

## Boundary Scan Tutorial

A tutorial prepared by Dr R G “Ben” Bennetts  
DFT Consultant and Director, ASSET InterTech Inc.

Tel: +44 1489 581276 E-mail: [ben@dft.co.uk](mailto:ben@dft.co.uk)

Welcome!!



A Tutorial prepared by Dr R G “Ben” Bennetts  
DFT Consultant,  
Director, ASSET InterTech Inc.  
+44 1489 581276  
[ben@dft.co.uk](mailto:ben@dft.co.uk) [www.dft.co.uk](http://www.dft.co.uk)

Figure 1

## Introduction and Objectives

# IEEE Standard 1149.1 Boundary-Scan Standard

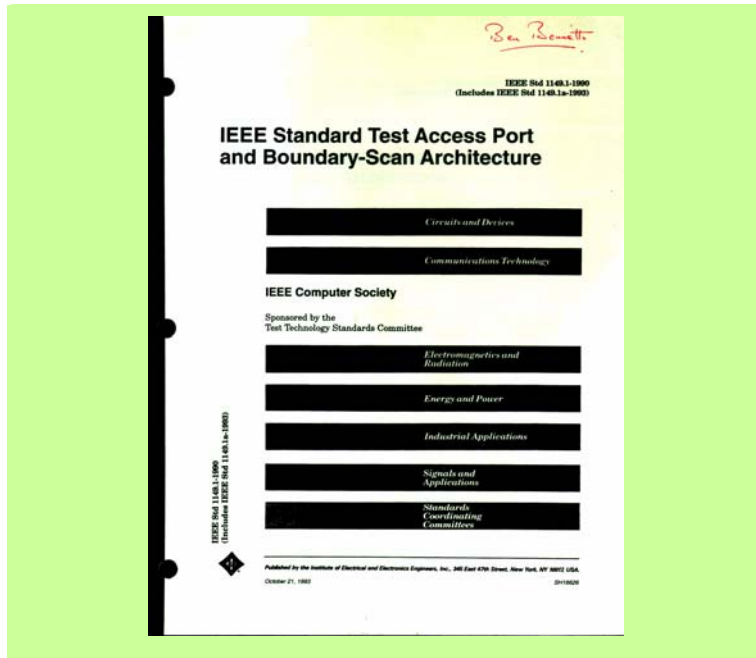


Figure 2

In this tutorial, you will learn the basic elements of boundary-scan architecture — where it came from, what problem it solves, and the implications on the design of an integrated-circuit device.

The core reference is the IEEE 1149.1 Standard:

IEEE Standard 1149.1-2001 “Test Access Port and Boundary-Scan Architecture,” available from the IEEE, 445 Hoes Lane, PO Box 1331, Piscataway, New Jersey 08855-1331, USA. The standard was first published in 1990, revised in 1993 and 1994, and most recently in 2001. You can obtain a copy of the Standard via the world wide web on the IEEE home page at: <http://standards.ieee.org/catalog>.

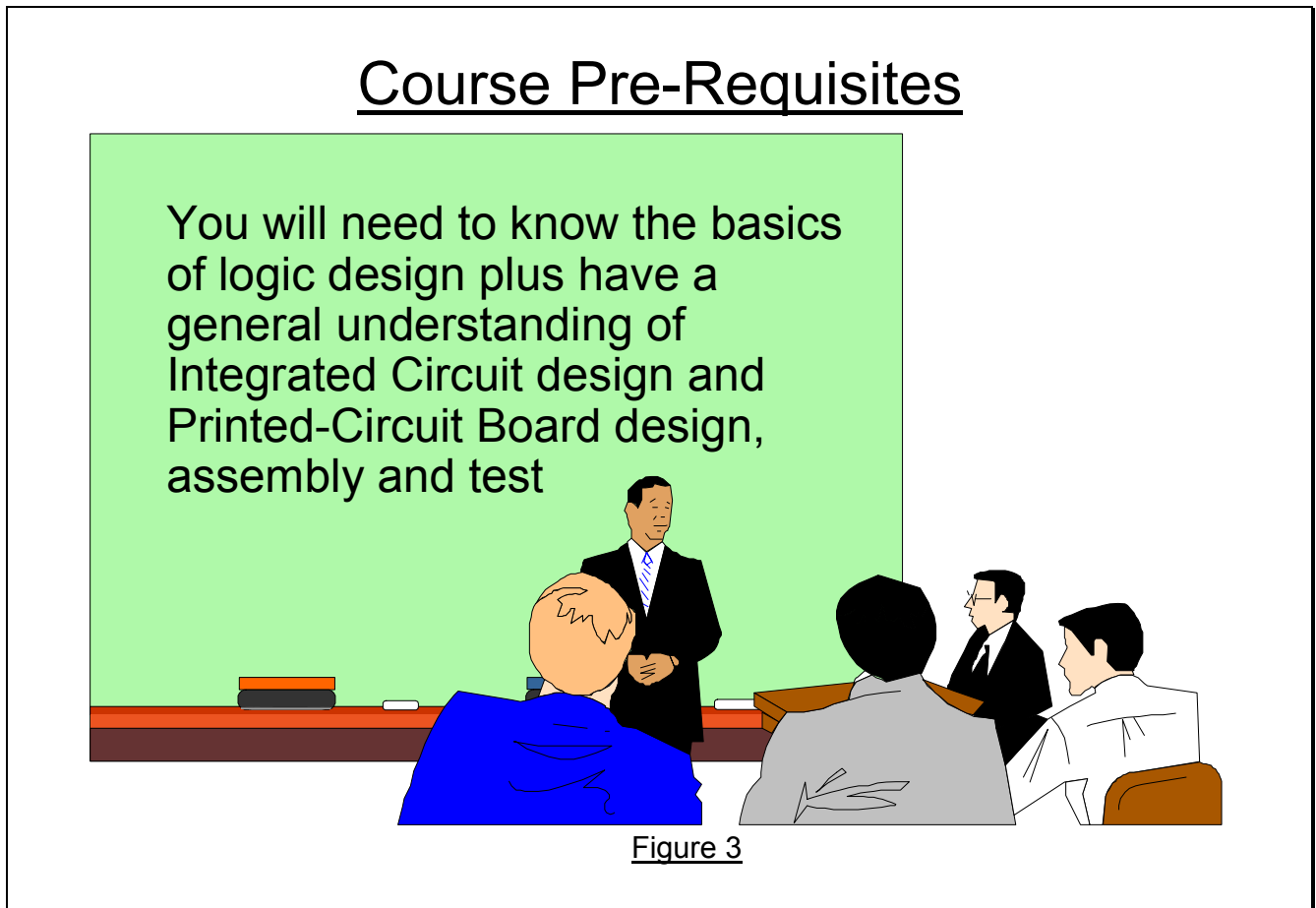
The 1993 revision to the standard, referred to as “1149.1a-1993,” contained many clarifications, corrections, and minor enhancements. Two new instructions were introduced in 1149.1a and these are described in this tutorial.

The 1149.1b-1994 supplement contained a description of the **Boundary-Scan Description Language** (BSDL).

The 1149.1-2001 version contains enhancements to the wording, plus removal of the use of the all-0s code for the **Exttest** instruction. In addition, the mandatory **Sample/Preload** instruction has been split into two separate instructions: **Preload** and **Sample**, both still mandatory.

For further, more recent publications on boundary-scan topics, see the *To Probe Further* section at the end of this tutorial.

### **Course Pre-Requisites**



Students who participate in this course are expected to know the basics of logic design plus have a general understanding of Integrated Circuit design principles and Printed-Circuit Board electronic design, board assembly and test techniques.

## About The Author

### About The Author



Dr R G “Ben” Bennetts is an independent consultant in Design-For-Test (DFT), consulting in product life-cycle DFT strategies, and delivering on-site and open educational courses in DFT technologies.

Previously, he has worked for LogicVision, Synopsys, GenRad and Cirrus Computers. Between 1986 and 1993, he was a free-lance consultant and lecturer on Design-for-Test (DFT) topics. During this time, he was a member of JTAG, the organization that created the IEEE 1149.1 Boundary-Scan Standard. He is an Advisory member of the Board of Directors of ASSET InterTech

Ben has published over 90 papers plus three books on test and DFT subjects.

Figure 4

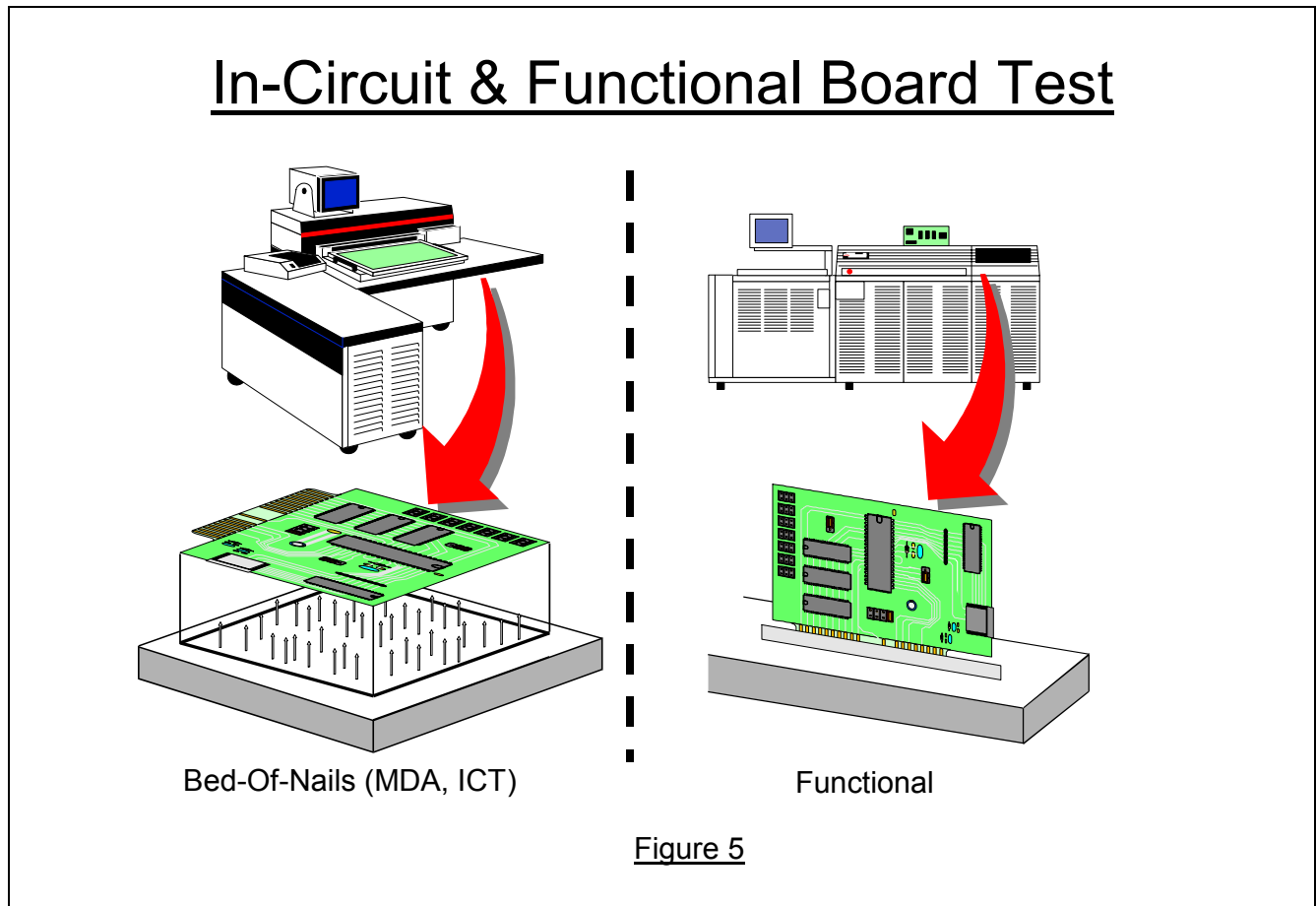
**Dr R G “Ben” Bennetts** is an independent consultant in Design-For-Test (DFT), consulting in product life-cycle DFT strategies, and delivering on-site and open educational courses in DFT technologies.

Previously, he has worked for **LogicVision, Synopsys, GenRad and Cirrus Computers**. Between 1986 and 1993, he was a free-lance consultant and lecturer on Design-for-Test (DFT) topics. During this time, he was a member of **JTAG**, the organization that created the IEEE 1149.1 Boundary-Scan Standard. He is an Advisory member of the Board of Directors of **ASSET InterTech**.

Ben has published over 90 papers plus three books on test and DFT subjects.

## The Motivation for Boundary-Scan Architecture

### *Historical Development: In-Circuit Test*



Since the mid-1970s, the structural testing of loaded printed circuit boards has relied very heavily on the use of the so-called in-circuit **bed-of-nails** technique (see Figure 5). This method of testing makes use of a fixture containing a bed-of-nails to access individual devices on the board through test lands laid into the copper interconnect, or into other convenient physical contact points. Testing then proceeds in two phases: power-off tests followed by power-on tests. Power-off tests check the integrity of the physical contact between nail and the on-board access point, followed by open and shorts tests based on impedance measurements.

Power-on tests apply stimulus to a chosen device, or collection of devices (known as a **cluster**), with an accompanying measurement of the response from that device or cluster. Other devices that are electrically connected to the device-under-test are usually placed into a safe state (a process called “guarding”). In this way, the tester is able to check the **presence**, **orientation**, and **bonding** of the device-under-test in place on the board.

## Changes in Device Packaging Styles

### Change of Device Packaging Styles

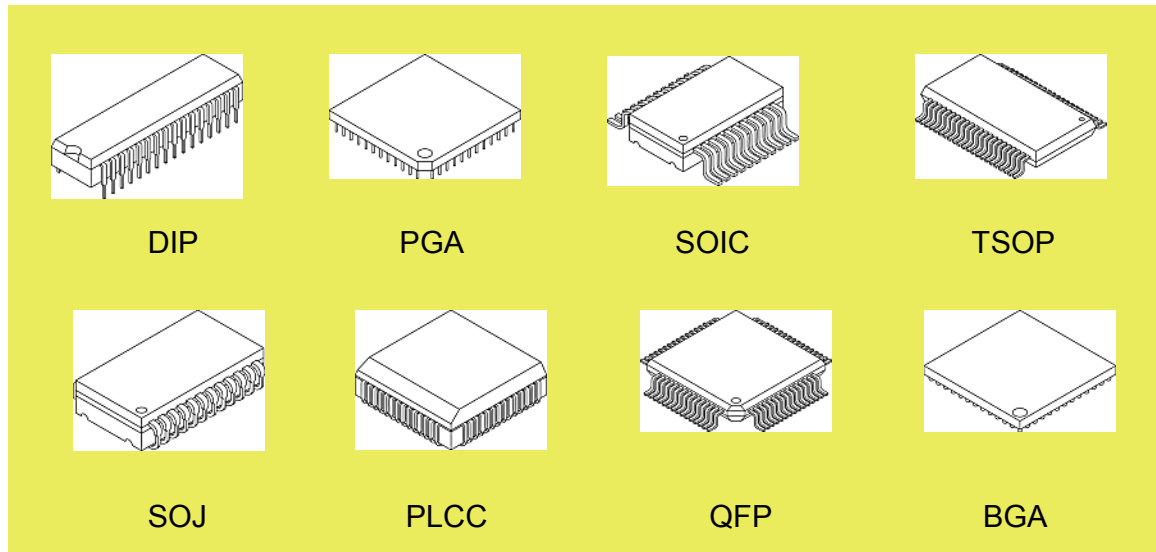
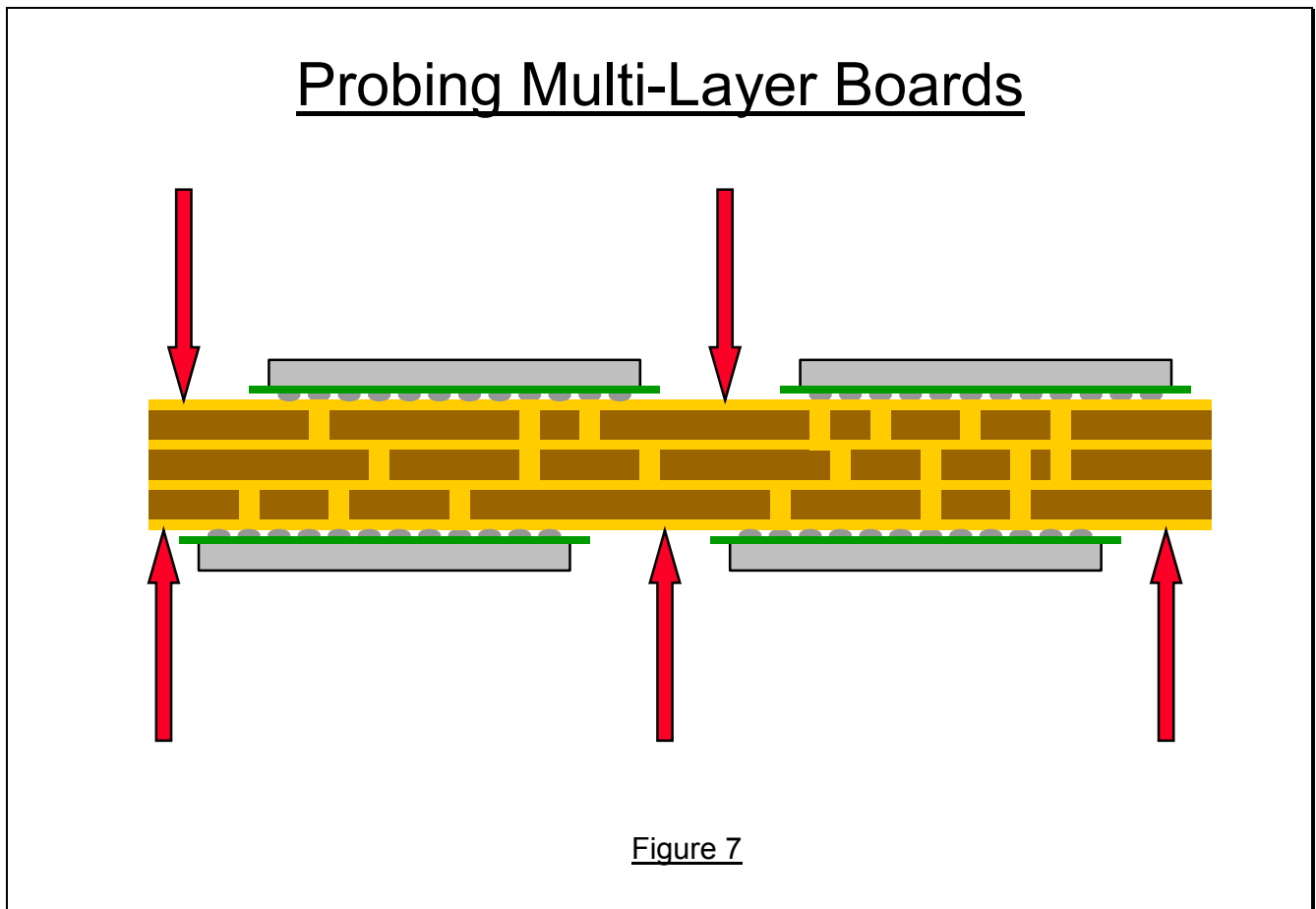


Figure 6

Fundamentally, the in-circuit bed-of-nails technique relied on physical access to all devices on a board. For plated-through-hole technology, the access is usually gained by adding test lands into the interconnects on the “**B**” side of the board — that is, the solder side of the board. The advent of inserted devices packaged in surface mount styles – see Figure 6 - meant that system manufacturers began to place components on both sides of the board — the “**A**” side and the “**B**” side. The smaller pitch between the leads of surface-mount components caused a corresponding decrease in the physical distance between the interconnects.

## Probing Multi-Layer Boards



The move to surface-mount packaging had a serious impact on the ability to place a nail accurately onto a target test land, as shown in Figure 7. The whole question of access was further compounded by the development of multi-layer boards created to accommodate the increased number of interconnects between all the devices. Basically, the ability to physically probe onto the board with a bed-of-nails system was going away: physical access was becoming limited.

## *The Emergence of JTAG*

### JTAG Meeting, 17 September, 1988



Figure 8

Such was the situation in the mid-1980s when a group of concerned test engineers in a number of European electronics systems companies got together to examine the board-test problem of limited access and its possible solutions. The group of people initially called themselves the **Joint European Test Action Group (JETAG)**. Their preferred method of solution was to bring back the access to device pins by means of an internal serial shift register around the boundary of the device - a **boundary scan** register.

Later, the group was joined by representatives from North American companies and the 'E' for "European" was dropped from the title of the organization leaving it **Joint Test Action Group, JTAG** - see Figure 8. (The author is in the front row, third from the right-hand end.) JTAG did not invent the concept of boundary scan. Several companies, such as IBM, Texas Instruments and Philips, were already working on the idea. What JTAG did was to convert the ideas into an international Standard, the IEEE 1149.1-1990 Standard, first published in April 1990.

## Summary

### Motivation for Boundary Scan: Summary


- Basic motivation was miniaturization of device packaging, leading to ...
  - surface mount packaging styles, leading to ...
  - double sided boards, leading to ...
  - multi-layer boards, leading to ...
  - a reduction of physical access test lands for traditional bed-of-nail in-circuit testers
- 
- Problem: how to test for manufacturing defects in the future?
  - Solution: add boundary-scan registers to the devices

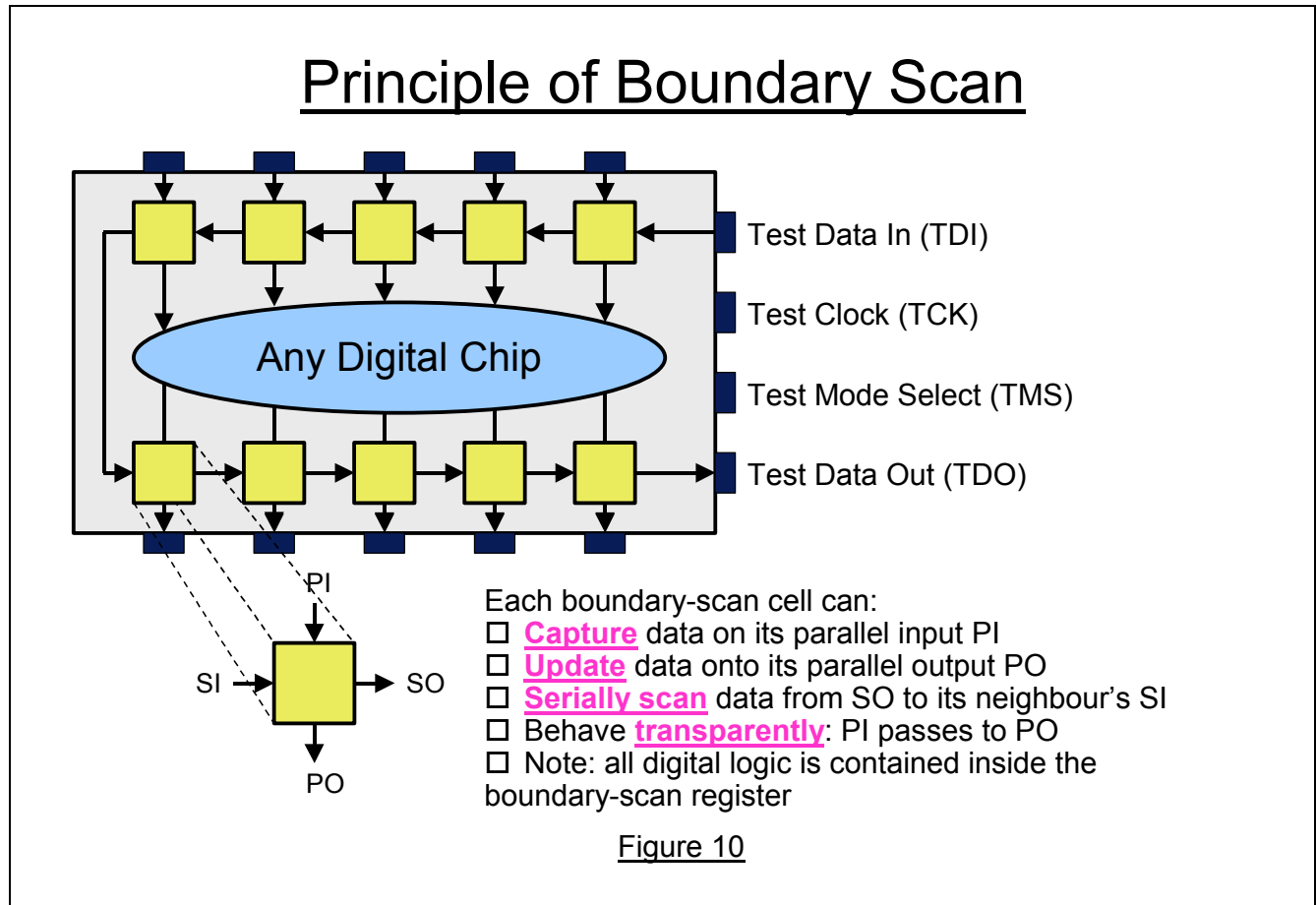
Figure 9

To summarize, the basic motivation for boundary scan was the miniaturization of device packaging, the development of surface-mounted packaging, and the associated development of the multi-layer board to accommodate the extra interconnects between the increased density of devices on the board. These factors led to a reduction of the one thing an in-circuit tester requires: physical access for the bed-of-nails probes.

The long-term solution to this reduction in physical probe access was to consider building the access inside the device i.e. a boundary scan register. In the next section, we will take a look at the device-level architecture of a boundary-scan device, and begin to understand how the boundary-scan register solves the limited-access board-test problem.

## The Principle of Boundary-Scan Architecture

### What is Boundary Scan?



In a boundary-scan device, each digital primary input signal and primary output signal is supplemented with a multi-purpose memory element called a boundary-scan cell. Cells on device primary inputs are referred to as “*input cells*”; cells on primary outputs are referred to as “*output cells*.” “*Input*” and “*output*” is relative to the internal logic of the device. (Later, we will see that it is more convenient to reference the terms “*input*” and “*output*” to the interconnect between two or more devices.) See Figure 10.

The collection of boundary-scan cells is configured into a parallel-in, parallel-out shift register. A parallel load operation — called a **Capture** operation — causes signal values on device input pins to be loaded into input cells, and signal values passing from the internal logic to device output pins to be loaded into output cells. A parallel unload operation — called an **Update** operation — causes signal values already present in the output scan cells to be passed out through the device output pins. Signal values already present in the input scan cells will be passed into the internal logic.

Data can also be **Shifted** around the shift register, in serial mode, starting from a dedicated device input pin called **Test Data In (TDI)** and terminating at a dedicated device output pin called **Test Data**

**Out (TDO).** The **Test Clock, TCK**, is fed in via yet another dedicated device input pin and the various modes of operation are controlled by a dedicated **Test Mode Select (TMS)** serial control signal.

### Using the Scan Path

## Using The Boundary-Scan Path

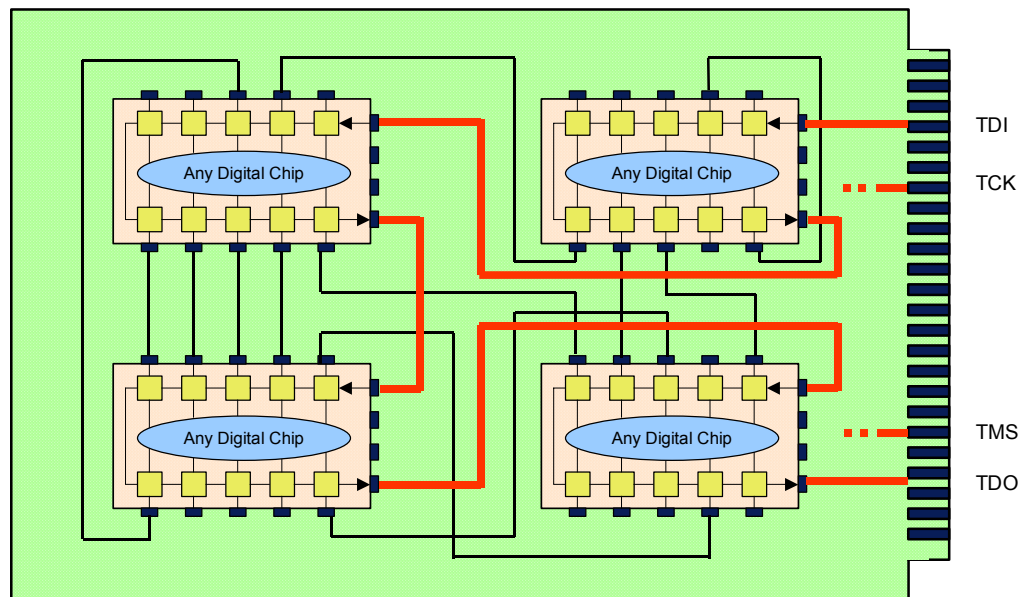
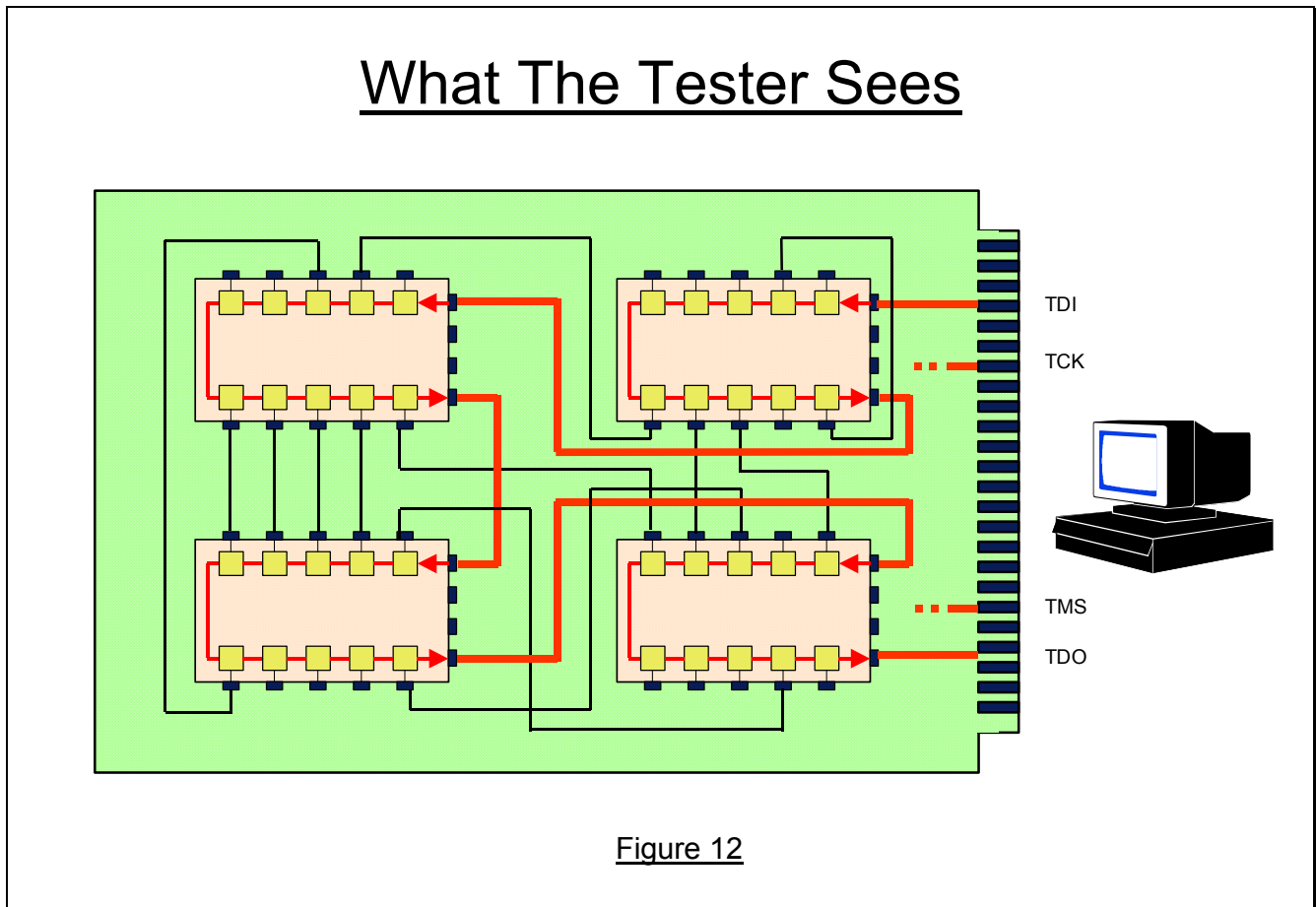


Figure 11

At the device level, the boundary-scan elements contribute nothing to the functionality of the internal logic. In fact, the boundary-scan path is independent of the function of the device. The value of the scan path is at the board level as shown in Figure 11.

The figure shows a board containing four boundary-scan devices. Notice that there is an edge-connector input called TDI connected to the TDI of the first device. TDO from the first device is permanently connected to TDI of the second device, and so on, creating a global scan path terminating at the edge connector output called TDO. TCK is connected in parallel to each device TCK input. TMS is connected in parallel to each device TMS input.

**What the Tester Sees**

What the tester sees from the edge connector is simply the concatenation of the various boundary-scan registers – that is, a single register that provides access to all device outputs now considered to be **drivers** (sometimes called a **transmitter**) onto an interconnect and all device inputs now considered to be **sensors** (sometimes called a **receiver**) from the interconnect – see Figure 12.

In this way, particular tests can be applied across the device interconnects via the global scan path by loading the stimulus values into the appropriate device-output scan cells via the edge connector TDI (**shift-in** operation), applying the stimulus across the interconnects (**update** operation), capturing the responses at device-input scan cells (**capture** operation), and shifting the response values out to the edge connector TDO (**shift-out** operation).

Essentially, boundary-scan cells can be thought of as **virtual nails**, having an ability to set up and apply tests across the interconnect structures on the board.

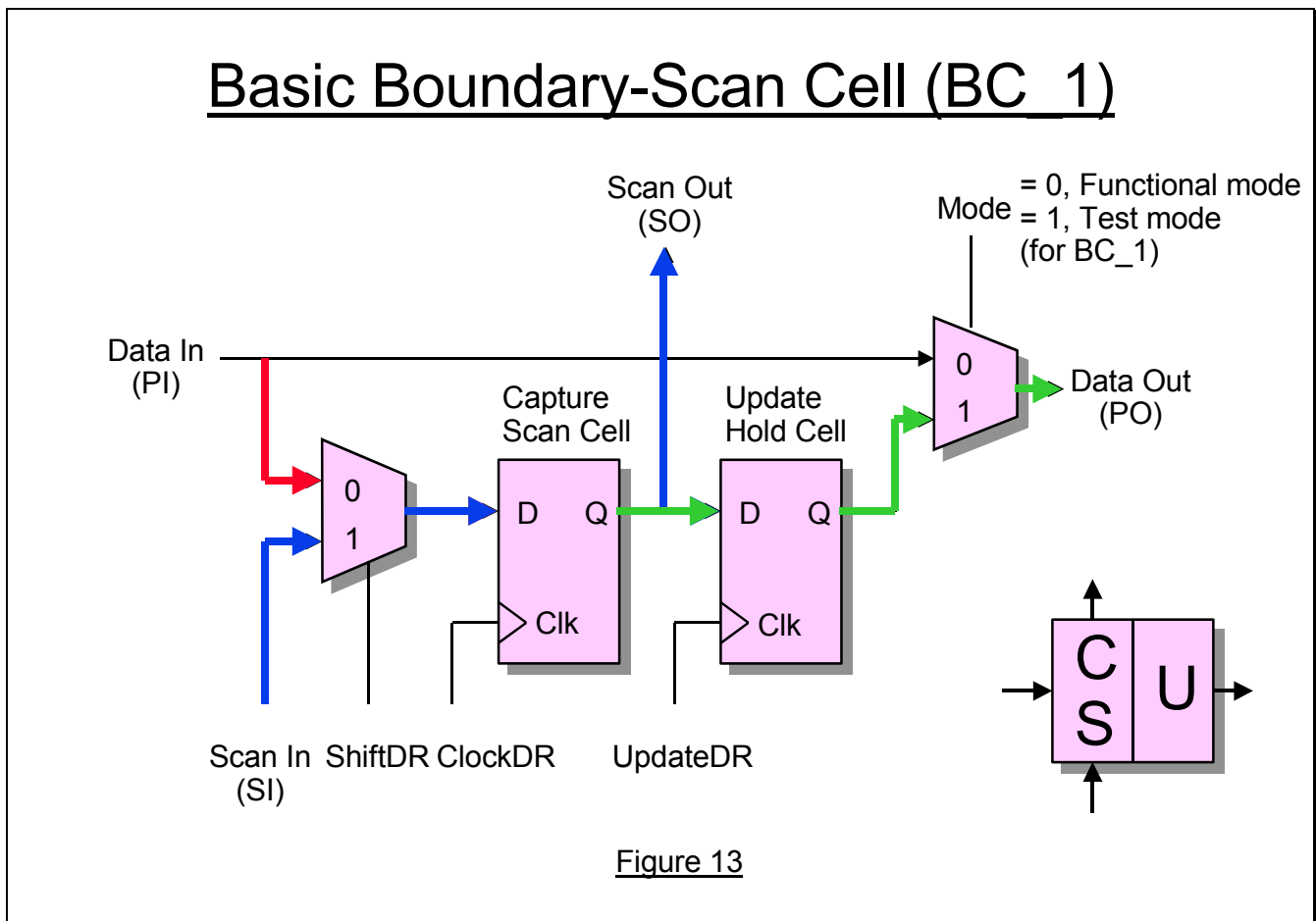
**Basic Boundary-Scan Cell (BC\_1)**

Figure 13 shows a basic universal boundary-scan cell, known as a **BC\_1**. The cell has four modes of operation: normal, update, capture, and serial shift. The memory elements are two D-type flip-flops with front-end and back-end multiplexing of data. (As with all circuits in this tutorial, it is important to note that the circuit shown in Figure 13 is only an example of how the requirement defined in the Standard could be realized. The IEEE 1149.1 Standard does not mandate the design of the circuit, only its functional specification.)

During normal mode, **Data\_In** is passed straight through to **Data\_Out**.

During update mode, the content of the Update Hold cell is passed through to **Data\_Out**.

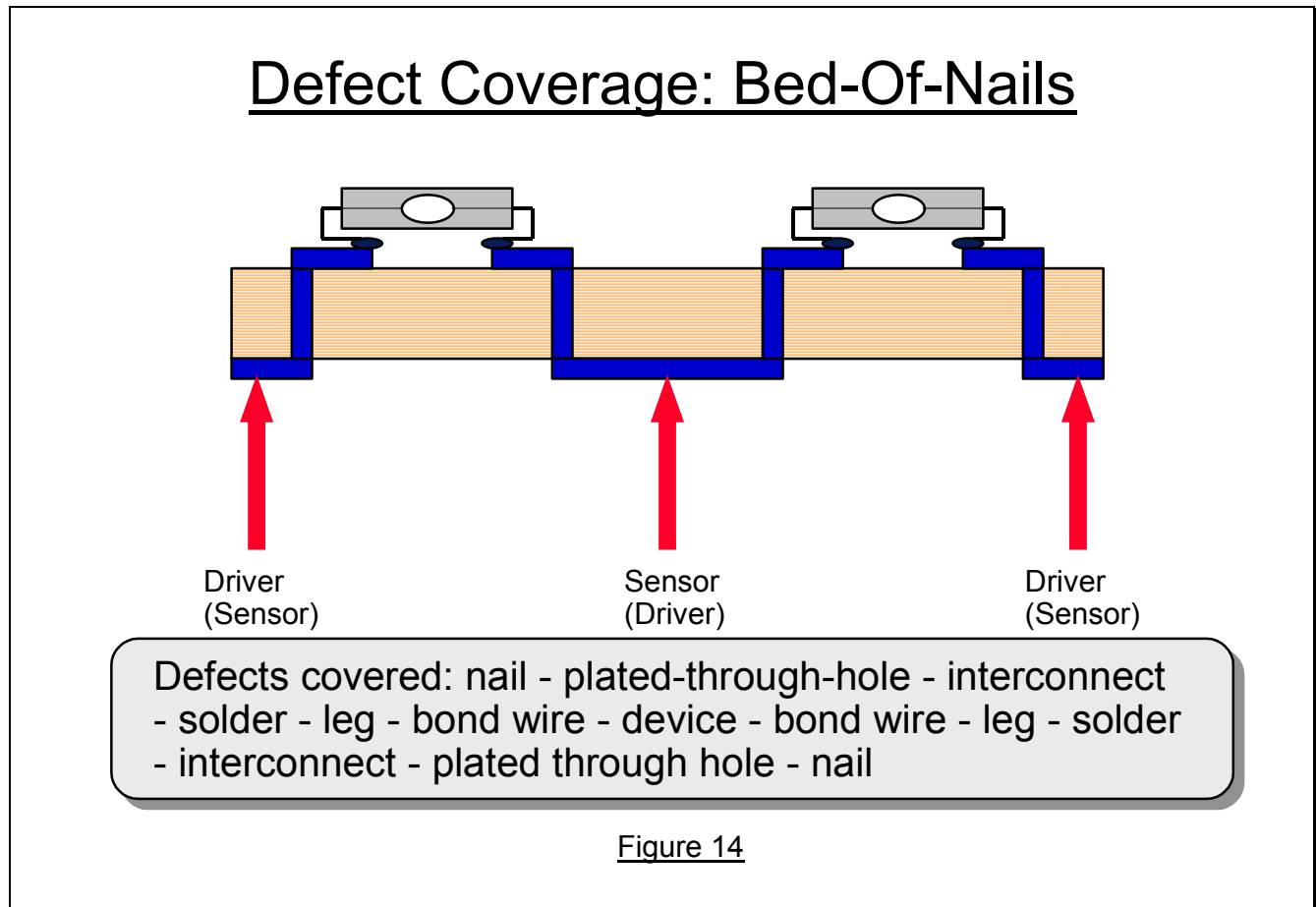
During capture mode, the **Data\_In** signal is routed to the input Capture Scan cell and the value is captured by the next **ClockDR**. **ClockDR** is a derivative of TCK.

During shift mode, the **Scan\_Out** of one Capture Scan cell is passed to the **Scan\_In** of the next Capture Scan cell via a hard-wired path.

Note that both capture and shift operations do not interfere with the normal passing of data from the parallel-in terminal to the parallel-out terminal. This allows **on the fly** capture of operational values and the shifting out of these values for inspection without interference. This application of the boundary-scan register has tremendous potential for real-time monitoring of the operational status of

a system — a sort of electronic camera taking snapshots — and is one reason why TCK is kept separate from any system clocks.

### Comparing Boundary Scan with In-Circuit Test



The use of boundary-scan cells to test the **presence, orientation, and bonding** of devices was the original motivation for inclusion in a device. The use of scan cells as a means of applying tests to individual devices is not the major application of boundary-scan architecture. Consider the reason for boundary-scan architecture in the first place. The prime function of the bed-of-nails in-circuit tester was to test for manufacturing defects, such as missing devices, damaged devices, open and short circuits, misaligned devices, and wrong devices. See Figure 14.

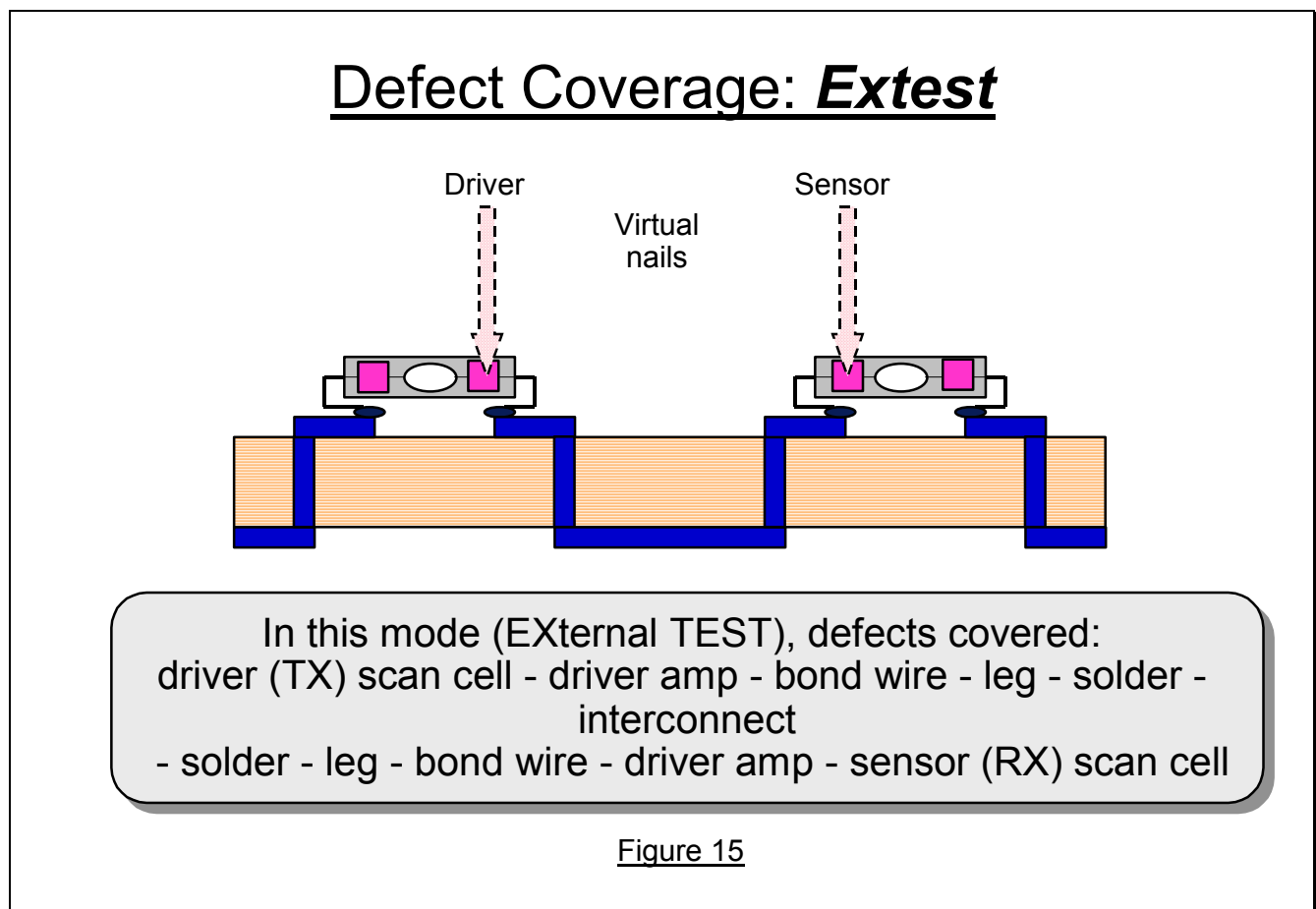
In-circuit testers were not intended to prove the overall functionality of the on-board devices. It was assumed that devices had already been tested for functionality when they existed only as devices (i.e., prior to assembly on the board). Unfortunately, in-circuit test techniques had to make use of device functionality in order to test the interconnect structure — hence the rather large libraries of merchant device functions and the problems caused by increasing use of ASICs.

Given that boundary-scan registers were seen as an alternative way of testing for the presence of manufacturing defects, we should question what these defects are, what causes them, and where they occur.

An examination of the root cause for board manufacturing defects shows them to be caused by any one of three reasons: electrical stress (e.g., electrostatic discharge causing damage to input or output amplifiers), mechanical stress (e.g., bent legs caused by clumsy handling when mounting devices on the board), or thermal stress (e.g., hot spots caused by the solder operation). A defect, if it occurs, is likely present either in the periphery of the device (leg, bond wire, driver amplifier), in the solder, or in the interconnect between devices. It is very unusual to find damage to the internal logic without some associated damage to the periphery of the device.

In this respect, the boundary-scan cells are precisely where we want them — at the beginning and ends of the region most likely to be damaged during board assembly i.e. the region between the output driver scan cell and the input sensor scan cell. This region is more-commonly referred to as the *interconnect* region.

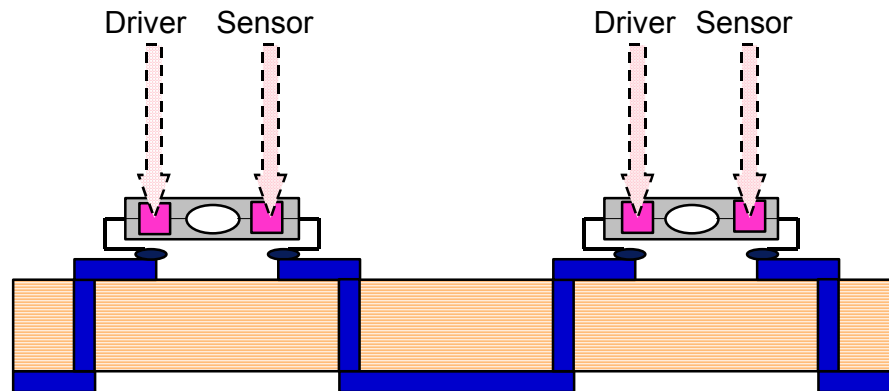
### **Extest Mode**



Using the boundary-scan cells to test the interconnect structure between two devices is called **External Test**, shortened to **Extest** – see Figure 15. The use of the cells for **Extest** is the major application of boundary-scan structures, searching for opens and shorts plus damage to the periphery of the device. In this mode, the boundary-scan cells are often referred to as **virtual nails**

### **Intest Mode**

## Defect Coverage: *Intest*



In this mode (INternal TEST), defects covered:  
driver scan cell - device - sensor scan cell

Figure 16

It is also possible to use boundary-scan cells to test the internal functionality of a device (Figure 16). This use of the boundary-scan register is called **Internal Test**, shortened to **Intest**. **Intest** is only really used for very limited testing of the internal functionality to identify defects such as the wrong variant of a device, or to detect some gross internal defect.

In the next section, we will take a closer look at the overall architecture of an 1149.1-compliant device and, particularly, the function of the **Instruction** register.

## IEEE 1149.1 Device Architecture and Instruction register

### Device Architecture

## 1149.1 Chip Architecture

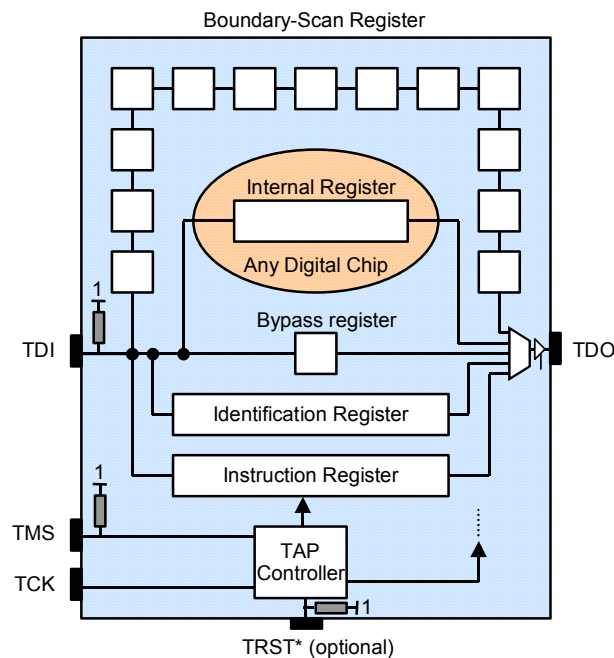


Figure 17

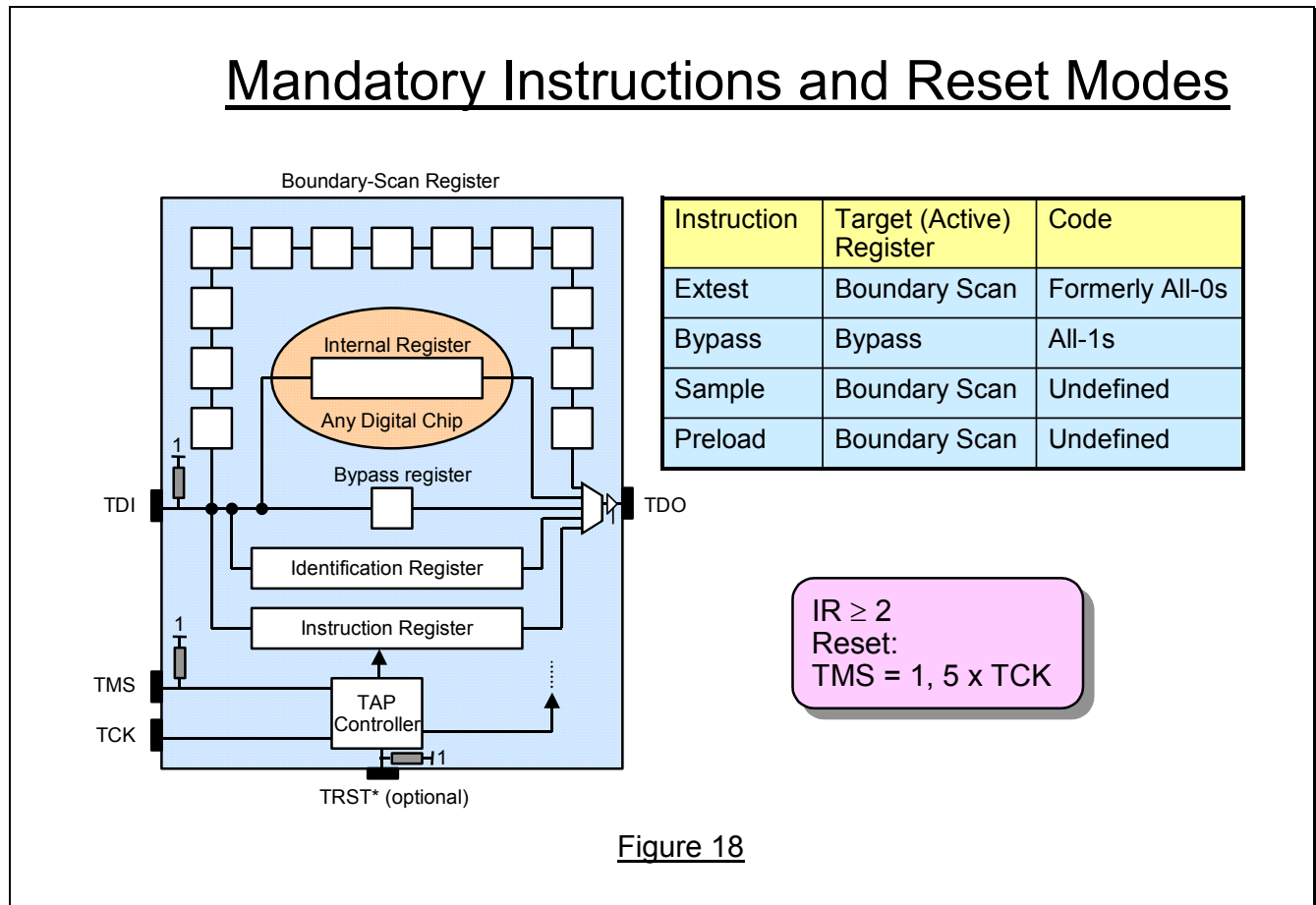
After nearly five year's discussion, the JTAG organization finally proposed the device architecture shown in Figure 17.

The Figure shows the following elements:

- ❑ A set of four dedicated test pins — Test Data In (TDI), Test Mode Select (TMS), Test Clock (TCK), Test Data Out (TDO) — and one optional test pin Test Reset (TRST\*). These pins are collectively referred to as the Test Access Port (TAP).
- ❑ A boundary-scan cell on each device primary input and primary output pin, connected internally to form a serial boundary-scan register (Boundary Scan).
- ❑ A finite-state machine TAP controller with inputs TCK, TMS, and TRST\*.
- ❑ An n-bit ( $n \geq 2$ ) Instruction register, holding the current instruction.
- ❑ A 1-bit Bypass register (Bypass).
- ❑ An optional 32-bit Identification register capable of being loaded with a permanent device identification code.

At any time, only one Data register can be connected between TDI and TDO e.g., the Instruction register, Bypass, Boundary-Scan, Identification, or even some appropriate register internal to the device. The selected Data register is identified by the decoded parallel outputs of the Instruction register. Certain instructions are mandatory, such as **Extest** (boundary-scan register selected), whereas others are optional, such as the **Idcode** instruction (Identification register selected).

### Mandatory Instructions and Reset Modes

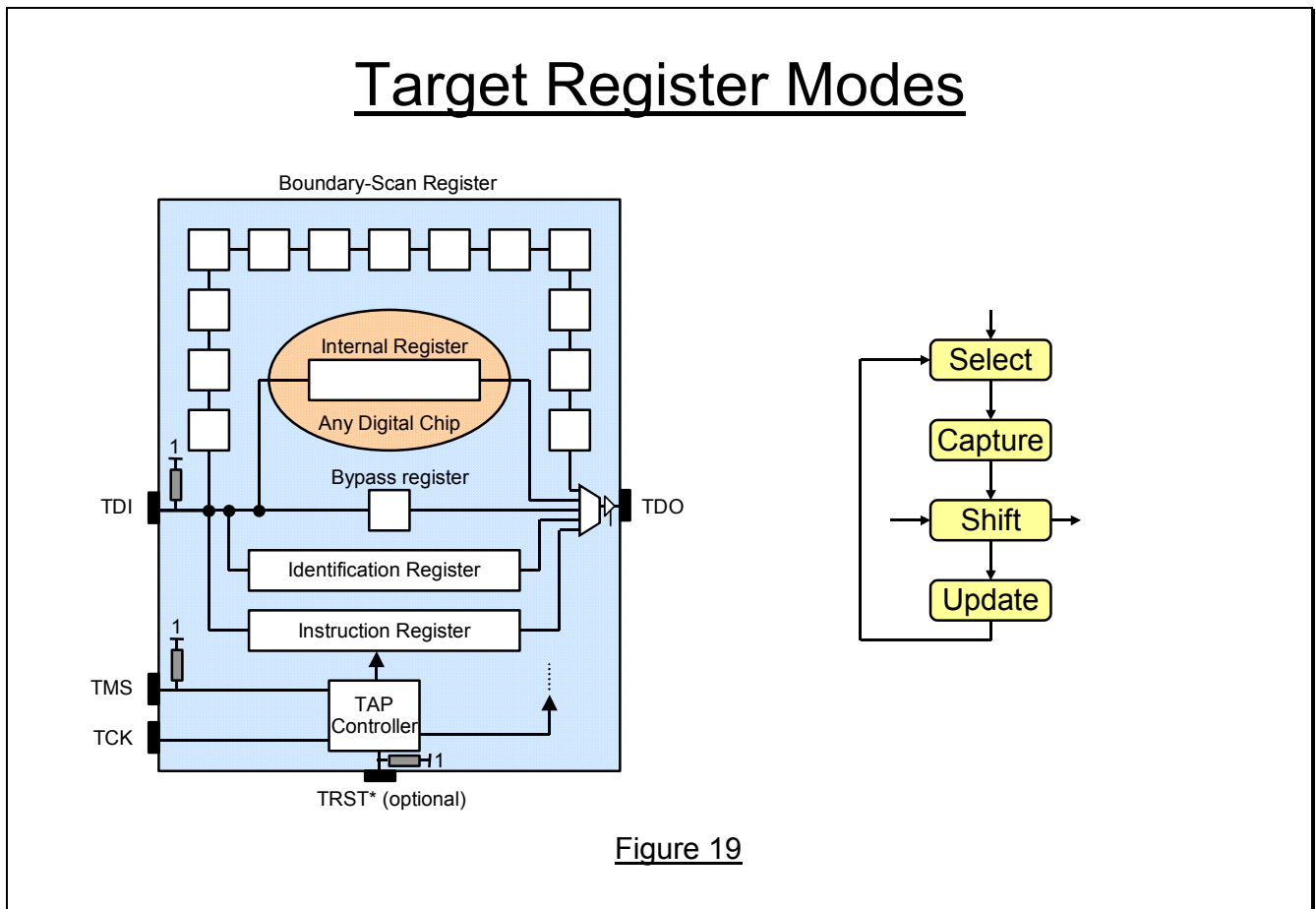


Before we look closer at each part of this architecture there are two general points to note about Figure 18:

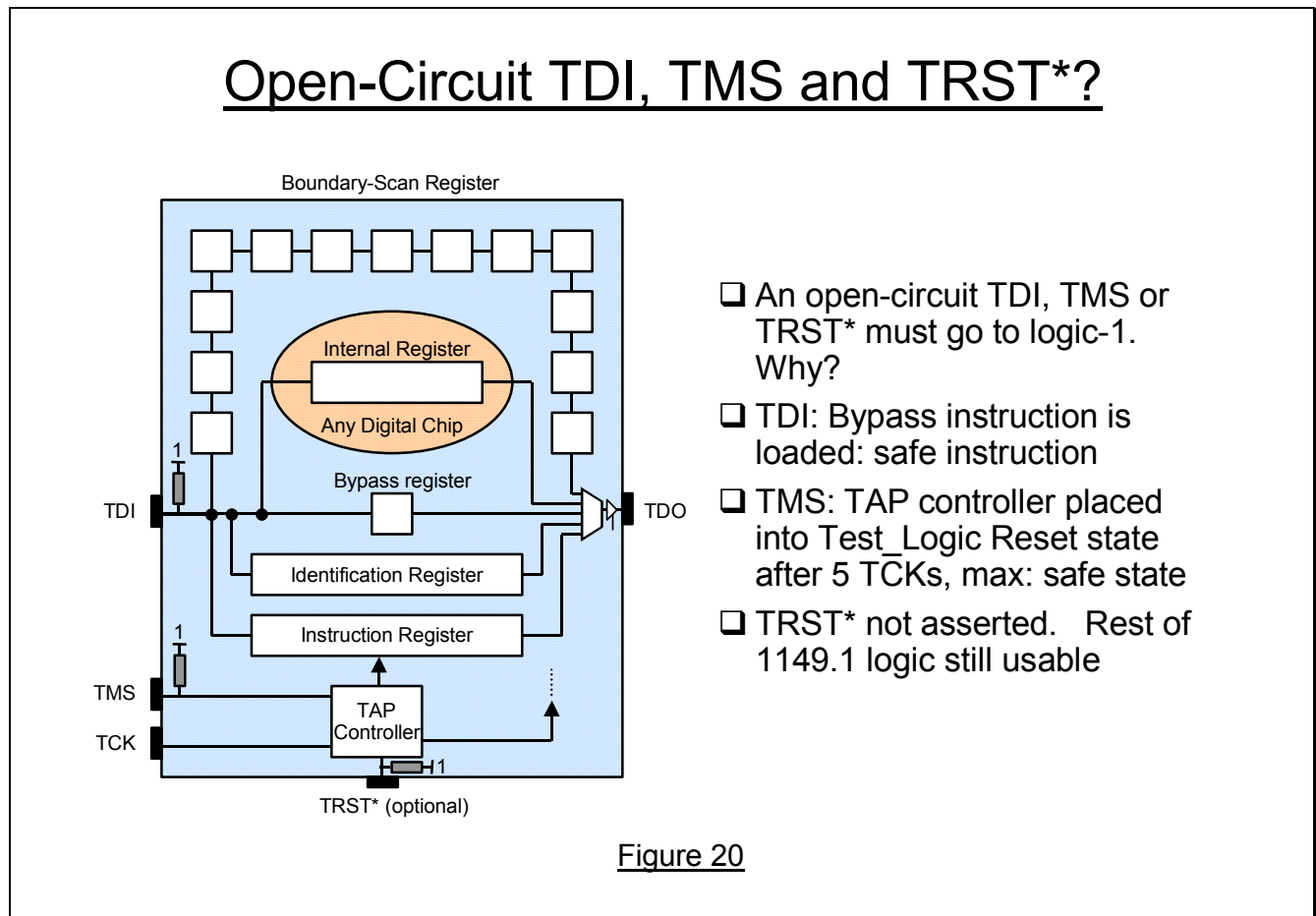
Point 1. Since 1149.1-2001, the latest version of the Standard, there are only four mandatory instructions: **Extest**, **Bypass**, **Sample** and **Preload**.

Point 2. The asynchronous Reset signal, **TRST\***, is optional. If present, the signal is active low. If not present, there is always a synchronous reset available within the TAP controller. If TMS is held at logic 1, a maximum of five consecutive TCKs is guaranteed to return the TAP controller to the reset state of **Test\_Logic\_Reset**. This will be referred to as the **TMS = 1, 5 x TCK** synchronous reset.

## Target Register Modes



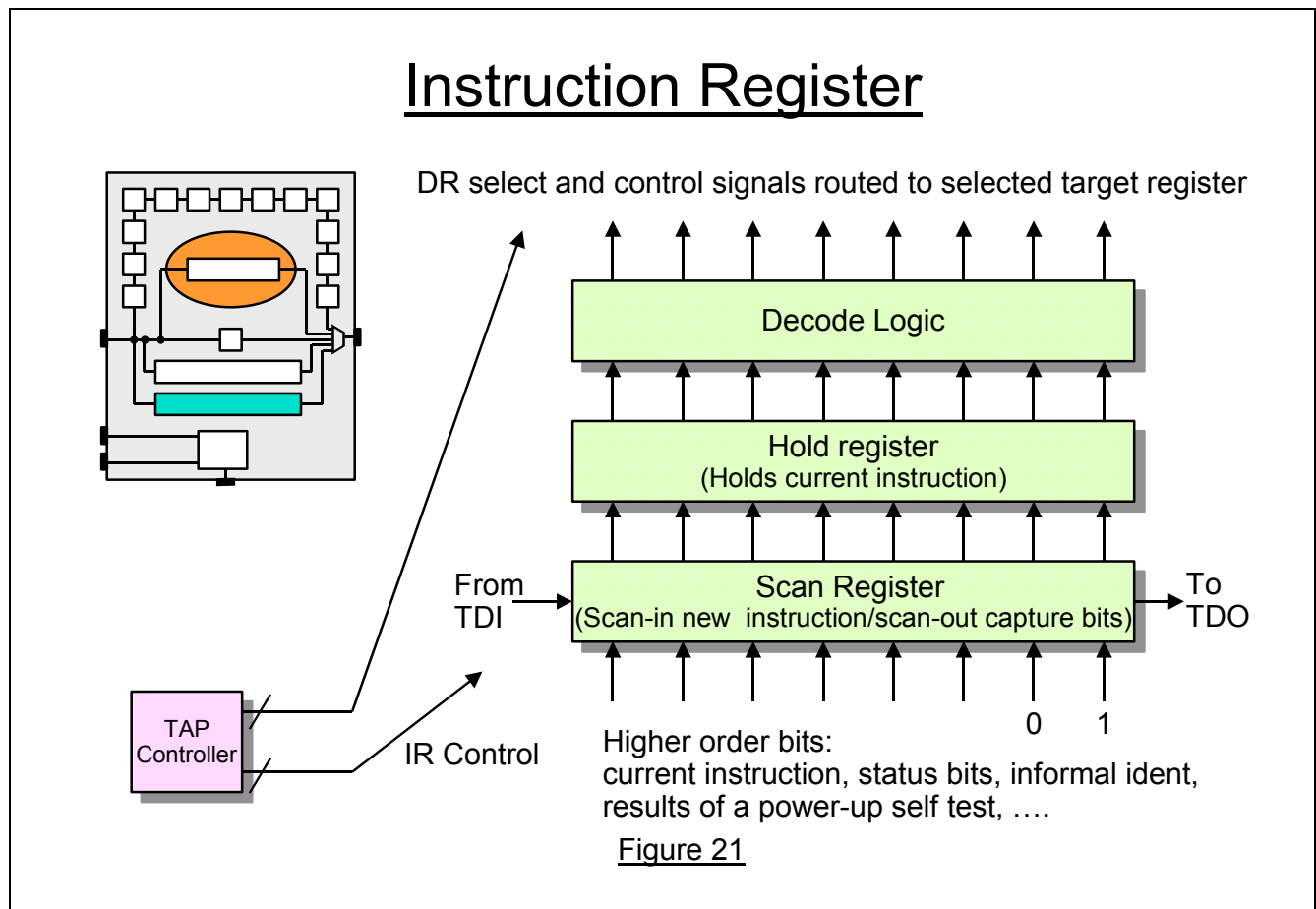
Whenever a register is selected to become active between TDI and TDO, it is always possible to perform three operations on the register: parallel **Capture** followed by serial **Shift** followed by parallel **Update**. The order of these operations is fixed by the state-sequencing design of the TAP controller. For some target Data registers, some of these operations will be effectively null operations, *no ops*.

**Open Circuit TDI, TMS and TRST\*?**

The 1149.1 Standard mandates that an open circuit TDI, TMS or TRST\* input must go to logic 1. This can be achieved with internal weak resistive pull ups, or with active transistor pull ups. The reasons are as follows:

- For TDI. If the Instruction register is selected as the target register between TDI and TDO ready to be loaded with a new instruction, then a safe instruction (**Bypass**, all-1s code) is loaded and executed into the device with the open-circuit TDI and to all devices downstream of this device.
- For TMS. In a maximum of 5 x TCK cycles, the TAP controller of this device will be placed into its **Test\_Logic Reset** state. In this state, the boundary-scan logic is inactive but the device can continue to operate functionally.
- For TRST\*, logic 1 is the inactive state and so the device is not prevented from being used either in functional mode or in 1149.1 modes. The device must be reset with the synchronous reset cycle (**TMS = 1, 5 x TCK**) rather than through its asynchronous TRST\* signal.

## The Instruction Register

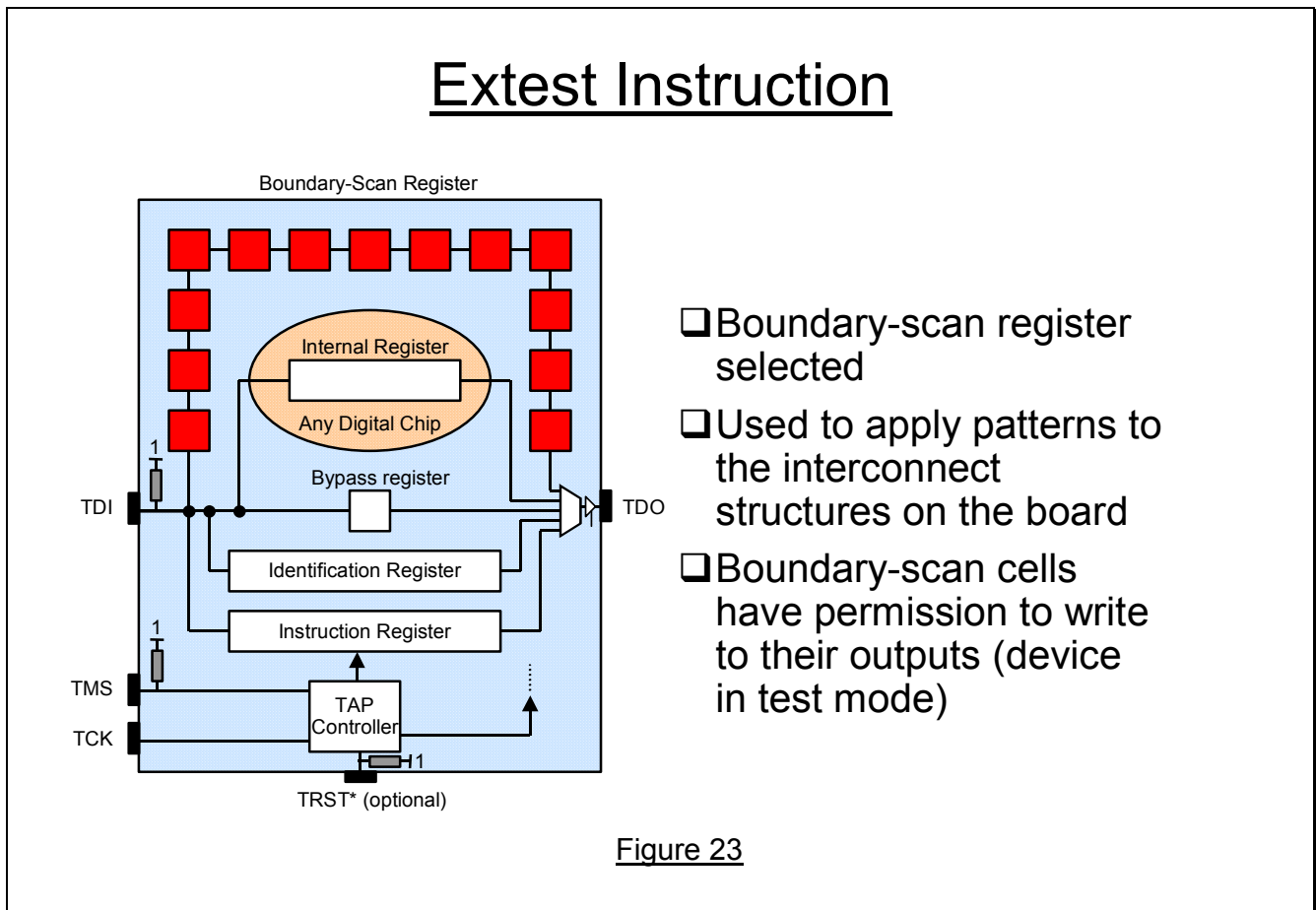


An **Instruction** register (Figure 21) has a shift **scan** section that can be connected between TDI and TDO, and a hold section that holds the current instruction. There may be some **decoding logic** beyond the hold section depending on the width of the register and the number of different instructions. The control signals to the Instruction register originate from the TAP controller and either cause a **shift-in/shift-out** through the Instruction register shift section, or cause the contents of the shift section to be passed across to the hold section (parallel **Update** operation). It is also possible to load (**Capture**) internal hard-wired values into the shift section of the Instruction register. The Instruction register must be at least two-bits long to allow coding of the four mandatory instructions — **Extest, Bypass, Sample, Preload** — but the maximum length of the Instruction register is not defined. In capture mode, the two least significant bits must capture a **01** pattern. (Note: by convention, the least-significant bit of any register connected between the device TDI and TDO pins, is always the bit closest to TDO.) The values captured into higher-order bits of the Instruction register are not defined in the Standard. One possible use of these higher-order bits is to capture an informal identification code if the optional 32-bit Identification register is not implemented. In practice, the only mandated bits for the Instruction register capture is the **01** pattern in the two least-significant bits. We will return to the value of capturing this pattern later in the tutorial.

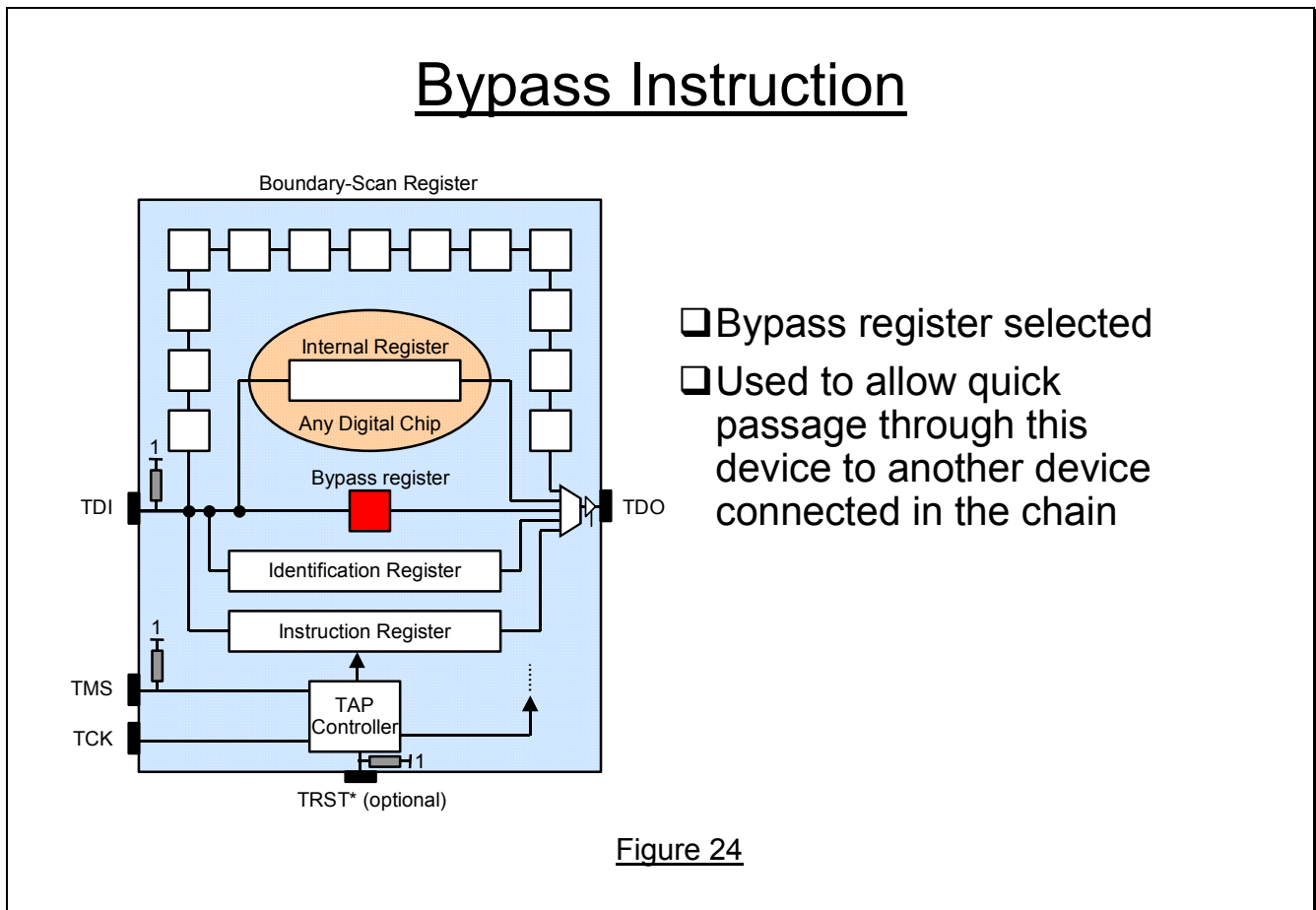
**The Standard Instructions**

<u>Standard Instructions</u>	
Instruction	Selected Data Register
Mandatory: <u><b>Extest</b></u> <u><b>Bypass</b></u> <u><b>Sample</b></u> <u><b>Preload</b></u>	Boundary scan (formerly all-0s code) Bypass (initialised state, all-1s code) Boundary scan (device in functional mode) Boundary scan (device in functional mode)
Optional: <u><b>Intest</b></u> <u><b>Idcode</b></u> <u><b>Usercode</b></u> <u><b>Runbist</b></u> <u><b>Clamp</b></u> <u><b>HighZ</b></u>	Boundary scan Identification (initialised state if present) Identification (for PLDs) Result register Bypass (output pins in safe state) Bypass (output pins in high-Z state)
NB. All unused instruction codes must default to <u><b>Bypass</b></u>	
<u>Figure 22</u>	

The IEEE 1149.1 Standard describes four mandatory instructions: **Extest**, **Bypass**, **Sample**, and **Preload**, and six optional instructions: **Intest**, **Idcode**, **Usercode**, **Runbist**, **Clamp** and **HighZ**. These ten instructions are known as the **public** instructions. We will look first at the mandatory instructions.

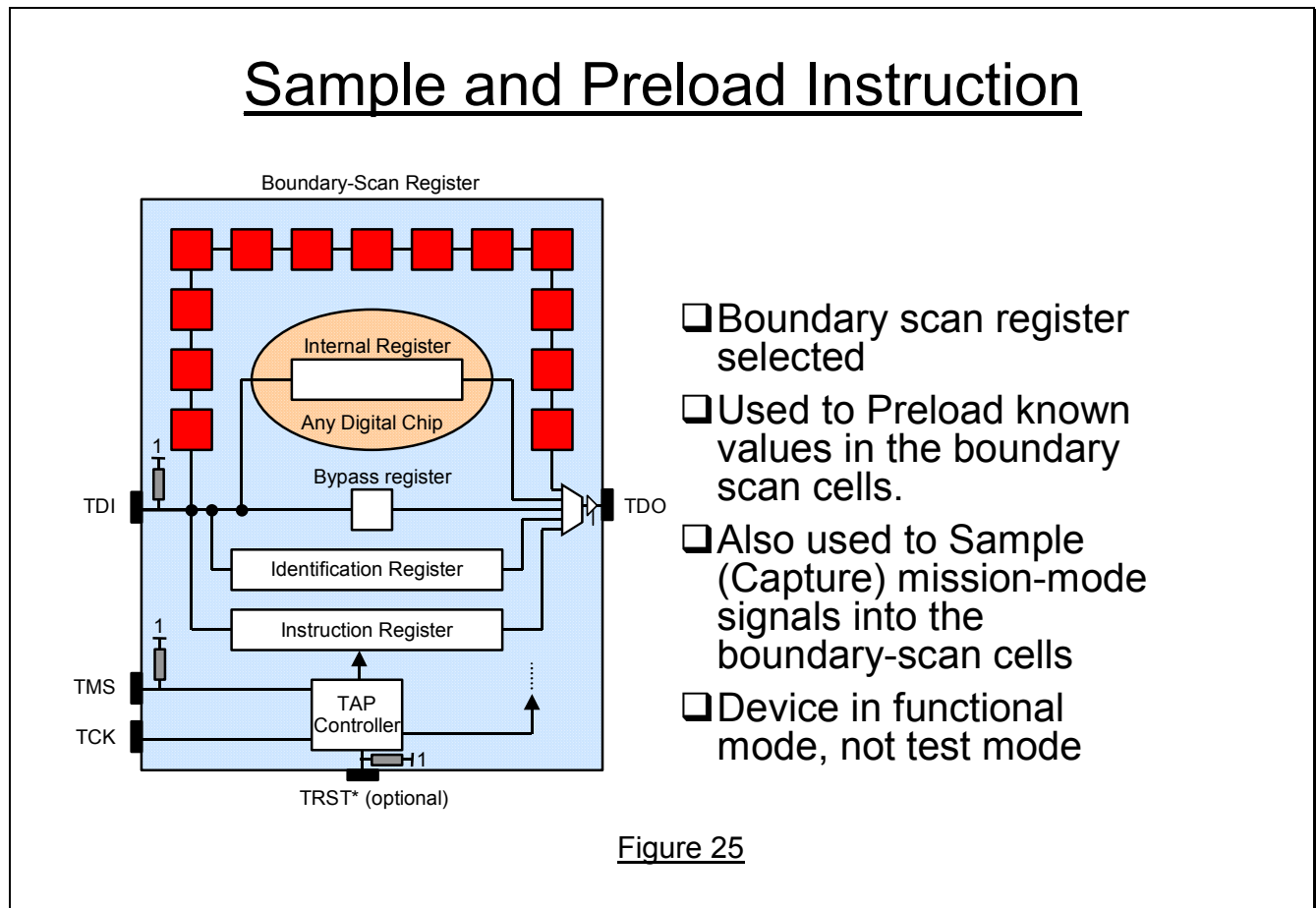
**Extest Instruction**

The **Extest** instruction selects the boundary-scan register when executed, preparatory to interconnect testing. The code for **Extest** used to be defined to be the all-0s code. This requirement has been relaxed in the 2001 version of the Standard.

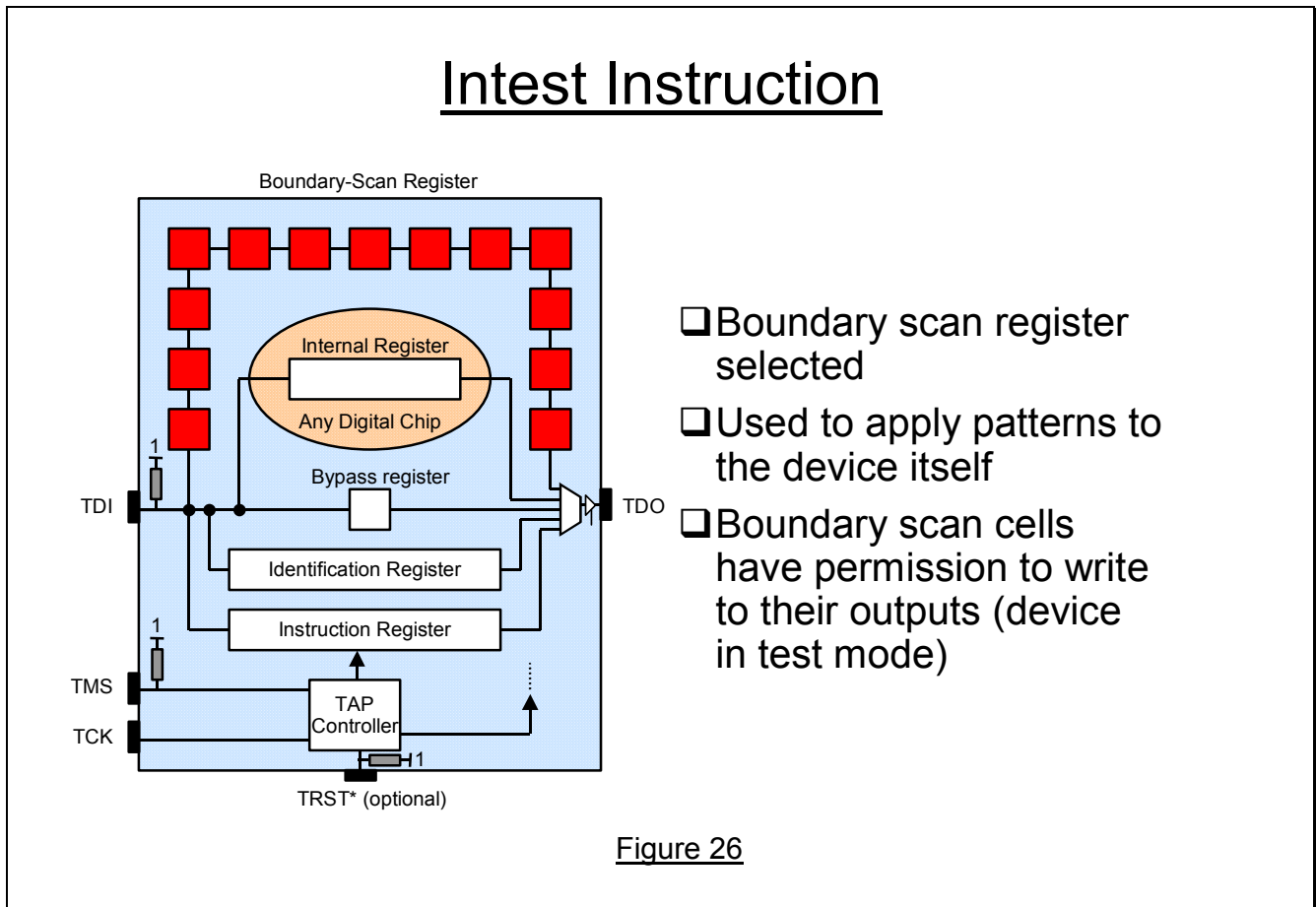
**Bypass Instruction**

The **Bypass** instruction must be assigned an all-1s code and when executed, causes the Bypass register to be placed between the TDI and TDO pins.

## Sample and Preload Instructions

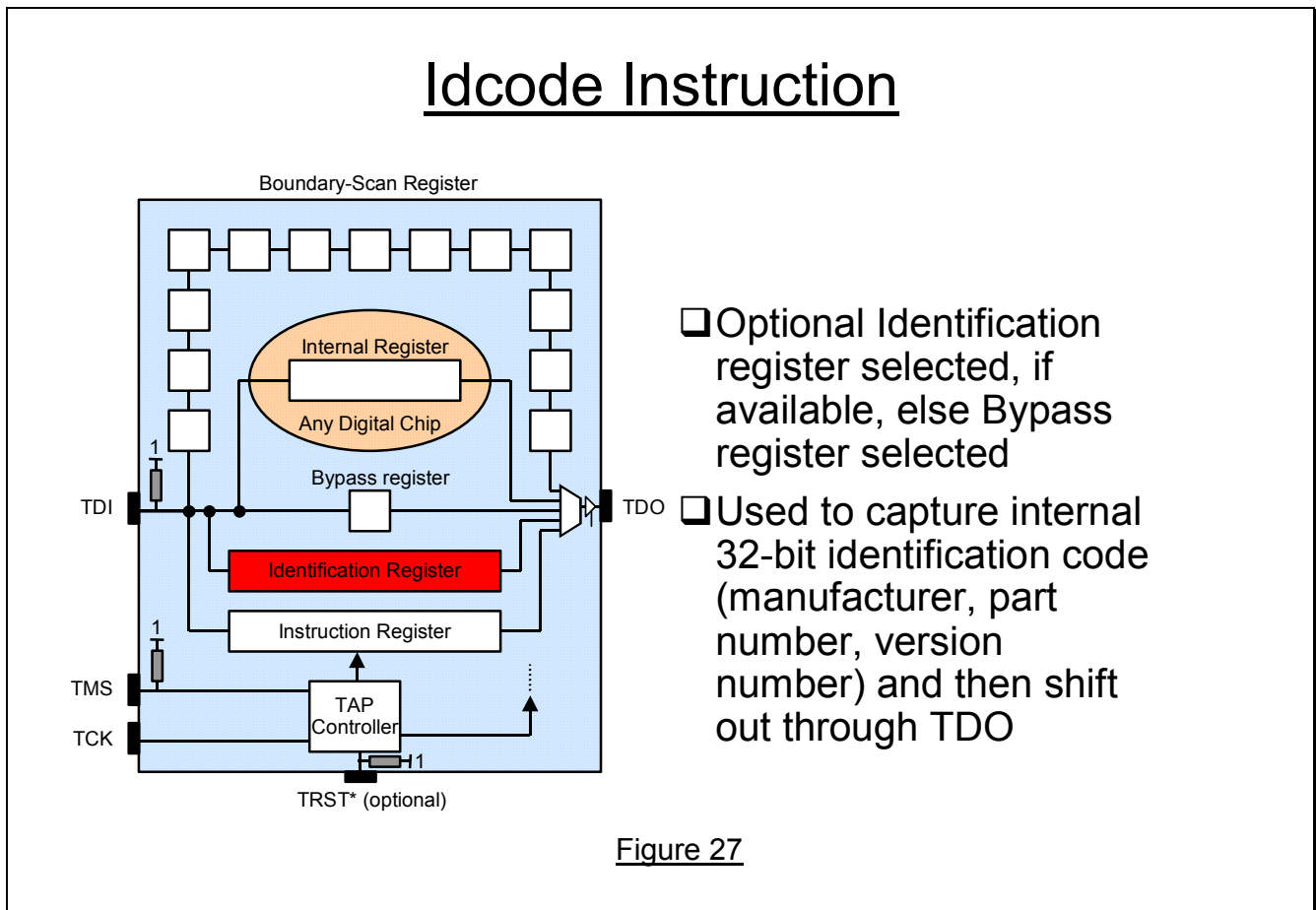


The **Sample** and **Preload** instructions, and their predecessor the **Sample/Preload** instruction, selects the Boundary-Scan register when executed. The instruction sets up the boundary-scan cells either to **sample** (capture) values or to **preload** known values into the boundary-scan cells prior to some follow-on operation.

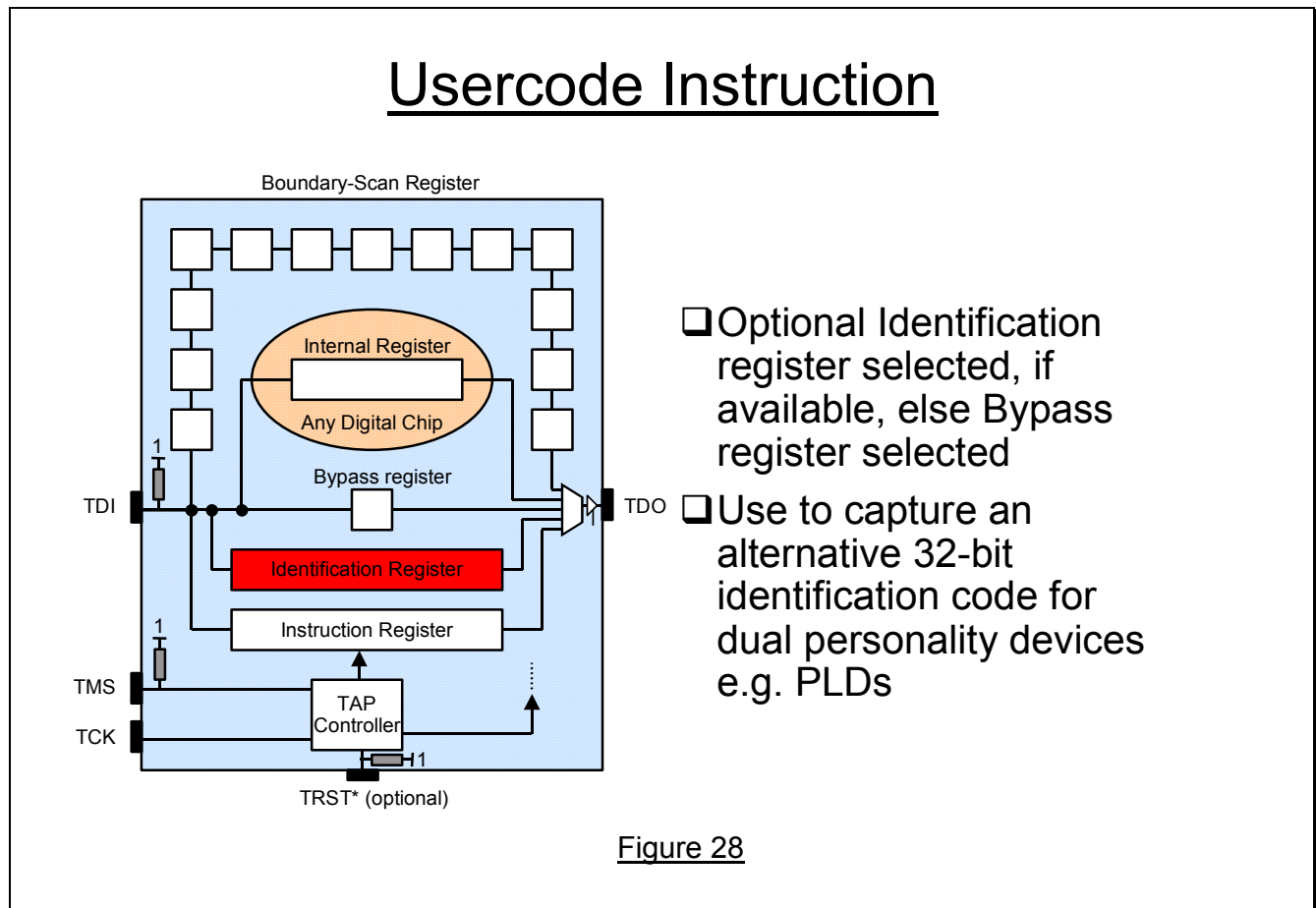
**Intest Instruction**

The IEEE 1149.1 Standard defines a number of optional instructions – that is, instructions that do not need to be implemented but which have a prescribed operation if they are implemented.

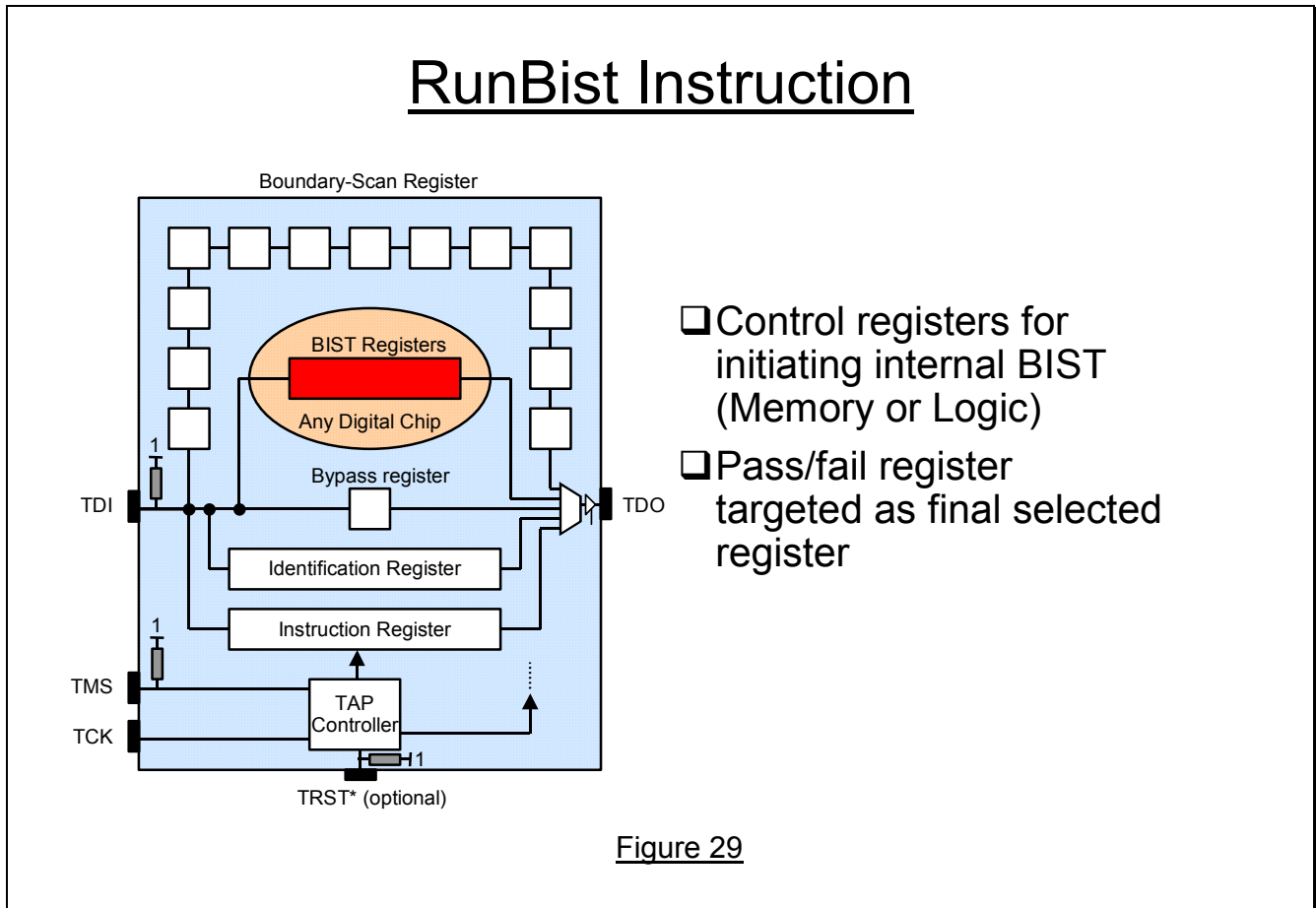
The instruction illustrated here is **Intest**, the instruction that selects the boundary-scan register preparatory to applying tests to the internal logic of the device.

**Idcode Instruction**

**Idcode** is the instruction to select the **Identification** register between TDI and TDO, preparatory to loading the internally-held 32-bit identification code and reading it out through TDO. The 32 bits are used to identify the manufacturer of the device, its part number and its version number.

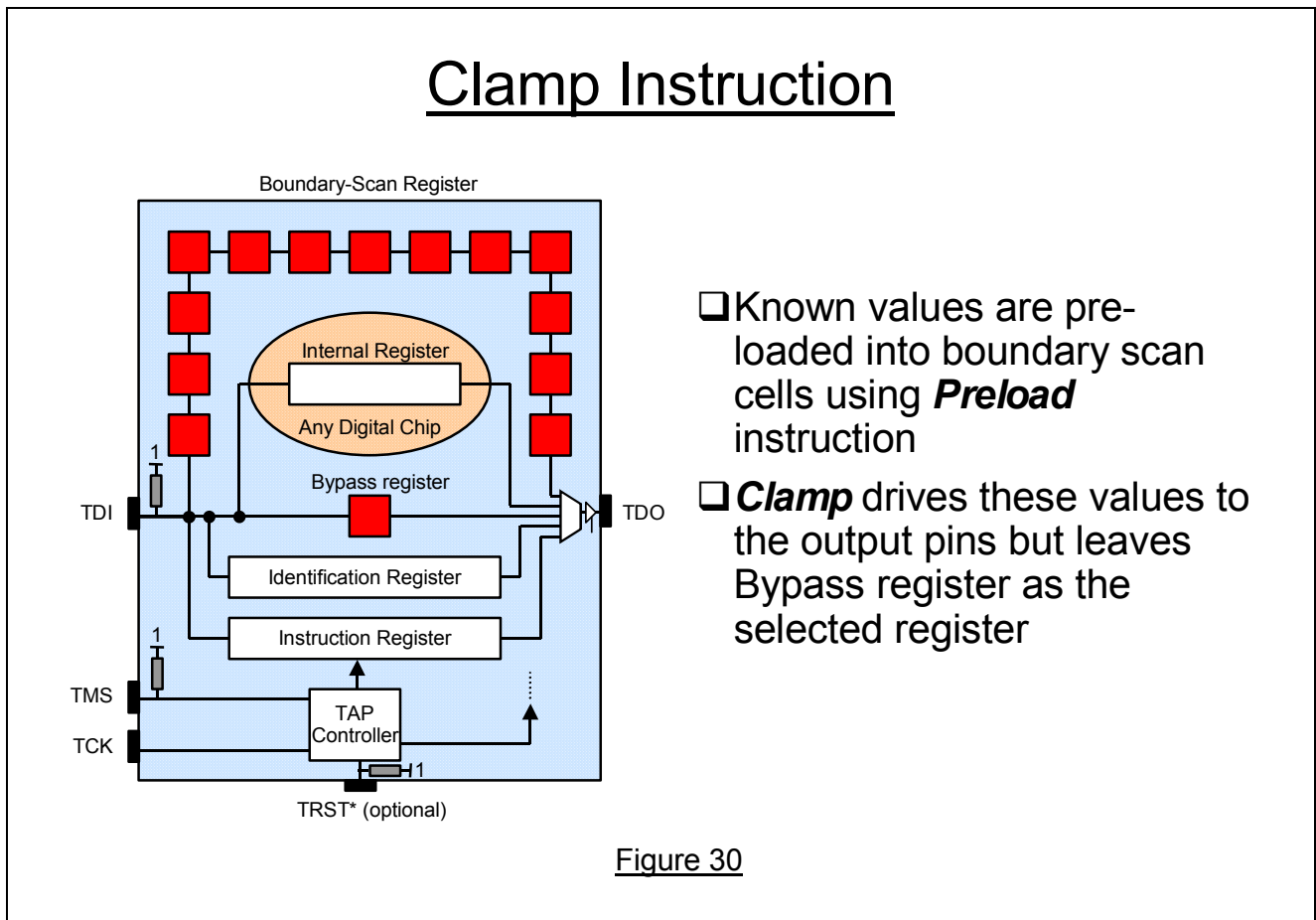
**Usercode Instruction**

**Usercode** selects the same 32-bit register as **Idcode**, but allows an alternative 32 bits of identity data to be loaded and serially shifted out. This instruction is used for dual-personality devices, such as **Complex Programmable Logic Devices** and **Field Programmable Gate Arrays**.

**RunBist Instruction**

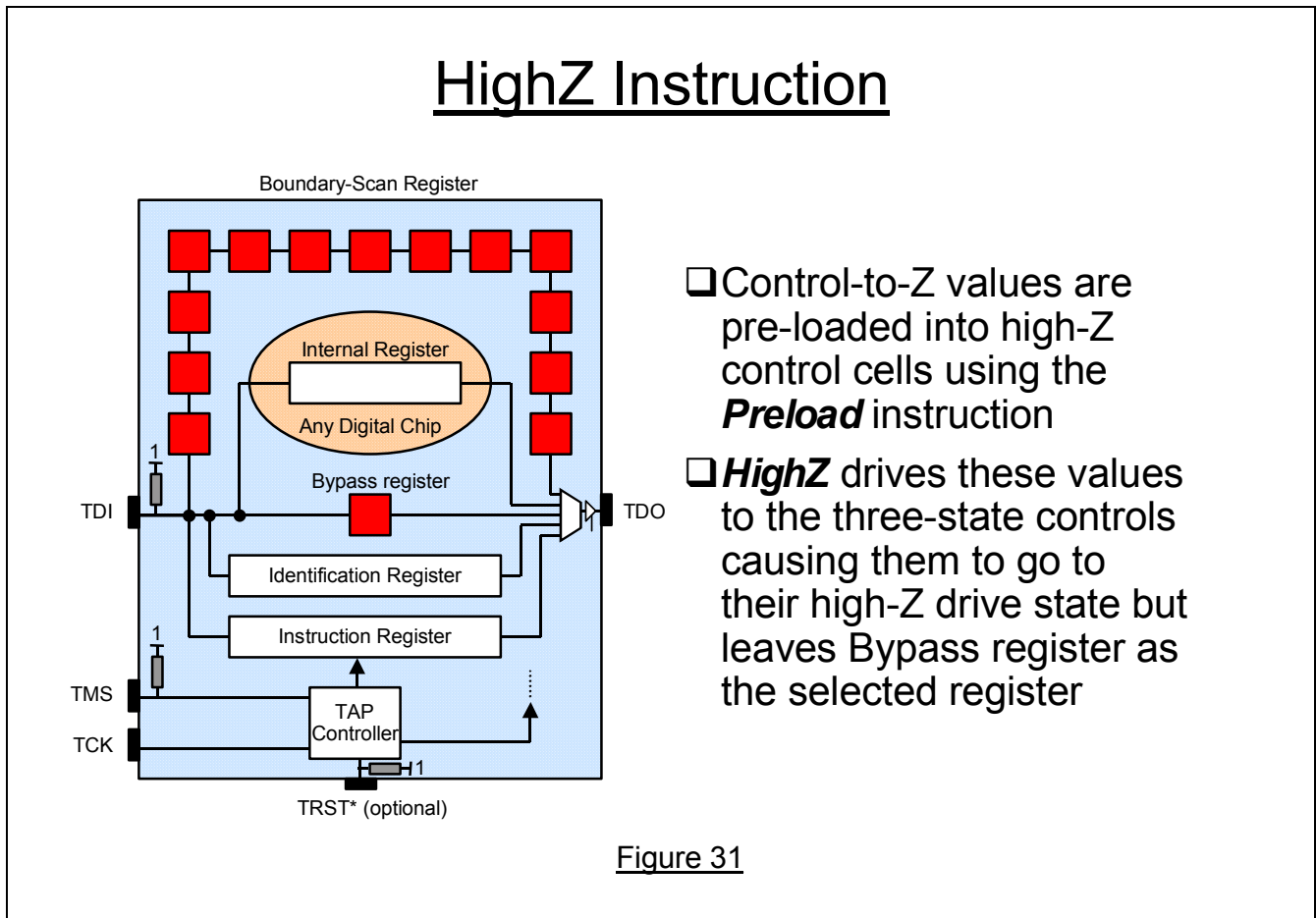
An important optional instruction is **RunBist**. Because of the growing importance of internal self-test structures, the behavior of **RunBist** is defined in the Standard. The self-test routine must be self-initializing (i.e., no external seed values are allowed), and the execution of **RunBist** essentially targets a self-test result register between TDI and TDO. At the end of the self-test cycle, the targeted data register holds the **Pass/Fail** result.

## Clamp Instruction



Two new instructions introduced in the 1993 revision, 1149.1a-1993, were **Clamp** and **HighZ**. **Clamp** is an instruction that uses boundary-scan cells to drive preset values established initially with the **Preload** instruction onto the outputs of devices, and then selects the Bypass register between TDI and TDO (unlike the **Preload** instruction which leaves the device with the boundary-scan register still selected until a new instruction is executed or the device is returned to the **Test\_Logic Reset** state).

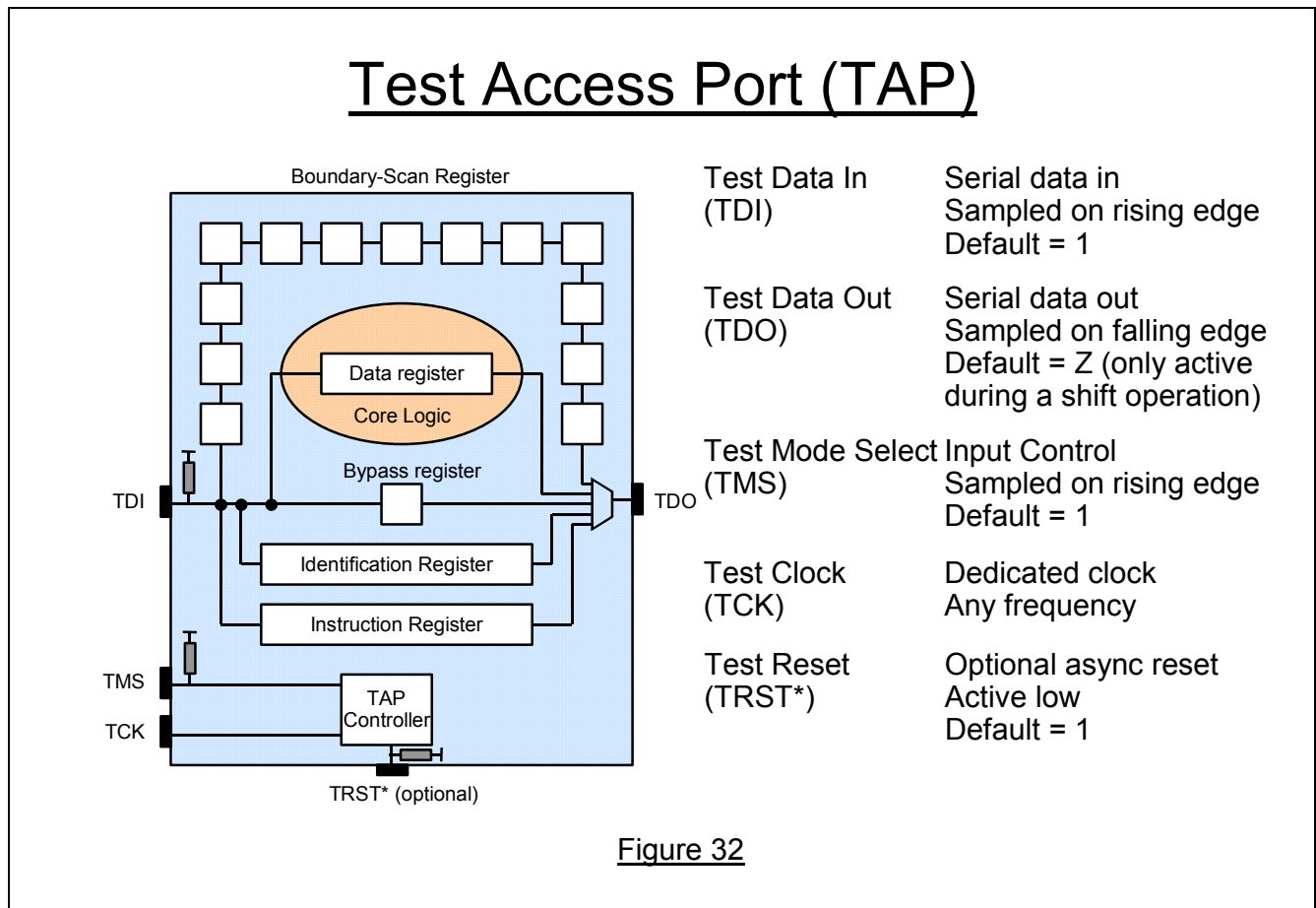
**Clamp** would be used to set up safe **guarding** values on the outputs of certain devices in order to avoid bus contention problems, for example.

**HighZ Instruction**

**HighZ** is similar to **Clamp**, but it leaves the device output pins in a high-impedance state rather than drive fixed logic-1 or logic-0 values. **HighZ** also selects the Bypass register between TDI and TDO.

## The Test Access Port (TAP)

### The TAP Signals



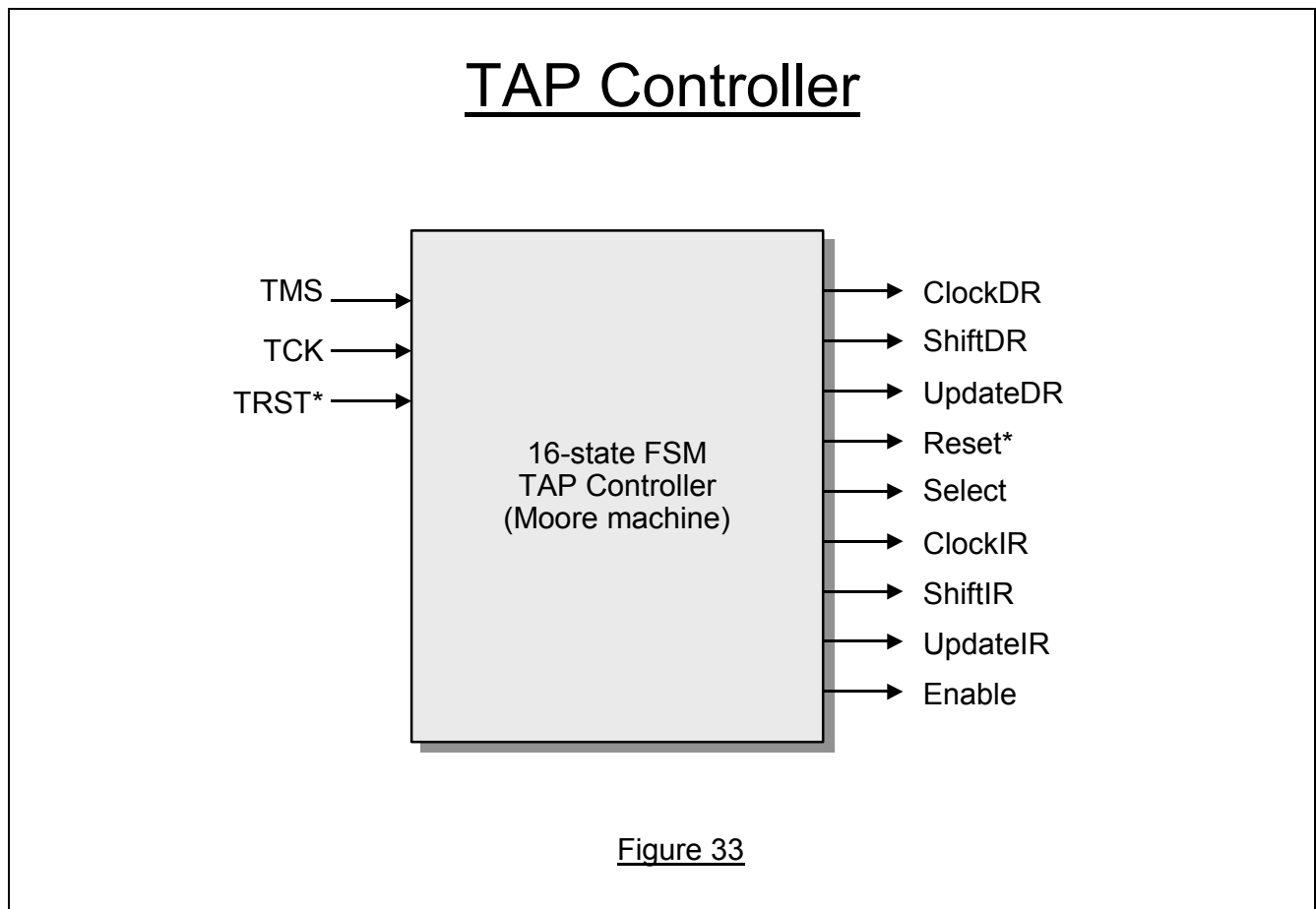
We turn now to the **Test Access Port (TAP)** and its **Controller**. The TAP consists of four mandatory terminals plus one optional terminal.

The mandatory terminals are:

- ❑ Test Data In (TDI): serial test data in with a default value of 1.
- ❑ Test Data Out (TDO): serial test data out with a default value of Z and only active during a shift operation.
- ❑ Test Mode Select (TMS): serial input control signal with a default value of 1.
- ❑ Test Clock (TCK): dedicated test clock, any convenient frequency (usually determined by the maximum TCK frequency of the external tester and the slowest boundary-scan device on the board).

The optional terminal is:

- ❑ Test Reset (TRST\*): asynchronous TAP controller reset with default value of 1 and active low.

**Top-Level View of the TAP Controller**

TMS and TCK (and the optional TRST\*) go to a 16-state finite-state machine controller, which produces the various control signals. These signals include dedicated signals to the Instruction register (***ClockIR***, ***ShiftIR***, ***UpdateIR***) and generic signals to all data registers (***ClockDR***, ***ShiftDR***, ***UpdateDR***). The data register that actually responds is the one enabled by the conditional control signals generated at the parallel outputs of the Instruction register, according to the particular instruction.

The other signals, ***Reset***, ***Select*** and ***Enable*** are distributed as follows:

- ***Reset*** is distributed to the Instruction register and to the target **Data Register**
- ***Select*** is distributed to the output multiplexer
- ***Enable*** is distributed to the output driver amplifier

Note: the Standard uses the term **Data Register** to mean any target register except the Instruction register

## The TAP Controller

### TAP Controller State Diagram

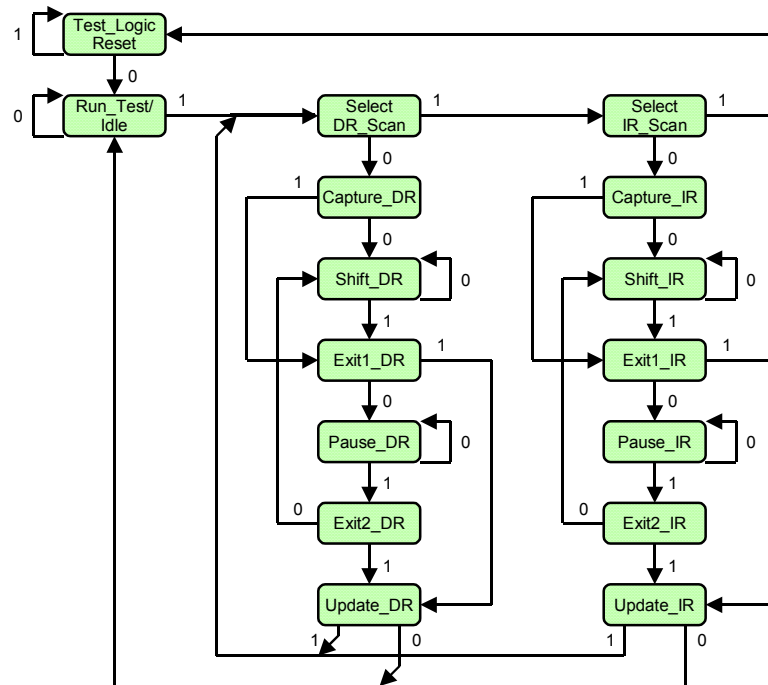


Figure 34

Figure 34 shows the 16-state state table for the TAP controller. The value on the state transition arcs is the value of TMS. A state transition occurs on the positive edge of TCK and the controller output values change on the negative edge of TCK.

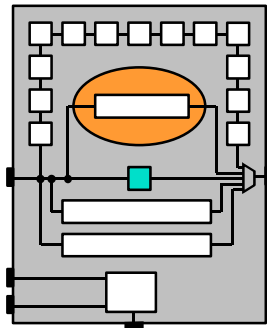
The TAP controller initializes in the **Test\_Logic\_Reset** state (“**Asleep**” state). While TMS remains a 1 (the default value), the state remains unchanged. In the **Test\_Logic\_Reset** state and the active (selected) register is determined by the contents of the Hold section of the Instruction register. The selected register is either the Identification register, if present, else the Bypass register. Pulling TMS low causes a transition to the **Run\_Test/Idle** state (“**Awake, and do nothing**” state). Normally, we want to move to the **Select IR\_Scan** state ready to load and execute a new instruction.

An additional 11 sequence on TMS will achieve this. From here, we can move through the various **Capture\_IR**, **Shift\_IR**, and **Update\_IR** states as required. The last operation is the **Update\_IR** operation and, at this point, the instruction loaded into the shift section of the Instruction register is transferred to the Hold section of the Instruction register to become the new current instruction – refer again to Figure 21. This causes the Instruction register to be de-selected as the register connected between TDI and TDO and the Data register identified by the new current instruction to be selected as the new target Data register between TDI and TDO. For example, if the instruction is **Bypass**, the Bypass register becomes the selected data register. From now on, we can manipulate the target data register with the generic **Capture\_DR**, **Shift\_DR**, and **Update\_DR** control signals.

## The Bypass and Identification Registers

### The Bypass register

### Bypass register



- ❑ One-bit shift register, selected by the ***Bypass*** instruction
- ❑ Captures a hard-wired 0
- ❑ Note: in the *Test-Logic/Reset* state, the Bypass register is the default register if no Identification Register present

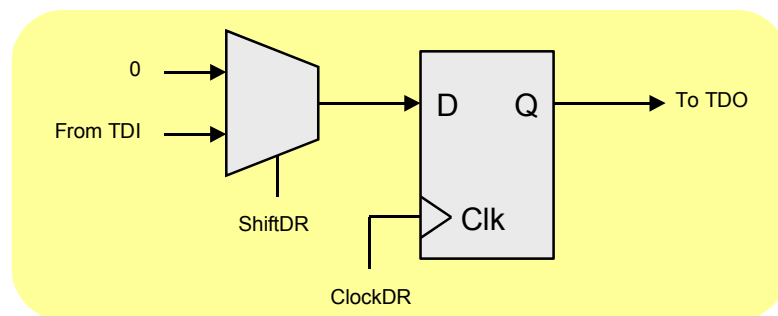
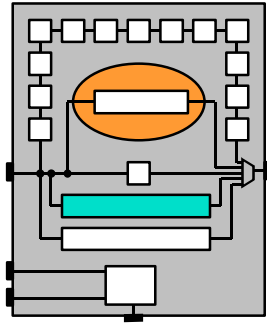


Figure 35

Figure 35 shows a typical design for a Bypass register. It is a 1-bit register, selected by the ***Bypass*** instruction and provides a basic serial-shift function. There is no parallel output (which means that the ***Update\_DR*** control has no effect on the register), but there is a defined effect with the ***Capture\_DR*** control — the register captures a hard-wired value of logic 0.

## The Identification Register

### Identification Register



- ❑ 32-bit shift register
- ❑ Selected by ***Idcode*** and ***Usercode*** instruction
- ❑ No parallel output
- ❑ Captures a hard-wired 32-bit word
- ❑ Main function: identify device owner and part number
- ❑ Note: ***Idcode*** is power-up instruction if Identification Register is present, else ***Bypass***

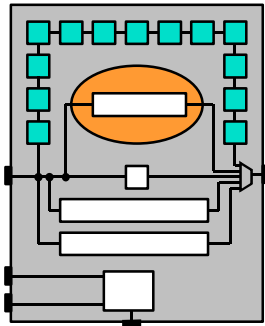
Figure 36

The optional Identification register (Figure 36) is a 32-bit register with capture and shift modes of operation. The register is selected by the ***Idcode*** and ***Usercode*** instructions and the 32-bits of internal data are loaded into the shift part of the register and scanned out through the device TDO pin. Recall also that this register, if present, is the selected active register when the TAP controller is in the ***Test\_Logic\_Reset*** state, else the Bypass register is selected in this state..

## The Boundary-Scan Register

### *Where to Place Boundary-Scan Cells?*

## Boundary-Scan Register



❑ Shift register with boundary-scan cells on:

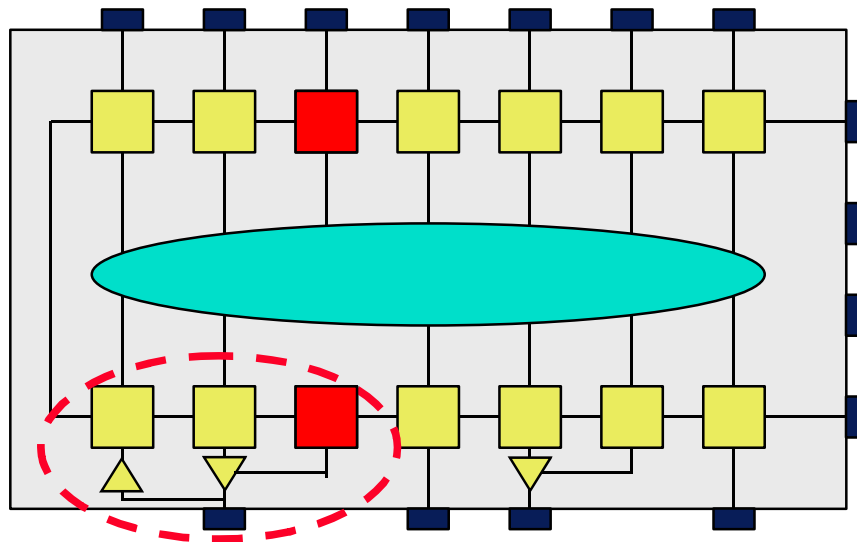
- device input pins
- device output pins
- control of three-state outputs
- control of bidirectional cells

❑ Selected by the ***Exttest***, ***Intest***, ***Preload*** and ***Sample*** instructions

Figure 37

We are now ready to take a more detailed look at the boundary-scan cells and their concatenation into a general-purpose boundary-scan register. For a given device, boundary-scan cells are placed on the device digital input ports, digital output ports, and on the control lines of bidirectional (IO) ports and tristate (0Z) ports. The scan cells are linked together to form the boundary-scan register. The order of linking within the device is determined by the physical adjacency of the pins and/or by other layout constraints. The boundary-scan register is selected by the ***Exttest***, ***Sample***, ***Preload***, and ***Intest*** instructions.



**Bidirectional Pins**Providing Boundary-Scan Cells: IO

On control of bidirectional IO:  
dual-mode input signal

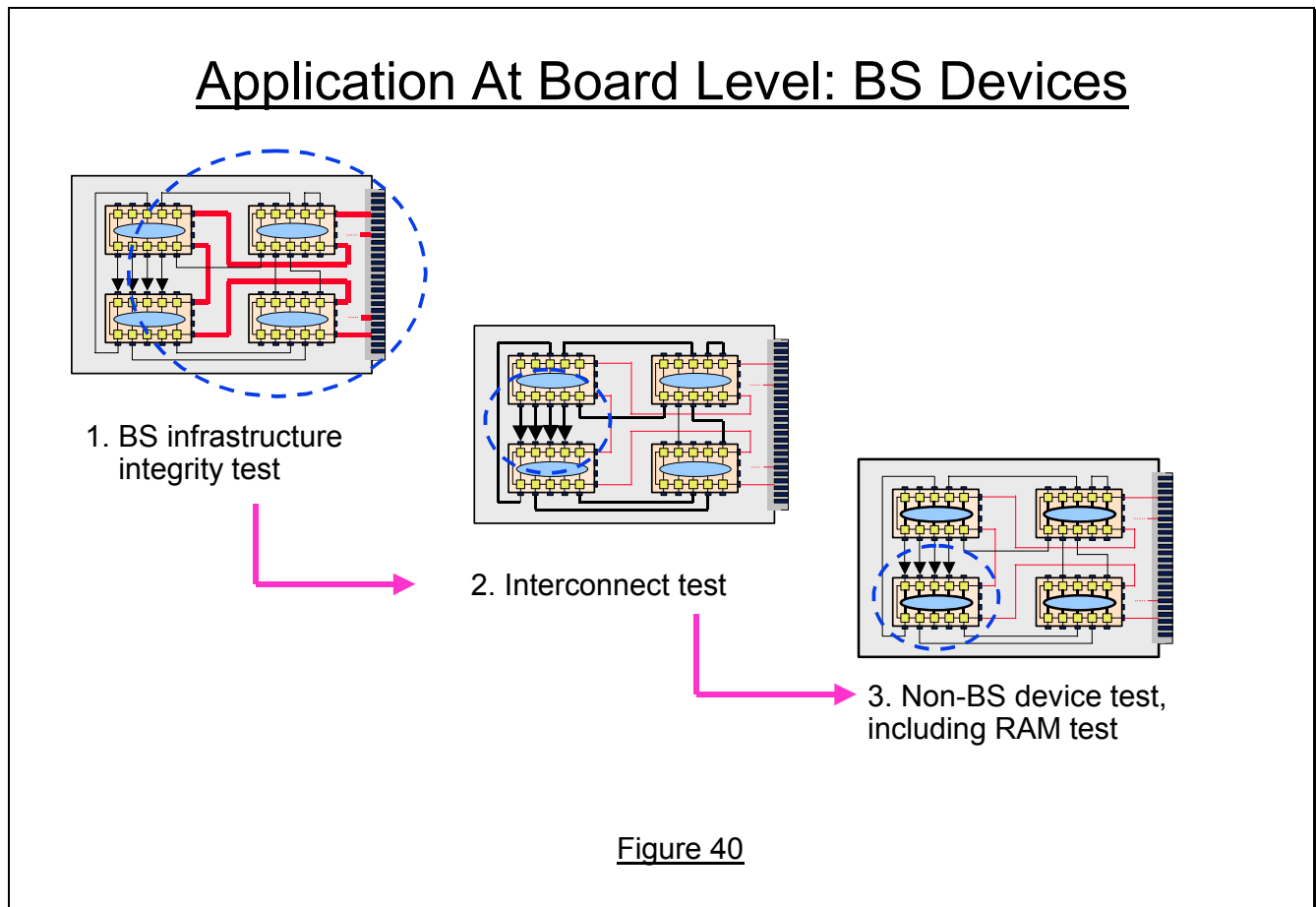
Figure 39

Figure 39 shows the set up for a bidirectional **IO** pin. Here, we see that, conceptually at least, three boundary-scan cells are required: one on the input side, one on the output side, and one to allow control of the **IO** status. In practice, the two IO scan cells are usually combined into a single multi-function cell called a **BC\_7**.

This is the end of the part of the tutorial that has mostly concentrated on the device-level features of an 1149.1-compliant device. From now on, we will develop the board-level application of 1149.1, starting with the interconnect test-pattern generation process.

## Application at the Board Level

### General Strategy



For the remainder of this tutorial, we will concentrate on the application of boundary scan at the board level. First, we will look at the major stages of board-test strategy for a board populated just by IEEE 1149.1-compliant devices (a “pure” boundary-scan board). Later, we will consider the more-realistic situation of a board containing both boundary-scan devices and non-boundary-scan devices.

A general-purpose strategy for testing a boundary-scan board is:

- Step 1. Carry out a board-level boundary-scan infrastructure test. One way to do this is to select the Instruction register and then cycle through the **Capture\_IR** and **Shift\_IR** operations to load and shift the **01** checkerboard values built in to the Instruction register. Further optional infrastructure tests can be carried out if time permits.
- Step 2. Use the **Extest** instruction to select the boundary-scan registers to apply stimulus and capture responses across the interconnect structures between the boundary-scan devices on the board. Take care not to damage non-boundary-scan devices attached to these interconnects.

Step 3. Apply tests to the non-boundary-scan devices, such as memory devices or unstructured clusters, that can be accessed from the boundary-scan registers of the boundary-scan devices.

At the end of Step 1, we have **tested the tester**. At the end of Step 2, we have tested the **pure** boundary-scan to boundary-scan region i.e. the region most susceptible to assembly damage caused by electrical, mechanical, or thermal stress. Step 2 is the major application of the boundary-scan structures and we will take a close look at the tests necessary to find open and shorts on the interconnects.

Step 3 takes us into the non-boundary-scan region and as we will see, this is the most complex region to test. For now, we will focus on Step 2 but, first, we must consider how to model the manufacturing defects to determine the objectives and efficiency of the test patterns.

### **Modeling Board Defects**

## Board Defects

- Missing component, wrong component, mis-oriented component, broken track, shorted tracks, pin-to-solder open circuit, pin-to-pin solder shorts
- Number of 2-net short circuit faults between  $k$  interconnects =  $k(k-1)/2$
- Equivalent fault models for shorts: bridging of type wired-AND and wired-OR
- Open circuits are modelled down-stream as stuck-at-1 or stuck-at-0 faults

Figure 41

The major causes of board manufacturing defects are: missing components, wrong components, mis-oriented components, broken track (opens), shorted tracks (track-to-track shorts), pin-to-solder opens, pin-to-pin solder shorts. To consider the shorts, we assume that the behaviour of any short is logical i.e. the short behaves as if it were an unwanted wired-AND gate (strong 0, weak 1) or wired-OR gate (strong 1, weak 0). This is a realistic assumption given that the circuits receiving the result of a short are themselves digital and therefore will digitize the analog outcome of the short circuit.

There are many opportunities for short circuits: track-to-track, pin-to-pin, etc., but we will restrict attention to all possible 2-net shorts, where a **net** starts and finishes at a device pin and therefore includes both the pins and the track (interconnect) in between the pins. Recall that the number of 2-net short circuit faults between  $k$  interconnects =  $k(k-1)/2$ . In practice, we realize that not all possible 2-net shorts can exist but, as we will see, it is so easy to target all 2-net shorts that we do not worry about the unrealistic shorts. Note also that targeting all 2-net shorts means that we are also targeting all 3-net shorts, all 4-net shorts, etc. Any short greater than 2-net can be considered to be a collection of associated 2-net shorts i.e. a superset of the component 2-net shorts.

Because of the digital nature of the sensors, open circuits are modeled down-stream as boundary-scan inputs that are either stuck-at-1 or stuck-at-0. Because we are not sure which way the open-circuit will behave, we will target both polarities of stuck-at faults.

### ***Interconnect Test Patterns***

## Faults on a 4-Net Interconnect

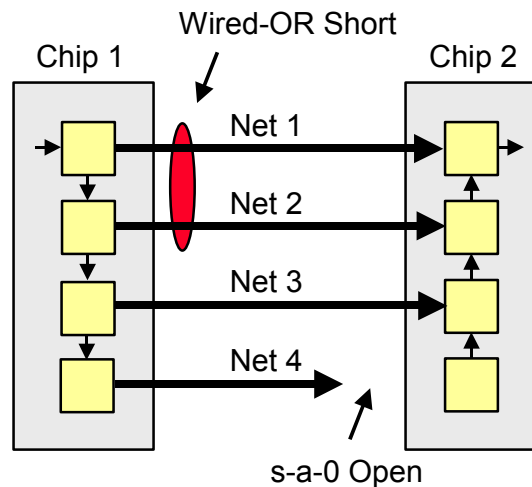


Figure 42

Consider the simple four-net interconnect structure shown in Figure 42. Assume both devices are IEEE 1149.1 compliant and the left-hand driver boundary-scan cells (Chip 1) drives values into the right-hand sensor boundary-scan cells (Chip 2). Assume further that there is an unwanted short-circuit defect between Nets 1 and 2, and an unwanted open-circuit defect along Net 4. How can we test for such defects?

**Opens and Shorts Test**

## Generating Open & Short Tests

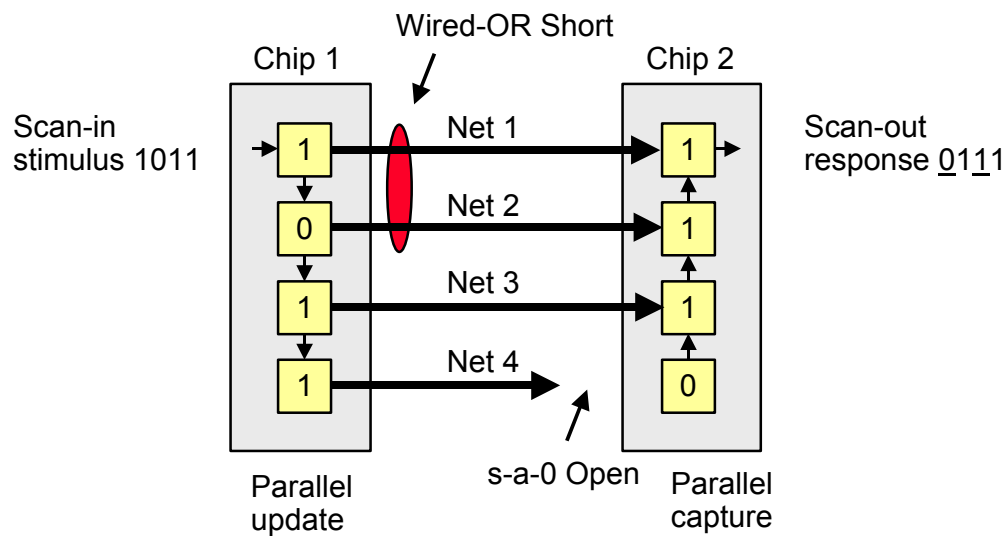


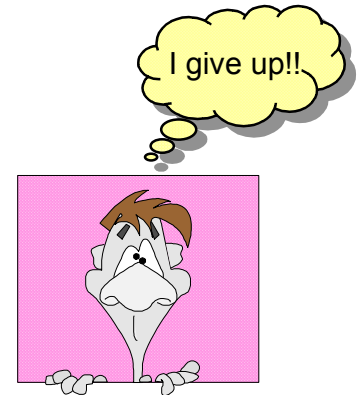
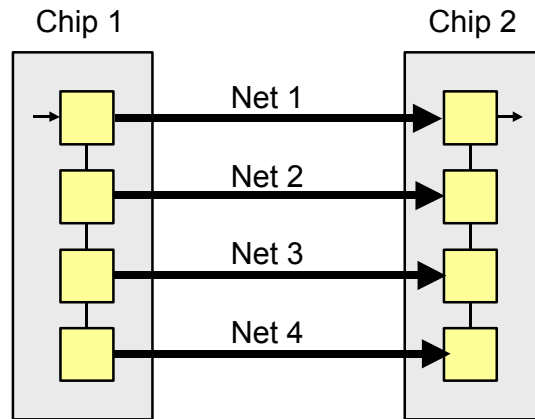
Figure 43

Figure 43 shows a solution. The short circuit (assumed to behave logically like a wired-OR gate) is detected by applying unequal logic values (i.e., logic 1 on Net 1, logic 0 on Net 2) from Chip 1 to Chip 2. The wired-OR behavior causes Chip 2 to receive two logic 1s, allowing detection of the defect.

Similarly, if the open-circuit input behaves like a stuck-at-0 fault, the defect is detected by applying a logic 1 from Chip 1 on Net 4 and observing that Chip 2 captures a logic 0.

## How Many Tests All Together?

### Question: How Many Tests Are Needed?

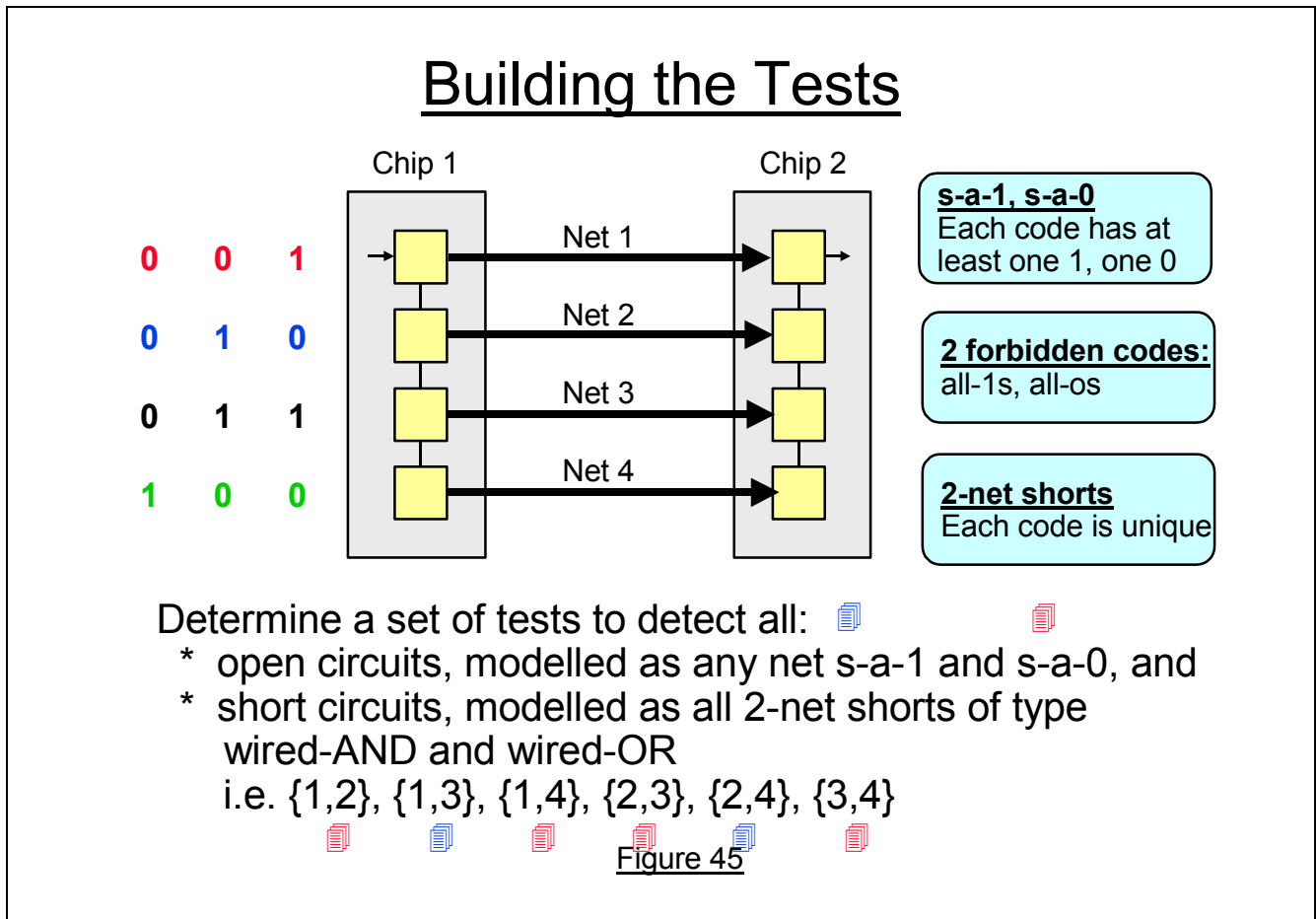


Determine a set of tests to detect all:

- open circuits, modelled as any net s-a-1 and s-a-0, and
- short circuits, modelled as all 2-net shorts of type wired-AND and wired-OR i.e. {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}

Figure 44

A question arises — can we devise a general-purpose algorithm for creating a series of tests capable of detecting any 2-net short circuit (of either a wired-AND or a wired-OR type) and any single-net open circuit (causing either a downstream stuck-at-1 or a downstream stuck-at-0 fault)?

**Building The Tests**

A way to think about the answer is to look at the problem as a net-coding problem and to reason about the properties of the codes. Look at Figure 45 but this time consider the horizontal code allocated to each net e.g. **010** allocated to net 1, etc.

To ensure that each net is tested for a stuck-at-1 and stuck-at-0 fault, there must be **at least one 1 and at least one 0 in the code**. If stuck-at-1, stuck-at-0 coverage were the only requirements, any code except the all-0s and all-1s codes would be sufficient. This means that there are **two forbidden codes: all-0s and all-1s**.

To ensure that two nets, net *i* and net *j*, are tested for a short circuit, there must be at least one bit different between the two codes allocated to net *i* and net *j* in order to apply complementary logic values across the two nets. Complementary logic values are a necessary and sufficient condition to detect a short circuit, assuming logical behaviour of the short. Given that we are targeting all possible 2-net combinations, this generalizes into a requirement that says that the same code can never be allocated to two separate nets i.e. **each code assignment is unique**.

With this as background, a simple code-assignment is shown in Figure 45. The Figure shows three consecutive tests applied to Nets 1 to 4. The first test is the vertical pattern **1010**; the second is **0110**; and the third is **0001**. Now look at the horizontal codes. The code assigned to net 1 is **001**; to net 2 is **010**, etc. Each code has at least one **1** and one **0**, and each code is only used once i.e. is unique.

Furthermore, each code is simply formed by writing the binary equivalent of the decimal number of the net. This code-assignment algorithm is known as the **Counting Algorithm** and is an extremely simple way of satisfying the code-property requirements. Because of the two forbidden codes, the total number of bits in each code (which equals the number of tests) is given by  $\text{ceil} [\log_2(k + 2)]$ , where **ceil** means ceiling (the upper integer value of the logarithm) and **k** is the number of nets. The “+ 2” accounts for the two forbidden codes.

### ***The Counting Algorithm***

## Number of Tests?

Number of tests

= Number of bits in the code

=  $\text{ceil} \log_2(k + 2)$ ,  $k$  = number of interconnects

= 13 for  $k = 8000$  interconnects



It's so simple,  
it's beautiful!!

Figure 46

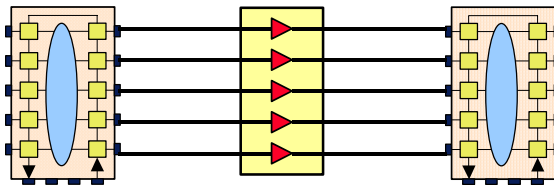
In summary, the number of tests for  $k$  interconnects to detect any open circuit behaving downstream as either a stuck-at-1 or a stuck-at-0, or any two nets from the collection of all  $k$  nets short circuited and behaving as either a wired-AND or a wired-OR gate is given by  $\text{ceil} \log_2 (k + 2)$ .

As Albert Einstein might have said “It’s so simple, it’s beautiful!!”

## Practical Aspects of Using Boundary-Scan Technology

### Handling Non-Boundary-Scan Clusters

### Testing non-BS Clusters



- ❑ On modern boards, most non-boundary-scan devices are simple pass-thru devices e.g. line drivers
- ❑ Consequently, tests for presence, orientation and bonding are easily generated and easily applied via the embracing boundary-scan devices

Figure 47

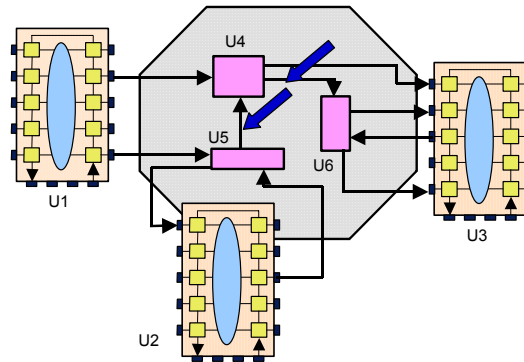
In reality, boards are populated with both boundary-scan and non-boundary-scan devices. The question arises, “what can we do to test the presence, orientation and bonding of the non-boundary-scan devices?” The answer to the question depends, in part, on the degree of controllability and observability afforded to the non-boundary-scan devices through the boundary-scan registers of the boundary-scan devices and, in part, on the complexity of the non-boundary-scan devices.

On most modern boards, the only non-boundary-scan devices are simple line drivers (buffers), with or without inversion, or re-routing devices such as multiplexers. These devices are generally known as “pass thru” devices. It is a simple matter to generate Presence, Orientation and Bonding tests for such devices and then apply the tests via the embracing boundary-scan devices.

But, on older boards, there may be non-boundary-scan MSI devices i.e. devices with more complex functions, such as flip-flops, counters, shift registers, etc. The next slide discusses how to handle such devices.

## Cluster Test Objectives

### But, if the Cluster is MSI Devices ...



- ❑ Use ICT nails to access uncontrollable/unobservable cluster-internal nets
- ❑ Select the real-nail locations on non-BS nets according to access to:
  - strategic disables for guarding or preventing bus conflicts.
  - buried nets in non-embraced clusters
  - other key control signals e.g. O\_Enab, Bidir or 3-state control signals

Figure 48

The diagram shows a **cluster** of three non-boundary-scan MSI devices, U4, U5, U6, surrounded by three boundary-scan devices, U1, U2, U3. The boundary-scan registers in U1, U2, U3 can be used to drive test-pattern stimuli into the non-boundary-scan cluster, and to observe the cluster responses but the difficulty will be to control and observe the truly buried nets inside the cluster (e.g., between U4 and U5).

Given that we are not testing the full functionality of the non-boundary-scan devices — only their **presence, orientation and bonding** — one solution is to develop tests for the non-boundary-scan cluster that are applied from the boundary-scan driver cells and which drive signal values through the cluster targeted on cluster opens and shorts. The responses are propagated out to the boundary-scan sensor cells. Suitable patterns for the non-boundary-scan clusters can be taken from the extensive libraries of In-Circuit Testers and validated via a fault simulator.

An alternative solution is to make use of real nails to access the buried nets, as shown in the diagram. Clearly, these nets have to be brought to the surface of the board (to allow physical probing) and the cost of test will increase (because of the extra cost of the bed-of-nails fixture), but this may be the only way to solve the problem. A solution that combines the virtual access of boundary scan and the real access of a bed-of-nails system is generally known as a **Limited Access** solution

**Access to RAM arrays**

## Testing A RAM Array Via Boundary Scan

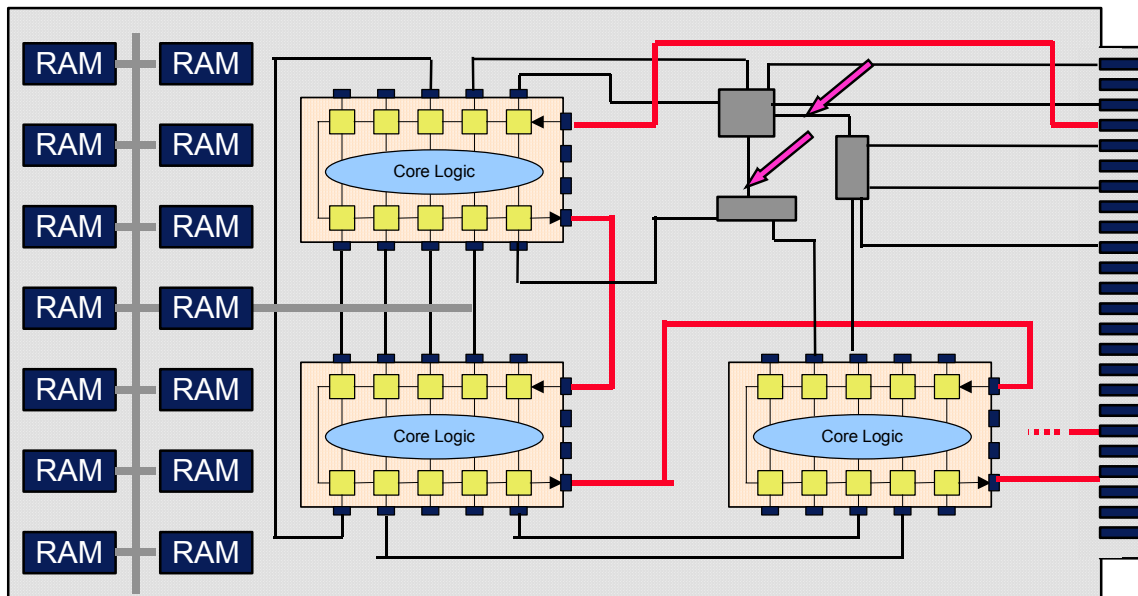


Figure 49

Many boards contain arrays of **Random Access Memory (RAM)** devices (see Figure 49). RAMs are not usually equipped with boundary scan and so they too present manufacturing-defect testing challenges. In a way, an array of RAMs is a special case of a cluster of non-boundary-scan devices.

Boards that contain RAMs typically also contain a programmable device, such as a microprocessor. The usual practice is to use the microprocessor to test the presence, orientation and bonding of the RAM devices (i.e., the microprocessor becomes an on-board tester). If there is no programmable device, then the RAMs can be tested for presence, orientation and bonding defects through the boundary-scan registers of boundary-scan devices as long as the boundary-scan devices have boundary-scan register access to the control, data and address ports of the RAMs. Test times will be slow but the number of tests are not that great given that the purpose of the tests is to identify any opens or shorts on the RAM pins. Suitable tests can be derived from the classical **walking-1/walking-0** patterns or from the **ceil  $\lceil \log_2 (k+2) \rceil$**  patterns described earlier.

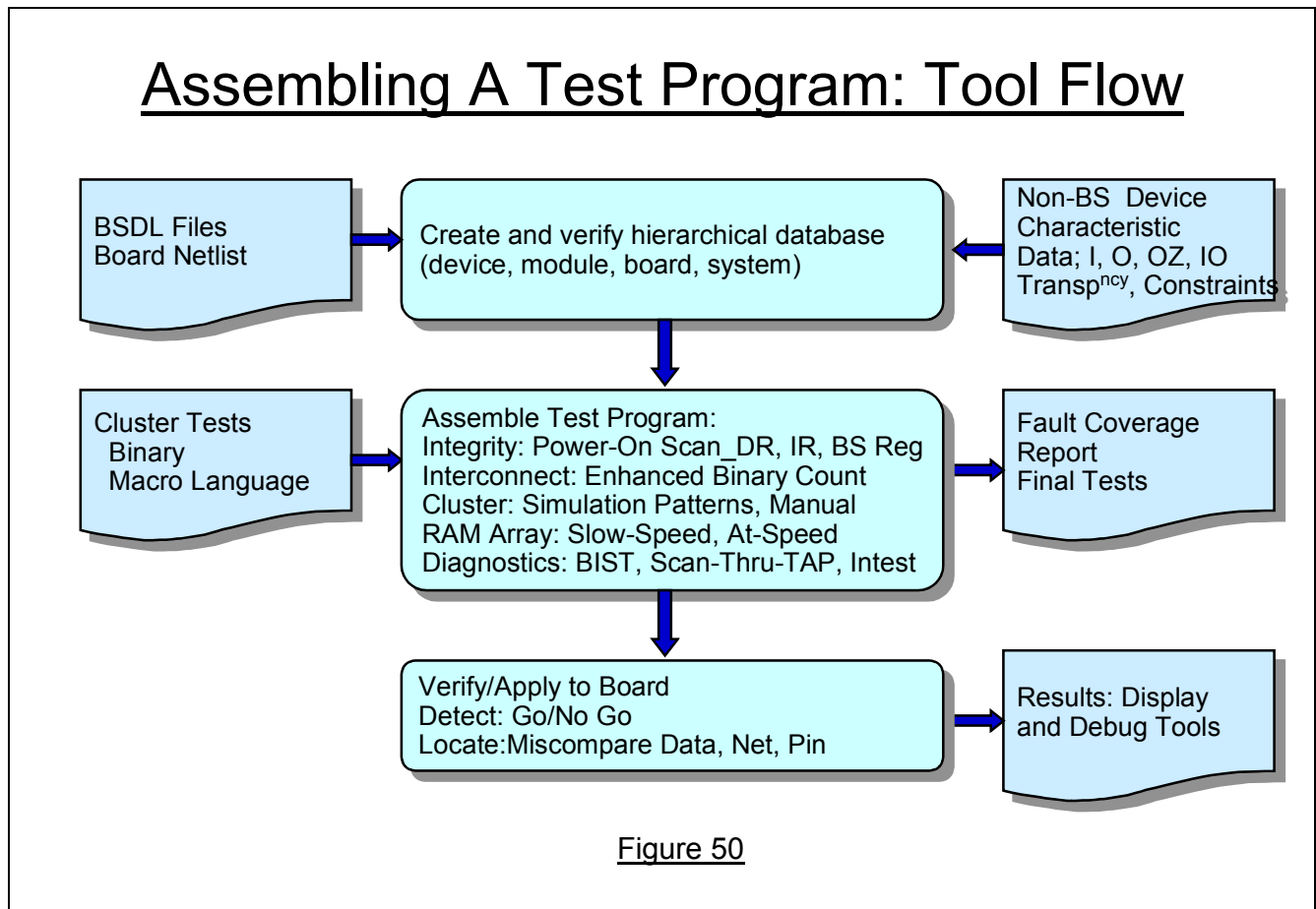
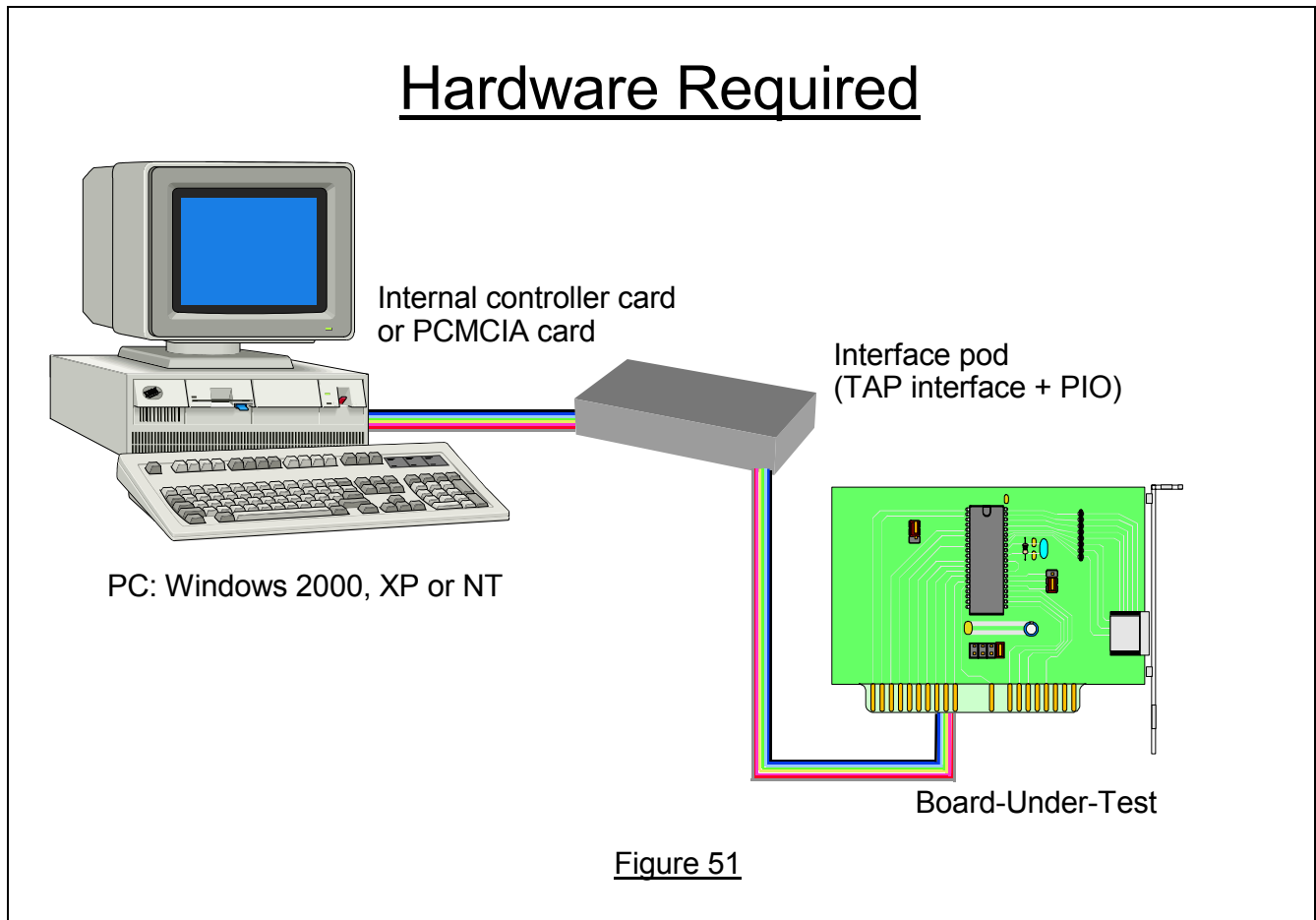
**Assembling The Final Test Program**

Figure 50 summarizes the major stages of assembling a final test program. First, the device **Boundary-Scan Description Language (BSDL)** files and board **netlist** data is used to compile a database. A BSDL file describes all the boundary-scan features included in a specific device. The device designer normally creates the file after all the boundary-scan features have been synthesized. Non-boundary-scan characteristic data is also assembled ready to be used by the various pattern generators. The test program itself is composed of several segments:

- ❑ Board-level test infrastructure integrity test: inter-device TDO-to-TDI interconnects, distribution of TMS, TCK and TRST\*, if present. Typically, these tests use both a **DR\_Scan** cycle and an **IR\_Scan** cycle. The former is an application of what is known as the **blind interrogation** test whereas the latter uses the **01** captured into the Instruction Register, as described earlier.
- ❑ Full Enhanced Binary Count tests between all boundary-scan interconnects, setting non-boundary-scan devices into safe states.
- ❑ Tests to be applied to non-boundary-scan clusters via a combination of boundary-scan devices, real nails (if available), and the normal board edge-connector signals. These tests may be input in a simple **Up, Down, Pulse/High, Low, Don't Care** textual format, or by using a higher-level test language, such as a **macro** language or **C++**.
- ❑ Tests to be applied to on-board RAM devices, either via an on-board microprocessor or via the boundary-scan registers of the boundary-scan devices.

Diagnostics applied to production boards may then make use of internal design-for-test structures such as internal scan (often called **Scan-Thru-TAP**), **Built-In Self Test** or simply through the **InTest** Instruction, if available. The final test results are displayed to the user through an interface which allows line-by-line real-time debug, or by means of a graphical display of applied stimulus and captured test waveforms compared with the expected values.

### Tester Hardware

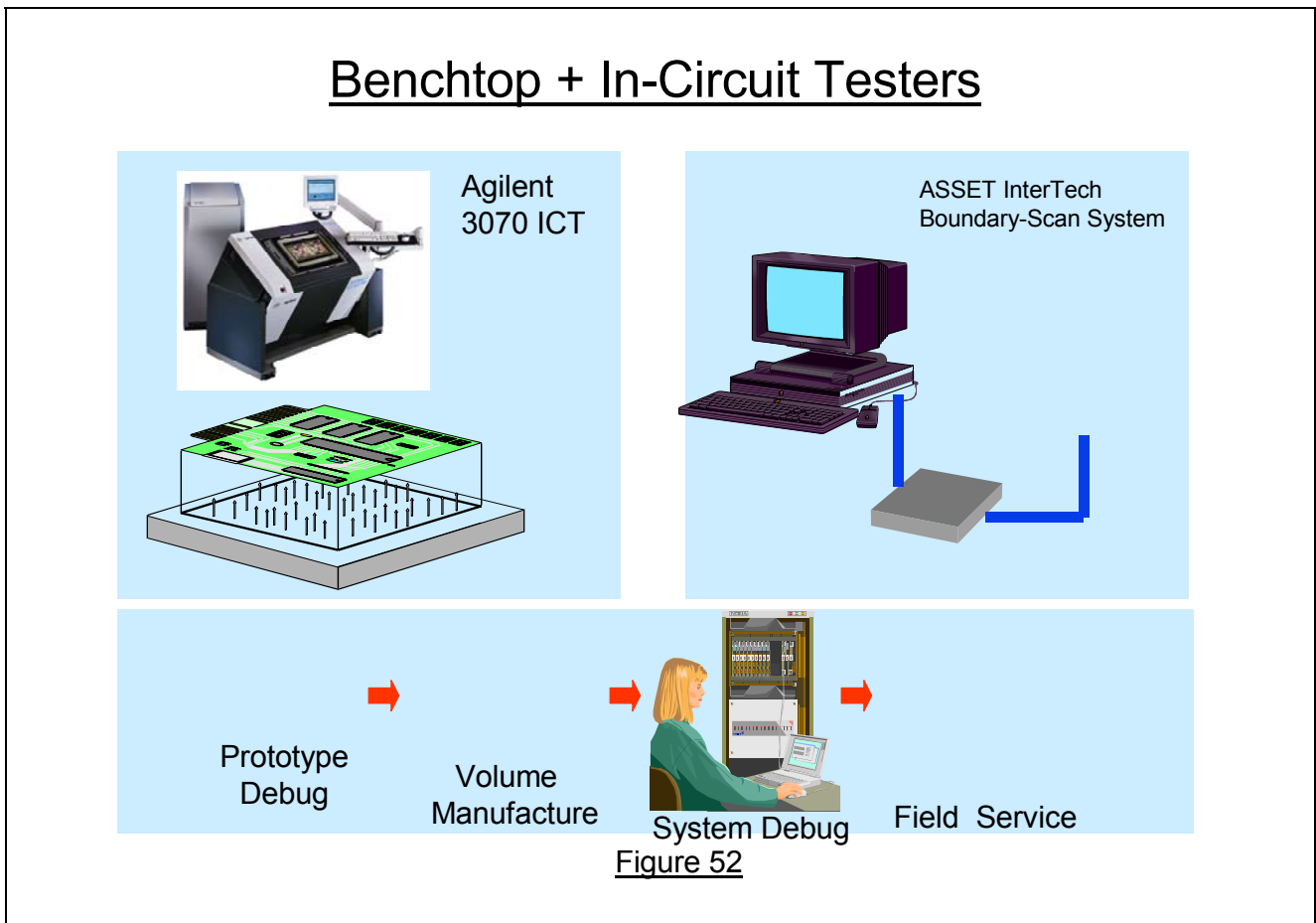


Modern low-cost board testers for boards populated with boundary-scan devices are based on a Personal Computer (see Figure 51). The limited drive/sense capability of the PC is enhanced through a controller card fitted either into an expansion slot (PC-AT, PCI or VXI) or into a PC Card slot or USB port, connecting to the board-under-test via a signal interface pod. TCK speeds are generally in the region of 10 MHz to 25 MHz, but can be higher. Additional driver/sensors are often available to provide direct control and observe on selected edge-connector positions (e.g., control a board Master Reset signal). The stimulus/response patterns themselves, along with the correct value-changes on TMS, are stored in RAM devices mounted on the controller card. These devices form a hardware buffer to hold applied stimulus values and collect actual response values for comparison with the expected values. Overall, the test-preparation and test-application software in the PC is controlled under Windows 2000, XP or NT.

Such board testers are low-cost, compared to traditional in-circuit testers, and very portable, opening up the possibility to make use of the test program in other test requirements on the boards e.g. in multi-board system integration and debug, and in field service.

## Boundary-Scan Economics, Summary and Conclusions

### Summary



Traditionally, manufacturing defects at the board level have been detected and located by bed-of-nail In-Circuit Testers. These testers utilize a bed-of-nail fixture that penetrates into test lands on the surface of the board and, in so doing, gain access to the bonding points of the devices. The nails are controlled by the driver-sensor channels of the In Circuit Tester and allow the application of tests to determine device **presence**, device **orientation**, and correct device **bonding** (opens and shorts) at the solder points.

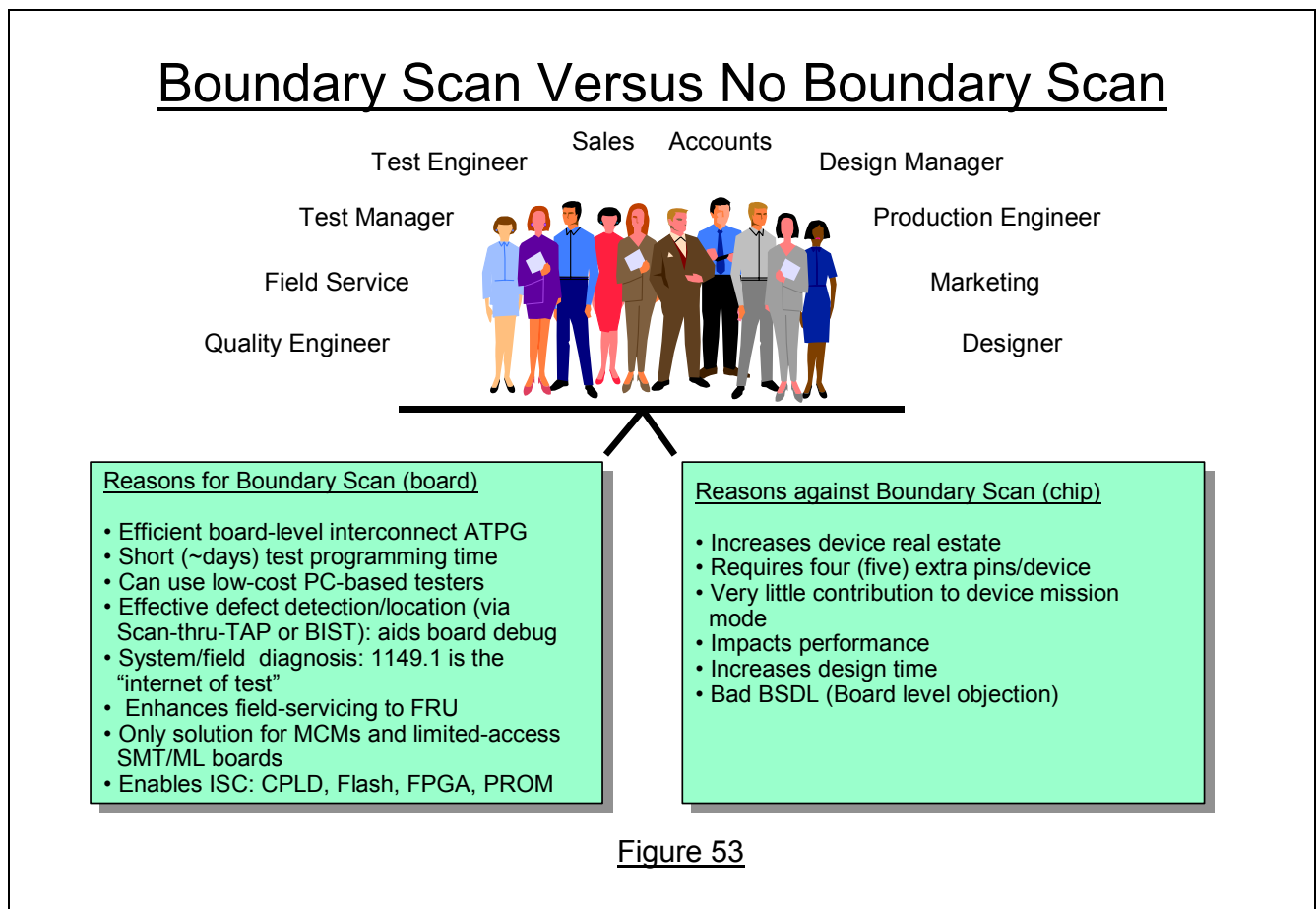
In the mid-1980s, traditional through-hole device packaging began to give way to surface-mount packaging styles. Surface-mount devices are soldered on the same side of the board as the device package itself. This opened up the possibility of mounting devices on both surfaces of the board, thereby increasing the density of devices on the board, and the need to interconnect between them. Consequently, multi-layer board technology was developed to cope with the increased density of interconnection.

The result was that the one thing that an In-Circuit Tester requires – physical access for nail probe – started to decrease. An alternative solution was formulated by an organization called the **Joint Test Action Group (JTAG)**. This solution defined a new universal register around the boundary of a digital device to bring back the access to the bonding points. This **boundary-scan** register added nothing to the functionality of the device but solved the problem of limited physical access. JTAG

developed the technology of boundary scan into the first true international standard on testability: the IEEE 1149.1-1990 Boundary-Scan Standard.

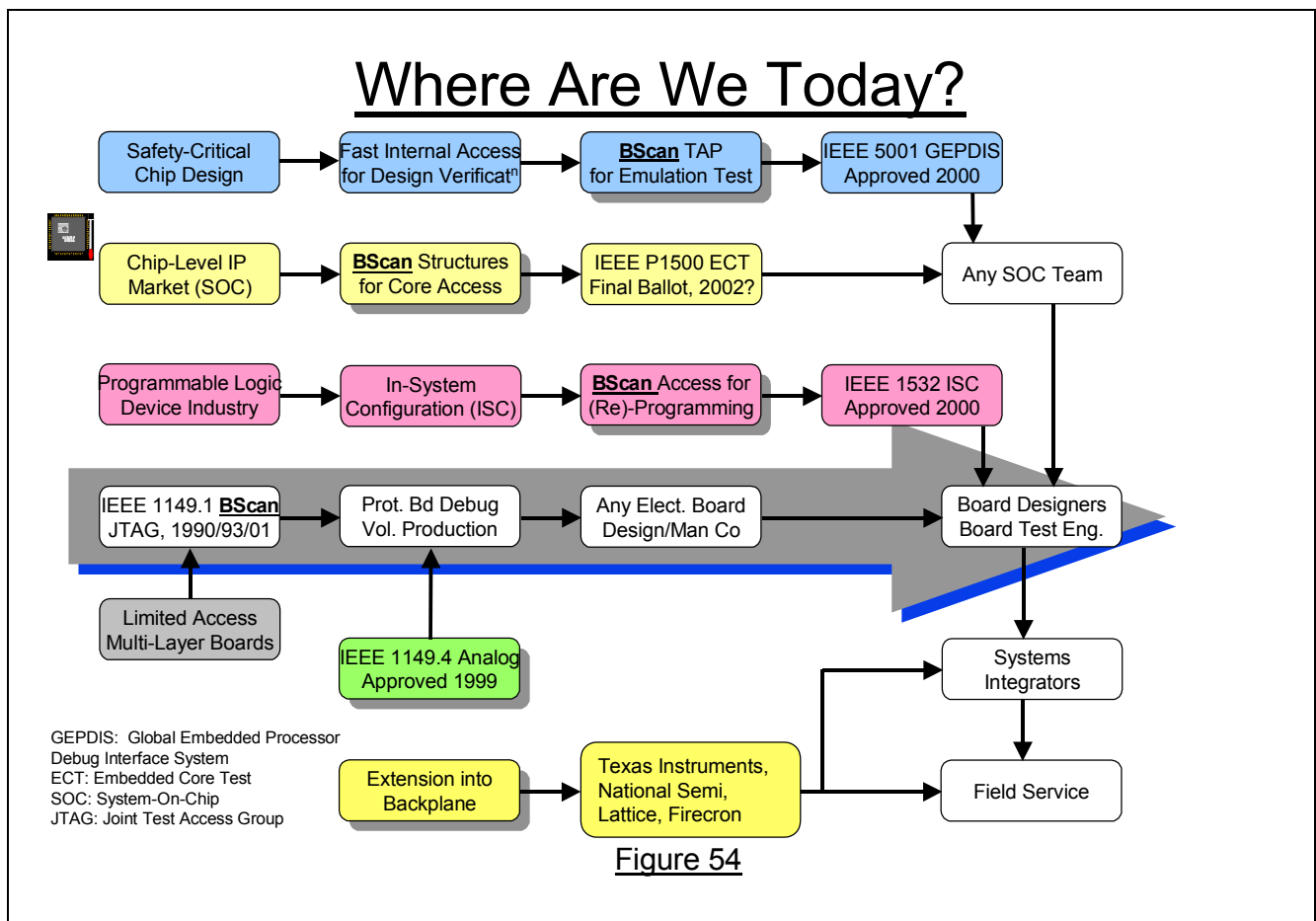
The boundary-scan registers bring back the lost access thereby allowing presence, orientation and bonding tests to be carried out. The registers also allow access inside the device, if required, to assist board-level diagnostic test requirements. Particularly, the boundary-scan structures allow re-use of other internal **Design-For-Test** structures, such as the internal scan and Built-In Self Test structures described earlier. Re-use eases the economic cost justification of Design-For-Test by allowing amortization of Design-For-Test over the life-cycle of the product.

### **Boundary Scan Economics**



The economic analysis of boundary-scan is notoriously difficult. To do the analysis properly requires an in-depth understanding of the economic system used within a company, the type of board and dynamics of the end-user market, the current way of testing boards, and so on. To help the reader, Figure 53 summarizes the main factors for and against the use of boundary scan but be warned: this is a **very** simplified view. Often, engineers start using boundary scan simply because there is no other way to solve the limited-access problem at board test!!

### **Where are we today?**



There are three major developments fuelling the current boundary-scan market. The first is the increasing acceptance of **ball-grid array** styles of IC device packaging. The second is the recent extension of the boundary-scan Standard into the analog domain: the publication of the **IEEE 1149.4-99 Mixed Signal Test Bus Standard**. The third is the increasing desire to program programmable logic devices on the board, rather than off the board.

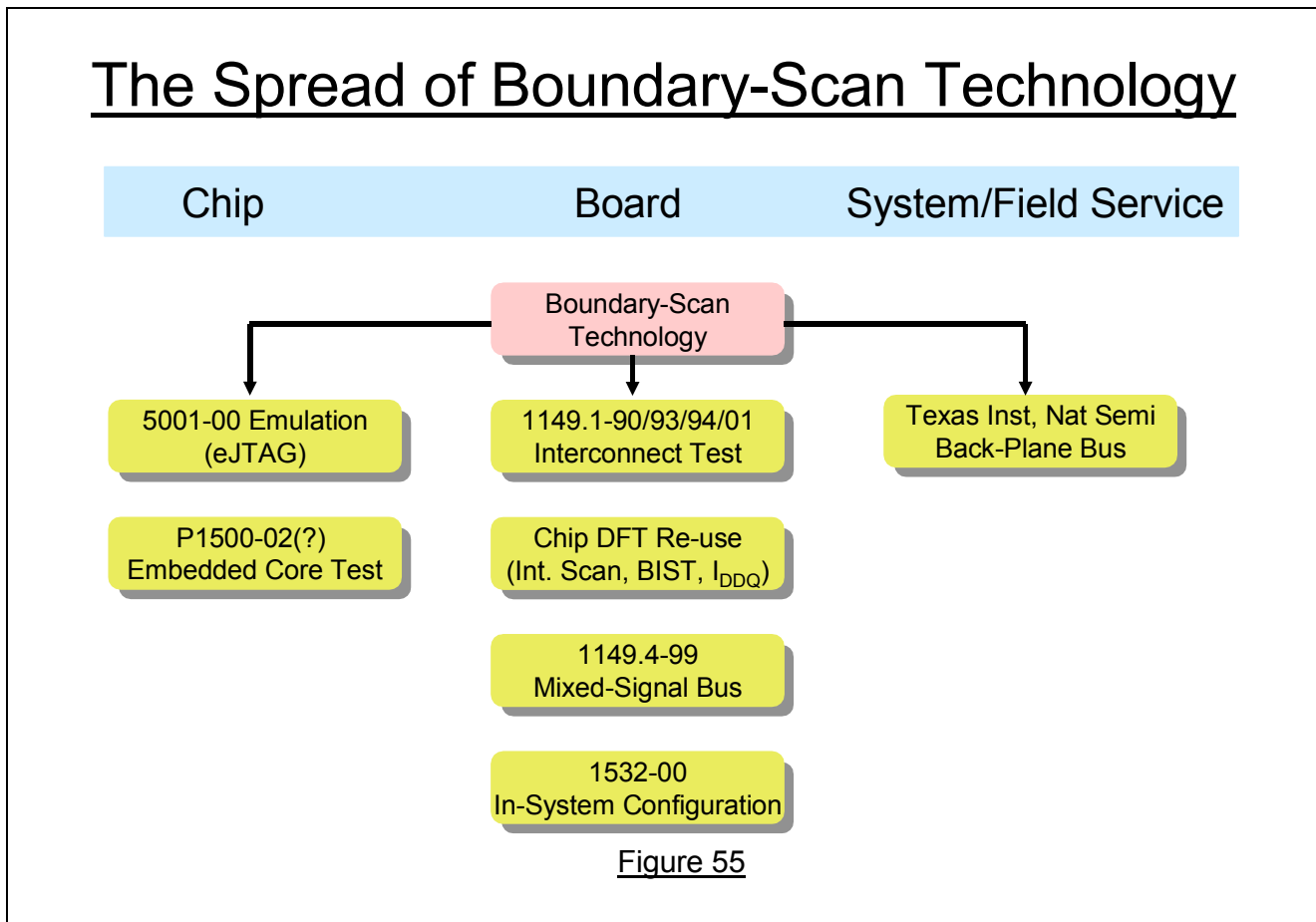
**Ball-grid array** styles of packaging place balls of solder underneath the package of an IC, replacing the traditional pins of a device and allowing the device to be bonded to the board within the confines of its own footprint. This increases yet further the density of devices on the board but means that once the device is bonded to the board, it is not possible to use any form of visual inspection, such as an automated optical inspection system, to determine the correctness of bonding. The result is that boundary-scan adoption is now mainstream.

A recent addition to boundary-scan technology is the **IEEE 1149.4-1999 Mixed-Signal Test Standard**. The original **1149.1 Boundary-Scan Standard** addressed only the digital signal pins of an IC. 1149.4 addresses the analogue pins of mixed-signal devices and builds on the 1149.1 boundary-scan structures. It is too early to say whether this new Standard will meet with industrial acceptance but interest is high, especially in the telecommunications market.

Many new logic designs are first created and debugged in programmable IC devices such as **Complex Programmable Logic Devices (CPLDs)** or **Field-Programmable Gate Arrays (FPGAs)**. Such devices are usually programmed off the board on a free-standing programming station but there is a growing trend to place the devices on the board, un-programmed, and then to program them on

the board using the board-level boundary-scan pathways. This technique, known as **In-System Configuration (ISC)**, offers many advantages over the traditional programming station technique and has caused yet another new IEEE Standards group to be created. The **IEEE 1532-2000 In-System Configuration Standard** was approved in 2000 and it relies on and makes use of 1149.1 boundary-scan structures within the CPLD and FPGA devices. In-System Configuration could become an even bigger application of boundary scan technology than the original JTAG application to manufacturing board test.

## Conclusion



Widespread adoption of the IEEE 1149.1 Boundary-Scan Standard reflects an industry-wide need to simplify the complex problem of testing boards and systems for a range of manufacturing defects and performing other design debug tasks. This standard provides a unique opportunity to simplify the design debug and test processes by enabling a simple and standard means of automatically creating and applying tests at the device, board, and system levels. Several companies have responded with boundary-scan-based software tools that take advantage of the access and control provided by boundary-scan architecture to ease the testing process.

In this tutorial, we have discussed the motivation for the standard, the architecture of an IEEE 1149.1-compliant device, and presented a simple introduction to the use of the IEEE 1149.1 features at the board level — both to detect and to locate manufacturing defects. For further details on boundary-scan — at the device level, board level, or system level — see the references listed in the **To Probe Further** section or visit [www.vitalelect.com](http://www.vitalelect.com) for an extended web-based-learning version of this tutorial.

Ben Bennetts  
DFT Consultant  
17 August, 2001  
(Updated 25 September, 2002)

To Probe Further ....

- **IEEE 1149.1-2002 Test Access Port and Boundary Scan Architecture Standard** web site: <http://grouper.ieee.org/groups/1149/1> or <http://standards.ieee.org/catalog/> Ken Parker, “**The boundary-scan handbook: analog and digital**”, Kluwer Academic Press, 1998 (2nd Edition), <http://www.wkap.nl/>
- Proceedings of the **IEEE International Test Conference (ITC)**, <http://www.itctestweek.org/> (*Latest developments on boundary scan technology and applications*)
- **IEEE Design & Test of Computers (D&TC)** magazine, <http://computer.org/> (*The official journal for new announcements about the Standard*)
- **Agilent Technologies BSDL verification service**: [http://www.agilent.com/see/bsdl\\_service](http://www.agilent.com/see/bsdl_service) or [bsdl\\_server@lvld.agilent.com](mailto:bsdl_server@lvld.agilent.com) (*Free BSDL syntax checker, semantics checker and pattern-generation service*)
- **ASSET InterTech** web site: [www.asset-intertech.com](http://www.asset-intertech.com) (*Market leader in low-cost PC-based testers for boards populated with boundary-scan and non-boundary-scan devices*)
- **Texas Instruments’** Web site: <http://www.ti.com/sc/docs/jtag/jtaghome.htm> (*Details of all TI’s boundary-scan devices, plus downloadable animated tutorial material*)
- **National Semiconductor’s** web site: <http://www.national.com/scan/> (*Details of all National’s boundary-scan devices*)
- **More Advanced Reading**
- **IEEE 1149.4 Mixed-Signal Test Bus Standard** web site: <http://grouper.ieee.org/groups/1149/4>
- **IEEE 1532 In-System Configuration Standard** web site: <http://grouper.ieee.org/groups/1532/>
- Tegethoff, Parker and Lee (Agilent Technologies), “**Open boards test coverage: when is 99% really 40%?**”, IEEE ITC Proceedings, 1996, Paper 12.2, pp. 333 – 339. See also de Jong et al., ITC2000, P22.1 (*Problems and solutions to the placement of boundary-scan cells on Power and Ground*)
- Sasidhar et al., “**Testing NASA’s 3D-stack MCM space flight computer**”, IEEE Design & Test of Computers, July-Sept., 98, pp. 44-55 (*NASA’s application of boundary scan and multi-chip modules for space computers*)
- Barr et al (Lucent Technologies), “**End-to-end testing for boards and systems using boundary scan**”, IEEE ITC Proceedings, 2000, Paper 22.2, pp. 585 – 592. (*Lucent Technologies’ use of 1149.1 at board level and as a backplane test bus for system level test*)
- Harrison et al (Motorola Network Solutions), “**The implementation of IEEE 1149.1 boundary scan test strategy within a cellular infrastructure production environment**”, IEEE ITC Proceedings, 2000, Paper 2.3, pp. 45 – 54. See also follow-up paper at ITC 2001 (P17.2) (*Motorola’s use of 1149.1 at board level and as a backplane test bus for system level test*)
- David Marsh, “**Simple boundary-scan techniques tackle sophisticated systems**”, Electronic Design News (EDN) Europe, July, 2001, pp. 34 – 42. (Excellent survey of more-recent applications of boundary scan, including 1532 In-System Configuration and 1149.4 Mixed-Signal Bus. Available on <http://www.ednmag.com>)
- Rick Nelson, “**PCB test: nails or TAP?**”, Test & Measurement World, Sept, 2002, pp. 17 – 24. (*Excellent survey of the technical and business relationships between the low-cost PC-based boundary-scan testers and the more traditional in-circuit test products and suppliers.*) Available from [www.tmworld.com](http://www.tmworld.com)