BENHA UNIVERSITY
FACULTY OF ENGINEERING AT SHOUBRA

Post-Graduate
ECE-606
CAD of Electronics

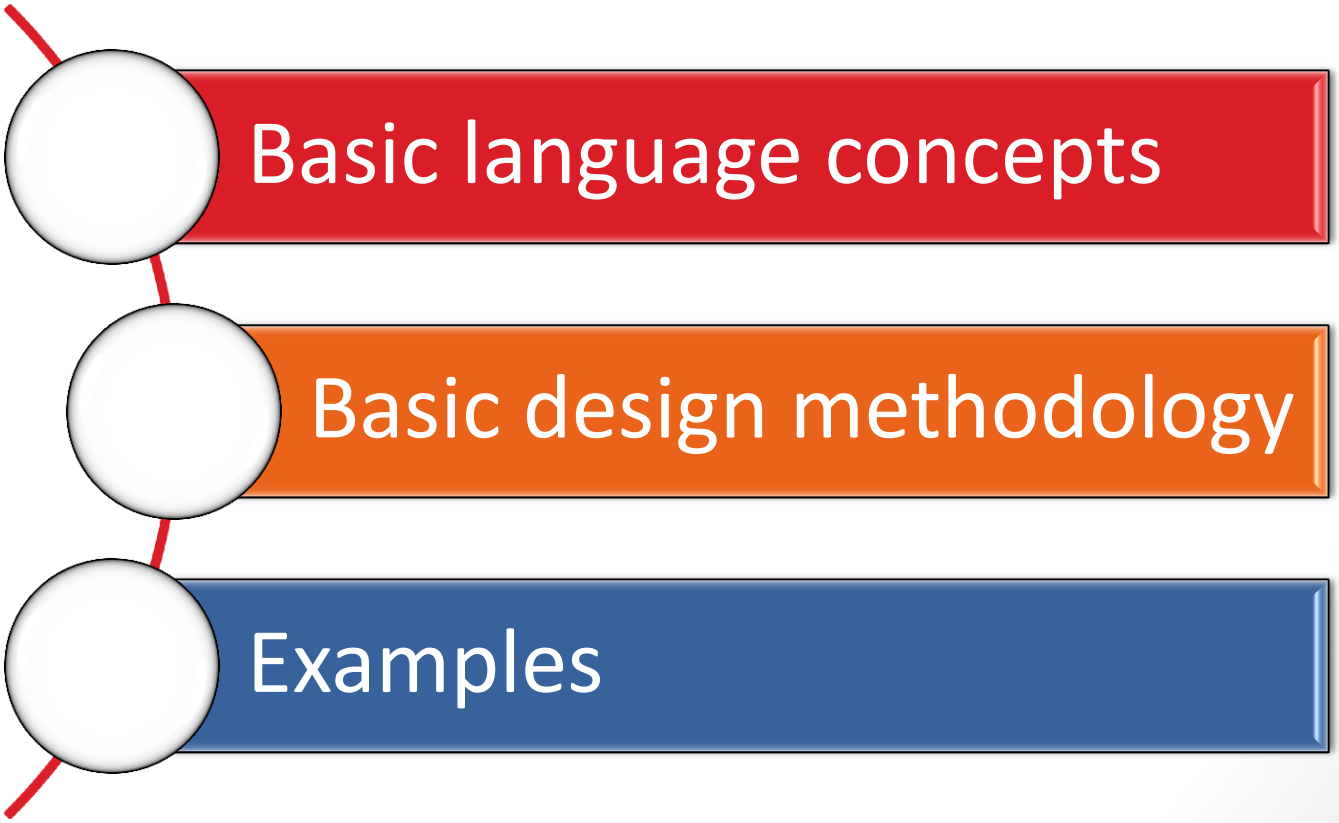
Lecture #2
VHDL Basics

Instructor:
Dr. Ahmad El-Banna

DECEMBER 2014

Agenda

Quick introduction to VHDL



VHDL

- Hardware description languages (HDL)
  - Language to describe hardware
  - Two popular languages
    - VHDL: **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage
      - Developed by DOD from 1983
      - IEEE Standard 1076-1987/1993/200x
      - Based on the ADA language
    - Verilog
      - IEEE Standard 1364-1995/2001/2005
      - Based on the C language
- Applications of HDL:
  - Model and document digital systems
    - Different levels of abstraction
      - Behavioral, structural, etc.
  - Verify design
  - Synthesize circuits
    - Convert from higher abstraction levels to lower abstraction levels

Modeling Digital Systems

- VHDL is for coding models of a digital system...
- Reasons for modeling
  - requirements specification
  - documentation
  - testing using simulation
  - formal verification
  - synthesis
  - class assignments
- Goal
  - most ‘reliable’ design process, with minimum cost and time
  - avoid design errors!

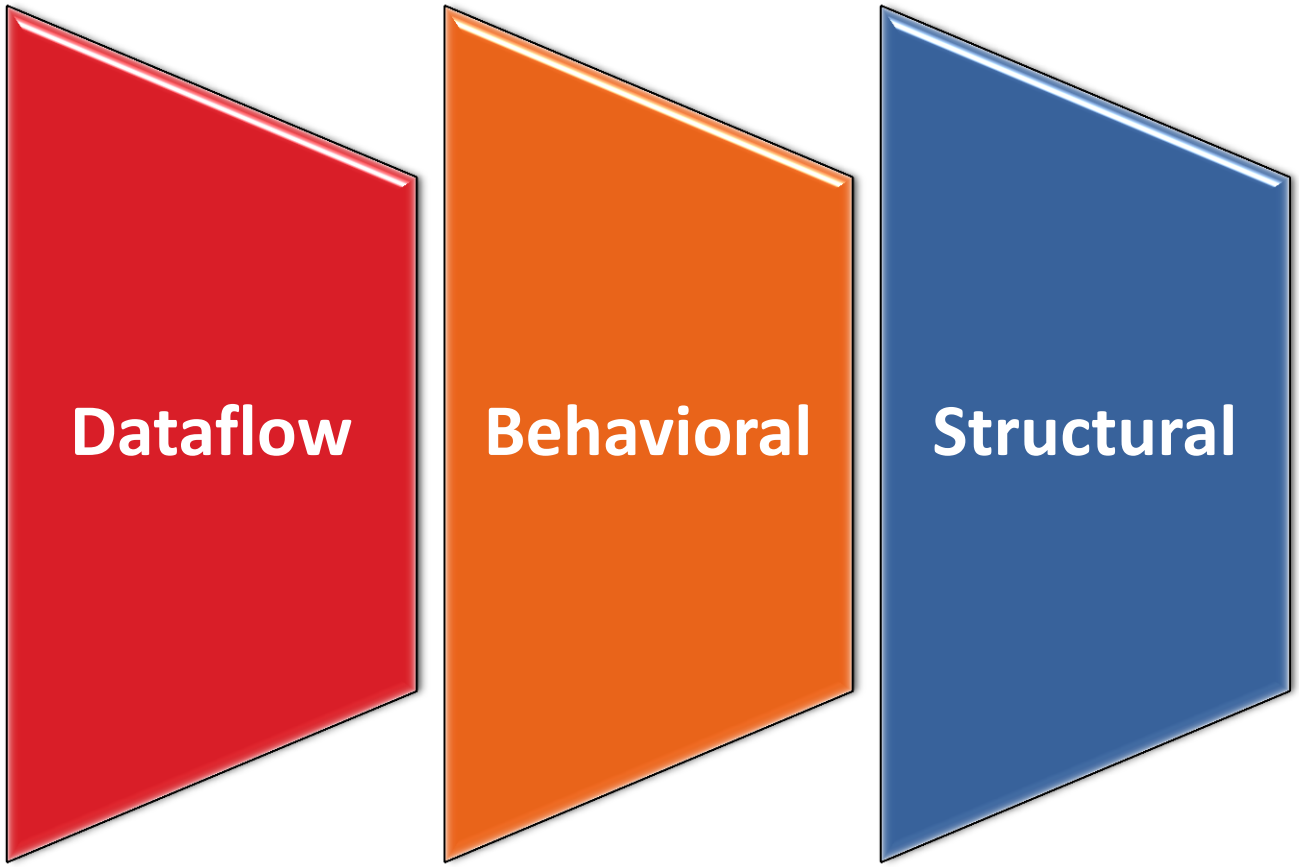
Basic VHDL Concepts

- Main Terms
  - Interfaces -- i.e. ports
  - Behavior
  - Structure
  - Test Benches
  - Analysis, simulation
  - Synthesis

- VHDL is a programming language that allows one to model and develop complex digital systems in a dynamic environment.
- Object Oriented methodology -- modules can be used and reused.
- Allows you to designate in/out ports (bits) and specify behavior or response of the system.

- But VHDL is NOT C ...
  There are some similarities, as with any programming language, but syntax and logic are quite different; so get over it !!

3 ways to DO IT -- the VHDL way



Modeling the Dataflow way

- uses statements that defines the actual flow of data.....

  such as,

  x <= y -- this is NOT less than equal to

  -- told you its not C

  this assigns the boolean signal x to the value of boolean signal y... i.e.
  x = y

  this will occur whenever y changes....

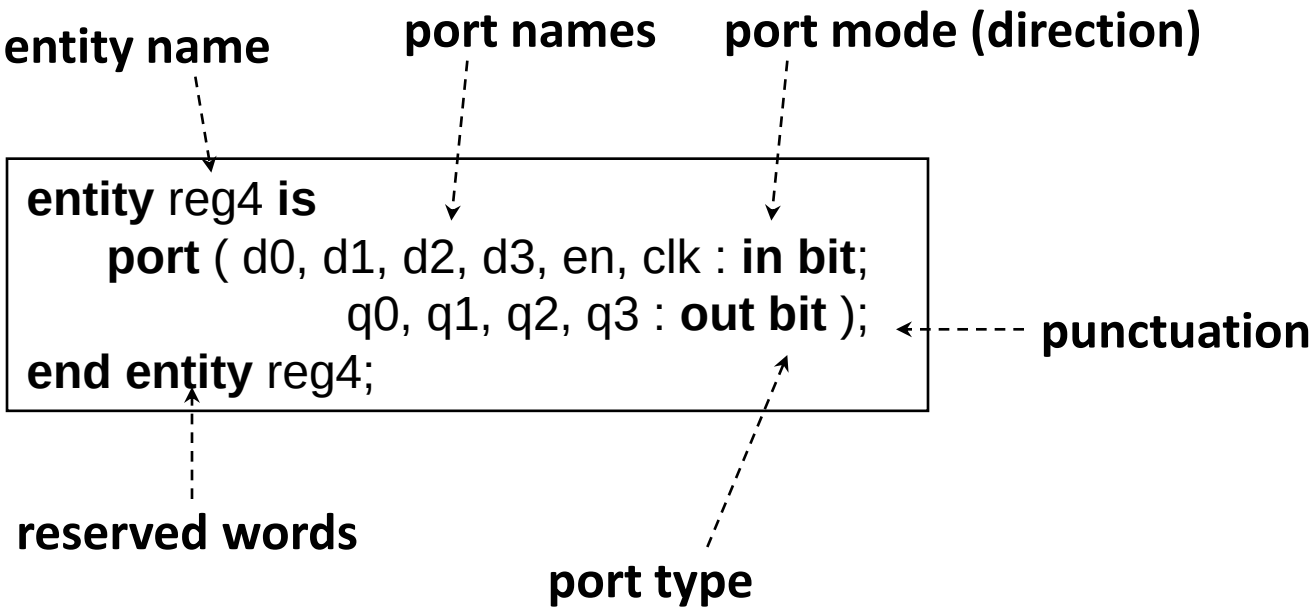
Jumping right in to a Model

- lets look at a d - flip-flop model -- doing it the dataflow way..... ignore the extra junk for now --

```
entity dff_flow is
    port (          d       :in     bit;
                    rn      :in     bit;
                    clrn    :in     bit;
                    q       :out    bit;
                    qbar    :out    bit
                );
end dff_flow;
architecture  arch1 of dff_flow  is
begin
    q  <=  not prn     Or   (clrn  And  d);     % this is  the DATAFLOW %
    qbar <= prn And  (not clrn  Or not d);     %  STUFF    %
end arch1;
```

Modeling Interfaces

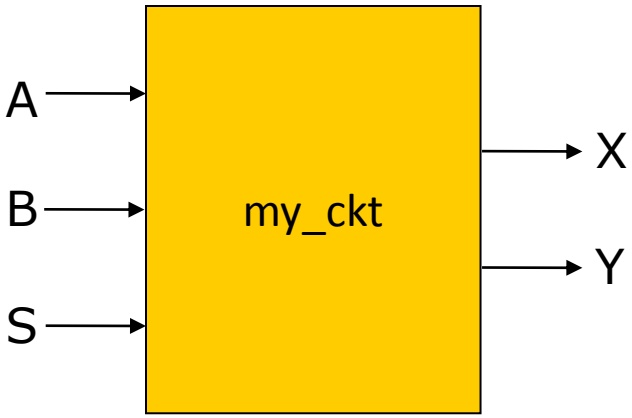
- *Entity* declaration
  - describes the input/output *ports* of a module



Modeling the Behavior way

- *Architecture body*
  - describes an implementation 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*

Example



- Example: my_ckt
  - Inputs: A, B, C
  - Outputs: X, Y
- VHDL description:

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

- Functional Spec.
  - Behavior for output X:
    - When S = 0
      X <= A
    - When S = 1
      X <= B
  - Behavior for output Y:
    - When X = 0 and S =0
      Y <= '1'
    - Else
      Y <= '0'

VHDL Architecture

- VHDL description (sequential behavior):

```
architecture arch_name of my_ckt is
begin
 p1: process (A,B,S)
 begin
  if (S='0') then
   X <= A;
  else
   X <= B;
  end if;

  if ((X = '0') and (S = '0')) then
   Y <= '1';
  else
   Y <= '0';
  end if;

 end process p1;
end;
```

Error: Signals defined as output ports can only be driven and not read

VHDL Architecture..

```
architecture behav_seq of my_ckt is

  signal Xtmp: bit;

  begin
   p1: process (A,B,S,Xtmp)
      begin
    if (S='0') then
     Xtmp <= A;
    else
     Xtmp <= B;
    end if;

    if ((Xtmp = '0') and (S = '0')) then
     Y <= '1';
    else
     Y <= '0';
    end if;

    X <= Xtmp;
   end process p1;
end;
```

Signals can only be defined in this place before the `begin` keyword

General rule: Include all signals in the sensitivity list of the process which either appear in relational comparisons or on the right side of the assignment operator inside the `process` construct.

In our example:
`Xtmp` and `S` occur in relational comparisons
`A`, `B` and `Xtmp` occur on the right side of the assignment operators

VHDL Architecture…

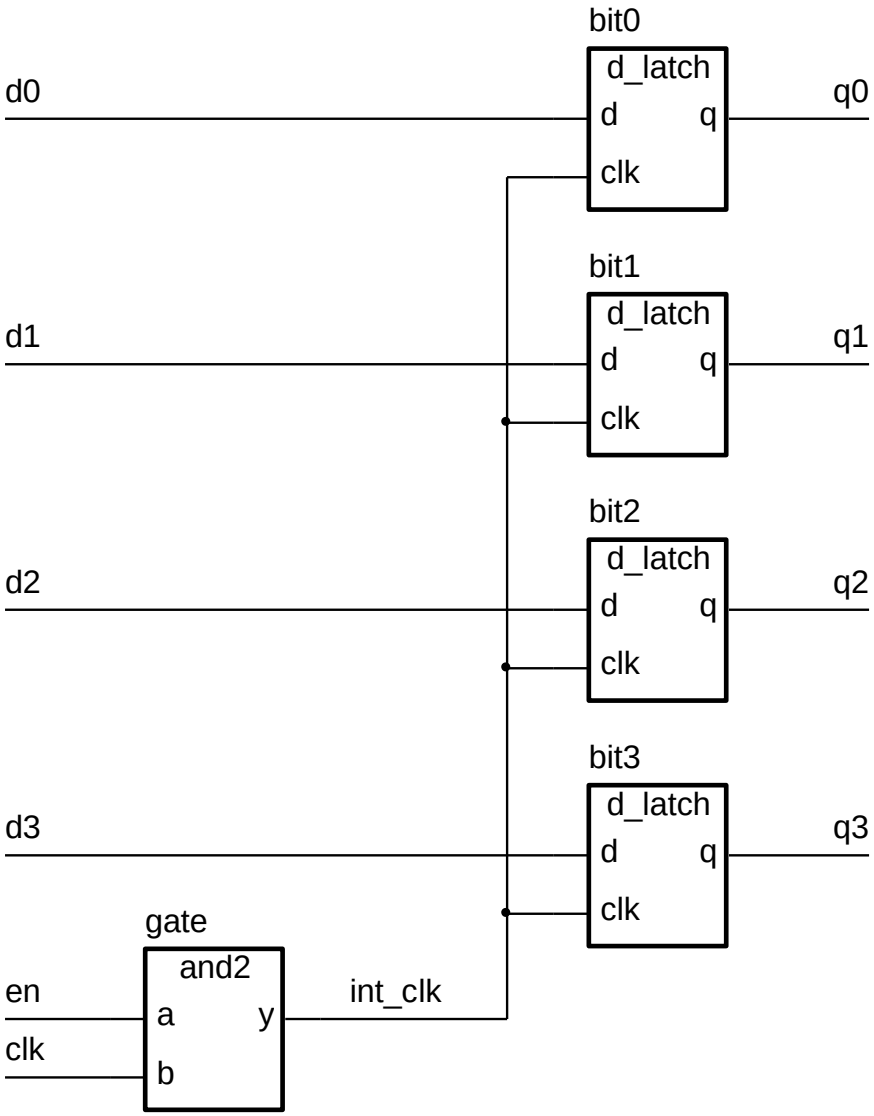
- VHDL description (concurrent behavior):

```
architecture behav_conc of my_ckt is

  signal Xtmp: bit;

begin

  Xtmp <= A when (S='0') else
          B;
  Y <= '1' when ((Xtmp = '0') and (S = '0')) else
       '0';

  X <= Xtmp;

end ;
```

Modeling the Structurural way

- *Structural* architecture
  - 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

Structural way Example



Structural way..

- 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 way...

- 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 way....

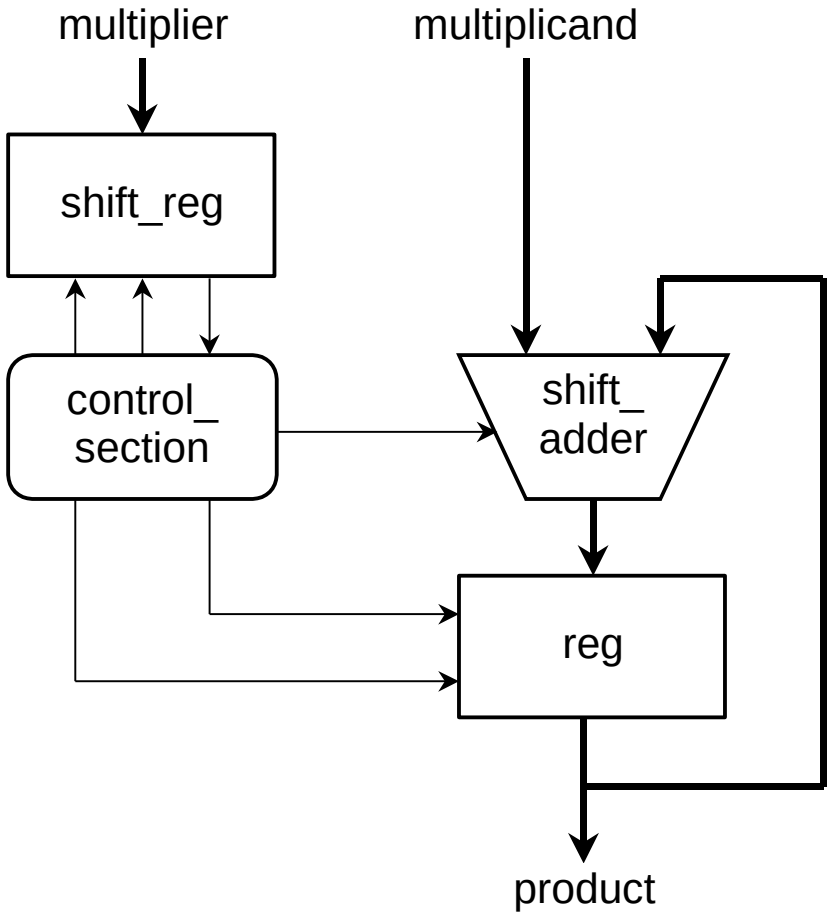
- Now use them to implement the register

```
...
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;
```

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 to signals
- Example: register-transfer-level (RTL) Model
  - data path described structurally
  - control section described behaviorally

Mixed Example



Assignment: Install the FPGA Advantage ® tool and model this example on it !

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 );
    ...
```

Mixed Example..

```
    …
    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 Bench your Model

- Testing a design by simulation
- Use a *test bench* model
  - a Model that uses your Model
  - apply test sequences to your inputs
  - monitors values on output signals
    - either using simulator
    - or with a process that verifies correct operation
    - or logic analyzer

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*
  - make sure your Model is truly OOP

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
  - resumes and schedules new values on output signals
    - schedules *transactions*
    - *event* on a signal if value changes

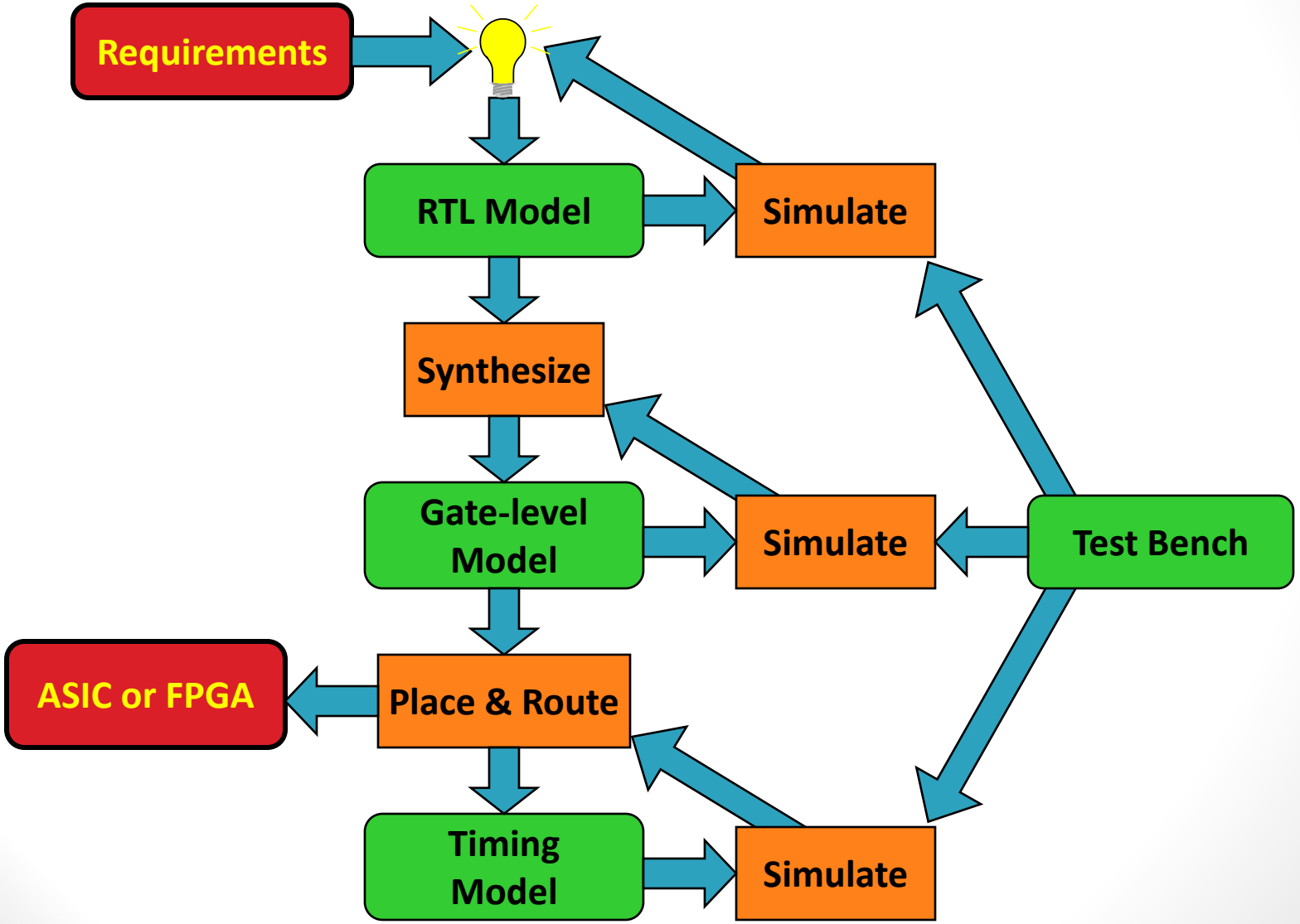
Simulation Algorithm

- Initialization phase
  - each signal is given its initial value
  - simulation time set to 0
  - for each process
    - activate
    - execute until a wait statement, then suspend
      - execution usually involves scheduling transactions on signals for later times

Simulation Algorithm..

- Simulation cycle
  - advance simulation time to time of next transaction
  - for each transaction at this time
    - update signal value
      - event if new value is different from old value
  - for each process sensitive to any of these events, or whose “wait for …” time-out has expired
    - resume
    - execute until a wait statement, then suspend
- Simulation finishes when there are no further scheduled transactions

Basic Design Methodology



- For more details, refer to:
  - VHDL Tutorial: Learn by Example *by Weijun Zhang*
  - http://esd.cs.ucr.edu/labs/tutorial/
  - "**Introduction to VHDL**" presentation by Dr. Adnan Shaout, *The University of Michigan-Dearborn*
  - **The VHDL Cookbook**, *Peter J. Ashenden, 1st edition, 1990.*
- The lecture is available online at:
  - *http://bu.edu.eg/staff/ahmad.elbanna-courses*
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg