VHSIC Hardware Description Language

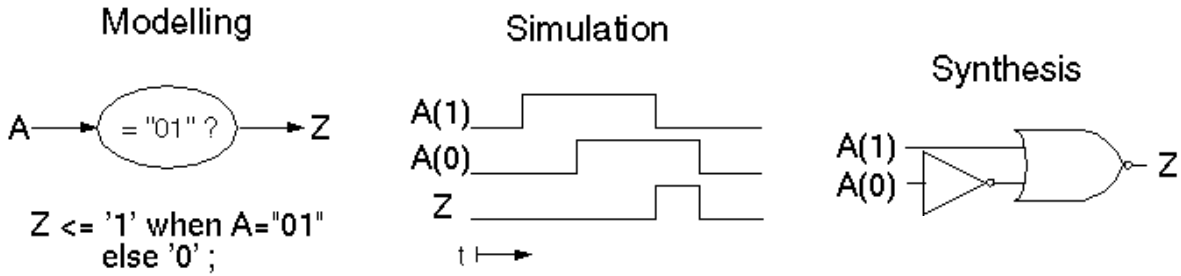
CO 2206 Computer Organization

Topics

- Hardware Description Language
- VHDL History
- VHDL Characteristics
- Abstract Levels in IC Design
- Abstract Levels and VHDL
- VHDL – file structure, syntax
- VHDL Models: Structural, Behavioral
- Test Bench

Hardware Description Language - 1

- ***HDL*** resembles programming language, but are specifically oriented to describing hardware structures and behavior
  - HDL represent extensive parallel operation
    - whereas most programming languages represent serial operation
- Usages (for digital networks) - to describe and simulate complex digital systems:
  - Modelling
  - Simulation
  - Synthesis



Hardware Description Language - 2

- HDL models are reusable
  - re-used in several designs/projects
  - frequently needed function blocks are collected in model libraries
- Reasons for HDL
  - Logic simulation and synthesis are the main reason for the growth in the use of HDL
  - HDLs are portable across CAD tools, whereas schematic capture tools are typically unique to particular vendor
- Few HDL languages
  - VHDL (VHSIC HDL) – we will only learn VHDL
  - Verilog
  - ABEL (Advanced Boolean Equation Language)

VHDL History

VHDL = VHSIC HDL = Very High Speed Integrated Circuit HDL

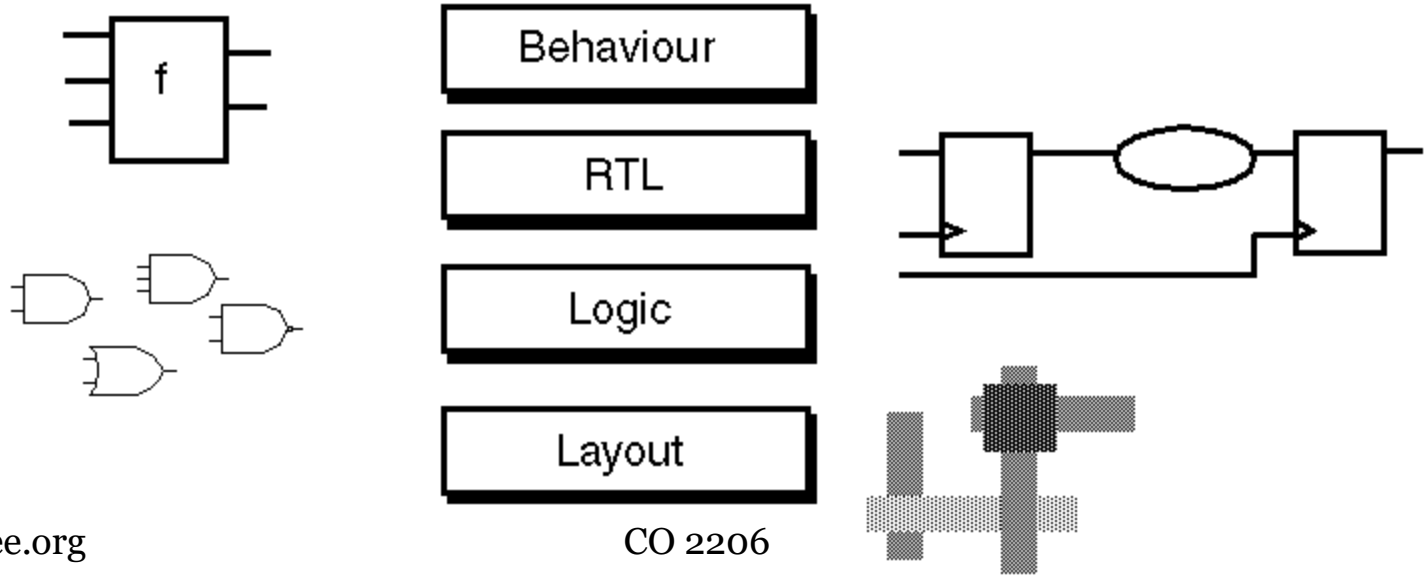
- Early ‘70s: Initial discussions
- 1981: First introduced for the US Department of Defense (DoD) under the VHSIC programe with the goal to develop very high-speed integrated circuit; IEEE sponsored
- 1983: IBM, Texas instruments and Intermetrics started to develop this language
- 1985: VHDL 7.2 version was released
- 1987: IEEE standardized the language as IEEE 1076
- 1993: New standard IEEE 1076-1993

VHDL Characteristics

- Execution of assignments:
  - Sequential statements are executed one after another. The order of the assignment must be considered when sequential statements are used.
  - Concurrent statements are active continuously. So the order of the statements is not relevant. Concurrent statements are especially suited to model the parallelism of hardware.
- Methodologies:
  - Abstraction allows for the description of different parts of a system with different amount of detail.
  - Modularity enables the designer(s) to split big functional blocks and to write a model for each part.
  - Hierarchy lets the designer build a design out of submodules which may consist of several submodules, themselves.

Abstraction Levels in IC Design - 1

- Abstraction is hiding of details:
  - Differentiation between essential and nonessential information
- Creation of abstraction levels:
  - On every abstraction level only the essential information is considered, nonessential information is left out
- Equability of the abstraction:
  - All information of a model on one abstraction level contains the same degree of abstraction

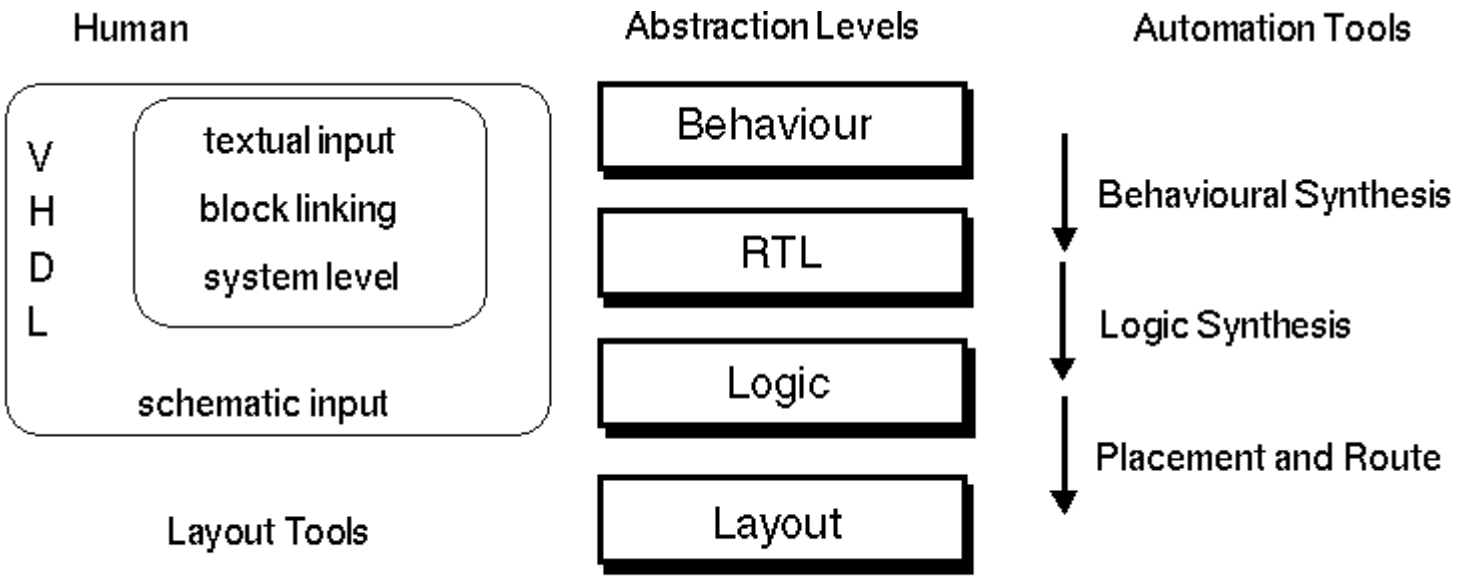


Abstraction Levels in IC Design - 2

- Behavioral - the function
  - no system clock and signal transitions are asynchronous with respect to the switching time
  - usually simulatable but not synthesizable.
- Register Transfer Level (RTL) - the design is divided into combinational logic and storage elements
  - controlled by a system clock
  - called synthesizable description
- Logic level - netlist (interconnections) with logic gates (AND, OR, NOT, ...) and storage elements
- Layout - at the bottom of the hierarchy
  - different cells of the target technology are placed on the chip and the connections are routed

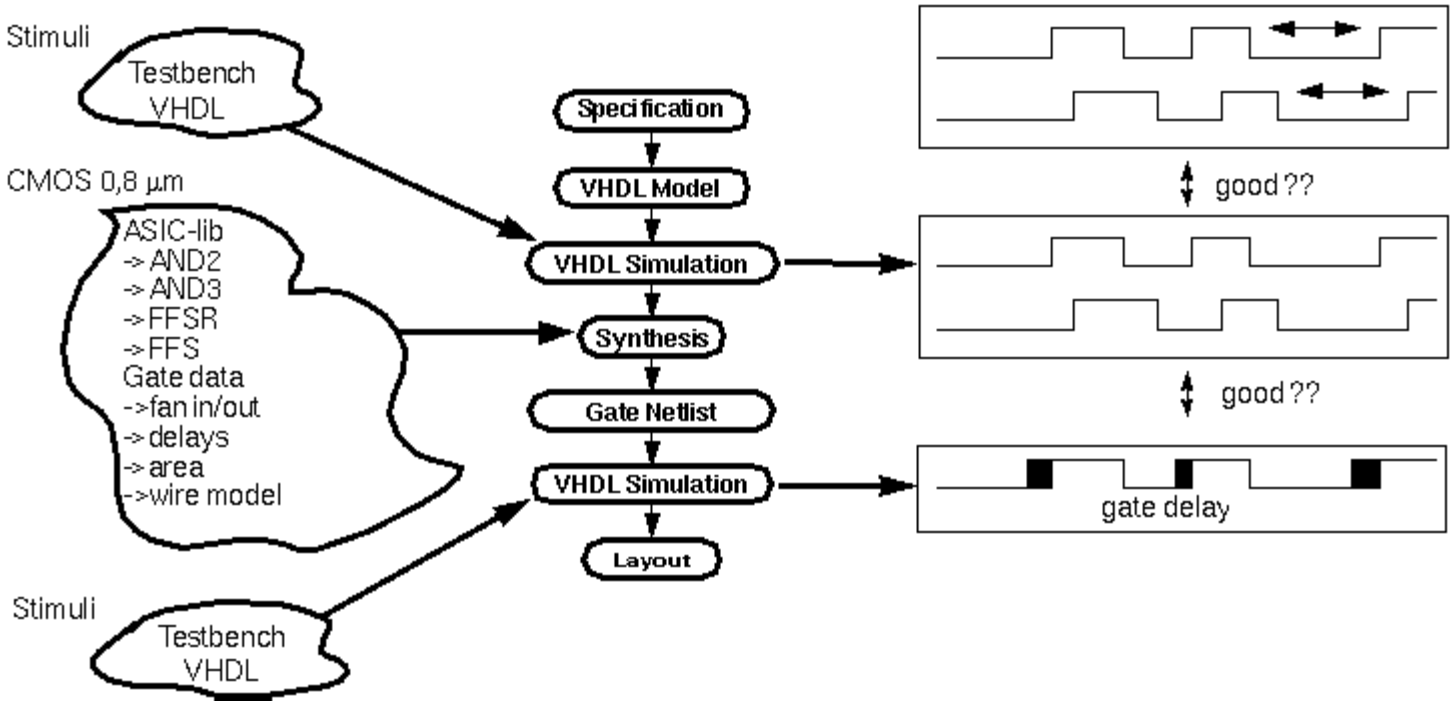
Abstraction levels and VHDL

- VHDL is applicable to the upper three abstraction levels
  - not suitable to describe a layout



ASIC Development Process

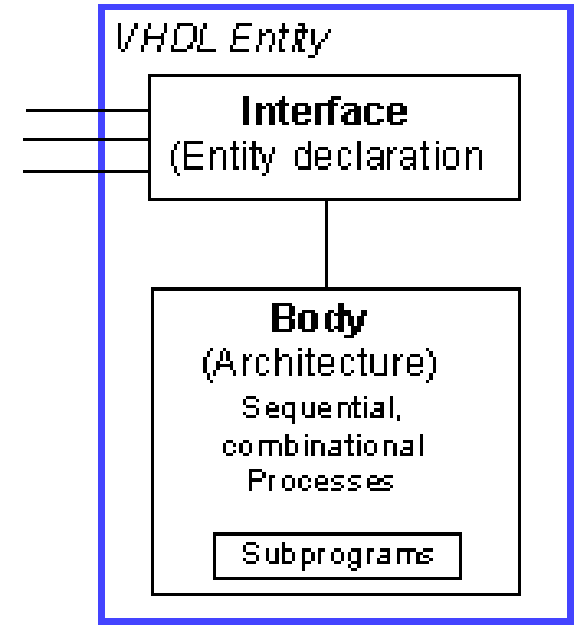
An Example VHDL Application



VHDL File Structure

- Two major parts:
  - **Entity** describes the interface
  - **Architecture** contains the description of the function

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

-- declaration of the entity
entity entity_name is
     -- declarations
end entity_name;

-- body (architecture)
architecture arch_name of entity_name is
    -- declarative
begin
    -- definitions
end arch_name;
```



VHDL Syntax Rules

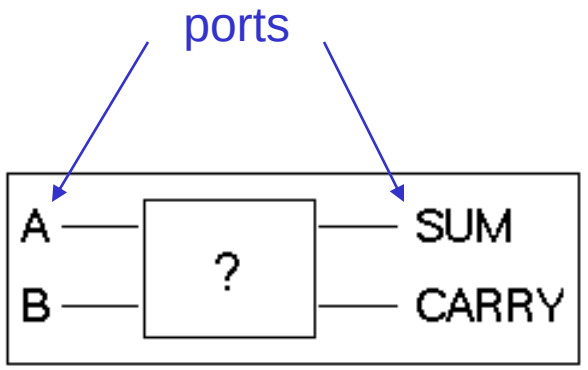
- Case insensitive
- Comments: '--' until end of line
- Statements are terminated by ';'
  - (may span multiple lines)
- List delimiter: ','
- Signal assignment: '<='
- User defined names:
  - letters, numbers, underscores
  - start with a letter

Entity Declaration

```
entity NAME_OF_ENTITY is [generic generic_declarations);]
     port (signal_names: mode type;
           signal_names: mode type;
                 :
           signal_names: mode type;
end [NAME_OF_ENTITY];
```

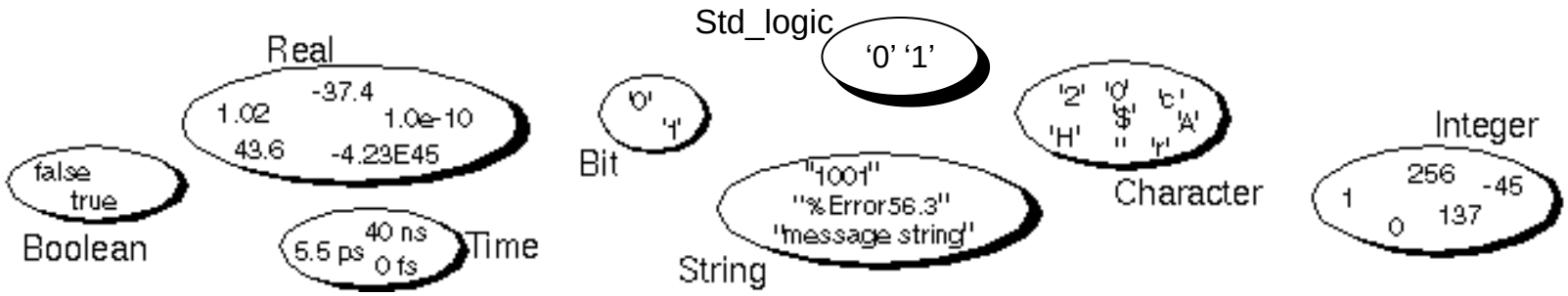
- Interface description
- No behavioral definition

```
entity HALFADDER is
   port(
      A,B: in bit;
      SUM,CARRY: out bit);
end HALFADDER;
```



Port Modes and Data Types

- *in*:
  - signal values are read-only
- *out*:
  - signal values are write-only
  - multiple drivers
- *buffer*:
  - comparable to out
  - signal values may be read, as well
  - only 1 driver
- *inout*:
  - bidirectional port

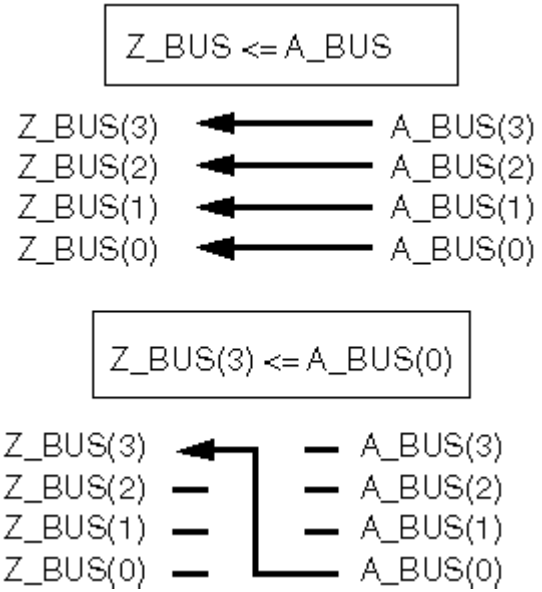


Definition of Array

- Collection of signals of the same type
- Predefined arrays
  - bit_vector (array of bit)
  - string (array of character)

```
signal A_BUS, Z_BUS : bit_vector (3 downto 0);
```

alternative: *bit_vector(0 to 3)*



Types of Assignment for 'bit' Data Types

- Single bit values are enclosed in '.'
- Vector values are enclosed in "..."
  - optional base specification (default: binary)
  - values may be separated by underscores to improve readability

```
architecture EXAMPLE of ASSIGNMENT is

   signal Z_BUS : bit_vector (3 downto 0);
   signal BIG_BUS : bit_vector (15 downto 0);

begin

   -- legal assignments:
   Z_BUS(3) <= '1';
   Z_BUS    <= "1100";

   Z_BUS    <=  b"1100";
   BIG_BUS  <=  B"0000_0001_0010_0011";

end EXAMPLE;
```

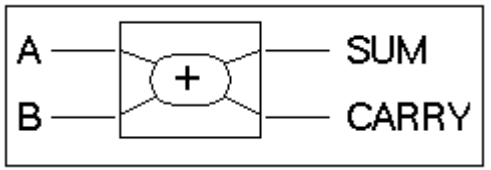
Architecture - 1

- Implementation of the design

```
architecture ARCHITECTURE_NAME of ENTITY_NAME is
   -- declarative statements
begin
   -- definitive concurrent statements
end ARCHITECTURE_NAME;
```

  - always connected with a specific entity
  - one entity can have several architectures
  - entity ports are available as signals within the architecture
  - contains concurrent statements

```
architecture RTL of HALFADDER is

begin
   SUM   <= A xor B;
   CARRY <= A and B;
end RTL;
```



Architecture - 2

- Each architecture is split into an optional *declarative part* and the *definition part*
- **Declarative part** (between the keywords ‘is’ and ‘begin’):
  - new objects that are needed only within the architecture
    - constants, data types, signals (actual), subprograms, etc. can be declared here
    - components are declared here
- **Definition part** (after 'begin'):
  - holds concurrent statements (order not important)
  - can be simple signal assignments, process statements, which group together sequential statements, and component instantiations

Architecture Models - 1

- Architecture descriptions can be in the following models:
  - **Structural modeling** - ports
  - **Behavioral**:
    - **Algorithmic** modeling – process with sequential statements
    - **Dataflow** modeling – signals in concurrent statements
  - Mixed
- Both structural and dataflow describes the entity based on its implementation, i.e. connections (of signals or ports)
- Behavioral (algorithmic) approach to modeling hardware components is different from the other two methods in that it does not necessarily in any way reflect how the design is implemented
  - basically the black box approach to modeling

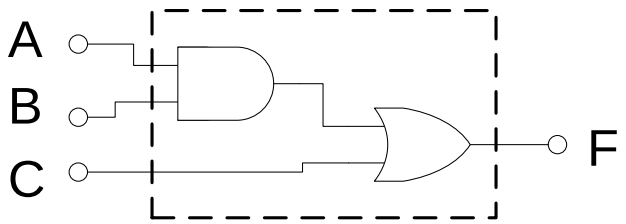
Structural Model

- Describes a circuit in terms of the interconnection of components, consisting of:
  - Components declaration
  - Signal assignments
  - Components instantiation
  - Concurrent statements
- "Connections" are made through:
  - Named association (port mapping using port=>local):
    - left side: "formals" (port names from component declaration)
    - right side: "actuals" (architecture signals)
    - independent of order in component declaration
  - If using unnamed port mapping then order is important

Structural Model: An Example

Entity Declaration

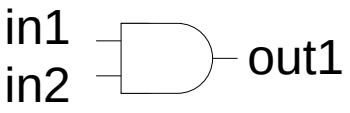
```
-- Simple Logic Circuit
library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;

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



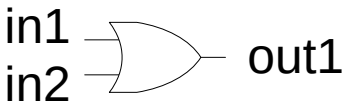
Structural Model: An Example

Components Declaration and Signal Assignment

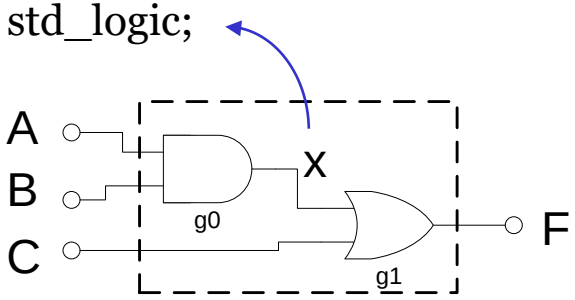
```
architecture structural_1 of s_circuit is

  component AND2
    port( in1, in2: in std_logic;
          out1: out std_logic);
  end component;

  component OR2
    port( in1, in2: in std_logic;
          out1: out std_logic);
  end component;

  signal x: std_logic;
```



AND2



OR2



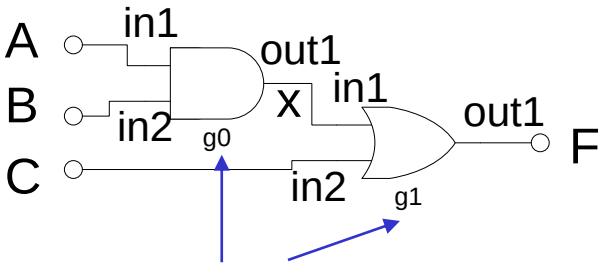
AND2 and OR2 are described in the "lcdf_vhdl.func_prims" library

Structural Model: An Example

Connections (Port Mapping)

We can use named association, where order is not important:

```
begin
  g0: AND2 port map (in1 =>A, out1 =>x, in2 =>B);
  g1: OR2 port map (in1 =>x, in2=>C, out1 =>F);
end structural_1;
```



Alternatively, using unnamed association:

```
begin
  g0: AND2 port map (A, B, x);
  g1: OR2 port map (x,C,F);
end structural_1;
```

Order must be in (in1,in2,out2) as per declaration

Behavioral Model: Dataflow

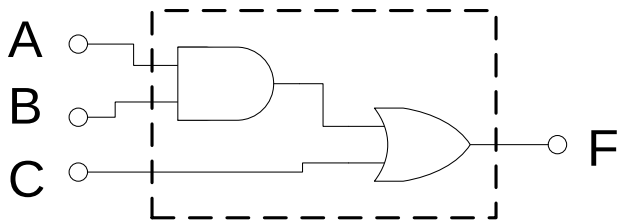
- Behavioral modeling can be done with sequential statements using the process construct or with concurrent statements
- **Dataflow** modeling describes a circuit, using concurrent statements, in terms of its function and the flow of data through the circuit
  - different from the structural modeling that describes a circuit in terms of the interconnection of components
- Consists of:
  - Signal assignment statements
  - Concurrent statements

Dataflow Model: The Example

Entity Declaration

```
-- Simple Logic Circuit
library ieee;
use ieee.std_logic_1164.all;

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

Basically the same, however we are not using any component from the "lcdf_vhdl" library

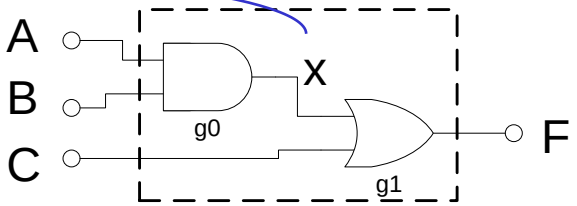


Dataflow Model: The Example

Signal Assignment & Concurrent Statements

```
architecture dataflow_1 of s_circuit is
    signal x: std_logic;

begin
    x <= A and B;
    F <= x or C;
end dataflow_1;
```

no components declaration



- no port mapping, use operators instead
- dataflow from left to right (in to out)
- these statements execute concurrently

note the statements in dataflow model are based on the circuit implementation, i.e. the path of signals

Behavioral Model: Algorithmic

Functional Description a.k.a. "High Level" Behavioral

- Model the function just like writing programming language
  - black box approach – no implementation information
  - to model complex components that would be tedious to model using the other methods
  - uses **process** construct
- Consists of:
  - Process statements
    - Sensitivity list (parameters)
    - Sequential statements
    - Signal (<=) and variable (:=) assignment statements
    - Wait statements

Process

```
PROCESS_NAME: process (SENSITIVITY_LIST)
begin
  -- SEQUESTIAL STATEMENTS
end process;
```

process name and sensitivity list are optional

- Appear in the body of an architecture declaration just as the signal assignment statement does
- Include sequential statements like those found in software programming languages
- Compute the outputs of the process from its inputs
- No direct correspondence to a hardware implementation
- Signal assignment within process is only evaluated when events occur on the signals in the process' sensitivity list
- The process are performed (or executed) in order from first to last

```
s_process: process (A,B,C)
begin
  -- SEQUESTIAL STATEMENTS
end process;
```

```
process (A,B,C)
begin
  -- SEQUESTIAL STATEMENTS
end process;
```

```
s_process: process
begin
  -- SEQUESTIAL STATEMENTS
end process;
```

Algorithmic Model: The Example

```
-- Simple Logic Circuit
library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all;

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

architecture algorithmic_1 of s_circuit is

s_process: process (A,B,C)
    begin
      F <= (A and B) or C;
    end process;

end algorithmic_1;
```

same as in dataflow

may have signal assignments

sensitivity list (optional)

Logic Simulator: Test Bench

- *Logic Simulator* interprets HDL description and uses a defined stimulus to produce readable output, such as timing diagram that predicts how hardware will behave
  - the stimulus is called a *test bench*
- ***Test bench*** is used for generating stimulus for the entity under test
  - allows design verification and detection of functional errors without having to physically create circuit
- *Test bench* is written also in VHDL, which delivers the verification environment for the model
  - stimuli are described as input signals for the model
  - expected model responses can be checked
  - appears as the top hierarchy level, and therefore has neither input- nor output ports

Test Bench: The Example - 1

-- library where s_circuit is described (or can be omitted if s_circuit is already added in the compiler environment)

```
entity tb_scircuit is
end tb_scircuit;

architecture test_1 of tb_scircuit is
    component s_circuit
        port (A, B, C: in std_logic;
        F: out std_logic);
    end component;

    signal A_I, B_I, C_I, F_I: std_logic;
```

declare test bench entity, without ports

component to test (must be in library)

test signals

Test Bench: The Example

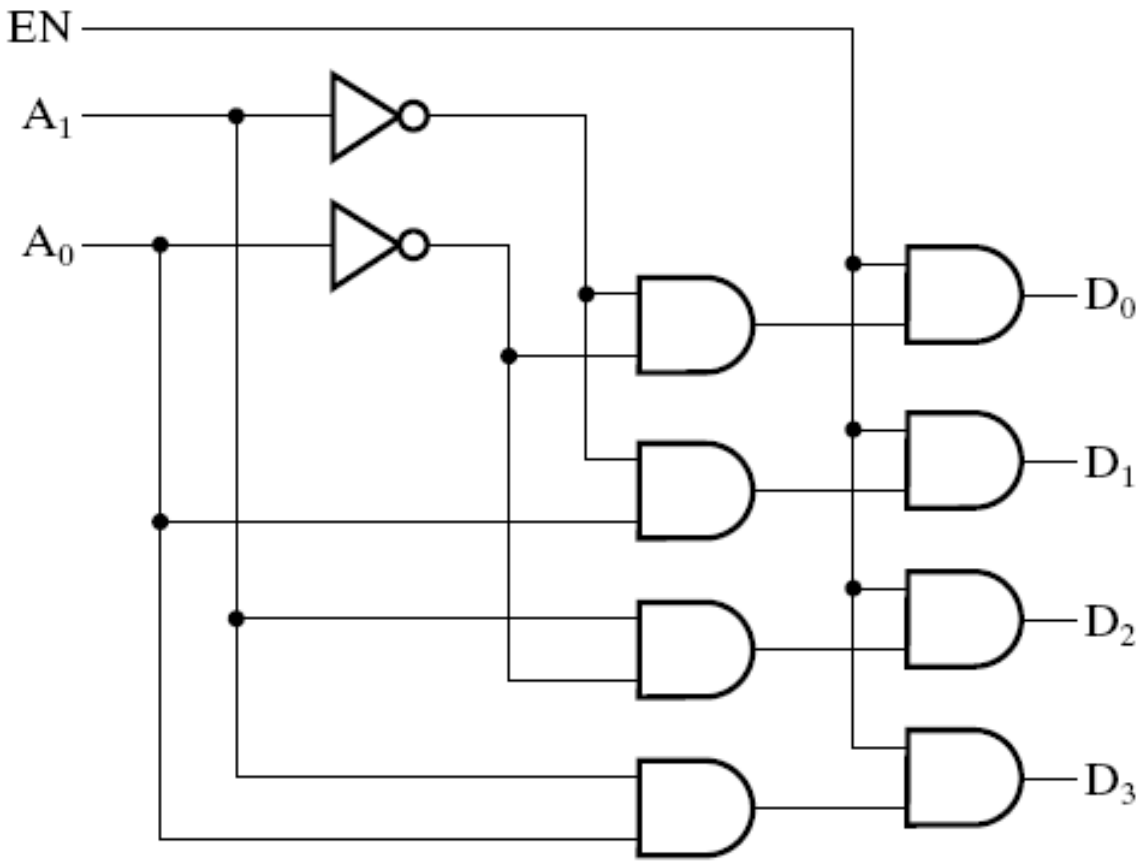
```
begin
    DUT: s_circuit port map (A_I, B_I, C_I, F_I);

    STIMULUS: process
        begin
            A_I <= ´0´ ; B_I <= ´0´ ; C_I <= '0'
            wait for 10 ns;
            A_I <= ´0´ ; B_I <= ´0´ ; C_I <= '1'
            wait for 10 ns;
            -- and so on ...
    end process STIMULUS;
end test_1;

configuration cfg_tb_scircuit of tb_scircuit is
    for test_1
    end for;
end cfg_tb_scircuit;
```

- instantiate test component
- DUT is the instance name

- assign test signals,
- input combinations

configure to use test_1 architecture

Structural VHDL 2-to-4 Decoder



Structural VHDL 2-to-4 Decoder

```
-- 2-to-4 Line Decoder with Enable: Structural VHDL
-- Description

library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;

entity decoder_2_to_4_w_enable is
  port( EN, A0, A1: in std_logic;
        D0, D1, D2, D3: out std_logic);
end decoder_2_to_4_w_enable;
```

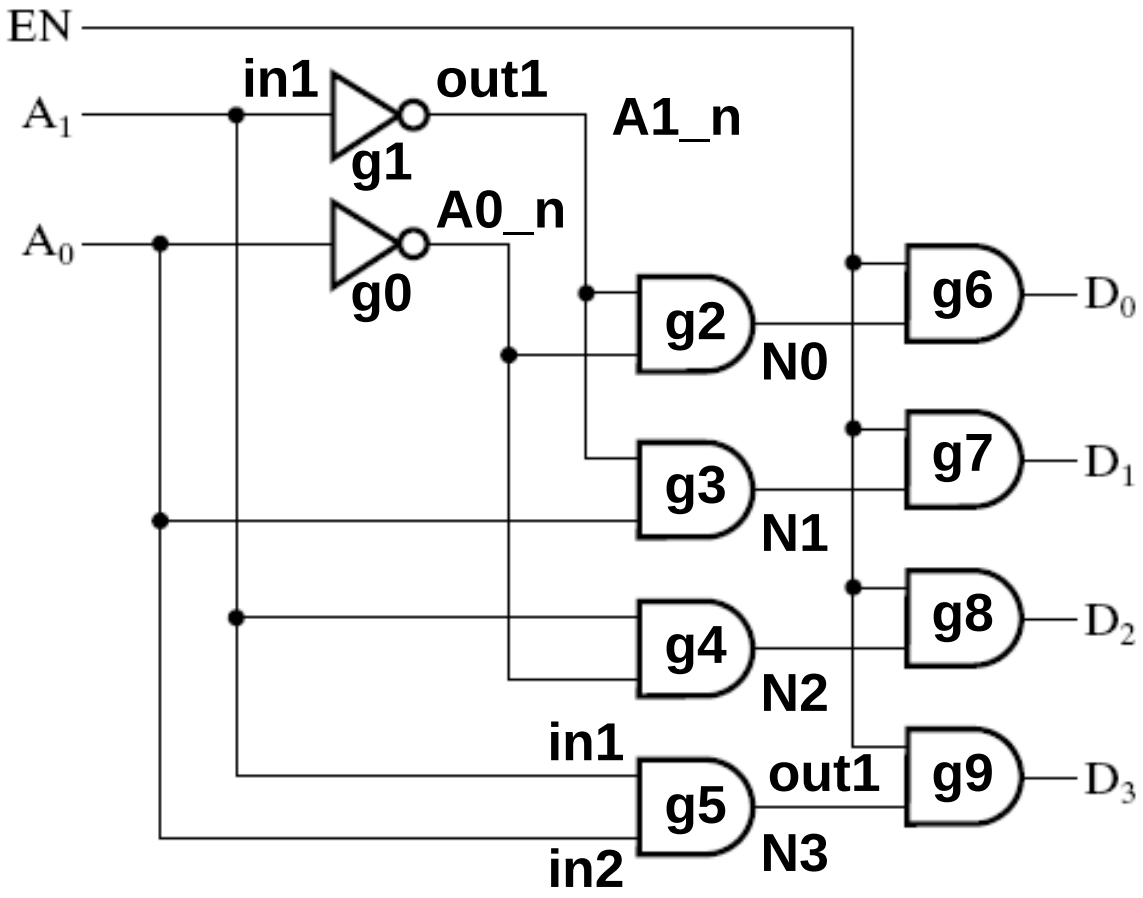
Structural VHDL 2-to-4 Decoder

```
architecture structural_1 of decoder_2_to_4_w_enable is

    component NOT1
        port( in1: in std_logic;
              out1: out std_logic);
    end component;

    component AND2
        port( in1, in2: in std_logic;
              out1: out std_logic);
    end component;

    signal A0_n, A1_n, N0, N1, N2, N3: std_logic;
```

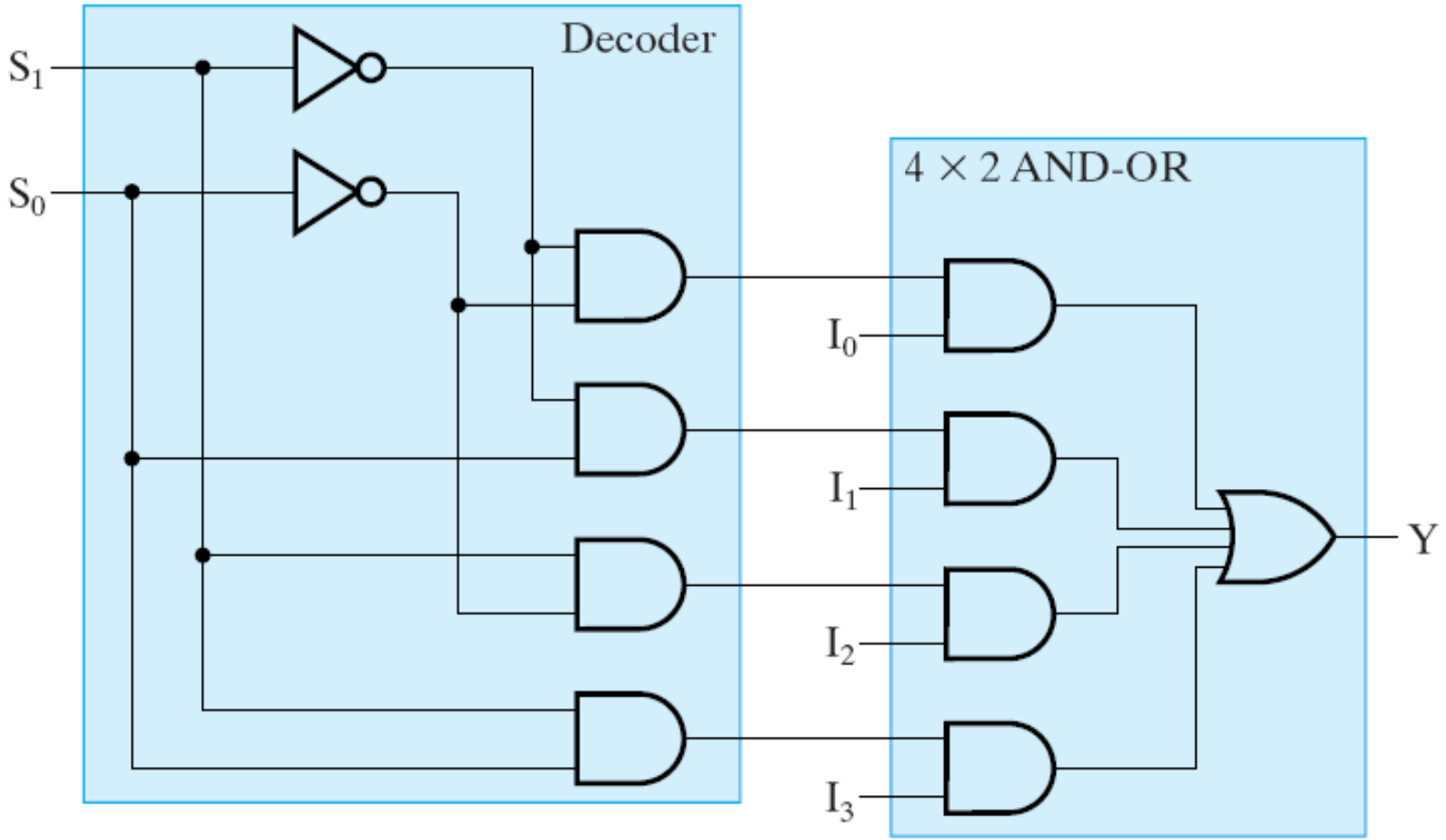
Structural VHDL 2-to-4 Decoder

```
begin
   g0: NOT1 port map (in1 => A0, out1 =>A0_n);
   g1: NOT1 port map (in1 => A1, out1 => A1_n);
   g2: AND2 port map (in1 => A0_n, in2 => A1_n, out1 => N0);
   g3: AND2 port map (in1 => A0, in2 => A1_n, out1 => N1);
   g4: AND2 port map (in1 => A0_n, in2 => A1, out1 => N2);
   g5: AND2 port map (in1 => A0, in2 => A1, out1 => N3);
   g6: AND2 port map (in1 => EN, in2 => N0, out1 => D0);
   g7: AND2 port map (in1 => EN, in2 => N1, out1 => D1);
   g8: AND2 port map (in1 => EN, in2 => N2, out1 => D2);
   g9: AND2 port map (in1  => EN, in2 => N3, out1 => D3);
end structural_1;
```

Structural VHDL 2-to-4 Decoder



Structural VHDL 4-to-1 Multiplexer



Structural VHDL 4-to-1 Multiplexer

```
-- 4-to-1 Line Multiplexer: Structural VHDL Description

library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;

entity multiplexer_4_to_1_st is
  port( S: in std_logic_vector(0 to 1);
        I: in std_logic_vector(0 to 3);
        Y: out std_logic);
end multiplexer_4_to_1_st;
```

Structural VHDL 4-to-1 Multiplexer

```
architecture structural_2 of multiplexer_4_to_1_st is

    component NOT1
        port( in1: in std_logic;
              out1: out std_logic);
    end component;

    component AND2
        port( in1, in2: in std_logic;
              out1: out std_logic);
    end component;
```

Structural VHDL 4-to-1 Multiplexer

```
component OR4
        port( in1, in2, in3, in4: in std_logic;
                out1: out std_logic);
end component;

signal S_n: std_logic_vector(0 to 1);
signal D, N: std_logic_vector(0 to 3);
```

Structural VHDL 4-to-1 Multiplexer

```
begin
  g0: NOT1 port map (S(0), S_n(0));
  g1: NOT1 port map (S(1), S_n(1));
  g2: AND2 port map (S_n(1), S_n(0), D(0));
  g3: AND2 port map (S_n(1), S(0), D(1));
  g4: AND2 port map (S(1), S_n(0), D(2));
  g5: AND2 port map (S(1), S(0), D(3));
  g6: AND2 port map (D(0), I(0), N(0));
  g7: AND2 port map (D(1), I(1), N(1));
  g8: AND2 port map (D(2), I(2), N(2));
  g9: AND2 port map (D(3), I(3), N(3));
  g10: OR4 port map (N(0), N(1), N(2), N(3), Y);
end structural_2;
```

Dataflow VHDL 2-to-4 Decoder

- A dataflow description describes a circuit in terms of function rather than structure
- Concurrent assignment statements are executed concurrently whenever one of the values on the right-side of the statement changes

Dataflow VHDL 2-to-4 Decoder

```
-- 2-to-4 Line Decoder: Dataflow VHDL Description

library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all;

entity decoder_2_to_4_w_enable is
  port( EN, A0, A1: in std_logic;
        D0, D1, D2, D3: out std_logic);
end decoder_2_to_4_w_enable;
```

Dataflow VHDL 2-to-4 Decoder

```
architecture dataflow_1 of decoder_2_to_4_w_enable is

    signal A0_n, A1_n: std_logic;
begin
    A0_n <= not A0;
    A1_n <= not A1;
    D0 <= A0_n and A1_n and EN;
    D1 <= A0 and A1_n and EN;
    D2 <= A0_n and A1 and EN;
    D3 <= A0 and A1 and EN;
end dataflow_1;
```

Behavioral VHDL 4-to-1 Multiplexer (1) - 1

(using When-Else)

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

entity multiplexer_4_to_1_we is
    port ( S : in std_logic_vector(1 downto 0);
           I : in std_logic_vector(3 downto 0);
           Y : out std_logic);
end multiplexer_4_to_1_we;
```

Behavioral VHDL 4-to-1 Multiplexer (1) - 2

(using When-Else)

```
architecture function_table of multiplexer_4_to_1_we is
begin
    Y <= I(0) when S = "00" else
         I(1) when S = "01" else
         I(2) when S = "10" else
         I(3) when S = "11" else
         'X';
end function_table;
```

Behavioral VHDL 4-to-1 Multiplexer (2) - 1

(using With-Select)

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

entity multiplexer_4_to_1_ws is
    port (S : in std_logic_vector(1 downto 0);
          I : in std_logic_vector(3 downto 0);
          Y : out std_logic);
end multiplexer_4_to_1_ws;
```

Behavioral VHDL 4-to-1 Multiplexer (2) - 2

(using With-Select)

```
architecture function_table_ws of multiplexer_4_to_1_ws is
begin
    with S select
    Y <= I(0) when "00",
         I(1) when "01",
         I(2) when "10",
         I(3) when "11",
         'X' when others;
end function_table_ws;
```

Behavioral VHDL 4-to-1 Multiplexer

- When-else permits decisions on multiple distinct signals
  - Synthesis typically results in more complex logical structure
  - Needs to take into account priority order
- With-select can depend on only a single Boolean condition
- 'x' is an 'unknown' value, which will be generated during simulation. However, Y will always take on a 0 or 1 value in an actual circuit.

More examples: hierarchical

```
-- 4-bit Adder: Hierarchical
Dataflow/Structural
-- (See Figures 4-4 and 4-5 for logic
diagrams)

library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
    port (x, y : in std_logic;
    s, c : out std_logic);
end half_adder;

architecture dataflow_3 of
                half_adder is
    begin
        s <= x xor y;
        c <= x and y;
    end dataflow_3;
```

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

entity full_adder is
    port (x, y, z : in std_logic;
    s, c : out std_logic);
end full_adder;

architecture struc_dataflow_3 of
                        full_adder is

component half_adder
    port(x, y : in std_logic;
    s, c : out std_logic);
end component;

signal hs, hc, tc: std_logic;
-- use 2 HA to make 1 FA
begin
    HA1: half_adder
        port map (x, y, hs, hc);
    HA2: half_adder
        port map (hs, z, s, tc);
    c <= tc or hc;
end struc_dataflow_3;
```

previous slide

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

entity adder_4 is
    port(B, A : in std_logic_vector(3 downto 0);
    C0 : in std_logic;
    S : out std_logic_vector(3 downto 0);
    C4: out std_logic);
end adder_4;

architecture structural_4 of adder_4 is

component full_adder
    port(x, y, z : in std_logic;
    s, c : out std_logic);
end component;

signal C: std_logic_vector(4 downto 0);
-- use 4 FA to make 4-bit FA
begin
    Bit0: full_adder
    port map (B(0), A(0), C(0), S(0), C(1));
    Bit1: full_adder
    port map (B(1), A(1), C(1), S(1), C(2));
    Bit2: full_adder
    port map (B(2), A(2), C(2), S(2), C(3));
    Bit3: full_adder
    port map (B(3), A(3), C(3), S(3), C(4));
    C(0) <= C0;
    C4 <= C(4);
end structural_4;
```

More examples: high-level behavioral

```
-- 4-bit Adder: Behavioral Description
-- compare with dataflow/structural description in previous slides
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder_4_b is
    port(B, A : in std_logic_vector(3 downto 0);
    C0 : in std_logic;
    S : out std_logic_vector(3 downto 0);
    C4: out std_logic);
end adder_4_b;

architecture behavioral of adder_4_b is

signal sum : std_logic_vector(4 downto 0);

begin
    sum <= ('0' & A) + ('0' & B) + ("0000" & C0);
    C4 <= sum(4);
    S <= sum(3 downto 0);
end behavioral;
```

References

- http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html
- http://www.vhdl-online.de/tutorial/englisch/t_2.htm
- http://www.gmvhdl.com/VHDL.html
- M Morris Mano, Charles R Kime, “Logic and Computer Design Fundamentals”, 4th Edition

Summary

- HDL – high level representation of hardware
- VHDL is one of few popular HDL
- VHDL describes hardware in three models:
  - Structural
  - Behavioral - Dataflow
  - Behavioral – Algorithmic (High Level)
- Test Bench is VHDL describing stimulus to the inputs of a hardware device to observe its output