Faculty of Engineering

CSE115: Digital Design

Lecture 22:
VHDL

Reasons for Using VHDL

- VHDL is an international IEEE standard specification language (IEEE 1076-1993) for describing digital hardware used by industry worldwide.
  - VHDL is an acronym for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language.
- VHDL enables hardware modeling from the gate to system level.
- VHDL provides a mechanism for digital design and reusable design documentation.

History of VHDL

- Very High Speed Integrated Circuit (VHSIC) Program
  - Launched in 1980
  - Aggressive effort to advance state of the art
  - Object was to achieve significant gains in VLSI technology
  - Need for common descriptive language
  - $17 Million for direct VHDL development
  - $16 Million for VHDL design tools
- Woods Hole Workshop
  - Held in June 1981 in Massachusetts
  - Discussion of VHSIC goals
  - Comprised of members of industry, government, and academia

History of VHDL (2)

- In July 1983, a team of Intermetrics, IBM and Texas Instruments were awarded a contract to develop VHDL.
- In August 1985, the final version of the language under government contract was released: VHDL Version 7.2.
- In December 1987, VHDL became IEEE Standard 1076-1987 and in 1988 an ANSI standard.
- In September 1993, VHDL was restandardized to clarify and enhance the language.
- VHDL has been accepted as a Draft International Standard by the IEC.

Additional Benefits of VHDL

- Allows various design methodologies
- Provides technology independence
- Describes a wide variety of digital hardware
- Eases communication through standard language
- Allows for better design management
- Provides a flexible design language
- Has given rise to derivative standards
  - WAVES, VITAL, Analog VHDL

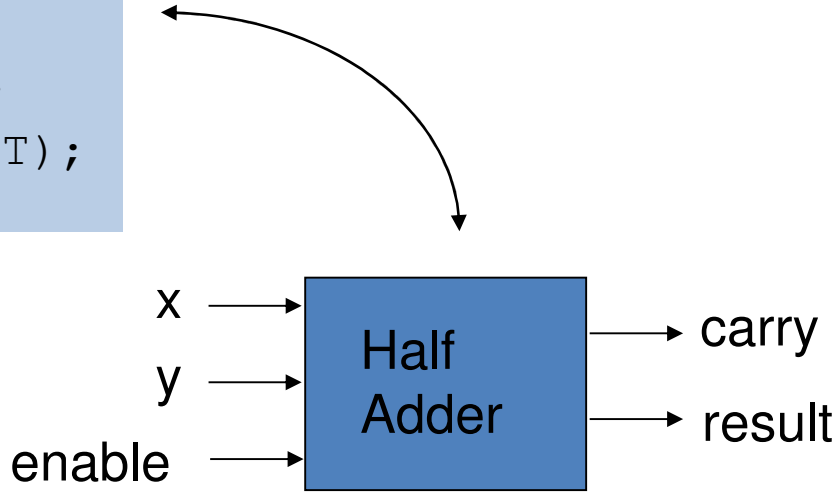
VHDL Design Example

- Problem: Design a single bit half adder with carry and enable
- Specifications
  - Inputs and outputs are each one bit
  - When enable is high, result gets x plus y
  - When enable is high, carry gets any carry of x plus y
  - Outputs are zero when enable input is low

VHDL Design Example: Entity Declaration

- As a first step, the entity declaration describes the interface of the component
  - Input and output ports are declared

```
ENTITY half_adder IS

   PORT( x, y, enable: IN BIT;
         carry, result: OUT BIT);

END half_adder;
```



Specifications

- Behavioral
- Data Flow
- Structural

VHDL Design Example: Behavioral Specification

- A high level description can be used to describe the function of the adder

```
ARCHITECTURE half_adder_a OF half_adder IS
     BEGIN
       PROCESS (x, y, enable)
          BEGIN
                    IF enable = '1' THEN
                              result <= x XOR y;
                              carry <= x AND y;
                    ELSE
                              carry <= '0';
                              result <= '0';
                    END IF;
          END PROCESS;
     END half_adder_a;
```

- The model can then be simulated to verify correct functionality of the component

VHDL Design Example: Data Flow Specification

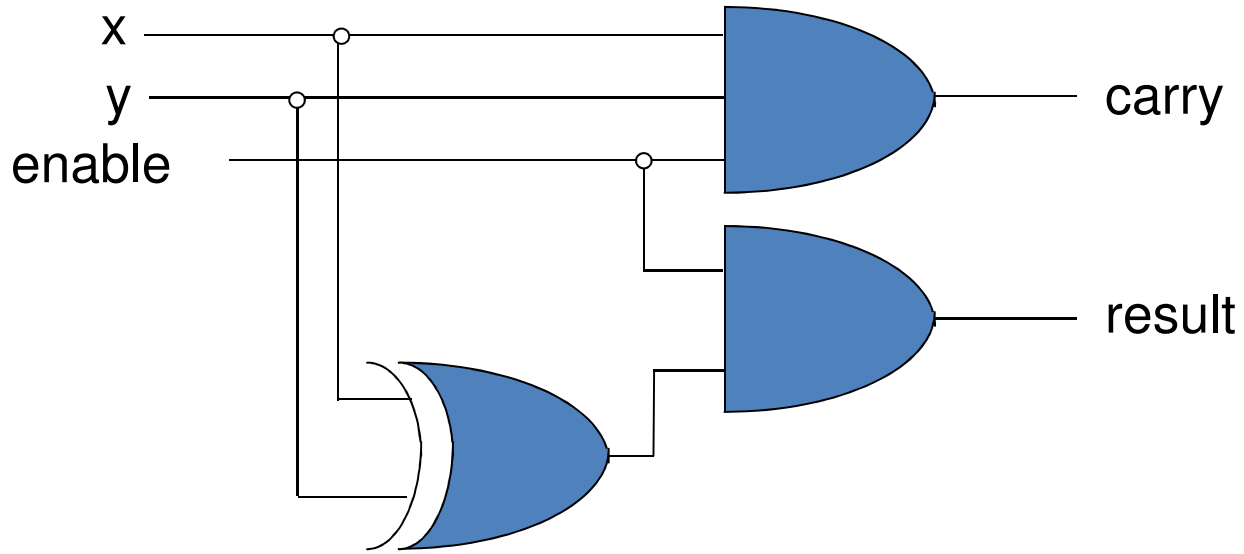
- A second method is to use logic equations to develop a data flow description

```
ARCHITECTURE half_adder_b OF half_adder IS
   BEGIN
     carry <= enable AND (x AND y);
     result <= enable AND (x XOR y);
   END half_adder_b;
```

- Again, the model can be simulated at this level to confirm the logic equations

VHDL Design Example: Structural Specification

- As a third method, a structural description can be created from predescribed components



- These gates can be pulled from a library of parts

VHDL Design Example: Structural Specification (2)

```
ARCHITECTURE half_adder_c OF half_adder IS

     COMPONENT and2
       PORT (in0, in1 : IN BIT;
             out0 : OUT BIT);
     END COMPONENT;

     COMPONENT and3
       PORT (in0, in1, in2 : IN BIT;
             out0 : OUT BIT);
     END COMPONENT;

     COMPONENT xor2
       PORT (in0, in1 : IN BIT;
             out0 : OUT BIT);
     END COMPONENT;

     FOR ALL : and2 USE ENTITY gate_lib.and2_Nty(and2_a);
     FOR ALL : and3 USE ENTITY gate_lib.and3_Nty(and3_a);
     FOR ALL : xor2 USE ENTITY gate_lib.xor2_Nty(xor2_a);

-- description is continued on next slide
```

VHDL Design Example: Structural Specification (3)

```
-- continuing half_adder_c description

    SIGNAL xor_res : BIT; -- internal signal
    -- Note that other signals are already declared in entity

    BEGIN

      A0 : and2 PORT MAP (enable, xor_res, result);
      A1 : and3 PORT MAP (x, y, enable, carry);
      X0 : xor2 PORT MAP (x, y, xor_res);

    END half_adder_c;
```

VHDL Model Components

- Entity
  - Defines a component's interface
- Architecture
  - Defines a component's function
- Alternative architectures

VHDL Component Descriptions

- Structural
- Behavioral
- Timing and delay

VHDL Model Component Behavior Description

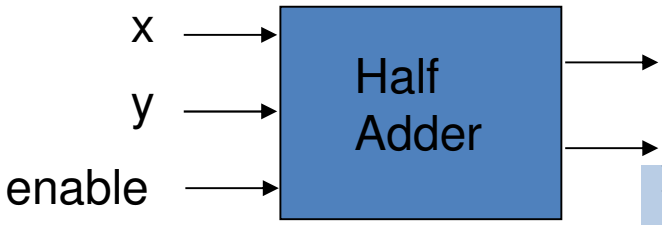
- Process - fundamental unit for component behavior description
  - Processes may be explicitly or implicitly defined and are packaged in architectures
- Signal - primary communication mechanism
  - Process executions result in new values being assigned to signals which are then accessible to other processes
  - Similarly, a signal may be accessed by a process in another architecture by connecting the signal to ports in the entities associated with the two architectures

Example signal assignment statement :

```
Output <= My_id + 10;
```

Entity Declarations

- The primary purpose of the entity is to declare the signals in the component's interface
  - The interface signals are listed in the PORT clause
    - In this respect, the entity is akin to the schematic symbol for the component




```
ENTITY half_adder IS

    GENERIC(prop_delay : TIME := 10 ns);

    PORT( x, y, enable : IN BIT;
          carry, result : OUT BIT);

END half_adder;
```

Entity Declarations: Port Clause

- PORT clause declares the interface signals of the object to the outside world
- Three parts of the PORT clause
  - Name
  - Mode
  - Data type

```
PORT (signal_name : mode data_type);
```

- Example PORT clause:

```
PORT ( input : IN BIT_VECTOR(3 DOWNTO 0);
       ready, output : OUT BIT );
```

  - Port signals (i.e. 'ports') of the same mode and type or subtype may be declared on the same line

Entity Declarations: Port Clause (2)

- The port mode of the interface describes the direction in which data travels with respect to the component
- Port modes
  - In
    - Data comes in this port and can only be read
  - Out
    - Data travels out this port
  - Buffer
    - Data may travel in either direction, but only one signal driver may be on at any one time
  - Inout
    - Data may travel in either direction with any number of active drivers allowed; requires a Bus Resolution Function
  - Linkage
    - Direction of data flow is unknown

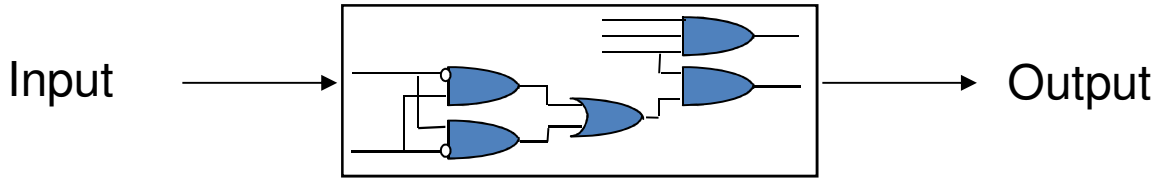
Architecture Bodies

- Describe the operation of the component
- Consist of two parts:
  - Declarative part - includes necessary declarations
    - For example, type declarations, signal declarations, component declarations, subprogram declarations
  - Statement part - includes statements that describe organization and/or functional operation of component
    - For example, concurrent signal assignment statements, process statements, component instantiation statements

```
ARCHITECTURE half_adder_d OF half_adder IS
     SIGNAL xor_res : BIT;      -- architecture declarative part
     BEGIN                      -- begins architecture statement part
       carry <= enable AND (x AND y);
       result <= enable AND xor_res;
       xor_res <= x XOR y;
     END half_adder_d;
```

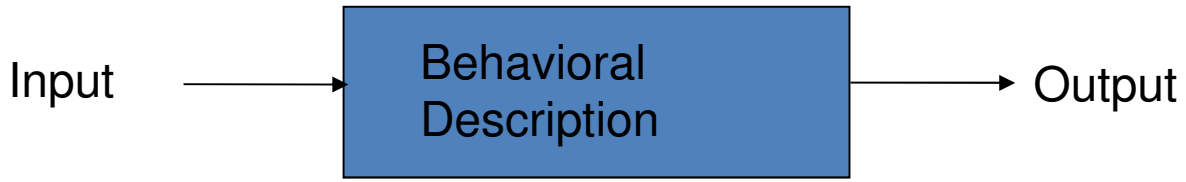
Structural Descriptions

- Pre-defined VHDL components are instantiated and connected together
- Structural descriptions may connect simple gates or complex, abstract components
- VHDL provides mechanisms to support hierarchical description
- VHDL provides mechanisms to describe highly repetitive structures easily



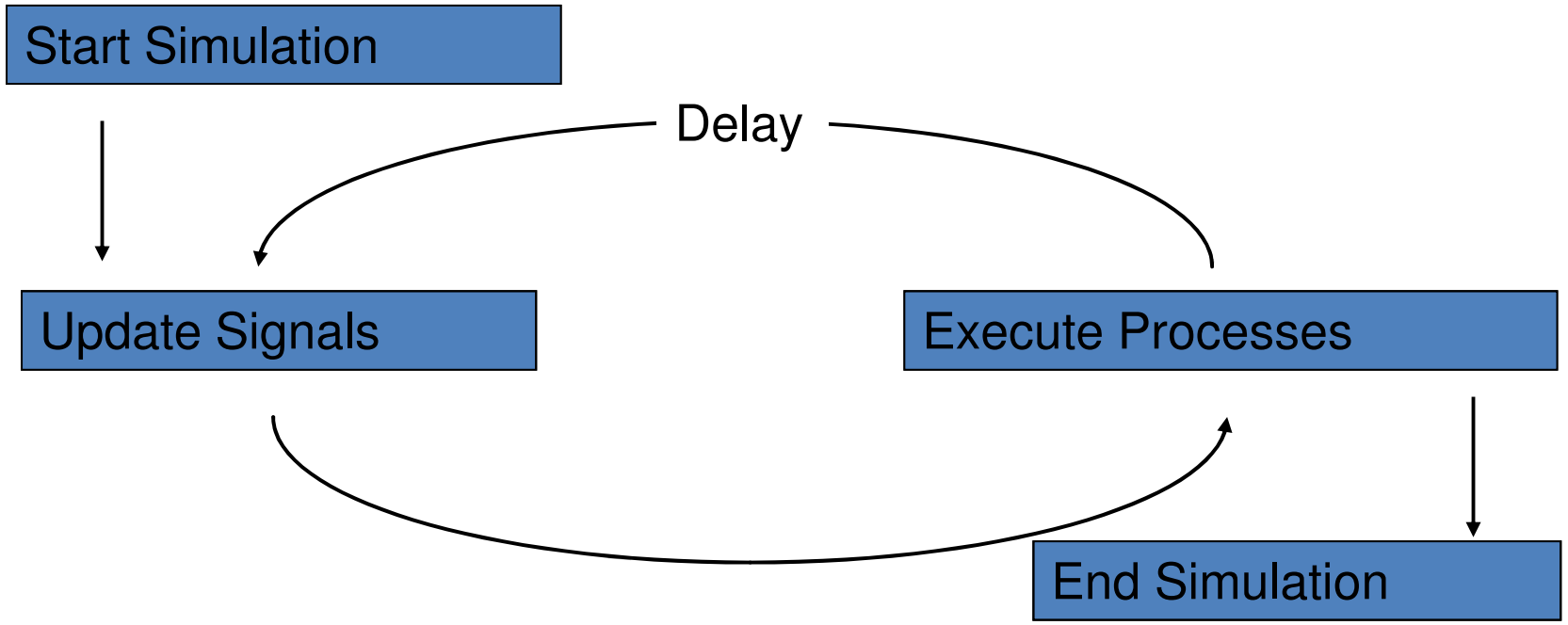
Behavioral Descriptions

- VHDL provides two styles of describing component behavior
  - Data Flow: concurrent signal assignment statements
  - Behavioral: processes used to describe complex behavior by means of high-level language constructs
    - Variables, loops, if-then-else statements
- A behavioral model may bear little resemblance to system implementation
  - Structure not necessarily implied



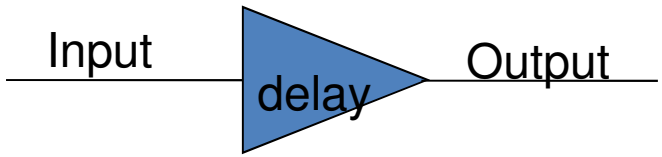
Timing Model

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



Delay Types

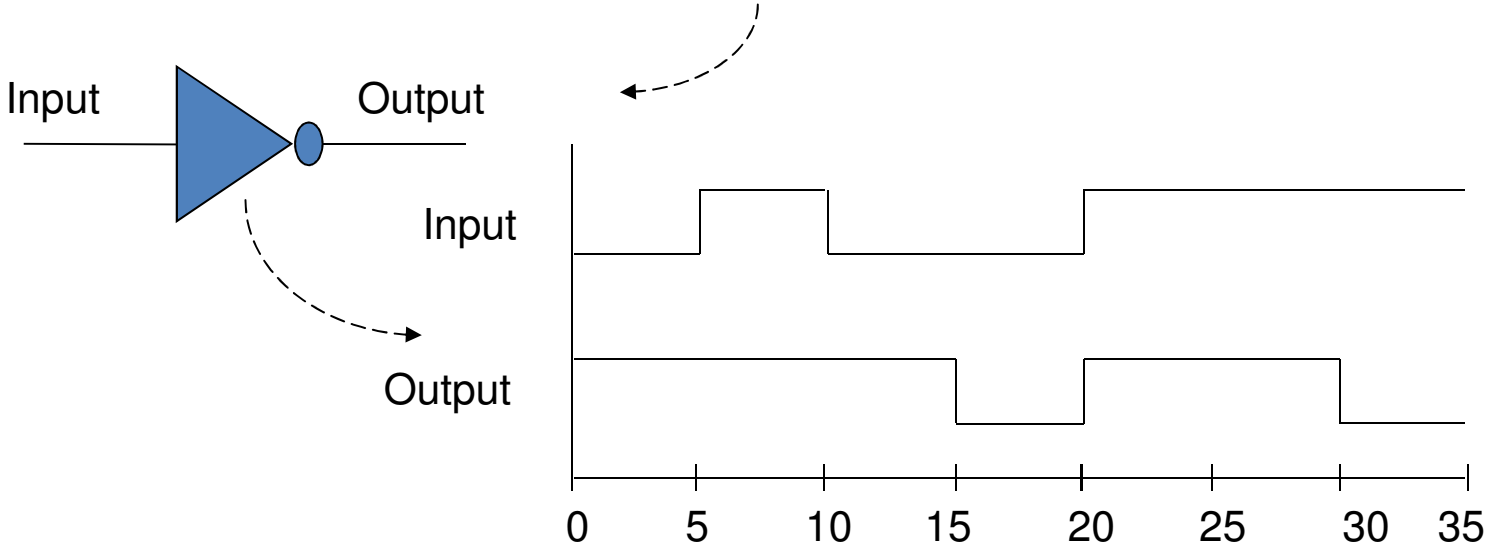
- All VHDL signal assignment statements prescribe an amount of time that must transpire before the signal assumes its new value
- Delay forms
  - Transport
    - Prescribes propagation delay only
  - Inertial
    - Prescribes propagation delay and minimum input pulse width
  - Delta
    - The default if no delay time is explicitly specified



Transport Delay

- Transport delay must be explicitly specified
  - I.e. keyword “TRANSPORT” must be used
- Signal assumes its new value after specified delay

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



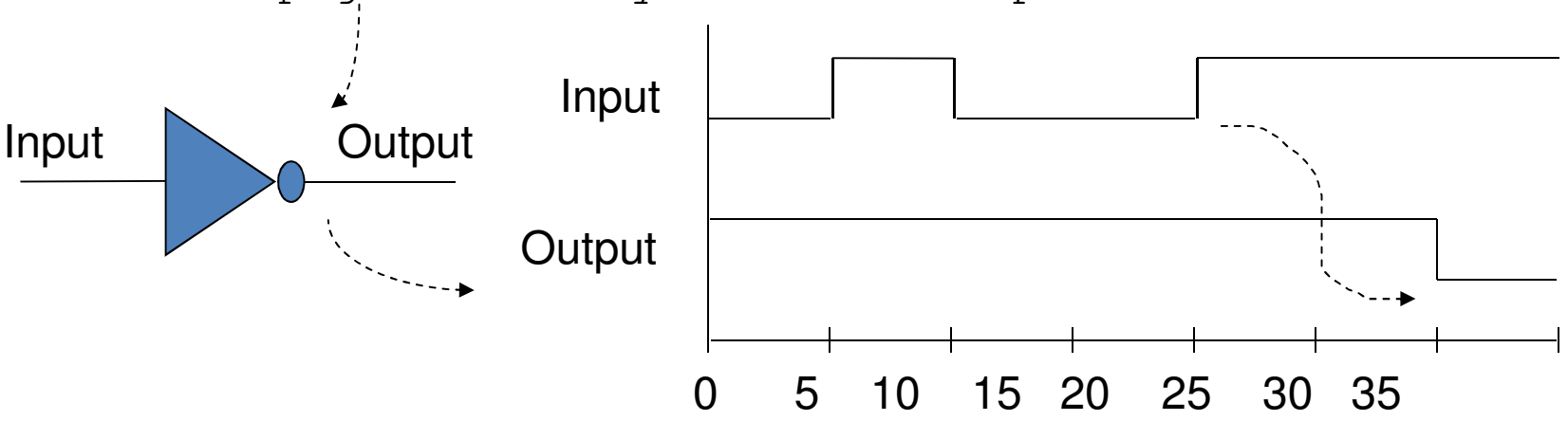
Inertial Delay

- Provides 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:

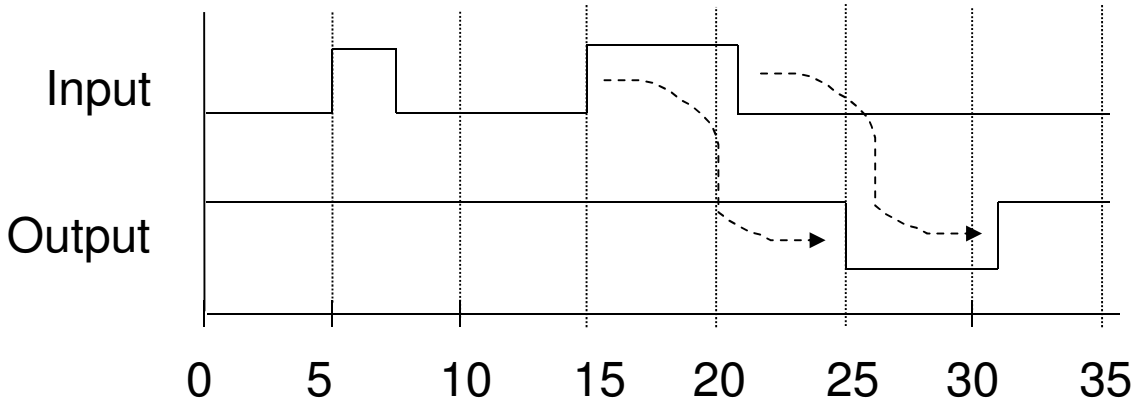
```
Output <= NOT Input AFTER 10 ns;
-- Propagation delay and minimum pulse width are 10ns
```



Inertial Delay (2)

- Example of gate with ‘inertia’ smaller than propagation delay
  - Inverter with propagation delay of 10ns which suppresses pulses shorter than 5ns

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



- The REJECT feature is new to VHDL 1076-1993

Delta Delay

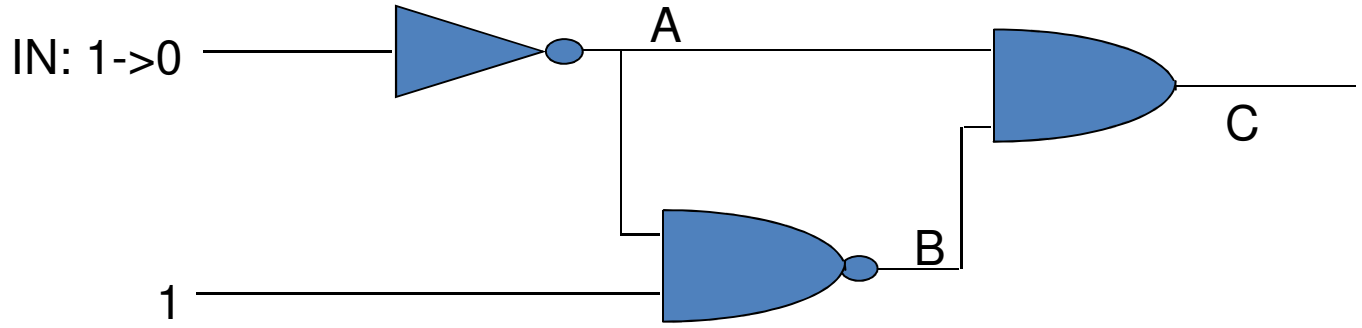
- Default signal assignment propagation delay if no delay is explicitly prescribed
  - VHDL signal assignments do not take place immediately
  - Delta is an infinite simulation time unit so that all signal assignments can result in signals assuming their values at a future time
  - E.g.

```
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

Delta Delay: An Example without Delta Delay

- The behavior of C



NAND gate evaluated first:

IN: 1->0

A: 0->1

B: 1->0

C: 0->0

AND gate evaluated first:

IN: 1->0

A: 0->1

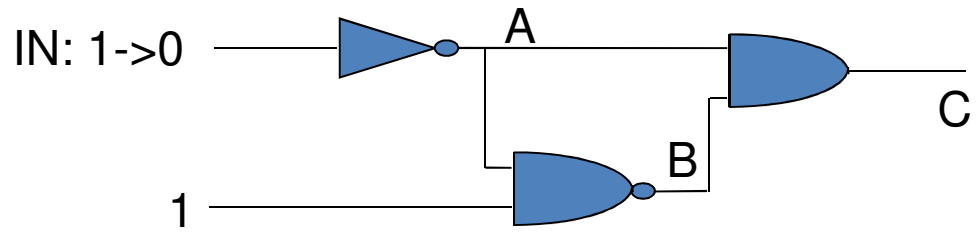
C: 0->1

B: 1->0

C: 1->0

Delta Delay: An Example without Delta Delay (2)

- The behavior of C



Using delta delay scheduling		
Time	Delta	Event
0 ns	1	IN: 1->0
	2	eval INVERTER A: 0->1
	3	eval NAND, AND B: 1->0 C: 0->1
	4	eval AND C: 1->0
1 ns		

Data Types

- All declarations of VHDL ports, signals, and variables must specify their corresponding type or subtype
- Types
  - Access
  - Scalar
    - Integer
    - Real
    - Enumerated
    - Physical
  - Composite
    - Array
    - Record

VHDL Data Types: Scalar Types

- Integer
  - Minimum range for any implementation as defined by standard:

    - 2,147,483,647 to 2,147,483,647
  - Example assignments to a variable of type integer:

```
ARCHITECTURE test_int OF test IS
BEGIN
    PROCESS (X)
      VARIABLE a: INTEGER;
    BEGIN
      a := 1;  -- OK
      a := -1;  -- OK
      a := 1.0;  -- illegal
    END PROCESS;
END test_int;
```

VHDL Data Types: Scalar Types (2)

- Real
  - Minimum range for any implementation as defined by standard: -1.0E38 to 1.0E38
  - Example assignments to a variable of type real :

```
ARCHITECTURE test_real OF test IS
BEGIN
    PROCESS (X)
      VARIABLE a: REAL;
    BEGIN
      a := 1.3;  -- OK
      a := -7.5;  -- OK
      a := 1;  -- illegal
      a := 1.7E13;  -- OK
      a := 5.3 ns;  -- illegal
    END PROCESS;
END test_real;
```

VHDL Data Types: Scalar Types (3)

- Enumerated
  - User specifies list of possible values
  - Example declaration and usage of enumerated data type:

```
TYPE binary IS ( ON, OFF );
... some statements ...
ARCHITECTURE test_enum OF test IS
BEGIN
   PROCESS (X)
     VARIABLE a: binary;
   BEGIN
     a := ON;  -- OK
     ... more statements ...
     a := OFF;  -- OK
     ... more statements ...
   END PROCESS;
END test_enum;
```

VHDL Data Types: Scalar Types (4)

- Physical
  - Requires associated units
  - Range must be specified
  - Example of physical type declaration:

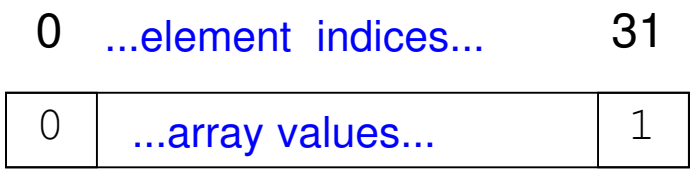
```
TYPE resistance IS RANGE 0 TO 10000000

UNITS
ohm;  -- ohm
Kohm = 1000 ohm;  -- i.e. 1 KΩ
Mohm = 1000 kohm;  -- i.e. 1 MΩ
END UNITS;
```

Time is the only physical type predefined in VHDL standard

VHDL Data Types: Composite Types

- Array
  - Used to group elements of the same type into a single VHDL object
  - Range may be unconstrained in declaration
    - Range would then be constrained when array is used
  - Example declaration for 1D array (vector):

```
TYPE data_bus IS ARRAY(0 TO 31) OF BIT;
```

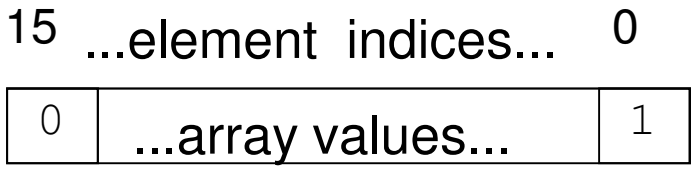



```
VARIABLE X : data_bus;
VARIABLE Y : BIT;

Y := X(12);  -- Y gets value of element at index 12
```

VHDL Data Types: Composite Types (2)

- Example of 1D array using DOWNTO :

```
TYPE reg_type IS ARRAY(15 DOWNTO 0) OF BIT;
```




```
VARIABLE X : reg_type;
VARIABLE Y : BIT;

Y := X(4);  -- Y gets value of element at index 4
```

- DOWNTO keyword must be used if leftmost index is greater than rightmost index
  - e.g. 'Big-Endian' bit ordering

VHDL Data Types: Composite Types (3)

- Records
  - Used to group elements of possibly different types into a single VHDL object
  - Elements are indexed via field names
  - Examples of record declaration and usage :

```
TYPE binary IS ( ON, OFF );
TYPE switch_info IS
    RECORD
      status : BINARY;
      IDnumber : INTEGER;
END RECORD;

VARIABLE switch : switch_info;
switch.status := ON;  -- status of the switch
switch.IDnumber := 30;  -- e.g. number of the switch
```

VHDL Data Types: Access Type

- Access
  - Analogous to pointers in other languages
  - Allows dynamic allocation of storage
  - Useful for implementing queues, fifos, etc.

VHDL Data Types: Subtypes

- Subtype
  - Allows user defined constraints on a data type
    - e.g. a subtype based on an unconstrained VHDL type
  - May include entire range of base type
  - Assignments that are out of the subtype range are illegal
    - Range violation detected at run time rather than compile time because only base type is checked at compile time
  - Subtype declaration syntax:

```
SUBTYPE name IS base_type RANGE <user range>;
```

  - Subtype example:

```
SUBTYPE first_ten IS INTEGER RANGE 0 TO 9;
```

VHDL Data Types: Summary

- All declarations of VHDL ports, signals, and variables must include their associated type or subtype
- Three forms of VHDL data types
  - Access - pointers for dynamic storage allocation
  - Scalar - includes Integer, Real, Enumerated, and Physical
  - Composite - includes Array, and Record
- A set of built-in data types are defined in VHDL standard
  - User can also define own data types and subtypes

VHDL Objects

- Object types
  - Constants
  - Variables
  - Signals
  - Files

VHDL Objects (2)

- The scope of an object
  - Objects declared in a package are available to all VHDL descriptions that use that package
  - Objects declared in an entity are available to all architectures associated with that entity
  - Objects declared in an architecture are available to all statements in that architecture
  - Objects declared in a process are available only within that process

VHDL Objects: Constants

- Name assigned to a specific value of a type
- Allow easy update and readability
- Declaration of constant may omit value so that the value assignment may be deferred
  - Facilitates reconfiguration
- Declaration syntax:

```
CONSTANT constant_name : type_name [:= value];
```

- Declaration examples:

```
CONSTANT PI : REAL := 3.14;
CONSTANT SPEED : INTEGER;
```

VHDL Objects: Variables

- Provide convenient mechanism for local storage
  - e.g. loop counters, intermediate values
- Scope is the process in which they are declared
  - VHDL '93 provides global variables
- All variable assignments take place immediately
  - No delta or user specified delay is incurred
- Declaration syntax:

```
VARIABLE variable_name : type_name [:= value];
```

- Declaration examples:

```
VARIABLE opcode : BIT_VECTOR(3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

VHDL Objects: Signals

- Used for communication between VHDL components
- Real, physical signals in system often mapped to VHDL signals
- ALL VHDL signal assignments require either delta cycle or user-specified delay before new value is assumed
- Declaration syntax:

```
SIGNAL signal_name : type_name [:= value];
```

- Declaration and assignment examples:

```
SIGNAL brdy : BIT;
brdy <= '0' AFTER 5ns, '1' AFTER 10ns;
```

Signals and Variables

- This example highlights the difference between signals and variables

```
ARCHITECTURE test1 OF mux IS
    SIGNAL x : BIT := '1';
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, x, y)
      BEGIN
        x <= in_sig XOR y;
        y <= in_sig XOR x;
      END PROCESS;
END test1;
```

```
ARCHITECTURE test2 OF mux IS
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, y)
      VARIABLE x : BIT := '1';
    BEGIN
      x := in_sig XOR y;
      y <= in_sig XOR x;
    END PROCESS;
END test2;
```

VHDL Objects: Signals vs. Variables

- A key difference between variables and signals is the assignment delay

```
ARCHITECTURE sig_ex OF test IS
    PROCESS (a, b, c, out_1)
    BEGIN
      out_1 <= a NAND b;
      out_2 <= out_1 XOR c;
    END PROCESS;
END sig_ex;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

VHDL Objects: Signals vs. Variables (2)

```
ARCHITECTURE var_ex OF test IS
BEGIN
    PROCESS (a, b, c)
    VARIABLE out_3 : BIT;
    BEGIN
      out_3 := a NAND b;
      out_4 <= out_3 XOR c;
    END PROCESS;
END var_ex;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	0
1+d	1	1	1	0	1

VHDL Objects: Files

- Files provide a way for a VHDL design to communicate with the host environment
- File declarations make a file available for use to a design
- Files can be opened for reading and writing
  - In VHDL87, files are opened and closed when their associated objects come into and out of scope
  - In VHDL93 explicit FILE_OPEN() and FILE_CLOSE() procedures were added
- The package STANDARD defines basic file I/O routines for VHDL types
- The package TEXTIO defines more powerful routines handling I/O of text files

Simulation Cycle Revisited: Sequential vs. Concurrent Statements

- VHDL is inherently a concurrent language
  - All VHDL processes execute concurrently
  - Concurrent signal assignment statements are actually one-line processes
- VHDL statements execute sequentially within a process
- Concurrent processes with sequential execution within a process offers maximum flexibility
  - Supports various levels of abstraction
  - Supports modeling of concurrent and sequential events as observed in real systems

Concurrent Statements

- Basic granularity of concurrency is the process
  - Processes are executed concurrently
  - Concurrent signal assignment statements are one-line processes
- Mechanism to achieve concurrency
  - Processes communicate with each other via signals
  - Signal assignments require delay before new value is assumed
  - Simulation time advances when all active processes complete
  - Effect is concurrent processing
    - i.e. order in which processes are actually executed by simulator does not affect behavior
- Concurrent VHDL statements
  - Block, process, assert, signal assignment, procedure call, component instantiation

Sequential Statements

- Statements inside a process are executed sequentially

```
ARCHITECTURE sequential OF test_mux IS
BEGIN
    select_proc : PROCESS (x,y)
    BEGIN
    IF (select_sig = '0') THEN
        z <= x;
    ELSIF (select_sig = '1') THEN
        z <= y;
    ELSE
        z <= "XXXX";
    END IF;
    END PROCESS select_proc;
END sequential;
```

Packages and Libraries

- User defined constructs declared inside architectures and entities are not visible to other VHDL components
  - Scope of subprograms, user defined data types, constants, and signals is limited to the VHDL components in which they are declared
- Packages and libraries provide the ability to reuse constructs in multiple entities and architectures
  - Items declared in packages can be used (i.e. included) in other VHDL components

Packages

- Packages consist of two parts
  - Package declaration - declarations of objects defined in the package
  - Package body - necessary definitions for certain objects in package declaration
    - e.g. subprogram descriptions
- Examples of VHDL items included in packages
  - Basic declarations
    - Types, subtypes
    - Constants
    - Subprograms
    - Use clause
  - Signal declarations
  - Attribute declarations
  - Component declarations

Packages: Declaration

- An example of a package declaration

```
PACKAGE my_stuff IS
    TYPE binary IS ( ON, OFF );
    CONSTANT PI : REAL := 3.14;
    CONSTANT My_ID : INTEGER;
    PROCEDURE add_bits3(SIGNAL a, b, en : IN BIT;
              SIGNAL temp_result, temp_carry : OUT BIT);
END my_stuff;
```

- Some items only require declaration while others need further detail provided in subsequent package body
  - For type and subtype definitions, declaration is sufficient
  - Ssubprograms require declarations and descriptions

Packages: Package Body

- The package body includes the necessary functional descriptions needed for objects declared in the package declaration
  - e.g. subprogram descriptions, assignments to constants

```
PACKAGE BODY my_stuff IS
    CONSTANT My_ID : INTEGER := 2;

    PROCEDURE add_bits3(SIGNAL a, b, en : IN BIT;
            SIGNAL temp_result, temp_carry : OUT BIT) IS
    BEGIN    -- this function can return a carry
        temp_result <= (a XOR b) AND en;
        temp_carry <= a AND b AND en;
    END add_bits3;
END my_stuff;
```

Packages: Use Clause

- Packages must be made visible before their contents can be used
  - The USE clause makes packages visible to entities, architectures, and other packages

```
-- use only the binary and add_bits3
declarations
USE my_stuff.binary,
my_stuff.add_bits3;

... ENTITY declaration...
... ARCHITECTURE declaration ...
```

```
-- use all of the declarations in
package my_stuff
USE my_stuff.ALL;

... ENTITY declaration...
... ARCHITECTURE declaration ...
```

Libraries

- Analogous to directories of files
  - VHDL libraries contain analyzed (i.e. compiled) VHDL entities, architectures, and packages
- Facilitate administration of configuration and revision control
  - Libraries of previous designs
- Libraries accessed via an assigned logical name
  - Current design unit is compiled into the Work library
  - Both work and STD libraries are always available
  - Many other libraries usually supplied by VHDL simulator vendor
    - Proprietary libraries and IEEE standard libraries

Attributes

- Attributes provide information about certain items in VHDL
  - Types, subtypes, procedures, functions, signals, variables, constants, entities, architectures, configurations, packages, components
- General form of attribute use:

```
name'attribute_identifier  -- read as “tick”
```

- VHDL has several predefined attributes:
  - X'EVENT - TRUE when there is an event on signal X
  - X'LAST_VALUE - returns the previous value of signal X
  - Y'HIGH - returns the highest value in the range of Y
  - X'STABLE(t) - TRUE when no event has occurred on signal X in the past ‘t’ time

Attributes: Register Example

- The following example shows how attributes can be used to make an 8-bit register
- Specifications
  - Triggers on rising clock edge
  - Latches only on enable high
  - Has a data setup time of x_setup
  - Has propagation delay of prop_delay

```
ENTITY 8_bit_reg IS
    GENERIC (x_setup, prop_delay : TIME);
    PORT(enable, clk : IN qsim_state;
         a : IN qsim_state_vector(7 DOWNTO 0);
        b : OUT qsim_state_vector(7 DOWNTO 0));
END 8_bit_reg;
```

Attributes: Register Example (2)

- The following architecture is a first attempt at the register
- The use of 'STABLE is used to detect setup violations in the data input

```
ARCHITECTURE first_attempt OF 8_bit_reg IS
BEGIN
    PROCESS (clk)
    BEGIN
    IF (enable = '1') AND a'STABLE(x_setup) AND
        (clk = '1') THEN
        b <= a AFTER prop_delay;
    END IF;
    END PROCESS;
END first_attempt;
```

Attributes: Register Example (3)

- The following architecture is a second and more robust attempt
- The use of 'LAST_VALUE ensures the clock is rising from a value of ‘0’

```
ARCHITECTURE behavior OF 8_bit_reg IS
BEGIN
        PROCESS (clk)
        BEGIN
        IF (enable = '1') AND a'STABLE(x_setup) AND
                (clk = '1') AND (clk'LAST_VALUE = '0') THEN
                    b <= a AFTER delay;
        END IF;
        END PROCESS;
END behavior;
```

- An ELSE clause could be added to define the behavior when the requirements are not met

Operators

- Operators can be chained to form complex expressions:

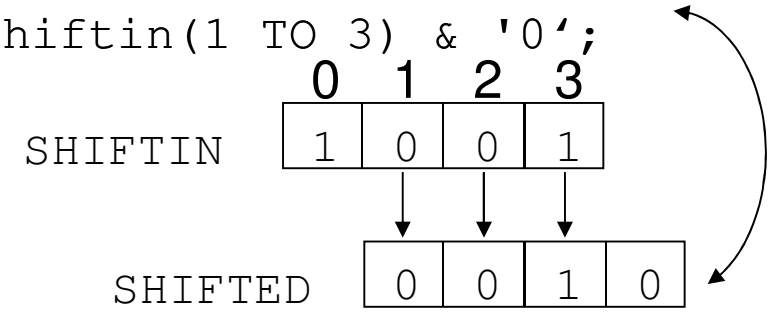
```
res <= a AND NOT(B) OR NOT(a) AND b;
```

  - Can use parentheses for readability and control the association of operators and operands
- Defined precedence levels in decreasing order:
  - Miscellaneous operators -- **, abs, not
  - Multiplication operators -- *, /, mod, rem
  - Sign operator -- +, -
  - Addition operators -- +, -, &
  - Shift operators -- sll, srl, sla, sra, rol, ror
  - Relational operators -- =, /=, <, <=, >, >=
  - Logical operators -- AND, OR, NAND, NOR, XOR, XNOR

Operators: Examples

- The concatenation operator &

```
VARIABLE shifted, shiftin : BIT_VECTOR(0 TO
  3);
...
shifted := shiftin(1 TO 3) & '0';
```



- The exponentiation operator **

```
x := 5**5  -- 5^5, OK
y := 0.5**3 -- 0.5^3, OK
x := 4**0.5 -- 4^0.5, Illegal
y := 0.5**(-2) -- 0.5^(-2), OK
```

Examples

- Design a library of logic gates
  - AND, OR, NAND, NOR, INV, etc.
- Include sequential elements
  - DFF, Register, etc.
- Include tri-state devices
- Use 4-valued logic
  - ‘X’, ‘0’, ‘1’, ‘Z’
  - Encapsulate global declarations in a package

Global Package

```
PACKAGE resources IS

   TYPE level IS ('X', '0', '1', 'Z'); -- enumerated type

   TYPE level_vector IS ARRAY (NATURAL RANGE <>) OF level;
   -- type for vectors (buses)

   SUBTYPE delay IS TIME; -- subtype for gate delays

   -- Function and procedure declarations go here

END resources;
```

Two Input AND Gate Example

```
USE work.resources.all;

ENTITY and2 IS

  GENERIC(trise : delay := 1
          tfall : delay := 8

  PORT(a, b : IN level;
       c : OUT level);
END and2;
```

```
ARCHITECTURE behav OF and2 IS

  BEGIN

    one : PROCESS (a,b)

      BEGIN
        IF (a = '1' AND b = '1') THE
          c <= '1' AFTER trise;
        ELSIF (a = '0' OR b = '0') T
          c <= '0' AFTER tfall;
        ELSE
          c<= 'X' AFTER (trise+tfall
        END IF;

    END PROCESS one;

END behav;
```

Tri-State Buffer Example

```
USE work.resources.all;

ENTITY tri_state IS

  GENERIC(trise : delay := 6
          tfall : delay := 5
          thiz  : delay := 8

  PORT(a : IN level;
       e : IN level;
       b : OUT level);

END tri_state;
```

```
ARCHITECTURE behav OF tri_state IS

  BEGIN

    one : PROCESS (a,e)

      BEGIN
        IF (e = '1' AND a = '1') THEN
                -- enabled and valid data
          b <= '1' AFTER trise;
        ELSIF (e = '1' AND a = '0') THEN
          b <= '0' AFTER tfall;
        ELSIF (e = '0') THEN -- disabled
          b <= 'Z' AFTER thiz;
        ELSE -- invalid data or enable
          b <= 'X' AFTER (trise+tfall)/2;
        END IF;

    END PROCESS one;

END behav;
```

D Flip Flop Example

```
USE work.resources.all;

ENTITY dff IS

  GENERIC(tprop : delay := 8 ns;
          tsu   : delay := 2 ns);

  PORT(d      : IN level;
       clk    : IN level;
       enable : IN level;
       q      : OUT level;
       qn     : OUT level);

END dff;
```

```
ARCHITECTURE behav  OF dff IS
  BEGIN
   one : PROCESS (clk)
     BEGIN
       -- check for rising clock edge
       IF ((clk = '1' AND clk'LAST_VALUE = '0')
            AND enable = '1') THEN -- ff enabled
         -- first, check setup time requirement
         IF (d'STABLE(tsu)) THEN
           -- check valid input data
           IF (d = '0') THEN
             q <= '0' AFTER tprop;
             qn <= '1' AFTER tprop;
           ELSIF (d = '1') THEN
             q <= '1' AFTER tprop;
             qn <= '0' AFTER tprop;
           ELSE -- else invalid data
             q <= 'X';
             qn <= 'X';
           END IF;
         ELSE -- else violated setup time requirement
           q <= 'X';
           qn <= 'X';
         END IF;
       END IF;
   END PROCESS one;
END behav;
```

Summary

- VHDL is a worldwide standard for the description and modeling of digital hardware
- VHDL gives the designer many different ways to describe hardware
- Familiar programming tools are available for complex and simple problems
- Sequential and concurrent modes of execution meet a large variety of design needs
- Packages and libraries support design management and component reuse

Putting It All Together