



The Abdus Salam
International Centre for Theoretical Physics



310/1780-10

**ICTP-INFN Advanced Training Course on
FPGA and VHDL for Hardware Simulation and Synthesis
27 November - 22 December 2006**

VHDL & FPGA – Session 2

***Nizar ABDALLH
ACTEL Corp.
2061 Stierlin Court
Mountain View, CA 94043-4655
U.S.A.***

These lecture notes are intended only for distribution to participants



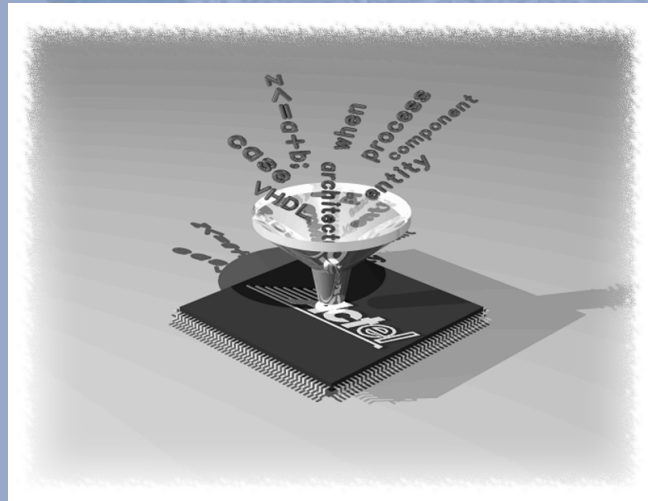
Lectures: VHDL & FPGA Architectures



- Introduction to FPGA & FPGA Design Flow
- **Synthesis I – Introduction**
- **Synthesis II - Introduction to VHDL**
- **Synthesis III - Advanced VHDL**
- **Design verification & timing concepts**
- **Programmable logic & FPGA architectures**
- **Actel ProASIC3 FPGA architecture**



Synthesis II – Introduction to VHDL



- **HDL is a software solution due to limits in hardware solutions and to:**
 - Increasing design complexity
 - Increasing cost in time and investment
 - Increasing knowledge requirement
 - Inadequacy of other existing languages

- **VHDL is a response to problems for system manufacturers in verifying their system fully**
 - Vendor dependency
 - Different vendors with different incompatible HDLs
 - Problems in design documentation exchange

**A standard HDL from the System
Manufacturer's Point of View: VHDL**

- **1981: an Extensive Public Review (DOD): VHSIC Program for modeling digital systems**
- **1983: a Request for Proposal (Intermetrics, IBM, and Texas Instruments)**
- **1986: VHDL in the Public Domain**
- **1987: Standard VHDL'87 (IEEE-1076-1987)**
- **1993: New Standard VHDL'93 (IEEE-1076-1993)**
- **1994: VITAL (VHDL Initiative Toward ASIC Libraries)**
 - Defines Standard Format for Gate Libraries, 1994
 - Allows Specification of Common Timing Models
- **Revised Standard “VHDL-2001” in review now**

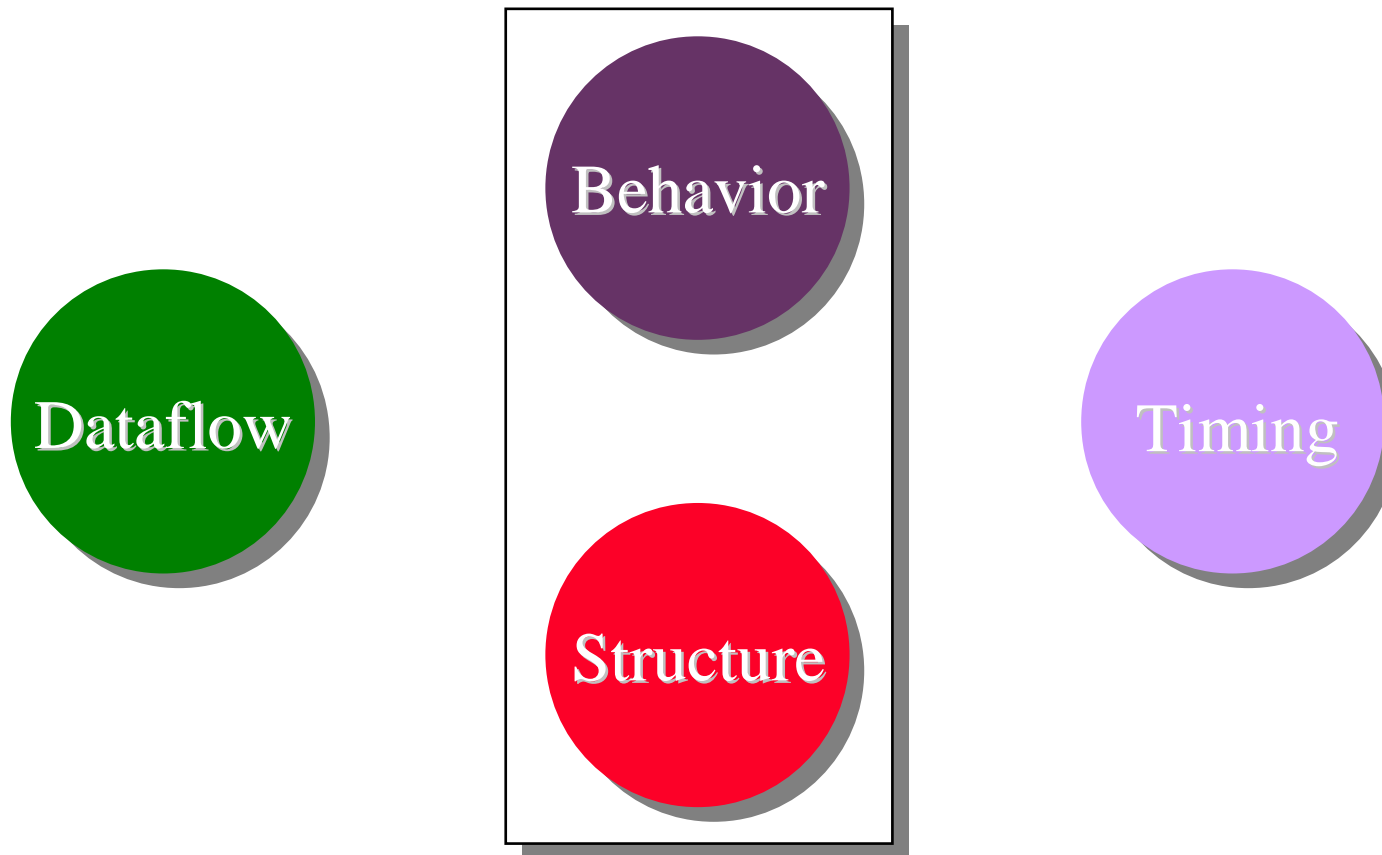
■ **Philosophy: readable, docs-based on a clear and predictable simulation behavior**

■ **Advantages**

- Standard format for design exchange
- Technology independent
- Multiple vendor support
- Support for large as well as small designs
- Support for wide range of abstraction in modeling
- Simulation oriented (including for writing testbench)
- User defined value abstractions
- Timing constructs

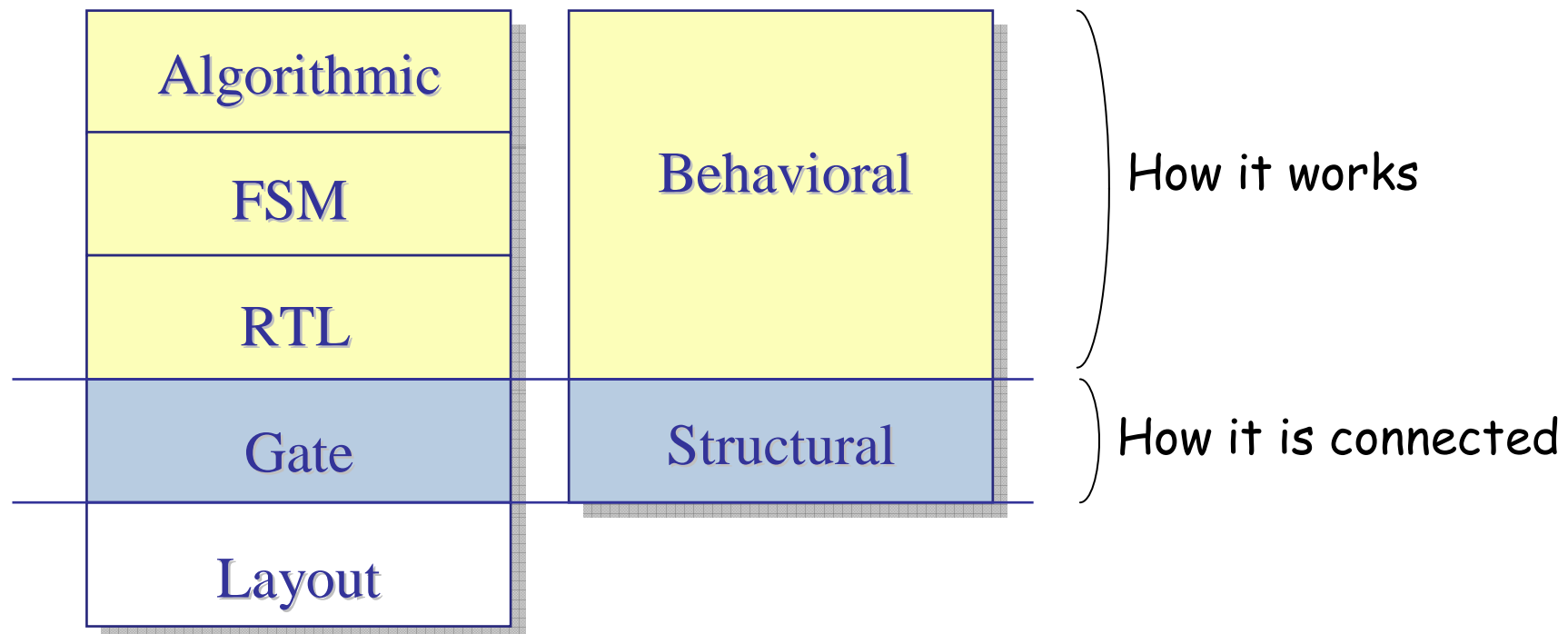
■ **Drawbacks**

- Complex tools
- Slow tools



- **Does not allow a layout description**

Abstraction Levels VHDL Architectures



CONTROLFLOW \neq DATAFLOW

EX: C language assignment

$X = A \ \& \ B;$

X is computed out of A and B ONLY each time this assignment is executed

EX: VHDL signal assignment



$X \leq A \ \text{and} \ B;$

A PERMANENT link is created between A, B, and X

X is computed out of A and B WHENEVER A or B changes

CONTROLFLOW \neq DATAFLOW

EX: C language assignment

X = A & B;

X = C & D;

✓ YES

EX: VHDL signal assignment

~~**X <= A and B;**~~

~~**X <= C and D;**~~

✗ NO

■ D-FF with asynch low reset & pos-edge clock

```
process (CLK, RESET) Behavioral
begin
    if (RESET = '0') then
        Q <= '0';
    elsif (CLK'event and CLK='1') then
        Q <= DATA;
    end if ;
end process;
```

```
-- component declaration Structural
component DFF
    port ( D, CLR, CLK : in std_logic;
          Q : out std_logic);
end component;
-- component instantiation
U1: DFF
port map (D => DATA, CLR => RESET, CLK => CLK, Q => OUT);
```

- **Entity**
- Architecture
- Configuration
- Package
- Library

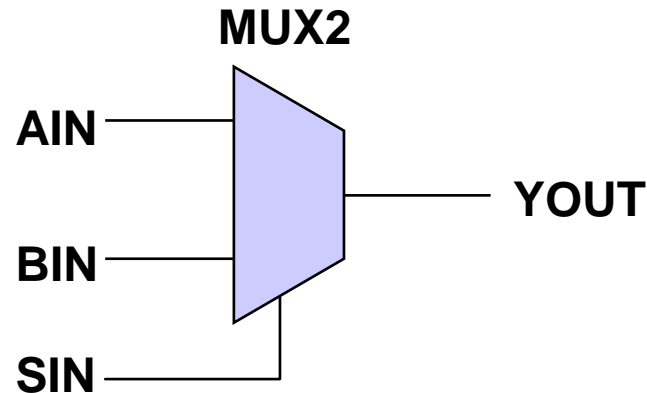
■ The External Aspect of a Design Unit

Required → `entity` entity_name `is`
Optional → [generic_declaration]
 [port_clause]
 [entity_declarative_item]

Required (NAME optional) → `end` [entity_name];

What is visible

Entity Example: 2-to-1 Mux



```
entity MUX2 is
  port (AIN, BIN, SIN      : in std_logic;
        YOUT              : out std_logic);
end MUX2;
```

Port
mode

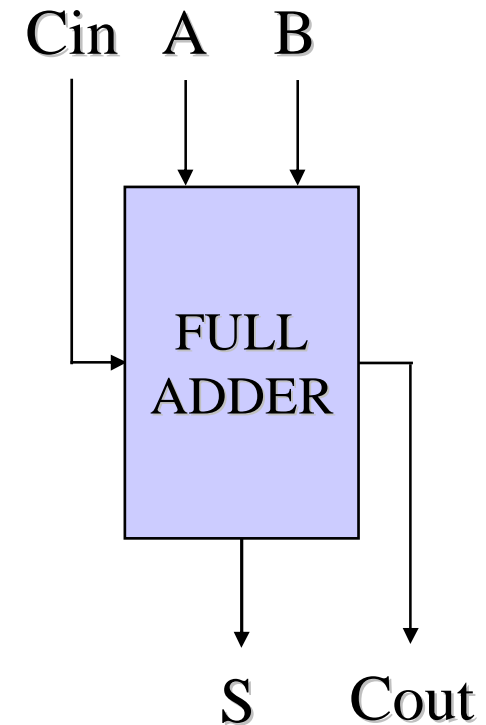
Port
type

Entity Example: Full Adder

```
entity FULL_ADDER is
  port (A, B, Cin : in BIT;
        S, Cout  : out BIT);
end FULL_ADDER;
```

Port
mode

Port
type



- **Provide communication with other components**
- **Must have signal name, type and mode**
- **Port Modes:**
 - in (data goes into entity only)
 - out (data goes out of entity only and not used internally)
 - inout (data is bi-directional)
 - buffer (data goes out of entity and used internally)

- Entity
- **Architecture**
- Configuration
- Package
- Library

- **The Internal Aspect of a Design Unit**
- **Can be behavioral (RTL) or structural**
- **Always associated with single entity**
- **Single entity can have multiple architectures**

```
architecture architecture_name of entity_name is
    {architecture_declarative_part}
begin
    {architecture_descriptive_part}
end [architecture_name];
```

■ Two architecture flavors: Behavioral & Structural

```
architecture ONE of MUX2 is
begin
  YOUT <= (AIN and not SIN) or (BIN and SIN);
end ONE;
```

Behavioral

```
architecture TWO of MUX2 is
  component MX2 -- a macro from a library
    port (A, B, S:in std_logic;
          Y      :out std_logic);
  end component;
begin
  -- instantiate MX2
  U1: MX2
  port map(A=>AIN, B=>BIN, S=>SIN, Y=>YOUT);
end TWO;
```

Structural

} Declarative part

} Descriptive part

■ Two architecture flavors: Behavioral & Structural

Behavioral

```
entity FULL_ADDER is
  port (A, B, Cin : in BIT;
        S, Cout  : out BIT);
end FULL_ADDER;
architecture DATAFLOW of FULL_ADDER is
  signal X : BIT;
begin
  X    <= A xor B;
  S    <= X xor Cin after 10ns;
  Cout <= (A and B) or (X and Cin) after 5ns;
end DATAFLOW;
```

Architecture Example: Full Adder (continued)

■ Two architecture flavors: Behavioral & Structural

Structural:
Declarative part

```
architecture STRUCTURE of FULL_ADDER is
  component HALF_ADDER
    port ( I1, I2 : in BIT;
          Carry, S : out BIT);
  end component;

  component OR_GATE
    port ( I1, I2 : in BIT;
          O : out BIT);
  end component;
  signal X1, X2, X3 : BIT;
```

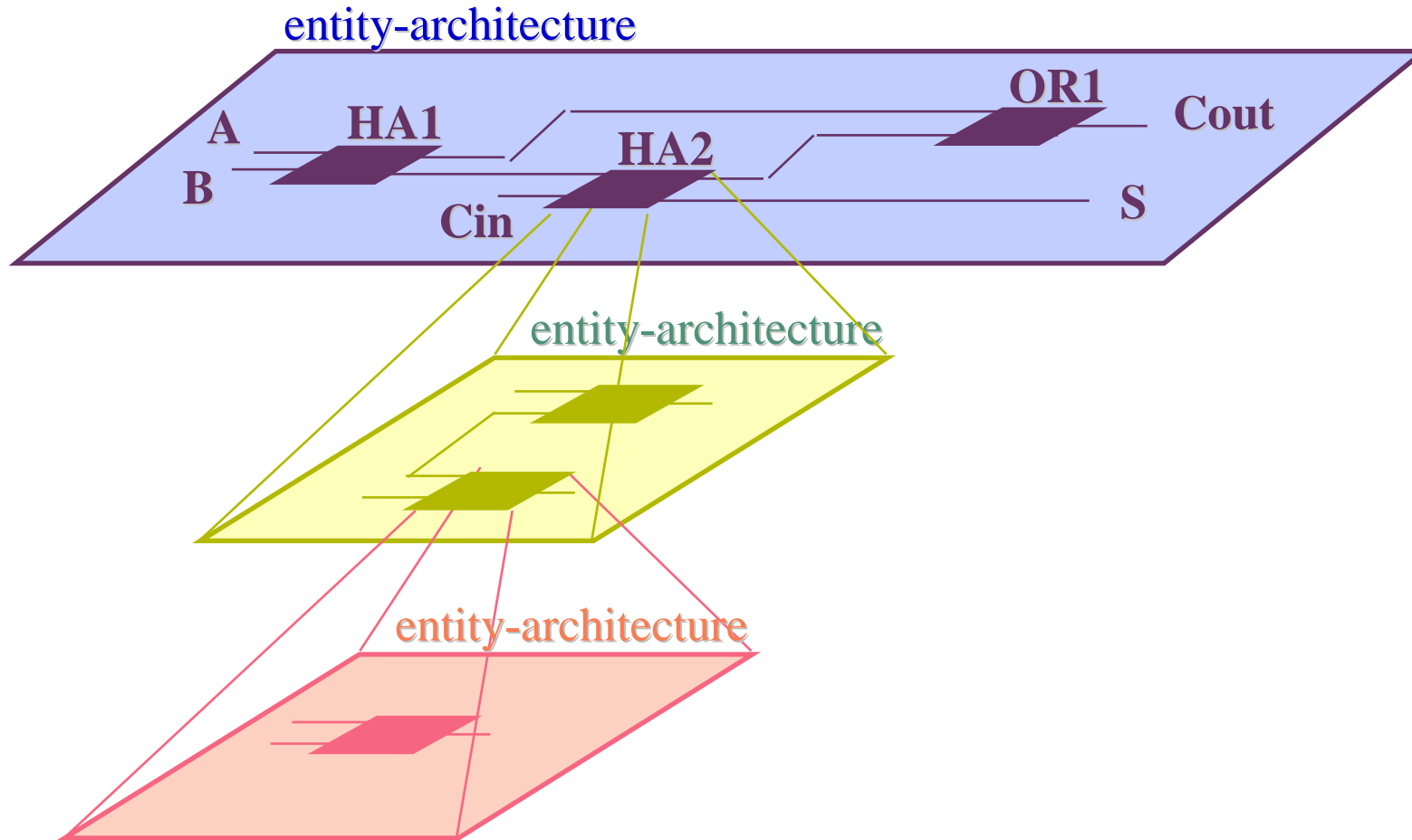
Architecture Example: Full Adder (continued)

■ Two architecture flavors: Behavioral & Structural

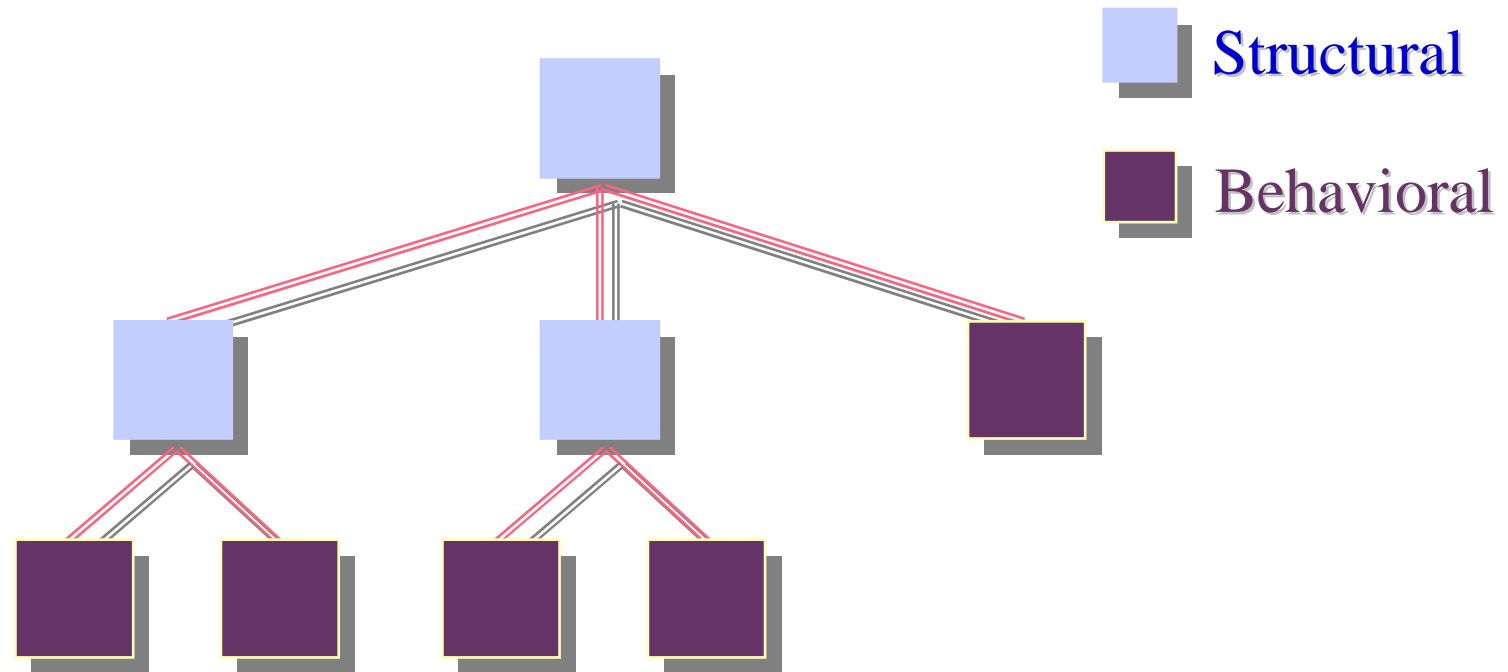
Structural:
Descriptive part

```
begin
  HA1: HALF_ADDER    port map (
    I1 => A, I2 => B, Carry => X1, S => X2);
  HA2: HALF_ADDER    port map (
    I1 => X2, I2 => Cin, Carry => X3, S => S);
  OR1: OR_GATE       port map (
    I1 => X1, I2 => X3, O => Cout);
end STRUCTURE ;
```

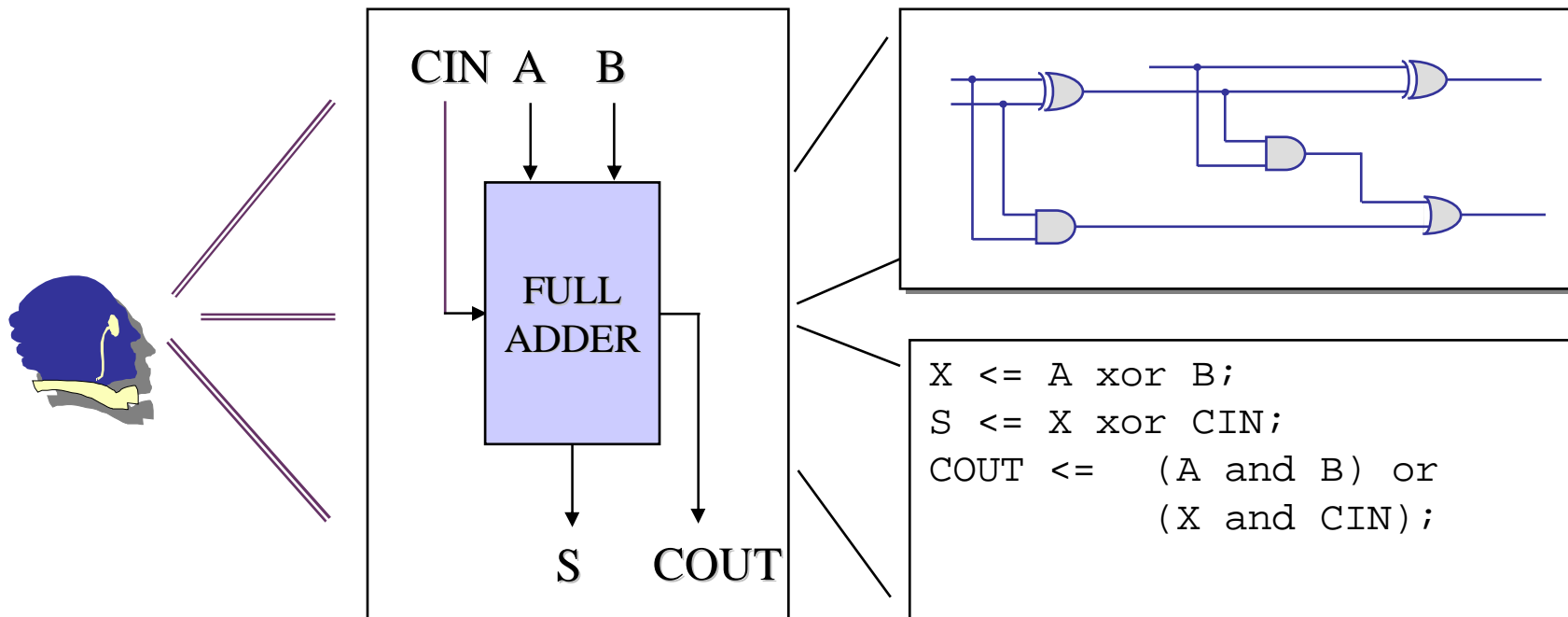
■ Structural Style to represent Hierarchy



■ Structure & Behavior in a Design Tree



■ entity/architecture: a One-to-Many relationship



- Entity
- Architecture
- Configuration
- **Package**
- **Library**

- **Libraries contain packages**
- **Packages contain commonly used types, operators, constants, functions, etc.**
- **Both must be “opened” before their contents can be used in an entity or architecture**

```
library ieee;  
use ieee.std_logic_1164.all;
```

Commonly Used Libraries & Packages

STD Library

standard

← Always visible! - no
LIBRARY or USE
statement required

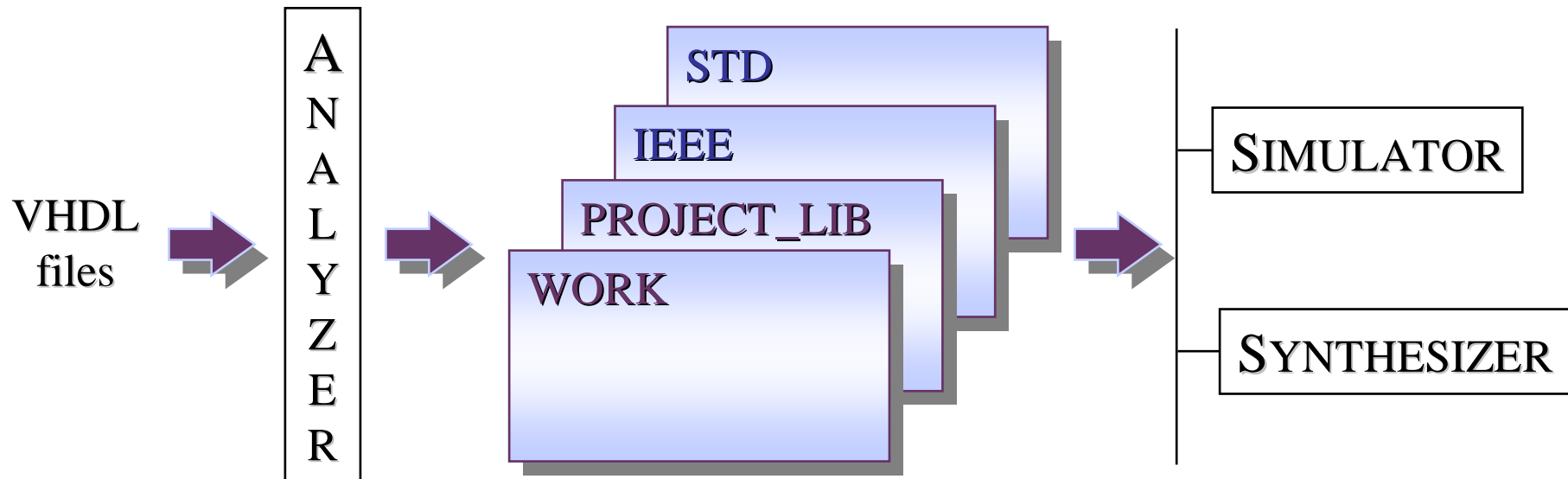
IEEE Library

std_logic_1164

std_logic_arith

std_logic_unsigned

← Requires LIBRARY and
USE statements



```
library library_name ;  
use library_name.package_name.all ;
```

Complete Design Example: Multiplier

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity MULT is  
port(A: in std_logic_vector(3 downto 0);  
      B: in std_logic_vector(3 downto 0);  
      Y: out std_logic_vector(7 downto 0));  
end MULT;  
  
architecture TEST of MULT is  
begin  
  Y <= A * B;  
end TEST;
```

Declare the library
and packages

Now we can use
std_logic_vector
types and unsigned
arithmetic

Is this a structural or behavioral description ?

■ Constants

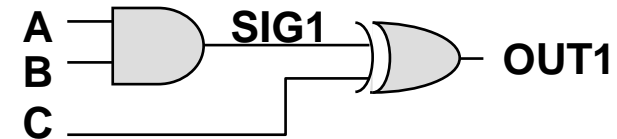
■ Variables

- syntax: `var:= expression`
- can be declared in body (inside process) or subprogram (outside process)
- a body-declared variable is never reinitialized
- a sub-program declared variable is initialized for each call to the subprogram
- value assignment as immediate effect

■ Signals

- syntax: `signal <= value`
- delayed value assignment
- optional propagation delay attribute
- no global variables to avoid synchronization problems
- value *resolution* for multiple assignments

- Used for internal connections
- Must be declared before use
- Must have a type
- Assignment is done with `<=`



```
...  
architecture TEST of EXAMPLE is  
    signal SIG1: std_logic; ← Declaration  
begin  
    SIG1 <= A and B; ← Notice: A, B, C, and OUT1  
    OUT1 <= SIG1 xor C; are not Signals; They are  
end TEST; Ports declared in the Entity.
```

- **Bit order may be ascending (0 to 7) or descending (7 downto 0)**

Q is an 8-bit vector

```
entity COUNTER is
port (CLK: in std_logic;
      RST: in std_logic;
      Q : out std_logic_vector(7 downto 0));
end COUNTER;
```

S2 is a 3-bit vector

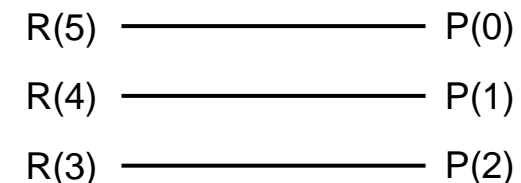
```
...
architecture BEHAVE of COUNTER is
  signal S1: std_logic_vector(7 downto 0);
  signal S2: std_logic_vector(2 downto 0);
  signal S3: std_logic;
```

- **Vector slicing can be used on either side of a signal or variable assignment statement**
- **Extracts subset of Vector for Reading or Writing**

```
port (P: in std_logic_vector(0 to 3);  
      R: out std_logic_vector(8 downto 0);  
...  
architecture BEHAVE of EXAMPLE is  
...  
    R(5 downto 3) <= P(0 to 2);
```

Declaration Direction and Slice Direction must be the same. (P is up; R is down)

Size of slices must match



R(5) ————— P(0)
R(4) ————— P(1)
R(3) ————— P(2)

- **All VHDL data objects must have a type**
 - Port types are declared in the entity
 - Signal, variable, and constant types are declared in the architecture.

- **Types we will cover:**
 - STANDARD package types
 - User-defined enumeration types
 - IEEE std_logic_1164 package types

■ Predefined VHDL Data Types

- Is always visible
- No declaration is needed

■ Types available

- **BOOLEAN** : (false , true)
- **BIT** : ('0', '1')
- **BIT_VECTOR** : array of BIT values
- **INTEGER** : range -2 147 483 647 to +2 147 483 647
- **CHARACTER**
- **NATURAL** : Subtype of INTEGER (Non Negative)
- **POSITIVE** : Subtype of INTEGER (positive)
- **STRING** : array of CHARACTERS
- **REAL** : range -1.0E+38 to +1.0E+38
- **TIME** : Physical type used for simulation

- **Allowed values are mathematical integers**
- **Minimum range: 2³¹ to -2³¹ (32 bits minimum) if no range is specified**
- **Useful as index holders for loops or generics**
- **Supported operations are add, subtract, multiply, and divide**

```
architecture BEHAVE of COUNTER is
begin
  process (clk)
    variable cntr: integer range 0 to 63 := 0;
  begin
```

IEEE 1076-1987: bit / bit_vector Type

- Allowed values are '0' and '1'
- Default initialization to '0'

```
entity MUX is
  port (A, B, S: in bit;
        Y: out bit);
end MUX;
```

- bit_vector is an array of bits

```
architecture BEHAVE of MUX is
  constant LOW: bit_vector(1 downto 0) := "00";
begin
```

- **Allowed values are *true* and *false***
- **Not a bit literal – has no relationship to a bit**
- **Operations only allowed in IF-ELSE statements, in processes and always produce a Boolean result**

```
library ieee;
use ieee.std_logic_1164.all;

entity CONTROLLER is
  port (SEL  : in boolean;
        X, Y : in std_logic;
        Z   : out std_logic);
end CONTROLLER;

architecture BEHAVE of CONTROLLER is
begin
  process (X, Y, SEL)
  begin
    if (SEL) then
      Z <= X;
    else
      Z <= Y;
    end if ;
  end process;
end BEHAVE;
```


- Allow designers to specify exact values for operation
- Useful for state machine designs

```
architecture MACHINE of TRAFFIC_LIGHT is
  type STATE is (RED, YELLOW, GREEN);
  signal CURRENT_STATE, NEXT_STATE: STATE;
begin
```

The data type of two signals (current_state and next_state) is the user-defined type called state

IEEE 1164-1993 Standard Logic Package



- **std_logic_1164 must be declared**

```
library ieee;  
use ieee.std_logic_1164.all;
```

- **Supported by most VHDL simulators and synthesis tools**
- **Includes a multi-value logic system**
 - Nine signal strengths defined
 - Can resolve multiple signal drivers
- **Generally used instead of bit/bit_vector**

- **std_ulogic, std_ulogic_vector**
 - Unresolved type
 - Only one signal driver allowed

- **std_logic, std_logic_vector**
 - Resolved type - multiple drivers allowed
 - Used when tri-state logic required

- **std_logic is best choice for behavioral**

■ Unresolved data type

type STD_ULOGIC is (

'U'	--	Uninitialized
'X'	--	Forcing Unknown
'0'	--	Forcing Low (driven)
'1'	--	Forcing High (driven)
'Z'	--	High Impedance
'W'	--	Weak Unknown
'L'	--	Weak Low (read)
'H'	--	Weak High (read)
'-'	--	Don't Care

);



Synthesizable
for
FPGA

■ Six classes

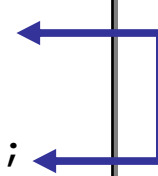
LOGIC OPERATOR	and , or , nand , nor , xor
RELATIONAL OPERATOR	= , /= , < , <= , > , >=
ADDING OPERATOR	+ , - , &
SIGN	+ , -
MULTIPLYING OPERATOR	* , / , mod , rem
MISCELLANEOUS OPERATOR	** , abs , not

PRECEDENCE ORDER

- **Require opening the following packages:**

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_signed.all;  
--OR  
use ieee.std_logic_unsigned.all;
```

Depending on whether signed or unsigned arithmetic is used.

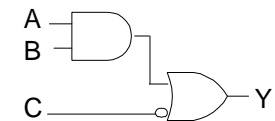


- **May be used on real, integer, bit or std_logic types.**

Operators: Example

```
library ieee;
use ieee.std_logic_1164.all;
entity LOGIC is
port (A, B, C: in std_logic;
      Y, Z   : out std_logic);
end LOGIC;

architecture BEHAVE of LOGIC is
begin
    Z <= A and B;
    Y <= (A and B) or not C;
end BEHAVE;
```

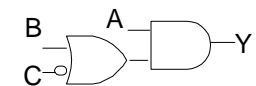


() prevent ambiguity. Otherwise this could be:

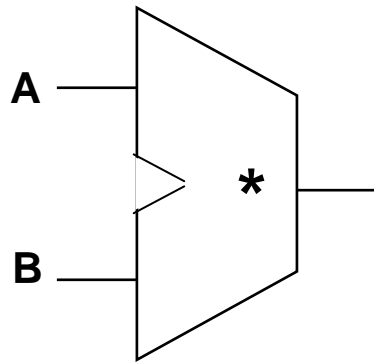
(A and B) or not C

OR

A and (B or not C)



■ 4-bit multiplier

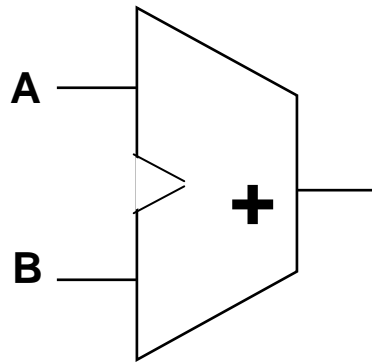


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity MULT is
    port(A: in std_logic_vector(3 downto 0);
         B: in std_logic_vector(3 downto 0);
         Y: out std_logic_vector(7 downto 0));
end MULT;

architecture BEHAVE of MULT is
begin
    Y <= A * B;
end BEHAVE;
```


■ 8-bit adder with 9-bit result



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ADDER is
  port(
    A, B : in std_logic_vector(7 downto 0);
    S     : out std_logic_vector(8 downto 0)
  );
end adder;

architecture BEHAVIORAL of ADDER is
  begin
    S <= (A(7) & A) + (B(7) & B);
  end BEHAVIORAL;
```

■ A Data Attached to VHDL Objects

- **S'LEFT** : Index of the leftmost element of the data type
- **S'RIGHT** : Index of the rightmost element of the data type
- **S'HIGH** : Index of the highest element of the data type
- **S'LOW** : Index of the lowest element of the data type
- **S'RANGE** : Index range of the data type
- **S'REVERSE_RANGE** : Reverse index range
- **S'LENGTH** : Number of elements of an array
- **S'EVENT** : A change value at the current simulation time
- **S'STABLE** : No change value at the current simulation time
if (CK = 0 and not CK'STABLE)

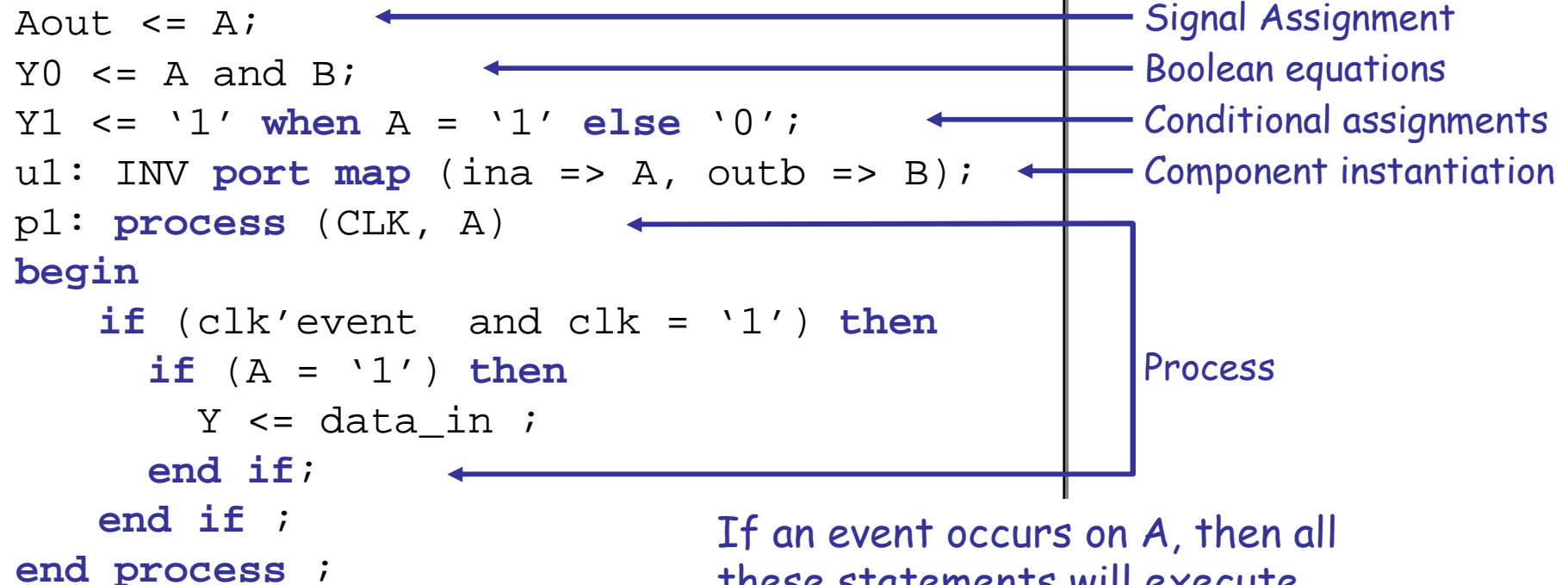
-



■ **Concurrent Statements**

- **Concurrent Signal Assignment**
- **Conditional Signal Assignment**
- **Selected Signal Assignment**
- **Block Statement**
- **Concurrent Assertion Statement**
- **Process Statement**

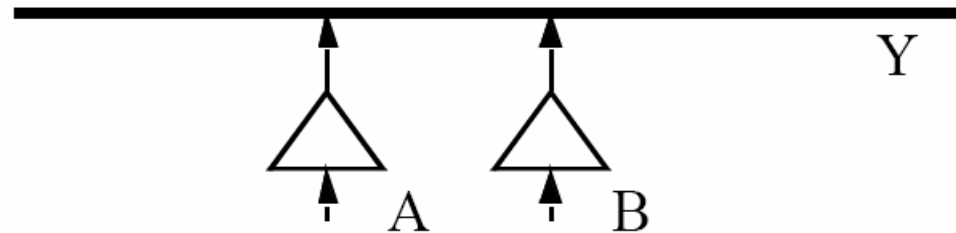
- **Execute at the same time**
- **Signals impacted by an event are resolved in the same simulation time**



If an event occurs on A, then all these statements will execute at the same time (concurrently)

- Always used within an architecture
- Change on the right-hand side causes immediate reassignment to the left-hand side
- Used in behavioral and structural descriptions
- Signals are associated with TIME
- With "after", the assignment is scheduled to a future simulation time
- Without "after", the assignment is scheduled at a DELTA TIME after the current simulation time
- Assignment operator is <=

target <= expression [after time_expression] ;



$Y \leq A;$ -- in process1
and, $Y \leq B;$ -- in process2

What is the value of the signal in such a case?

- **Concept of a Resolution Function; attached to a signal or a type, and is called every time the value of signal needs to be determined -- that is every time a driver changes value**

- **Concurrent Version of IF statement**
- **Condition/expression except for last expression**
- **One and only one of the expressions is used at a given time**

- **Syntax:**

```
target <= first_value when (condition1) else  
          second_value when (condition1) else  
          third_value;
```

- **Example:**

```
Y <= IN1 when S = '0' else  
     IN2 when S = '1' else  
     '0';
```

"else" clause is required

```
Y <= "00" when COUNT >= 8 else  
     "11";
```


■ Concurrent Version of CASE Statement

■ Syntax:

```
with EXPRESSION  
  TARGET <= {expression when choices};
```


■ Example:

“with” selection must be a signal, not an expression



```
with DATAIN select      -- selected signal assignment  
Y <= IN0 when "00",  
  IN1 when "01",  
  IN2 when "10",  
  IN3 when others;
```

Note that
commas are
used here.

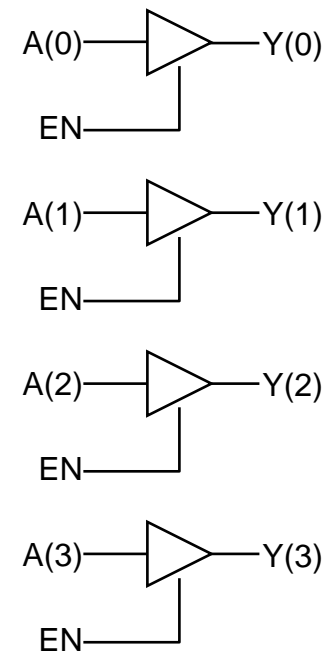


When-Else Example: Tri-State Buffer

```
library ieee;
use ieee.std_logic_1164.all;
entity MY_TRI is
    port(A: in std_logic_vector(3 downto 0);
         EN: in std_logic;
         Y: out std_logic_vector(3 downto 0));
end MY_TRI;

architecture BEHAVE of MY_TRI is
begin
    Y <= A when EN = '1' else
        (others => 'Z') ;
end BEHAVE;
```

means $Y \leftarrow \text{"ZZZZ"};$



With-Select Example: Truth Table



A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	-
1	1	1	-

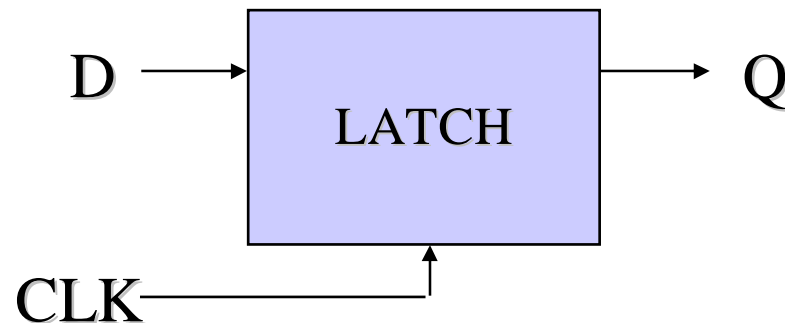
'-' means don't care

```
library ieee;
use ieee.std_logic_1164.all;
entity TRUTH_TABLE is
port(A, B, C: in std_logic;
      Y: out std_logic) ;
end TRUTH_TABLE;
architecture BEHAVE of TRUTH_TABLE is
  signal S1: std_logic_vector(2 downto 0);
begin
  S1 <= A & B & C; -- concatenate A, B, C
  with S1 select
  Y <= '1' when "000" | "010" | "100" ,
        '0' when "001" | "011" | "101",
        '-' when others;
end BEHAVE;
```

| means OR only when used in "with" or "case"

■ Used in synchronous descriptions

```
latch : block ( CLK = '1' )  
begin  
    Q <= GUARDED D ;  
end block latch ;
```



- **If the condition is false, it reports a diagnostic message**
- **Useful for detecting condition violation during simulation**
- **Not used in synthesis**

- **Syntax:**

```
assert condition
    [ report error_message ]
    [ severity severity_level ] ;
```

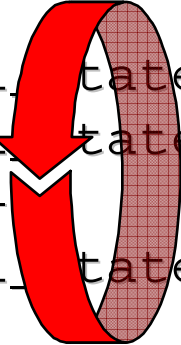
- **A Set of Sequential Statements**
- **All processes in a design executes CONCURRENTLY**
- **At a given time, ONLY ONE sequential statement executed within each process**
- **Communicates with the rest of a design through signals**

- **Syntax:**

```
[ label : ] process [ ( sensitivity_list ) ]  
    { process_declarative_part }  
begin  
    { sequential_statements }  
end process [ label ] ;
```

- **A Pseudo Infinite Loop**
 - A Synchronization Mechanism is Needed

```
process
begin
    sequential_statement_1 ;
    sequential_statement_2 ;
    -----
    sequential_statement_n ;
end process;
```



- **Sensitivity list or wait statement will be used**

Process With Sensitivity List

Process label
(recommended)

Sensitivity list

```
PQR:process (A, B, C)
begin
  X <= A and B;
  if (B = '1') then
    Y <= A xor C;
  else
    Y <= A xnor C;
  end if;
end process PQR;
```

Statements
executed
sequentially

End statement matches process label

NOTE! No sensitivity list

```
AORB: process
```

```
begin
```

```
  wait until rising_edge(clk);
```

```
    if (B = '1') then
```

```
      Y <= A xor C;
```

```
    else
```

```
      Y <= A xnor C;
```

```
    end if ;
```

```
end process AORB;
```

← WAIT condition
is on signals only

} Assignments execute
when their WAIT
condition is satisfied

Sequential Statements in Processes

- **Executed line-by-line inside of a process or sub-program**
- **Typically include:**
 - WAIT statements
 - Signal and variable assignments
 - Conditionals like IF-THEN-ELSE, CASE, LOOP
- **Support other advanced statements**

- **Always executed in Zero Simulation Time**
- **Used as temporary storages**
- **Can not be seen by other concurrent statements**

- **Syntax:**

```
target_variable := expression ;
```

- Defines a Driver of the Signal
- Within a process, Only One driver for each signal
- When assigned in multiple processes, it has Multiple Drivers. A Resolution Function should be defined
- **Syntax:**

```
target_signal <= [ transport ] expression [ after time ] ;
```

- **Default mode is Inertial**
- **Inertial is useful in modeling devices that ignore spikes on the inputs**

Inertial Delay Model



- **This is the default mode**
- **It is useful in modeling devices that ignore spikes on the inputs**

```
signal S : BIT := '0' ;  
process  
    S <= '1' after 5 ns ;  
    S <= '0' after 10 ns ;  
end ;
```

Overrides the first
assignment

```
signal S : BIT := '0' ;  
process  
    S <= '0' after 10 ns ;  
    S <= '1' after 5 ns ;  
end ;
```

Overrides the first
assignment

Transport Delay Model



- **Signals are propagated without filtering**

```
signal S : BIT := '0' ;  
process  
    S <= '1' after 5 ns ;  
    S <= '0' after 10 ns ;  
end ;
```



```
signal S : BIT := '0' ;  
process  
    S <= '0' after 10 ns ;  
    S <= '1' after 5 ns ;  
end ;
```



If-Then-Else Statement

```
if CONDITION then
  --sequential statements
end if;
```

```
if CONDITION then
  --sequential statements
else
  --sequential statements
end if;
```

More than one elsif allowed
"elsif" is one word



Only one else allowed



"end if" is two words

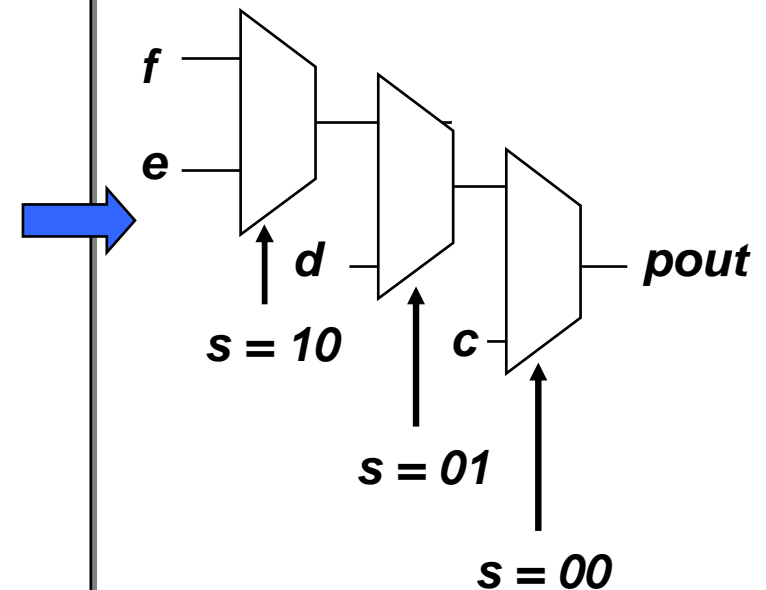


```
if CONDITION then
  --sequential statements
elsif CONDITION then
  --sequential statements
elsif CONDITION then
  --sequential statements
else
  -- sequential statements
end if;
```

If-Then-Else Example

```
library ieee;
use ieee.std_logic_1164.all;
entity IF_MUX is
  port (C, D, E, F : in std_logic;
        S : in std_logic_vector(1 downto 0);
        POUT : out std_logic);
end IF_MUX;
architecture BEHAVE of IF_MUX is
begin
ONE: process (S, C, D, E, F)
begin
  if (S = "00") then
    POUT <= C;
  elsif (S = "01") then
    POUT <= D;
  elsif (S = "10") then
    POUT <= E;
  else POUT <= F;
  end if ;
end process ONE;
end BEHAVE;
```

IF implies Priority




```
case (SELECTOR) is
  when value =>
    --sequential statements
  when value1 | value2 | value3 =>
    --sequential statements
  when value1 to value2 =>
    --sequential statements
  when others =>
    --sequential statements
end case;
```

'when SELECTOR = value, then ...'

'when SELECTOR = value1 OR value2 OR value 3, then ...'

'when SELECTOR falls within the range from value1 to value2, then ...'

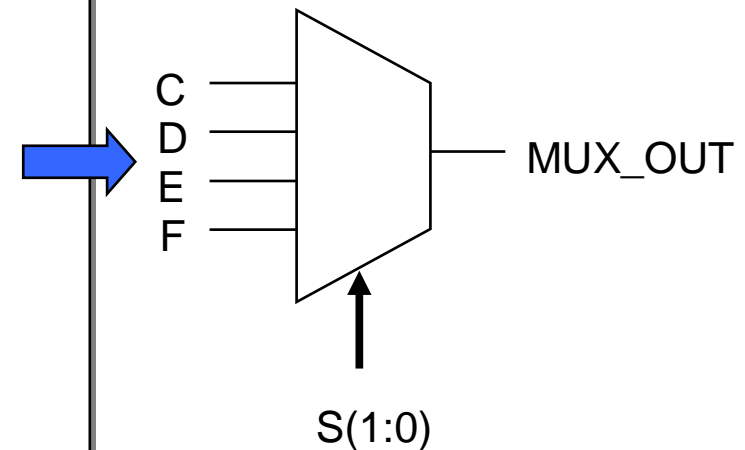
'when SELECTOR = any other value, then ...'

Case Example: 4-to-1 Mux

```
library ieee;
use ieee.std_logic_1164.all;
entity CASE_MUX is
  port (C, D, E, F: in std_logic;
        S : in std_logic_vector(1 downto 0);
        MUX_OUT : out std_logic );
end CASE_MUX;

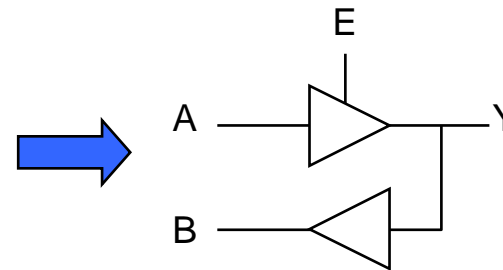
architecture BEHAVE of CASE_MUX is
begin
mux1: process (S, C, D, E, F)
begin
  case S is
    when "00" => MUX_OUT <= C;
    when "01" => MUX_OUT <= D;
    when "10" => MUX_OUT <= E;
    when others => MUX_OUT <= F;
  end case;
end process mux1;
end BEHAVE;
```

4:1 Multiplexer



Case Example: Bi-Directional Buffers

```
library ieee;
use ieee.std_logic_1164.all;
entity BIBUF is
  port (A, E: in std_logic;
        Y : inout std_logic;
        B : out std_logic );
end BIBUF ;
architecture BEHAVE of BIBUF is
begin
  ONE: process (A,E)
  begin
    case E is
      when '1' => Y <= A;
      when '0' => Y <= 'Z';
      when others => Y <= 'X';
    end case;
  end process ONE;
  B <= Y;
end BEHAVE;
```



Questions ?

