Introduction to VHDL

Sadiq M. Sait, Ph.D.

sadiq@ccse.kfupm.edu.sa
Department of Computer Engineering
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

March 17 - 21, 2001

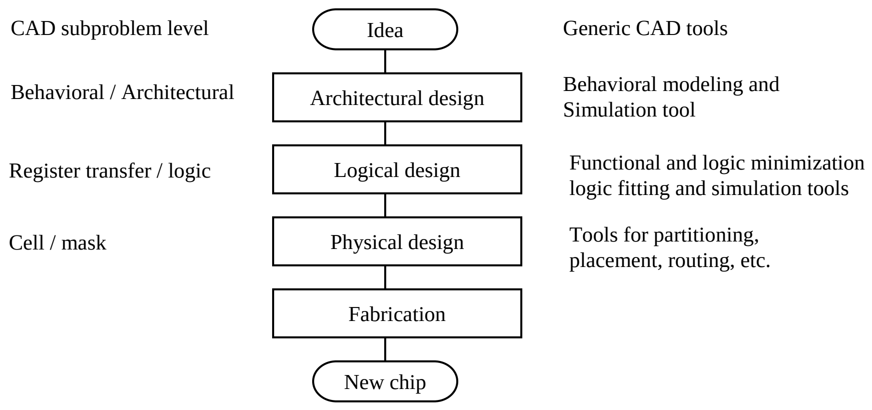
Content Discussed in this Session

- Modeling
- History
- Applications
- Design Flow
- VHDL Characteristics
- VHDL Basics/Overview
- Examples
- Summary

What is A Modeling Language

- A Modeling Language.
- What is Modeling.
- When do you Model?
- What do you Model
  - » Various Levels of Modeling & Abstraction (what does it mean?)
    - Behavioral/Architectural Level modeling
    - Structural Level modeling
    - Dataflow Level modeling
- Characteristics/Requirements of Models.
- What are the main advantages

Design Flow & Levels of Abstraction


CAD subproblem level
Idea
Generic CAD tools
Behavioral / Architectural
Architectural design
Behavioral modeling and Simulation tool
Register transfer / logic
Logical design
Functional and logic minimization logic fitting and simulation tools
Cell / mask
Physical design
Tools for partitioning, placement, routing, etc.
Fabrication
New chip

Architectural Design

- Generally carried out by expert human engineers.
- Decisions made effect cost/performance of the design. Some examples are
  - » What should be the instruction set (IS) of the Processor (P) ?
    - – What memory addressing modes should be supported?
    - – Should the IS be compatible with another in the market?
  - » Should instruction pipelining be employed?
    - – If yes, then what should be the depth?
  - » Should the P be provided by an on-chip cache?
    - – How big should the cache be?
    - – What should be the organization of the cache?
    - – Should instruction cache be separated from data cache?

Architectural Design

- How should the arithmetic unit be designed?
  - » Bit Serial or Bit Parallel?
  - » Bit Serial will save on hardware but lose on performance!
- How will the Processor interface to the external world?
  - » Are there any International Standards to be met?
- Notes: This level of design cannot be done by a computer program, there are tools that can aid the system Architect.

Architectural Design Tools

- This level of design cannot be done by a computer program, there are tools that can aid the system Architect. Example:
  - » Tools can be used to experiment with innovative ideas
  - » Tools are also available to
    - – tune the size of the cache, or
    - – Determine the depth of the pipeline, etc.

Data/Control Path Design

- Once the architecture is defined, it is necessary to carry out two things:
  1. Detailed logic design of individual circuit modules
  2. Derive the control signals necessary to activate and deactivate the circuit modules.

1. The First step is known as *data path design.*
2. The Second step is called *control path design.*

Data Path Design

- The data path of the circuit includes the various
  - » functional blocks (such as adders, multipliers, ALU, etc),
  - » storage elements (shift register, RAM, buffers, stacks, queues),
  - » hardware components to allow transfer of data among functional blocks and storage elements.
    - – Data transfer is achieved using tri-state busses or a combination of multiplexer(s)/de- multiplexer(s).

Control Path Design

- The control path of the circuit generates the various control signals necessary to operate the circuit:
  - » These are necessary to initialize the storage elements,
  - » To initiate data transfers among functional blocks and storage elements,
  - » And so on.
- The control path may be implemented using
  - » Hardwired control (random logic), or
  - » Through micro-programmed logic.

An Example

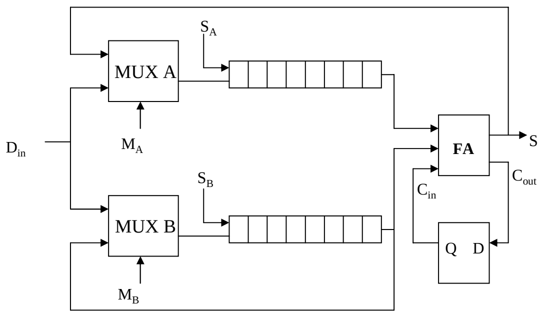
- The Problem: It is required to design an 8-bit adder. The two operands are stored in two 8-bit shift registers *A* and *B*. At the end of the addition operation, the sum must be stored in the *A* register. The contents of the *B* register must not be destroyed. The design must be as economical as possible in terms of hardware.

Possible Solutions

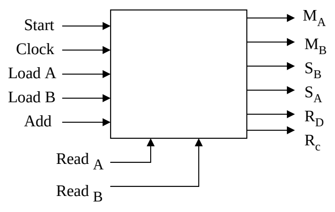
- There are numerous ways to design the above circuit,some of which are listed below.
  - » Use an 8-bit carry look-ahead adder.
  - » Use an 8-bit ripple-carry adder
  - » Use two 4-bit carry look-ahead adders and ripple the carry between stages.
  - » Use a 1-bit adder and perform the addition serially in 8 clock cycles.

Our Choice

- Since it is specified that the cost must be minimum, the last option seems best.

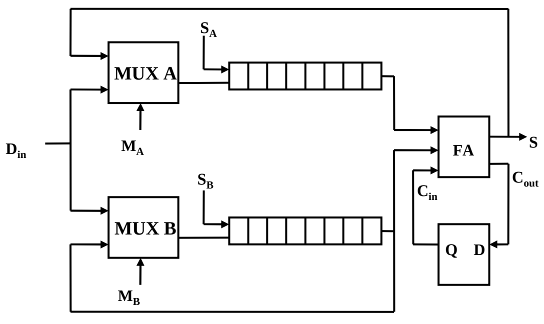


(a)

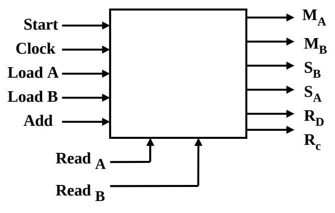


(b)

Organization of a Serial Adder



(a)



(b)

Data Path of Serial Adder

- Consists of two 8-bit shift regiseters
- A full adder
- A D-flipflop
- Two multiplexers
- A three bit-counter (to count the number of times bit-wise addition is performed)

Control Path of Serial Adder

- The relevant control signals are:
  - » S_A to Shift the register A right by one bit
  - » S_B to shift the register B right by one bit
  - » M_A to control multiplexer A.
  - » M_B to control multiplexer B
  - » R_D to Reset the D flip-flop
  - » R_C to Reset the counter
  - » START to commences the addition

Control Algorithm of Serial Adder

```
Forever do
While (START = 0) skip;
Reset the D flip-flop and the counter;
Set $M_A$ and $M_B$ to 0;
 Repeat
    Shift registers A and B right by one
    counter = counter + 1;
Until counter = 8;
```

Some Observations

- Design involves trade-offs between
  - » Cost
  - » Performance
  - » Testability
  - » Power dissipation
  - » Fault tolerance
  - » Ease of design
  - » Ease of making changes to the design.
- Serial is cheap but slow, parallel fastest in terms of performance but most costly
- The different ways we can think of building an 8-bit adder constitutes what is known as *design space* (at a particular level of abstraction).
  - » Each method of implementation is called a point in the design space.

High-Level Synthesis

- A circuit specification may pose *constraints* on one or more aspects of the final design.
  - » Example: When the spec says that the circuit must operate at 15MHz, we have a constraint on timing performance of the circuit.
- Given a spec, the objective is to arrive at the design which meets all the constraints, and optimizes on or more design aspect.
- This problem is known as hardware synthesis
- Programs are available for both data path synthesis and control path synthesis
- The automatic generation of data path and control path is known as *high-level synthesis.*

High-Level Synthesis

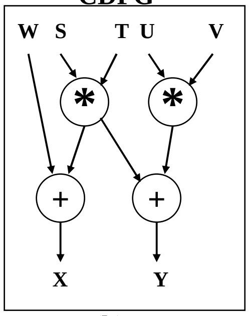
- Tasks involved in HLS (the process of automatically translating abstract behavioral models of digital systems to implementable hardware) are scheduling and allocation
- Scheduling distributes the execution of operations throughout time steps
- Allocation assigns hardware to operations and values.
  - Allocation of hardware cells include functional unit allocation, register allocation and bus allocation.
  - Allocation determines the interconnections required.

Behavioral Description and its CDFG

X = W + (S * T)
Y = (S * T) + (U * V)

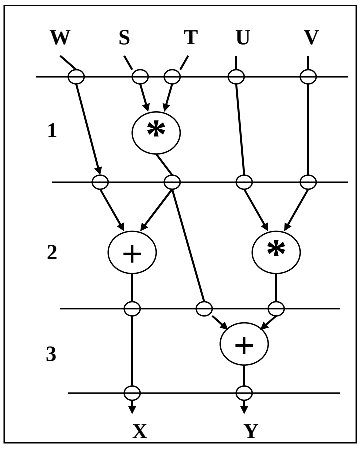
(a)

CDFG



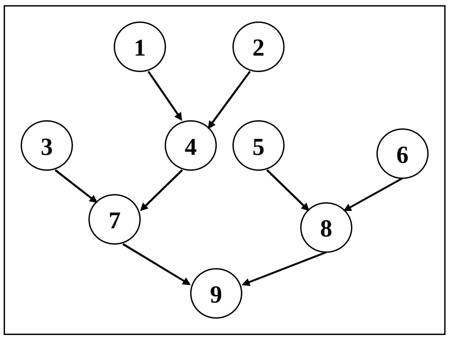
(b)

Scheduled CDFG



(c)

Example 1: A CDFG and 3 Solutions

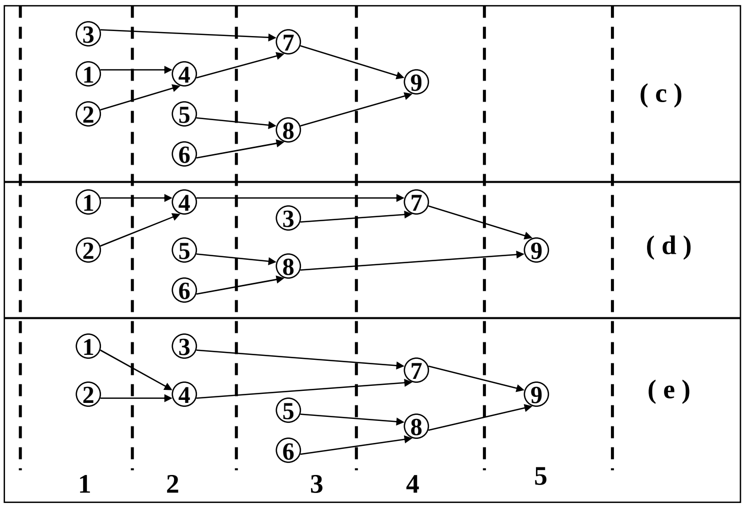


(a)

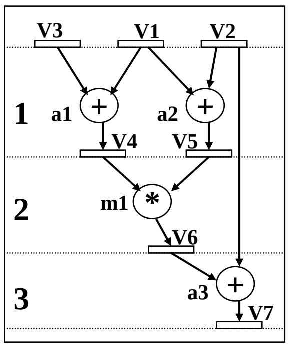
	CSs	REGs	FUs
(c)	4	4	3
(d)	5	3	3
(e)	5	4	2

(b)

Example 1



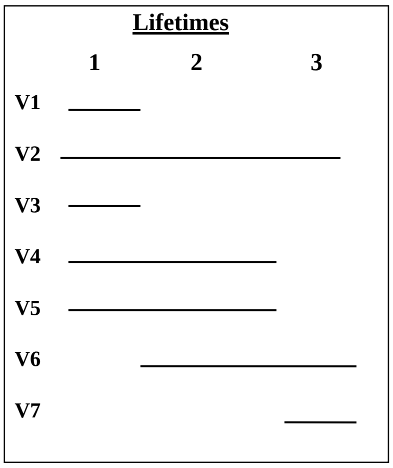
Register Allocation



(a)

V4 = V1 + V3
V5 = V1 + V2
V6 = V4 * V5
V7 = V2 + V6

(b)



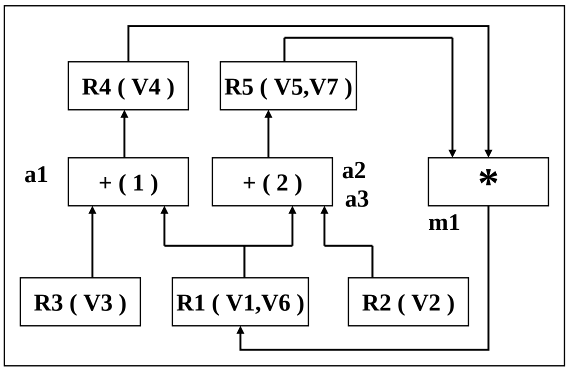
(c)

Register Assignment & Synthesis

Register Assignment

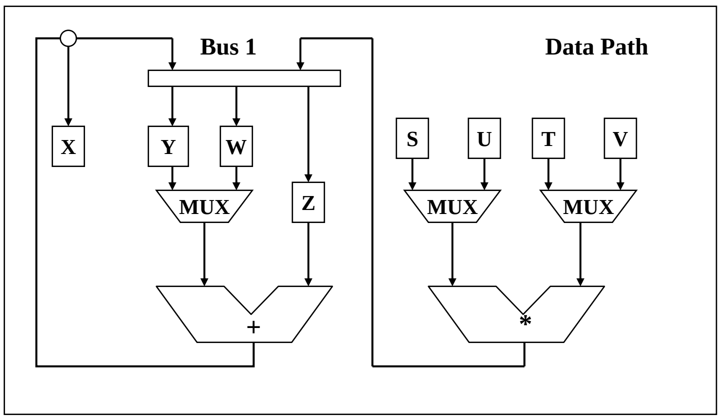
R1 : V1 , V6
R2 : V2
R3 : V3
R4 : V4
R5 : V5, V7

(d)



(e)

Final Output



(f)

Logic Design

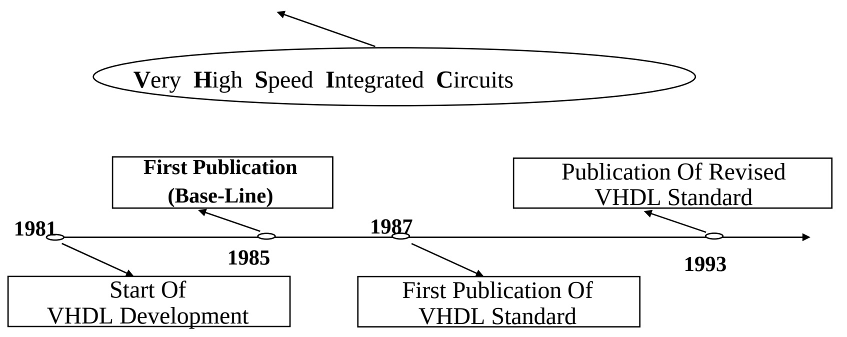
- Data path and control path (derived automatically or manually) will have components such as ALU, registers, multiplexers, buffers, and so on.
- Further design steps depend on
  - » How the circuit is implemented (PCB/VLSI)
  - » Are all the components readily available as off-the-shelf components/ICs
- If the circuit is implemented on PCB, then the next stage is to select the components to minimize cost/performance
- Following the selection procedure ICs are selected and placed on one or more circuit boards and interconnected.
- A similar procedure is used when implementing on a VLSI chip using pre-designed components from a *module library* (called *macro cells*).

Historical Glimpses/Background of HDLs

- **1960's - 1980's AHPL, CDL, DDL in classroom.**
  - » Number of languages mushrooms to over 200 languages, all of them either proprietary or academic
- **1983 VHSIC Program initiates definition of VHDL**
- **1987 VHDL Standard (IEEE 1076) approved**
- **1990 Verilog dominates the marketplace**.
  - » VHDL gaining acceptance as the second language due to standards effort and DoD mandated use.

VHDL- Time Line

- **VHDL** = VHSIC **H**ardware **D**escription **L**anguage


Very **H**igh **S**peed **I**ntegrated **C**ircuits
First Publication (Base-Line)
Publication Of Revised VHDL Standard
1981
1987
1985
1993
Start Of VHDL Development
First Publication Of VHDL Standard

Background, continued

- **1992 IEEE 1164 (3, 4, 9-valued logic standard adopted)**
- **1993 VHDL re-balloted**
  - » **minor changes make it more user-friendly.**
- **1994 Widespread acceptance of VHDL.**
  - » **CAE tools operate at speeds comparable to Verilog. Mentor, Cadence, Viewlogic, Synopsys all provide full VHDL compilation/simulation and synthesize with subsets of VHDL.**

Applications of HDLs

- **HDLs exist to satisfy a variety of purposes and are used in a number of different ways.**
- Therefore,
  - » there are features of VHDL (and any other design language) that will be useless in some applications, and confusing in other applications.
  - » Some descriptive features of VHDL may even lead to bad synthesis.
- A major aspect of this course will be understanding how VHDL should be used to permit synthesis tools to produce good implementations of designs.

Objective of VHDL

- The main objective of VHDL was to provide a unified notation to describe Electronic Systems (digital hardware) at various levels of abstractions.
- Abstraction, what does it mean
- Levels of Abstractions (gate level, upwards)

Objectives (re-visited)

- The models (or VHDL descriptions) are both human/machine readable.
  - » Like all HDLs, VHDL also comes with a simulator (to verify the correctness of models)
- Also, one single hardware description is used for
  - » simulation and verification,
  - » synthesis (translation to hardware), and
  - » testing.
- To support the communication of design data (no more black boxes)
- To aid the maintenance, modification, and procurement of hardware.

Purposes Served by VHDL

To understand why VHDL looks the way it does, it will help to examine the variety of purposes it serves

- » **Design Specification**
- » **Design Documentation**
- » **Design Verification**
- » **Product Test Generation**
- » **Hardware Synthesis**

- Language must seek to support these!

Design Specification

- Definition of functional interfaces
  - » **concurrent ==> structure (space)**
  - » **sequential ==> behavior (time)**
- Definition of design functions (behavior)
  - » **by control flow (procedural)**
  - » **by data flow (concurrent)**
- Project partitioning
  - » **in space (structural)**
  - » **in time (behavioral)**

Design Documentation

- Language standardization
  - » to improve quality and efficiency of communication, broaden audience
- Interface description for users
  - » often as components of next level higher system: physical, structural, and "pin" functions
- Express usage constraints
  - » e.g. disallowed input timings, combinations, output loads
- Express functional behavior of the design:
  - » internal states, data structures (e.g. format used in a floating point unit)
  - » algorithms implemented
- Show communication protocols between design entities

Design Verification - SIMULATION

- The Usual Method
  - » Highly developed tools
  - » Inherently behavioral (structural simulators consist of ordered calls to primitive procedures to model corresponding primitive structures)
  - » Limited by the effectiveness of the design test program developed
- Requires
  - » Language semantics to be executable

Design Verification - Formal Methods

- Require
  - » common language for specification of design goals and description of implementation to meet those goals
  - » formal (mathematically rigorous) language definition to permit logical transformation of descriptions to prove equivalence. Such mathematical languages inherently are declarative
  - » language that can describe both time and structure
- Correctness by construction (silicon compilation) more nearly realized
  - » than automatic verification systems

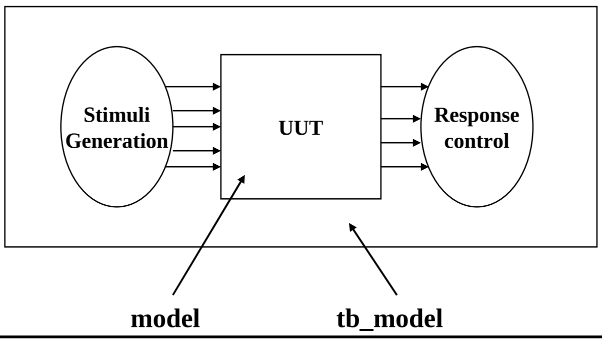
Product Test Generation

Requires

» constrained sets of values, especially for inputs

» behavioral descriptions of sequential designs

» machine analyzable design specifications



Hardware Synthesis

- Stepwise refinement of design
  - » by architectural decomposition (either structural or behavioral)
- Transformations from behavioral models to corresponding structural and physical models,
  - » e.g. PLA generators, standard cells for shift registers, adders, etc.
- Relating scaling parameters with expressions
- Enforcement of design constraints
- Register transfer level allocation,
  - » dataflow optimizations
  - » expression transformations for optimization

VHSIC Motivations for Creating/Using VHDL

- Standardization of Documentation
  - » improved communication of requirements between military, contractors, and subcontractors
- System Design Time and Cost
  - » reduced ambiguity in specification of design interfaces and design functions
  - » reusability of existing designs

VHSIC Motivations for Creating/Using VHDL

- Open-system CAE Tools
  - » can change CAE system without losing use of existing designs
  - » elimination of language translators
- Improved Integration of Multi-vendor Designs
  - » shared design databases become possible
  - » standard cells, behavioral models
- Improved Understanding of Design Science
  - » top-down, middle-out, bottom-up

VHDL Basics/Overview

Sadiq M. Sait, Ph.D.

sadiq@ccse.kfupm.edu.sa
Department of Computer Engineering
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

March 17 - 21, 2001

VHDL Basics/Overview

- VHDL an acronym : V (of VHSIC, stands for Very High Speed Integrated Circuit) Hardware Description Language.
- Names coined by DOD (department of defense, the first institution to understand the benefits of design language based documentation, modeling, and simulation of electronic devices)
- VHDL simulators took hold in early '90s.
- PLA/FPGA designers started using them on larger designs.
- Note: The word "synthesis" was not mentioned as one of the reasons for creating VHDL (as it was primarily intended for documentation, and modeling/simulation)

How a System Communicates

- How a System Communicates with its environment?
  - » Whatever function a system performs, it must get some input from its environment and output some data.
  - » In other words the system must communicate with its environment.
  - » The communication part of a system specification is called the interface (without which the system would be useless).
- The system's interface is described in VHDL by its ***entity.***
  - » This is the basic design unit for any system
  - » It is impossible to have a VHDL system without entity

Body of a System

- To accomplish the desired functionality, data must undergo some transformation inside the system.
- This is handled by the system's inner part of *body* (also called the *architecture*).
- The functionality can be simple as turning some switch on/off, or as complicated as an autopilot of a jetliner.
- However, in each case a system can be considered as a composition of a body (architecture) and an interface (entity)

External Support

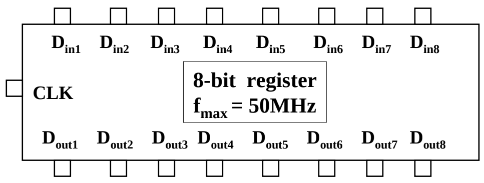
- Some systems may derive additional features through supportive elements.
  - » Example: A plug-in card that speed graphic in PC
- While such an add-on card can be considered a part of the system, for the sake of clarity let's consider it as a third element of systems's design (called *package* in VHDL).
- All 3 elements of a system design
  - » Interface (entity)
  - » Body (architecture)
  - » Add-ons (package)

  will be discussed in this session.

Entity

- A good practice to start any design is with the analysis of its environment, entity is the main part of any spec.
- In VHDL, the name of the system is the same as the name of its entity.
- Entity comprises two parts:
  - » *parameters* of the system as seen from outside such as bus-width of a processor or max clock frequency
  - » *connections* which are transferring information to and from the system (system's inputs and outputs)

Illustration of Entity


D_{in1} D_{in2} D_{in3} D_{in4} D_{in5} D_{in6} D_{in7} D_{in8}
CLK
8-bit register
f_{max} = 50MHz
D_{out1} D_{out2} D_{out3} D_{out4} D_{out5} D_{out6} D_{out7} D_{out8}

entity Eight_bit_register **is**

parameters

CLK one-bit input

connections

end entity Eight_bit_register

Parameters/Connections

- All parameters are declared as generics and are passed on to the body of the system
- Connections, which carry data to and from the system, are called *ports*. They form the second part of the entity.
- Note: The importance of entity is so great that the architecture in VHDL is specified as the *architecture of entity*
  - » Example

```
entity My-Digi-System is
 ..
 ..
 ..
end entity My-Digi-System;
```

Entity

- Examples

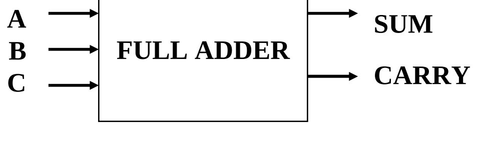
```
entity ALU is
 Port ( A,B          : in     bit_vector(7 downto 0);  -- comment 1
        Mod1, Mod2 : inout  bit_vector(3 downto 0); -- comment 2
           C          : out    bit_vector(7 downto 0)  -- comment 3
      );
 end entity ALU;

 -- name = Mod
 -- mode = bidirectional
 -- type  = 4-bit wide bus
 -- note,
```

- port declarations are separated by semicolon,
- no semicolon after the last port declaration.
- Double hyphen for comments.
- Multiple ports of the same mode and type can be declared in one statement.
- Properly documenting ports is as good as a good entity description
- Also you will see generics (and their applications) in subsequent lectures.

Entity

Examples

```
   entity FULLADDER is
 - -(After a double minus sign (-) the rest of
 -- the line is treated as a comment)
 - -
 - -Interface description of FULLADDER
 port (           A, B, C: in bit;
             SUM, CARRY: out bit);
end FULLADDER;
```

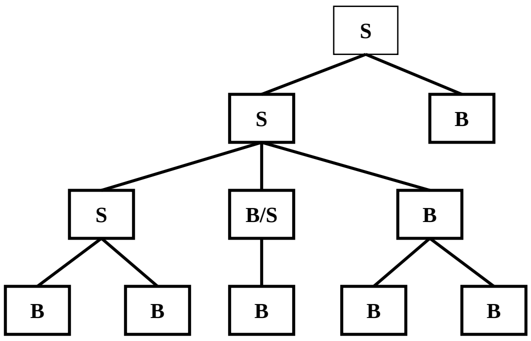


Entity

```
-- Example Data Flipflop;
    entity DFF is
    -- parameter: width of the data
    generic (width: integer);
    --input and output signals
    port (  CLK, NR: in bit;
            D: in bit_vector(1 to width);
            Q: out bit_vector(1 to width));
  end DFF;
```

Mixing S/B in Architecture Hierarchical Design

S: structural description
B: behavioral description
B/S: mixed description



Types of Architecture Descriptions

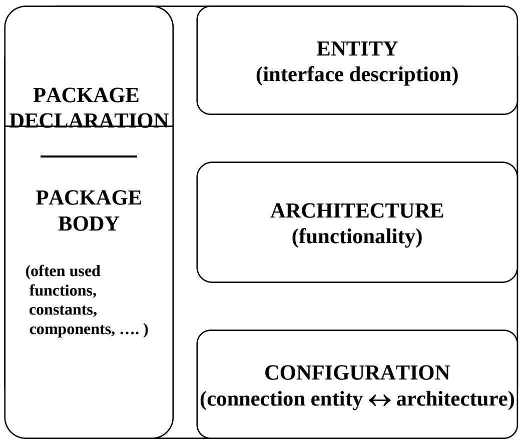
- Each system can be either in terms of its functionality (behavior) or structure, which requires different kinds of information about the system.
- Usually the synthesis tool works with both of them.
  - » First the expected functionality has to be specified and formalized;
  - » And then it has to be transformed into structural equivalent
  - » The structural equivalent is more suitable for synthesis tool
- Parts of this translation can be done automatically
- However a fully functional (behavioral) synthesis tools is not yet available.
- This is because the behavioral spec is only a description of outputs' response to inputs' changes.

Relationship between Entity & Architecture

Example

```
    entity My-Digi-System is
     (
     ...
     ...
     );
    end entity My-Digi-System;

 architecture MY-Arch of My-Digi-System is
 ...
 end architecture MY-Arch;
```

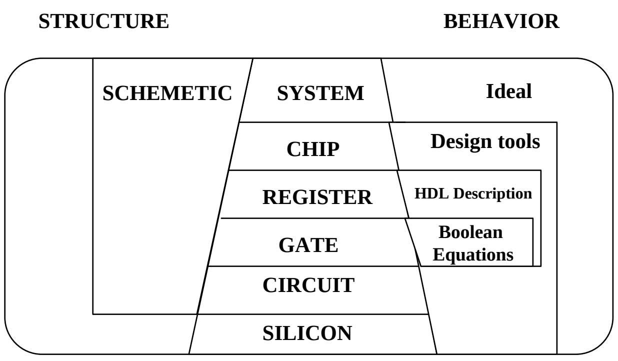
VHDL


PACKAGE DECLARATION
PACKAGE BODY
(often used functions, constants, components, ….)
ENTITY (interface description)
ARCHITECTURE (functionality)
CONFIGURATION (connection entity ↔ architecture)

Structural Description

- Does not cope with functionality of the system, but instead specifies
  - » what components should be used
  - » and how they should be connected to achieve the expected results
- Structural design is much easier to synthesis than behavioral because
  - » it refers to concrete physical components
- However, creating a structural system specification is more difficult because it requires an experienced designer to do it most effectively.

Structure/Behavior

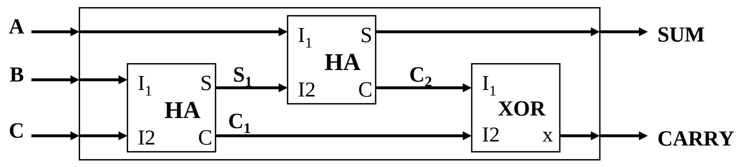


Structural Description

» Example:

```
architecture STRUCTURAL of FULLADDER is
  signal S1, C1, C2 : bit;
  component HA
    port (I1, I2 : in bit; S, C : out bit);
  end component;
  component XOR
    port (I1, I2 : in bit; X : out bit);
  end component;
begin
  INST_HA1 : HA
    port map (I1 => B, I2 => C, S => S1, C => C1);
  INST_HA2 : HA
    port map (I1 => A, I2 => S1, S => SUM, C => C2);
  INST_XOR : XOR
    port map (I1 => C2, I2 => C1, X => CARRY);
end STRUCTURAL;
```

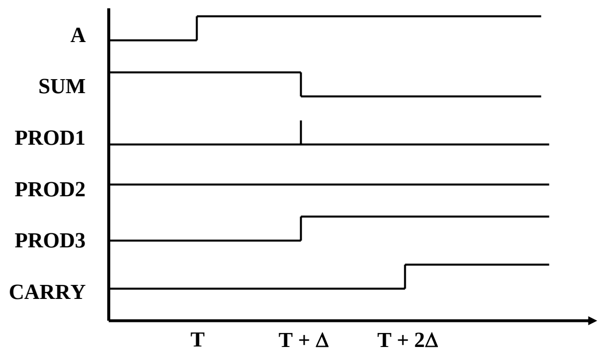
Structure of Full Adder



Concurrent Behavioral Description

```
architecture CONCURRENT of FULLADDER is
   begin
     SUM <= A xor B xor C after 5 ns;
     CARRY <= (A and B) or (B and C) or (A and C)
   after 3 ns;
   end CONCURRENT;
```

Concurrent Behavioral Description



Basic VHDL Constructs

- To specify architectures, or describe behavior, we need some basic VHDL constructs, and some basic VHDL *types*.
- Every piece of info inside a digital system is stored in the form of *bits* and *bit vectors*.
- Instead of long bit-vectors, hex characters may be used.
- Complex but regular data structures can be represented by *arrays*.

An Example of Behavioral Description

```
entity LOW_HIGH is
  port (A,B C: integer;
      MI,  MA: out integer);
  end LOW_HIGH;

architecture BEHAV of LOW_HIGH is

begin
L: process
    variable LOW: integer := 0;
        begin
          wait on A, B, C;
          if A < B    then LOW := A;
                          else LOW :=
    B;
          end if;
          if C < LOW  then LOW := C;
          end if;
          MI <= LOW after 1 ns;
        end process;

H:process
   variable HIGH: integer := 0;
        begin
          wait on A, B, C;
          if A > B    then HIGH := A;
                  else HIGH := B;
          end if;
          if C > HIGH then HIGH := C;
          end if;
          MA <= HIGH after 1 ns;
        end process;
    end BEHAV;
```

•

Quick Overview: Types

- *Scalar type* is a generic name referring to all types whose objects have a single value at any time instant.
- All values of scalar type are ordered and the values are specified either as falling within a range or explicitly enumerated. Examples:
  - » Boolean (T/F),
  - » Character (all characters defined by ISO 8859-1,
  - » Integer (all integers within a specified range),
  - » Real (floating point numbers with a specified range),
  - » Bit (enumeration type specified by '0' or '1').
- We can also have user defined enumeration types, such as
  - » type FSMStates is (Idle, Fetch, Decode, Execute);
- There are also other, such as physical types, user defined records, arrays, etc. (You will see this in a later lecture).

Quick Overview: Expressions/Operators

- Input signals must always be transformed or processed in some way to generate the desired output signals.
  - » Outputs <- transformations(inputs)
- Transformations are performed by expressions (formulae that consist of *operators* with an appropriate number of *operands*)
- Any specification of a systems behavior can be seen as an ordered set of expressions on the inputs assigned to the outputs.
- The basic element of each expression is an operator (*which is assigned a specific type*) and requires one or more operands.
- It is illegal to use an operator on operands of non supported type (except via operator overloading).

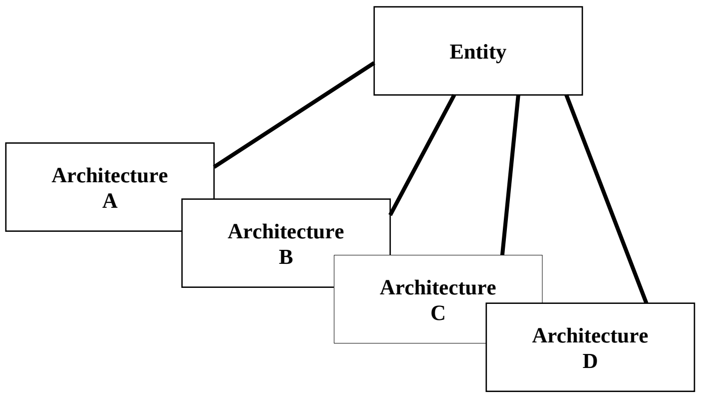
Quick Overview: Other Operators

- Logical Operators (and, or, nand, nor, xor, xnor, not, these are defined for types Bit, Boolean, and Bit_Vector)
- Numerical Operators (+, -, *, /, mod, rem, **, abs)
- Relational Operators (=, /=, <, >, <=, >=) that return T/F
- Shift Operators (sll, srl, sla, sra, rol, ror) restricted to arrays whose elements are of the type Bit or Boolean
- Catenation operator (denoted by '&')
- Assignment operator (assigning expressions to signals
  - » x<=y<=z; (y less or equal to z, to a Boolean signal x)
  - » a<= b or c; (logical operation, correct for Boolean, bit and bit_vector)
  - » k<= '1'; (assignment of a single bit or character, note apostrophes)
  - » m<="0101"; (assignment of bit vector or string)
  - » n<= m & k; (target signal must be declared as 5-bits, why?).

Quick Overview: Other Operators

- Other things you will see later include
  - » Delays
    - – How an assignment can be delayed?
    - – What is propagation/inertial delay?
    - – What is transport delay?
  - » How to declare constants
    - – Use of constants and constants versus generics

One Entity Many Descriptions



One Entity Many Descriptions

- Since different types of architectures can perform the same function, a system (an *entity*) can be specified with different *architectures*. For example:
  - » A number of 80xx51 processors produced by different vendors can be used in place of each other as long as they have the same interface.
  - » This means that one *entity* can be assigned multiple architectures.
- The reverse is not true, since architectures may NOT have different interfaces.

The Concept of Package

- What do you do when you face an unknown concept or word like “subclavian”? Most probably you would turn for some help, either from a person or a dictionary!
- If you are planning to use a dictionary, you could say that you are going to use a *library unit* (i.e., a book)
- Moreover, if you live far away from any library, the book may be delivered to you by mail as a *package*.
- Use these *packages* (external sources of description) when something is undefined in the standard language.

More on the Concept of Package

- A packages is used as a collection of often used
  - » datatypes,
  - » components,
  - » functions, and so on.
- Once these object are declared and defined in a package, they can be used by different VHDL design units.
- In particular, the definition of global information and important shared parameters in complex designs or within a project team is recommended to be done in packages.
- It is possible to split a package into a declaration part and the so-called body.
  - » Advantage is that after changing definitions in the package body only this part has to be recompiled and the rest of the design can be left untouched. Therefore, a lot of time consumed by compiling can be saved.

Using Packages in VHDL

- A similar situation to the one described in the previous slide happens with VHDL specs.
- You may occasionally need to use something not defined in the standard VHDL libraries.
- Packages allow you to define items that are outside the VHDL standards.
- The only restriction in using packages is that they need to be declared in advance, usually at the beginning of an entity.
- There are two clauses, which serve this purpose:
  - » Library and
  - » Use.

Using Packages in VHDL

```
entity SomeSyst is
....
end entity SomeSyst;


architecture FirstARC of SomeSyst is
...
Logarithm
...
end architecture FirstArch;
```

Non Standard Concept

Call a library NewConceptLib; “use concept Logarithm which is defined in package Arithm stored in library NewConceptLib” (read it always from the end).

```
Library NewConceptLib;
Use NewConceptLib.Arithm.Logarithm

entity SomeSyst is
…
end entity SomeSyst;

architecture FirstArch of SomeSyst is
    Logarithm
end architecture FirstArch;
```

Predefined Packages

- VHDL is a robust design environment and has some well defined packages. The most popular packages defined by the IEEE are:
  - » STANDARD – contains basic declarations and definitions of language constructs and it is included in all VHDL specifications by default;
  - » TEXTIO – contains declarations of basic operations on texts;
  - » STD_LOGIC_1164 – this package is not a part of the VHDL standard but is a standard on its own; it contains the most often used language extensions.
- Apart from the three IEEE packages, each VHDL—tool vendor adds (and encourages to use) his/her own package.
- In such cases the library usually bears the vendor's name.

Predefined Packages--Standard

- STANDARD
  - » The package STANDARD is usually integrated directly in the simulation or synthesis program
  - » therefore, it does not exist as a VHDL description.
  - » It contains all basic types: boolean, bit, bit_vector, character, integer, and the like.
  - » Additional logical, comparison and arithmetic operators are defined for these types within the package.
  - » The package STANDARD is a part of the STD library.
  - » Thus, it does not have to be explicitly included by the use statement

Predefined Packages--TEXTIO

- TEXTIO
  - » The package TEXTIO contains procedures and functions which are needed to read from and write to text files.
  - » This package is also a part of the library STD.
  - » It is *not* included in every VHDL description by default.
  - » Therefore, if required, it has to be included by the statement use STD.TEXTIO.all;.

Predefined Packages—STD_LOGIC_1164

- STD_LOGIC_1164
  - » The STD_LOGIC_1164 package has been developed and standardized by the IEEE.
  - » It introduces a special type called std_ulogic which has nine different logic values (shown below).
  - » The reason for this enhancement is that the type bit is not suitable for the precise modeling of digital circuits due to the missing values, such as un-initialized or high impedance.
    - » Declaration:

```
type std_ulogic is (
        '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"
```

More VHDL Examples

- **VHDL** is **NOT** ***CaSe-SeNsItIvE*** **, Thus:**
  Begin = begin = beGiN
- Semicolon " **;** " Terminates Declarations or Statements.
- Line Feeds and Carriage Returns are not Significant in VHDL.
- Lets see some views (or descriptions) of a certain design (ones count)!

Ones Count Interface Specification

```
entity   ONES_CNT   is

port ( A      : in   BIT_VECTOR(2 downto 0);
       C      : out BIT_VECTOR(1 downto 0));

-- Function Documentation of ONES_CNT
--   (Truth  Table  Form)
-- -------------------------------------------------
--  This is a COMMENT
--
--   A2  A1  A0         C1  C0
--    0    0    0          0   0
--    0    0    1          0   1
--    0    1    0          0   1
--    0    1    1          1   0
--    1    0    0          0   1
--    1    0    1          1   0
--    1    1    0          1   0
--    1    1    1          1   1
-- --
end  ONES_CNT;
```

Ones Count Architectural/Behavioral

```
architecture  Algorithmic  of   ONES_CNT  is
begin
Process(A) -- Sensitivity List Contains only Vector A
Variable  num: INTEGER range 0 to 3;
    begin
         num :=0;
         For i    in    0 to 2
         Loop
                  IF A(i) = '1' then
                           num := num+1;
                  end if;
         end Loop;
--       Transfer "num" Variable Value to a SIGNAL
         CASE num is
                  WHEN 0 => C <= "00";
                  WHEN 1 => C <= "01";
                  WHEN 2 => C <= "10";
                  WHEN 3 => C <= "11";
         end CASE;
--
    end process;
end Algorithmic;
```

Ones Count Data_Flow

- C1 = A1 A0 + A2 A0 + A2 A1
- C0 = A2 A1’ A0’ + A2’ A1’ A0 + A2 A1 A0 + A2’ A1 A0’

```
architecture  Two_Level of  ONES_CNT  is
begin

   C(1) <=(A(1) and A(0)) or (A(2) and A(0))or (A(2) and A(1));
--
   C(0) <= (A(2) and not A(1) and not A(0))
         or (not A(2) and not A(1) and A(0))
         or (A(2) and A(1) and A(0))
         or (not A(2) and A(1) and not A(0));

end Two_Level;
```

Architectural/Behavioral (Data Flow) Using Functions

```
architecture  Macro of  ONES_CNT is
begin
   C(1) <= MAJ3(A);
   C(0) <= OPAR3(A);
end Macro ;
```

Functions OPAR3 and MAJ3 Must Have Been Declared and Defined Previously

Architectural   Body (another view)

```
architecture  Truth_Table  of  ONES_CNT  is
begin
--
Process(A) -- Sensitivity List Contains only Vector A
   Variable num: BIT_VECTOR(2 downto 0);
    begin
        num :=A;
        CASE num  is
                    WHEN "000" => C <= "00";
                    WHEN "001" => C <= "01";
                    WHEN "010" => C <= "01";
                    WHEN "011" => C <= "10";
                    WHEN "100" => C <= "01";
                    WHEN "101" => C <= "10";
                    WHEN "110" => C <= "10";
                    WHEN "111" => C <= "11";
        end CASE;
   end process;
end  Truth_Table;
```

Other Examples

- Modeling Adder(s) as a function
- Describing Latches
- Modeling Flip-flops (DFF, edge triggered)
- Describing FSMs
- Structural Descriptions and Components Instantiations
- Etc.

Adder as a Function

```
function "+" (A, B: bit_vector (3 downto 0))   return bit_vector is
  variable SUM: bit_vector (3 downto 0);
  variable CARRY: bit;

begin
  CARRY := '0';
  for I in 0 to 3 loop
    SUM(I)  := A(I) xor B(I) xor CARRY;
    CARRY   := ((A(I) and B(I)) or (A(I) and CARRY) or (B(I) and CARRY));
  end loop;
  return SUM;
end;
```

DFF Edge Triggered

```
entity DFLOP is                        --   D-Type FF
    port (CLK, D: in std_logic;  Q: out std_logic)
 end DFLOP;

architecture BEHAV of DFLOP is
 begin
   process (CLK)
   begin
     if  (CLK = '1') and                  --   CLK = 1
         (CLK'event) and            -- and a new event
         (CLK'last_value = '0')
     --     and previous value was 0 (because of `X` ...)
       then  Q <= D;          -- rising edge
     end if;
   end process;
```

Summary & Discussion

- Superset of ADA
- Concurrent modeling (Blocks for example)
- Generating of instances
- Use of packages, libraries
- Configurations, generics
- Grammar and BNF
- Simulation (and what they will do in the lab)

Summary

- OBJECTIVES: Provide a Standard Medium For
  - » Design Modeling
  - » Design Documentation
- VHDL is an IEEE Standard (Language Standard)
  - » IEEE VHDL-87
  - » IEEE VHDL-93
  - » VHDL-93 is Upward Compatible with the VHDL-87 (Minor Differences May Require Code Modification)
- Requires Simulation and Synthesis Tools
- By The End of 1995, Most Simulation Tools Have Incorporated Support for the VHDL-93
- VHDL Is Also Used For Design Synthesis

Summary

- **Modular**, **Hierarchical**, Allows Design Description (TOP - DOWN, BOTTOM – UP), **Portable,** etc.
- Can Describe the Same Design Entity using More than one View (Domain):
  - » The Behavioral View (e.g. as an algorithm, Register-Transfer (Data Flow), Input-Output Relations, etc)
  - » The Structural View.
- This Allows Investigation of *Design Alternatives* of the Same Entity
- It Also Allows *Delayed Detailed Implementations*.
- Can Model Systems at Various **Levels of Abstraction** (System, chip RTL, Logic (Gate))
- VHDL Can be Made to ***Simulate Timing*** At Reasonable Accuracy.