Hardware Description Languages
VHDL
• Midterm 25%
• Final 35%
• Homework and Quiz 20%
• Project 20%

• Reference:
Chapter 1

Hardware Design Environment
Digital System Design Process

1. Design Idea
2. Behavioral Design → Flow Graph, Pseudo Code
3. Data Path Design → Bus & Register Structure
4. Logic Design → Gate Wirelist, Netlist
5. Physical Design → Transistor List, Layout
6. Manufacturing
7. Chip or Board

benyamin@mehr.sharif.edu
Design Automation

• Design is completed when an idea is transformed into architecture or data path description
• Transforming a design from one form to another
• Verifying a design stage output
• Generating test data
The Art Of Modeling
Hardware Description Languages

• HDLs are used to describe hardware for the purpose of:
  – Modeling
  – Simulation
  – Testing
  – Design
  – Synthesis
  – Documentation
Hardware Description Languages

• Special purpose HDLs:
  – ISPS\textsuperscript{1}, behavioral description HDL
  – AHPL\textsuperscript{2}, data flow description HDL

• General purpose HDLs:
  – Verilog
  – VHDL
  – SystemC

\textsuperscript{1} Instruction Set Processor Specification
\textsuperscript{2} A Hardware Programming Language
Hardware Simulation

- Simulation time
- Output details

Automatic test generation

HDL Model

Simulation Hardware Model

Simulation Engine

Component Library

Test Data

Simulation Results
Simulator Types

- Based on HDL:
  - Behavioral Simulator
  - Data Flow Simulator
  - Gate Level Simulator
  - Device Simulator
- Based on Engine:
  - Oblivious
  - Event Driven
Hardware Synthesis

• A design aid that automatically transforms a design description from one form to another is a synthesis tool.

• Many commercial synthesis tools use the output of the data path design.

• Many commercial synthesis tools have targeted the FPGA market.
Hardware Synthesis

- Synthesis process:
  1. Resource sharing
  2. Logic optimization
  3. Binding

\[
a \leq b + c; \quad c \leq a \text{ AND } e
\]
Chapter 2

VHDL Background
VHDL Features

• General features
  – Describing from system to gate
  – Concurrency
• Support for design hierarchy
  – Operation of system can be specified based on:
    • Its functionality
    • Its smaller subcomponents
• Library support
VHDL Features

• Sequential statements
  – Design based on functionality
• Generic design
  – Some conditions may influence model operation, but it is not necessary to generate a new model
    • Physical characteristics (delay)
    • Environment parameters (temperature)
VHDL Features

• Type declaration (strongly typed language)
  – Integer type
  – Floating point type
  – Enumerate types
  – User defined types
  – Operator overloading
  – Array types
  – Composite-type (records)
VHDL Features

• Use of subprograms (Function, Procedure)
  – Explicit type conversion
  – Logic unit definition
  – Operator redefinition
  – New operation definition

• Structural specification
  – Constructs for specifying structural decomposition of hardware at all levels
VHDL Features

• Timing Control
  – Schedule values to signals
  – Delay the actual assignments until a later time
  – Allow any number of explicitly defined clock
  – Constructs for edge detection
  – Setup and hold time specification
  – Pulse width checking
  – Setting various time constraints
Chapter 3

Design Methodology Based on VHDL
VHDL Elements

• Components
  – Entity
  – Architecture
• Packages
  – Package declaration
  – Package body
• Configuration (binding)
✓ Libraries
Component Description

Component ENTITY

Component ARCHITECTUREs

Logic Function

Interface to the Real World
Component Description

ENTITY ComponentName IS
    input and output ports
    physical and other parameters
END ComponentName;

ARCHITECTURE identifier OF ComponentName IS
    declarations
BEGIN
    specification of model in term of its
    inputs and influenced by physical
    and other parameters
END identifier;

benyamin@mehr.sharif.edu
Multiple Architecture Specification

ENTITY cpu IS PORT(...)

ARCHITECTURE behavioral OF cpu IS

ARCHITECTURE rtl OF cpu IS

ARCHITECTURE structural OF cpu IS

ARCHITECTURE other OF cpu IS
Packages

• Groups components and utilities used for description

```
PACKAGE PackageName IS

    Component Declaration.

    sub-program declarations.

    type definitions

END PackageName;

PACKAGE BODY PackageName IS

    sub-programs.

END PackageName;
```
Configurations

• Binds subcomponents of a design to elements of various libraries

```
CONFIGURATION ConfName OF ComponentName IS
  bindings of Entities and Architectures.
  specifying parameters of a design.
END CONFIGURATION;
```
Libraries

• Groups packages and components to use in another design (reusability)

```
LIBRARY LibName;
USE LibName.SubPackage.Scope

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

LIBRARY WORK;
USE WORK.util_package.int2bit;
```
Top-Down Design

- Is divide-and-conquer method
- Is referred to as **recursive partitioning** until all sub-components become **manageable**

```plaintext
Partition(System)
    IF HardwareMappingOf(System) IS done THEN
        SaveHardwareOf(System)
    ELSE
        FOR EVERY Functionally-Distinct Part_I OF System
            Partition(Part_I)
        END FOR;
    END IF;
END Partition;
```
Top-Down Design

SUD: System Under Design
SSC: System Sub-Component

Design Flow
Implementation

benyamin@mehr.sharif.edu
Verification

• design must be simulated to verify the designer’s understanding of the problem

• This simulation must be done for every SSD
Top-Down Design with VHDL

• Design a 8 bit serial adder with
  – Two data inputs “a” and “b”
  – One input control signal “start”
  – One “Clock” input signal
  – 8 bit “Result” output
  – “Ready” output
Serial Adder – Functionality

Clk

A

B

Result

Ready

\[
\begin{array}{cccccccc}
\text{Clk} & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
\text{A} & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\text{B} & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\text{Result} & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\]

\[11000111 + 00001111 = 11010110\]
Serial Adder – Behavioral Model

If (clk='1' and clk'EVENT) then
  if(start='1') then
    count:=0;
    carry:=0;
  else
    IF count<8 then
      count:=count+1;
      sum:= a XOR b XOR carry;
      Carry:=(a AND b) OR (a AND carry) AND (b AND carry);
      Result<=sum & result(7 downto 1);
    END IF;
  end if;
  IF count=8 then ready<='1';
  else ready<='0'; end if;
end if;
Serial Adder – Top Down Design

- 8 bit Serial Adder
  - FullAdder
  - ShiftRegister
  - 8 bit Counter
  - Flip Flop
Serial Adder – Data Flow Model

Diagram showing the data flow of a serial adder with inputs a and b, carry in and carry out, an adder, a shift register, a counter, a flip-flop, and clock signals.
Design Flow

- 8 bit Serial Adder
  - FullAdder
  - ShiftRegister
  - 8 bit Counter
  - Flip Flop
ENTITY flop IS
    GENERIC(td_reset,td_in: TIME:=8 NS);
    PORT(reset,din,clk: IN BIT; qout: BUFFER:=‘0’);
END flop;
ARCHITECTURE behavioral OF flop IS
BEGIN
    PROCESS(clk)
    BEGIN
        IF(clk='0' AND clk'EVENT) THEN
            IF reset='1' THEN
                qout<='0' AFTER td_reset;
            ELSE
                qout<=din AFTER td_in;
            END IF;
        END IF;
    END PROCESS;
END behavioral;
Design Flow

8 bit Serial Adder

- FullAdder
- ShiftRegister
- 8 bit Counter
- Flip Flop
Full-Adder Description

Entity fulladder IS
    PORT(a,b,cin: IN BIT;sum,count:OUT BIT);
END fulladder;

ARCHITECTURE behavioral OF fulladder IS
BEGIN
    Sum<=a XOR b XOR cin;
    Cout<=(a AND b) OR (a AND cin) OR (b AND cin);
END behavioral;
Design Flow

- 8 bit Serial Adder
  - FullAdder
  - ShiftRegister
  - 8 bit Counter
  - Flip Flop
ENTITY shifter IS
  PORT(sin, reset, enable, clk: IN BIT;
       parout BUFFER BIT_VECTOR(7 DOWNTO 0))
END shifter;
ARCHITECTURE dataflow OF shifter IS
BEGIN
  Sh: BLOCK(clk='0' AND NOT clk'STABLE)
  BEGIN
    parout<= GUARDED
      "00000000" WHEN reset='1'
      ELSE
      sin & parout(7 DOWNTO 1) WHEN enable='1'
      ELSE
      UNAFFECTED;
  END BLOCK;
END dataflow;
Structural Description of Serial Adder

ENTITY serial_adder IS
    PORT(a,b,start,clock: IN BIT; ready: OUT BIT;
    Result : BUFFER BIT_VECTOR(7 DOWNTO 0))
END;

ARCHITECTURE structural OF serial_adder IS
    SIGNAL serial_sum, carry_in, carry_out, counting: BIT;
BEGIN
    u1: ENTITY WORK.fulladder PORT MAP(a, b, carry_in, serial_sum, carry_out);
    u2: ENTITY WORK.flop PORT MAP(start, carry_out, clock, carry_in);
    u3: ENTITY WORK.counter PORT MAP(start, clock, counting);
    u4: ENTITY WORK.shifter PORT MAP(serial_sum, start, counting, clock, result);
    u5: ready <= NOT counting;
END;