FPGA Design
EECE 277

Verification and Testing Part 2

Dr. William H. Robinson
March 21, 2005

http://eecs.vanderbilt.edu/courses/eece277/

Topics

“Don't find fault, find a remedy.”  
– Henry Ford

• Administrative stuff
   – Read Chapter 11 in your textbook
   – Laboratory Assignment #3 (due Monday, March 28)
   – Initial guidelines for final project
   – No class on Friday, March 25

• JTAG

• Developing testbenches

Laboratory Assignment #3

• Quartus II licenses
   – Renew your license if it expires

• Flex 10K70
   – Contains the equivalent of 70,000 gates

• There is no spoon...
   – Nor is there a space between clock and ‘EVENT

• Timing analysis tool
   – Can use the longest delay path to calculate

• Area/speed optimization
   – Device technology limits speed

Laboratory Assignment #3

• State machine
   – Sample code on the Rapid Prototyping CD

• ALU
   – Support for shift operators not in std_logic_1164

• Compiling the Simple µP
   – Change address control in memory module to “registered”

• In-class lab session
   – Wednesday, March 23, 9am – 11am
Final Project

General requirements
• Consists of designing and implementing a microelectronic system application that can be demonstrated using the Altera UP-2 prototyping board
• Do not modify your UP2 prototyping board
• Application should involve interfacing with inputs (push-buttons, toggle switches, connectors) and outputs (LEDs, 7-segment displays, connectors)
• It should include some arithmetic computation (datapath) and a state machine (control) as well as a built-in self-test module

Ideas for Final Project
http://microsys6.engr.utk.edu/ece/bouldin_courses/551/candidates.html
• 01. Differential Scoreboard
• 02. Elevator Controller
• 03. Encryption/Compression
• 04. Decompression/Decryption
• 05. Hamming Code Transceiver
• 06. Digital Frequency Meter
• 07. Blackjack Game
• 08. Music Generator with 4-Part Harmony
• 09. Race Track Timer
• 10. Tic-Tac-Toe Game
• 11. String Comparator
• 12. Simon Game
• 13. Other (submit proposal for approval)

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
Real Tests

- Based on analyzable fault models, which may not map on real defects
- Incomplete coverage of modeled faults due to high complexity
- Some good chips are rejected
  - The fraction (or percentage) of such chips is called the yield loss
- Some bad chips pass tests
  - The fraction (or percentage) of bad chips among all passing chips is called the defect level

Definitions

Examples on why precise definitions are so important for reliability

- Is a programming mistake a fault, error, or failure?
  - Are we talking about the time it was designed or the time the program is run?
  - If the running program doesn’t exercise the mistake, is it still a fault/error/failure?

- If an alpha particle hits a DRAM memory cell, is it a fault/error/failure if it doesn’t change the value?
  - Is it a fault/error/failure if the memory doesn’t access the changed bit?
  - Did a fault/error/failure still occur if the memory had error correction and delivered the corrected value to the CPU?

IFIP Standard Terminology

- Computer system dependability: quality of delivered service such that reliance can be placed on service
- Service is observed actual behavior as perceived by other system(s) interacting with this system’s users
- Each module has ideal specified behavior, where service specification is agreed description of expected behavior
- A system failure occurs when the actual behavior deviates from the specified behavior
- Failure occurred because an error, a defect in module
- The cause of an error is a fault
- When a fault occurs it creates a latent error, which becomes effective when it is activated
- When error actually affects the delivered service, a failure occurs (time from error to failure is error latency)

Fault vs. (Latent) Error vs. Failure

- A fault creates one or more latent errors
- Properties of errors are
  - A latent error becomes effective once activated
  - An error may cycle between its latent and effective states
  - An effective error often propagates from one component to another, thereby creating new errors
- Effective error is either a formerly-latent error in that component or it propagated from another error
- A component failure occurs when the error affects the delivered service
- These properties are recursive, and apply to any component in the system
- An error is manifestation in the system of a fault, a failure is manifestation on the service of an error
Fault vs. (Latent) Error vs. Failure

- An **error** is manifestation *in the system* of a fault, a failure is manifestation *on the service of an error*
- **Is a programming mistake a fault, error, or failure?**
  - Are we talking about the time it was designed or the time the program is run?
  - If the running program doesn’t exercise the mistake, is it still a fault/error/failure?
- A programming mistake is a **fault**
- The consequence is an **error (or latent error)** in the software
- Upon activation, the error becomes **effective**
- When this effective error produces erroneous data which affect the delivered service, a **failure** occurs

Fault vs. (Latent) Error vs. Failure

- An **error** is manifestation *in the system* of a fault, a failure is manifestation *on the service of an error*
- **If an alpha particle hits a DRAM memory cell, is it a fault/error/failure if it doesn’t change the value?**
  - Is it a fault/error/failure if the memory doesn’t access the changed bit?
  - Did a fault/error/failure still occur if the memory had error correction and delivered the corrected value to the CPU?
- An alpha particle hitting a DRAM can be a **fault**
- If it changes the memory, it creates an **error**
- Error remains **latent** until effected memory word is read
- If the effected word error affects the delivered service, a **failure** occurs

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

Automated Test Vector Generation

- **Software exists to analyze a digital design and perform the following functions:**
  - Answer the question “What percentage of gates have been completely tested by the test vectors applied to the device?” This is known as **Fault Coverage**
  - Generate test vectors to cover the remaining gates
- **Fault Coverage curves look like:**

![Fault Coverage Curve](image)
Issues in Test Vector Generation

• Issue #1: Diminishing Returns
  – Vectors start covering smaller and smaller fractions of the device

• Issues #2: Some tests may be nearly impossible or truly impossible (non-testable faults)

• Without extra hardware added explicitly to help with testing, fault coverage is typically in the 90 to 95% range

Fault Coverage

• Consider study results that show defect level vs. fault coverage

  ![Graph showing defect level vs. fault coverage](image)

  - To get a single chip defect rate of 100 defective parts per million parts, you need 99.9% fault coverage! Do you really need a defect rate of less than 100 ppm?

  Source: 1980 study by Delco and Motorola, as cited in TI's IEEE 1149.1 Testability Primer

The Effects of Single Chip Defects

• Consider the effects of having multiple chips on a board

  ![Graph showing ASIC ppm Rate vs. Goall PCB ppm Rate](image)

  - If you have 5 ASIC’s on a board, you need to have around 60 ppm defective ASIC’s to achieve 300 ppm defective boards.

What is the Meaning?

• You need fault coverage in the 99% to 99.9% range

• You need automated test vector generation and fault grading tools
  – Fault grading tells you % fault coverage for a test vector suite

• You need extra hardware to improve testability
  – Controllability and observability
**Using A Scan Path**

- Scan path ties all flip-flops together into a shift register
- Using “scan mode” you can preload a particular state (test vector)
- Using “normal mode” you then operate the design for one cycle
- Then you scan out the result while at the same time load the next test vector

**Introducing IEEE 1149.1 (JTAG)**

- In order to standardize scan path design and support, the Joint Test Action Group, comprised of over 200 electronics companies, developed an industry-wide standard for scan path support
- It has been adopted as IEEE Standard 1149.1
- It is usually called “the JTAG port” in sales literature (or Boundary Scan)
- Every JTAG compatible port can be chained together to form a serial device
- Can be used to support fault detection at the single chip level, detection of board level faults, and even some PLDs/FPGAs can be programmed with it

**Basics of JTAG**

- The JTAG standard adds four signals to every compatible device
  - TCK: a clock signal for the test port
  - TMS: Test Mode Select - determines whether or not the scan path is active and controls the mode it is in
  - TDI: Test Data In - serial data input
  - TDO: Test Data Out - serial data output
- The standard also specifies some standard operating modes
  - For example, the serial shift register can be set to bypass the chip, making access to other chips faster.
- Vendor specific commands can be added

**JTAG Architecture**

![JTAG Architecture Diagram]
Keys to an Effective Test Strategy

• **Decide early, not late, on test strategy**
  – Other test approaches beside BIST, JTAG

• Identify software tools for test vector generation, fault grading

• Have to plan test strategy through development, prototyping, manufacturing, and maintenance

• Adding a JTAG bus to a board (and JTAG compatible components)
  – Can be a cost effective way of adding a large amount of testability to your design

Formal Verification

• **Equivalence checking**
  – Compares two netlists to ensure that some netlist post-processing (e.g. scan-chain insertion, clock-tree synthesis, manual modification) did not change circuit functionality

• **Model checking**
  – Assertions or characteristics of a design are formally proven or disproved (e.g. state machines checked for unreachable or isolated states)

Verification vs. Testing

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?**
Functional Verification

- **Stimulus**
- **Design Under Verification (DUV)**
- **Response**

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

Verification Strategy

- **Specification**
  - Formal document that describes the functionality

- **Features**
  - Enumerated characteristics within the design that must be verified independent of implementation

- **Test cases**
  - Grouping of related features (e.g., configuration, granularity, verification strategy)

- **Testbenches**
  - Grouping of test cases
  - Test bench SHOULD produce error messages

Example

- **Pepsi Machine**
  - You are to design a controller for a Pepsi machine where each drink costs 20 cents. The machine accepts nickels and dimes, but does not give change. The output of a coin unit indicates which coin is received. Slugs are not detected, but a coin that places the total over 20 cents is rejected.

- **Features?**
- **Test cases?**
- **Testbench?**
Summary

• Usually have to add extra hardware to improve testability

• Scan paths can increase chip and board level testability

• JTAG adds four pins and supports a variety of test modes; almost all electronics vendors support JTAG

• Verification determines if a design meets the desired functionality