PRESENTATION OF RESEARCH SPRING – FALL 2010

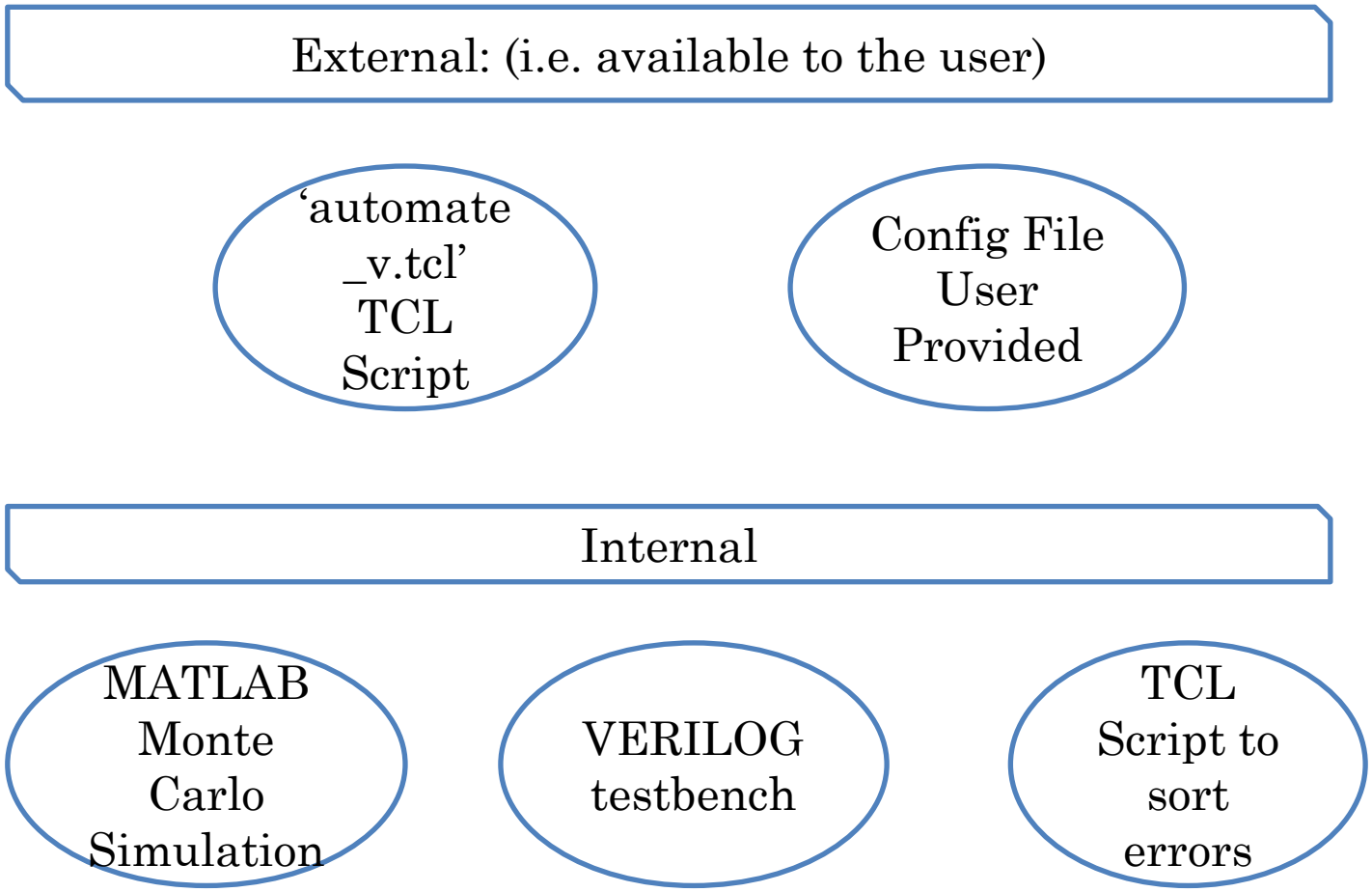
Dheeraj Srinivasan
dheerajsrini@ucla.edu

Advisor: Prof. Puneet Gupta
Electrical Engineering Department

Research Objectives

- Writing a testbench to test different types of modules in Verilog
- Designing a Monte-Carlo Simulation resulting in a uniform testing of different inputs
- Reading the output of NCSIM, and recording errors in '.csv' format
- Automating the above processes.

Basic Overview of Main Scripts

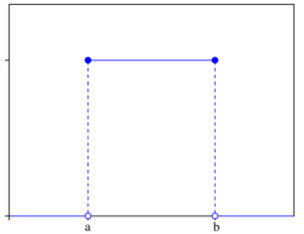


Config File

Variable name	Value	Description
regsize	4	Register size
sample_entries	10000	Number of entries
timedelay	1.5	Delay factor
a	0	beginning of test interval
b	15	end of test interval
lib	pointer to library	library
net	pointer to netlist	netlist
path	pointer to testbench	testbench
name_tb	array_test	name of testbench

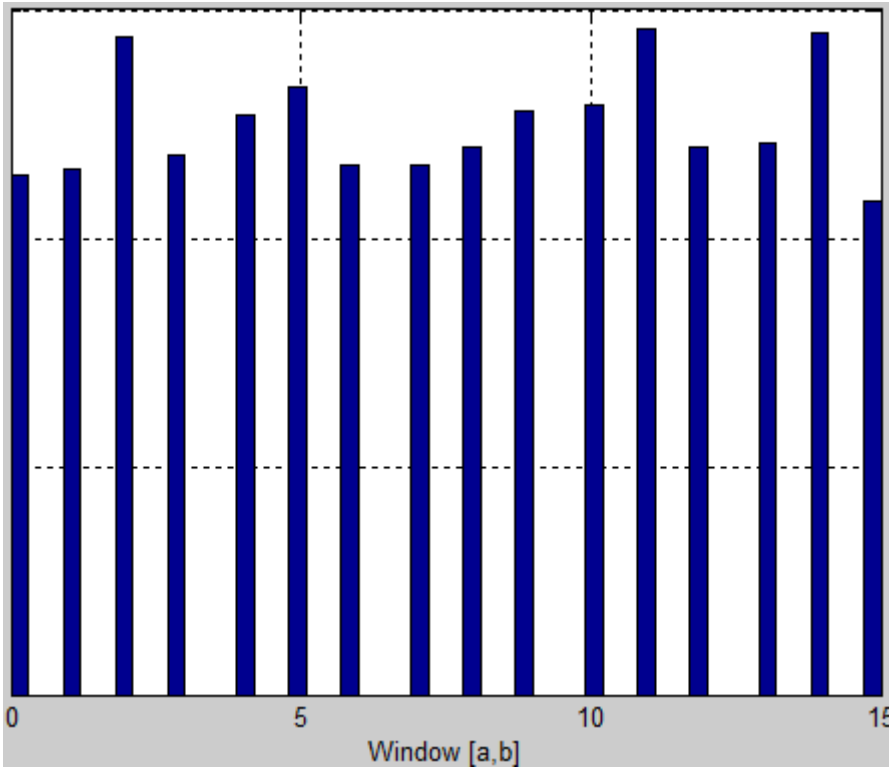
Monte Carlo Simulation (MATLAB)

- Motivation:
  - To save runtime and to get a uniform sampling of numbers(for inputs).
- MATLAB has a uniform distribution function:
  - randi([start point, end point], # samples, 1(integer))
  - User specifies the parameters
  - Can be adapted for x-by-x case
  - Accuracy improves as #samples increases
    - Ideal Uniform distribution: all values of the distribution are equally probable. The range is defined by the two parameters, a and b, which are its minimum and maximum values.



MC SIMULATION CONTINUED (2)

- Example: for the 4x4 case:
  - start point : 0; end point : 15; n = 1000 samples



MC Simulation continued (3)

- Verification of uniform distribution :
  - Using Expected Value:

$$E[X] = \frac{1}{2}(a+b) \rightarrow 7.5$$

  - In our case

$$E[X_1] = 7.350$$
$$E[X_2] = 7.630$$

VERILOG TESTBENCH

- Receives following inputs from the user (config) :
  - Register size (x-by-x)
  - Number of inputs (same as samples in Matlab)
  - The output file of the Monte Carlo Simulation
- Reads in the inputs from the MATLAB output file, and runs those values through the module.
- The results of the NCSIM simulation are stored in the ncsim.log file

TESTBENCH CODE

```
`timescale 1ns/1ps
module array_test;
`include "inc.v"
reg [regsize-1:0] a;  //creates register regsize-by-regsize
reg [regsize-1:0] b;
wire [sumsize_ab-1:0] out; //output wire
top TOP (a,b,out);
integer i,j,n,h;  //local variables
integer file,r,l;
integer m[sample_numentries-1:0],k[sample_numentries-1:0];
//array to store read in values
initial
begin
           file = $fopen("newfile.txt","r");
           r =2;
           n = 0;
           while(r == 2) //this loop reads in the inputs
           begin
                      r = $fscanf(file,"%d %d",i,j);
                      m[n]=i;
                      k[n]=j;
                      n = n+1;
           end
           h = 0;
           while(h !== sample_numentries) //pushes values into the module
           begin
                      a = m[h];
                      b = k[h];
                      h = h+1;
                      #timedelay;
                      $monitor("%d %d %d %d",a,b,out,$time); //writes to log file
           end
end
endmodule
```

TCL ERROR CHECKING

- No inputs needed from config file
- Reads in the results of the NCSIM Simulation
  - Ncsim.log file
- Uses a function to test whether the results produced by the simulation are correct
  - In this case the module is a multiplier
- Writes errors in ‘csv’ format to a file, that can be later analyzed using Excel etc.

TCL ERROR CHECKING CODE

```
set infile [open "ncsim.log" r]
#read in the results of NCSIM
set data [read $infile]
set data [split $data "\n"]
#split the data by newline
set slice [lrange $data 2 end-3]
#select part needed
set k 0 #local variable
set outfile [open "error_array.csv" w]
#set outfile to write
for {set m 0} {$m<[llength $slice]} {incr m} {
    scan [lindex $slice [expr $m]]  "%d %d %s  %d" i j out time
           #assigns values of array to variables
 proc mod {a b c} {
    if { $a*$b == $c } {return 1} else { return 0}
     #function to check the operation of the module (in this case multiplier)
     #just replace this function with same syntax a,b,c as arguments
     #for a different operation
}
}
set test [mod $i $j $out]
    if { $test == 1 }  {} else {
           puts $outfile "$i,$j,$out,$time \n"
           #call the function and write the errors to csv file
}
close $outfile
close $infile
```

TCL AUTOMATION SCRIPT

- Reads in the config file and stores it in an array.
- Supplies MC Simulation and testbench with the user-specified parameters needed
- Runs the MATLAB simulation in the background without opening it
- Runs the NCSIM simulation using the testbench
- Runs the TCL error checking code

TCL AUTOMATION CODE

```
set inc [open inc.v w]
#opens variable file for testbench

set mat [open matadd.m w]
#opens variable file for matlab

set text [open array.txt w]
#writes the file containing netlist

set infile [open "config.txt" r]
set data [read $infile]
set data [split $data "\n"]
set slice [split $data ","]
#after reading in from config file, assigns information read to certain
strings

set str "parameter regsize = [lindex $slice 3] ;\nparameter sumsize_ab =
2*[lindex $slice 3];
\nparameter sample_numentries = [lindex $slice 5];\nparameter timedelay =
[lindex $slice 7];"
```

TCL AUTOMATION CODE

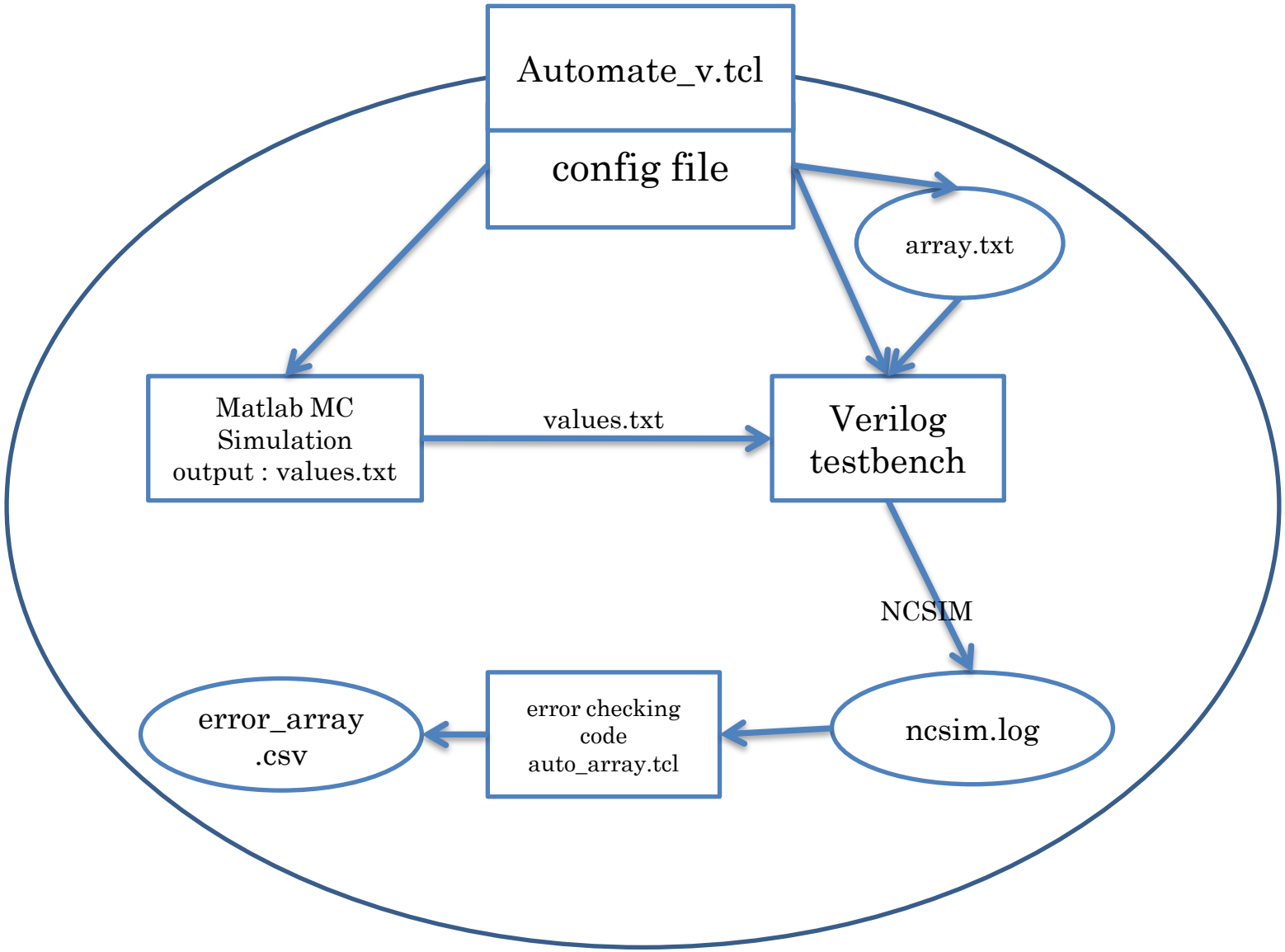
```
set stm "a = [lindex $slice 9];\nb= [lindex $slice 11];\nn= [lindex $slice 5];"
set stt "[lindex $slice 13]\n[lindex $slice 15]\n[lindex $slice 17]"

puts $text $stt
puts $inc $str
puts $mat $stm
#the three lines above write the strings to each type of files

close $infile
close $inc
close $mat
close $text

#executes MATLAB, NCSIM and error checking script
exec matlab -r monsim -nodesktop -nosplash
exec ncvlog -f array.txt
exec ncelab -acc +rwc array_test
exec ncsim array_test
exec tclsh auto_array.tcl
```

SUMMARY FLOWCHART



Summary

- The user only needs to have
  - Config.txt ; auto_array.tcl ; array_test.v and automate_v.tcl , and monsim.m (MC simulation)

  in the same directory, and run the automate_v.tcl file.
- Every other process is done in the background and the file that we are interested in is the error_array.csv file

Sample Simulation

- For the config file:

```
Variable name,value,description
regsize,4,register size

sample_entries,50,number of entries

timedelay,1.5,delay factor

a,0,beginning of testing interval

b,15,end of testing interval

lib,/w/ee.00/puneet/dheeraj/dheeraj/4_4/libs/NangateOpenCell
Library_typical_conditional.v,library

net,/w/ee.00/puneet/dheeraj/dheeraj/4_4/error_nl.v,netlist

path to testbench,
/w/ee.00/puneet/dheeraj/dheeraj/4_4/array_test.v
```

Sample Simulation (2) MATLAB result

- MATLAB results in the following values:

in1	in2
13	14
2	14
10	1
4	8
15	15
2	15
15	7
12	2
6	14
12	15
10	0
13	14
…	….
…	….
2	0

Sample Simulation (3) testbench

- NCSIM.LOG Output

in1	in2	out	time
13	14	X	0
13	14	X	0
13	14	150	1
2	14	150	2
2	14	44	2
2	14	28	3
10	1	28	3
10	1	26	3
10	1	10	4
4	8	40	5
4	8	32	5
.....			
.....			

Sample Simulation (4) testbench

- Error output

```
in1,in2,out,time
13,14,X,0
13,14,X,1
13,14,150,1
2,14,150,2
2,14,144,2
10,1,28,3
10,1,26,3
4,8,10,5
4,8,8,5
4,8,40,5
15,15,32,6
15,15,34,6
15,15,39,6
15,15,231,6
……..
```

How others in the lab can use this

- The code is easily adaptable to any different scenario with some minor changes
- For example of an alternate use is if you want to test an adder instead of the multiplier (assuming same 4x4 case) You would need to change:
  - Config:
    - The netlist
  - MC Simulation (unchanged)
  - VERILOG Testbench (unchanged)
  - TCL Error Checking File
    - Insert a function “proc mod” from testing (a·b) to testing (a+b)
  - TCL Automate File (unchanged)

How others in the lab can use this (2)

- There are many other applications for which this package of code can be used.
- This setup can be used to characterize any design against its accurate version, while keeping the abstraction at a purely software level.
- You do not have to manually run Matlab, then NCSIM, and so on, rather you could go from the netlist to the error result file with few changes (like in the example)