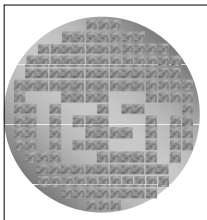


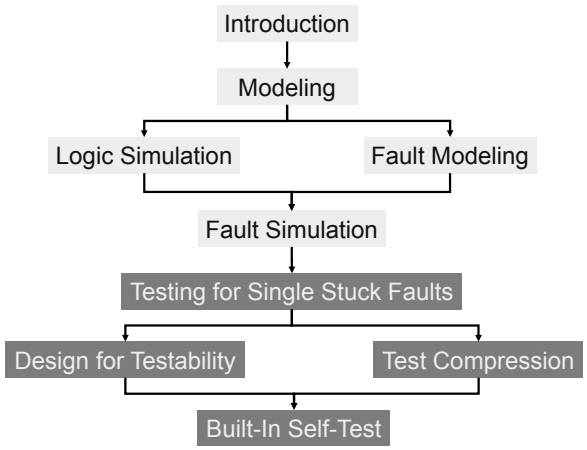
VLSI Testing

Tsung-Chu Huang

Department of Electronic Engineering
National Changhua University of Education
Email: tch@cc.ncue.edu.tw

2016/02/15

Syllabus & Chapter Precedence



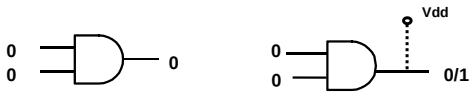
Introduction to IC Test

Outline

1. **What's Testing**
2. **Why Test?**
3. **Difficulties of Testing**
4. **How to Do Testing?**
5. **Logic/Fault Simulations**
6. **Test Generation**
7. **Built-In Self-Test**
8. **Test Compression**
9. **DFT**

What's Testing

To tell whether a system is good or bad



Related fields

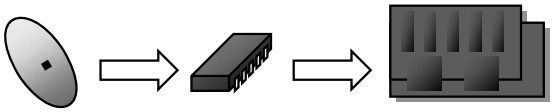
Verification: To verify the correctness of a design

Diagnosis: To tell the faulty site

Fault-tolerance: To work normally even faults exist

Why Test?

1. **Why not ship without test?**
2. **Why not final product test only?**



$1 $10 $100

Rule of Tens

3. **Why not functional test only?**

- **Without test at stage k**
- **Cost wasted:** $(1\text{-}Y)(P_{k+1}\text{-}P_k)$

Example for Yield Loss due to Size

Importance of Test

N = # transistors in a chip
p = prob. (a transistor is faulty)
Pf = prob. (the chip is faulty)

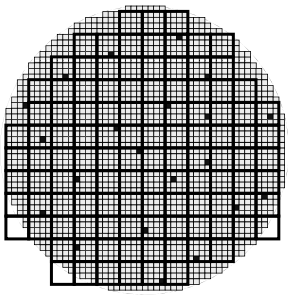
$\Rightarrow$ **$Pf = 1 - (1-p)^N$**

If $p = 10^{-6}$
$N = 10^6$

$\Rightarrow$ **$Pf = 63.2\%$**

Example for Yield Loss due to Density or Size

When chips are very small, assume the probability of defected chip is α Y=1- α



Why Not Final Product Test Only?

Importance of Test

1. **Testability degradation**
2. **Faults may occur at any phase**
3. **Average Penalty Increasing**

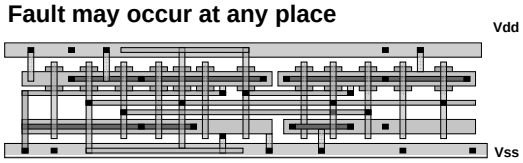
Why not functional test only?

Problems to think

1. **A 32 bit adder**
2. **A 32 bit count-up counter with RESET function**
3. **A 1MB cache memory**
4. **A 10M-transistor CPU**

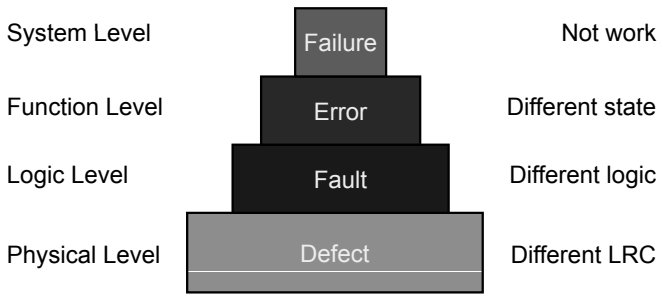
Difficulties in Testing

- **Fault may occur anytime**
  - Design
  - Process
  - Package
  - Field
- **Fault may occur at any place**



- **VLSI circuit are large**
  - Most problems encountered in testing are NP-complete
- **I/O access is limited**

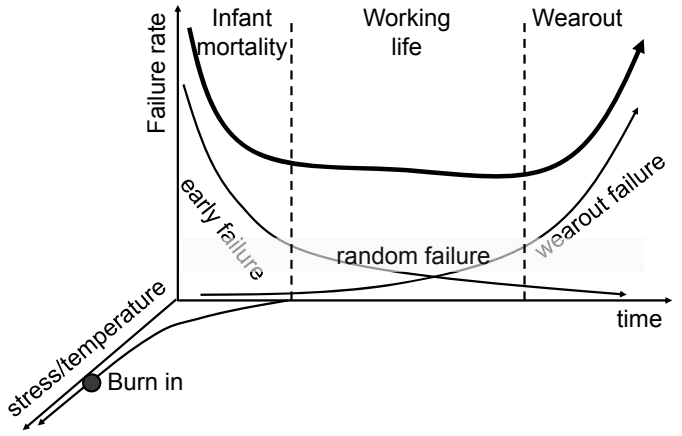
From Defect to Failure



Fault Manifestation

- **Permanent Faults**
- **Non-permanent Faults**
  - ✓ Transient Faults
    - Soft Faults
  - ✓ Intermittent Faults

Bathtub Curve



How to Do Testing

- **Circuit modeling**
- **Fault modeling**

} **Modeling**

- **Logic simulation**
- **Fault simulation**
- **Test generation**

} **ATPG**

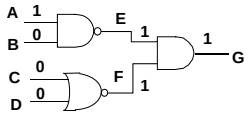
- **Design for test**
- **Built-in self test**

} **Testable design**

- **Synthesis for testability**

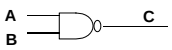
Fault Simulation

- **To determine the behavior of faulty circuits**



- **Given a test vector, determine all faults that are detected by this test vector.**

  Example:

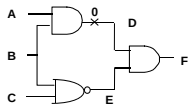


Test vector (1 1) detects
{ a_0, b_0, c_1}

Test Generation

- **Given a fault, identify a test to detect this fault**

  Example:



To detect D s-a-0, D must be set to 1.
Thus A=B=1.

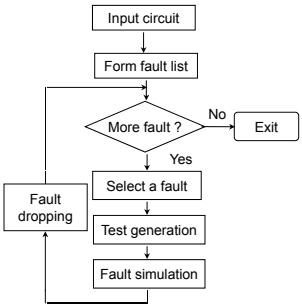
To propagate fault effect to the primary output E must be 1. Thus C must be 0.

Test vector: A=1, B=1, C=0

ATPG

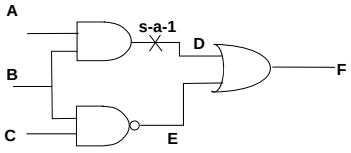
Automatic Test Pattern Generation

- **Given a circuit, identify a set of test vectors to detect all faults under consideration.**

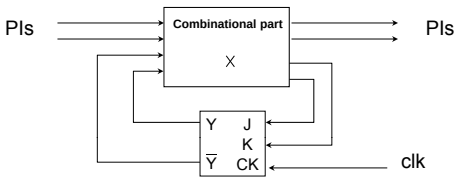


Difficulties in test generation

1. Reconvergent fanout

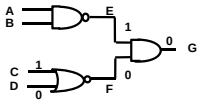


2. Sequential test generation



Circuit Modeling

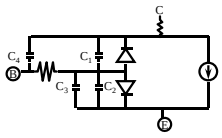
- **Functional model---** logic function
  - **f(x1,x2,...)=...**
  - **Truth table**
- **Behavioral model---** functional + timing
  - **f(x1,x2,...)=... , Delay = 10**
- **Structural model---** collection of interconnected components or elements



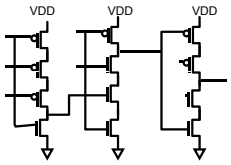
⇒ **All can be described in Verilog**

Levels of description

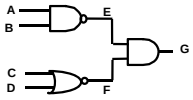
- **Circuit level**



- **Switch level**



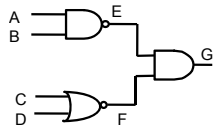
- **Gate level**



- **Higher/ System level**

Fault modeling

- **The effects of physical defects**
- **Most commonly used fault model: Single stuck-at fault**



A s-a-1 B s-a-1 C s-a-1 D s-a-1
A s-a-0 B s-a-0 C s-a-0 D s-a-0

E s-a-1 F s-a-1 G s-a-1
E s-a-0 F s-a-0 G s-a-0

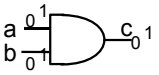
14 faults

- **Other fault models:**
  - Break faults, Bridging faults, Transistor stuck-open faults , Transistor stuck-on faults, Delay faults

Fault coverage (FC)

$$FC = \frac{\text{\# faults detected}}{\text{\# faults in fault list}}$$

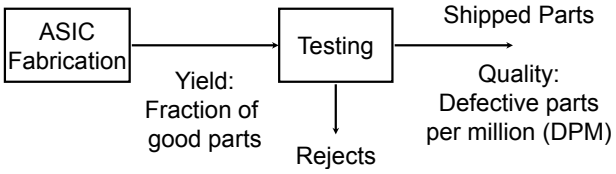
Example:



6 stuck-at faults (a_0,a_1,b_0,b_1,c_0,c_1)

Test	faults detected	FC
{(0,0)}	c_1	16.67%
{(0,1)}	a_1,c_1	33.33%
{(1,1)}	a_0,b_0,c_0	50.00%
{(0,0),(1,1)}	a_0,b_0,c_0,c_1	66.67%
{(1,0),(0,1),(1,1)}	all	100.00%

Testing and Quality



- **Quality of shipped parts is a function of yield Y and the test (fault) coverage T**
- **Defect level (DL) : fraction of shipped parts that are defective**

Defect Level, Yield & Fault Coverage

$$DL = 1 - Y^{(1-T)}$$

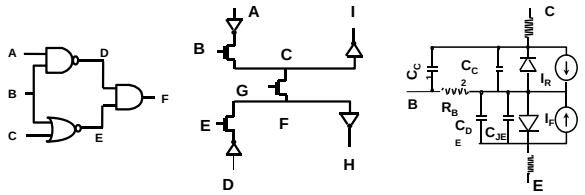
DL: defect level
Y: yield
T: fault coverage

Yield (Y)	Fault Coverage (T)	DPM (DL)
50%	90%	67,000
75%	90%	28,000
90%	90%	10,000
95%	90%	5,000
99%	90%	1,000
90%	90%	10,000
90%	95%	5,000
90%	99%	1,000
90%	99.9%	100

Logic simulation

To determine how a good circuit should work

- **Given input vectors, determine the normal circuit response**

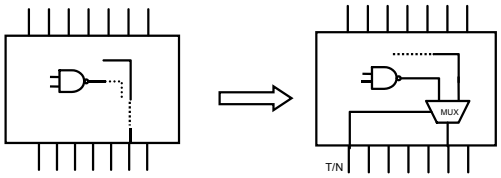


Testable Design

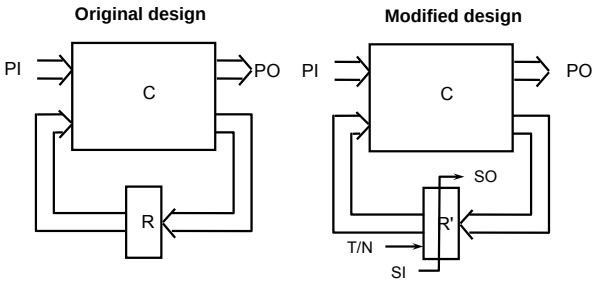
- **Design for testability (DFT)**
  - ***ad hoc* techniques**
  - **Scan design**
  - **Boundary Scan**
- **Built-In Self Test (BIST)**
  - **Random number generator (RNG)**
  - **Signature Analyzer (SA)**
- **Synthesis for Testability**

Example of ad hoc techniques

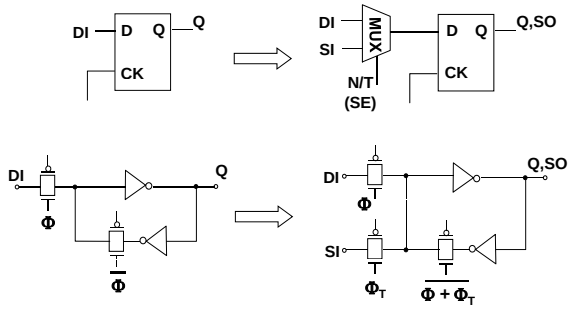
Insert test point



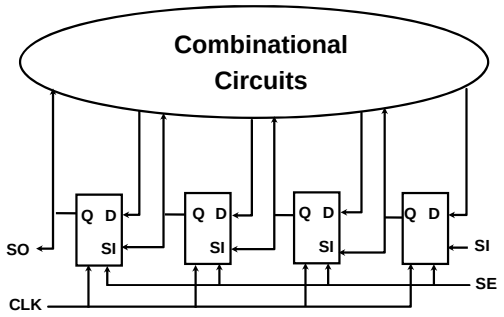
Scan System



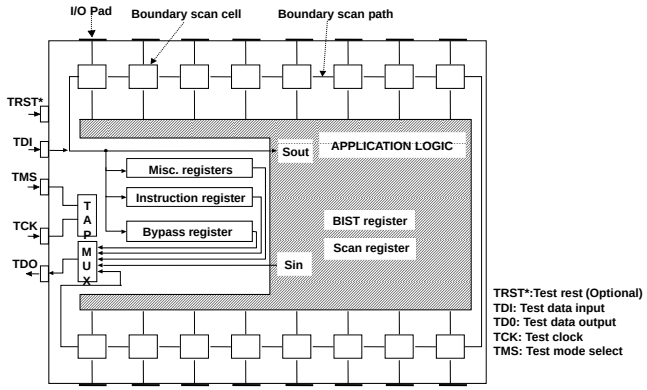
Scan Cell Design



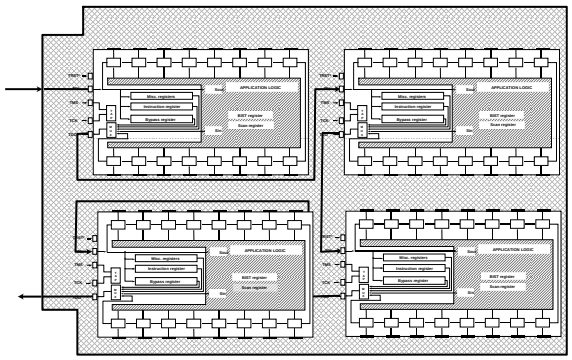
Scan Register



Boundary Scan

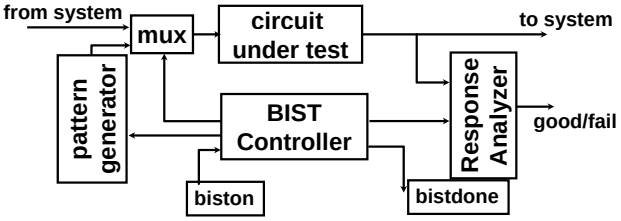


Boundary Scan (Cont.)



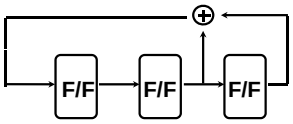
Built-In-Self Test (BIST)

- **Places the job of device testing inside the device itself**
- **Generates its own stimulus and analyzes its own response**



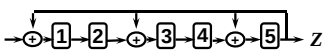
Built-In-Self Test (BIST) (cont.)

- **Two major tasks**
  - **Test pattern generation**
  - **Test result compaction**
- **Usually implemented by linear feedback shift register**



Signature Analyzer (SA)

Input sequence 11110101 (8 bits)



$G(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1$ $P(x) = 1 + x^2 + x^4 + x^5$

Time	Input stream	Register contents	Output stream
0	1 0 1 0 1 1 1 1	0 0 0 0 0 ←	Initial state
1	1 0 1 0 1 1 1	1 0 0 0 0	
.	.	.	
.	.	.	
5	1 0 1	0 1 1 1 1	
6	1 0	1 0 1 1 1	1
7	1	0 1 0 1 1	0 1
8		0 0 1 0 1	1 0 1
		Remainder	Quotient
		$R(x) = x^2+x^4$	$1+x^2$

Signature Analyzer (SA) (cont.)

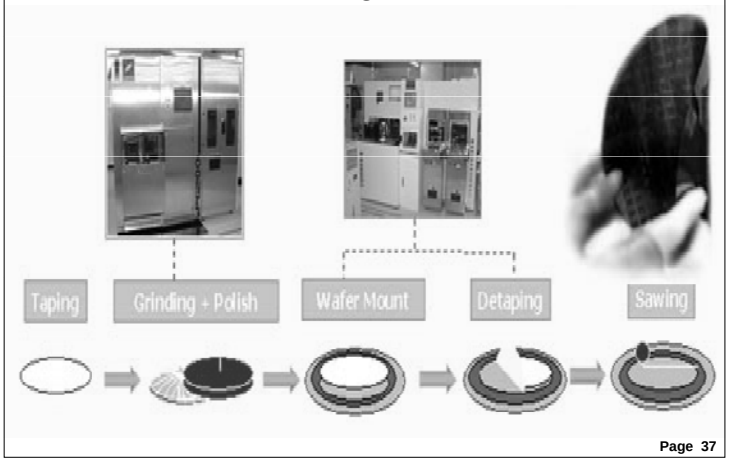
$$P(x): x^5 + x^4 + x^2 + 1$$
$$\times\ Q(x): x^2 + 1$$
$$x^7 + x^6 + x^4 + x^2 + x^5 + x^4 + x^2 + 1$$
$$= x^7 + x^6 + x^5 + 1$$

$$P(x)Q(x) + R(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1 = G(x)$$

Prob. of aliasing error = $1/2^n$

where n is # of FFs

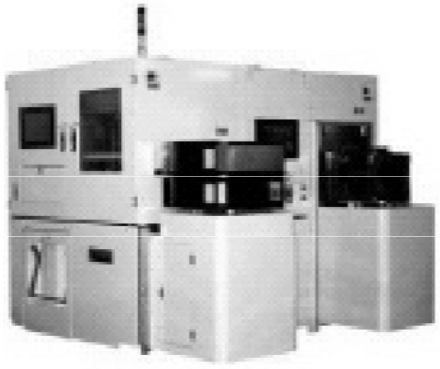
Automatic Test Equipment (ATE)
Flow



Automatic Test Equipment (ATE)
Wafer Laminater

Automatic Test Equipment (ATE)
Wafer Backside Grinder

Automatic Test Equipment (ATE)
Wafer Mounter



Automatic Test Equipment (ATE)
Auto Dicing Saw

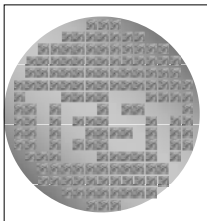
Automatic Test Equipment (ATE)
Auto UV Irradiation

Automatic Test Equipment (ATE)

Die Attach

Automatic Test Equipment (ATE)

Handler



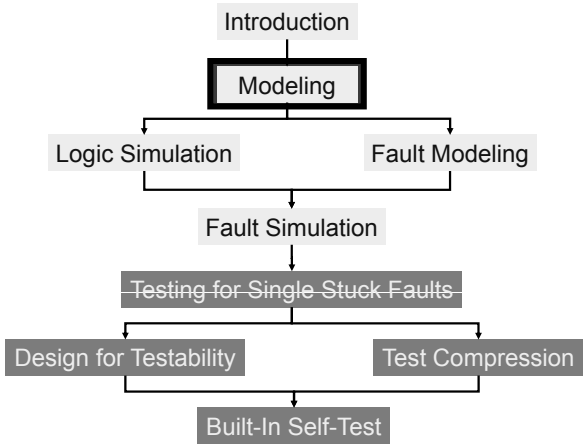
VLSI Test

Tsung-Chu Huang

Department of Electronic Engineering
National Changhua University of Education
Email: tch@cc.ncue.edu.tw

2016/02/22

Syllabus & Chapter Precedence

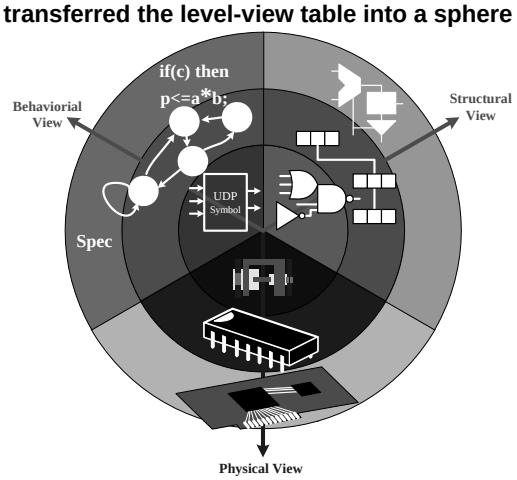


Levels & Views

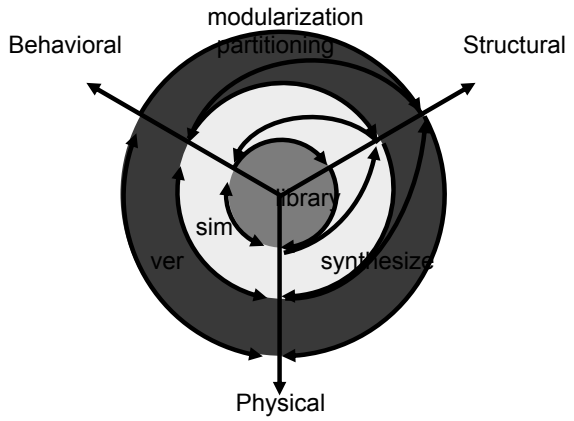
	Functional-View	Structural-View	Physical-View
System-Level			
Board-Level			
Register-Level			
Logic-Level			
Circuit-Level			
Transistor-Level			
Physics-Level			

Y-Chart

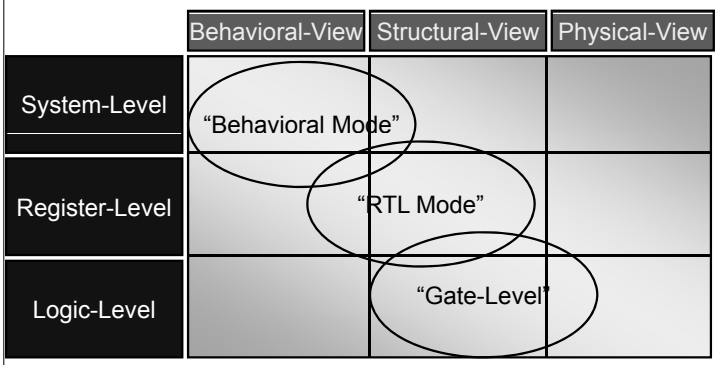
Gajski transferred the level-view table into a sphere chart:



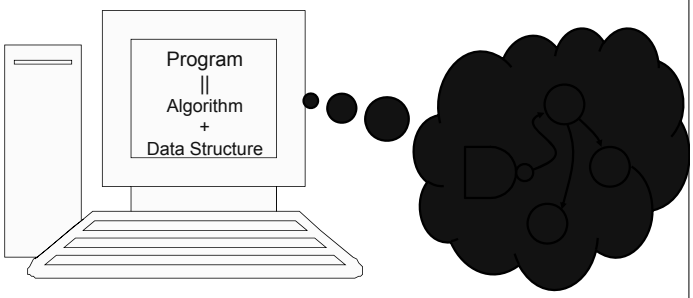
SoC EDA



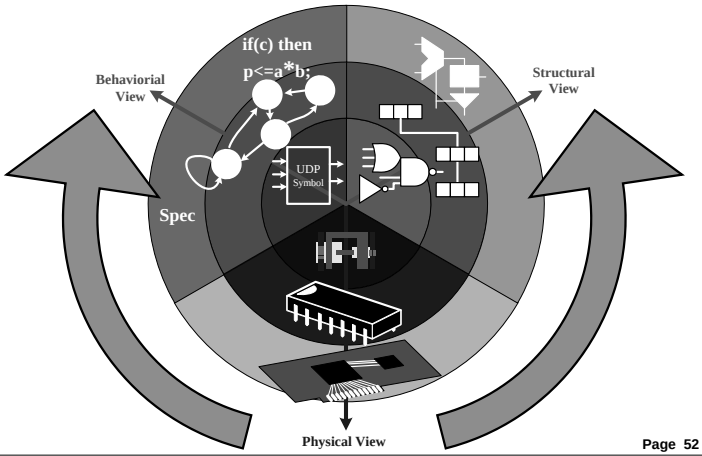
Usual Levels & Views



Internal & External Models



Structural & Functional Models



Structural Models

Recursive Definition

1. **Symbol View: A system or circuit can be represented by a symbol with its function.**
2. **A Structural View of a system (or circuit) is a representation or model that consists nodes for subsystems (or subcircuits) and elements (or components) represented by some symbol view, and arcs for their input/output relations.**
3. **Usually hierarchical.**
4. **The bottom-level boxes (components) are called primitive elements, which functional model is assumed to be known such as and, nand, or, nor, not, etc.**

Structural Modeling at Logic Level

1. **External Representation**
2. **Structural Properties**
3. **Hardware Descriptive Languages**
4. **Internal Representation**
5. **Example: A Simple Verilog Parser in C Language.**

External Representation

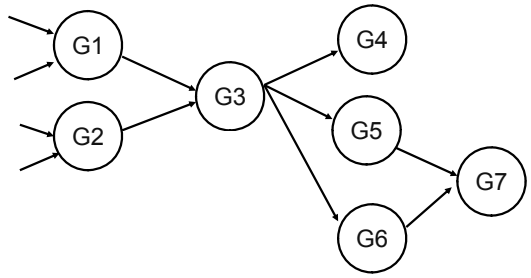
1. Text or schematic for human.
   Connectivity specifies I/O, components with signals.
2. Text or Language:
   HDL: VHDL, Verilog, C, etc.
   Netlist: SPICE, gat, tdl, etc.
3. Schematics:
   Cadence Schedmatics
   OrCAD Schedmatics, ..etc.

Structural Properties in Logic Level

1. Branches/Fanout
2. Stem
3. Fanout-free
4. Reconvergent Fanout
5. Gate Type
6. Inversion
7. Inversion Parity
8. Level of a Gate in Circuit

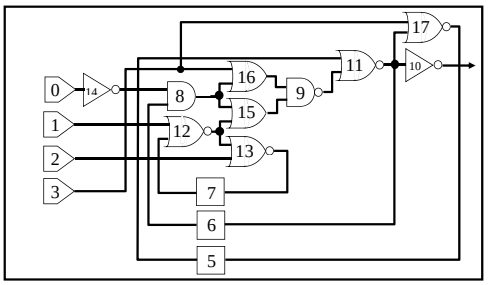
Structural Properties in Logic Level

Example



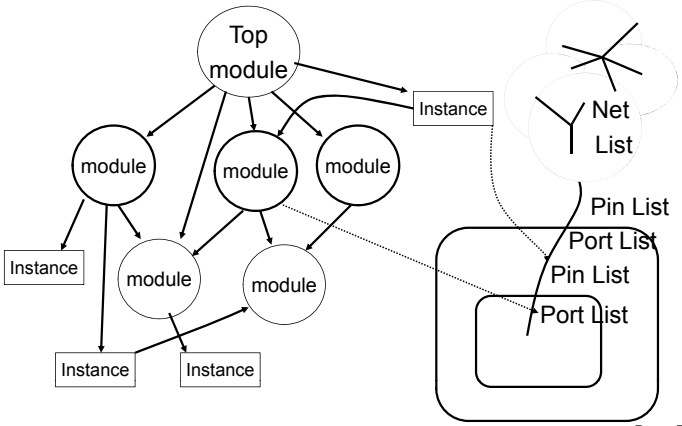
Structural Properties in Logic Level

Example: s27 in ISCAS89 benchmark

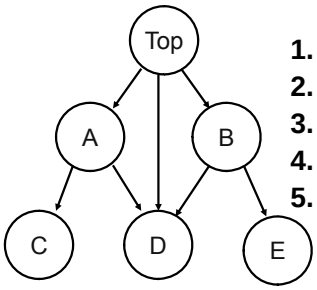


Internal Representation

Example: My Simple Verilog Parser



Structure-Oriented Design Methodology



1. Top-Down Design
2. Bottom-Up
3. Middle-Out
4. Ends-In
5. Hybrid (Greedy)

Functional Modeling at Logic Level

1. **Truth Table & Primitive Cubes.**
2. **State Table & Flow Table.**
3. **Binary Decision Diagrams.**
4. **Programs as Functional Models.**

Truth Table & Primitive Cubes

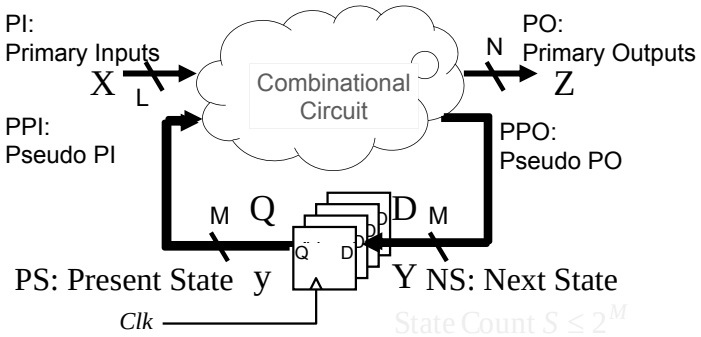
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

A	B	C	F
X	1	0	0
1	1	X	0
X	0	X	1
0	1	1	1

$F = \overline{B} + \overline{A}BC$

Huffman Model for a Finite State Machine

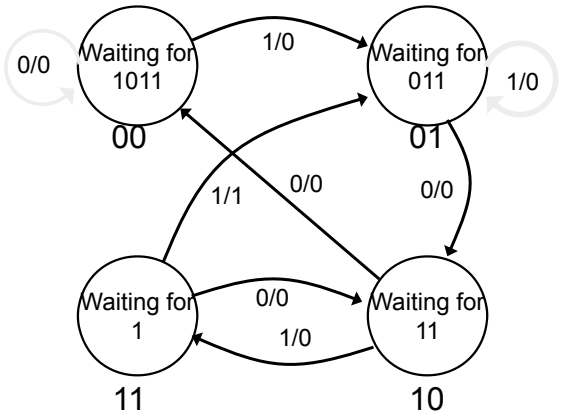
Single Clock, Synchronous, DFF-based



State Count $S \le 2^M$

State Diagram

Example: A Lock with Password 1101



State Table & Flow Table

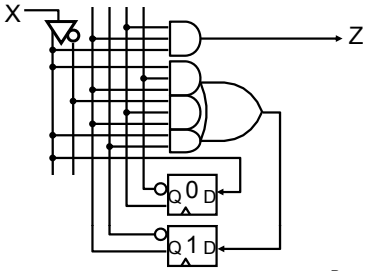
Q		X	D		Z
Q_1	Q_0	X_0	D_1	D_0	Z_0
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	0	1	1

K-map or
McKlusky Method:

$D_1 = \overline{Q}_1 X + Q_1 Q_0 \overline{X} + Q_1 \overline{Q}_0 X$

$D_0 = X$

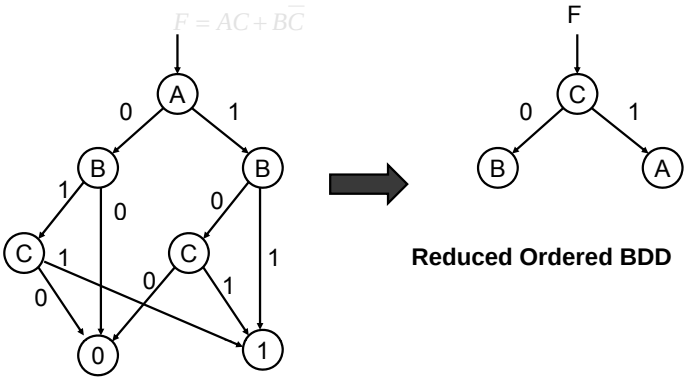
$Z = Q_1 Q_0 X$



Binary Decision Diagrams

1. **A Binary Decision Diagram (BDD) is a graph model of the function of a circuit.**

$F = AC + B\overline{C}$



Programs as Functional Models

1. **Assembly-Like:**

```
LDA     A
AND     B
INV
OR      C
STA     F
```

2. **C-Like:**

```
main(F, A, B, C)
{    int F, A, B, C;
     F =  ~ ( A && B ) || C;
}
```

3. **C-Code is the direct and fast simulator.**

```
C=A+B;
A=R1*R2;
B=~B;
```


Functional Modeling at Register Level

1. **Basic RTL Constructs**
2. **Timing Modeling in RTLs**
3. **Internal RTL Models (See Internal structural model)**

Basic RTL Constructs

1. **RTL (Register-Transfer Language in Register-Transfer Level)**

PC←PC+1; A←A+B+C
PC←PC+1; if C=1 PC←#FA07
R3←A – R4; A←0

2. **C or Verilog-like Language:**

```
module Adder(Co, Sum, A, B, Ci);
    Input       [15:0]   A, B;
    Input       Ci;
    output      Co;
    output      [15:0]   Sum;
    assign  {Co, Sum} = A + B + Ci;
endmodule
```

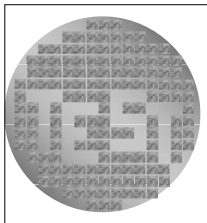

Timing Modeling in RTLs

1. **Verilog-like Code:**

```
Sum = #20 A+B;
initial
    begin
        A=1; B=0;
        #10 B=1; A=0;
        #40 B=0;
        #10 $stop;
    end
```


Quick Tutorial on Verilog

1. **Usual version:**
    1. VerilogXL/Cadence
    2. Verilog/Altera
    3. Verilog/Xilinx
    4. Verilog/MyCAD
    5. SynaptiCAD Verilogger
    6. VeriWell
    7. e.t.c.
2. **Structural View Gate Level;**
3. **Behavorial View RTL Level;**
4. **Finite State Machine**
5. **Memory Module**



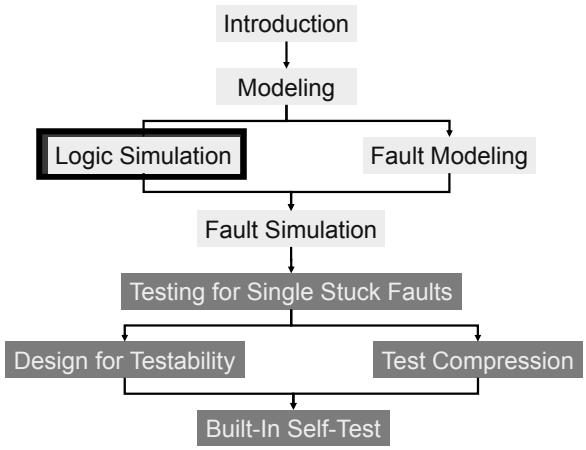
VLSI Test

Tsung-Chu Huang

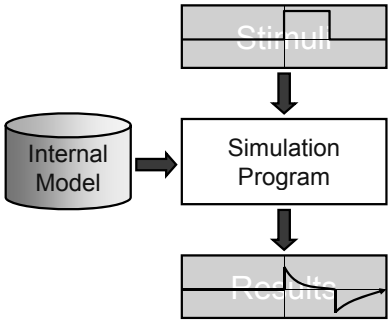
Department of Electronic Engineering
National Changhua University of Education
Email: tch@cc.ncue.edu.tw

2016/02/22

Syllabus & Chapter Precedence



Simulation Process



Applications of Logic Simulations

1. **Simulation Based Verification (Design Verification Test): Verification for Design Errors**
2. **Simulation Based Pattern Generation**
3. **Debugging Software/Microcode**
4. **IP Modeling**

Classification of Verification

1. **Formal Verification**
   1. **Axioms (Facts, Primitives)**
   2. **Theorems (Rules, Structural Relations)**
   3. **Proof (Derivation)**

- **Informal Verification**
  - **Simulation-based**

Problems of Simulation-Based Verification

1. **How does one generate the input stimuli (i.e. design verification test generation)?**
2. **How does one know the results are correct?**
3. **How good (complete) are applied input stimuli? (design test evaluation)**

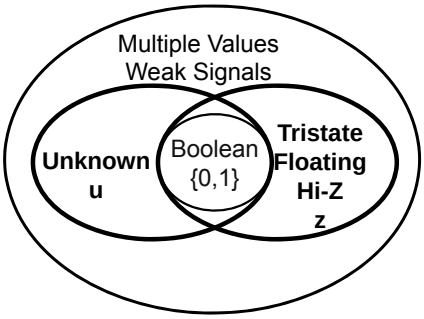
Types of Simulations

1. **Compiler-driven simulator**
2. **Table-driven simulator**
3. **Activity-driven simulator**
- ⊂ **Event-Driven simulator**

Levels of Simulations

1. **Register-Level**
2. **Functional-Level**
3. **Gate-Level**
4. **Mixed-Level**
    1. **Time-domain**
    2. **Frequency-domain**
5. **Transistor-Level**

Simulated Values



Don't-care, X is a union of some values, that is different from u.

The Unknown Logic Value

u

1. **u represents one value of {0, 1}.**

AND	0	1	u
0	0	0	0
1	0	1	u
u	0	u	u

OR	0	1	u
0	0	1	u
1	1	1	1
u	u	1	u

NOT	0	1	u
	1	0	u

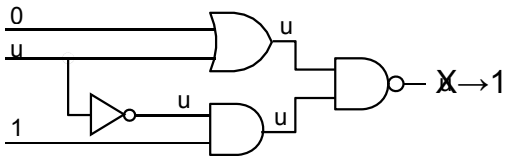
2. **u vs. X**

∩	0	1	X	u
0	0	∅	0	∅
1	∅	1	1	∅
X	0	1	X	u
u	∅	∅	u	u

3-Value Logic Simulation

Example

1. **Loss of Information in 3-V LSim**

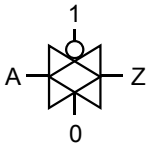


2. **The fourth value u may solve the above problem but those with more two states.**
3. **u may indicate an oscillation or glitch in some simulation systems.**

The Hi-Z Logic Value

z

1. It can be looked as {weak-0, weak-1, intermediate-value}



2. 4-V logic (0, 1, Z, X) is used in many HDL, where X is not don't_care but unknown.
3. Logic operations except wired-logic are conservatively derived into an unknown value X.

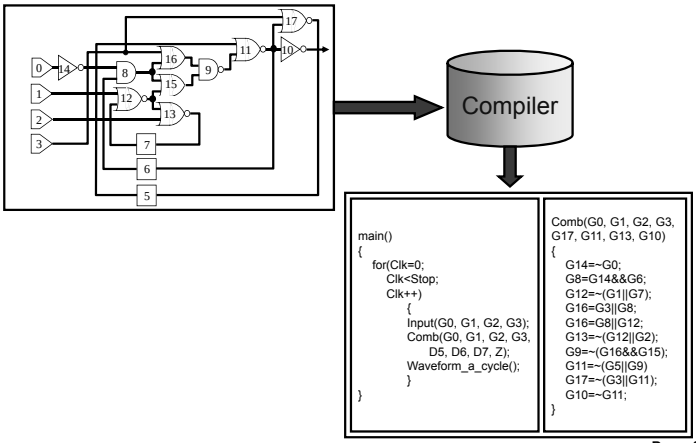
Compiled Simulation

Compiler-based Simulator

1. **Usually the combinational circuit is levelized into a direct acyclic graph (DAG) from inputs.**
2. **Each gate is represented by the generic primitive of the compiled language, say, assembly or C.**
3. **Circuit-timing is difficult to simulate.**
4. **The compiled code (usually called C-code) is the fast functional simulation approach.**

Compiled Simulation

Example



Events in HDLs

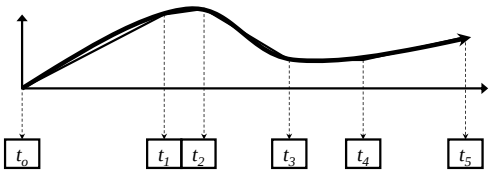
Signal Events

1. **Envents are Input and State Changes.**
2. **Primitive events correspondent to some signal Clk includes its rise (posedge Clk) and fall (negedge Clk).**
3. **Composite events are the union of more than 2 evnets.**
4. **The change of Clk is an composite event (Clk).**
5. **They are propagated using a structural model of a circuit.**
6. **Terminology:**
    1. **Circuit simulated time, simulation time, circuit time.**
    2. **CPU time.**
    3. **Compiling time.**

Event-Driven Simulation

Time-Wheel

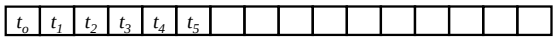
1. **Only simulated times are extracted from real time.**



Event-Driven Simulation

Time-Wheel

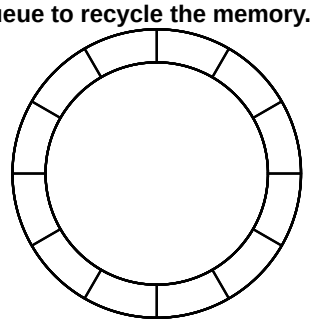
1. **The conceptual simulation time is recorded to a dynamic time list.**



Event-Driven Simulation

Time-Wheel

1. **The simulation time is twisted into a time wheel using a ring as a queue to recycle the memory.**

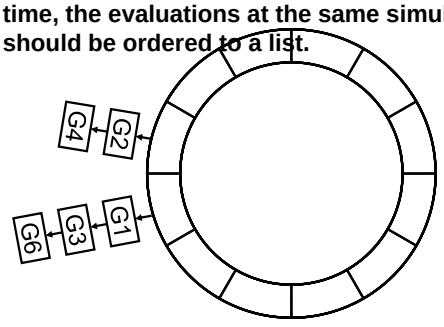


2. **The simulation time is twisted into a time wheel using a ring as a queue to recycle the memory.**

Event-Driven Simulation

Time-Wheel

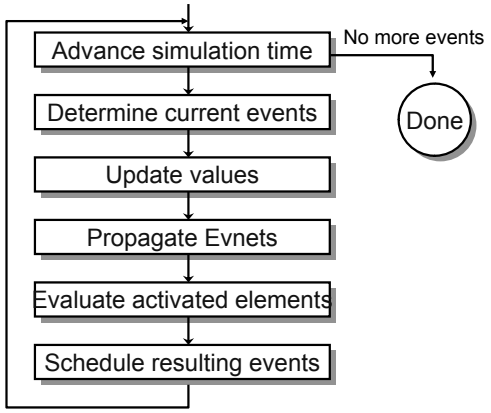
1. **Since the computation is actually sequential in real time, the evaluations at the same simulation time should be ordered to a list.**



2. **New events are generated by function evaluating and event propagating.**

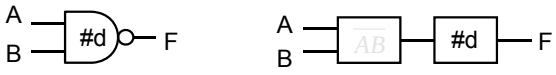
Event-Driven Simulation

Main Flow



Delay Models

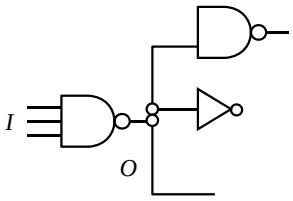
1. **The basic (functional) delay model is that of a transport delay, which specifies the interval d (units) separating an output change from input changes which caused it.**



2. **inertial delay: the min. duration to activate delay transport.**
3. **Spikes (<inertial) will be filtered by the gate.**
4. **Slightly different btw I/O based inertial delays.**
5. **Exhaustively, for N-input, M-output functional element, 4MN delays are considered.**
6. **3 values (min, typical, max) are extended in many HDLs.**

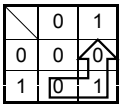
Delay Models in Uniform Transistor Sizing

1. **The delay is approximately proportional to its fanout, thus one circuit-dependent delay model derives from $d \sim RC = IO d_{\square}$.**
2. **Due to the stack effect, R is proportional to the input count / in uniform tarsistor sizing.**
3. **C is proportional to its fanout *O*.**



Hazard Detection

1. **Static**



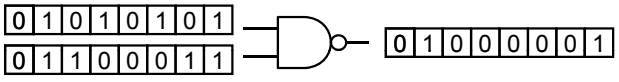
2. **Dynamic: similarly for >3 inputs or with feedback**

Basic Detection:

1. **Consider combinational network *C* and the changed input *I'* btw time *t* and *t+1*.**
2. **Stimulate each feedback wire or changed input with unknown value *u* to compare.**

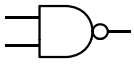
Logic Simulation Parallelization

1. **When the operation sequence and the bitwise dimension are independent.**
2. **Taking advantage of the parallelism of the word size in the computer.**
3. **Usually, a word (32 bits for a 32-bit computer) of gate-values are propagated and evaluated through the DAG-based combinational circuit.**



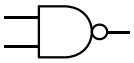
Internal Model of a Gate for Logic Simulation

```
typedef struct _gate GATE;

struct _gate {
     GATE *in1, *in2;
     GATE *out;
     TYPE type;
     int  value;
     };

typedef enum _type TYPE;
enum _type {AND, OR, NAND, NOR, INV};
```



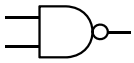
Internal Model of a Gate for Logic Simulation

```
typedef struct _gate GATE;

struct _gate {
     GATE *in[10];
     GATE *out;
     TYPE type;
     int  value;
     };
```



Internal Model of a Gate for Logic Simulation

```
typedef struct _gate GATE;

struct _gate {
     GateList  *in;
     GATE *out;
     TYPE type;
     int  value;
     };

typedef struct _gatelistGateList;
struct _gatelist {
     GATE *gate;
     GateList *next;
     }
```



Element Evaluation

1. **(Truth) Table Look-up:** **exponential, only for elements.**
2. **Dummy Input:** **for almost uniform input counts**
3. **Input Scanning:** **for small input counts**
   1. **Controlling value *c*:** **0 for AND-type gates**
      1 for OR-type gates
   2. **Inversion *i*:** **0 for AND, OR**
      1 for NAND, NOR, NOT

```
eval(g, c, i) {
   u_value = false;
   for every input value v of g
      if (v == c) return (c ^ i); else if (v == u)
      u_value = true;
   if u_value return u; else return !c ^ i;
}
```

4. **Input Counting:** **for large input counts**

Potential Topic of the Project

- **Write a logic simulation program to generate the output vectors after parsing the ISCAS85 benchmark circuits and a list of input vectors.**

```
sim input.txt c17.tdl output.txt
```