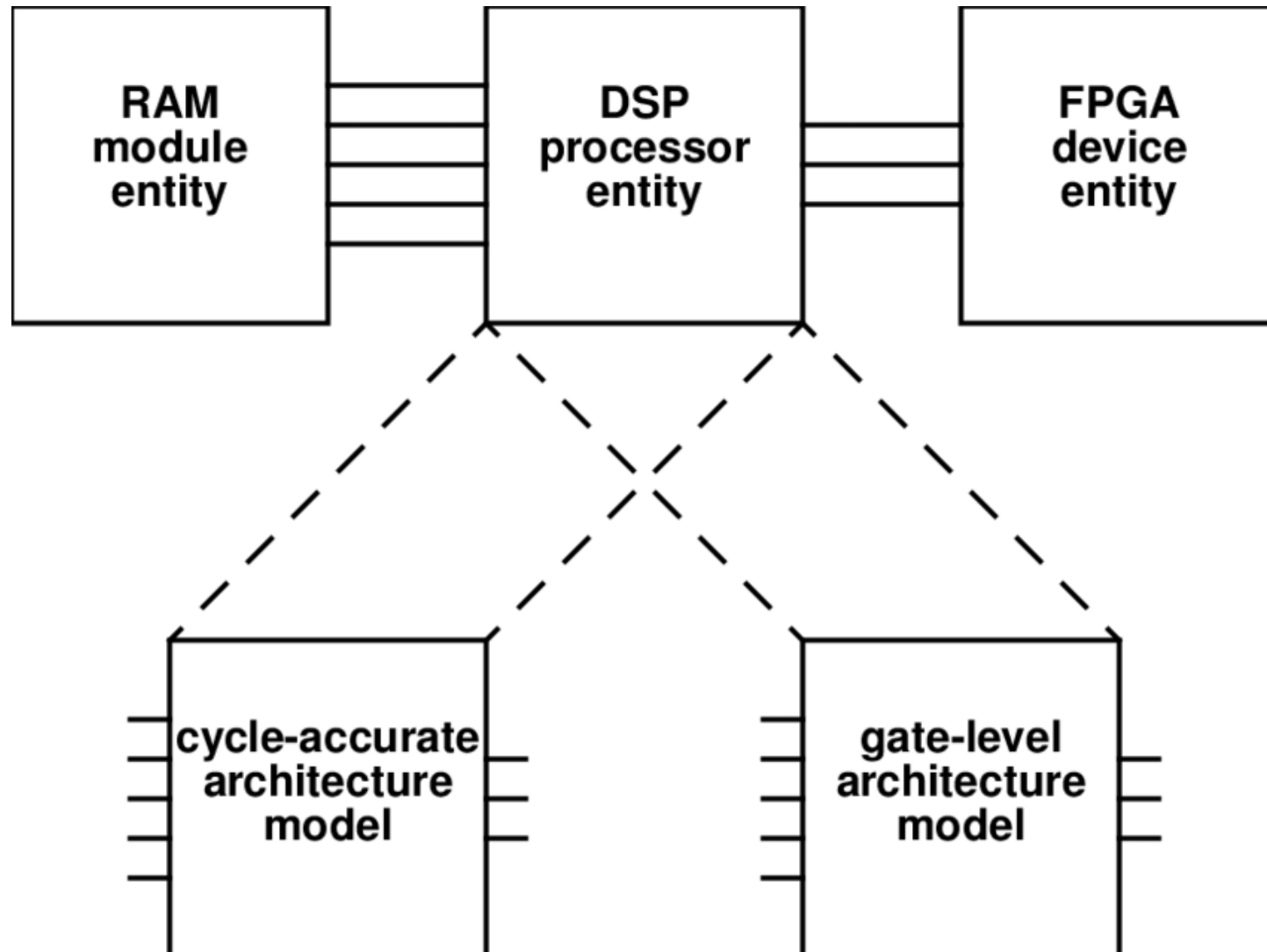
HDL Format and Syntax - VHDL

- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase.

```
ENTITY and_gate IS
PORT (   a, b   :IN BIT;
        y       :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
    y <= a AND b;
END ckt;
```

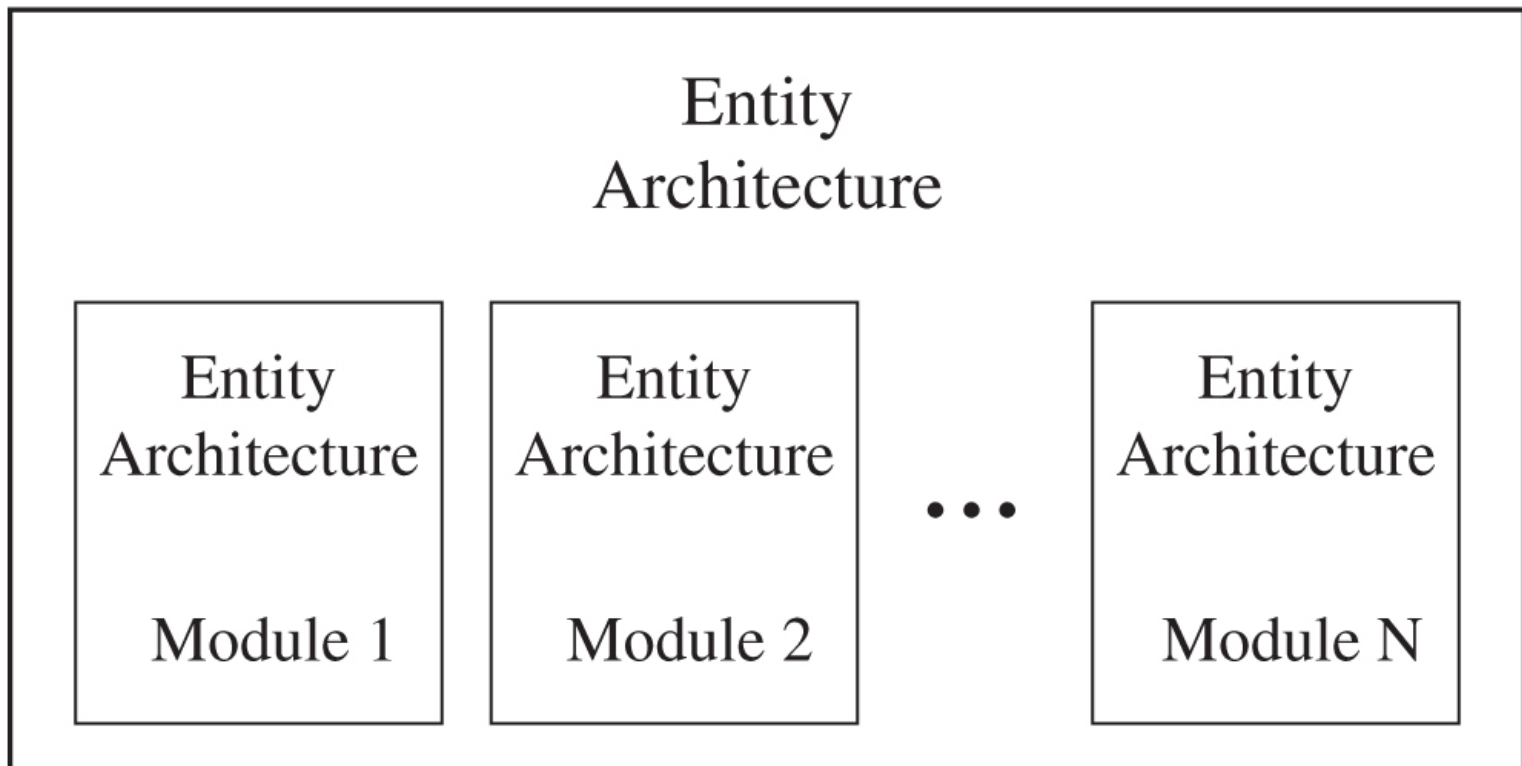
The ARCHITECTURE declaration is used to describe the operation of everything inside the block. Every ENTITY must have at least one ARCHITECTURE associated with it.

Multiple Architecture Block



Multiple-level of Entity

- Top level design
- Each component module is part of the system



HDL Format and Syntax - VHDL

- The keyword ENTITY gives a name to the circuit block, which in this case is and_gate. Variables named by the compiler should be lowercase.

```
ENTITY and_gate IS
PORT (    a, b    :IN BIT;
        y        :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
    y <= a AND b;
END ckt;
```

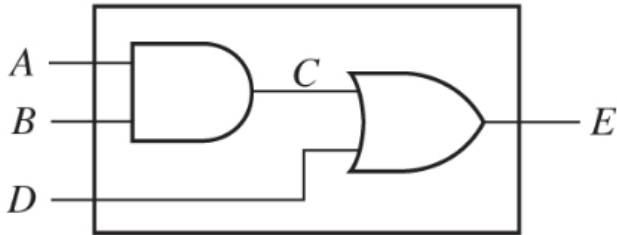
Within the body (between BEGIN and END) is the description of the block's operation.

Architecture Body

- First line contains architecture name (identifier) and name of design entity
- Architecture declarative part is between first line and BEGIN line
- After BEGIN line is architecture statement part
- Ends with END and name of architecture

Example - Two Gates

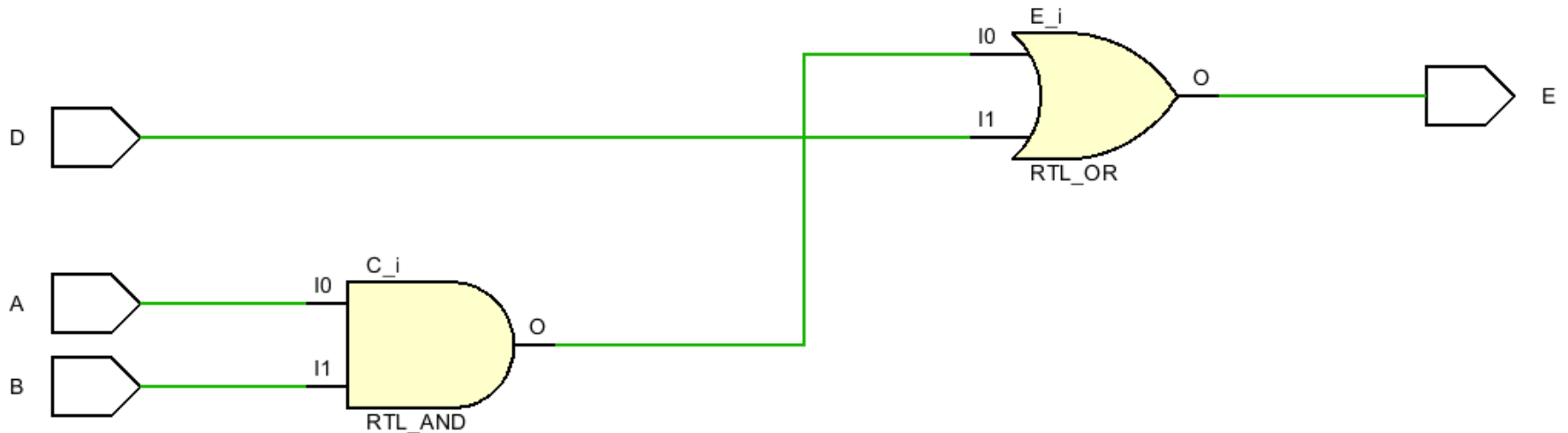
- Check the full description of this design



```
entity two_gates is  
    port(A, B, D: in bit; E: out bit);  
end two_gates;  
  
architecture gates of two_gates is  
    signal C: bit;  
    begin  
        C <= A and B; -- concurrent  
        E <= C or D; -- statements  
    end gates;
```

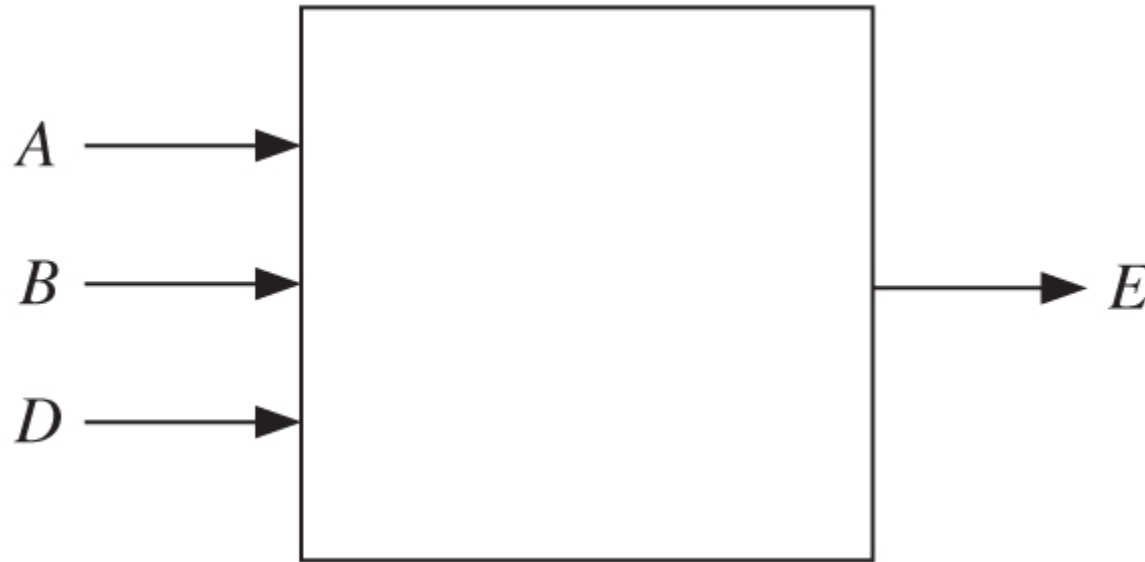
Schematic

- You can also use your Vivado software to synthesize it



Black Box View

- Any entity can be considered as a black box picture of a module being designed, and its external interface.



VHDL Signals

- Simple signal assignment
 - Target <= Waveform
 - Target is output that can be a port or signal
 - Waveform is logic value or Boolean expression
 - Single quotes used for logic values; double quotes used for groups or bits or bit vectors
 - ' x ' or " x "?
 - Can be listed in any order in architecture body

VHDL Signals

```
signal_name <= [delay_mechanism ] waveform ;
```

```
signal_name <= [delay_mechanism ] waveform1 when condition1 else  
    [delay_mechanism ] waveform2 when condition2 else  
    . . .  
    [delay_mechanism ] waveformn;
```

with selection select

```
signal_name <= [delay_mechanism ] waveform1 when choice1,  
    [delay_mechanism ] waveform2 when choice2,  
    . . .  
    [delay_mechanism ] waveformn when others;
```

VHDL Signals

- Conditional signal assignment
 - Uses WHEN and ELSE
 - Target output can be port or signal; waveform can be logic or Boolean expression
 - Cannot be used in a process or inside an IF statement
 - NULL can be used where there is a condition, but cannot be used as default value at end of statement

VHDL Signals

- Selected signal assignment
 - Uses WITH, SELECT, WHEN, and WHEN OTHERS
 - Target output can be port or signal, waveform values must be fixed logic values
 - Cannot appear inside a process or IF statement
 - NULL can appear as default value

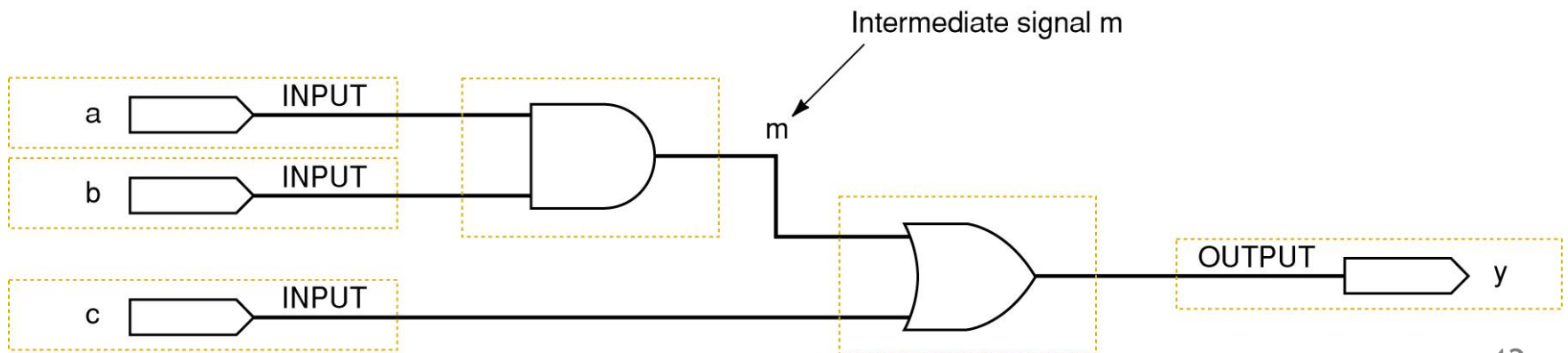
```
architecture Conditional of TriStateBuffer is  
begin  
    BufOut <= BufIn when Enable = '1'  
    else 'Z';  
end architecture Conditional;
```

VHDL Signals

- Signals
 - Work like a buffer type port
 - Strictly an internal quantity not directly accessible outside of the design entity
 - Normally declared in architecture declarative part
 - Allows us to make connections between functions and processes
- Signals
 - Declaration of the form:
`signal list_of_signal_names: type_name [ := initial_value];`
 - Must be declared outside a process.
 - Can be used anywhere within the architecture if declared at the start.
 - Updated after a delay using:
`signal_name <= expression [after delay];`

Intermediate Signals

- In many designs, there is a need to define signal points “inside” the circuit block—called buried nodes or local signals.
 - Points in the circuit that may be useful as a reference point, that are not inputs or outputs.

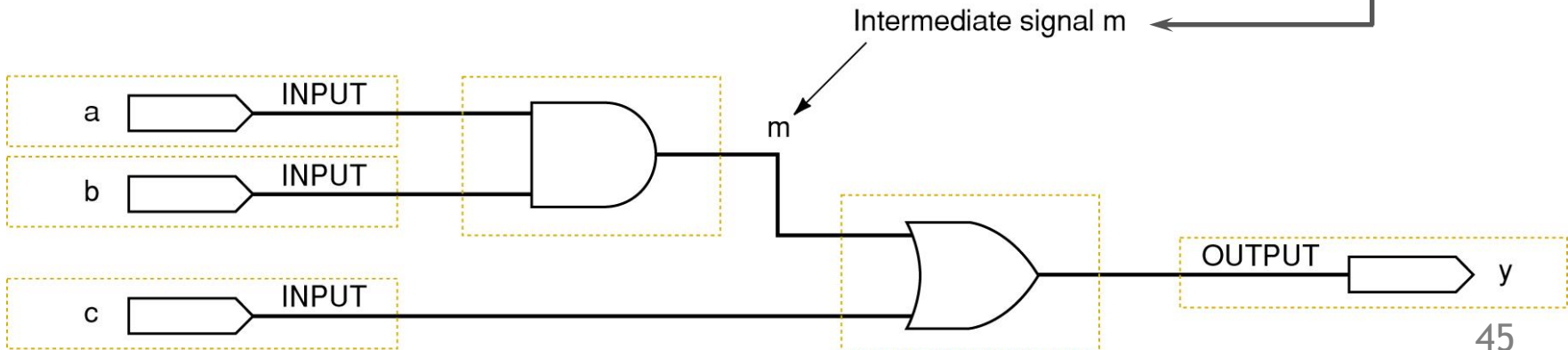


Intermediate Signals - VHDL

- VHDL local signals:
 - Text after two dashes is for documentation only.
 - -- Example Comments
 - Keyword SIGNAL defines intermediate signal.
 - Keyword BIT designates the type of signal

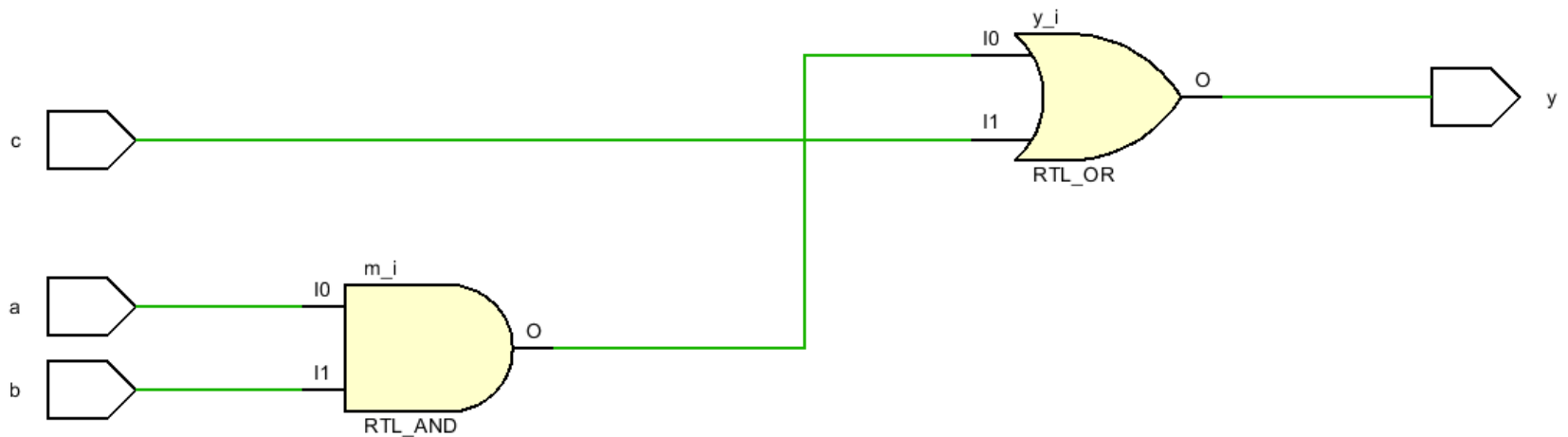
Intermediate Signals - VHDL

```
25 entity Signal_Example is
26   -- Port ( );
27   port (
28       a, b, c: in bit;  -- 3 inputs
29       y:      out bit
30   );
31 end Signal_Example;
32
33 architecture Behavioral of Signal_Example is
34
35     signal m: bit;
36 begin
37     m <= a AND b;
38     y <= m OR c;
39 end Behavioral;
```



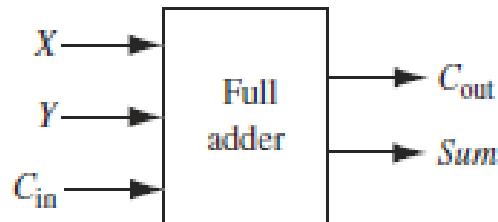
Schematic Design

- You can open it from -> Elaborated Design in Vivado IDE.



Recall: VHDL Modules

- Example for a Full Adder:



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

- In this example, the VHDL assignment statements for Sum and C_{out} represent the logic equations for the full adder.

Example: Architecture Declaration

- Architecture name “Equations” is arbitrary
- The entity name “FullAdder” must match the name of the associated entity declaration

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

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

architecture Equations of FullAdder is
begin    -- concurrent assignment statements
    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;
```

Example: TestBench

```
entity FullAdder_tb is
-- Port ( );
end FullAdder_tb;

architecture Behavioral of FullAdder_tb is
    component FullAdder
        port (X, Y, Cin: in bit;    -- Inputs
              Cout, Sum: out bit); -- Outputs
    end component;

    signal X, Y, Cin: bit;
    signal Cout, Sum: bit;
begin

    uut: FullAdder port map (X, Y, Cin, Cout, Sum);

    sim: process
    begin
        X <= '0';
        Y <= '0';
        Cin <= '0';

        wait for 100 ns;
        X <= '1';

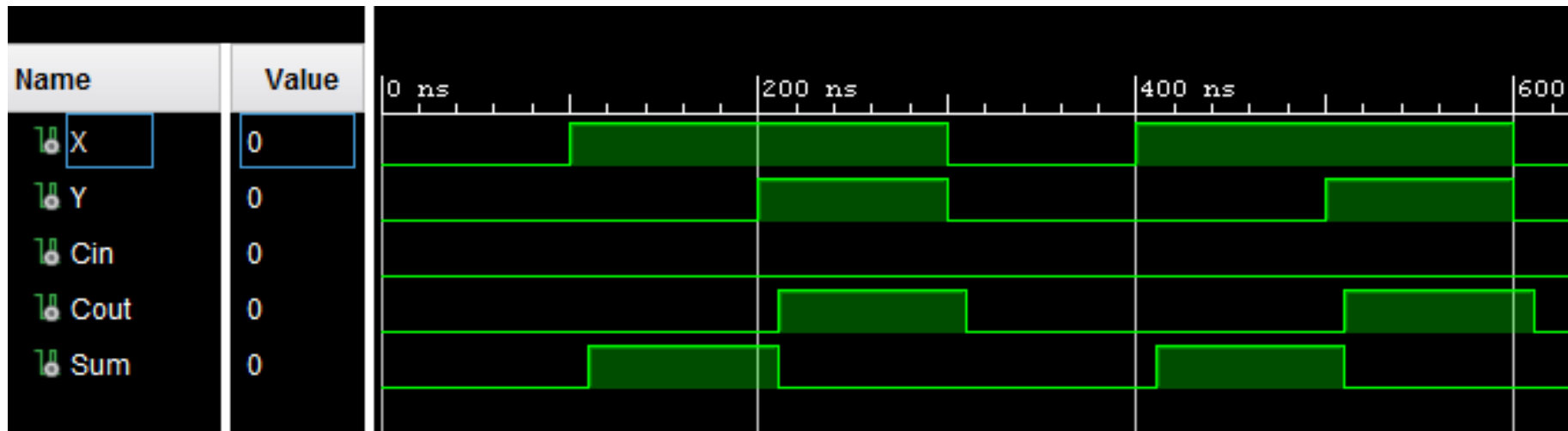
        wait for 100 ns;
        Y <= '1';

        wait for 100 ns;
    end process;
end Behavioral;
```

- Empty Entity block
- Check the component name
- Check the signal name
- UUT is correctly port mapped
- What is the waveform?

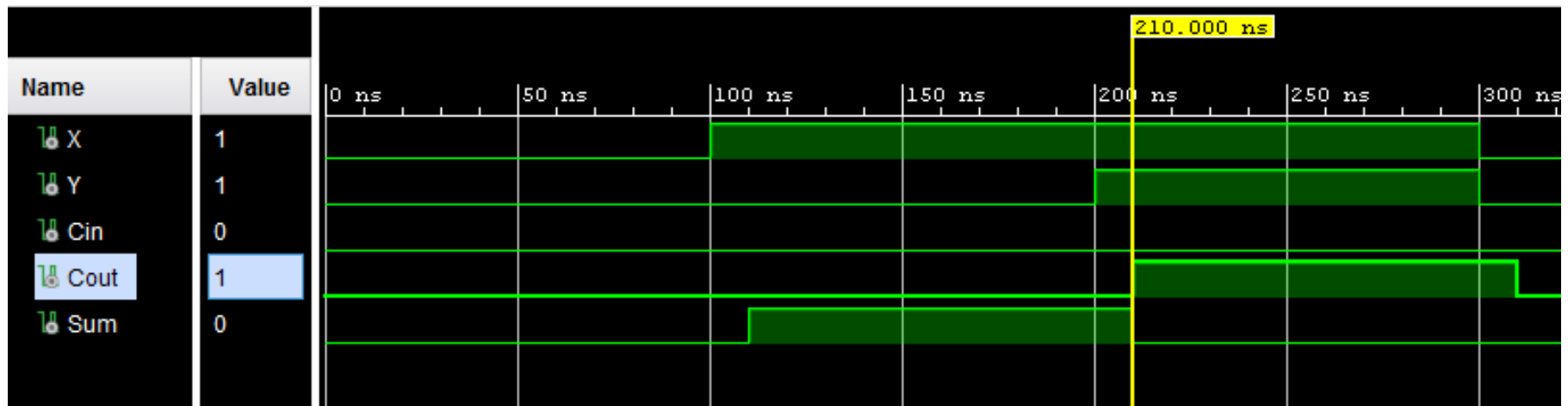
Waveform

- Is this what you expected?
- Do you see the 10 ns delayed?



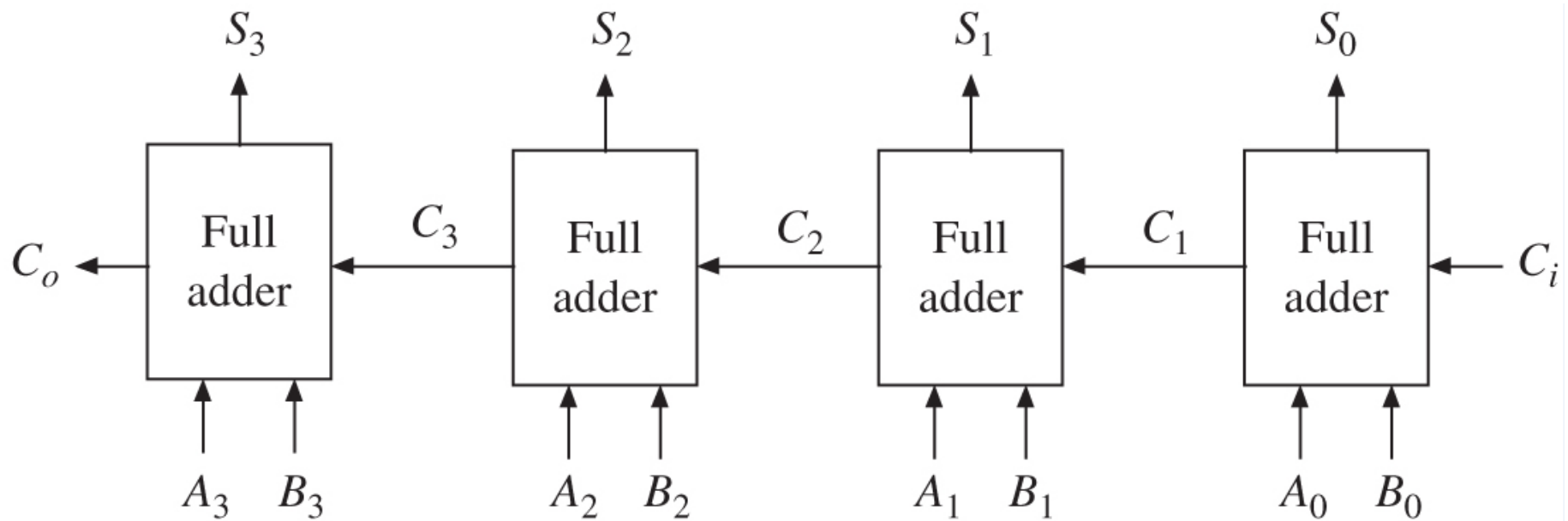
Simulated Delay

- You can select any delay that you want.
- 100 ns -> X = '1'
- 200 ns -> Y = '1'



4-bit Full Adder

- How can we design a 4-bit binary adder?



VHDL code

- Structural VHDL Design
- Why Signal C $\rightarrow$ (3 downto 1)?

```
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); -- C is an internal signal
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;
```

Cannot Compile Code

- Statement 2 uses D on the right side of the assignment, input or output?

entity gates **is**

```
    port(A, B, C: in bit; D, E: out bit);  
end gates;
```

architecture example **of** gates **is**

begin

```
    D <= A or B after 5 ns;  -- statement 1  
    E <= C or D after 5 ns; -- statement 2  
end example;
```

Use of “Buffer” Mode

- Either “inout” or “buffer” will work

entity gates is

```
    port(A, B, C: in bit; D: buffer bit; E: out bit);  
end gates;
```

architecture example of gates is

begin

```
    D <= A or B after 5 ns; -- statement 1  
    E <= C or D after 5 ns; -- statement 2  
end example;
```

Outline

- Introduction
- VHDL Module
 - Entity & Architecture
 - Signal
- ***VHDL Data Type***
- VHDL Operator
- VHDL Flow Control
- More VHDL Examples

VHDL Data Types

- Data types:
 - Predefined types:

Type	Description
bit	'0' or '1'
boolean	FALSE or TRUE
integer	an integer in the range $-(2^{31} - 1)$ to $+(2^{31} - 1)$
real	floating-point number in the range $-1.0E38$ to $+1.0E38$
character	any legal VHDL character including upper and lowercase letters, digits, and special characters
time	An integer with units fs, ps, ns, us, ms, sec, min, or hr

- Can also have user-defined types
- bit_vector, string

Example: std_logic type

- 'U': uninitialized, hasn't been set value yet
- 'X': unknown, unable to determine value/result
- '0': logic 0
- '1': logic 1
- 'Z': High Impedance
- 'W': Weak signal, can't tell if it should be 0 or 1.
- 'L': Weak signal that should probably go to 0
- 'H': Weak signal that should probably go to 1
- '-': Don't care.

Outline

- Introduction
- VHDL Module
 - Entity & Architecture
 - Signal
- VHDL Data Type
- ***VHDL Operator***
- VHDL Flow Control
- More VHDL Examples

VHDL Operators

- VHDL Operators:
 1. Binary logical operators: **and or nand nor xor xnor**
 2. Relational operators: **= /= < <= > >=**
 3. Shift operators: **sll srl sla sra rol ror**
 4. Adding operators: **+ - &** (concatenation)
 5. Unary sign operators: **+ -**
 6. Multiplying operators: *** / mod rem**
 7. Miscellaneous operators: **not abs ****
- Class 7 has highest precedence, then 6, 5, 4, ...
- Operators in the same class are applied left to right
- Parentheses change the order of precedence

Operator Example

- You need to know how to define a signal.
- Can you define: $A \leq "1010010"$?

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

entity Operator_tb is
-- Port ( );
end Operator_tb;

architecture Behavioral of Operator_tb is

    signal A, B, C, D, E, F, G: bit_vector(7 downto 0);

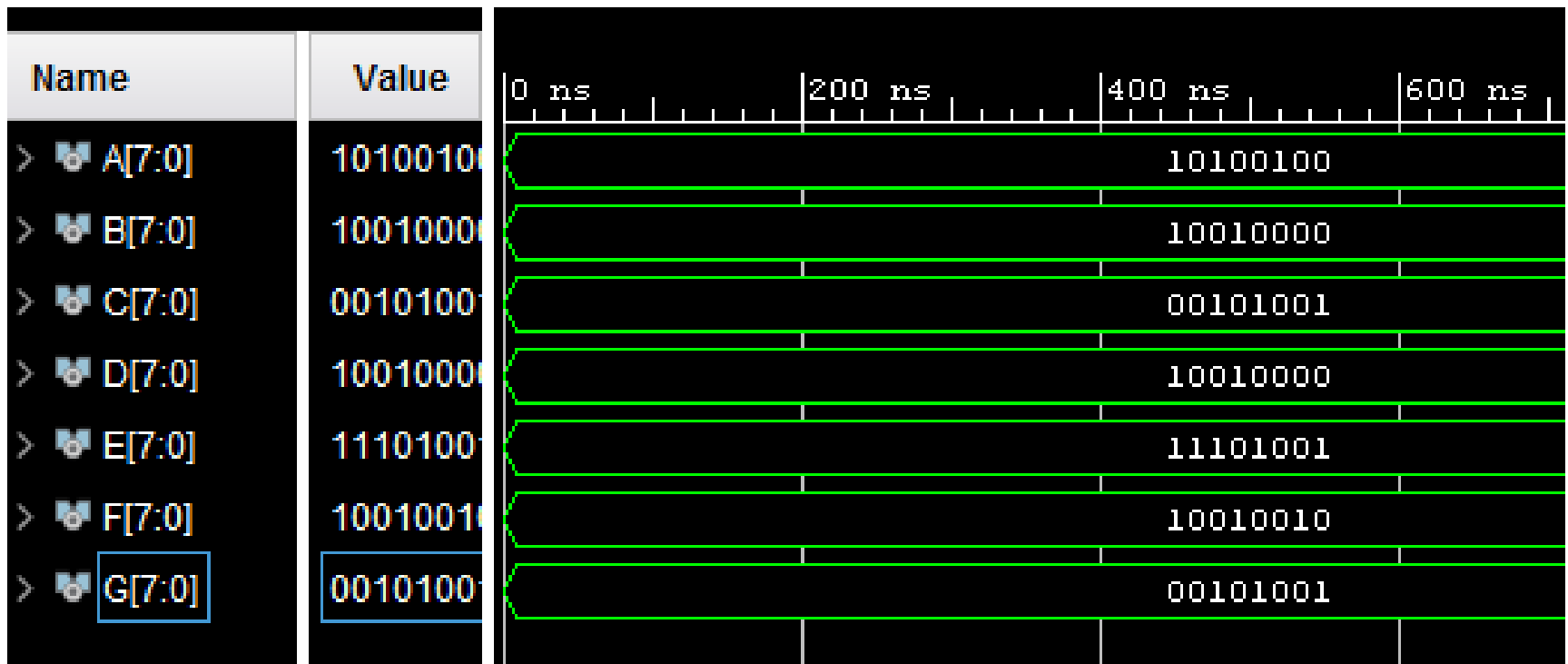
begin

    A <= "10100100";
    B <= A sll 2;
    C <= A srl 2;
    D <= A sla 2;
    E <= A sra 2;
    F <= A rol 2;
    G <= A ror 2;

end Behavioral;
```

Waveform

- Do you expect the same result?



Constants, Generic Constants, Signals, and Variables

TYPE	PREFIX	EXAMPLES
Constant	con	conN, conLast
Generic	gen	genX, genMax
Signal	sig	sigA, sigConnector1
Variable	var	varL, varCounter

Table shows the Hungarian notation used for naming constants, generics, signals, and variables.

VHDL Constants

- Constants

- Fixed-value quantity
- Can be declared in architecture declarative part or in process declarative part
- General format:
 - `CONSTANT <conName>: <type>:= <value>;`

- Constants

- Common declaration form:

```
constant constant_name: type_name := constant_value;
```
- Constants declared at the start of an architecture can be used anywhere in that architecture.
- Constants declared within a process are local to that process.

Constant Example

```
53 signal seqcount :    unsigned(10 downto 0) := "000000000000";
54 signal Eng_Sck_en :  STD_LOGIC := '0';
55 signal clkdiv2 :      STD_LOGIC := '0';
56 signal Eng_State :    STD_LOGIC_VECTOR(2 downto 0) := "000";
57     CONSTANT IDLE :      std_logic_vector(2 downto 0) := "000";
58     CONSTANT OUTPUT_SRAM_HEADER : std_logic_vector(2 downto 0) := "001";
59     CONSTANT SET_LRCK_HIGH :      std_logic_vector(2 downto 0) := "010";
60     CONSTANT CLEAR_LRCK_HIGH :    std_logic_vector(2 downto 0) := "011";
61     CONSTANT GET_CHANNEL_DATA :   std_logic_vector(2 downto 0) := "100";
62     CONSTANT WAIT_FOR_SAMPLE :    std_logic_vector(2 downto 0) := "101";
63     CONSTANT CHECK_SAMPLE_COUNT : std_logic_vector(2 downto 0) := "110";
64     CONSTANT CHECK_FOR_IDLE :     std_logic_vector(2 downto 0) := "111";
65 signal CycleCnt :      unsigned(10 downto 0) := "111111111111";
66 -----
67 -- SPI Latch signals
68 -----
69 signal SPIbitCnt :      unsigned(2 downto 0) := "000";
70 signal InputReg :       std_logic_vector(7 downto 0) := "00000000";
71 signal NewInput :       STD_LOGIC := '0';
72
73 signal Run :            STD_LOGIC := '0';
74
75 CONSTANT DE_SELECT :    std_logic_vector(1 downto 0) := "11";  -- No select
76 CONSTANT ADC_CS_LOW :   std_logic_vector(1 downto 0) := "00";  -- ADC_CS low
77 CONSTANT nvSRAM1_CS_LOW : std_logic_vector(1 downto 0) := "01";  -- nvSRAM1_CS low
78 CONSTANT CPLD_SELECT :  std_logic_vector(1 downto 0) := "10";  -- CPLD selected
```

Generic Constants

- Generic constants or “Generics”
 - Value can be used throughout architecture body and in port clause
 - Value can be changed by another design entity
 - Value given in VHDL code is default value
 - Structure is similar to port clause
 - Suitable for defining bit vector lengths

```
LIBRARY ieee;
USE ieee.NUMERIC_STD.all; -- required to use "unsigned" type

Entity generic_multiplier is
    generic (N : integer := 4 );
    port(A, B: in unsigned(N-1 downto 0);
         Z : out unsigned(2*N-1 downto 0) );
End entity;

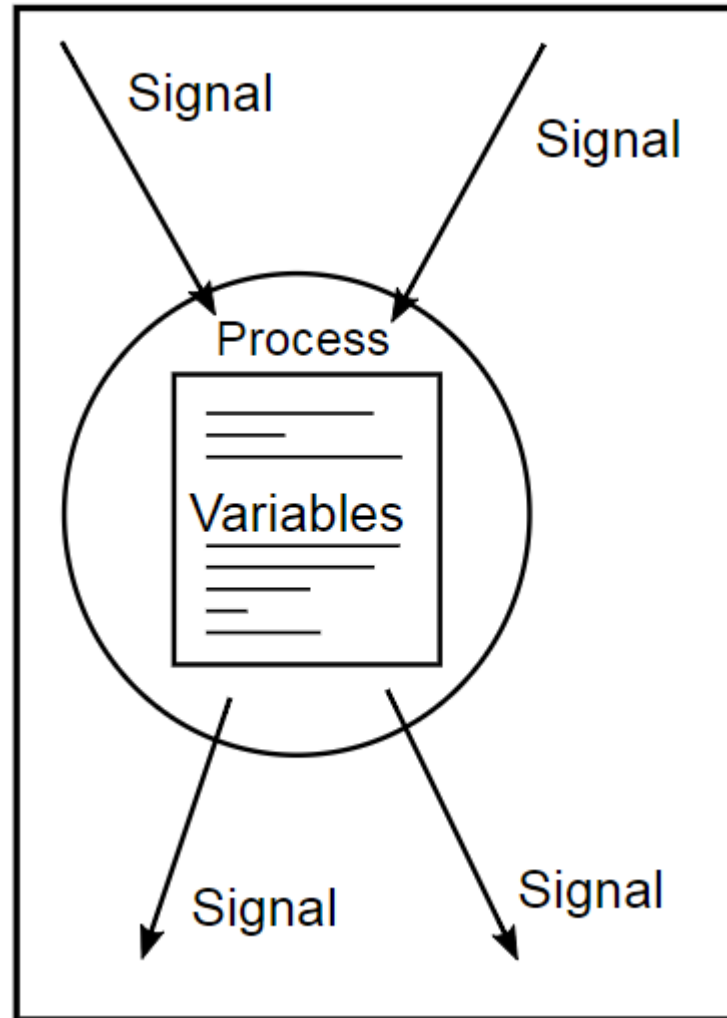
Architecture behave of generic_multiplier is
Begin
    Z <= A * B;
End architecture;
```

VHDL Variables

- Variables
 - Internal storage quantity that is sequential by nature
 - Can only be defined in process declarative part
 - Value is changed immediately when encountered in process
- Variables
 - Declaration of the form:
`variable list_of_variable_names: type_name [ := initial_value];`
 - Must be declared inside a process.
 - Are local to the process.
 - Updated instantly using:
`variable_name := expression;`

Variable Concept

- Store the value until the next process call.
- Revisit this part later on when we talk about process.



Variable Example

- Check the position of Variable

```
22 entity PARITY is
23     port (DATA: in bit_vector (3 downto 0);
24           ODD : out bit);
25 end PARITY;
26
27 architecture RTL of PARITY is
28 begin
29     process (DATA)
30         variable TMP : bit;
31     begin
32         TMP := '0';
33         for I in DATA'low to DATA'high loop
34             TMP := TMP xor DATA(I);
35         end loop;
36         ODD <= TMP;
37     end process;
38 end RTL;
```

Bit Slice and Concatenation

- Bit slicing allows us to separate out part of a bit vector into a smaller bit vector or individual bit
- Concatenation allows us to construct a bit vector from individual bits and/or smaller bit vectors
- Applies to ports, constants, generics, signals, and variables

Example

```
22 entity CONCATENATION is
23 end CONCATENATION;
24
25 architecture EXAMPLE_1 of CONCATENATION is
26     signal BYTE : bit_vector (7 downto 0);
27     signal A_BUS, B_BUS : bit_vector (3 downto 0);
28
29     begin
30         BYTE    <= A_BUS & B_BUS;
31     end EXAMPLE_1;
32
33 architecture EXAMPLE_2 of CONCATENATION is
34     signal Z_BUS : bit_vector (3 downto 0);
35     signal A_BIT, B_BIT, C_BIT, D_BIT : bit;
36
37     begin
38         Z_BUS    <= A_BIT & B_BIT & C_BIT & D_BIT;
39     end EXAMPLE_2;
```

Representing Data in HDL

- Every programming language & HDL has its own unique way of identifying number systems.
 - Generally done with a prefix to indicate the system.
- When we read one of these number designations, we must think of it as a symbol that represents a binary bit pattern.

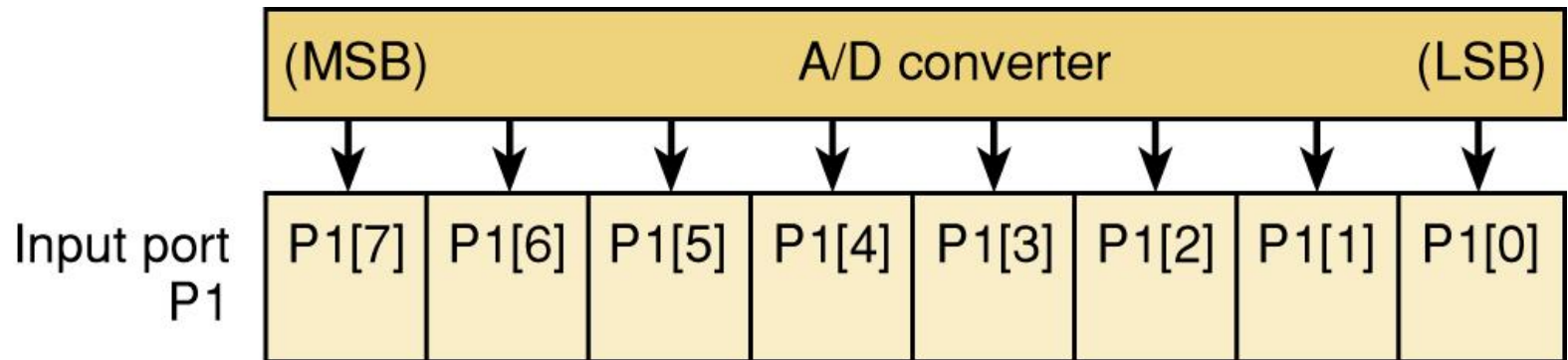
Representing Data in HDL

- These numeric values are referred to as scalars or literals.

Number System	AHDL	VHDL	Bit Pattern	Decimal Equivalent
Binary	B"101"	B"101"	101	5
Hexadecimal	H"101"	X"101"	100000001	257
Decimal	101	101	1100101	101

- In order to describe a port with more than one data bit we assign a name and the number of bits. This is called a bit array or bit vector.

Representing Data in HDL



- Each element (bit) has a unique index number (0-7) to describe position in the overall structure.
 - HDLs & computer programming languages use this notation.

Example Slices

```
--
22 entity SLICES is
23 end SLICES;
24
25 architecture EXAMPLE of SLICES is
26     signal BYTE : bit_vector (7 downto 0);
27     signal A_BUS : bit_vector (3 downto 0);
28     signal Z_BUS : bit_vector (3 downto 0);
29     signal A_BIT : bit;
30 begin
31
32     BYTE (5 downto 2) <= A_BUS;
33     BYTE (5 downto 0) <= A_BUS;      -- wrong
34
35     Z_BUS (1 downto 0) <= '0' & A_BIT;
36     Z_BUS <= BYTE (6 downto 3);
37     Z_BUS (0 to 1) <= '0' & A_BIT;  -- wrong
38
39     A_BIT <= A_BUS (0);
40
41 end EXAMPLE;
```

Representing Data in HDL - VHDL Syntax

- Intermediate variables can be declared as an array of bits—in the ARCHITECTURE section

Eight-bit temperature port *p1* assigned to a signal named *temp*...

```
SIGNAL temp :BIT_VECTOR {7 DOWNT0 0};  
BEGIN  
    temp <= p1;  
END;
```

When no elements in the bit vector are specified,
all bits in the array are being connected.
Individual bits could be connected by specifying
bit numbers inside the parentheses.

Representing Data in HDL

- VHDL offers some standardized data types in libraries—collections of VHDL code that can be used to avoid reinventing the wheel.
- Many convenient functions such as standard TTL device descriptions are contained in macro-functions.

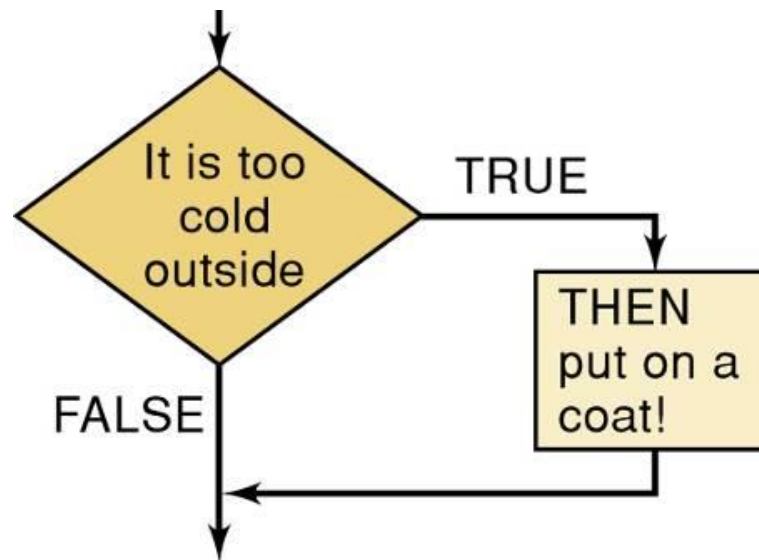
Data Type	Sample Declaration	Possible Values	Use
BIT	y :OUT BIT;	'0' '1'	y <= '0';
STD_LOGIC	driver :STD_LOGIC	'0' '1' 'z' 'x' '-'	driver <= 'z';
BIT_VECTOR	bcd_data :BIT_VECTOR (3 DOWNT0 0);	"0101" "1001" "0000"	digit <= bcd_data;
STD_LOGIC_VECTOR	dbus :STD_LOGIC_VECTOR (3 DOWNT0 0);	"0Z1X"	IF rd = '0' THEN dbus <= "zzzz";
INTEGER	SIGNAL z:INTEGER RANGE -32 TO 31;	-32.. -2, -1,0,1,2 . . . 31	IF z > 5 THEN . . .

Outline

- Introduction
- VHDL Module
 - Entity & Architecture
 - Signal
- VHDL Data Type
- VHDL Operator
- *VHDL Flow Control*
- More VHDL Examples

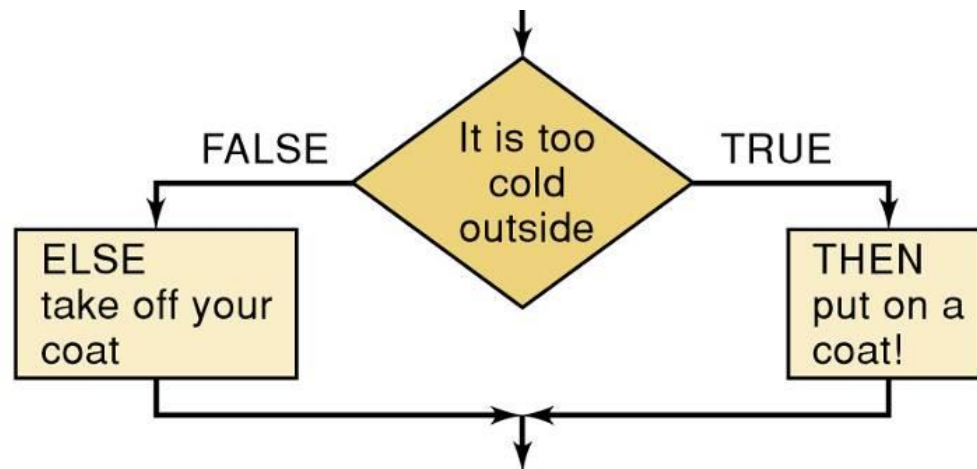
Decision Control Structures in HDL - IF/THEN/ELSE

- IF/THEN/ELSE statements provide a framework for making logical decisions in a system
 - IF/THEN is used when there is a choice between doing something and doing nothing.



Decision Control Structures in HDL - IF/THEN/ELSE

- IF/THEN/ELSE statements provide a framework for making logical decisions in a system
 - IF/THEN/ELSE is used when there is a choice of two possible actions.



Decision Control Structures in HDL - IF/THEN/ELSE

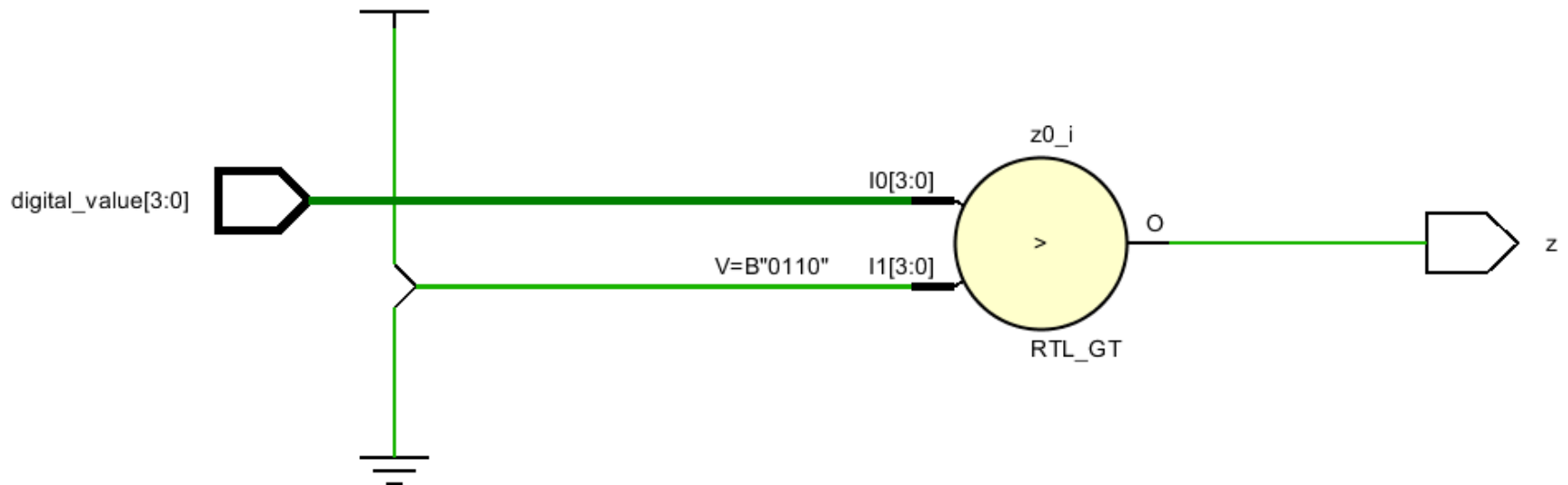
- IF/THEN/ELSE in VHDL:

```
entity Control_Example is
    port (
        digital_value: in integer range 0 to 15; -- 4-bit input
        z: out bit
    );
end Control_Example;

architecture Behavioral of Control_Example is
begin
    process (digital_value)
    begin
        if (digital_value > 6) then
            z <= '1';
        else
            z <= '0';
        end if;
    end process;
end Behavioral;
```

Schematic Design

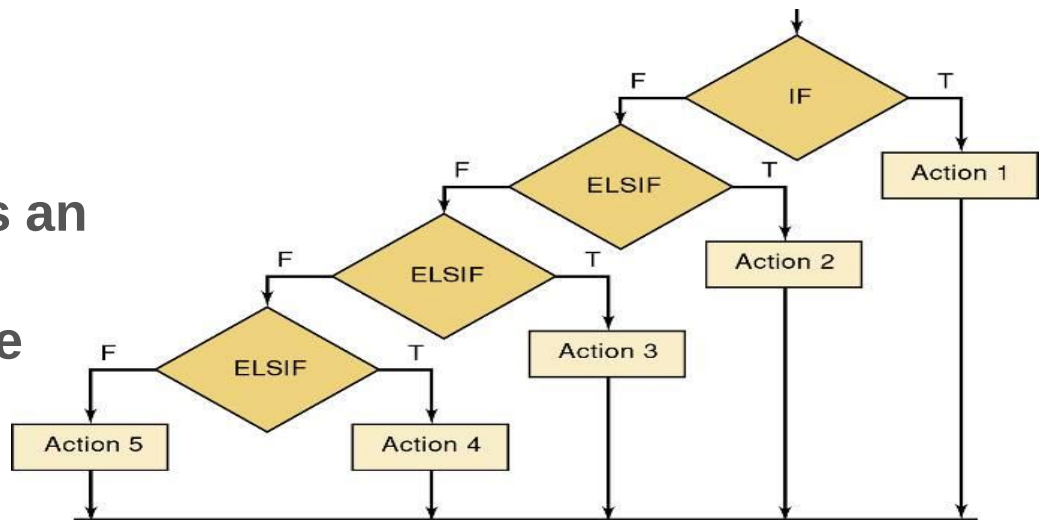
- Is this what you expect to see?



Decision Control Structures in HDL - ELSIF

- By combining IF and ELSE decisions, we can create a control structure referred to as ELSIF
- Which chooses one of many possible outcomes.

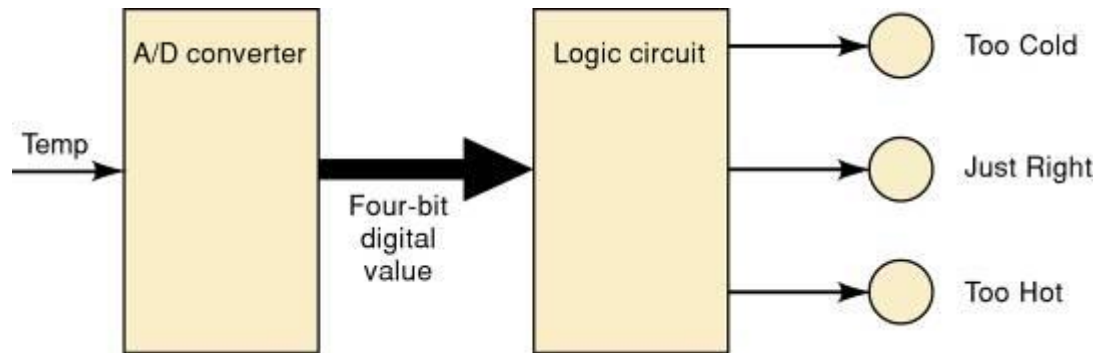
As each condition is evaluated, it performs an action—if true—or goes on to evaluate the next condition.



Decision Control Structures in HDL - ELSIF

- IF the digital value is less than or equal to 8...THEN light only the Too Cold indicator.

A temperature measuring system
using an A/D converter.



<i>Digital Values</i>	<i>Category</i>
0000–1000	Too Cold
1001–1010	Just Right
1011–1111	Too Hot

ELSE IF the digital value is greater than 8 **AND** less than 11...
THEN light only the **Just Right** indicator.
ELSE light only the **Too Hot** indicator.

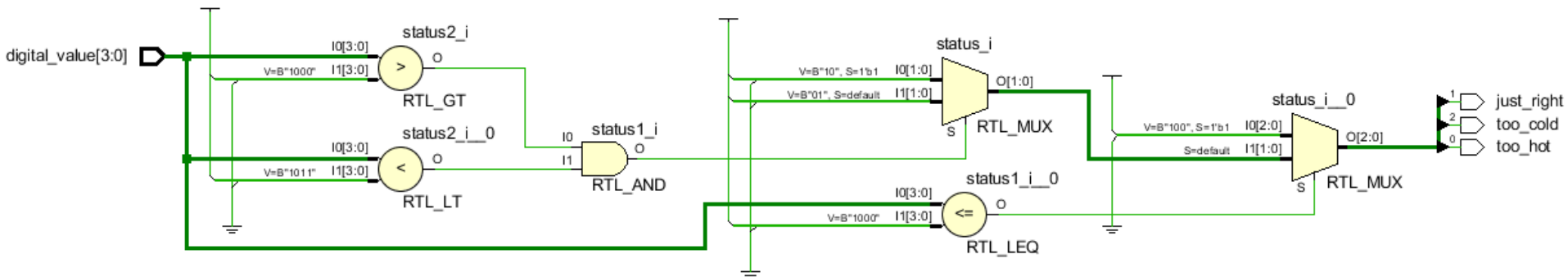
Decision Control Structures in HDL - ELSIF

- ELSIF in VHDL:

```
1 entity Control_Example is
2     port (
3         digital_value:          in integer range 0 to 15; -- 4-bit input
4         too_cold, just_right, too_hot: out bit
5     );
6 end Control_Example;
7
8 architecture Behavioral of Control_Example is
9     signal status: bit_vector (2 downto 0);
10 begin
11     process (digital_value)
12     begin
13         if (digital_value <= 8)
14         then status <= "100";
15         elsif (digital_value > 8 AND digital_value < 11)
16         then status <= "010";
17         else
18             status <= "001";
19         end if;
20     end process;
21
22     too_cold    <= status(2);
23     just_right  <= status(1);
24     too_hot     <= status(0);
25 end Behavioral;
```

Schematic

- A more complicated Design



Case Statement

- Executes different sections of code based on test condition
- Must be contained in a process
- Frequently used in conjunction with enumerations

Decision Control Structures in HDL - CASE

- The CASE construct determines the value of an expression or object.
- Then goes through a list of values (cases) to determine what action to take.
- Different than the IF/ELSEIF, as there is only one action or match for a case statement.

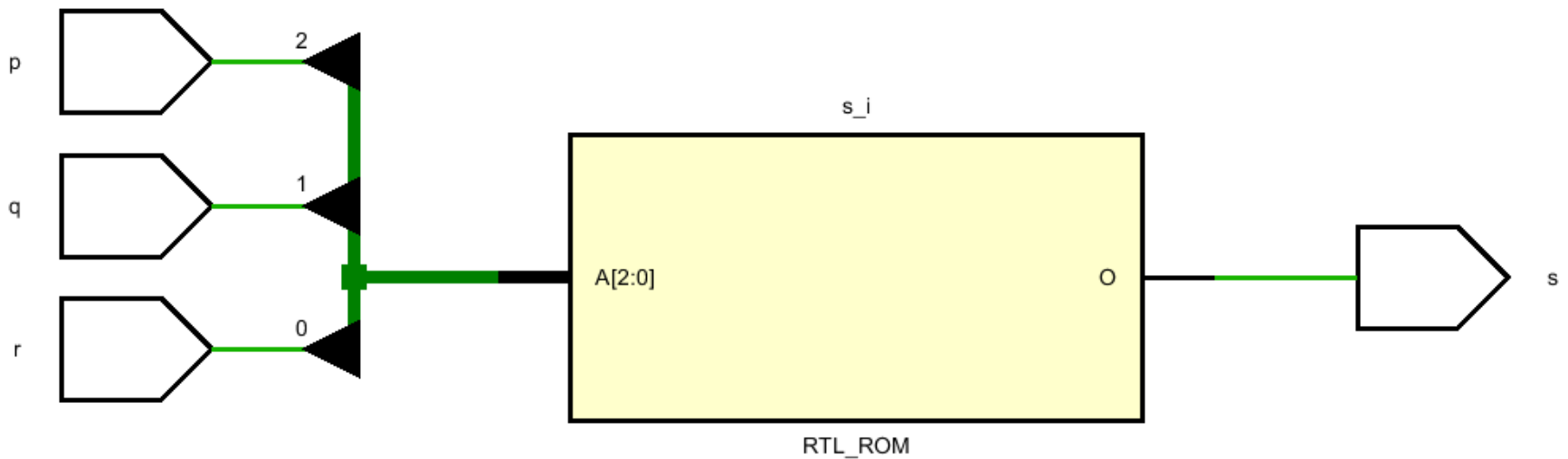
Decision Control Structures in HDL - CASE

- CASE construct in VHDL:

```
25 entity Control_Example is
26     port (
27         p, q, r:    in bit; -- 3 bits input signal
28         s:         out bit
29     );
30 end Control_Example;
31
32 architecture Behavioral of Control_Example is
33     signal status: bit_vector (2 downto 0);
34     begin
35         status <= p & q & r;
36         process (status)
37         begin
38             case status is
39                 when "100" =>    s <= '0';
40                 when "101" =>    s <= '0';
41                 when "110" =>    s <= '0';
42                 when others =>    s <= '1';
43             end case;
44         end process;
45     end Behavioral;
```

Schematic

- ROM content: What are the values?



Outline

- Introduction
- VHDL Module
 - Entity & Architecture
 - Signal
- VHDL Data Type
- VHDL Operator
- VHDL Flow Control
- *More VHDL Examples*
- Summary

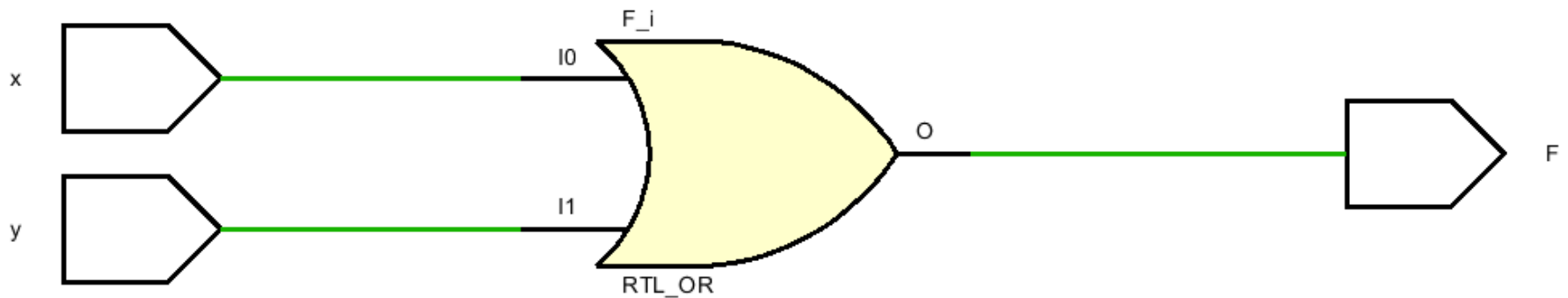
VHDL Example - 1) OR Gate

- Two Architecture Blocks

```
24 | -----
25 | entity OR_ent is
26 |     port(
27 |         x: in std_logic;
28 |         y: in std_logic;
29 |         F: out std_logic);
30 | end OR_ent;
31 |
32 | architecture OR_arch of OR_ent is
33 | begin
34 |     process(x, y)
35 |     begin
36 |         -- compare to truth table
37 |         if ((x='0') and (y='0')) then
38 |             F <= '0';
39 |         else
40 |             F <= '1';
41 |         end if;
42 |     end process;
43 | end OR_arch;
44 |
45 | architecture OR_beh of OR_ent is
46 | begin
47 |     F <= x or y;
48 | end OR_beh;
```

Schematic

- Check the input / output ports

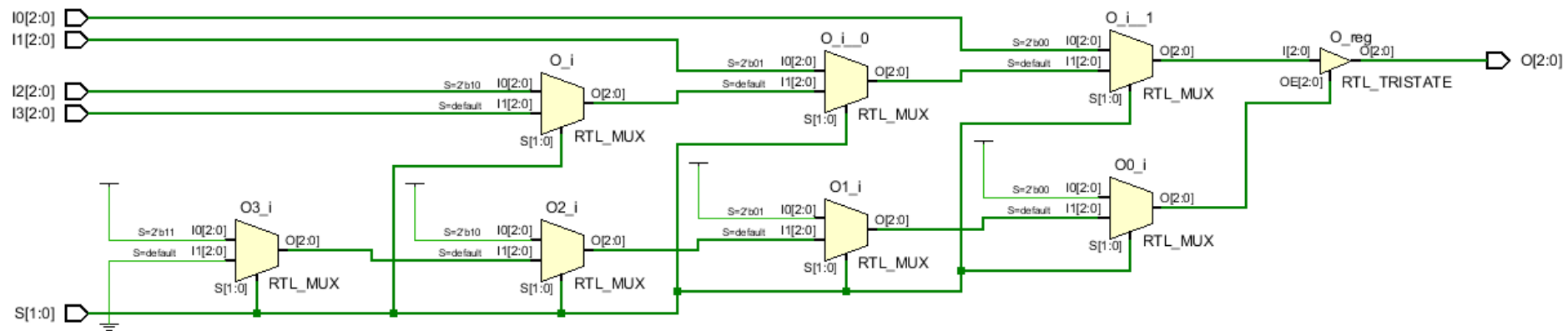


2) MUX Example

```
--
22  library ieee;
23  use ieee.std_logic_1164.all;
24  -----
25  entity Mux is
26      port(
27          I3:  in std_logic_vector(2 downto 0);
28          I2:  in std_logic_vector(2 downto 0);
29          I1:  in std_logic_vector(2 downto 0);
30          I0:  in std_logic_vector(2 downto 0);
31          S:   in std_logic_vector(1 downto 0);
32          O:   out std_logic_vector(2 downto 0));
33  end Mux;
34
35  architecture Behavioral of Mux is
36  begin
37      -- use when.. else statement
38      O <=  I0 when S="00" else
39            I1 when S="01" else
40            I2 when S="10" else
41            I3 when S="11" else
42            "ZZZ";
43  end Behavioral;
```

Schematic

- Can you prepare the testbench for this design?



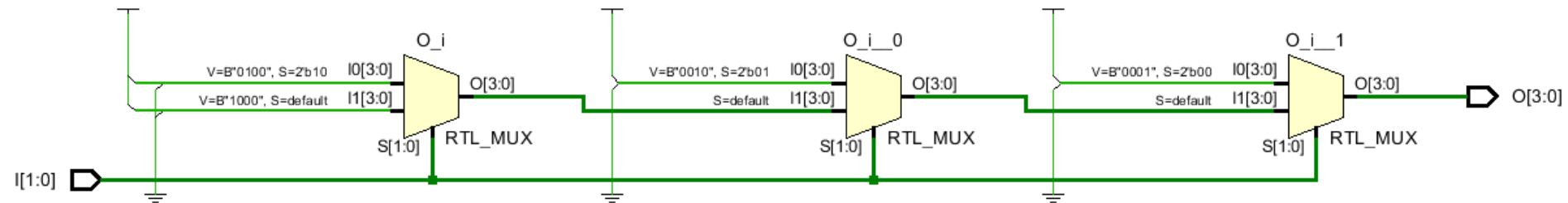
3) Decoder Example

- 2-input to 4-output decoder

```
25 | -----  
26 | entity DECODER is  
27 |     port(  
28 |         I:  in std_logic_vector(1 downto 0);  
29 |         O:  out std_logic_vector(3 downto 0));  
30 | end DECODER;  
31 | -----  
32 | architecture Behavioral of DECODER is  
33 | begin  
34 |     -- use when..else statement  
35 |     O <=  "0001" when I = "00" else  
36 |           "0010" when I = "01" else  
37 |           "0100" when I = "10" else  
38 |           "1000" when I = "11" else  
39 |           "XXXX";  
40 | end Behavioral;
```

Schematic

- How about other combinational Design?
- Four possible outputs:
 - 1000, 0100, 0010, 0001



Summary

- VHDL can be used to model combinational logic and sequential machines.
- VHDL is a hardware description language.
- VHDL statements execute concurrently.
- Statements within a process execute sequentially, covered later.
- VHDL signals model actual signals in hardware.
- Variables may be used for internal computation.

