AUSIM: Auburn University SIMulator
A Simulator for Education & Research

Chuck Stroud
Dept. of Electrical & Computer Engineering
Auburn University

Built-In Self-Test
Outline

- Overview of Simulation Capabilities
- ASL (Auburn Simulation Language)
  - Format and Features
- AUSIM (Auburn University SIMulator)
  - Logic Simulation
    - Capabilities and Features
  - Fault Simulation
    - Capabilities and Features
- Summary and Conclusions
  - Future Development
Overview of Simulation Capabilities

- Compiled simulator (aka unit delay)
- Logic simulation for design verification
  - Simple versions for ELEC 2200
- FPGA configuration memory bits
- Fault simulation features
  - Serial and parallel simulations
  - Single and multiple fault simulation
  - Fault models supported
    - Gate-level stuck-at-fault
      - Collapsed vs. uncollapsed
    - Bridging faults
      - Dominant
      - Dominant-AND/dominant-OR
  - Serial fault simulation specialties
    - Oscillation faults
    - Notrip and N-detect simulation
- Fault profiling
Auburn Simulation Language

- Hierarchy (subcircuits)
- Connection via signal names (or netnames)
  - Positional notation
- Bus notation
- Delimiters
  - White spaces

# ASL description of 2-input MUX;
subckt: MUX in: A B S out: Z;
not: G1 in: S out: SN;
and: G2 in: A SN out: G2;
and: G3 in: S B out: G3;
or: G4 in: G2 G3 out: Z;

# ASL description of 4-input MUX;
ckt: MUX4 in: In[0:3] S[0:1] out: Out;
MUX: M1 in: In0 In1 S0 out: M1;
MUX: M2 in: In2 In3 S0 out: M2;
MUX: M3 in: M1 M2 S1 out: Out;
ASL Statements

- Case sensitive - X1 and x1 are two different names
- All statements can be multi-line with new-line delimiters
- Comment statements
  - Begin: ‘#’ followed by delimiter
  - End: delimiter followed by ‘;’
  - Include textual information or remove portions of circuit w/o deleting
    - Cannot be nested inside circuit or component statements
- Circuit statements
  - Define attributes of circuit or subcircuit
    - Name of circuit or subcircuit
    - Inputs (in:) Configuration bits (con:) Outputs (out:)
    - SUBCKT: mux IN: a b CON: s OUT: z ;
- Component statements
  - Defines name & I/O connections of a component (gate or subcircuit)
    - MUX: m1 IN: i0 i1 CON: sel OUT: mout ;
ASL Reserved Characters

- Can not (or should not) be used in names:
  - _ (underscore)
    - Used to construct names when flattening hierarchy
  - # (pound sign)
    - Denotes beginning of comment statement
  - ; (semicolon)
    - Denotes end of all types of statements
  - : (colon)
    - Denotes last character of keyword
  - [ and ] (square brackets)
    - Used for bus notation
      - Supported only in advanced versions of AUSIM
Bus Notation

- Saves time and reduces design errors
  - Supported in hierarchical ASL
    - Bus expansion during flattening of hierarchy
  - Must be consecutive numbering
  - Can be anywhere in name
  - Only one bus notation per name

- Examples:
  - \texttt{DIN[7:4]} produces
    - DIN7 DIN6 DIN5 DIN4
  - \texttt{D[1:3] D[10:5]} produces
    - D1 D2 D3 D10 D9 D8 D7 D6 D5
  - \texttt{[19:25]junk} produces
    - 19junk 20junk 21junk 22junk 23junk 24junk 25junk
  - \texttt{Mr[4:0]Dude} produces
    - Mr4Dude Mr3Dude Mr2Dude Mr1Dude Mr0Dude
ASL Keywords

- **CKT**: (ckt:) and **SUBCKT**: (subckt:)
  - Denotes circuit and subcircuit statements
- **IN**: (in:) and **OUT**: (out:)
  - Denotes inputs & outputs of circuit, subcircuit, or gate
- **CON**: (con:)
  - Denotes configuration bits for FPGAs/CPLDs
- **AND**: (and:), **OR**: (or:), **NAND**: (nand:), **NOR**: (nor:), & **NOT**: (not:)
  - Denotes elementary logic gate component statements
  - Can have any number of inputs (except for NOT gate)
Keywords for Functional Models

- **XOR**: (xor:) and **NXOR**: (nxor:)
  - Denotes exclusive-OR and exclusive-NOR gates
  - Any number of inputs

- **MUX2**: (mux2:)
  - Denotes 2-to-1 multiplexer

- **DFF**: (dff:) and **NDFF**: (ndff:)
  - Denotes rising and falling edge-triggered D flip-flops

- **LAT**: (lat:) and **NLAT**: (nlat:)
  - Denotes active high and active low level-sensitive D latches

- **RAM**: (ram:), **WRAM**: (wram:) and **DRAM**: (dram:)
  - Denotes single port, separate write/read port, and dual-port with active high write enable
ASL for ELEC 2200 AUSIM

- ELEC 2200 versions of AUSIM support:
  - AND:, OR:, NOT:, NAND:, & NOR:
    - Does not include XOR: since not elementary gate
      - Considering inclusion in later versions
  - Functional models for flip-flops include:
    - DFF: (dff:)
      - Input order: CLK, D
    - SRFF: (srff:)
      - Input order: CLK, S, R
    - JKFF: (jkff:)
      - Input order: CLK, J, K
    - Output order for all flip-flops: Q, QN (optional)
AUSIM Files

- AUSIM checks for missing input files
- All input and output files are ASCII text
- Any text editor can be used to create input files
  - Be sure to save as text file
    - Don’t save as .rtf or .doc
    - When using default file names be sure to delete .txt suffix from some editors (like NotePad) before executing AUSIM
      - Example: change name.asl.txt to name.asl
Design Verification

- ckt: MUX in: A B S out: Z ;
- not: G1 in: S out: SN ;
- and: G2 in: A SN out: G2 ;
- and: G3 in: S B out: G3 ;
- or: G4 in: G2 G3 out: Z ;

Logic Simulation

- data structures hold logic values
- pointers to source logic values
- logical operations

VLSI Design & Test Seminar

3/7/07
ELEC 2200 AUSIM Audit File

Loading and delays:
Name | Loads | Driver | Delay=intrinsic+extrinsic:
--- | --- | --- | ---
w | 4 | Input | 4=0+4
x | 9 | Input | 9=0+9
y | 5 | Input | 5=0+5
z | 7 | Input | 7=0+7
a | 0 | OR | 4=4+0 Output
b | 2 | OR | 4=2+2 Output
h | 0 | AND | 2=2+0 Output
y | 3 | NOT | 4=1+3 Output
...
p8 | 2 | NOR | 4=2+2
p9 | 1 | NOR | 3=2+1
p10 | 2 | NOR | 4=2+2
...
p17 | 1 | NOR | 3=2+1
p18 | 1 | OR | 4=3+1

Timing path analysis only in ELEC 2200 versions of AUSIM

2’s comp octal to 8-seg decoder

Digital Logic Circuit

8-segment display
AUSIM Input Files (default names)

- Both logic and fault simulation need
  - ASL (name.asl)
  - Library file (name.lib) – optional ASL file with subckt descriptions
    - Advanced version of AUSIM looks for .lib file
      - an empty file or file with a valid comment will do
  - Configuration bit file (name.con) if CKT: statement has CON: keyword and list of config bits (values can be multiple strings)

- Logic simulation only
  - Input vector file (name.vec) (each vector must be one string)
  - Optional: to monitor internal node file (name.nod)

- Fault simulation only
  - Simulation results file (name.out)
    - Generated by AUSIM during logic simulation (simul8 command)
  - Fault list (name.flt) – list of faults to be emulated
    - Can be generated by user or by AUSIM
      - fltgen command generates gate-level stuck-at faults
      - bftgen command generates bridging faults
Fault List Generation

- **uncol** – sets flag to generate uncollapsed fault list
  - Otherwise, a collapsed fault list is generated removing:
    - Gate-level equivalent faults
    - Structural equivalent faults

- **fltgen** – generates gate-level stuck-at fault list

- **bftgen** **param** – generates bridging fault list
  - Where **param** specifies bridging fault model:
    - **dom** – for dominant bridging faults
    - **dand** – for dominant-AND bridging faults
    - **dor** – for dominant-OR bridging faults
  - Warning - bridging fault list generation is based on consecutive pairs of nets in same order as in **name.net**
    - Not all possible combinations of bridging faults generated
  - No fault collapsing for bridging faults
Gate Level Equivalent Faults and Fault Collapsing

Collapsed fault set
- $Z_{sa0} = A_{sa0} = B_{sa0}$
- $Z_{sa1} = A_{sa1} = B_{sa1}$

### AND

<table>
<thead>
<tr>
<th>AB</th>
<th>$Z$</th>
<th>$A_{sa0}$</th>
<th>$A_{sa1}$</th>
<th>$B_{sa0}$</th>
<th>$B_{sa1}$</th>
<th>$Z_{sa0}$</th>
<th>$Z_{sa1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### NOT

<table>
<thead>
<tr>
<th>AB</th>
<th>$Z$</th>
<th>$A_{sa0}$</th>
<th>$A_{sa1}$</th>
<th>$Z_{sa0}$</th>
<th>$Z_{sa1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### OR

<table>
<thead>
<tr>
<th>AB</th>
<th>$Z$</th>
<th>$A_{sa0}$</th>
<th>$A_{sa1}$</th>
<th>$B_{sa0}$</th>
<th>$B_{sa1}$</th>
<th>$Z_{sa0}$</th>
<th>$Z_{sa1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

For uncollapsed fault list include *uncol* before *fltgen* command
Default *fltgen* command is a collapsed fault list
Structural Equivalent Faults and Fault Collapsing

Cannot collapse faults at fan-out stem since it would violate single stuck-at fault model.

Fan-out stem sa1 = K sa1 & J sa1
Fan-out stem sa0 = K sa0 & j sa0

Only 2 collapsed faults for inverter chain (Z sa0 & Z sa1)

AUSIM collapses toward outputs
Collapsed vs. Uncollapsed Faults

For a given circuit (assuming elementary logic gates)

- # uncollapsed faults = \(2(G + G_I)\)
  - \(G\) = total # gates
  - \(G_I\) = total # gate inputs

- # collapsed faults = \(2(O_P + F_S) + G_I - N_I\)
  - \(O_P\) = # primary outputs
  - \(F_S\) = # fan-out stems
  - \(N_I\) = # inverters

- typically # collapsed flts \(\approx \frac{1}{2}\) # uncollapsed flts

Should faults be collapsed?

- YES: for TPG and fault simulation (more efficient)
- NO: for computing fault coverage (more accurate)
  - but longer fault simulation times
Functional Model Gate-Level Faults

- No fault collapsing for functional models
  - Stuck-at-0 (sa0) and stuck-at-1 (sa1) at pins
- To detect all XOR pin faults:
  - Need 3 vectors {01, 10, and 00 or 11}
  - Mourad & McCluskey, Trans. on IE ‘89
  - But note that any 3 vectors will work
- To detect all gate-level faults:
  - Need all 4 vectors

<table>
<thead>
<tr>
<th>Fault-free</th>
<th>S</th>
<th>T</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sa0</td>
<td>sa1</td>
<td>sa0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 0</td>
<td>1 0</td>
<td>0 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>0 0</td>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>0 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>1 0</td>
<td>1 0</td>
<td>0 1</td>
</tr>
</tbody>
</table>
Bridging Fault Models

- **Wired-AND/Wired-OR faults (old bipolar technology)**
  - 1 vector (01 or 10) with 2 outputs (A’ and B’), or
  - 1 output (A’ or B’) with 2 vectors (01 and 10)

- **Dominant faults (better for CMOS technology)**
  - 1 vector (01 or 10) with 2 outputs (A’ and B’)
    - harder to detect that Wired-AND/OR (less observable)
    - detecting all dominant BFs ⇒ detects all Wired-AND/OR BF
  - Easier to emulate than wired-AND/OR BF

![Wired-AND fault model](image1)

<table>
<thead>
<tr>
<th>AB</th>
<th>A’B’</th>
<th>WAND</th>
<th>WOR</th>
<th>AdomB</th>
<th>BdomA</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

![Wired-OR fault model](image2)

- A dominates B model
- B dominates A model
A Newer Bridging Fault Model

- **Dominant-AND/Dominant-OR:**
  - Stronger driving gate dominates the short for only one logic value
    - Behavior has been observed in ASICs and FPGAs
  - Disadvantage: 4 faults per fault site
    - 2 vectors (01 and 10) with 2 outputs (A' and B')
      - harder to detect that dominant BFs (*less observable*)
  - Detecting all dominant-AND/OR BFs ⇒ detects all dominant BFs and all Wired-AND/OR BFs

### Diagrams

#### Dominant-AND fault model
- A: A
- B: B
- A': A'
- B': B'

#### Dominant-OR fault model
- A: A
- B: B
- A': A'
- B': B'

### Table
<table>
<thead>
<tr>
<th>AB</th>
<th>A'B'</th>
<th>AdandB</th>
<th>BdandA</th>
<th>AdorB</th>
<th>BdorA</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Bridging Fault Generation

- Consider all possible shorts between any two wires is not practical
  - Larger number of faults to simulate
    - For $N$ wires, # of possible fault sites = $N$-choose-2 = $(N^2-N)/2$
    - # faults = $N^2-N$ for dominant and wired-AND/OR bridging faults
    - # faults = $2(N^2-N)$ for dominant-AND/OR bridging faults
  - But wires at opposite sides of chip/PCB not likely to short
- Capacitance extraction identifies wires likely to short
- **bftgen** considers only pair-wise ordered net list
  - Generates $2(N-1)$ faults
  - Generate by hand

$$FC_{WBF} = \sum_{i=1}^{D} \frac{C_i}{T} \sum_{j=1}^{C_j} l_1$$

Higher probability of bridging faults
Bridging Fault Detection

- High gate-level stuck-at FC assumed to give high BFC
  - Good assumption for general logic
  - Not true for non-decoded multiplexer-based routing


<table>
<thead>
<tr>
<th>Bridging Fault Coverage</th>
<th>Original Test Vectors</th>
<th>New BF Test Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unweighted</td>
<td>57.5%</td>
<td>96.5%</td>
</tr>
<tr>
<td>Weighted</td>
<td>29.9%</td>
<td>99.994%</td>
</tr>
</tbody>
</table>

Cypress 39K FPGA manufacturing test development used AUSIM
AUSIM Output Files (default names)

- **General files**
  - Flattened ASL *(name.fas)*
    - Flattened ASL description produced by *keepfas* command
  - Circuit audit *(name.aud)*
    - Circuit statistics produced by *audit* command
  - Nets file *(name.net)*
    - Lexicon ordered list of nets in data structures produced by *nets* command
      - Same order as used for BF generation by *bftgen* command

- **Logic simulation files**
  - Simulation results *(name.out)*
    - Produced by logic simulation *simul8* command
AUSIM Output Files (default names)

- Fault simulation files by produced fault simulation commands
  - Detected fault list (name.det)
    - Includes vector and primary output where fault is first detected
  - Undetected fault list (name.udt)
  - Potentially detected fault list (name.pdt): faulty circuit = 2
    - Includes total # vectors for which fault is potentially detected
    - Faults written to .udt file since they are only potentially detected
  - Oscillation fault list (name.osc): faulty circuit oscillates
    - Serial fault simulation only
    - Recall that AUSIM is a compiled simulator
  - Fault detection profile (name.pro)
    - Gives # faults detected for vector set on a per vector & cumulative basis – produced by fltpro command
AUSIM Example

**mux.asl file contents:**

```asm
# ASL description of 2-input MUX ;
subckt: MUX in: A B S out: Z ;
not: G1 in: S out: SN ;
and: G2 in: A SN out: G2 ;
and: G3 in: S B out: G3 ;
or: G4 in: G2 G3 out: Z ;
```

**mux.vec file contents:**

```asm
# in: In[0:3] S[0:1] ;
000000
100000
011100
000001
010001
```

**mux.cnt file contents:**

```cnt
default mux
lib slop.lib
proc
audit
nets
fltgen
simul8
pftsim
fltpro
```
AUSIM Example

mux.cnt file:
default mux
lib slop.lib
proc
keepfas
audit
nets
fltgen
simul8
pftsim
fltpro

Prompt> ausim mux.cnt
AUSIM version 2.6
Begin ASL syntax check . . .
  Syntax appears to be OK!
4 SUBCKTls found in ASL file - circuit will be flattened
Begin LIB syntax check . . .
  Syntax appears to be OK!
Begin flattening ASL description . . .
  Flattening complete (12 gates)
Loading circuit data structures . . .
  Parsing Complete
Beginning audit of circuit 'MUX4' . .
  checking for duplicate gate input names . .
  checking connectivity of circuit . .
  Audit complete - circuit appears to be OK!
G=12, Gio=33, Pi=6, Pc=0, Po=1, Fo=0, Nets=18
ASL file processing complete
processing time = 60
Audits for Common Problems

- Unconnected gate inputs
- Multiple gates driving same net
  - AUSIM does not support 3-state or bi-directional buses
- Subcircuit statement & instantiation have different # of inputs, outputs, or config bits
- Duplicate names for gates, inputs, outputs, & config bits
- Reserved characters (#, _, ;) in names
- Multiple or missing CKT: statements in ASL file
- CKT: statement(s) in library file
  - Only SUBCKT: statements allowed in .lib file
- Missing IN: keyword after circuit/component name
- Incorrect # inputs to inverter, flip-flop, latch, or mux
- Configuration bits for gates (should not be any)
  - Includes all elementary logic gates and functional models
- Net names too long (current length 75 characters)
  - Before hierarchical flattening (not too useful)
AUSIM Example

**mux.cnt file:**
- flattened ASL for ckt 'MUX4' written to 'mux.fas'
- default mux
- lib slop.lib
- proc
- keepfas
- audit nets
- fltgen
- simul8
- pftsim
- fltpro

Begin Audit file generation . . .
results written to 'mux.aud'

Begin fault list generation . . .
20 faults generated for 'MUX4'
collapsing time = 10

Begin logic simulation for circuit 'MUX4' notrip=0, fltsim=0 ...
AUSIM logic simulation for circuit 'MUX4' complete for 12 vectors
logic simulation results written to 'mux.out'
logic simulation time = 40

Begin parallel fault simulation for circuit 'MUX4' notrip=0, fltsim=3 ...
AUSIM fault simulation for circuit 'MUX4' complete
20 faults were simulated
18 detected faults written to 'mux.det'
2 undetected faults written to 'mux.udt'
parallel fault simulation time = 70

Begin profiling fault detection ...
fault detection profile written in 'mux.pro'
AUSIM Example

mux.cnt file:
default mux
lib slop.lib
proc
keepfas
audit

mux.asl file contents:
# ASL description of 2-input MUX ;
subckt: MUX in: A B S out: Z ;
ot: G1 in: S out: SN ;
and: G2 in: A SN out: G2 ;
and: G3 in: S B out: G3 ;
or: G4 in: G2 G3 out: Z ;
# ASL description of 4-input MUX ;
ckt: MUX4 in: In[0:3] S[0:1] out: Out ;
MUX: M1 in: In0 In1 S0 out: M1 ;
MUX: M2 in: In2 In3 S0 out: M2 ;
MUX: M3 in: M1 M2 S1 out: Out ;

mux.fas file contents:
# Flattened ASL written from structures for 'MUX4' ;
CKT: MUX4 IN: In0 In1 In2 In3 S0 S1 OUT: Out ;
ot: M1_G1 IN: S0 OUT: M1_SN ;
and: M1_G2 IN: In0 M1_SN OUT: M1_G2 ;
and: M1_G3 IN: S0 In1 OUT: M1_G3 ;
or: M1_G4 IN: M1_G2 M1_G3 OUT: M1 ;
ot: M2_G1 IN: S0 OUT: M2_SN ;
and: M2_G2 IN: In2 M2_SN OUT: M2_G2 ;
and: M2_G3 IN: S0 In3 OUT: M2_G3 ;
or: M2_G4 IN: M2_G2 M2_G3 OUT: M2 ;
ot: M3_G1 IN: S1 OUT: M3_SN ;
and: M3_G2 IN: M1 M3_SN OUT: M3_G2 ;
and: M3_G3 IN: S1 M2 OUT: M3_G3 ;
or: M3_G4 IN: M3_G2 M3_G3 OUT: Out ;
**AUSIM Example**

**mux.cnt file:**
- default mux
- lib slop.lib
- proc
- keepfas
- audit

**mux.aud file contents:**
AUSIM (2.6) Audit Results Circuit 'MUX4'
- Number of Primary Inputs = 6
- Number of Primary Outputs = 1
- Number of gates = 12
- Number of gate I/O pins = 33
- Number of nets = 18
- Number of fan-out stems = 0
- Num of uncollapsed gate-level stuck-at flts = 66
- Num of collapsed gate-level stuck-at faults = 20

**Gate type and number of uses:**
- NOT: 3
- AND: 6
- OR: 3

**Loading Profile:**
- #loads #nets
- 0 1
- 1 15
- 2 1
- 4 1

**mux.asl file contents:**
- # ASL description of 2-input MUX ;
- subckt: MUX in: A B S out: Z ;
- not: G1 in: S out: SN ;
- and: G2 in: A SN out: G2 ;
- and: G3 in: S B out: G3 ;
- or: G4 in: G2 G3 out: Z ;
- # ASL description of 4-input MUX ;
- ckt: MUX4 in: In[0:3] S[0:1] out: Out ;
- MUX: M1 in: In0 In1 S0 out: M1 ;
- MUX: M2 in: In2 In3 S0 out: M2 ;
- MUX: M3 in: M1 M2 S1 out: Out ;
AUSIM Example

mux.cnt file:
default mux
lib slop.lib
proc
keepfas
audit
nets
fltgen
simul8
pftsim
fltpro

mux.net file contents:
In0
In1
In2
In3
M1
M1_G2
M1_G3
M1_SN
M2
M2_G2
M2_G3
M2_SN
M3_G2
M3_G3
M3_SN
Out
S0

mux.fas file contents:
# Flattened ASL written from structures for 'MUX4';
CKT: MUX4 IN: In0 In1 In2 In3 S0 S1 OUT: Out;
not: M1_G1 IN: S0 OUT: M1_SN;
and: M1_G2 IN: In0 M1_SN OUT: M1_G2;
and: M1_G3 IN: S0 In1 OUT: M1_G3;
or: M1_G4 IN: M1_G2 M1_G3 OUT: M1;
not: M2_G1 IN: S0 OUT: M2_SN;
and: M2_G2 IN: In2 M2_SN OUT: M2_G2;
and: M2_G3 IN: S0 In3 OUT: M2_G3;
or: M2_G4 IN: M2_G2 M2_G3 OUT: M2;
not: M3_G1 IN: S1 OUT: M3_SN;
and: M3_G2 IN: M1 M3_SN OUT: M3_G2;
and: M3_G3 IN: S1 M2 OUT: M3_G3;
or: M3_G4 IN: M3_G2 M3_G3 OUT: Out;
AUSIM Example

**mux.cnt file:**
default mux
lib slop.lib
proc
keepfas
audit
nets
fltgen
simul8
pftsim
fltpro

**mux.flt file contents:**
# collapsed faults for 'MUX4';
# gate net I/O s-a-f;
M1_G2 In0 in sa1
M1_G2 M1_SN in sa1
M1_G3 S0 in sa1
M1_G3 In1 in sa1
M1_G4 M1_G2 in sa0
M1_G4 M1_G3 in sa0
M2_G2 In2 in sa1
M2_G2 M2_SN in sa1
M2_G3 S0 in sa1
M2_G3 In3 in sa1
M2_G4 M2_G2 in sa0
M2_G4 M2_G3 in sa0
M3_G2 M1 in sa1
M3_G2 M3_SN in sa1
M3_G3 S1 in sa1
M3_G3 M2 in sa1
M3_G4 M3_G2 in sa0
M3_G4 M3_G3 in sa0
M3_G4 Out out sa0
M3_G4 Out out sa1

**mux.fas file contents:**
# Flattened ASL written from structures for 'MUX4';
CKT: MUX4 IN: In0 In1 In2 In3 S0 S1 OUT: Out;
not: M1_G1 IN: S0 OUT: M1_SN;
and: M1_G2 IN: In0 M1_SN OUT: M1_G2;
and: M1_G3 IN: S0 In1 OUT: M1_G3;
or: M1_G4 IN: M1_G2 M1_G3 OUT: M1;
not: M2_G1 IN: S0 OUT: M2_SN;
and: M2_G2 IN: In2 M2_SN OUT: M2_G2;
and: M2_G3 IN: S0 In3 OUT: M2_G3;
or: M2_G4 IN: M2_G2 M2_G3 OUT: M2;
not: M3_G1 IN: S1 OUT: M3_SN;
and: M3_G2 IN: M1 M3_SN OUT: M3_G2;
and: M3_G3 IN: S1 M2 OUT: M3_G3;
or: M3_G4 IN: M3_G2 M3_G3 OUT: Out;
AUSIM Example

mux.cnt file:
default mux
lib slop.lib
proc
keepfas
audit
nets
fltgen
simul8
pftsim
fltpro

mux.out file contents:
# AUSIM (2.6) Simulation Results ;
# IIII O ;
# nnnnSS u ;
# 012301 t ;
 000000 0
 100000 1
 011100 0
 000001 0
 010001 0
 101101 1
 000010 0
 001010 0
 110110 1
 000011 0
 000111 1
 111011 0

used as input to fault simulation for fault-free circuit response
AUSIM Example

**mux.cnt file:**
default mux lib slop.lib proc keepfas audit nets fltgen simul8 pftsim fltpro

**mux.det file contents:**
# faults detected for 'MUX4';
# gate net I/O s-a-f (vec,out);
M1_G2 In0 in sa1 (v=1,o=1)  
M1_G3 S0 in sa1 (v=3,o=1)  
M1_G3 In1 in sa1 (v=7,o=1)  
M1_G4 M1_G2 in sa0 (v=2,o=1)  
M1_G4 M1_G3 in sa0 (v=9,o=1)  
M2_G2 In2 in sa1 (v=4,o=1)  
M2_G3 In3 in sa1 (v=10,o=1)  
M2_G4 M2_G2 in sa0 (v=6,o=1)  
M2_G4 M2_G3 in sa0 (v=11,o=1)  
M3_G2 M1 in sa1 (v=1,o=1)  
M3_G2 M3_SN in sa1 (v=12,o=1)  
M3_G3 S1 in sa1 (v=3,o=1)  
M3_G3 M2 in sa1 (v=4,o=1)  
M3_G4 M3_G2 in sa0 (v=2,o=1)  
M3_G4 M3_G3 in sa0 (v=6,o=1)  
M3_G4 Out out sa0 (v=2,o=1)  
M3_G4 Out out sa1 (v=1,o=1)

**mux.udt file contents:**
# faults not detected for 'MUX4';
# gate net I/O s-a-f;
M1_G2 M1_SN in sa1  
M1_G3 S0 in sa1  
M2_G3 S0 in sa1

**Undetected fault list is same format as input fault list (for simulation)**

- Vector # where fault was first detected
- Output # where fault was first detected
AUSIM Example

**mux.cnt file:**
default mux
lib slop.lib
proc
keepfas
audit
nets
fltgen
simul8
pftsim
fltpro

**For use in plotting fault coverage results**

**mux.pro file contents:**
mux.cnt
fault detection distribution
vecnum numflts cumm
1 3 3
2 3 6
3 2 8
4 2 10
6 2 12
7 1 13
9 1 14
10 1 15
11 1 16
12 2 18

3/7/07 VLSI Design & Test Seminar
Fault Simulation Control

- Specific commands for fault simulation
  - `notrip` – sets flag for fault simulation to run through entire vector set
    - Default: simulation stops (trips) at first vector to detect fault
      - Only valid for serial fault simulations
  - `ndetect` – similar to `notrip`
  - `fltsim` – serial gate-level stuck-at fault simulation
  - `pftsim` – parallel gate-level stuck-at fault simulation
  - `bftsim` – serial bridging fault simulation
  - `pbfsim` – parallel bridging fault simulation
Another AUSIM Example

**mux.cnt file:**

- default mux
- lib slop.lib
- proc
- bftgen dom
- notrip
- bftsim

AUSIM version 2.6

Begin ASL syntax check . . . Syntax appears to be OK!

ASL file processing complete

processing time = 40

24 bridging faults (type=dom) generated for 'MUX4' & written to mux.flt

fault generation time = 0

Begin serial bridging fault simulation for 'MUX4' notrip=1, fltsim=2 . . .

AUSIM fault simulation for circuit 'MUX4' complete

24 faults were simulated

23 detected faults written to 'mux.det'

1 undetected faults written to 'mux.udt'

1 faults were potentially detected and written to 'mux.pdt'

serial bridging fault simulation time = 220

*Note: uses simulation results from previous AUSIM example*
Another AUSIM Example

mux.cnt file:
defaultmux
libslop.lib
proc
bftgen dom
notrip
bftsim

mux.flt file contents:
# bridging faults (type=dom) generated for 'MUX4';
# purp-net fault-type victim-net ;
In3 dom M1
M1 dom M1_G2
M1_G2 dom M1
M1_G2 dom M1_G3
M1_G3 dom M1_G2
M1_G3 dom M1_SN
M1_SN dom M1_G3
M1_SN dom M2
M2 dom M1_SN
M2_G2 dom M2
M2_G2 dom M2_G3
M2_G3 dom M2
M2_G3 dom M2_G2
M2_G2 dom M2
M2_G2 dom M2_G3
M2_G3 dom M2_G2
M2_G3 dom M2_G2

Note: an internal net cannot dominate a primary input
(an AUSIM restriction) but you can add a buffer

Recall
mux.net:
In0
In1
In2
In3
M1
M1_G2
M1_G3
M1_SN
M2
M2_G2
M2_G3
M2_SN
M3_G2
M3_G3
M3_SN
Out
S0
Another AUSIM Example

**mux.cnt file:**
default mux lib slop.lib proc bftgen dom notrip bftsim

**mux.det file contents:**
# faults detected for 'MUX4';
# purp-net bridge-fault victim-net (vec,out);
In3 dom M1 =(v=2,o=1)(v=3,o=1) M1_G2 dom M1 =(v=9,o=1) M1_G2 dom M1_G3 =(v=9,o=1) M1_G3 dom M1_G2 =(v=2,o=1) M1_G3 dom M1_SN =(v=2,o=1) M1_SN dom M1_G3 =(v=1,o=1)(v=3,o=1)(v=9,o=1) M1_SN dom M2 =(v=4,o=1)(v=5,o=1)(v=11,o=1) ...
M2_SN dom M2_G3 =(v=4,o=1)(v=5,o=1)(v=11,o=1) M2_SN dom M3_G2 =(v=1,o=1)(v=3,o=1)(v=4,o=1)(v=5,o=1)(v=9,o=1) M3_G2 dom M2_SN =(v=6,o=1) M3_G2 dom M3_G3 =(v=6,o=1)(v=11,o=1) M3_G3 dom M3_G2 =(v=2,o=1)(v=9,o=1) M3_G3 dom M3_SN =(v=2,o=1)(v=9,o=1) M3_SN dom M3_G3 =(v=1,o=1)(v=3,o=1)(v=6,o=1)(v=7,o=1)...(v=11,o=1) M3_SN dom Out =(v=1,o=1)(v=3,o=1)(v=6,o=1)(v=7,o=1)...(v=11,o=1) Out dom M3_SN =(v=2,o=1)(v=9,o=1)(v=12,o=1) S0 dom Out =(v=2,o=1)(v=6,o=1)(v=7,o=1)(v=8,o=1)(v=10,o=1)(v=12,o=1)

every vector & output detecting fault is reported

**mux.pdt file contents:**
# faults potentially detected for 'MUX4';
# purp-net fault victim-net (#_potential_dets);
M1 dom M1_G2 (5) # times fault was potentially detected
Another AUSIM Example

mux.cnt file:
default mux
lib slop.lib
proc
bftgen dom
ndetect
bftsim

mux.det file contents:
# faults detected for 'MUX4' ;
# purp-net bridge-fault victim-net list of vec #s detecting fault ;
In3 dom M1 = 2 3
M1_G2 dom M1 = 9
M1_G2 dom M1_G3 = 9
M1_G3 dom M1_G2 = 2
M1_G3 dom M1_SN = 2
M1_SN dom M1_G3 = 1 3 9
M1_SN dom M2 = 4 5 11
...
M2_SN dom M2_G3 = 4 5 11
M2_SN dom M3_G2 = 1 3 4 5 9
M3_G2 dom M2_SN = 6
M3_G2 dom M3_G3 = 6 11
M3_G3 dom M3_G2 = 2 9
M3_G3 dom M3_SN = 2 9
M3_SN dom M3_G3 = 1 3 6 7 8 11
M3_SN dom Out = 1 3 6 7 8 11
Out dom M3_SN = 2 9 12
S0 dom Out = 2 6 7 8 10 12

Only every vector numbers detecting fault is reported

- Potentially detected format same as before
- Potentially detected faults also written to undetect fault list for simulation
- Oscillation fault list format similar to potential detect
Scanless Scan Testing

- Scan files can be included in vector file
  - Scan chain order in .scn file
  - Vector in scan# file placed in DFFs
  - Contents in DFFs written to scan#o

- Can determine scan design fault coverage without implementing scan design circuitry
  - Also good for initialization and reading contents of flip-flops

- No serial shifting of scan vectors
  - Very fast fault simulation times

- Problem: one scan file per vector
  - Needs to be improved

Example:
```
example.vec

01111
11111
00011
10011
00101
10101
```
Summary and Conclusions

- AUSIM available at http://www.eng.auburn.edu/~strouce/ausim.html
  - For other options type: ausim help
  - Fault simulation version used in testing classes and research lab
  - Logic simulation version used in ELEC 2200

Future development
- Fault simulation GUI for windows
  - Instead of using command prompt
- Revive multiple fault simulation
- Will consider requests for new features
  - Several have already been incorporated