ASL: Auburn Simulation Language and AUSIM: Auburn University SIMulator

Chuck Stroud
Professor
Electrical & Computer Engineering
Auburn University
Overview of ASL

ASL: Auburn Simulation Language
- A positional notation hardware description language for digital logic
- Used with AUSIM (Auburn University SIMulator) for
  - Logic simulation
    - Design verification
  - Fault simulation
    - Gate-level stuck-at faults
    - Bridging faults (shorts between signals)
Delimiters

- Used to separate all:
  - Keywords
  - Some reserved characters
    - Specifically ‘#’ and ‘;’
  - Names
    - Gate names
    - Signal (net) names

- Includes:
  - Space, tab, and new line

- Can be used freely
  - Missing delimiter is a frequent source of syntax errors


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 statement
  - ‘:’ (colon)
    - Denotes keyword
  - ‘[’ and ‘]’
    - Used for bus notation
      - Supported only in advanced versions of AUSIM
Keywords

- Can be all capital or all small letters
  - Always end with colon ‘:’

- Include:
  - CKT: (ckt:) and SUBCKT: (subckt:)
    - Denotes circuit and subcircuit statements
  - IN: (in:) and OUT: (out:)
    - Denotes inputs and outputs of circuit, subcircuit, or gate
  - CON: (con:)
    - Denotes configuration bits for FPGAs/CPLDs
      - Supported only in advanced versions of AUSIM
  - AND: (and:), OR: (or:), NAND: (nand:), NOR: (nor:), &
    - NOT: (not:)
      - Denotes elementary logic gate component statements
  - User defined subcircuit names
More Keywords

- Functional models
  - Supported only in advanced versions of AUSIM
- Include:
  - XOR: (xor:) and NXOR: (nxor:)
    - Denotes exclusive-OR and exclusive-NOR gates
  - 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
Statements

- Three types include:
  - Comment statements
    - Include textual information or remove portions of circuit without 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:)
  - Component statements
    - Defines unique name and I/O connections of a component (gate or subcircuit)
Comment Statements

- Format:
  - Begin with ‘#’ followed by delimiter
  - Next comes text of comment
  - End with a delimiter followed by ‘;’
    - Example of valid comment
      
        # this is a valid comment ;

- Comments can be multiple lines
  - Leaving off the ‘;’ or the delimiter before the ‘;’ at the end of a comment is a common mistake
Circuit statements

- **Format:**
  - CKT: cktnname IN: input name(s) OUT: output name(s) ;
  - SUBCKT: subcktnname IN: input name(s) OUT: output name(s) ;
  - Format with configuration bits (supported only in advanced versions of AUSIM):
    - CKT: cktnname IN: input name(s) CON: configuration bit name(s) OUT: output name(s) ;
    - SUBCKT: subcktnname IN: input name(s) CON: configuration bit name(s) OUT: output name(s) ;

- **Circuit statements can span multiple lines**
  - Good for large number of inputs, outputs, or config bits
Component Statements

Format for gates:
- **GATE**: name IN: input name(s) OUT: output name(s) ;
  - where **GATE**: can be:
    ✓ AND, OR, NAND, NOR, or NOT in basic versions of AUSIM
    ✓ XOR, NXOR, MUX2, DFF, NDFF, LAT, or NLAT in advanced versions of AUSIM

Format for subcircuits
- **SUBCKTNAME**: name IN: input name(s) OUT: output name(s) ;
  - where **SUBCKTNAME** is name defined in subckt: statement
- **SUBCKTNAME** with configuration bits:
  - **SUBCKTNAME**: name IN: input name(s) CON: configuration bit name(s) OUT: output name(s) ;
Component Statements

- **AND:, OR:, NAND:, NOR:, XOR:, & NXOR:**
  - Any number of inputs (order does not matter) & only one output

- **NOT:**
  - Only one input & one output

- **MUX2:**
  - Input ordering: In0 In1 Sel

- **Flip-flop and latch:**
  - **DFF: & NDF:** input ordering: Clock Data
  - **LAT: & NLAT:** input ordering: Enable Data
  - Output ordering: Q Qbar
    - Qbar is optional for both flip-flop and latch
  - **No set/reset (preset/clear) in functional model**
Naming Conventions

- Need unique names for:
  - Gates
  - Signals (nets)
  - Circuits and subcircuits

- Any combination of characters & numbers
  - Except for reserved characters & keywords

- Case sensitive
  - Example: X1 and x1 are two different names

- Signal name can be same as gate name
  - Common practice by designers to:
    - Reduce number of names
    - Aid in debugging
Bus Notation

- Saves time and reduces design errors
  - Supported in hierarchical ASL
  - Only in advanced versions of AUSIM
  - Must be consecutive numbering
  - Can be anywhere in name
  - Only one bus notation per name

- Examples:
  - DIN[7:4] produces
    - DIN7  DIN6  DIN5  DIN4
    - D1  D2  D3  D10  D9  D8  D7  D6  D5
  - [19:25]junk produces
    - 19junk  20junk  21junk  22junk  23junk  24junk  25junk
  - Mr[5:0]Dude produces
    - Mr5Dude  Mr4Dude  Mr3Dude  Mr2Dude  Mr1Dude  Mr0Dude
Example of Digital System Design

- The integrated circuit design process
  - Note all the simulation
    - Ensure the design works and assists in debugging design errors (design verification)
    - Detect defects in manufacturing process or faults in system (testing)
  - To simulate a circuit, we must describe it in a manner that can be interpreted and understood by the simulator
Design Capture with HDLs

All of these captured design descriptions can go to simulation:

- Netlist
  - Connections of components made via signal name
    - Example: ASL

- Schematic
  - Connections explicit (via wires) or via signal name
  - Produces a netlist for simulation

- Higher level language (VHDL or Verilog)
  - Synthesis to gate level netlist
Nonhierarchical ASL Example

- **Circuit statement:**
  - Circuit name
  - Inputs
  - Outputs

- **Component statements:**
  - Component type
  - Component instantiation
  - Inputs signals
  - Output signals

```asl
# ASL description of MUX ;
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 ;
```
Hierarchical Design

- Saves time and reduces design errors
  - Once a subcircuit has been simulated & verified
- Most HDLs support hierarchical design
Hierarchical ASL Example

- Hierarchy (subcircuits)
- Connection via signal names (or netnames)
  - Keyword notation
  - Positional notation
    - Used by ASL
- Bus notation

```
# 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 ;
```
Design Verification

**Schematic (or logic diagram)**

![Logic Diagram]

**Netlist**

```plaintext
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**

- Netlist
- Stimuli

Simulation Results
Design Verification

- Logic simulation results used to:
  - Verify proper operation of design
  - Find & fix problems (errors) in design

  ➤ aka: debugging

<table>
<thead>
<tr>
<th>Input Stimuli (or vectors)</th>
<th>Simulation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td># ABS ;</td>
<td># ABS Z ;</td>
</tr>
<tr>
<td>000</td>
<td>000 0</td>
</tr>
<tr>
<td>001</td>
<td>001 0</td>
</tr>
<tr>
<td>010</td>
<td>010 0</td>
</tr>
<tr>
<td>011</td>
<td>011 1</td>
</tr>
<tr>
<td>100</td>
<td>100 1</td>
</tr>
<tr>
<td>101</td>
<td>101 0</td>
</tr>
<tr>
<td>110</td>
<td>110 1</td>
</tr>
<tr>
<td>111</td>
<td>111 1</td>
</tr>
</tbody>
</table>

Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

compare simulation results to truth table
Simulation

Design Capture Input

Schematic diagram

\[ \text{ckt: MUX in: A B S out: Z ;} \]
\[ \text{not: G1 in: S out: SN ;} \]
\[ \text{and: G2 in: A SN out: G2 ;} \]
\[ \text{and: G3 in: S B out: G3 ;} \]
\[ \text{or: G4 in: G2 G3 out: Z ;} \]

Computer Emulation

Pointers to source logic values

Data structures hold logic values

Logical operations performed by computer
Types of Simulators

- Compiled Simulator (AUSIM)
  - Simulation continues until circuit is stable
    - No changing logic values within circuit
  - aka: unit delay or logic simulator
    - All gates in circuit have a finite unit delay
  - Good for initial design verification
    - Short simulation times

- Event-Driven Simulator
  - Simulation events scheduled in time
    - Circuit may not be stable when input changes
  - aka: timing simulator
    - Gates have real delays base on intrinsic & extrinsic factors
  - More accurate for real circuits
    - Longer simulation times and more computer intensive
Audits for Common Problems

- Usually part of CAD tool
  - Schematic capture
  - Simulator (AUSIM)

- AUSIM audits for potential design errors
  - Unconnected gate inputs
  - Multiple gates driving same net
    - AUSIM does not support tri-state or bi-directional busses
  - Subcircuit statement and instantiation have different number of inputs and/or outputs
    - Configuration bits in advanced versions of AUSIM
Other AUSIM Audits

- Duplicate names for:
  - Gates, inputs, outputs, and configuration 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 or component name
- Some rare errors
  - Incorrect number of inputs to inverter, flip-flop, latch or multiplexer
  - Configuration bits for gates (should not be any)
    - includes all elementary logic gates and functional models
  - Net names too long
    - Before hierarchical flattening (so not too useful)
    - Current length is 75 characters
AUSIM 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.
  - 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`.
  - Be careful when using PC editors to create files and then running AUSIM on UNIX.
    - To remove DOS control characters type the following command line (example for ASL only):
      ```
dos2unix name.asl name.asl
      ```
    - Use same command with appropriate file suffix for other input files as needed.

- AUSIM checks for missing input files.
AUSIM Input Files (default names)

- Both logic and fault simulation need
  - ASL (name.asl)
  - Library file (name.lib) – ASL file containing subcircuit descriptions
    - Workstation version of AUSIM needs for the .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 bit inputs

- Logic simulation only
  - Input vector file (name.vec)
  - Optional: internal node file (name.nod) to monitor internal nodes

- 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 AUSIM (fltgen or bftgen commands)
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

- 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*)
    - Includes total number of vectors for which fault is potentially detected
    - These faults also written to *udt* file since they are only potentially detected
  - Oscillation fault list (*name.osc*)
    - For serial fault simulation only
    - Contains faults that produce oscillations in circuit
      - Recall that AUSIM is a compiled simulator
  - Fault detection profile (*name.pro*)
    - Give number of faults detected for vector set on a per vector and cumulative basis – produced by *fltpro* command
AUSIM Fault List File

- Fault simulation requires a list of faults to be emulated
  - 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*)
    - Includes total number of vectors for which fault is potentially detected
    - These faults also written to *udt* file since they are only potentially detected
  - Oscillation fault list (*name.osc*)
    - For serial fault simulation only
    - Contains faults that produce oscillations in circuit
      - Recall that AUSIM is a compiled simulator
  - Fault detection profile (*name.pro*)
    - Give number of faults detected for vector set on a per vector and cumulative basis – produced by *fltpro* command
Control File

- Specifies file names
  - Default names can be specified with default name
    - Where name is common prefix for all files
  - Default names overridden for any file with suffix new_file_name
    - Where suffix is the default for a given file type
    - Example: out slop.txt
      - Specifies simulation results to be written to slop.txt instead of name.out
Control File

- Specifies commands for processing
  - proc – processes files, loads data structures, and runs audits
    - Initiated after file names are specified and before simulation and other processing commands
  - keepfas – writes flattened ASL from data structures to name.fas file
  - audit – collects circuit statistics and writes to name.aud
  - nets – writes list of signals (nets) into name.aud in order that they appear in data structures
    - Determines order of bridging faults generated by bftgen command
  - simul8 – runs logic simulation and write results in name.out
Control File

- Specifies commands for fault simulation

  - `notrip` – sets flag for fault simulation to run through entire vector set
    - Otherwise, simulation stops (trips) at first vector to detect fault
      - Only valid for serial fault simulations
  
  - `fltsim` – runs serial gate-level stuck-at fault simulation
  
  - `pftsim` – runs parallel gate-level stuck-at fault simulation
  
  - `bftsim` – runs serial bridging fault simulation
  
  - `pbfsim` – runs parallel bridging fault simulation
  
  - `fltpro` – generate fault detection profile in `name.pro`
Control File

- Specifies commands for 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
      - There is no fault collapsing for bridging faults
Logic Simulation Example

Prompt> ~strouce/bin/ausim mux.cnt
AUSIM version 2.1
Begin ASL syntax check . . .
   Syntax appears to be OK!
4 SUBCKTs 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
ASL file processing complete
Begin Audit file generation . . .
   results written to 'mux.aud'
flattened ASL for ckt 'MUX4' written to 'mux.fas'
Begin logic simulation for circuit 'MUX4' notrip=0, fltsim=0 . . .
   AUSIM logic simulation for circuit 'MUX4' complete for 20 vectors
   logic simulation results written to 'mux.out'
Prompt>