Using the Workstation version of AUSIM - Version 2.1

by Dr. Charles E. Stroud, Professor
Dept. of Electrical & Computer Engineering
Auburn University
March 12, 2004

ABSTRACT

This document gives a brief overview of the logic simulation capabilities of the workstation version of AUSIM. It is assumed that the reader is familiar with the ASL hardware description language [1] as well as the PC version of AUSIM [2].

INTRODUCTION

The workstation version of AUSIM is intended for the simulation of large circuits in terms of advanced fault simulation capabilities. But the workstation version of AUSIM has better syntax checking and some advanced logic simulation features when compared to the PC version of AUSIM [2]. For logic simulation, the workstation version supports bus notation and functional models. The functional models in this version of AUSIM include exclusive-OR and exclusive-NOR gates, active high and low level sensitive latches, rising and falling edge triggered D flip-flops, and asynchronous Random Access Memory. Instead of being menu driven as is the case in the PC version of AUSIM, the workstation version is command file driven.

EXECUTING AUSIM

The execute the workstation version of AUSIM (which will be referred to as simply AUSIM from this point on), the user simply types:

```
~strouce/bin/ausim cont_file_name
```

where the *control_file_name* is the name of the control file containing the commands to AUSIM for directing the desired simulation. This assumes that the user's path is set to the directory containing the AUSIM executable. If no arguments are given on command line (for example, the user only type 'AUSIM' and return), the command line format is reported as follows:

```
AUSIM version 2.1
command line format: ausim cont_file
type 'help' instead of a cont_file name for format of control file
```

As indicated, if the word 'help' is typed instead of a control file name, then AUSIM returns the following listing of the control file format:

```
format for control file:

file names:
default file_prefix - uses default suffixes listed below
asl asl_file - input ASL circuit description file
fas fas_file - flattened ASL circuit description from AUSIM
lib lib_file - input ASL library file (for no library use empty file)
aud aud_file - audit file produced by AUSIM with 'audit' command
vec vec_file - input vector file used for 'simul8' command
con con_file - configuration bit file for CPLD/FPGA simulations
scn scn_file - scan chain ordering for scan test simulations
out out_file - simulation output file by AUSIM with 'simul8' command
flt flt_file - input fault list OR fault list from AUSIM with 'fltgen' command
det det_file - detected fault file by AUSIM with 'fltsim' or 'pftsim' command
udt udt_file - undetected fault file by AUSIM with 'fltsim' or 'pftsim'
pdt pdt_file - potentially detected fault by AUSIM with 'fltsim' or 'pftsim'
nod nod_file - internal nodes to be monitored during 'simul8' command
net net_file - netname file generated by nets command
```

```
osc osc_file - oscillation fault file produced by 'fltsim' command
pro pro_file - fault detection profile produced by 'fltpro' command

commands:
proc    - process kslfile and flatten if necessary
keepfas - tells AUSIM not to delete flattened ASL file
audit   - area and performance analysis
nets    - generates list of flattened net names in order of data structure
iniramX - initialize RAMs, X=0,1,e,o where
      '0' initializes to all 0s,
      '1' to all 1s,
      'e' initializes even addresses to 1s, and
      'o' initializes odd addresses to 1s
simul8  - logic simulation
uncol   - uncollapsed faults to be generated\n");
fltgen  - generate fault list (coll or uncoll)\n");
bftgen param - generate bridging fault list (coll or uncoll) where param =
      'dor' for dominant OR (or diode OR)
      'dand' for dominant AND (or diode AND)
      'dom' for the dominant bridging fault model
notrip  - don't end simulation of a given fault when detected
fltsim  - serial fault simulation
pftsim  - parallel fault simulation
fltpro  - fault profile of detected fault list
bftsim  - serial bridging fault simulation
pbfsim  - parallel bridging fault simulation
komand  - input your commands interactively - for those slow-to-load ckts
```

THE CONTROL FILE

The control file basically gives an ordered list of commands similar to those that would be manually executed by the user in the PC version of AUSIM [2]. To begin the files must be specified. If the proper naming convention is used then the user can simply enter one line (let's assume that we are simulating the full adder example from [1] and [2], *fadd*) as follows:

```
default fadd
```

This assumes the the required imput files are named *fadd.ASL*, *fadd.vec*, and *fadd.lib* (where these three files are the minimum that are required. Note that if no library file is being used then an empty file or a file with a syntactically correct comment line in it will do. If any none default file name is used (or desired in the case of output files) then additional lines are used to specify each file name. For example, let's assume that we don't want a bunch of empty files (or files with a comment line) in our directory for the library files – then we can specify one such dummy library file (let's assume its name is slop.lib) as follows:

```
default fadd
lib slop.lib
```

After specification of the desired file names, we are ready to enter commands. Before any simulation can begin, we must process the file with the *proc* command. This command indicates that all file names have been specified and processing is to begin. The processing of the files begins with syntax checks of the library file and the ASL file as well as a check for subckt: statements to initiate flattening of the hierarchy. After hierarchical flattening is complete, the data structures are loaded and a number of audits and circuit checks are performed for items such as nets with multiple driving sources, nets with no driving sources, etc. The failure of any of these check will halt the simulation with messages to the user that will hopefully help in fixing the problem in the ASL description.

During falttening process, bus notation (which will be discussed in the next section) is expanded. This means that in order to use bus notation, the user must also use hierarchy in the ASL description.

The reason for this is that bus notation expansion is built into the hierarchy flattener which is only initialed when subckt: statements are encountered in the ASL.

After the *proc* command, the user can begin simulation. However, during the early stages of design verification, additional information may be helpful in debugging the circuit. This additional information can be found in the flattened ASL file (*fadd.fas* in our example), the audit file (*fadd.aud* in our example), and a file that contains a list of all the net names in the circuit as they appear in the data structures (the net file or *fadd.net* in our example). In order to get either of these files, commands for their generation must be included after the *proc* command. These commands include *keepfas* (to keep the flattened ASL file that would otherwise be deleted), *audit* (to generate the audit file), and *nets* (to generate the netlist file). The formats for the *.fas* and *.aud* files are similar to that described in the manual for the PC version of AUSIM [2] and will not be described here.

For logic simulation, the *simul8* command is needed to initiate the application of the vector file (*fadd.vec* in our example) to the circuit loaded into the data structures. Remember to avoid those nasty clock-data races in your *.vec* file!!! This produces the simulation output results file (*fadd.out* in our example) which has a similar format to the *.out* file described in [2]. The rest of the commands are specific to fault simulation and are beyond the scope of this section. Therefore a complete control file for logic simulation could be as follows (depending on the desired actions by the user):

```
default fadd
lib slop.lib
proc
audit
simul8
```

BUS NOTATION

The bus notation is supported by AUSIM for hierarchical ASL descriptions. For large busses, it is often error prone to type in the long list of names and this results in a likely source of design errors. Therefore, bus notation is often supported to improve productivity as well as to reduce design errors. Bus notation assumes consecutive numbering associated with the same basic net name. For example, a 7-bit data bus with root name DIN could be denoted at DIN[7:0] instead of specifying each individual net name as follows:

DIN7 DIN6 DIN5 DIN4 DIN3 DIN2 DIN1 DIN0

Bus notation can begin at any number and end at any number (either ascending or descending order) but assumes that the numbering is consecutive within the range. To skip one or more net names in the sequence then two bus notation based net names would be required. In addition, the bus notation can appear anywhere within the net name. However, there can be only one bus notation within a net name. Also bus notation assumes decimal numbers (not alphabetical characters). Finally, bus notation can only be used with net names; it cannot be used to instantiate multiple components. For example, the following instances of bus notation are all valid with in AUSIM:

DIN[7:4]	produces	DIN7 DIN6 DIN5 DIN4
D[1:3] D[10:5]	produces	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

FUNCTIONAL MODELS

The workstation version of AUSIM also supports some functional models for both logic and fault simulation for the following circuits:

XOR (xor)	Exclusive-OR gate with multiple inputs
NXOR (nxor)	Exclusive-NOR gate with multiple inputs
MUX2(mux2)	2-to-1 multiplexer (input ordering: IN0 IN1 SEL)
LAT (lat)	Active high level sensitive D latch (input ordering: EN D)
NLAT (nlat)	Active low level sensitive D latch (input ordering: EN D)

DFF (dff)	Rising edge triggered D flip-flop (input ordering: CK D)
NDFF (ndff)	Falling edge triggered D flip-flop (input ordering: CK D)
RAM (ram)	Asynchronous Random Access Memory with 64 words of 1 bit each (input ordering: WE D AD5 AD4 AD3 AD2 AD1 AD0)

The latch and flip-flop functional models can be specified with either one output (Q) or two outputs (Q and Qbar, in that order). To avoid using these functional models, the user should not used these names for subckt names in either all uppercase or all lowercase (otherwise AUSIM is override the subckt call with the functional model). This can easily be done by adding additional characters to the subckt name.

It should be noted that gate level fault generation and simulation with these functional models is only considered at the inputs and outputs of the functional circuit (for example, the CK input to the DFF can be stuck-at-0 or stuck-at-1).

FAULT SIMULATION COMMANDS

The *fltgen* and *bftgen* commands generate gate-level stuck-at and bridging fault lists, respectively, and writes the list to the *.flt* file. Normally the *fltgen* command produces a collapsed gate-level stuck-at fault list but the *uncol* command preceding the *fltgen* command will result in the generation of an uncollapsed fault list. The bftgen command generates a set of bridging faults between all consecutive pairs of nets based on the lexiconical ordering of the nets in the circuit. The type of bridging fault list generated can be the dominant, dominant-AND, or dominant-OR bridging fault model as defined by the parameter (*dom*, *dand*, or *dor*, respectively) immediately following the *bftgen* command.

The *notrip* command is used to continue fault simulation of a fault following its initial detection. In this case the *.det* file information includes all vectors and primary outputs of the circuit for which the fault was detected. The notrip command works only with serial fault simulation and, as a result, must preceded the actual serial fault simulation command. Both serial and parallel fault simulation are supported for both gate-level stuck-at and bridging faults. The serial and parallel gate-level stuck-at fault simulation commands are *fltsim* and *pftsim*, respectively. The serial and parallel bridging fault simulation commands are *bftsim* and *pbfsim*, respectively. Note that only one type of fault simulation command should be included in a control file. The fltpro command produces a .pro output files which gives a profile of the fault detection associated with a given set of test vectors for the circuit being simulated in terms of the number of faults detected by a given vector along the the cumulative number of faults detected at that point in the set of test vectors.

FPGA/CPLD CONFIGURATION BITS

AUSIM supports configuration bit inputs in the ASL description for FPGA and CPLD logic and fault simulations. Since the configuration memory bits are typically programmed prior to operation (simulation) of the FPGA or CPLD, AUSIM assumes that the configuration bits are contained in the *.con* file. In order to distinguish between system data inputs and configuration bits, the ASL description should include a list of “CON:” net names immediately following the “IN:” list and immediately preceding the “OUT:” list in “CKT:” and “SUBCKT:” statements. These configuration inputs are also considered in fault list generation and fault simulation for both gate-level stuck-at and bridging faults.

REFERENCES

[1] Charles E. Stroud, "ASL: Auburn University Simulation Language", Dept. of Electrical & Computer Engineering, Auburn University, July 7, 2003.

[2] Charles E. Stroud, "AUSIM: Auburn University SIMulator – Version 2.0", Dept. of Electrical & Computer Engineering, Auburn University, July 7, 2003.