
EE534
VLSI Design System

Lecture 14:

Design for Testability
IC Packaging

Part 1: Design for Testability

What is it?
Why do we need it?
How do we get it?

Testability

- ❑ Why is testing so important in IC's?
- ❑ Typical device may contain 120,000,000 transistors
- ❑ 1 bad transistor means that the part probably will not work correctly
- ❑ Question: How do you make sure every single transistor is working?

Terminology

- ❑ The terms used in testability and reliability analysis can be very confusing...
- ❑ A *fault* is some sort of flaw or mistake in the system
- ❑ A *failure* is some sort of observably incorrect behavior in the system
- ❑ Not every fault is a failure!
- ❑ For example, if a system has a redundant component, a fault in one component will not lead to a failure

Aside: Measuring Quality

- ❑ So why the concern about faults and failures?
 - You want to make sure that you are shipping products that work!
- ❑ In volume manufacturing, there are two ratios of concern:
 - Yield: Of all the parts I make, how many do I consider “good”?
 - Product quality: Of all the parts that I *ship* to customers, how many do *they* consider “good”?
- ❑ The former (some number of parts being bad) is considered “normal” in the IC industry
 - Many factors cause a certain number of parts to be bad in every batch
- ❑ The latter is what we are concerned about – how many parts that I ship are considered “bad”?

Quality Metrics

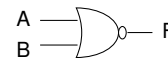
- ❑ The number of defective, shipped parts is normally expressed as “defective parts per million” or DPPM
 - The number of defective parts out of every lot of 1,000,000
- ❑ Some industries may require even higher rates of quality and use more stringent metrics, such as DPPB
- ❑ Thought questions:
 - How do you guarantee a certain DPPM rate?
 - Does that mean you have to completely test every single part?
 - Which is more expensive – a lot of testing or shipping some bad parts?
- ❑ These questions have no single right answer!

Test Philosophy, ctd.

- ❑ Coming back to integrated circuits...
- ❑ So how do you test a single circuit?
- ❑ You have to apply every possible input and observe the correct output in every case
 - There is no shortcut here – if you don’t do both, then there is the possibility that the circuit will not work under untested input conditions
- ❑ This drives two very important issues in testing:
 - Controllability: Can you apply every single input case?
 - Observability: Can you sense if the input is correct or not?

Consider a simple example...

- ❑ 2-input NOR gate:
 - Apply all four input combinations
 - Check the output for each case

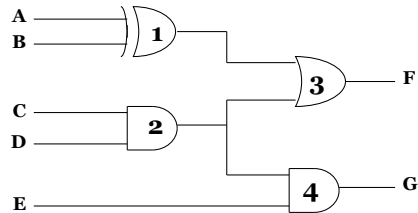


A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

- ❑ Piece of cake, right?
- ❑ But what if that gate is in the middle of a bunch of other logic?

Consider another example...

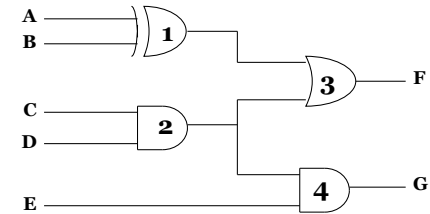
- Consider the following circuit



- What combinations of inputs and outputs test all 4 gates?
 - Must apply every input combination and observe every output

Consider another example...

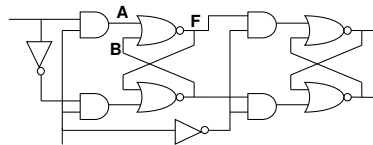
- My answer...



- Each combinations of inputs and outputs is called a *test vector*
 - This example has 7 vectors
- Goal is to maximize coverage with minimal number of test vectors

A	B	C	D	E	F	G
0	0	0	0	0	0	0
0	1	0	0	1	1	0
1	0	0	1	0	1	0
1	1	0	1	1	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	1	1	1	1	1

Not-so-simple example...



- How do you *control* and *observe* every single gate in a complex circuit?
- For example, how do you apply all 4 combinations to the NOR gate above?
- How do you observe the output of all 4 input combinations?
 - That may be impossible – the clock will cause the output to be non-observable 1/2 of the time

We need a computer to keep track of it all!

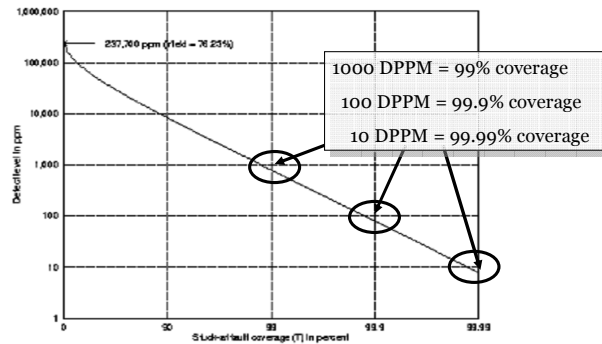
- It soon becomes clear that we need special software tools to track every single transistor
 - Find every instance where the output of the gate directly affects a measurable output
 - Find the input combination for that output
 - Track how many input combinations have been applied and then observed
- The overall result is the fraction of the chip that has been tested

- Simple example: 4 logic gates
 - 14/20 = 70% test coverage

Gate	# of possible combinations	# actually tested
1	4	3
2	4	2
3	8	6
4	4	3
Total	20	14

How good is “good enough”?

- Studies show how much testing is needed to achieve a certain level of reliability



- Source: sina.sharif.edu/~hessabi/Test/lec4.pdf

And so the problem emerges...

- If you want 10 to 100 DPPM (typical quality numbers), you need 99.9% to 99.99% test coverage.
 - Example: 120,000,000 transistors = 30,000,000 gates = 120,000,000 input combinations
 - Must test about 119,880,000 out of 120,000,000 combinations to achieve 100 DPPM
- The approach of simply applying inputs and checking outputs “runs out of gas” at around 90% - 95% test coverage.

What shortcuts can we take?

- Eliminate input combinations
 - Example: Multiplier array (2-dimensional array of adders)
 - Some combinations of inputs are physically impossible and can be eliminated
- Very regular logic structures
 - Example: Counters
 - Most input combinations are tested by exhaustively testing the states of the structure

Aside: The evil of counters...

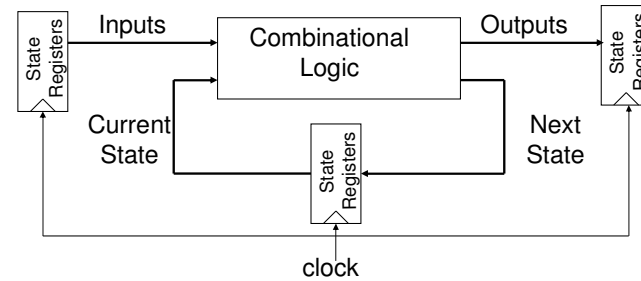
- Counters are notorious in testability circles
 - Example: A 32-bit counter is only completely tested after a sequence of $2^{32} = 4 \times 10^9$ iterations
- Classic solution: Add logic to break a single big counter into several smaller counters
 - Example: Break 32-bit counter into 4 8-bit counters
- Segues into the next step: *Add testability logic*

Testability Logic

- It soon becomes clear that extra logic will be needed just to finish the testing
 - Example: Breaking up counters
 - Example: Injecting number patterns into arithmetic logic
 - Example: Observing internal logic states on outputs
- What if the testability logic is itself broken?
 - Doesn't really matter, as long as faults are manifested in observable ways
 - You cannot fix an IC – you only want to throw it away if it is broken – so it doesn't matter if the testability logic is broken or the logic under test is broken

How do we get controllability and observability?

- In general, digital circuits consist of patches of combination logic strung between flip-flops



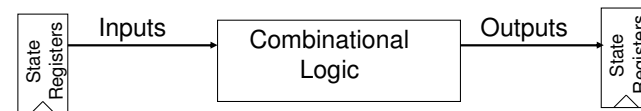
- If you can *control* the flip-flops, you can force input combinations into the combinational logic
- If you can *observe* the flip-flops, you can read the internal behaviors of the combinational logic

Method 1: Built-in Self-Test (BIST)

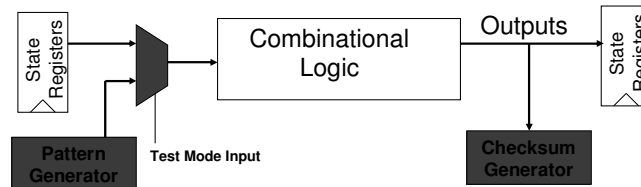
- It is easy to build circuits that create repeatable bit patterns
 - Built up from networks of XOR gates
 - Used in encryption and random-number generation, e.g.
- It is easy to build circuits that create *checksums* – bit patterns that change if the input data changes
- Control: create a series of bit-pattern inputs
- Observe: create a checksum of all output patterns

BIST: Before and After

- Before:



- After:



BIST: Advantages and Disadvantages

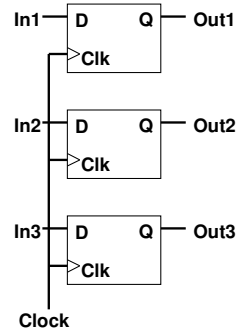
- Advantages
 - Powerful way to apply rich set of inputs and check outputs
 - Extremely efficient hardware implementation
 - Observability is simplified – only need to read checksum
- Disadvantages
 - Highly specialized function
 - Change input pattern generator every time logic changes
 - Some combinational structures are not suited for this approach

Method 2: Scan paths

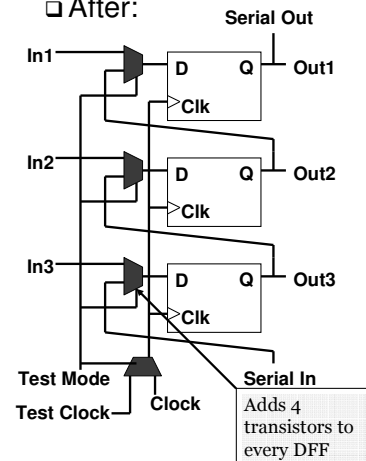
- Basic idea: All flip-flops are also connected in a long serial shift-register
 - The flip-flops become a shift register than stretches throughout the entire chip
 - The test interface can be used to shift in input patterns and read out output patterns

Scan: Before and After

□ Before:



□ After:



Scan: Advantages and Disadvantages

- Advantages
 - Large payoff for very little additional logic
 - Additional logic is spread around – impacts each DFF slightly
 - Scan path available for a variety of uses
 - Example: Can stop the chip and read out the state – very powerful debugger
 - Very small additional pin count
- Disadvantages
 - Have to change the patterns every time the logic changes

Scan: Hair-Raising Tales from Industry

- ❑ One interesting thing about running the scan chain:
Every single flip-flop is clocked on every single cycle
 - Normally only about 10% to 20% of the flip-flops toggle
- ❑ Scans normally run at a much lower clock speed (e.g. 50 MHz) but can run full speed
- ❑ One chip got so hot running full speed (because all flip-flops toggled every cycle) that it soldered itself to the tester!

Scan: The Testing Bottleneck

- ❑ Test time is extremely important
 - Must use dedicated chip testers
 - Becomes a production bottleneck
 - Typical goal: Run tests three times (min voltage, typical voltage, max voltage) in a total of 6 seconds
- ❑ It very quickly adds up
 - Example: 100,000 flip-flops and 20,000 test vectors = 2 *billion* clock cycles = 40 seconds at 50 MHz

Scan: Solutions to the Bottleneck

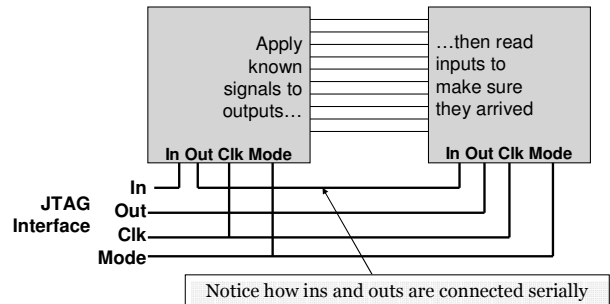
- ❑ So the scan path is broken up into multiple paths
- ❑ *Broadside scan*: Every I/O pad is put to use as a scan input or output
 - Breaks the chip into a few hundred chains, each a few thousand flip-flops long
- ❑ However, as the number of gates increase exponentially, the number of I/O's increases linearly
 - Even broadside scan has reached its limits
- ❑ Now add decompression / compression logic to the scan interface
 - Vectors are input in compressed form and decompressed on chip
 - Outputs are compressed before sending off-chip

JTAG interface

- ❑ The semiconductor industry formed the Joint Test Action Group or JTAG to address this
 - JTAG became an extremely popular industry standard for chip testing
- ❑ The basic interface consists of four pins:
 - Serial input
 - Serial output
 - Serial clock
 - Reset / Mode select
- ❑ When the interface is active (when the flip-flops form a shift register), the chip is in a test mode
- ❑ The standard also calls out methods for accessing test-control registers inside the chip

The power of JTAG

- By adding a small amount of logic to every flip-flop, the IC can approach the necessary test level
- The same interface can be used to test the circuit board
 - Since the interface also allows direct control of chip I/O's

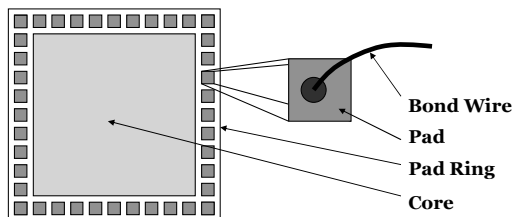


Part 2: IC Packaging

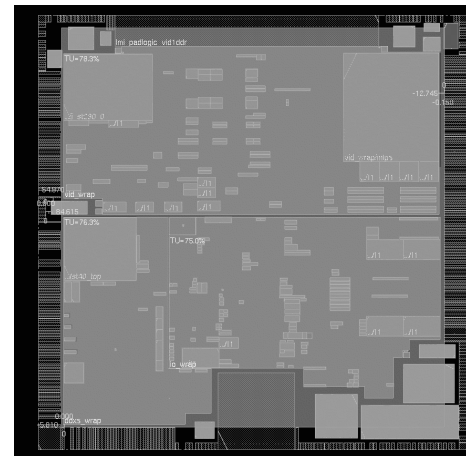
How is it done?
How does it impact designs?

All designs need packaging

- Your chip must have some interface to the “real world”
 - Power, ground, inputs, outputs, etc.
- The typical interface is a *pad*
 - Large square structure for attaching wiring or solder
- Typical IC's put these interfaces around the edges
- The collection of pads is typically called the *pad ring*



Pad-ring design is a key part of chip design



This is a screenshot of a chip during the design stage.

You can see the pad ring in blue around the edge.

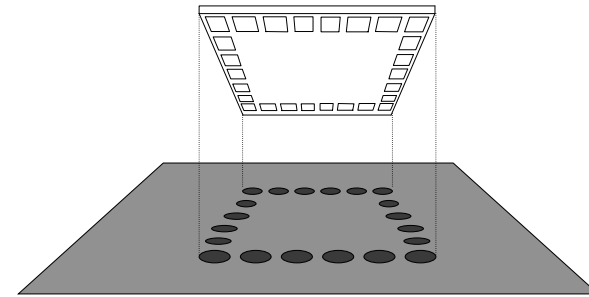
Next step: Decide what kind of package you want!

What type of packages?

- Package styles
 - Bare die
 - Sometimes buy bare die to put different IC's in a single package
 - Common example: 1 processor, 1 Flash, 1 RAM
 - Flip-chip
 - Chip is bonded to package pads-down
 - Wire-bonded – Most common form of packaging
 - Chip is bonded to package pads-up
- Can combine multiple chips into a single unit – Multi-chip Module or MCM

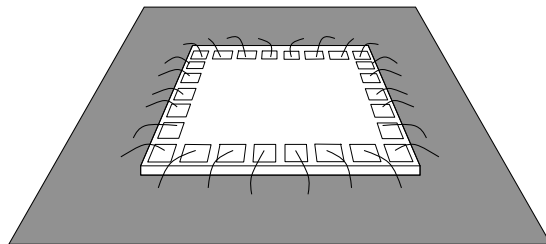
Flip-Chip Packaging

- Die is flipped upside-down and soldered directly to the package

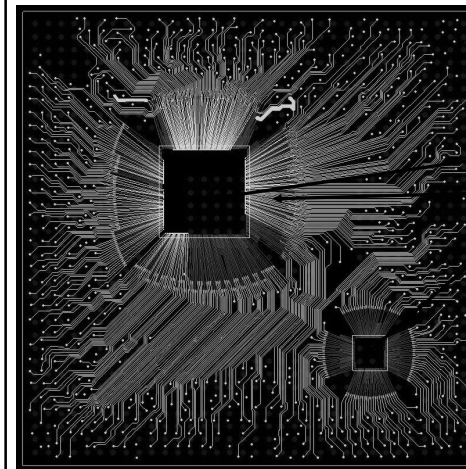


Wire-Bond Packaging

- Die is attached with pads on top and individual wires are run to the package



Example: Wire-bonded BGA MCM

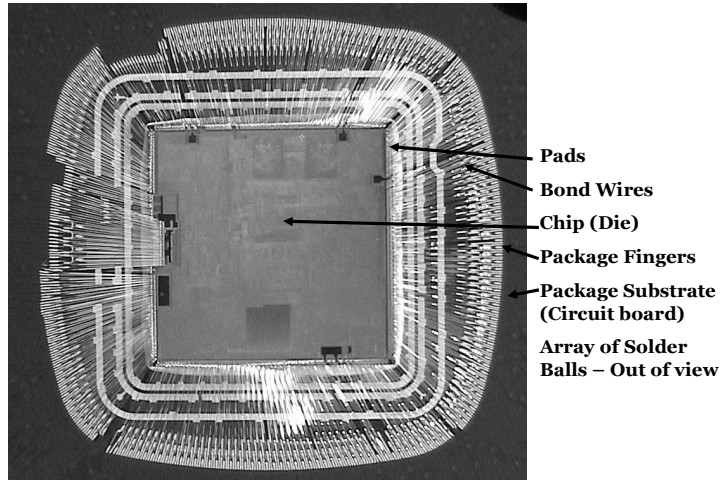


- Two wire-bonded chips that come out to a ball-grid array

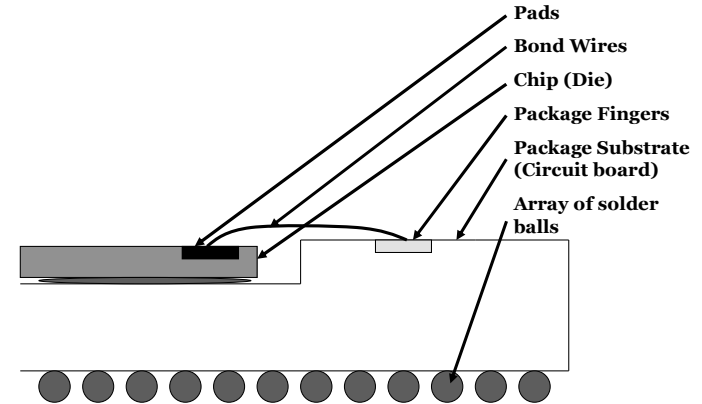
Chip 1
Wire bonding
Chip 2
Grid of Solder Balls

- Chips are placed and wire-bonded
- The "package" is a small circuit board that brings signals out to an array of solder balls

Example of Fully Assembled Part



What you just saw... (Side View)

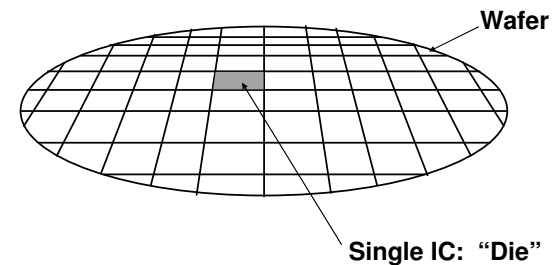


Types of wire-bonded packaging

- All the same on the die side – differences in how the pins come out of the package on the bottom
- Ball-grid array
 - Bottom of part is an array of solder balls for mounting on circuit boards
 - Advantages: Very low inductance, hundreds of pins available
 - Disadvantage: Can only be inspected by X-ray once assembled
- Quad flat pack
 - Pins on all four sides of the package that come down to circuit board
 - Less expensive, but fewer pins available (typically < 100)
- SO packaging
 - Pins on two sides of parts
 - Very small packaging available, but very few pins

Package Process Steps

- Result of fab process is an entire wafer of integrated circuits...



Which ones are good?

- ❑ Many of the dies are bad – considered normal in the industry.
 - Prefer not to go to the expense of packaging every single die
 - Typically probe the wafer first with a quick test...
- ❑ Build custom wafer-probe fixture
 - Lets you apply power and signals to each die
 - Find out which ones are good
 - Excellent use of scan paths – only need a few I/O's to test the part!
- ❑ Then cut them apart with a saw – Good parts are packaged

