ECE 485/585
Computer Organization and Design

Lecture 2: Introduction to VHDL
Fall 2022

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Outline

- Hardware Description Languages (HDL)
- VHDL Language
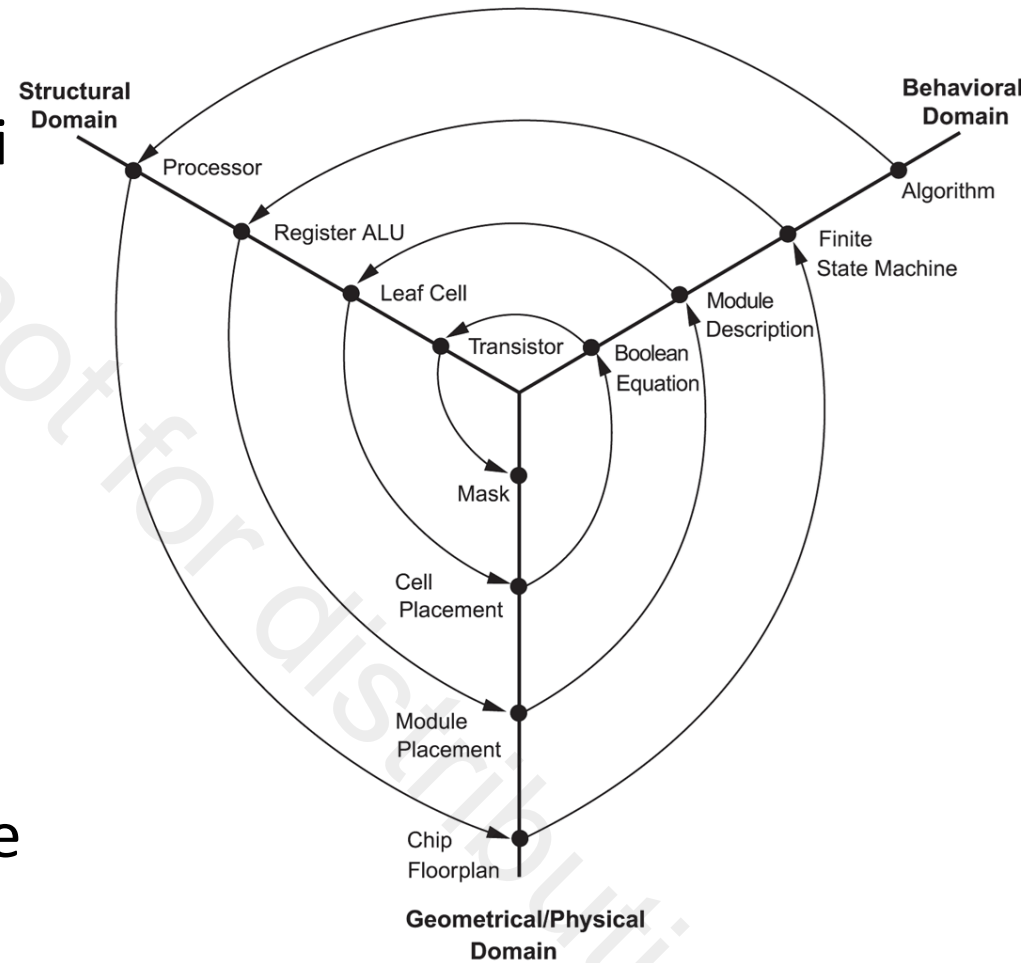- VHDL vs Verilog

VHDL Reference
    Peter J. Ashenden, VHDL Tutorial, Elsevier Science

Gajski-Kuhn Chart

- Gajski-Kuhn chart (or Y diagram) by Daniel Gajski and Robert Kuhn in 1983

- Three distinct design domains

- Levels of abstraction
  - **Top to bottom**

- Design process can be viewed as making transformations from one domain to another

3

Hardware Representation Languages

- Block Diagrams: Functional Units, Registers, Dataflows

- Register Transfer Diagrams: Choice of buses

- Flow charts

- State Diagrams

- Languages: Hardware Description Languages (HDLs)
  - Hardware module described as like programs with I/O ports, internal state & parallel execution of assignment statements

Hardware Description Languages

- Hardware Descriptions from these languages can be used as inputs to
    - Simulation systems : "software breadboard"
    - Synthesis systems: "generate hardware from high level description"

Simulation through HDL

- Physical breadboarding using discrete components no longer possible as designs reach higher levels of integration

- Simulation before implementation
    - Easy to test functionality
    - Test different what-if scenarios easily
    - Limited performance accuracy

Level of Description

- ## Architectural Simulation:
  - Models that programmers view at high level

- ## Functional/Behavioral:
  - More detailed view like block diagram

- ## Register/Transfer:
  - Datapath elements, Bus, registers, busses

- ## Logic:
  - Models in terms of logic gates

- ## Circuit:
  - Electrical behavior; accurate waveforms

Less abstract

More accurate

Slower simulation

VHDL Introduction

- Developed originally by DARPA

- International IEEE standard

- Hardware Description, Simulation, Synthesis

- Provides a mechanism for digital design and reusable design documentation

- Supports different description levels
  - Structural (interconnections of gates)
  - Dataflow (specifying logic equations)
  - Behavioral (specifying behavior)

- Top Down, technology dependent

VHDL

- VHDL: VHSIC Hardware Description Language
  - VHSIC: Very High Speed Integrated Circuit
- Goals
  - Describes the logical structure and function of digital systems
  - Support design, documentation, simulation and synthesis of hardware
  - Different abstraction levels: system level to gate level
- Concepts
  - Design entity
 - Time-based execution

Design Entity

- Design Entity describes a hardware component

Interface:
External
Characteristics

Architecture Body:
Internal Behavior
or Structure
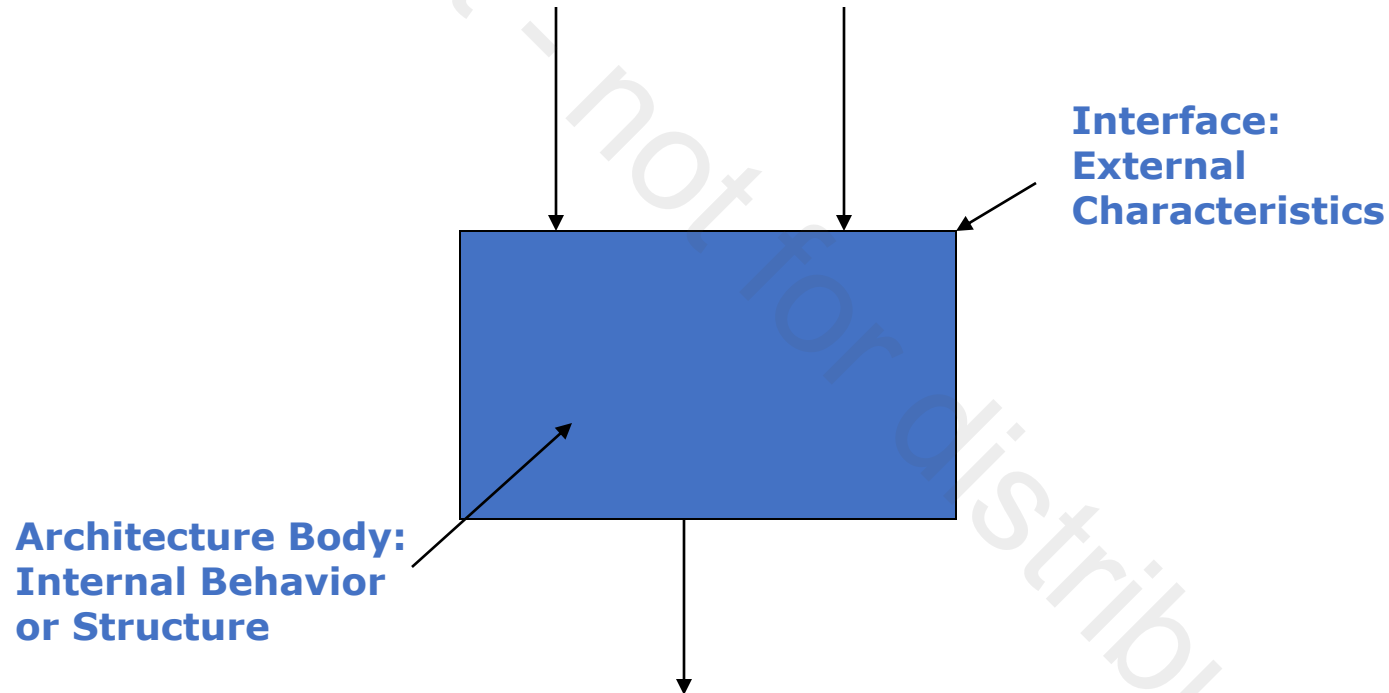
VHDL Entity

- Externally Visible Characteristics
  - Ports: channels of communication
    - Inputs, outputs, clocks, control
  - Generic parameters
    - Timing characteristics, size, fan out

- Internally Visible Characteristics
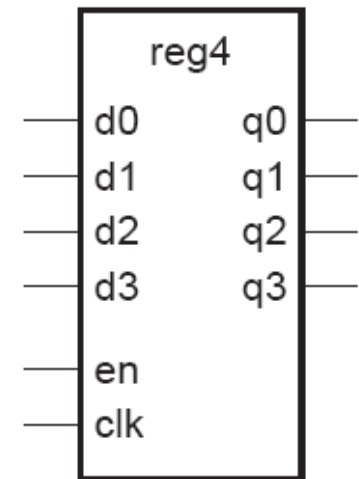 - Declarations
  - Assertions
  - Implementations

| Interface | ← | View to other modules |
| Architecture | ← | Details of implementation |
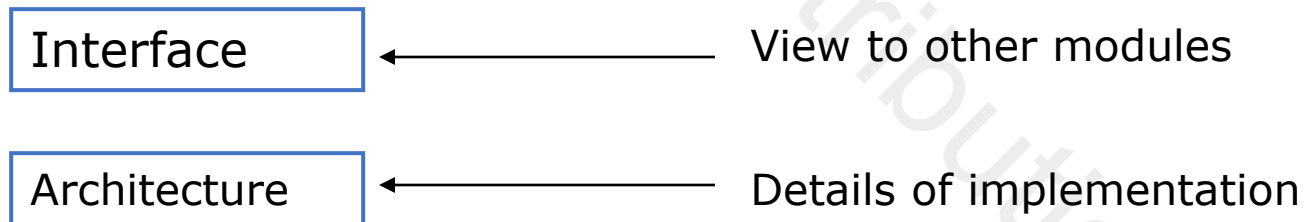
Entity Interface

- Produces a name for the entity and lists the inputs and outputs (ports)

- 4-bit register Example:

entity reg4 **is**
    port (d0, d1, d2, d3, en, clk : **in** bit;
                 q0, q1, q2, q3 : **out** bit);
end entity reg4;

```
           reg4
 ──  d0        q0  ──
 ──  d1        q1  ──
 ──  d2        q2  ──
 ──  d3        q3  ──

 ──  en
 ──  clk
```

Entity Interface

- Internal implementation of an entity is called **an architecture body of the entity**.

- An architecture body generally applies some operations to values on input ports, generating values to be assigned to output ports.

- The operations can be described either by **processes**, which contain sequential statements operating on values, or by a collection of components representing sub-circuits.
  - <u>Behavioral model</u>: describes the function in an abstract manner
  - <u>Structural model</u>: architecture body is composed only of interconnected subsystems

- Where the operation requires generation of intermediate values, these can be described using *signals*

Entity – Architecture Pair

```
entity Fulladder is
  port (X,Y,Cin: in bit; --inputs
          Cout,sum: out bit);   --outputs
end Fulladder;


architecture Equations of Fulladder is
begin
  sum <= X xor Y xor Cin after 10 ns;
  Cout <=(X and Y) or (X and Cin)
                      or (Y and Cin) after 10ns;
end Equations;
```
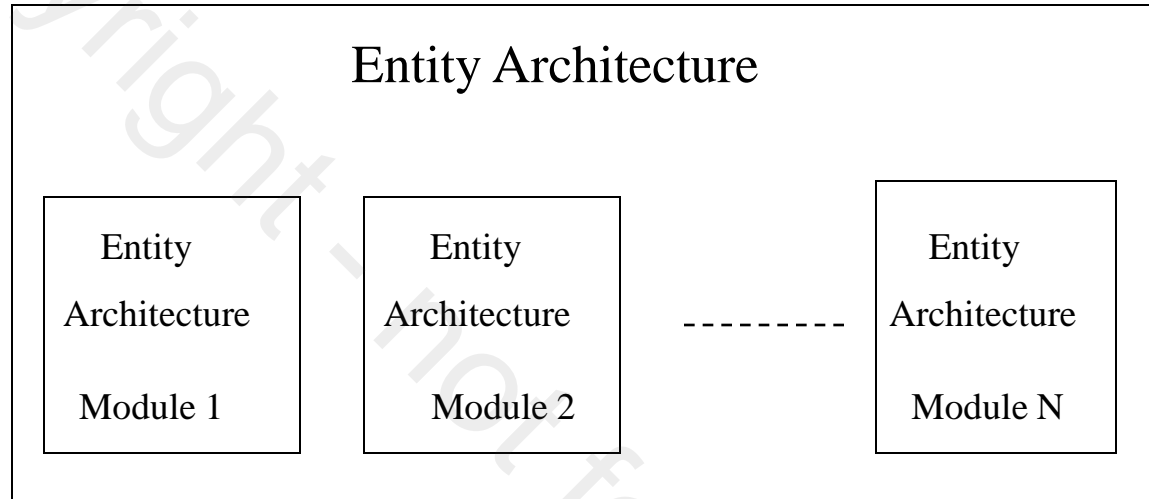
VHDL Program Structure

```
                    Entity Architecture

    ┌─────────────┐   ┌─────────────┐             ┌─────────────┐
    │   Entity    │   │   Entity    │             │   Entity    │
    │             │   │             │             │             │
    │ Architecture│   │ Architecture│  ─────────  │ Architecture│
    │             │   │             │             │             │
    │  Module 1   │   │  Module 2   │             │  Module N   │
    └─────────────┘   └─────────────┘             └─────────────┘
```

entity entity-name **is**

  [port (interface-signal-declaration);]

end [entity] entity-name ;

architecture architecture-name **of** entity-name **is**

[declarations]

begin

  architecture body – concurrent statements

end [architecture] architecture-name;

Concurrent Statements

- The *concurrent statements* in an architecture body describe the module's operation.

- Concurrent statements are so called because conceptually they can be **activated and perform their actions together**, that is, concurrently.

- Contrast this with the sequential statements inside a process, which are executed one after another.

- One form of concurrent statement is a process statement.

- Concurrency is useful for modeling the way real circuits behave.

Behavioral Model

- Behavioral Architecture body includes
  - **Process** statements: collections of actions to be executed in sequence
    - These actions are called *sequential statements*
    (similar to conventional programming languages)
      - Evaluating expressions
      - Assigning values to variables
      - Conditional execution
      - Signal assignment

4-bit Register Example

```vhdl
architecture behav of reg4 is
begin
  storage : process is
  variable stored_d0, stored_d1, stored_d2, stored_d3 : bit;
  begin
      wait until clk = '1';
      if en = '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 storage;
end architecture behav;
```

Structural Model

- Interconnected subsystems

- Hierarchy based design
  - Complex blocks are composed of simpler blocks

- Entity declarations and architecture bodies for subsystems / modules are required.
  - Signal declaration for internal signals of the architecture before `begin` keyword
  - Component instances representing the subsystems
  - Port map specifies the connection of the ports of each component instance to signals within the architecture body

- Do not mix structural and behavioral code within a single module

4-bit Register Example

```vhdl
architecture struct of reg4 is
   component declaration
   signal int_clk : bit;

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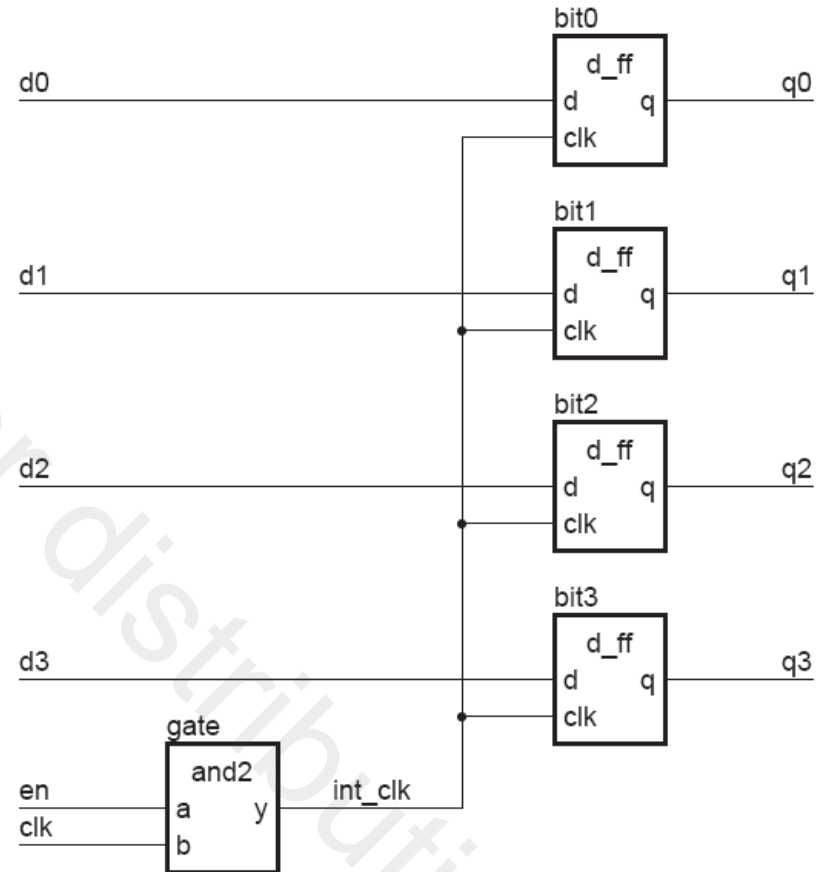
VHDL Processes

- A process is the basic unit of behavioral description

- A process is executed in response to changes of values of signals and uses the present values of signals it reads to determine new values for other signals.

- General for of processes:

```
process (sensitivity list)
  begin
       sequential statements
end process;
```

- Whenever one of the signals in the sensitivity list changes, the sequential statements are executed in sequence

Two Input MUX Process

```
mux : process (a, b, sel) is
begin
  case sel is
        when '0' =>
                z <= a after prop_delay;
        when '1' =>
                z <= b after prop_delay;
        end case;
end process mux;
```

- We can define expressions with optional delay time

if then else Statements

- Sequential statements for control structures in processes:
  - When the behavior of the model depends on a set of conditions that may or may not be true, if statements are used:

```
if sel = 0 then
  result <= input_0; -- executed if sel = 0
else
  result <= input_1; -- executed if sel /= 0
end if;
```

Case Statements

- If we have a model in which the behavior is to depend on the value of a single expression, we can use a *case statement*.

```
type opcodes is(nop, add, subtract, load, store,
                jump, jumpsub, branch, halt);

case opcode is

  when load | add | subtract =>

      operand := memory_operand;

  when store | jump | jumpsub | branch =>

      operand := address_operand;

  when others =>

      operand := 0;

end case;
```

VHDL Like a Programming Language

- Comments:

… a line of VHDL description … -- a descriptive comment

- Identifiers: Case of letter is not important
  - may only contain alphabetic letters ('A' to 'Z' and 'a' to 'z'), decimal digits ('0'to '9') and the underline character;
  - must start with an alphabetic letter
  - may not end with an underline character
  - may not include two successive underline characters

- Bit Strings:
  - `B"0100011" B"10" b"1111_0010_0001"`

- Time: a predefined type called time that is used to represent simulation times and delays:

$$5 \text{ ns} \quad 22 \text{ us}$$

- Reserved Words
  - Some identifiers are reserved for special use and cannot be used:

  `process, library, xor, return, type, else, if`

Constants

- Both constants and variables need to be declared before they can be used in a model.
  - A *declaration* simply introduces the name of the object, defines its type and may give it an initial value.

```
constant_declaration ⇐

constant identifier { , … } : subtype_indication :=
expression ;
```

- Example:

```
constant number_of_bytes : integer := 4;

constant number_of_bits : integer := 8 *number_of_bytes;
```

Enumeration Types

- Often when writing models of hardware at an abstract level, it is useful to use a set of names for the encoded values of some signals

```
type_declaration ⇐ type identifier is type_definition ;
```

- Example:

```
type alu_function is (disable, pass, add, subtract,
                             multiply, divide);

variable alu_op : alu_function;

alu_op := subtract;
```

Arrays and Memory

- **Example:**

```
type array1 is array (1 to 100) of integer;
type ramtype is array (63 downto 0) of
           Std_Logic_Vector(15 downto 0);
variable a1 : array1;
signal mem:   ramtype;


a1(11 to 20):       array of 10 elements
mem: 64 word x 16-bit memory
```

- Memory can be modeled with arrays

Packages and Libraries

- Provides a convenient way of referencing frequently used functions and components

- Package declaration provides external view:

```
package package name is
    package declarations
end package[package-name];
```

- Package body provides the implementation

```
Package body package name is
    package body declarations
end package body[package-name];
```

Packages and Libraries

- **Use** clause to refer directly to items in the package within the library

```vhdl
library ieee; use ieee.std_logic_1164.all;
entity logic_block is
  port ( a, b : in std_ulogic;
         y, z : out std_ulogic );
end entity logic_block;
```

- VHDL has built in **BIT and BIT_Vector** types that define only 0 and 1; however this is not enough for modeling logic with tristates and uninitialized nodes

IEEE 1164 Standard Logic

- 9-valued logic system:

```
type std_logic is ('U', 'X', '0', '1', 'Z', 'W',
        'L', 'H', '-');
```

```
U: uninitialized

X: forcing unknown

0: forcing 0

1: forcing 1

Z: high impedance

W: weak unknown

L: weak 0

H: weak 1

-: Don't care
```

- Array type:

```
type std_logic_vector is array ( natural range <>) of
  std_logic;
```

31

Other Libraries

- Unfortunately **std_logic_vector** does not define basic operations such as addition, comparison, shift

- Extension from Synopsys:
  - `std_logic_unsigned`
  - `std_logic_signed`

Parameterized Blocks-Generate

```vhdl
entity Adder4 is
   port (A,B: in bit_vector(3 downto 0);Ci: in bit;
            S: out bit_vector(3 downto 0);Co: out bit;
end Adder4;


architecture Structure of Adder4 is
component Fulladder
   port(X,Y,Cin: in bit;
          Cout,Sum: out bit);
end component;
signal C: bit_vector(4 downto 0);


begin
   C(0)<=Ci;
   FullAdd4: for i in 0 to 3 generate
   begin
        FAx: FullAdder port map(A(i),B(i),C(i),C(i+1),S(i));
   end generate FullAdd4;
   Co <= C(4);
end Structure;
```

VHDL vs Verilog

- Hardware structure can be modeled equally effectively in both VHDL and Verilog

- The choice of which to use is not therefore based solely on technical capability but on:
  - personal preferences
  - EDA (Electronic Design Automation) tool availability
  - commercial, business and marketing issues

- Verilog could be easier to learn; reads more like C code

- VHDL is strongly typed; makes it robust and powerful for the advanced user

VHDL vs Verilog

- There are more constructs and features for high-level modeling in VHDL than there are in Verilog
  - Packages and libraries for model reuse
  - Generate statements for replicating structure
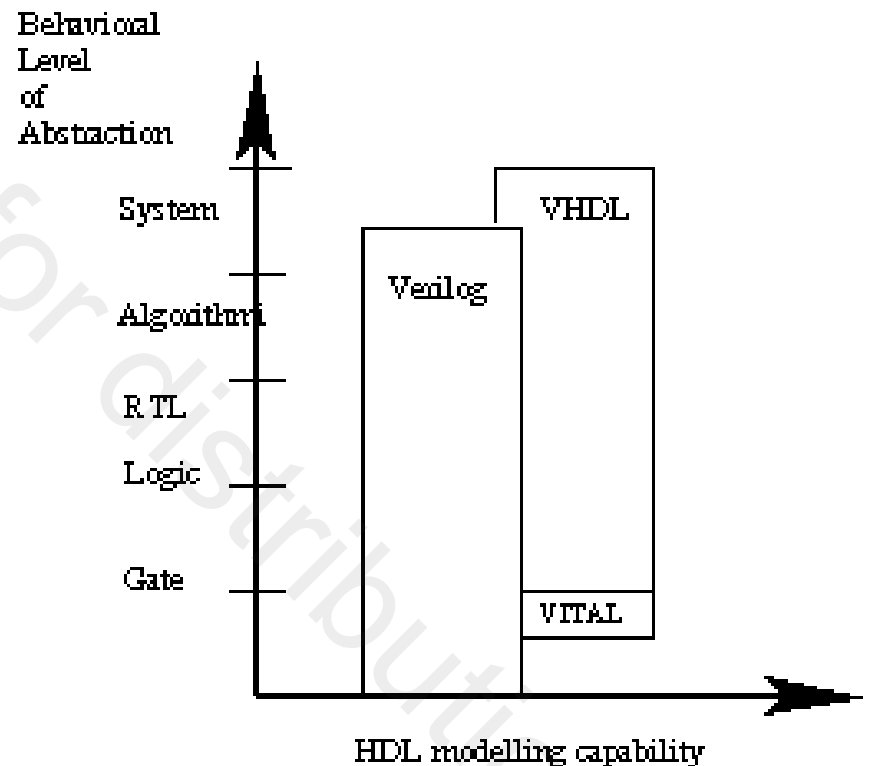
Summary

- Concurrent signal assignments
  - Reevaluated every time any term on the right side changes
  - Multiple assignments can happen in parallel
  - Implies combinational logic

- Processes
  - Re-evaluated only when signals in the sensitivity list change
  - Can imply combinational or sequential logic
  - If all the inputs are in the sensitivity list – combinational logic description