Topics

“If you study to remember, you will forget, but, if you study to understand, you will remember.”
– Unknown

- Review, review, review
  - Ask questions if you have them
  - Combinational and sequential circuits
  - Microarchitecture and ISA
  - Verification and testing

- Today’s 50-minute review is NOT comprehensive

Finite-Precision Numbers

- Computers have fixed number of digits to represent values (creates set of valid values)

- Computation can lead to “errors”
  - Overflow: result is larger than largest number in set
  - Underflow: result is smaller than smallest number in set

- Computation could result in a number that cannot be represented (non-integer result on integer machine)

Integer Representation

- Magnitude: $b_{n-1} b_{n-2} \ldots b_1 b_0$
- MSB
- Sign: 0 denotes +, 1 denotes -
- (a) Unsigned number
- (b) Signed number

Negative Binary Numbers

- **Signed magnitude**
  - Leftmost bit is sign, remaining bits are magnitude
  - Two representations for zero

- **One’s complement (obsolete)**
  - Invert all bits
  - Two representations for zero

- **Two’s complement**
  - Invert all bits and add “1”
  - Only one representation for zero

IEEE Floating Point Standard 754

- Provided designers with a correct model

- Allowed FP data to be exchanged among different computer systems

- Defines three formats
  - Single precision (32 bits)
  - Double precision (64 bits)
  - Extended precision (80 bits)
    - Only occurs within FP units

Floating Point Numbers

- **IEEE Numerical Types**
  - Underflow handled gracefully
    - Use denormalized numbers instead of jumping to zero
  - Overflow becomes infinity
  - Includes Not a Number (NaN)
Structured Computer Organization

- Digital logic builds microarchitecture
- Microarchitecture implements the ISA
- ISA is in machine language
- Assembly language allows us to use ISA

VHDL Design Entity

- **Interface**
  - Connections to the system
- **Behavior**
  - Type of processing on input signals and type of output signals produced

Logic Circuit Types

- **Combinational**
  - Output is a function of inputs ONLY
  - e.g. briefcase lock
- **Sequential**
  - Output is a function of inputs and previous state (memory)
  - e.g. vending machine

Multiplexers/Demultiplexers

- **MUX**
  - $2^n$ data inputs, 1 data output, n control signals
  - Binary code on select determines which input is routed to output
- **DEMUX**
  - 1 data input, $2^n$ data outputs, n control signals
  - Binary code on select routes a single input signal to one of $2^n$ outputs
Decoders/Encoders

- **Decoder**
  - Takes an n-bit number as input
  - Selects (sets to “1”) exactly 1 of $2^n$ outputs

- **Encoder**
  - Input is a group of parallel bits
  - Output is the binary code assigned to asserted input

Comparators and Shifters

- **Comparator**
  - Determine if two input words are equal
  - Based upon XOR gate

- **Shifter**
  - Arithmetic shift maintains the sign
    - 1-bit arithmetic shift left multiplies by 2
    - 1-bit arithmetic shift right divides by 2
  - Logical shift fills empty bits with “0”
    - Shift left logical 8 from the Mic-1

Adders

- **Half adder**
  - Inputs A, B
  - Outputs $C_{out}$, Sum

- **Full adder**
  - Inputs A, B, $C_{in}$
  - Outputs $C_{out}$, Sum

  - Ripple-carry adder
  - Carry-select adder

Types of Storage Elements

- **Latch**
  - Transparent when internal memory is being set
  - Level-triggered

- **Flip-flop**
  - Not transparent, reading and changing output are separate
  - Edge-triggered
Types of Storage Elements

- **D-type (data)**
  - Q output is determined by the D input at the clocking event
- **T-type (toggle)**
  - Toggles its state at input event
- **SR-type (set/reset)**
  - Set or reset by inputs (S=R=1 not allowed)
- **JK-type**
  - Allows both J and K to be 1, otherwise similar to SR

Clocks

- Circuit that emits a series of pulses
  - Precise pulse width
  - Precise interval between consecutive pulses
    - Clock cycle time – time interval between corresponding edges of two consecutive pulses
- Events can be triggered by
  - Clock high
  - Clock low
  - Rising edge
  - Falling edge

Sequential Machines

- Use registers to make primary output values depend on state plus primary inputs
- Types of Finite State Machines (FSM)
  - In a **Mealy** machine, outputs are a function of the present state and primary inputs
  - In a **Moore** machine, outputs depend only on present state

Generating Clocks and Periodic Waveforms for Simulation

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY periodic IS
  PORT (Z : OUT STD_LOGIC);
END periodic;

ARCHITECTURE Behavior OF periodic IS
BEGIN
  PROCESS
  BEGIN
    Z <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 40 ns;
    WAIT FOR 50 ns;
  END PROCESS;
END Behavior;
```

All processes are executed when a VHDL model is initialized

- Uses the WAIT FOR construct to reactivate the process

Introductory VHDL: Chapter 6 Copyright © 2001 Prentice Hall PTR

FPGA-Based System Design: Chapter 5 Copyright © 2004 Prentice Hall PTR
Example: Mealy State Machine

- Output \( z \) equals ‘1’ if during any two consecutive clock cycles the input \( w \) was equal to ‘1’
  - Restated, \( z \) equals ‘1’ in the same clock cycle that detects the second consecutive occurrence of \( w \) equal to ‘1’

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>( t_0 )</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
<th>( t_6 )</th>
<th>( t_7 )</th>
<th>( t_8 )</th>
<th>( t_9 )</th>
<th>( t_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w ):</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( z ):</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example: Mealy State Machine

State table

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state ( w = 0 )</th>
<th>Next state ( w = 1 )</th>
<th>Output ( z ) ( w = 0 )</th>
<th>Output ( z ) ( w = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>B</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Example: Moore State Machine

- Output \( z \) equals ‘1’ if during two immediately preceding clock cycles the input \( w \) was equal to ‘1’
  - Restated, \( z \) equals ‘1’ in the clock cycle that follows the detection of the second occurrence of \( w \) equal to ‘1’

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>( t_0 )</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
<th>( t_6 )</th>
<th>( t_7 )</th>
<th>( t_8 )</th>
<th>( t_9 )</th>
<th>( t_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w ):</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( z ):</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Moore State Machine

**State transition graph**

```
Reset
w = 0
A / z = 0
w = 1
B / z = 0
w = 0
w = 0
w = 1
C / z = 1
w = 1
```

**Example: Moore State Machine**

**State table**

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>w = 0</td>
<td>w = 1</td>
<td>z</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

**Fundamentals of Digital Logic: Chapter 8 Copyright © 2005 McGraw-Hill**

Moore State Machine Using a Single Process

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY simple IS
  PORT ( Clock, Resetn, w : IN STD_LOGIC ;
        z : OUT STD_LOGIC ) ;
END simple ;
ARCHITECTURE Behavior OF simple IS
  TYPE State_type IS (A, B, C) ;
  SIGNAL y : State_type ;
  BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
      IF Resetn = '0' THEN
        y <= A ;
      ELSIF (Clock'EVENT AND Clock = '1') THEN
        CASE y IS
        WHEN A =>
          IF w = '0' THEN
            y <= A ;
          ELSE
            y <= B ;
          END IF ;
        WHEN B =>
          IF w = '0' THEN
            y <= A ;
          ELSE
            y <= C ;
          END IF ;
        WHEN C =>
          IF w = '0' THEN
            y <= A ;
          ELSE
            y <= C ;
          END IF ;
        END CASE ;
      END IF ;
      z <= '1' WHEN y = C ELSE '0' ;
    END PROCESS ;
  END Behavior ;
```

**FPGA-Based System Design: Chapter 5 Copyright © 2004 Prentice Hall PTR**

**FSM Structure**

```
primary inputs

combinational
logic

primary outputs

memory elements

clock
```

**Fundamentals of Digital Logic: Chapter 8 Copyright © 2005 McGraw-Hill**
Moore State Machine Using Two Processes

ARCHITECTURE Behavior OF simple IS
TYPE State_type IS (A, B, C);
SIGNAL y_present, y_next : State_type;
BEGIN
PROCESS (w, y_present)
BEGIN
  CASE y_present IS
    WHEN A =>
      IF w = '0' THEN
        y_next <= A;
      ELSE
        y_next <= B;
      END IF;
    WHEN B =>
      IF w = '0' THEN
        y_next <= A;
      ELSE
        y_next <= C;
      END IF;
    WHEN C =>
      IF w = '0' THEN
        y_next <= A;
      ELSE
        y_next <= C;
      END IF;
  END CASE;
END PROCESS;
END Behavior;

One-Hot Encoding

- State machine with \( N \) states has \( N \)-bit encoding
- The \( i \)th bit is 1 if machine is in state \( i \)

Comparison
- Easy to tell what state the machine is in
- Easy to get the machine into an illegal state (0000, 1111, etc.)
- Uses a lot of registers

State Assignment

- Find a binary code for symbolic values in machine
  - Optimize area, performance
  - May be done on inputs, outputs as well

The Big Picture

- Since 1946 (ENIAC I) all computers have had 5 components

Adapted from John Kubiatowicz’s CS 152 lecture notes. Copyright © 2003 UCB.
Instruction Set Architecture Level

- Interface between software and hardware
- Both compilers and hardware must understand ISA
  - Compilers translate high-level language into object code
  - Hardware must directly execute or interpret ISA

Basic Instruction Cycle

- Fetch
  - Get the next instruction from memory
- Decode
  - Determine which instruction to perform
- Execute
  - Perform the instruction

74381 ALU

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>C, t, f</td>
<td>A, B, C</td>
<td>F</td>
</tr>
</tbody>
</table>

Basic ISA Classes

- Library: ieee
- USE ieee.std_logic_1164.all;
- USE ieee.std_logic_unsigned.all;
- ENTITY alu IS
  PORT ( s : IN STD_LOGIC_VECTOR(2 DOWNTO 0) ;
         A, B : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
         F : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END alu ;
- ARCHITECTURE Behavior OF alu IS
  BEGIN
    PROCESS ( s, A, B )
    BEGIN
      CASE s IS
        WHEN "000" =>
          F <= "0000" ;
        WHEN "001" =>
          F <= B - A ;
        WHEN "010" =>
          F <= A - B ;
        WHEN "011" =>
          F <= A + B ;
        WHEN "100" =>
          F <= A XOR B ;
        WHEN "101" =>
          F <= A OR B ;
        WHEN "110" =>
          F <= A AND B ;
        WHEN OTHERS =>
          F <= "1111" ;
      END CASE ;
    END PROCESS ;
  END Behavior ;
Definitions

- **Design synthesis**
  - Given an I/O function, develop a procedure to manufacture a device using known materials and processes

- **Verification**
  - Predictive analysis to ensure that the synthesized design, when manufactured, will perform the given I/O function

- **Test**
  - A manufacturing step that ensures that the physical device, manufactured from the synthesized design, has no manufacturing defect

Verification vs. Test

**Verification**
- Verifies correctness of design
- Performed by simulation, hardware emulation, or formal methods
- Performed once prior to manufacturing
- Responsible for quality of design

**Test**
- Verifies correctness of manufactured hardware
- Two-part process:
  - 1. Test generation: software process executed once during design
  - 2. Test application: electrical tests applied to hardware
- Test application performed on every manufactured device
- Responsible for quality of devices

The Concept of a Fault

- Testing centers around detection of faults in a circuit
- The digital world is made up of interconnected gates
  - Thus only two things can fail, gates and their interconnections
- A faulty gate fails to perform its function correctly
- A faulty interconnection produces
  - “stuck at 1,” “stuck at 0,” or permanently tri-stated input

**Problem:** can you set up experiments that produce one result if the gate/interconnection is good and another if it is bad?
- *This leads to two closely related concepts*
Observability and Controllability

- **Observability** – the ability to observe a gate output directly at external pins
- **Controllability** – the ability to apply any and all desired inputs to a gate via external pins
- To test a gate completely, all combinations of inputs must be applied and the output corresponding to each input must be observed

Functional Verification

- **Main purpose**
  - Ensure that a design implements intended functionality
  - Can show that a design meets intent of the specification but cannot prove it
- **Approaches**
  - **Black-Box**: performed without any knowledge of the actual implementation of a design; can only use available interfaces
  - **White-Box**: performed with full visibility and controllability of the internal structure/implementation of a design
  - **Grey-Box**: access similar to black-box but can exercise significant features of the design
- **What are pros/cons for each approach?**

Testbenches

- **A testbench is a model used to exercise a simulation**
  - Completely closed system
  - Provides stimulus
  - Checks outputs
- **Testbenches help automate design verification**
  - Rerun edited module against testbench
  - Run models (behavioral, RTL) against the same testbench
Static Timing Analysis

- After your gate netlist has been mapped to the FPGA, a timing analysis tool will analyze the paths in the design and compute the timings.

- The timing analyzer takes into account the routing delays in the physical routing and the speed grade of the part you have mapped to.

- Because routing can sometimes change somewhat drastically for even small changes, often run multiple device mappings to try to get a ‘good route’.

Static Timing Analysis (STA)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Required</th>
<th>Slack</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{\text{max}}$</td>
<td>361.1 MHz</td>
<td>200 MHz</td>
<td>1.17 ns</td>
<td>AddSub</td>
<td>Overflow</td>
</tr>
<tr>
<td>$t_{\text{su}}$</td>
<td>2.536 ns</td>
<td>10.0 ns</td>
<td>7.644 ns</td>
<td>$b_0$</td>
<td>$b_{\text{reg0}}$</td>
</tr>
<tr>
<td>$t_{\text{co}}$</td>
<td>6.772 ns</td>
<td>10.0 ns</td>
<td>3.228 ns</td>
<td>$c_{\text{reg}}$</td>
<td>$z_0$</td>
</tr>
<tr>
<td>$t_h$</td>
<td>0.240 ns</td>
<td>10.0 ns</td>
<td>9.76 ns</td>
<td>$b_1$</td>
<td>$b_{\text{reg1}}$</td>
</tr>
</tbody>
</table>

- $f_{\text{max}}$ – maximum operating frequency of clock
- $t_{\text{su}}$ – worst-case setup requirement
- $t_{\text{co}}$ – worst-case clock-to-output delay
- $t_h$ – maximum hold time

- Slack is the difference between requirement and result
  - Positive slack means constraint was satisfied
  - Negative slack means constraint was not satisfied

Flip Flop Based System Performance Analysis

- Clock Period

$$t_{\text{su}} + t_{\text{co}} + t_{\text{clk2q}} + t_{\text{rise}}$$

Speeding Up the Clock

- The register-to-register delay is usually the delay path that sets the maximum clock rate.

- From a design point of view, can only affect the combinational logic between the registers
  - Need to shorten the maximum combinational delay path
  - Setup/Hold time of registers are fixed

- Can shorten the delay by placing a register in the combinational logic to break longest delay path
  - This technique is called pipelining
  - Adds latency to the output (the number of clocks between an input value and its corresponding output result)
Mismatched Latency

- LPM_Mult can be used to pipeline the multiplier
- Inserts states automatically in the multiplier

Matched Latency

- Correct latency by adding registers in other paths
- Also need to pipeline control lines

Summary

- FPGA Design is still a GREAT class
- Dr. Robinson is still a GREAT professor
- Questions focus on material since Exam 1
  - Don’t forget concepts from earlier chapters
- Main ideas
  - Combinational and sequential circuits
  - Microarchitecture and ISA
  - Verification and testing
- Study hard and good luck!