A Quick Introduction to VHDL

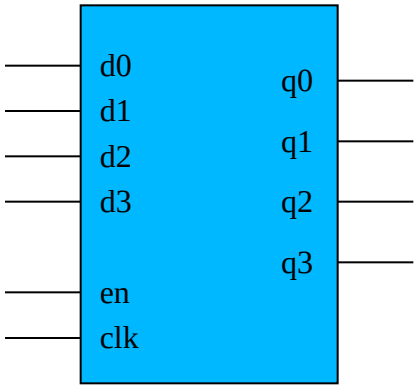
HDL

- Hardware Description Language
  - A high level programming language used to describe the structural and behavioral of digital circuits or electronic hardware systems
- Two commonly used HDLs
  - VHDL
  - Verilog

VHDL

- VHSIC Hardware Description Language
  - VHSIC: Very High Speed Integrated Circuit
- IEEE Standard in 1987
  - Revised in 1993
- A language for modeling and developing a digital system
  - documentation
  - requirements specification
  - testing
  - formal verification
  - synthesis

Example: A four-input register




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

Library and package declaration

```
-- here is the declaration of entity
entity reg4 is
  port ( d0, d1, d2, d3, en, clk : in bit;
         q0, q1, q2, q3 : out  bit );
end entity reg4;
```

Entity declaration

```
-- here is the body of the architecture
architecture behav of reg4 is
begin
  storage : process is
    variable stored_d0, stored_d1, stored_d2, stored_d3 : bit;
  begin
    if en = '1' and clk = '1' then
      stored_d0 := d0;
      stored_d1 := d1;
      stored_d2 := d2;
      stored_d3 := d3;
    end if;
    q0 <= stored_d0 after 5 ns;
    q1 <= stored_d1 after 5 ns;
    q2 <= stored_d2 after 5 ns;
    q3 <= stored_d3 after 5 ns;
    wait on d0, d1, d2, d3, en, clk;
  end process storage;
end architecture behav;
```

Architecture body

FIU
Florida International University

VHDL basic

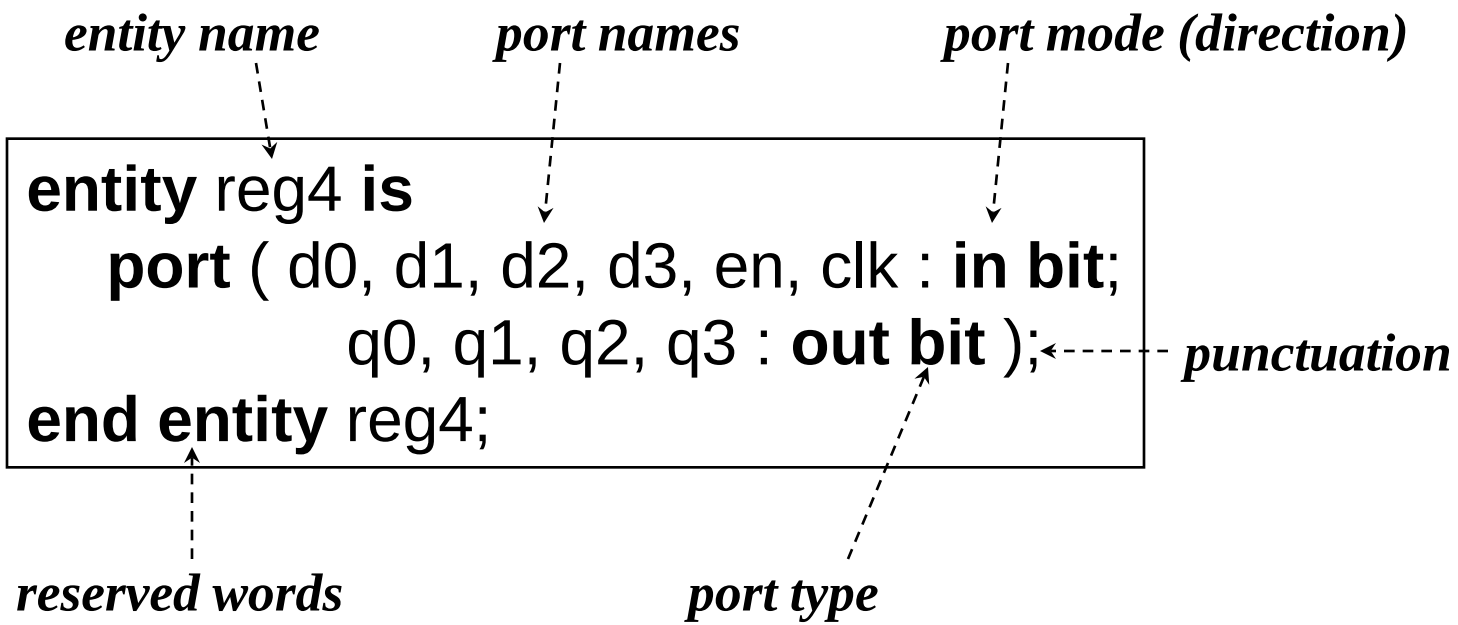
- Describing the component
  - Entity declaration
    - Name, Input/output ports and their types
  - Architecture body
    - Describe the functions or behaviors of the component

Packages and Libraries

- Packages are repositories for type definitions, procedures, and functions
  - User defined vs. system packages
  - Standardized packages
    - IEEE 1164 (data types)
    - IEEE 1076.3 (numeric)
    - IEEE 1076.4 (timing)
- Libraries are design units stored in the physical directories
  - When a design is analyzed, it is stored in the *working* library
  - If we need to access units stored in other libraries, they are called as the *resource* library
  - We use "*use*" clause to avoid having to write the library name each time.

Entity Declaration

- Describes the input/output *ports* of a module



Architecture Body

- Describes an implementation/function of an entity
  - may be several per entity
- *Behavioral* architecture
  - describes the algorithm performed by the module
  - contains
    - *process statements*, each containing
      - *sequential statements*, including
        - *signal assignment statements* and
        - *wait statements*

The Behavior Representation

```
architecture behav of reg4 is
begin
    process (d0, d1, d2, d3, en, clk)
        variable stored_d0, stored_d1, stored_d2, stored_d3 : bit;
    begin
        if en = '1' and clk = '1' then
            stored_d0 := d0;
            stored_d1 := d1;
            stored_d2 := d2;
            stored_d3 := d3;
        end if;
        q0 <= stored_d0 after 5 ns;
        q1 <= stored_d1 after 5 ns;
        q2 <= stored_d2 after 5 ns;
        q3 <= stored_d3 after 5 ns;
    end process;
end behav;
```

sensitivity list

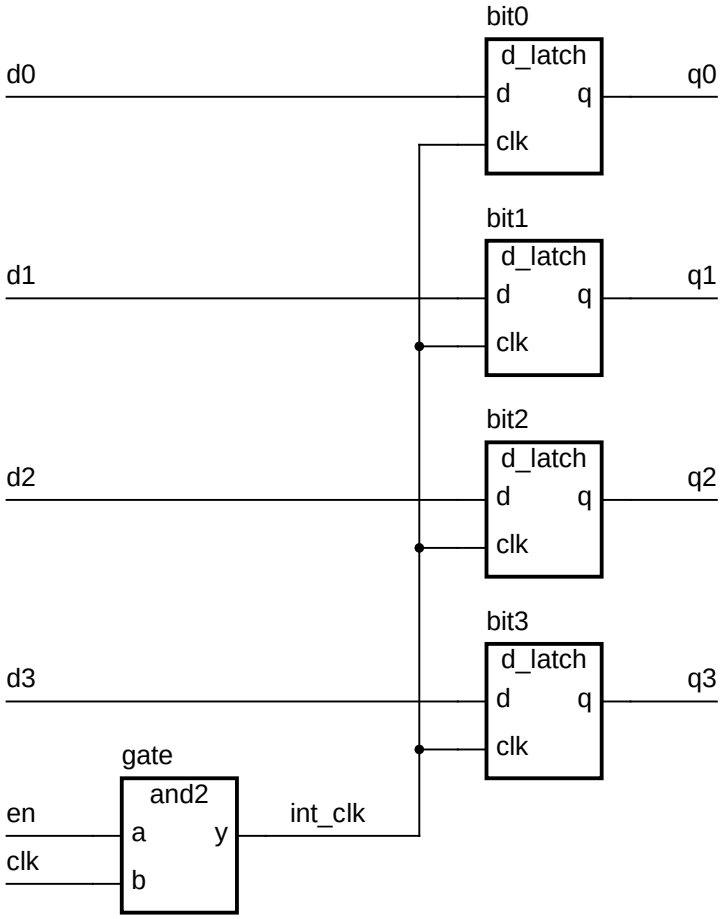
notice := syntax used for equating values from signals...

simulates real-world propagation delays.

The Structural Representation

- Implements the module as a composition of subsystems
- Contains
  - *signal declarations*, for internal interconnections
    - the entity ports are also treated as signals
  - *component instances*
    - instances of previously declared entity/architecture pairs
  - *port maps* in component instances
    - connect signals to component ports

The Schematic of reg4



Structural Architecture

- First declare D-latch and and-gate entities and architectures

notice semicolon placements -- odd as it is, omit from last statement

```
entity d_latch is
    port ( d, clk : in bit;  q : out bit );
end entity d_latch;

architecture basic of d_latch is
begin
    process (clk, d)
    begin
        if clk = '1' then
            q <= d after 2 ns;
        end if;
    end process;
end basic;
```

```
entity and2 is
    port ( a, b : in bit;  y : out bit );
end entity and2;

architecture basic of and2 is
begin
    process (a, b)
    begin
        y <= a and b after 2 ns;
    end process ;
end basic;
```

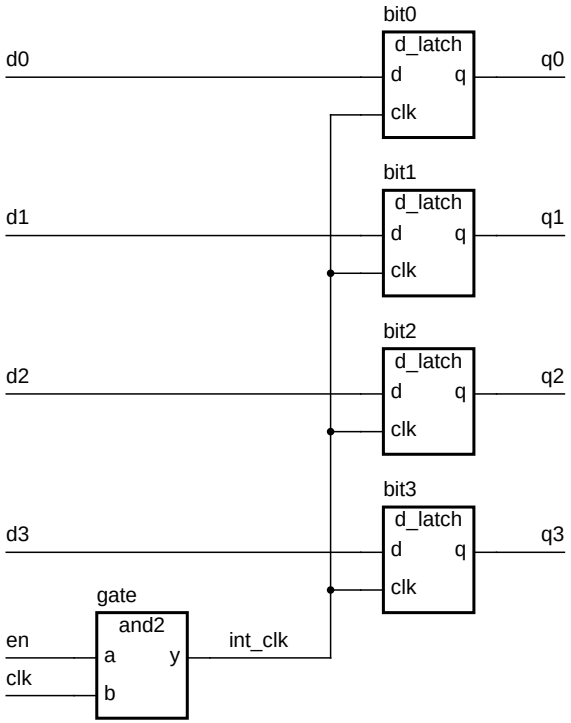
Structural Architecture (Cont'd)

- Declare corresponding components in register architecture body

```
architecture struct of reg4 is
    component d_latch
        port ( d, clk : in bit;  q : out bit );
    end component;
    component and2
        port ( a, b : in bit;  y : out bit );
    end component;
    signal int_clk : bit;
...
```

Structural Architecture (Cont'd)

- Now use them to implement the register




```
...
begin
    bit0 : d_latch
        port map ( d0, int_clk, q0 );
    bit1 : d_latch
        port map ( d1, int_clk, q1 );
    bit2 : d_latch
        port map ( d2, int_clk, q2 );
    bit3 : d_latch
        port map ( d3, int_clk, q3 );
    gate : and2
        port map ( en, clk, int_clk );
end struct;
```

Structural Architecture (Cont'd)

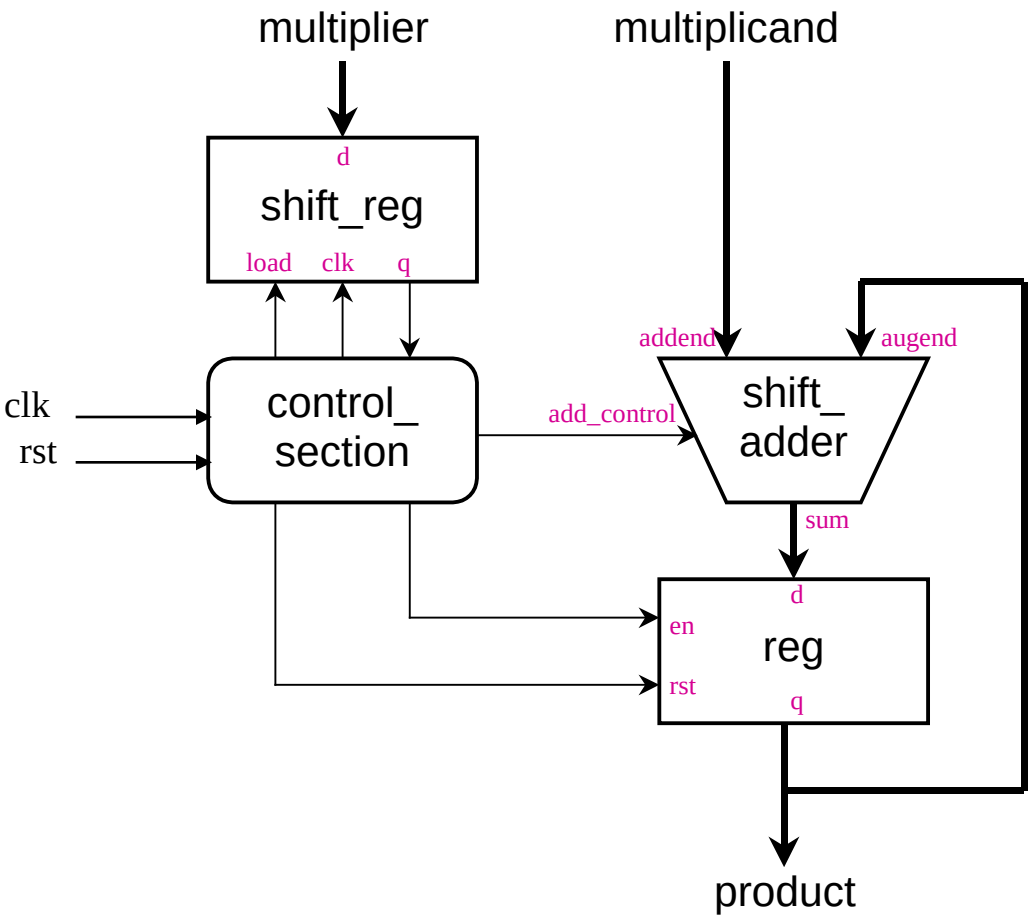
```
entity d_latch is
  port ( d, clk : in bit;  q : out bit );
end d_latch;

architecture basic of d_latch is
begin
  latch_behavior : process is
  begin
    if clk = '1' then
      q <= d after 2 ns;
    end if;
    wait on clk, d;
  end process latch_behavior;
end basic;

entity and2 is
  port ( a, b : in bit;  y : out bit );
end and2;

architecture basic of and2 is
begin
  and2_behavior : process is
  begin
    y <= a and b after 2 ns;
    wait on a, b;
  end and2_behavior;
end basic;

architecture struct of reg4 is
  signal int_clk : bit;
begin
  bit0 : d_latch
    port map (d0, int_clk, q0);
  bit1 : d_latch
    port map (d1, int_clk, q1);
  bit2 : d_latch
    port map (d2, int_clk, q2);
  bit3 : d_latch
    port map (d3, int_clk, q3);
  gate : and2
    port map (en, clk, int_clk);
end architecture struct;
```

FIU
Florida International University

Mixed Behavior and Structure

- An architecture can contain both behavioral and structural parts
  - process statements and component instances
    - collectively called *concurrent statements*
  - processes can read and assign signals
- Example: register-transfer-level (RTL) Model
  - data path described structurally
  - control section described behaviorally

Mixed Example



Mixed Example

```
entity multiplier is
        port ( clk, reset : in bit;
                multiplicand, multiplier : in integer;
                product : out integer );
end multiplier;

architecture mixed of mulitplier is
        signal partial_product, full_product : integer;
        signal arith_control, result_en, mult_bit, mult_load : bit;
begin
        arith_unit : entity work.shift_adder(behavior)
                port map ( addend => multiplicand,
                                augend => full_product,
                                sum => partial_product,
                                add_control => arith_control );
        result : entity work.reg(behavior)
                port map ( d => partial_product,
                                q => full_product,
                                en => result_en,
                                reset => reset );
        multiplier_sr : entity work.shift_reg(behavior)
                port map ( d => multiplier,
                                q => mult_bit,
                                load => mult_load,
                                clk => clk );
        product <= full_product;
        process (clk, reset)
-- variable declarations for control_section
-- …
        begin
-- sequential statements to assign values to control signals
        -- …
        end process;
end mixed;
```

Test Benches

- An enclosed model for testing a developed VHDL model by simulation
  - Simulating with "*signal generators*"
  - Observing with "*probes*"

- Include
  - An architecture body containing an instance of the designed to be tested
  - Test sequences with signals connected to the design

Example

```
entity test_bench is
end entity test_bench;

architecture test_reg4 of test_bench is
  signal d0, d1, d2, d3, en, clk, q0, q1, q2, q3 : bit;
begin
  dut : entity reg4(behav)
    port map ( d0, d1, d2, d3, en, clk, q0, q1, q2, q3 );

  stimulus : process is
  begin
    d0 <= '1';  d1 <= '1';  d2 <= '1';  d3 <= '1';
    en <= '0';  clk <= '0';
    wait for 20 ns;
    en <= '1';  wait for 20 ns;
    clk <= '1';  wait for 20 ns;
    d0 <= '0';  d1 <= '0';  d2 <= '0';  d3 <= '0';  wait for 20 ns;
    en <= '0';  wait for 20 ns;
    -- . . .
    wait;
  end process stimulus;
end architecture test_reg4;
```

Instance of Reg4

Test sequences

FIU
Florida International University

Analysis

- Check for syntax and logic errors
  - syntax: grammar of the language
  - logic: how your model responds to stimuli
- Analyze each *design unit* separately
  - entity declaration
  - architecture body
  - …
  - put each design unit in a separate file -- *helps a lot.*
- Analyzed design units are placed in a *library*

Simulation

- Discrete event simulation
  - time advances in discrete steps
  - when signal values change—*events* occur
- A processes is *sensitive* to events on input signals
  - specified in wait statements or the sensitive list
  - resumes and schedules new values on output signals
    - schedules *transactions*
    - *event* on a signal if value changes

Learning a New Language

- Lexical Elements
  - Case insensitive
  - Comments
    - preceded by two consecutive dashes
    - end at the end of current line
  - Identifiers, reserved words, special symbols, numbers, characters, strings, bit strings
- Syntax
  - Simple data types, operators
    - Integer, floating, arrays, records, …
    - And, Or, >, <, +, –, shift, …
  - Sequential statements
    - if, case, while, for
  - Entity declarations, architecture bodies, signal assignment, process, wait, procedure

Reference

- Peter J. Ashenden, The Designer's Guide to VHDL, Morgan Kaufmann, 2002
- Sudhakar Yalamanchili, VHDL Starter's Guide, Prentice Hall, 2005
- J. Bkasker, A VHDL Primer, Prentice Hall, 1999
- www.google.com