Module 2 – Part 4

VHDL To Hardware
VHDL Introduction, Syntax

Objectives

- This module is not a course on how to program VHDL
- This VHDL overview is provided to show the capability of design simulation and fabrication
- Many of the constructs that we have studied during this course can be implemented as VHDL, and then used as a component in a design
- FPGA vendors supply excellent Intellectual Property based designed that provide both free and licensed capability.
- Many good references and classes that expand on VHDL
  - 525.442 FPGA Design Using VHDL
  - VHDL For Logic Synthesis 3rd Edition – by Andrew Rushton (Wiley 2011)
  - Some references are included in the Discussion Forum

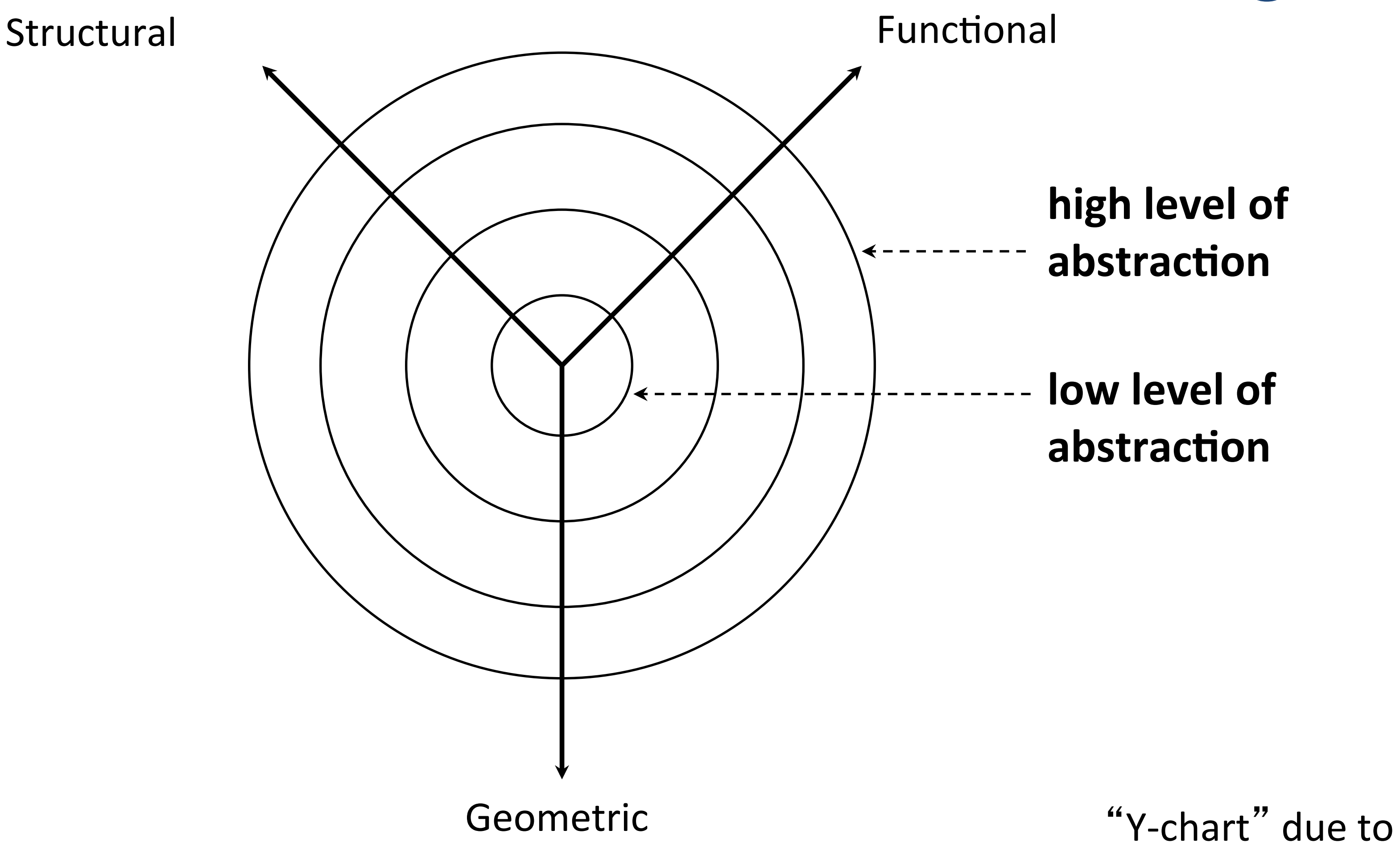
Topics

- Role of VHDL in Computer Architecture
- VHDL Basics

Role of VHDL in Computer Architecture

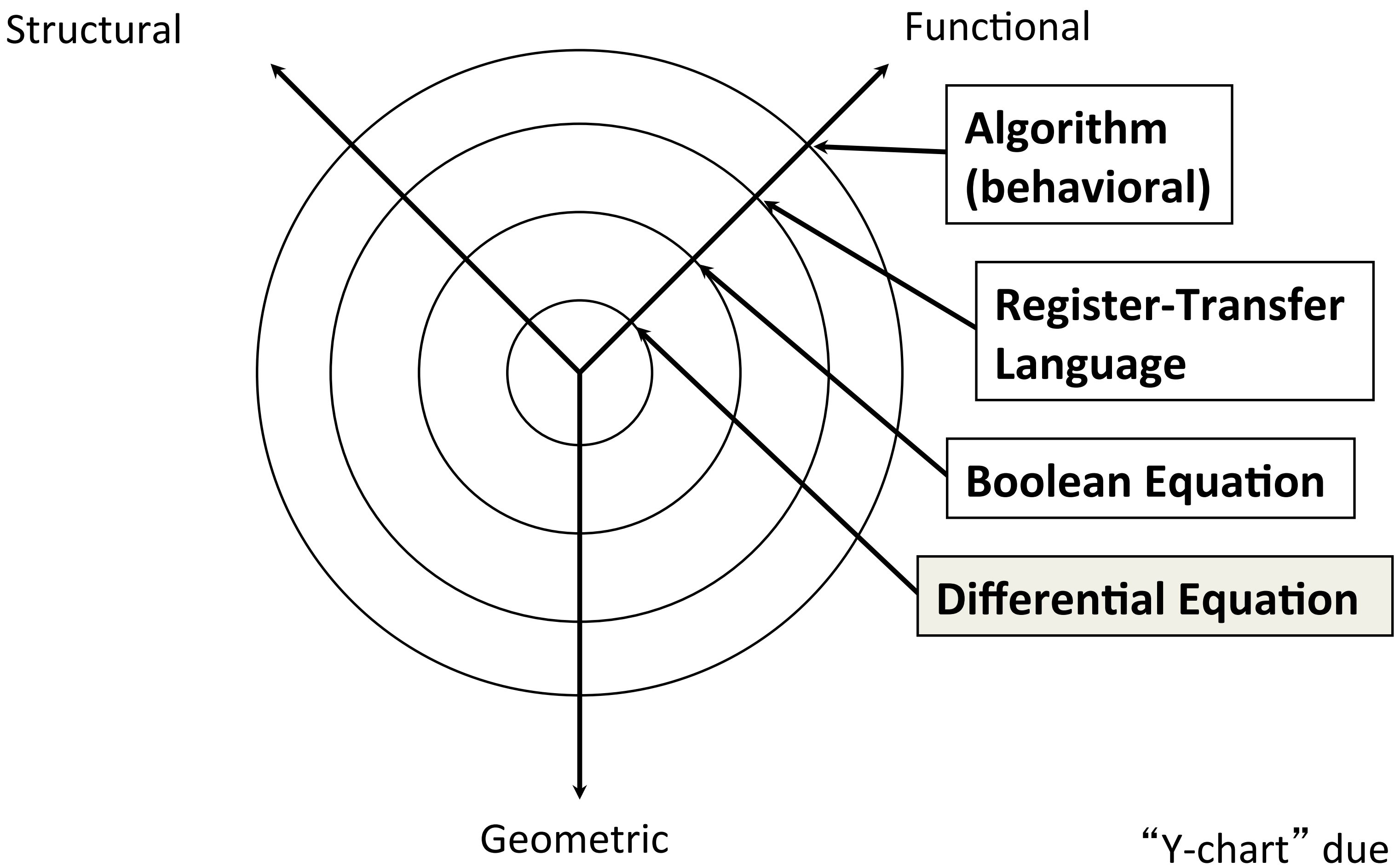
- VHDL can be used to write models of a system
- Goals of the models:
  - Requirements Specification
  - Documentation
  - Testing using Simulation
  - Formal Verification
  - Synthesis of the design
- Net result of the Model:
  - More reliable designs
  - Less errors

Domains and Levels of Modeling



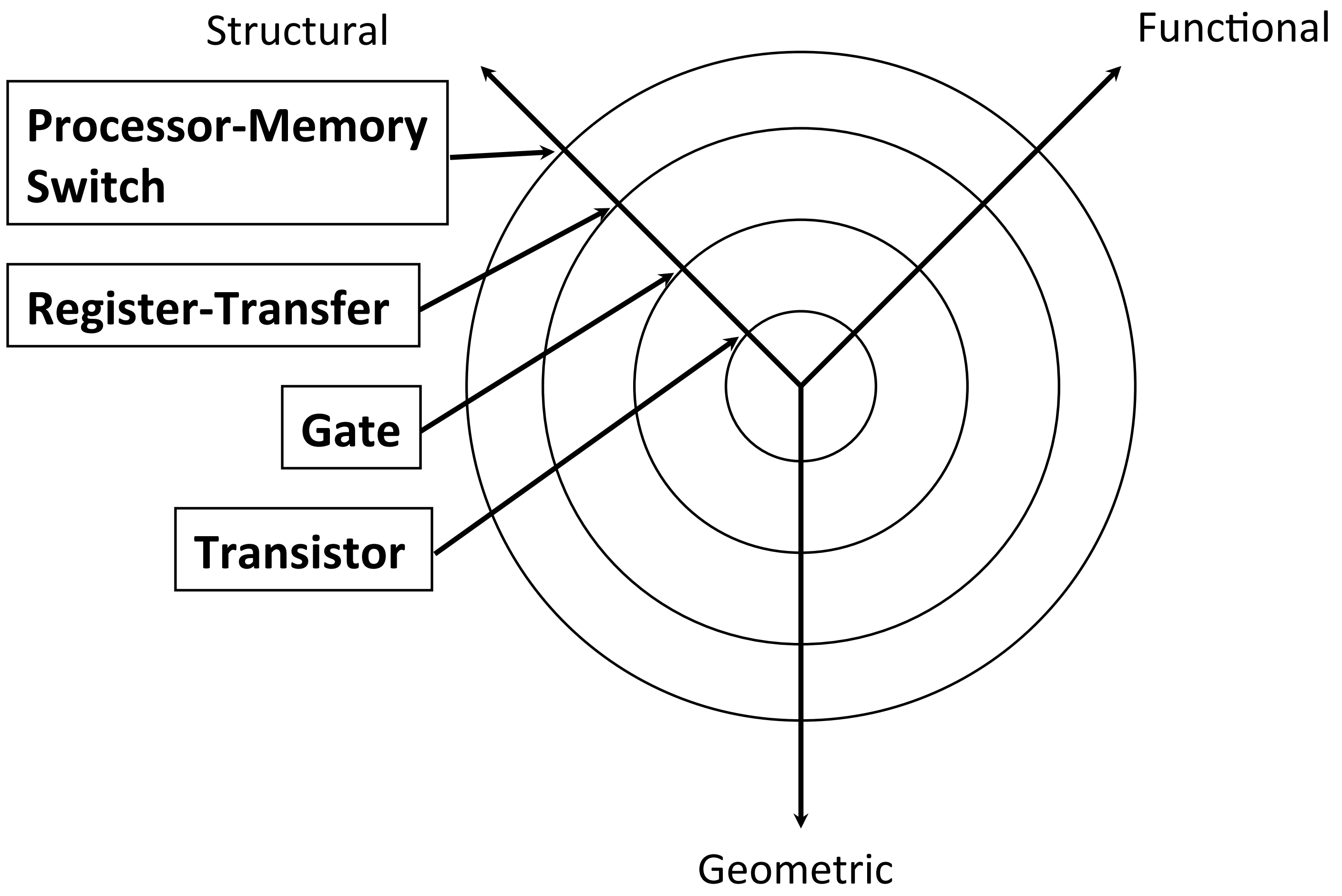
“Y-chart” due to Gajski & Kahn

Domains and Levels of Modeling



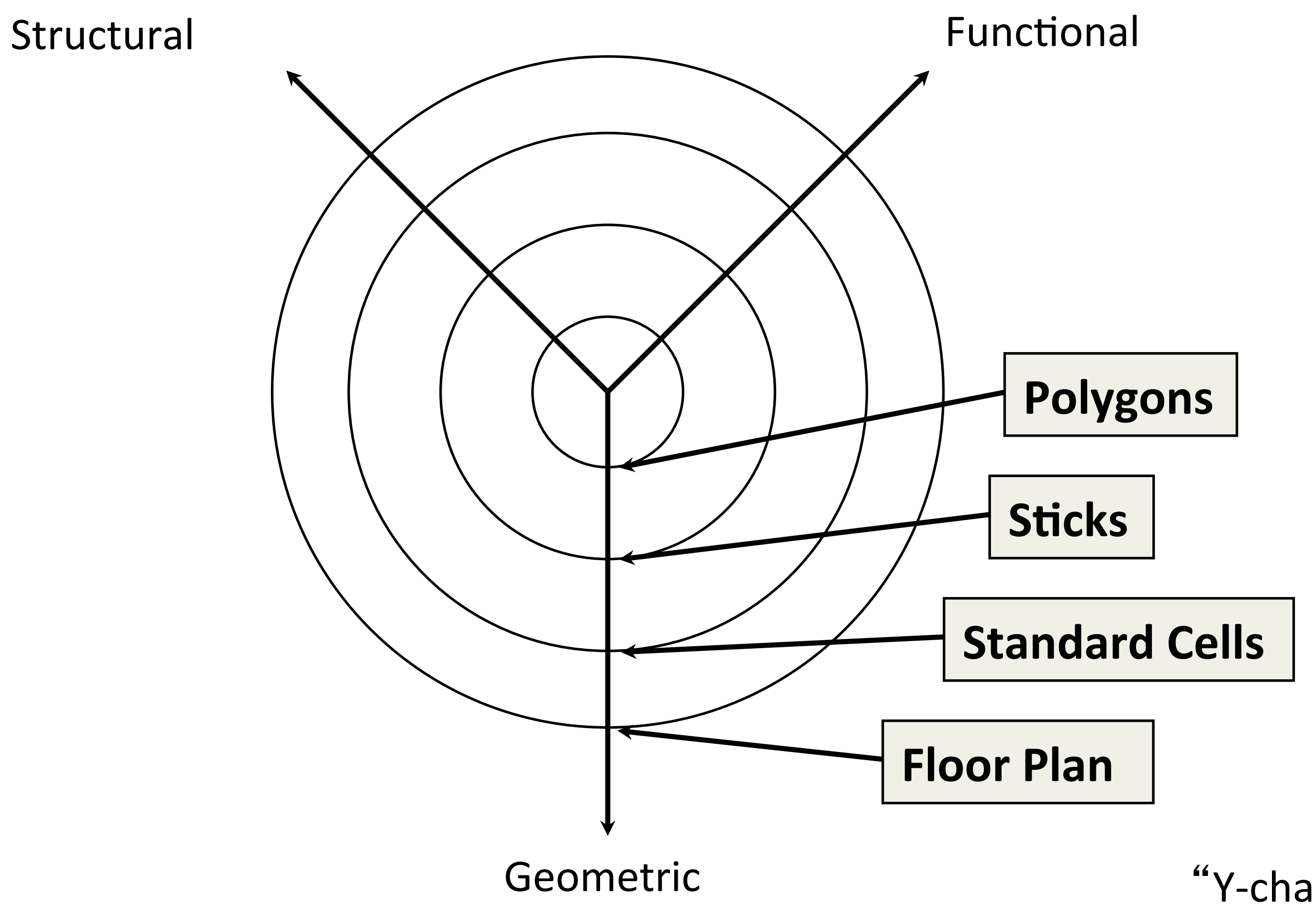
"Y-chart" due to Gajski & Kahn

Domains and Levels of Modeling



"Y-chart" due to Gajski & Kahn

Domains and Levels of Modeling



"Y-chart" due to Gajski & Kahn

VHDL Syntax

- Very much like other programming languages
  - Basic rules:
    - VHDL is not case sensitive
    - VHDL is not sensitive to white space
    - Comments begin with - -
    - Parenthesis should be used to promote clarity
    - VHDL statements end in ;
    - Know the rules for if, case and loop
  - Major Difference:
    - VHDL can describe both parallel and sequential activity.

VHDL Syntax

- More basic rules:
  - Identifiers
    - Variable names, signal names and port names
      - Identifiers should be self commenting
      - Identifiers can be as long as necessary
      - Identifiers can contain some combination of letters (A-Z, a-z), digits (0-9) and the underscore (_)
      - Identifiers start with an alphabetic character
      - Identifiers must not end with an underscore and can't have two consecutive underscores.

VHDL Syntax

VHDL Reserved Words (This is not complete)				
access	exit	mod	return	while
after	file	new	signal	with
alias	for	next	shared	
all	function	null	then	
attribute	generic	of	to	
block	group	on	type	
body	in	open	until	
buffer	is	out	use	
bus	label	range	variable	
constant	loop	rem	wait	

Module 2 – Part 4

VHDL To Hardware

Coding Style, Elements

Coding Style

- Make your code readable
  - You may remember what you are doing today, but tomorrow you mind will be elsewhere
  - You will find out very quickly that compilers have rules, and some of them are documented
    - Many of these rules are similar to other languages, but VHDL is attempting to first describe a physical circuit, and then how that circuit is laid out.
      - A hierarchical design is the goal:
        - Your design may start as a schematic which ties many (created) VHDL, as well as library functions
        - You are also designing for parallel processing

Elements of VHDL

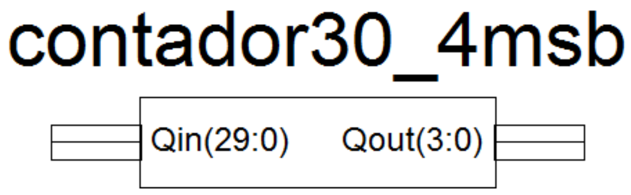
- Interfaces
- Behavior
- Structure
- Test Benches
- Analysis, elaboration, simulation
- Synthesis

Modeling Interfaces

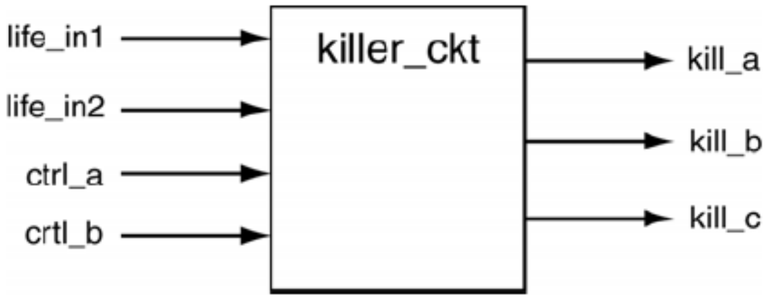
- Entity declaration
  - Describes the input/output ports of a module:

```
entity contador30_4msb is
    Port ( Qin : in  STD_LOGIC_VECTOR (29 downto 0);
           Qout : out  STD_LOGIC_VECTOR (3 downto 0));
end contador30_4msb;
```

  - Contador30_4msb is the entity name
  - Qin and Qout are the port names
  - In/Out are the directions
  - STD_VECTOR defines the port type
    - In this case they are two busses, 30 bit in and 4 bits out.
  - Note: This function takes the 4 MSB digits and outputs them.
- Symbol from Entity:



Example Block Box and VHDL entity declaration



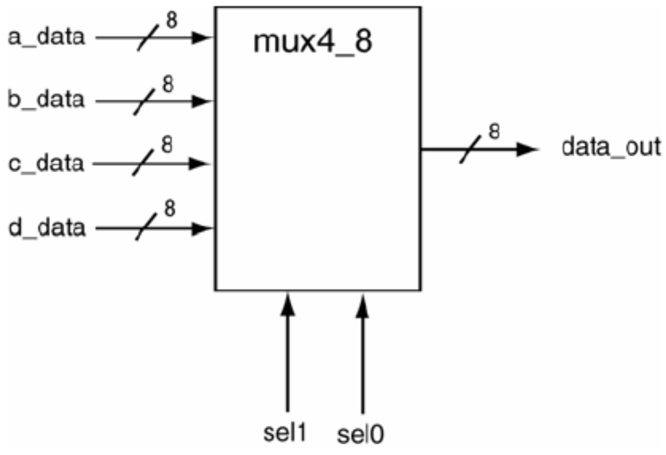
```
-----------------------------------------------------------------
-- Here's my interface description of the killer circuit
-- It does a lot of killer things.
-----------------------------------------------------------------
entity killer_ckt is
   port ( life_in1 : in std_logic;
          life_in2 : in std_logic;
            crtl_a, ctrl_b : in std_logic;
            kill_a : out std_logic;
            kill_b : out std_logic;
            kill_c : out std_logic);
end killer_ckt;
```

Busses

- Busses are easily described in the VHDL entity.
  - Need a new data type and special notation which describes the bus.
  - Mode remains the same, but the data type has changed
    - Std_Logic is now std_logic_vector with the ordering of the bits

```
magic_in_bus      : in std_logic_vector(0 to 3);
big_magic_in_bus  : in std_logic_vector(7 downto 0);
tragic_in_bus     : in std_logic_vector(16 downto 1);
data_bus_in_32    : in std_logic_vector(0 to 31);
mux_out_bus_16    : out std_logic_vector(0 to 15);
addr_out_bus_16   : out std_logic_vector(15 downto 0);
```

A Black Box example with Busses



```
------------------------------------------------------------------------
-- Unlike the other examples, this is actually an interface
-- for a MUX that selects one of four bus line for the
-- output.
------------------------------------------------------------------------
entity mux4_8 is
   port (   a_data : in  std_logic_vector(0 to 7);
            b_data : in  std_logic_vector(0 to 7);
            c_data : in  std_logic_vector(0 to 7);
            d_data : in  std_logic_vector(0 to 7);
        sel1,sel0 : in  std_logic;
          data_out : out std_logic_vector(0 to 7));
end mux4_8;
```

Some Additional Terms

- Generic:
  - A parameter that passes information to an entity
  - Example: For a gate-level model with rise an fall delay, values for the rise and fall delays are passed as generics
- Package:
  - A collection of common declarations, constants and/or subprograms to entities and architectures.
- Attribute:
  - Data attached to VHDL objects or predefined data about VHDL objects
  - Examples:
    - Maximum operation temperature of a device
    - Current drive capability of a buffer

Modeling Behavior

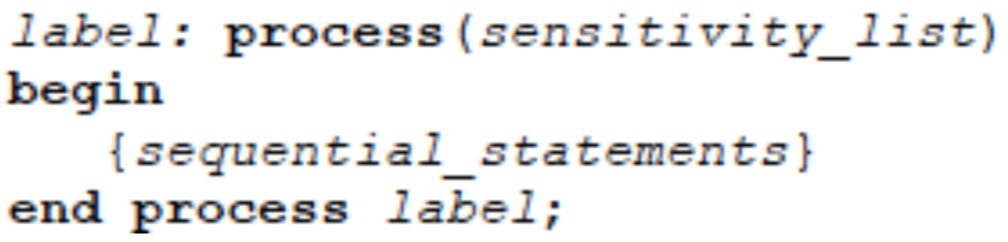
- Architecture body
  - Describes an implementation of an entity
  - May be several per entity
- Behavioral architecture
  - Describes the algorithm performed by the module
  - Contains:
    - Process statements, each containing
      - Sequential statements, including
        - Signal assignment statements
        - Wait statements

Approach to Writing VHDL Architectures

- Data Flow Style:
  - Specifies a circuit as a concurrent representation of the flow of data through the circuit
    - Think combinatoric – In, Out, and the logic processing
- Behavioral Style
  - Model circuit behavior, parallel activity
  - High Level, algorithmic, sequential execution (RTL)
- Structural Style
  - Define the circuit interconnections, and components
  - Low Level, netlists, component instantiations
  - Easy to synthesize
  - Very detailed

Approach to Writing VHDL Architectures

- Behavior Style:
  - Provide no details as to how the design is implemented in actual hardware
  - Models how the circuit outputs will react to or behave to the circuit inputs
  - The synthesizer tool will decide how the circuit is actually fabricated.
    - Process Statement – Everything inside the process statement happens sequentially
      - Concurrent statement – everything happens at once



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

Structural Style VHDL

- Allows the designer to represent a system in terms of components and their interconnections.
- VHDL designs can be generated and then incorporated into the Structural Architecture as components
- Information about interconnection, delays, etc are used to characterize the architecture.

Sequential Statements

- Signal Assignment Statements – Same as the concurrent signal assignment statement
- If Statements

```
if (condition) then
    { sequence of statements }
elsif (condition) then
    { sequence of statements }
else
    { sequence of statements }
end if;
```

- Case Statements

```
case (expression) is
   when choices =>
        {sequential statements}
   when choices =>
        {sequential statements}
   when others =>
        {sequential statements}
end case;
```

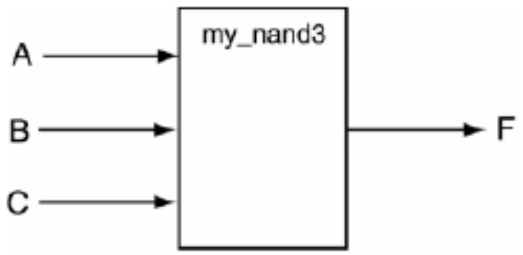
Concurrent and Parallel Statements

- VHDL describes concurrent and parallel activity
  - Four types of concurrent statements:
    - Concurrent signal assignment
    - Process statements
    - Conditional signal assignments
    - Selected signal assignments

Concurrent Signal Assignment Statement

- These statements express concurrency of execution (end in a ;)
- The signal assignment operator (<=) assigns the result
- Target <= expression;

Example VHDL



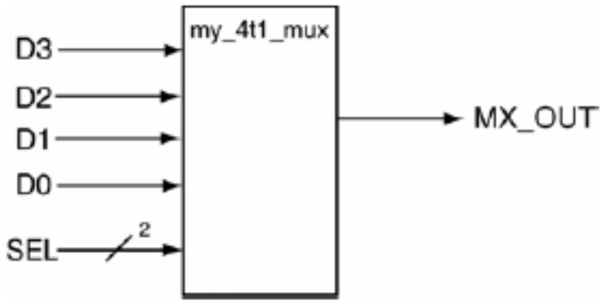
```
-- header files and library stuff
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity my_nand3 is
    port ( A,B,C : in std_logic;
               F : out std_logic);
end my_nand3;

architecture ex_nand3 of my_nand3 is
begin
    F <=  NOT (A AND B AND C);
--
--  An alternative approach for this statement:
--  F <= A NAND B NAND C;
--
end ex_nand3;
```

Conditional Signal Assignment

- Statements that have one target, but can have more than one associated expression
  - Each of the expressions is associated with a certain condition.
  - Like some C compilers, the individual conditions are evaluated sequentially until the first condition evaluates to TRUE.
  - Syntax:

```
target <= expression when condition else
          expression when condition else
          expression;
```

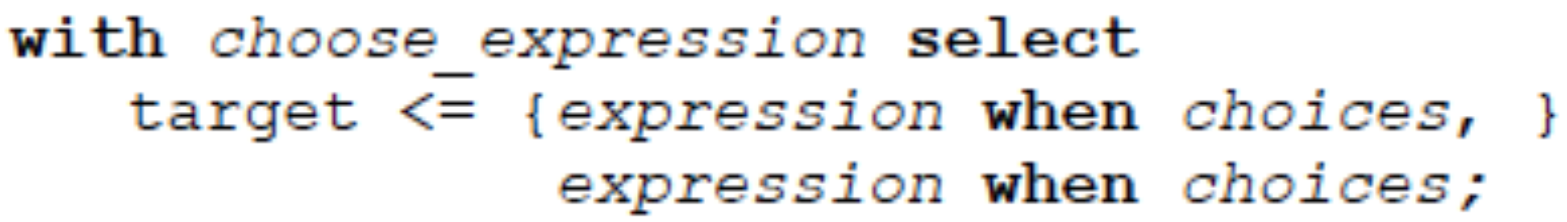
Example Conditional Signal Assignment



```
-----------------------------------------------------------------
-- entity and architecture of 4:1 Multiplexor implemented using
-- conditional signal assignment.
-----------------------------------------------------------------
entity my_4t1_mux is
    port ( D3,D2,D1,D0 : in std_logic;
                   SEL : in std_logic_vector(1 downto 0);
                MX_OUT : out std_logic);
end my_4t1_mux;

architecture mux4t1 of my_4t1_mux is
begin
   MX_OUT <= D3 when (SEL = "11") else
             D2 when (SEL = "10") else
             D1 when (SEL = "01") else
             D0 when (SEL = "00") else
             '0';
end mux4t1;
```

Selected Signal Assignment

- Statement has only one target signal and only one expression determines what the choices are based upon.



```
with choose_expression select
    target <= {expression when choices, }
               expression when choices;
```

Selected Signal Assignment Example

```
------------------------------------------------------------------
-- entity and architecture of 4:1 Multiplexor using selective
-- signal assignment.
------------------------------------------------------------------
entity my_4t1_mux is
    port ( D3,D2,D1,D0 : in std_logic;
                   SEL : in std_logic_vector(1 downto 0);
                MX_OUT : out std_logic);
end my_4t1_mux;

architecture mux4t1_2 of my_4t1_mux is
begin
   with SEL select
   MX_OUT <= D3  when "11",
             D2  when "10",
             D1  when "01",
             D0  when "00",
             '0' when others;
end mux4t1_2;
```

Module 2 – Part 4

VHDL To Hardware

Process and Models

Process Statement

- The process block describes sequential events that will be modeled against time
- Multiple process blocks describe simultaneous events:
- Example – Clock Generation

```
-- generate 50 MHz
clockclk_40MHz_gen : process
   begin
      wait for 2 ns;
      aclk <= not aclk;
end process;
```

This process generates a 50MHz clock that will continue in parallel to other processes.

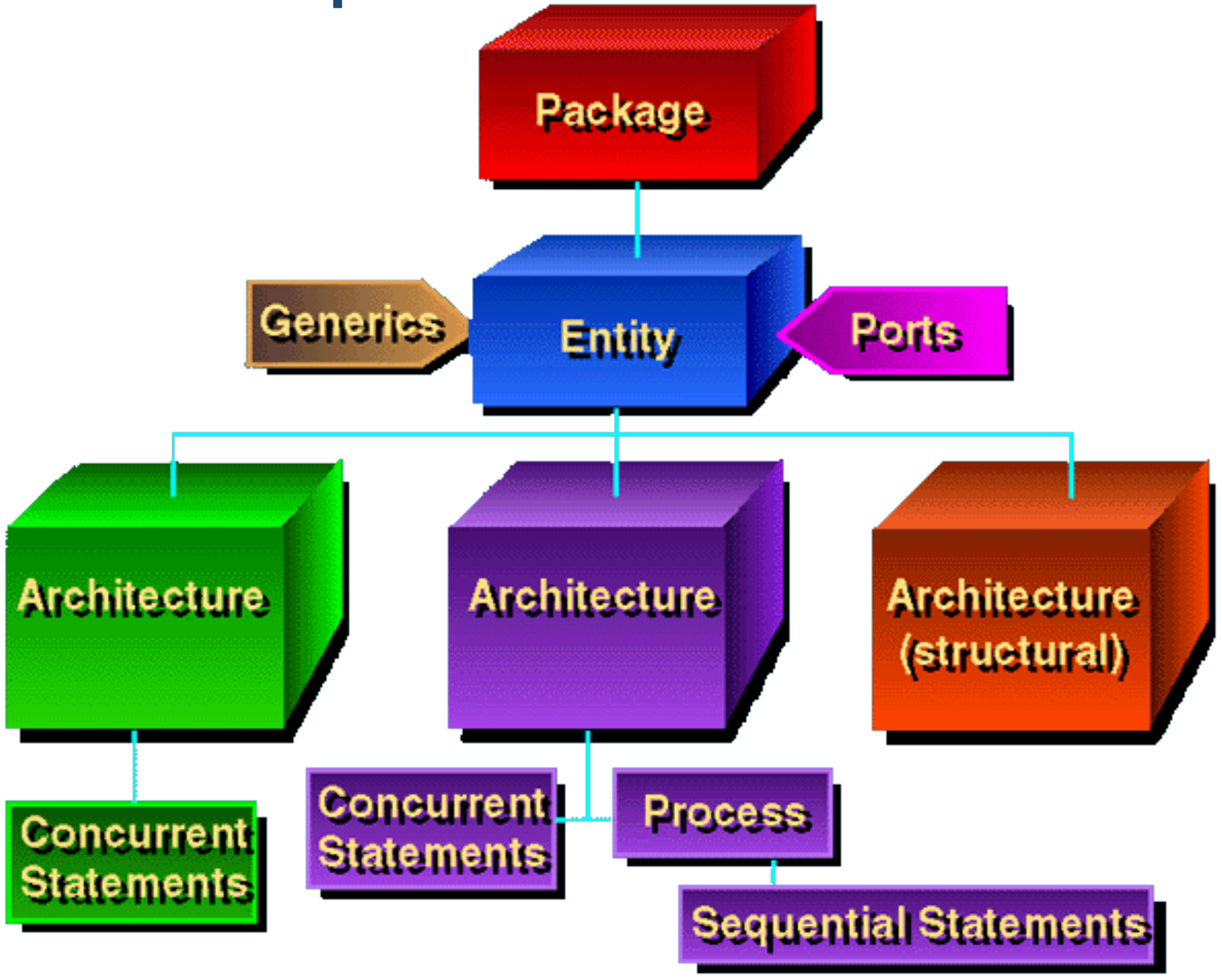
Behavior Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;
entity Booth_tb is
end Booth_tb;
architecture sim of Booth_tb is
signal x : std_logic_vector(3 downto 0);
signal y : std_logic_vector(3 downto 0);
signal mult_out : std_logic_vector(7 downto 0);
Begin
    Booth_inst : entity Booth port map (
    x => x,
    y => y,
    O => mult_out);

    x <= "1011";
    y <= "0111”;
end;
```

Modeling Structure

- *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
    - *wait statements*

Complete VHDL Model



VHDL Test Bench

- Test Benches are VHDL entity/architectures with the following:
    - Instantiate the design to be tested using components
    - Call the instantiations “Unit Under Test” (UUT) or “Device Under Test”
    - The entity has no ports
    - Create a stimulus generator within the architecture
    - Can use all reporting features to monitor the expected outputs
- This is a verification function
    - Devices are not synthesizable
    - Can test components that can be synthesized.

VHDL Analysis

- Check for syntax and semantic errors
  - syntax: grammar of the language
  - semantics: the meaning of the model
- Analyze each *design unit* separately
  - entity declaration
  - architecture body
  - …
  - best if each design unit is in a separate file
- Analyzed design units are placed in a *library*
  - in an implementation dependent internal form
  - current library is called work

VHDL Elaboration

- VHDL is a concise description of circuitry
  - Each instantiation of the components can be described by recursion.
- “Flattening” the design hierarchy
  - create ports
  - create signals and processes within architecture body
  - for each component instance, copy instantiated entity and architecture body
  - repeat recursively
    - bottom out at purely behavioral architecture bodies
- Final result of elaboration
  - flat collection of signal nets and processes

VHDL Simulation

- Execution of the processes in the elaborated model
- Discrete event simulation
  - time advances in discrete steps
  - when signal values change—*events*
- 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 new value different from old value
- The simulation does not imply realizable circuit.

Simulation Approach

- The test bench follows this type of simulation
- 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

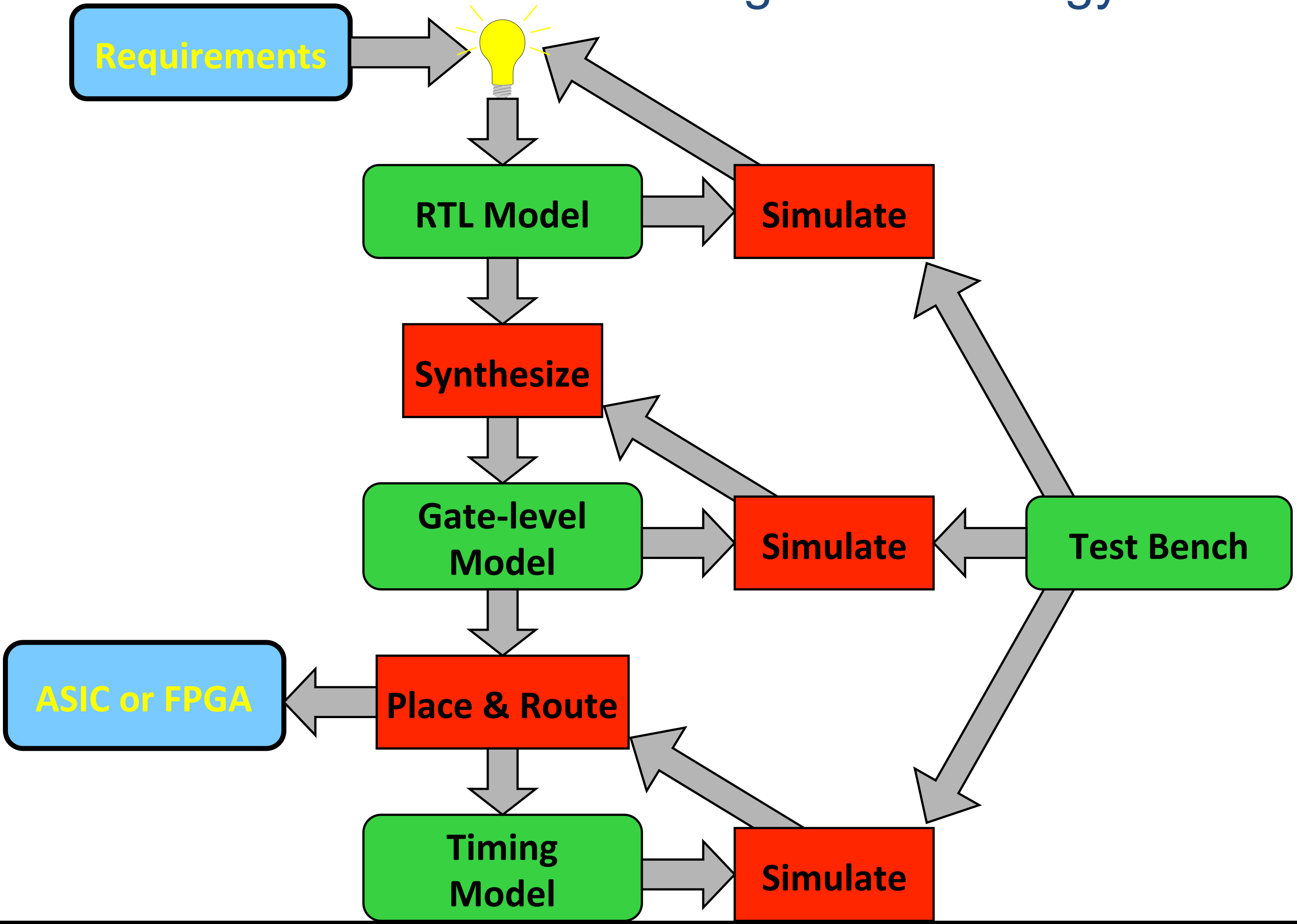
Simulation Approach

- 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

VHDL Synthesis

- Translates register-transfer-level (RTL) design into gate-level netlist
- Restrictions on coding style for RTL model
- Tool dependent
- Target features drive the ability to synthesize design
    - Simple FPGA versus complex FPGA

Basic Design Methodology



Coming Up Next

- Part 5 - Verilog