Testing

Overview

In this lesson we will

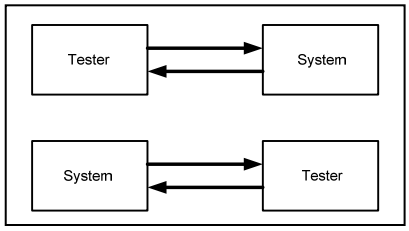
- ✓ Begin with an overview of the test process.
- ✓ Introduce the basic test terminology.
- ✓ Identify the reasons for testing.
- ✓ Identify the basic necessary documentation.
- ✓ Examine testing for yourself and for your customer.
- ✓ Identify contemporary test problems and techniques.
- ✓ Analyze scan design and boundary scan.
- ✓ Briefly cover design for testability and self-test.

Introduction

We are typically used to thinking of

System and tester

Using first model in accompanying figure



First we design

The system then

We design the tester

Then we use the tester to test the system

Consider reversing the roles to that reflected in second model

The system is supplying signals to the tester

The tester is supplying signals to the system

There is really little difference between the two

In fact it's almost like a quiz game

Tester supplies signal

System answers

Question now arises: was the answer correct

If yes

Proceed down one path

If no

Down which path to proceed

The associated process

Fault isolation

Debugging

As we've learned when we design the system

Begin at high level of abstraction
With functional view
Refine that view
In increasing levels of detail
Until we have sufficient information
To implement the design

When we design the tester
We utilize much same thought and approach process
As the design of the system

Prior to launching into design of tester
Let's think about what we are trying to do
Reflect on what our goals are
Look at the test process

A Walk Through the Test Process – An Overview

Testing can begin during any stage of design process
However goals are different at each stage

Begin with coarse grained view of test
Focus for the moment on test on the design side

Stages of Test

- During early stages
  - Testing for ourselves
    - Key point at this stage
      - *We have a design that has never worked*
    - We are trying to verify basic
      - Architecture
      - Functionality

  - Begin with design that is incomplete

Over time

- Design and model larger and larger portions (modules) of the system
  - ✓ Confirm that each modeled module or collection of modules
    Meets required high-level specifications
    May have to model missing physical functionality
    Still concerned about real world constraints
    For low-level internal timing
    Asynchronous interfaces with outside world
    Becoming increasingly common

    Initial focus on
    Basic functionality
    Portions of intermodule exchange

- As design of individual modules and basic integration progresses
  - ✓ Focus shifts to low-level details
    At the module level

- System level concerns defrayed until later
  Not time for system level low-level details

➢ During middle stages
  - ✓ Testing focus remains primarily directed towards
    Evolving yet incomplete design

Presumably by the middle phases
Major and minor specification issues have been resolved
Larger subsystems being integrated
Subsystem level functionality being confirmed

- During later stages
  - ✓ Testing focus remains primarily directed towards
    - Proving system level aspects of design

  Presumably by the later phases
    - Design issues have been resolved
    - Larger subsystems being integrated
      - System level functionality being confirmed
        - Includes meeting constraints
          - Temporal
          - Behavioural

Major difference between
  Design of system and design of test of the system

- Design of system
  - Should be top down process
    - Focus first on high level then on details
- Design of test of system
  - Should be bottom up
    - Focus first on details then on high level

A Simple Strategy

Approach to testing at all levels
  For any test

1. Always start from known state
   Module, subsystem, system
     - Must be initialized
       - Does not imply everything set to 0

2. Know what is to be tested
   Test in design more stringent than test in production
     - Design test
       - The design is unproven
         - Want to test ideas or approaches
         - Verify theory in practice
       - Ultimate goal is to confirm design
         - ✓ Performs as expected
         - ✓ Meets specifications

Production test
The design is known to be good
Ultimate goal is to identify
- ✓ Bad parts
- ✓ Problems arising from manufacturing

3. Know how to test the entity
   To ensure that it meets requirements
   Test with
   Nominal values
   Boundary condition values
   Inside and outside of boundary
4. Know why we are testing
   What you are looking for
   This is important

5. Know what results to expect
6. Understand the results
   If they are correct – what do they tell you
   If they are incorrect – what do they tell you

Let's now explore job of testing
In bit more detail
We'll begin with some important vocabulary

Test Terminology

UUT / DUT
Unit or device under test
Test Suite
Series of tests applied to a UUT
May be applied at any level
Open or short
Signal line designed to be connected to a circuit element or other signal line
Not connected
Signal line designed not to be connected to a circuit element or other signal line
Connected
Stuck at condition
Signal line held permanently in fixed state
Bridge condition
Several signal paths are unintentionally connected together

Race Condition
- Critical Race
  - State or output of circuit depends upon
    - Order in which several associated inputs change
- Non critical Race
  - State or output of circuit does not depend upon
    - Order in which several associated inputs change

Glitch
- Unintended output pulse of short duration

Hazard
- Circuit has possibility of producing short pulse
  - When transient behaviour of circuit may produce glitch

Golden Unit
- Unit whose behaviour is completely known
- Used as a standard

Test Vector / Pattern
- Test Vector
  - Collection of 0's and 1's
    - Applied to unit input
    - To verify piece of functionality
    - Produce specified output
- Test Pattern
  - Sequence of test vectors
  - Applied to a UUT
  - To test behaviour over time

Manufacturing Defects and Manufacturing Defects Test
- Unintended conditions introduced into a normally working circuit or system
  - Result of manufacturing process
- Test to determine if unintended conditions introduced during manufacturing

Bare Board and Bare Board Test
- Board with no components installed
- Test to ensure that circuit board manufactured properly

Functional test
- Test to ensure that circuit or system meets functional specifications

Test Coverage
- Percentage of circuit or system tested in
  - Specific test or series of tests

Debug

Process of identifying faults in new or modified circuit or system

Fault Isolation

Process of identifying faults in circuit or system

Generally applies to process of identifying source of fault

In known good design of circuit or system

Indirectly / Directly Observable

Identifies if signal behaviour can be seen (tested)

Directly at card edge or test point

Test Point / Connector

Signal or set of signals

Internal to UUT

That can be observed directly via a probe point or connector

Added specifically for test

Reasons to Test

There are four main reasons to test

- To verify that any of the following performs as we had intended
  - ✓ Code module or hardware prototype
  - ✓ Subsystem or collection of subsystems
  - ✓ System

  Early stages

  Emphasis on debugging

  Later stages

  Emphasis on error free build
- To ensure the system meets specification
- To ensure that any changes to system

  Work

  Do not alter other intended functionality
- To ensure the system functions properly after being built

Each of these tests

Is different

Has different objective and scope

Tests different things

Testing - The Documentation

As with design of system

Must begin with plan

Cannot throw bunch of signals at

Hardware

Software

Say looks good to me

Well you can

Can't say much about the results though

Documentation

Documentation for test based upon documentation for design

Relationship

Design	Test
Requirements Spec	Test Plan
Design Spec	Test Spec
Design	Test Cases

System Test Plan

Describes in general terms

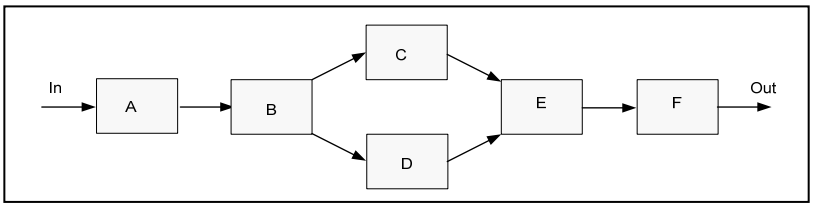
- *What* must be tested

  Based upon initial requirements specification

  Process similar to identifying original requirements

- *How* test will be carried out

- Testing order within each type of test



Can't test output of module E

Unless outputs of modules C and D verified

Similarly moving to left

- Assumptions made
- Algorithms that may be used

System Test Specification

- Like design specification test specification
  - Formalizes test plan
- Assigns specific values and limits
  - Parameters to be tested
- Must include
  - Description of and specification for
    - Each set of tests

System Test Procedure

- Test procedure
  - Made up of set of test cases
  - Each test case
    - Gives detailed steps of specific test

Testing formality increases towards latter phases of development

- Testing to ensure design functionality
  - Can be reasonably informal
    - Once again typically debugging phase
  - Should still have plan
- As system begins to come together
  - Formality must increase
- Systems becoming too complex
  - Too easy to miss critical yet subtle point

Testing for Yourself

Let’s look first at testing during the early phase

Analysis

- During early portion of design
  - Must identify
    - Risks
    - Potential failure modes

Egoless Design

- Early phase of testing begins with specification
- Among first steps
  - Design reviews
  - Code walk-throughs
  - Code inspection
- With movement to executing designs

- Utilizing models and HDLs
- Methods taken from software design
  - Now become appropriate

- Work and critical review during these stages
  - Help to ensure solid foundation
  - Simpler tasks as design progresses

- Future work
  - Execution of specification
  - Some tools exist now

Debugging

- This is the phase we call debugging
- Code is written
  - HDL and software
- Prototype available
- Now must ‘turn it on’ as expression goes
- To effectively debug we must know
  - What we are looking for
  - How we are going to produce the appropriate stimui
  - How to analyze results

First Steps

- Never wait until a module or subsystem completely
  - Built
  - Coded
  - Synthesized
- Never complete entire system then try to debug

Test Case Design

- Test case design is essential for testing at any level
- Content of test cases will vary
  - Nature and intent of test

Early Stages

Test Values

During early stages of test

- ✓ Initially when debugging we use
  - Expected values for inputs
    - Important to know
      - What input values are
      - What to expect as result
    - Simply trying to determine if module
      - Works as expected

- ✓ Later when trying to confirm robustness we use
  - Unexpected values
  - Boundaries of expected values
    - Inside
    - Outside
    - At
  - Such values
    - May be random or statistically based patterns
      - Reasonable for Combinational logic
      - Fall down on sequential

Test Coverage

All aspects of design must be tested

At coarse grained level

- Must ensure hw and sw functionality
  - As specified in requirements and design specifications

At fine grained level

- For hardware
  - Must ensure every device input and output checked
    - At least once
  - Each path through system verified
- For software
  - Must ensure every line of code executed
    - At least once
  - Each path through code executed

Module Test

- Three kinds of module testing
- Based upon assumed knowledge of system internals
  - Black box
  - Gray box
  - White box
- Apply to hardware or software

Black Box Test

- Black box tests are data driven
- Module tested from external point of view
  - Assumes no knowledge of system or subsystem internals
- Test cases generated and applied
- Test failure aborts test and fault identified and fixed
- Testing resumes from beginning
- Black box testing requires module interfaces
  - Be clearly defined
- Weaknesses
  - Potentially exhaustive test
    - Very time consuming
  - May miss
    - Certain paths
    - Dead code

White Box Testing

- White box tests are logic driven
- Module tested from internal point of view
  - Assumes perfect knowledge of system or subsystem internals
- Test cases generated and applied
  - Designed to exercise every internal path and code segment
- Test failure aborts test and fault identified and fixed
- Testing resumes from beginning
- White box testing requires module interfaces
  - Be clearly defined

Gray Box Testing

- Mix of white and black box testing
- Applies when we have modules we did not design
  - Complex LSI or gate arrays
  - Library modules

Subsystem and System Test

Subsystem Test

- Make each submodule maximally independent
- Have clean well defined interface
- Design and debug submodules individually
- Keep modules simple
  - Not more than 1-2 abstract and related functions

- Debugging
  - Understand module behaviour
  - Make sure module state is initialized
  - Select as simple an input or set of inputs as possible
    - For each input know what the output should
    - Apply simple set of inputs
      - Hard code if necessary
  - When several behaviours possible

- Test simplest first
- Once individual components tested
  - They need to be integrated and tested in larger subsystems
    - Until system comes together
- Many of techniques used at module level still apply
- Several other things considered at this level
  - One interesting approach called *fault seeding*

- *Fault Seeding*
  - Intentionally plants number of faults into system
- Testing proceeds as normal
- Count number of seeded faults identified
- Assume test cannot distinguish between seeded and nonseeded faults
  - If x% of seeded faults remain then x% of nonseeded remain as well

Regression Test

- Testing at system level should result in regression test suite

- Regression tests
  - Used later to ensure changes to system
    - Don’t unintentionally alter behavior
  - Updated
    - Changes discovered during
      - Alpha
      - Beta
      - Verification / Validation testing
    - To reflect system changes as it evolves

- Upon completion of testing at this level
  - Focus of testing shifts from design to production

Moving to Production

- Testing formality increases towards latter phases of development
  - Testing to ensure design functionality
    - Can be reasonably informal
    - Should still have plan
  - As system begins to come together
    - Formality must increase
  - Systems becoming too complex
    - Too easy to miss critical yet subtle point

Testing for your Customer

- Testing for customer begins at the specification stage of design
- Distinguish between
  - Production tests and ongoing testing for product support

Initial Thoughts

- If it works on the bench – it should work in production

Decisions and Trade-offs

- Why?
  - What do we have to think about

Approach

Testing at this stage

3 pronged attack

Alpha and Beta tests

Real world experience

Verification Test

Ensure we work and meet specs

Validation Test

Ensure the verification test does what it says it will

Pre-production

Alpha and Beta Tests

Intent of these test is to get real world experience on system

Either given to

Select customers

Internal users

Goal is to apply product as it is expected to be used

Alpha tests

Occur shortly after system has completed

Comprehensive internal test suite

Beta tests

Follow incorporation of fixes discovered during alpha test

Both alpha and beta tests series may be repeated number of times

Verification

Verification testing is designed to prove product

Meets specification

Is not as comprehensive as some of earlier system testing

The efficacy of this test suite is only as good as the specifications

May want to develop reduced regression suite for verification tests

Validation

Intent of validation testing

Prove test suite is

Testing what its supposed to test

Make required measurements

Within required tolerance

It can catch faults

Executed
Prior to releasing tests to production
Whenever product or tests modified

Acceptance Test

The acceptance test suite is set of tests customer uses
When accepting product
May include any or all of verification and validation tests
During production
May be randomly applied to ensure quality standards

Production Test

Assumes system design is correct and meets specs
Is not developed to verify integrity of design

Verification Tests

May be subset of regression tests
Goal two fold
Test system least amount to ensure quality system
Meets spec and will not be dead on arrival
Test as quickly as possible
Production testing does not add anything to product
It is a cost
If one could guarantee
Quality
Parts
Production
Production testing could go away

Contemporary Test Problems and Techniques

Introduction

Today we are becoming victims of our own successes
- Hoist with our own petard as the expression goes

As we've discussed contemporary designs are getting
- Smaller
- Faster
- Increasingly dense and complex

On single die we are able to implement
- Multiple traditional CPU cores
- DSP core
- Substantial amounts of memory
- Sophisticated interconnection schemes

While contemporary packages can support several hundred pins
- We do not have immediate access to internal
  - Signals
  - Busses
  - Processor registers

Without such access
- How can we hope to begin effective test of everything

We now want to look at several techniques that will give us such access
It is important to understand and remember
- These techniques are not magic
- Simply using them will not ensure that test will be trivial

Associated with the use of such techniques
- Is design process called *design for test*

Simply based upon name
- State that we must think ahead during design process
  - What additional circuitry we will have to add to design
    - To support test process
- Not something that we can add later once design is completed

In next several sections

Will address

- ✓ Scan design techniques
  - As basis for developing boundary scan methodology
- ✓ Boundary scan techniques
  - As powerful approach for testing contemporary complex circuitry
- ✓ Design for test
  - Considerations taken when executing design to support test
- ✓ Built in self test
  - Techniques to aid in improving reliability and safety of contemporary designs

Scan Design Techniques

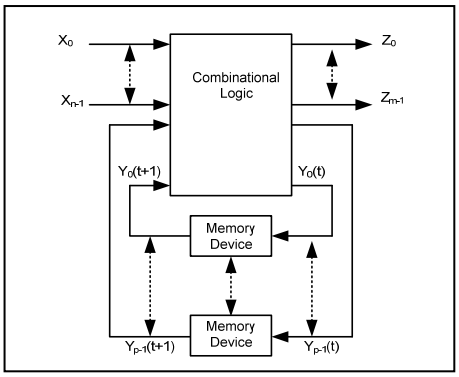
One of most challenging blocks of integrated circuitry to test

Finite state machines

We'll begin with those

Methods we learn here will have broad application to combinational as well

Will form basis for boundary scan techniques developed next



We learned earlier that a finite state machine

Can be decomposed into a

Combination logic block

Flip-flop block

If each piece can be tested separately

Task will certainly be a lot easier

Combinational portion can be tested

Using variety of techniques

Applying random set of patterns has proven particularly successful

Flip-flops can be verified much more easily

If one doesn't have to worry about

Long sequences or

Cyclic behavior

There are two minor problems

How can one segregate the two portions of the circuit and

How can one gain visibility of the flip-flop states

From the primary outputs

Recall that in the earlier designs of counters and shift registers

Several different pieces of functionality
Count up/count down, shift left/shift right
Were combined into a single circuit
Used a selector input to choose between them

Let's re-examine that idea
Reuse of designs is always a good plan

We begin by identifying the pieces of functionality that we'll need
To verify the proper operation of the flip-flops
Among other things
Must show that each device can be placed into the logical 0 or logical 1 state
If all the flip-flops
Can be configured into a shift register and
A pattern of logical 0's and 1's shifted through
Task can be accomplished
Thus we have the first piece of functionality
Shift register

Next the combinational logic must be confirmed
Confirmation can take either of two directions
Test for any stuck-at faults or
Confirm that all of the state transitions can be executed
Either path requires
Known values be placed on the inputs
To the combinational logic block.

Values $\{X_n\}$ can be entered through circuit's primary inputs
It's known how to configure our flip-flops as a shift register
When so configured
Possible to shift in any desired pattern
Specifically can shift in a pattern that can
Place any of the needed values on the variables $\{Y_m(t+1)\}$ from the model

Values $\{Z_k\}$ can be confirmed through the primary outputs
What remains is to confirm are the values of variables $\{Y_m(t)\}$
These too can be accessed through the shift register

Second piece of functionality

To configure the shift register to support a parallel load

This is no different from the normal behavior of the flip-flops

Scan Path

The test can be executed

By appropriately choosing between

Shift register functionality and

Normal operation of the system

To implement the strategy

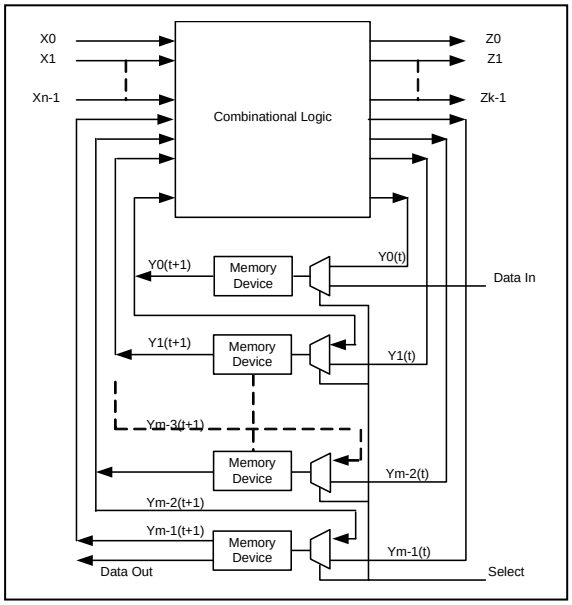
Original circuit must be modified to allow

Selection between the two modes

Such reconfiguration can be controlled

Through a simple multiplexer

As seen in accompanying figure



The shift register is called a *scan path*

Serial *Data In* and *Data Out* are called

Scan in and *scan out* respectively

Multiplexer control permits one to select

Between normal operation and test

These signals are brought to connector pins

To provide easy access and connection to a test fixture

A clock to the machine is included in the set of signals

Brought to the outside

Although one could use the internal system clock

Tests can be executed more effectively

If the internal clock can be disabled and

Test clock provided to control the machine.

- ✓ When the test mode is selected
  - Memory devices are configured as a shift register
  - Data can be
    - Entered through the *scan in* port and
    - Read from the *scan out* port

- ✓ In the normal mode
  - Values $\{Y_m(t)\}$ will appear on the flip-flop inputs and
  - Can be stored when the devices are clocked
  - Can then switch back to the test mode to shift the values out

Using a scan path now gives
- Some visibility into and control over the test of a finite state machine

Question - What if there are two or three such machines
- Are separate
  - Input and output paths
  - Clocks
  - Test mode control required for each machine
- Some good questions
  - This is where Boundary Scan comes in

Scan Design Testing

Strategy for debugging sequential machines
- Using scan design techniques is rather straightforward

Flip-flops are at the heart of
- Circuit and the technique

So it's best to ensure
- They are functioning properly first

In drawing above
- ✓ Initially *select* input is placed into the *test* mode
- ✓ With the flip-flops configured as a shift register
  - Pattern of alternating 0's and 1's
    - Entered on the *scan in* input
    - Monitored on the *scan out* output
    - If entered pattern
      - Appears properly on the output
      - Can assume
        - Devices are working correctly
        - Can proceed.

- If output appears as a constant 0 or 1
  Infer that one of the devices (or its input) is malfunctioning
- If manual access to the individual flip-flop pins is possible
  Debug by probing those signals

If such access is not available
Little that can be done

✓ Assuming necessary access is available
- First debugging step
  Verify the reset signal
  If it's present.
  To ensure that the machine is not being held in reset state
  Same done with the preset signal
- Next the presence of the clock on each device is confirmed

- If those signals are operating properly
  Flip-flop outputs are examined
  To determine how far through the register
  Test pattern has correctly propagated

✓ Next Combinational logic is addressed
Once the flip-flops are confirmed to be functioning correctly

Stuck at Strategy

If one is using a *test for stuck-at strategy*
The *test* mode is selected and
Shift register and the primary inputs $\{X_n\}$
Used to enter the test vectors
To be applied to the combinational net

The logic switched to *normal* mode
State of the $\{Y_m(t)\}$ signals is strobed into the flip-flops

Going back to the *test* mode once again
Allows the state of the combinational logic
To be shifted out through the *scan out* line
Where it and the $\{Z_k\}$ values
Compared with the proper values.

Verify Truth Table Strategy

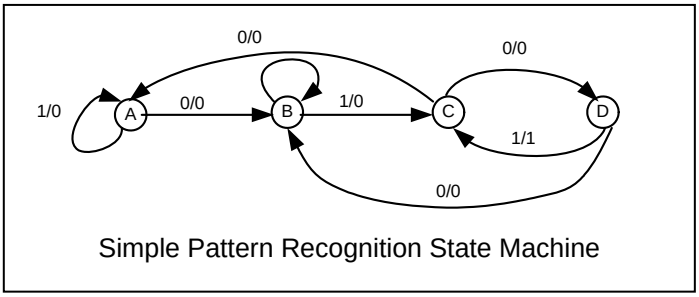
A *verify the truth table strategy*

Not much more complex.

Let's illustrate for simple pattern recognition state machine

Given in following figure

See how scan design now applies



Simple Pattern Recognition State Machine

As a first step

Select a state transition to be verified

Next select the *test* mode

Shift in the values for $\{Y_m(t)\}$

That will cause the transition

Enter the *normal* mode

Execute the transition

Go back to *test*

Clock out and confirm the flip-flop state

Process is repeated for each transition to be verified

The truth table and excitation tables are given in following figures

Present State	Next State/ Output	
	X=0	X=1
A	B/0	A/0
B	B/0	C/0
C	D/0	A/0
D	B/0	C/1

Simple binary assignment for the state variables

Used which yields accompanying excitation table

Test begins by placing the machine into state A

The {00} state

Present State	Next State/ Output	
M N	X=0	X=1
A 0 0	B/0 0 1 / 0	A/0 0 0 / 0
B 0 1	B/0 0 1 / 0	C/0 1 0 / 0
C 1 0	D/0 1 1 / 0	A/0 0 0 / 0
D 1 1	B/0 0 1 / 0	C/1 1 0 / 1

Initialization

Initialization is accomplished by

Placing the FSM into the test mode and

Entering the pattern {00} on the scan in line

The X input is placed in the logical 0 state and

State machine into the *normal mode*

Output, Z, is confirmed to be in the logical 0 state

Test

Next the machine is clocked one time

To execute the transition to state B

The {01} state

The test mode selection is reasserted

The contents of the flip-flops shifted out

Examined to ensure that the proper transition occurred

Accompanying table illustrates the test sequence

For the first two rows of the state table

Step	Select	***Action***	***State***	***Scan In***	***Scan Out***	***Input***	***Output***
1	Test	2 clocks, enter starting state {0,0}	00	00	-	-	-
2	Normal	Enter Input / Read Output	00	-	-	0	0
3	Normal	1 clock, enter next state	01	-	-	0	0
4	Test	2 clocks, shift out current state, shift in next starting state {0,0}	01	00	01	-	-
5	Normal	Enter Input / Read Output	00	-	-	1	0
6	Normal	1 clock, enter next state	00	-	-	1	0
7	Test	2 clocks, shift out current state, shift in next starting state{0,1}	00	01	00	-	-
8	Normal	Enter Input / Read Output	01	-	-	0	0
9	Normal	1 clock, enter next state	01	-	-	0	0
10	Test	2 clocks, shift out current state, shift in next starting state {0,1}	01	01	01	-	-
11	Normal	Enter Input / Read Output	01	-	-	1	0
12	Normal	1 clock, enter next state	10	-	-	1	0
13	Test	2 clocks, shift out current state, shift in next starting state{10}	10	01	00	-	-
14..19		Continue with remaining states					

If failure occurs

Combinational logic is debugged

Exactly as was done with a pure combinational logic system

The one difficulty may be limited visibility

Into the internal nodes of the circuit

Boundary Scan – Extending Scan Path Techniques

As we saw

Scan path techniques can provide visibility into the internals of a state machine

Without much work

Idea can be extended to an entire chip or system

Scan path concept

Forms the basis for what is known today as *boundary scan*.

Background

In the mid to late 1980's
Group of international electronics firms
Recognized that common means for
Testing and debugging complex systems and
Integrated circuits
Would be economically essential
For each to remain competitive in today's market
Together they formed
JOINT Test Action Group (JTAG)
Through their efforts developed for such systems
Standard Test Access Port and Boundary-Scan Architecture
Commonly known as JTAG,

Standard proposes
Simple four wire serial interface
Through which test vectors and test results
Can be entered into and read from a system or integrated circuit
In 1990 the IEEE adopted the work of the Joint Test Action Group
As a standard (IEEE 1149.1) defines
Common protocol
Boundary-scan architecture
Today IEEE 1149.1 has become accepted as an industry standard

It is a structured design-for-test approach
Well-suited for tools and automation

Tools developed to support the standard
Can control the *TAP - Test Access Port*
If they know how the boundary-scan architecture
Implemented in the device

Tools can also control the I/O pins of the device

BSDL Files

The subcommittee has developed and approved an industry
Standard language called *Boundary-Scan Description Language – BSDL*

Interestingly BSDL is a subset of VHDL
VHSIC Hardware Description Language
That describes
How IEEE 1149.1 is implemented in a device and
How it operates
For a device to be JTAG compliant
It must have an associated BSDL file.
These files are often available for download
From manufacturers' websites
http://www.altera.com/support/devices/bsdl/bsdl.html

BSDL
Captures the essential features of any IEEE 1149.1 implementation
BSDL was approved in 1994 as IEEE Std. 1149.1b

Provides a standard machine and human-readable data format
Describing how IEEE 1149.1 is implemented in a device

BSDL files contain the following elements:

- Entity Description: Statements naming the device or a section of its functionality.
- Generic Parameter: A value such as a package type. The value may come from outside the current entity.
- Port Description: Describes the nature of the pins on the device (input, output, bidirectional, linkage).
- Use Statements: References external definitions (such as IEEE 1149.1).
- Pin Mapping(s): Maps logical signals in the device to physical pins.
- Scan Port Identification: Defines the pins used on the device to access the JTAG capabilities (TDI, TDO, etc - the Test Access Port).
- Instruction Register Description: The signals used for accessing JTAG device modes.
- Register Access Description: Which register is placed between TDI and TDO for each JTAG instruction.

- Boundary Register Description: List of the boundary scan cells and their functionality.

Implementation

Boundary scan provides a software method

To control and observe the values on connector or I/O pins

On any circuit board or integrated circuit

That is compatible with the JTAG standard

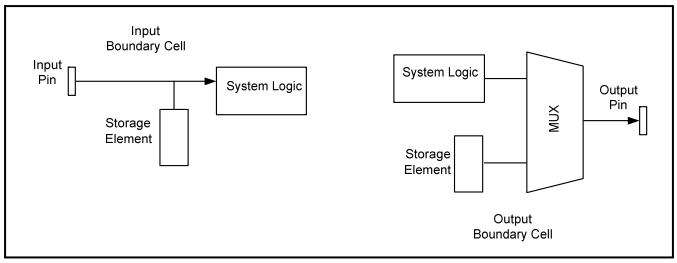
Let's take a look at

The basic components and architecture

For a boundary scan system

We'll begin with the input and output pin structure

These are illustrated in the following two diagrams



Normal Operation

During normal operation

Boundary cells are disabled and

I/O signals pass into or out of the system normally

Test Mode

In the test mode

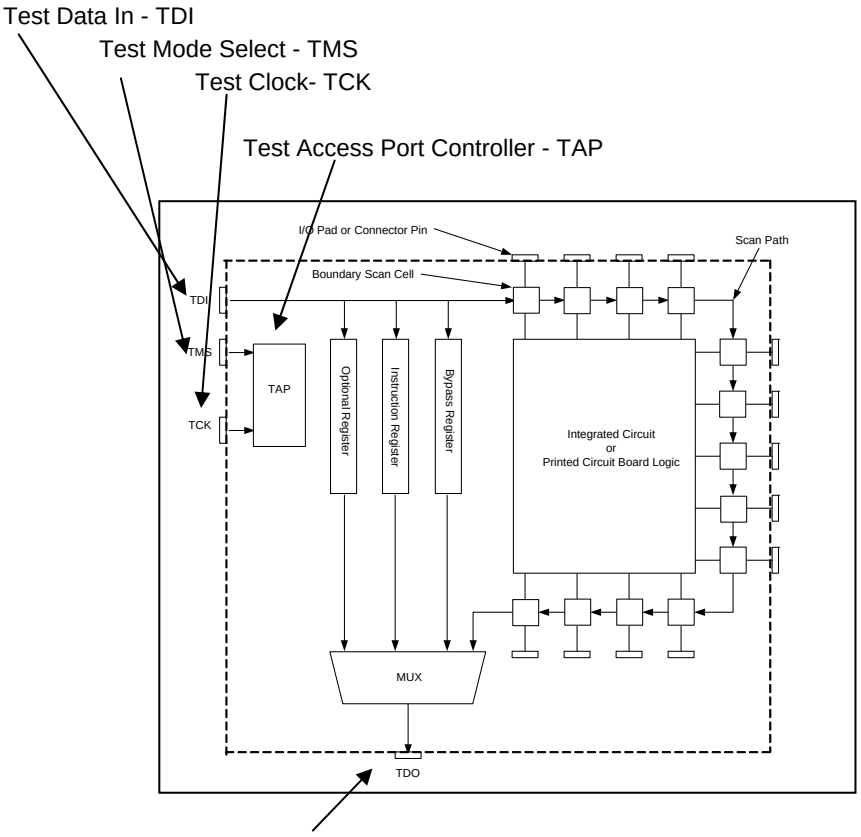
Input signals

Shadowed in the associated storage elements and

Output signals are set to selected test values

That are propagated down the scan path

To test other devices



Scan cells are controlled through

Test Access Port (TAP) and

Instruction Register

These are illustrated in previous figure

Tap Controller

The TAP Controller is a simple state machine
- With up to 16 possible states

It’s used to control the actions
- Associated with the boundary scan cells

The system uses the four signals
- *Test Mode Select* (TMS)
- *Test Data In* (TDI)
- *Test Data Out* (TDO)
- *Test Clock* (TCK)

To manage the operation of the boundary scan system

TDI and TDO signals used to
- Introduce data into the system or
- Propagate test signals or data out

The boundary scan system can easily be extended to
- Multiple integrated circuits or boards
  - Within a system
- Simply by connecting the TDO signal
  - From one portion
  - To the TDI signal of the next circuit
- By activating the Bypass Register
  - One can sidestep the scan path on selected components
    - While testing others

Test operation is initiated
- When a command is sent by a tester
  - On the *Test Mode Select* (TMS) input
    - Directing the *Instruction Register* (IR)
      - To load a sequence of bits appearing on the Test Data In (TDI) input
- The data received by the IR
  - Identifies which type of test the system is to perform

Required Instruction Set

Latest version of the specification, IEEE 1149.1-2001

Dictates that all devices must support the following four required instructions:

EXTEST

Places the device into an external boundary test mode and

Connects the boundary scan register (BSR) between the TDI and TDO pins

On execution

Output boundary scan cells are loaded with test patterns

To send to downstream devices

Input boundary scan cells

Are configured to shadow the input data

BYPASS

Directs that a device's scan path is to be bypassed and

The TDI input data be routed directly to the TDO output

Without interfering with the ongoing operation of the device

SAMPLE

Allows the device to remain in a functional and

Rather than a test) mode

Connects the BSR between the TDI and TDO pins

During execution,

Boundary scan register can be read to monitor data entering the device

PRELOAD

Also used to load known data values into the BSR

Can use such values for initialization for example

Both the PRELOAD and SAMPLE instructions

Used as a prelude to future activity

Optional Instruction Set

In addition to the required instructions

Standard specifies several optional instructions

These are given as

INTEST

Used to activate the BSR for tests of a device’s internal logic

IDCODE

Accesses a device’s (optional) internal identification register

That contains manufacturer and part specific information

Data is returned through the TDO register

USERCODE

Permits user identification data

To be loaded into the device's identification register

Such information is useful

If a device has

An original generic function and

Such as gate array

A value-added identification

such as a high speed graphical manipulator

BIST

Used to invoke user-defined built-in self-test functionality

It is assumed that the built in capability

Does not need any external initialization

Once the built-in tests have been completed

Results are sent to the outside world via the TDO output path

CLAMP

Works in conjunction with PRELOAD

Drives the device's output lines

With the values that have been entered during PRELOAD

Then enables the BYPASS functionality

The CLAMP instruction can be used to ensure

Device's outputs are held in a desired state

Such capability can be used to avoid conflicts

Between several different devices sharing a common bus.

HighZ

Works like the CLAMP instruction to ensure

Specific device outputs are placed into the high impedance state

The IEEE 1149.1 Standard also provides for an optional input pin TRST

That can be used to asynchronously reset the TAP controller

Boundary Scan Testing

The approach one takes to debugging larger parts of the system

Often directed by the technologies that have been used to execute the designs

Today we frequently are working with

ASICS, CPLDs, array logics, or custom designed integrated circuits

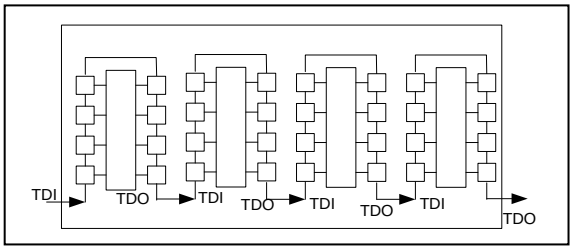
These are VLSI class components

In such a context

Boundary scan methods can provide a very powerful tool

If the system has been designed to support boundary scan capability

High level architecture appears as in accompanying figure



From here, testing based on three point strategy:

Test the tester

Test the device interconnect

Test the devices

- Starting with the *test the tester* heuristic

  Ensures boundary scan infrastructure is functional

  Simple way to do this

  - ✓ Force each device in the system to connect

    Instruction register between its TDI and TDO pins

    There is now a path

    From the TDI signal on the input side of the system

    Through all the devices

    To the TDO signal on the output side of the system

  - ✓ Each device in the chain

    Can be directed to place a 1-0 pattern into the IR

    The combined pattern

    Will propagate through all devices in the chain

    Can be detected at the output

    If the appropriate number of 1-0 pairs does not appear

    Can assume that there is a problem in the system

- Next the interconnections between

  Devices in the system are verified

  Generally the system uses a bus structure to

  Exchange commands or data between the constituent devices

  These interconnections can easily be checked using

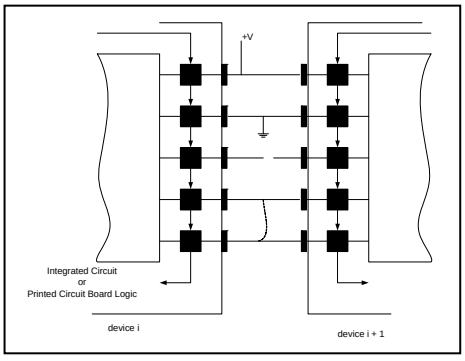
  Input / output capabilities of the boundary scan cells

  On the various scan paths.

  To see how this might work

  Consider circuit fragment in accompanying figure

  Three kinds of faults have been incorporated



- Next the EXTEST instruction
  - Used to propagate various test patterns
    - Through the scan register
  - Reasonable choice
    - Begin with all 0's then all 1's
  - With such patterns
    - Stuck-at (including open) faults are identified
      - These faults will appear as a bit
        - With the opposite value of the test pattern
          - For a stuck-at 0 and a pattern of 1's, the 0 will appear and vice versa
    - Testing follows with an alternating 0-1 test pattern
      - To identify the bridge type faults
        - Two consecutive bits appearing with the same value

- Finally we might
  - Use the RUNBIST instruction or
    - Command the individual devices
      - To execute their self-test sequences
  - Some version of INTEST
    - To test the device's internal logic

After completing the self-tests
- Following will have been confirmed
  - Boundary scan infrastructure is working
  - Most common manufacturing faults have been detected
    - Stuck-at, open, or bridge
  - Each of the devices that supports boundary scan in the system
    - Is present and has a basic level of functionality

One must include other non-boundary scan tests in the test suite
- To cover any remaining untested devices

Additional more detailed tests can be included as deemed necessary

As with the scan design approach
If a failure is detected
Debugging can be a bit complex
Generally when using scan design or boundary scan tools
System is made up of mainly VLSI class devices
With such devices we have only limited visibility into the internals
Such a restriction is not as severe as it may first appear long term
Remember in production
Objective is to identify manufacturing defects
Such defects are going to be external to the devices

Design for Testability

Background

Circuits will becoming more complex and
Circuit boards will become much more dense.
Today almost all ball grid array (BGA) devices and
Most surface mount devices incorporate 1149.1
Not every Engineer knows how to incorporate boundary scan into their design.
Boundary scan can be ready for board test and
Flash programming before the prototype board available for test
Boundary Scan implementation is highly automated.
Fixturing for Boundary Scan is minimal.
Boundary Scan will identify board faults
Down to the pin and net level.

General Guidelines

- ✓ Involve the Test Engineer early in the design process
- ✓ Choose components carefully
- ✓ Provide TAP connector(s) or other means of access to the TAP signals
  Don’t forget easy access to pwr/gnd
- ✓ Consider using multiple chains (no loss of test coverage)
- ✓ Scan signal terminations
  Pull-ups, buffers, TCK termination
- ✓ Select IEEE 1149.1 compliant devices where possible
- ✓ Where possible ensure that device silicon has been verified
  Against latest Boundary-Scan Description Language (BSDL) file
  Some IC vendors state this within BSDL file header
- ✓ Connect Boundary-scan devices into one chain or
  Partitioning into multiple, separate chains for
  Scan path optimization for flash programming

Segment different voltage families
3.3V, 2.5V and 1.8V
Providing improved test partitioning for
Executing at-speed BIST tests for testing high speed serial links
Meeting requirements of proprietary test and debugging tools

Self Test

Series of built in tests system can execute
Goal are to
Ensure system working
Basis for action if
Element fails
System locks up
Two general categories
- ✓ Those invoked on demand
  - Command
  - Push button somewhere
- ✓ Those running in background

Demand

These are a sanity check to ensure system basically operational
Often done at power up
Report a status on completion
Word of caution when developing such tests
Process of simply executing test often requires
Most of system to be working

Background

Can be as simple as watchdog timer
Must be periodically reset by CPU
If it expires
Forces action
Extreme as system reset
Benign as warning or error message
Complex as test suite running in background
Can check
Busses - stuck lines
Memory
ROM - signature
RAM - failed or stuck bits

Math processing
Built in
A/D
Measure a known reference
D/A
Convert at cardinal points
Test A/D against D/A
Timers

Caution
Anything added to system for testing can fail as well

Summary

In this lesson we

- ✓ Began with an overview of the test process.
- ✓ Introduced the basic test terminology.
- ✓ Identified the reasons for testing.
- ✓ Identified the basic necessary documentation.
- ✓ Examined testing for yourself and for your customer.
- ✓ Identified contemporary test problems and techniques.
- ✓ Analyzed scan design and boundary scan.
- ✓ Briefly covered design for testability and self-test.