Today (1)

- Administrative Info
- Lab #4 Solution
- Lab #5 Tips
- Project Overview
- Design Review Requirements

Today (2)

- CaLinx2 Expansion
- Checkpoint #1
- Button Parse
- N64 Controller
  - Commands
  - Bit Timing
  - Receive
  - Transmit
  - Testing

Administrative Info (1)

- Midterm I
  - Solutions will be posted
- Group Sign-ups
  - You should already have a group
  - If not, see me after class today
  - Group names need to be finalized today
Administrative Info (2)
- **N64 Controller Checkout**
  - One per group
  - Test it out as soon as you get it
  - Use the Checkpoint#1 demo posted online
- **Damage**
  - You break it, you buy it
  - We don’t have replacements, sorry
- **Return**
  - Due back by the final
  - Otherwise we don’t grade your final

Administrative Info (3)
- **What are Man Hours?**
  - \( \text{MH} = \sum_{i=1}^{n} H_i \)
  - \( H_i \) = # hours spent by person \( i \), \( n \) = # people
- **Example:**
  - David and I are project partners.
  - David spent 11 hours on CP#2, and I spent 8 hours on CP#2. 3 of each of our hours were spent working together.
  - \( H_1 = 11 \), \( H_2 = 8 \), \( n = 2 \); \( \text{MH}_{CP#2} = 11 + 8 = 19 \)
  - Last semester: average \( \text{MH}_{CP#1} = 30 \)

Lab #4 Solution (1)
```verilog
module Lab4Part2Tester(A, B, Sum, ...);
output [15:0] A, B, Sum;
input [1:0] FailMode;
input Clock, Reset, Go;
output Running, Error;
wire Done;
assign Error = Sum != (A+B);
Lab4Part2Adder TheAdder(.A(A), .B(B), .Sum(Sum), .FailMode(FailMode));
Counter TestCounter( .Clock(Clock), .Reset(Reset), .Set(1'b0),
.Load(1'b0), .Enable((Running & ~Error) | Go),
.In(32'hXXXXXXX), .Count((A, B)));
defparam TestCounter.width = 32;
Register RunReg( .Clock(Clock), .Reset(Reset | &{A, B}), .Set(Go),
.Enable(1'b0), .In(1'bX), .Out(Running));
defparam RunReg.width = 1;
```

Lab #4 Solution (2)
- **FSM Debugging**
  - Both FSMs were “almost Moore”
  - You’ll build a lot of these
  - Check for this CAREFULLY in your code
Lab #5 Tips

- Reading Logic Analyzer
  - Look at the falling edges of Clock
  - This avoids clock-to-output delay
  - Easier to read
  - CurrentState, Input, Output on THIS CYCLE
  - NextState is the NEXT “CurrentState”

- Using ChipScope
  - Use different trigger ports for different signals
  - Use “Position” to see before triggering event

EECS 150 Spring 2006 Class Project

Project Overview (1)

- Wireless Tron
  - N64 Controller -> Player Control
  - NTSC (TV) Video Output -> Battle Field Display
  - Chipcon RF Transceiver -> Wireless Communication
  - Game Engine -> Coordinate everything

Project Overview (2)

Block Diagram of the Design
Project Overview (3)

- Four check points
  - N64 Input
  - NTSC Output
  - Chipcon Transceiver Bidirectional communication
  - Game Engine
    - Absolutely symmetric (anyone can play with anyone, with another copy of yourself)
    - Interface with Chipcon, N64 and NTSC

Project Overview (4)

- Checkpoints
  - Require more design work than labs
    - We're not telling you exactly what to do
  - Part of your project
    - Design them well
    - Test modules thoroughly!
    - Don't lose your code
    - Require more time
  - See project schedule posted online for deadlines and grade weights

Design Review Requirements

- High level schematic of your module implementation
- Basic building blocks:
  - register, shift register, counter, mux, comparator, adder, state machine
- You can create your own module hierarchy
- Due at the beginning of your assigned lab the week after the lab lecture
- Graded on correctness, efficiency

Design Review Example
Calinx2 Expansion

Checkpoint #1: N64 (1)
- Primary input for your project
  - Direction Buttons: Move your car
  - Analog Joystick: Can also move your car
- Other Buttons:
  - Start new game after game over
  - Speed up / brake?
  - Camera angles / zooming?
  - Weapons?
  - Game codes?

Checkpoint #1: N64 (2)
- I/O Controller
  - Precise timing requirements
    - You have to match a real piece of hardware
    - You can't change both ends of the problem
  - Hard to simulate
    - You may want to build a model controller
    - We would give you one, but it's basically the solution with minor changes
      - Hint hint...

Button Parse (1)
- Real World Inputs
  - Problem
    - Circuits are fast (27MHz)
    - Humans are slow (100Hz)
    - Mechanical switches are very imperfect
  - Solution
    - Take very long (10ms), very bouncy pulses
    - Clean them up
    - Generate one cycle pulses (1'b0, 1'b1, 1'b0)
N64 Controller (1)

- **Functionality**
  - Fancy Parallel <-> Serial Conversion
    - Single bi-directional wire to N64 Controller
  - Core of this module is a Shift Register
- **Polling**
  - N64 uses Challenge/Response
  - We send a command, it responds
  - Might need a little state (send/receive)

N64 Controller (2)

- **Construction**
  - **Components**
    - Shift Register
    - FSM/State Register
    - What else?
  - Plan your design out in DETAIL

N64 Commands (1)

- **Available Commands**
  - **8'hFF: Reset Controller**
  - **8'h00: Get Status**
  - **8'h01: Read Buttons**
  - **8'h02: Read Mempack**
  - **8'h03: Write Mempack**
  - **8'h04: Read EEPROM**
  - **8'h05: Write EEPROM**

N64 Commands (2)

- **Command Format**
  - 8bits of Command
  - 1-bit Stop Bit (1'b1)
  - Example:
    - Read Buttons: 9'b000000001
  - How do we generate this?
    - Shift Register
    - Something better…
N64 Bit Timing (1)

Data
Stop
Start
1'b0
Stop
Start
1'b1
4us/Bit

N64 Bit Timing (2)

"Read Buttons" Command
- 8bits of Command: 8'h01
- 1bit Stop Marker: 1'b1

N64 Receive (1)

Points to Consider
- Reliability
  - The N64 controllers are NOT 100% reliable
  - Your circuit must be
- Timing
  - This is not a synchronous design
    - The controller has its own clock
    - We must detect new bits
  - A major part of this design is timing!

N64 Receive (2)

Detecting a new bit:
- Look for 1'b1 (Stop sub-bit) -> 1'b0 (Start sub-bit)
  - Falling Edge Detector
- Wait 2us
- Capture Data

Rising Edge Detector

In
Register

Clock

Register

Register

Out
N64 Receive (3)

- What if we never get another bit...
  - Detection
    - Falling edge never happens
    - How do we detect an event that never happens?
  - Recovery
    - Can't wait forever, game will lock up
    - Re-request the button status? Reset the controller?
    - You MUST recover from this condition!

N64 Transmit (1)

- For Each Bit
  - Count 1us (1 sub-bit), Transmit 1'b0
  - Count 1us (1 sub-bit), Transmit Data
  - Count 1us (1 sub-bit), Transmit Data
  - Count 1us (1 sub-bit), Transmit 1'b1

- Timing
  - We're driving the bus – not a problem
  - Still very important

N64 Transmit (2)

- Bidirectional Open Collector Bus

  Endpoint A
  - 10kΩ Pullup
  - DIn
  - Enable
  - DOut

  Endpoint B
  - 10kΩ Pullup
  - DIn
  - Enable
  - DOut

  Driver Logic
  - DIn
  - Enable
  - DOut

  Virtex Tristate
  - 1.5V

N64 Testing

- Required
  - Your circuit must work on board reliably
  - Look at our demo...

- Recommended Testing
  - Build a testbench model of the controller
  - Symmetric Protocol
    - Bits out look like bits in
    - Just a different number of bits
  - Modify your solution...
Reminders

- Sign-up your group
- Check out a N64 controller and test it
- Design review for CP#1 due in lab next week (along with Lab#5)
- FPGA Tron demo after lab lecture
- “Real” Tron demo at 4pm
  - Get some ideas for extra credit