In This Semester..

```verilog
module ripplecarry4bit (sum, c_out, a, b, c_in);
output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;
wire c1, c2, c3;
fulladder fa0(sum[0], c1, a[0], b[0], c_in);
fulladder fa1(sum[1], c2, a[1], b[1], c1);
fulladder fa2(sum[2], c3, a[2], b[2], c2);
fulladder fa3(sum[3], c_out, a[3], b[3], c3);
endmodule
```
Systematic Digital IC Design Flow

System Specification
- System Verification
- Behavioral Verification
  - (System-Level Synthesis)
  - (High-Level Synthesis)

Algorithm-Level Behavioral Description
- Behavioral Verification

Register-Transfer Level (RTL) Structural Description
- Logic Verification
- Logic Synthesis

Logic/Transistor Circuit Description
- Layout Verification
- Layout Synthesis

VLSI Mask Layout
System Specification

- System functionality (application)
- Operating environment (IO interface)
- Cost (development, manufacture, test)
- Size/weight (# of chips, board area, box size)
- Power consumption
- Flexibility (specification changes, added functionality)
- Human language (English, Japanese, Thai, etc.)
System Synthesis/Verification

System Specification
- Functional Simulation (SW/HW co-simulation)
- Manual translation
- Human language

Algorithm Description
- Data: types/widths, structures, arrays
- Process: expressions, control-flow, procedures, functions
- Communication: protocols
- Simulation: input stimulus, output verification

Software languages (C/C++, Java)
Hardware languages (Verilog, VHDL)
Algorithm Description

- Functional Simulation
- Software languages (C/C++, Java)
- Hardware languages (Verilog, VHDL)

RTL Structural Description

- manual translation
- (High-Level Synthesis)
- Verilog, VHDL

- Architecture description
  - Module (CPU, memory, register, functional unit, IO interface)
  - Bus architecture
- Module description (functional/structural)
  - Combinational/sequential circuit description
Logic Synthesis/Verification

RTL Structural Description
- Logic Verification
- Timing Verification
- Power analysis

Logic/Transistor Circuit Description
- Logic Minimization
- Technology Mapping

- Cell components (gates, registers, transistors)
- Nets
- IO pins

Verilog, VHDL
- Schematic
- Netlist
Layout Synthesis/Verification

Logic / Transistor Circuit Netlist
- Circuit topology verification
- Design rule check
- Timing Verification

VLSI Mask Layout
- Cell / module layout (manual or auto)
- Place and Route

- Verilog, VHDL
- Schematic
- Netlist

- Mask Pattern

- Layers (well, diffusion, polysilicon, metals, vias)
- Rectangle, polygons
Electronic design automation (EDA) is the category of tools for designing and producing electronic systems ranging from printed circuit boards (PCBs) to integrated circuits. This is sometimes referred to as ECAD (electronic computer-aided design) or just CAD. Among of the well-known EDA companies are Aldec. Inc., Cadence, Mentor Graphics, Silvaco and Synopsis, Inc. (more details in http://en.wikipedia.org/wiki/List_of_EDA_companies)
What is HDL?

HDL – Hardware Description Language

- Used to describe the **behavioral aspects** of a circuit function
- Used to describe the **logic functionality** of a circuit
- Sometimes used to show the **netlist** of a circuit
Types of HDL (widely used)
- **Verilog**
- **VHDL** (VHSIC HDL – Very High Speed Integrated Circuit Hardware Description Language)

Higher Level Design Abstraction Languages
- SystemVerilog
- SystemC

Hybrid Languages
- Supports mixed-signal circuit design
- Verilog-A, Verilog-AMS
Always an argument on which is a better form of HDL
Both has its advantages and disadvantages
Whichever is more suitable to be used as the standard HDL depends largely on individual designer
Most EDA/CAD design tools in the market can handle both Verilog & VHDL
- **Verilog**
  - Easy to write
  - Easy to read & understand as it is similar to C (syntax based on C)
  - Easier to learn compared to VHDL
  - Have switch-level modeling
  - More compact code
VHDL

- It is more complicated & more difficult to learn compared to Verilog (syntax based on ADA)
- More coding rules to follow
- More flexible compared to Verilog
- Can reflect real design more efficiently
Graphical Difference between Verilog and VHDL
HDL-based design is now a dominant design paradigm used by industry. EDA tools will act as a companion of HDL which it verifies the design’s functionality, optimize it and create appropriate netlist according to physical technology. HDL and EDA put much focus on functionality rather than individual transistors or gates.
EDA verifies the functionality and checked whether it satisfy the design constraints (such as area, speed, power, performance).

The portability and technology independency of HDL-based design makes it suitable for use with current EDAs. One can modify, reuse and improve designs anywhere, anytime.
HDL-based Design Flow

- Using appropriate EDA, any HDL-based design can be optimized accordingly to the designer’s target of implementation, whether **ASIC** or **FPGA**.
- Both of these targets do have a similar design steps, in such as **design**, **verify**, **synthesis**, and **test** of digital circuits.
- **ASIC-targeted design** – complex design flow (architecture of ASIC is not fixed)
- **FPGA-targeted design** – simpler design flow (design which passes the post-synthesis timing requirements could be downloaded into the fixed architecture FPGA)
Question
Where does HDL play its role?
Importance of HDL

- Designs can be described at very abstract level using HDL
  - can write without sticking to any technology
- Functional verification can be done early in the design cycle
  - can optimize & modify RTL description until meet desired functionality
- HDL design is analogous to computer programming
  - provide concise representation of design compared to schematic
Designing Dilemma..
EMT 353/3 DIGITAL IC DESIGN

INTRODUCTION TO EDA & HDL

EXTRA
Example: Design a 2-1 MUX

Specifications
- Module name: MUX
- Input pins: A, B, Sel
- Output pins: OutAB
- Function

- If Sel is ‘0’, choose A and pass it on to Out
- If Sel is ‘1’, choose B and pass it on to Out
- The value of A and B does not matter
  - Pass both 0’s and 1’s
Example: Design a 2-1 MUX

- Create a Truth Table

```
<table>
<thead>
<tr>
<th>Sel</th>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Characteristic Table

```
<table>
<thead>
<tr>
<th>Sel</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
```
**Boolean Equation Table**

\[ \text{Out}_{AB} = \overline{\text{Sel}} \cdot \text{A} + \text{Sel} \cdot \text{B} \]
The previous example just for a simple hardware design, what if designing a processor?

Use hardware Description language (HDL)

- have special hardware related constructs.
- can be used to build models for simulation, synthesis and test.
- have been extended to the system design level
module multiplexer (A, B, Sel, Out);
  input  A, B, Sel;
  output Out;
  wire   tmp1, tmp0;

  and  (tmp1, B, Sel);
  not  (selbar, Sel);
  and  (tmp0, A, selbar);
  or   (out, tmp0, tmp1);

endmodule
Example 2: 4-1 MUX (using logic equations)

```verilog
module mux4x1 (d3, d2, d1, d0, sel, out);
    input d3, d2, d1, d0;
    input [1:0] sel;
    output out;

    assign out = d0 & (~sel[1] & ~sel[0]) |
                 d1 & (~sel[1] & sel[0]) |
                 d2 & ( sel[1] & ~sel[0]) |
                 d3 & ( sel[1] & sel[0]);

endmodule
```
Example 2: 4-1 MUX (using If-Then-Else)

module mux4x1 (d3, d2, d1, d0, sel, out);
  input  d3, d2, d1, d0;
  input  [1:0] sel;
  output out;
  reg    out;

  always @ (d3 or d2 or d1 or d0 or sel)
  begin
    if (sel == 2'b00)
      out = d0;
    else if (sel == 2'b01)
      out = d1;
    else if (sel == 2'b10)
      out = d2;
    else if (sel == 2'b11)
      out = d3;
    else
      out = 1'b0;
  end
endmodule
Example 2: 4-1 MUX (using case)

```verilog
module mux4x1 (d3, d2, d1, d0, sel, out);
  input  d3, d2, d1, d0;
  input  [1:0] sel;
  output out;
  reg     out;

  always @ (d3 or d2 or d1 or d0 or sel)
  begin
    case (sel)
      0 : out = d0;
      1 : out = d1;
      2 : out = d2;
      3 : out = d3;
      default : out = 1'b0;
    endcase
  end
endmodule
```
Many ways to Code

module mux4x1 (d3, d2, d1, d0, sel, out);
  input  d3, d2, d1, d0;
  input  [1:0] sel;
  output out;
  reg  out;

  assign out = d0 & (~sel[1] & ~sel[0]) |
             d1 & (~sel[1] & sel[0]) |
             d2 & ( sel[1] & ~sel[0]) |
             d3 & ( sel[1] & sel[0]);
endmodule

module mux4x1 (d3, d2, d1, d0, sel, out);
  input  d3, d2, d1, d0;
  input  [1:0] sel;
  output out;
  reg  out;

  always @(d3 or d2 or d1 or d0 or sel)
  begin
    if (sel == 2'b00)
      out = d0;
    else if (sel == 2'b01)
      out = d1;
    if (sel == 2'b10)
      out = d2;
    else if (sel == 2'b11)
      out = d3;
    else
      out = 1'b0;
  end
endmodule

module mux4x1 (d3, d2, d1, d0, sel, out);
  input  d3, d2, d1, d0;
  input  [1:0] sel;
  output out;
  reg  out;

  always @(d3 or d2 or d1 or d0 or sel)
  begin
    case (sel)
      0 : out = d0;
      1 : out = d1;
      2 : out = d2;
      3 : out = d3;
      default : out = 1'b0;
    endcase
  end
endmodule
LET’S TRY THIS… 1-bit Full Adder

module Add_full(sum, c_out, a, b, c_in);
input a, b, c_in;
output sum, c_out;

assign sum = a ^ b ^ c_in;
assign c_out = (a & b) | (a & c_in) | (b & c_in);
endmodule
LAB 0: Design code

```verilog
module Lab0 (A, B, C, D, E, F);
    input A, B, C, D, E;
    output reg F;
    always @(A or B or C or D or E)
    begin
        F = A & B & C & D & E & F;
    end
endmodule
```

# Compile of Lab0.v failed with 2 errors.

```
vlog -work work C:/altera/91/quartus/EMI353/LabIntro/Lab0.v
Model Technology ModelSim PE Student Edition vlog 10.2c Compiler 2013.07 Jul 18 2013
-- Compiling module Lab0
** Error: C:/altera/91/quartus/EMI353/LabIntro/Lab0.v(2): near "": syntax error, unexpected '!', expecting ';'
** Error: C:/altera/91/quartus/EMI353/LabIntro/Lab0.v(7): (vlog-2730) Undefined variable: 'F'.
** Error: C:/altera/91/quartus/EMI353/LabIntro/Lab0.v(1): Identifier must be declared with a port mode: F.
```
LAB 0 (cont’d), spot the errors

```verilog
module Lab0 (A,B,C,D,E,F);
  input A,B,C,D,E;
  output reg F;

  always @(A or B or C or D or E)
    begin
      F = A & B & C & D & E;
    end
endmodule
```
module Lab0_tb();
reg A, B, C, D, E;
wire F;
integer i;
initial
begin
    for (i=0;i<32;i=i+1)
    begin
        {A,B,C,D,E} = i;
        #10;
    end
end
Lab0 Lab0_inst(A,B,C,D,E,F);
endmodule
Exercise 2.1

1) Design an equivalent Verilog HDL code
2) Design an equivalent Test bench code
3) Sketch the expected waveform showing all input and output involved
Exercise 2.2

1) Design an equivalent Verilog HDL code
2) Design an equivalent Test bench code
3) Sketch the expected waveform showing all input and output involved