

# ECE 337: ASIC Design

## Lab 1

*Week of 8/19/2013*



# If you missed Lecture on 8/20

- 9:30-10:20 AM, T & Th in EE 117
  - look at your detailed schedule in MyPurdue
- Look at lecture notes and materials as listed in course calendar



# Course Programs/Tools

- Blackboard (Only using the new Blackboard Learn)
  - Contains course material
  - Grades
  - Discussion Forums (Q&A and Course Updates)
  - Critical Updates will be also sent through Blackboard email
- Main software program used
  - Mentor Graphics Modelsim – For writing and testing Verilog



# Course Format Overview

- First several weeks are designed to learn the tools
- One project during the course
  - Groups of three or four
- Course Objectives
  - Read them on your own
  - Pass them by finishing labs, doing the final project, and answering certain test questions
  - You have opportunities to re-attempt objectives




# Lab Work Rules

- Lab exercises & Postlabs due at the beginning of following lab
- Must satisfactorily complete all lab exercises to pass
  - 90% of points on Labs 2,3, 7
  - 50% of automated tests on Labs 4-6
- Lab remediation(s)
  - Will only affect objectives for tool labs (2,3,7)
  - Labs 4-6: Can get up to 50% of automated grader points
- Allowed 1 late tool Lab at a 10% grade penalty



# Course Sessions Format Overview

- Labs:
  - Start with a quick meeting in conference area
  - Attendance is not kept during regular labs
  - Must attend for project briefings during the final project phase
- Lectures:
  - Explains concepts needed for lab
  - Unannounced quizzes, in-class exercises, exams
  - Grade is dependent on clicker use.



# Academic (dis)Honesty

- Automated and manual systems are in place to check for plagiarism/cheating.
- Working off someone else's code is unacceptable
- Work is on an individual basis
  - No sharing of code unless explicitly allowed by course staff
- You WILL be caught



# Grades

- Exams are 25%
- Labs are 35%
  - Labs 4-6 are weighted higher
- Final project is 35%
- Homework/Clicker participation is 5%
- +/- system





# Lab Rules

- Food and drinks only allowed in conference area
- Do not leave computers logged in or locked
  - Users with locked terminals will have all sessions killed
- Treat this room like the library
- Sign-up sheet is used for assistance/sign-offs
  - Usually done with whiteboard in front lab area



# Lab Access

- POTR 360
- MSEE 189
- EE 206/207
- Office hours
  - TA Hours:
    - Monday @ 6:30-9:30pm in POTR 360
    - Tuesday @ 5:30-8:30pm in POTR 360
    - Wednesday @ 6:30-8:30pm in POTR 360
    - Thursday @ 6:30-8:30pm in POTR 360
  - Prof Hours: See schedule on blackboard

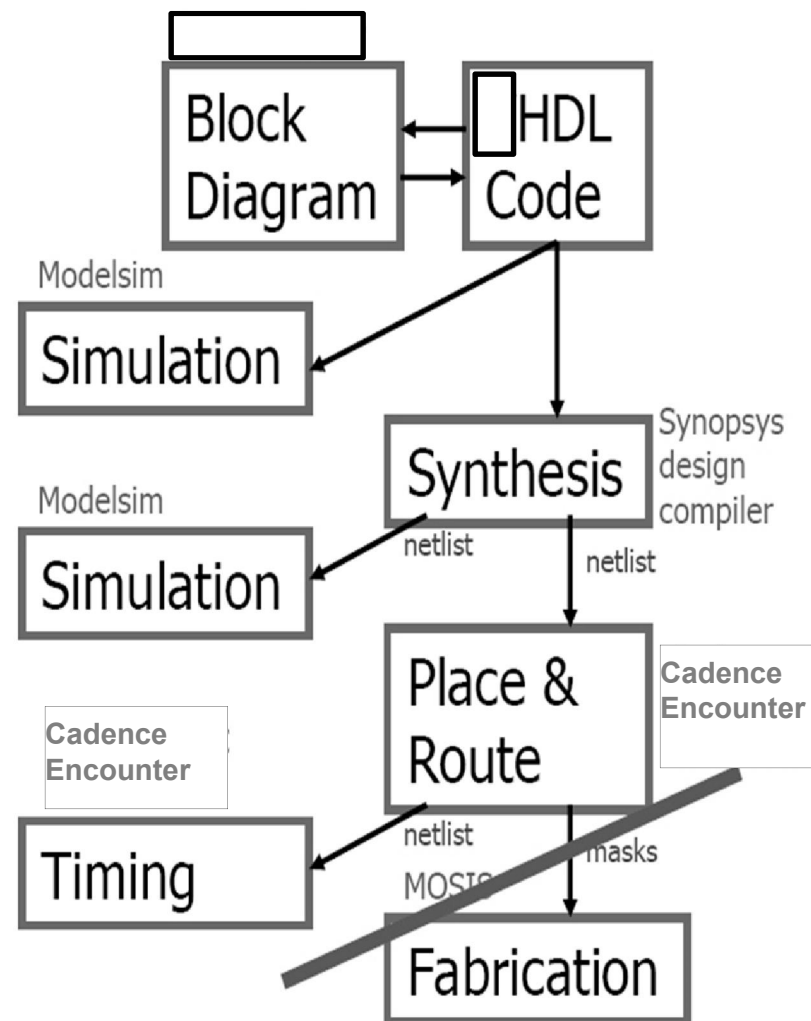


# First Lab Notes

- Modelsim, Leda, Design Compiler, SVN
- CHANGE YOUR PASSWORD
- Lab 1 WILL NOT be graded if you do not do Postlab
- There are plenty of online Verilog resources in addition to the required text book

# ASIC Design

- Just because your code works doesn't mean it's translatable to a chip!
- Think in terms of logic gates when you write your code





# Crash Course in the Basics of Verilog

- For details see lecture reading assignments
- This is going to just cover enough the basics



# Big Picture of Verilog

- Verilog is a hardware description language
  - Verilog is not a programming language
  - It is used to describe the behavior of hardware system(s)
  - Create a picture of what the hardware should be like before you start describing it with Verilog code
  - Hardware systems execute concurrently not sequentially
  - Modularity is key!
    - Helps with design work (allows a lot of reuse of prior work)
    - Helps with debugging (much easier and quicker to analyze)
    - Helps with verification (allows simpler and quicker component level testing)



# Verilog Basics – Main Code Constructs


- Nets:
  - Represent a wire in a system
- Constant:
  - Define a constant value for use in code
- Port:
  - Represent a physical port to a hardware component
- Parameters:
  - Constant(s) that's value(s) can be set during the inclusion of the component in another system



# Verilog Basics – Main Data Types

- Wire:
  - Represents the possible values of a wire
- Reg:
  - Similar to wire but can also be used as a ‘variable’ and controlled by behavioral blocks
  - No direct translation to hardware, used to describe behavior
- Integer (subtype of reg):
  - A signed integer (usually up to a 32-bit unless restricted)
- Time (subtype of reg):
  - An unsigned 64-bit integer
- Real (subtype of reg):
  - Double precision floating point





# Verilog Basics – Coding styles

- Dataflow:
  - Every line is a concurrent statement/piece of hardware
  - Similar to how you designed PLD's in ECE 270
- Behavioral:
  - Composed of “procedural blocks”
  - All blocks execute simultaneously with each other
  - Procedural blocks are interpreted sequentially starting at the top
    - However no time passes between consecutive lines
    - Can only set value of regs not wires
    - Values are assigned to regs at the end of the block or during a blocking assignment
    - Think of them as defining a precedence rule set for what a signal's value should be at any point in time

# Core Structure - Module

- Defines the name of the design block
- Defines the inputs and outputs of the design block
  - Known as the port map
- Contains the functional code for the design block

## Verilog Module Syntax:

```
module <design name> is
(
  <input/output/inout> <data type> <name>,
  <input/output/inout> <data type> <name>
);
  [Internal declarations]
  <functional code>
endmodule
```

# Sample Module Portmap Declaration

```
// XOR Gate
```

```
// Implemented using NAND and OR
```

```
module xorgate  
(  
    input wire A,  
    input wire B,  
    output wire X  
);
```

# Sample Module Body Declaration

- Contains declarations for any internal nets, variables, parameters, functions, and tasks
- Defines the behavior/functionality of the design block

```
<module portmap>  
  [parameter declarations]  
  [net declarations]  
  [variable declarations]  
  [function declarations]  
  [task declarations]  
  
  <dataflow and/or behavioral code>  
endmodule
```

# Sample Behavioral Implementation

```
<module portmap>
```

```
  reg s_NAND;
```

```
  reg s_OR;
```

```
  reg x_tmp;
```

```
  assign X = x_tmp;
```

```
  always @ (A, B) begin
```

```
    if (!(A & B)) begin
```

```
      s_NAND = 1'b1;
```

```
    end else begin
```

```
      s_NAND = 1'b0;
```

```
    end
```

```
    if (A | B) begin
```

```
      s_OR = 1'b1;
```

```
    end else begin
```

```
      s_OR = 1'b0;
```

```
    end
```

```
    if (s_NAND & s_OR) begin
```

```
      x_tmp = 1'b1;
```

```
    end else begin
```

```
      x_tmp = 1'b0;
```

```
    end
```

```
  end // end for the always
```

```
endmodule
```

# Sample Dataflow Architecture

```
<module portmap>
```

```
    wire s_NAND;
```

```
    wire s_OR;
```

```
    assign s_NAND = ~(A && B);
```

```
    assign s_OR    = A || B;
```

```
    assign X       = s_NAND && s_OR;
```

```
endmodule
```

## Sample 2 Module Portmap Declaration

```
// Lawmaking Machine  
// 4 voters and one president  
// 75%: law passes  
// or 50% + president: law passes
```

```
module lawmaker  
(  
    input wire [3:0] voter,  
    input wire prez,  
    output wire result  
);
```

# Sample 2 Dataflow Implementation

```
wire s_maj;
wire s_ovr;

assign result = s_ovr || (s_maj && prez);

assign s_maj = (voter[3] && (voter[2] || voter[1] || voter[0])) ||
               (voter[2] && (voter[1] || voter[0])) ||
               (voter[1] && voter[0]);

assign s_ovr = (voter == 4'b1111) ? 1'b1:
               ((voter == 4'b1110) ? 1'b1:
                ((voter == 4'b1101) ? 1'b1:
                 ((voter == 4'b1011) ? 1'b1:
                  ((voter == 4'b0111) ? 1'b1: 1'b0)))));

endmodule
```



# Sample 2 Behavioral Implementation

```
<module portmap>
  reg [2:0] vote_cnt;
  reg [2:0] i;
  reg result_tmp;

  assign result = result_tmp;
  always @ (voter, prez) begin
    vote_cnt = 0;
    for(i = 0; i <= 3; i = i + 1) begin
      if (voter[i] == 1'b1) begin
        vote_cnt = vote_cnt + 1;
      end
    end
    if (vote_cnt >= 2) & (prez == 1'b1) begin
      result_tmp = 1'b1;
    end else if (vote_cnt >= 3) begin
      result_tmp = 1'b1;
    end else begin
      result_tmp = 1'b0;
    end
  end // end for always
endmodule
```

