HillSIM: A Hierarchical Bench Logic Simulator for Zero Delay Combinational Circuits

HILLARY GRIMES III, GRADUATE STUDENT, *AUBURN UNIVERSITY*

Abstract

This paper presents an overview of HillSIM (Hill-ary's SIM-ulator), which is a logic simulator and verification program for zero-delay combinational circuits. HillSIM requires two input text files: a circuit description in a hierarchial bench format and a simulation input file containing input vectors to be simulated and the expected output responses to compare with simulated reponses. HillSIM provides a means of verifying a circuit's logic behavior for a set of input stimuli and expected responses.

1. Introduction

Logic simulation is used for design verification during the electronic design process of digital circuits (figure 1) [2]. The design process begins with specification, which is a description of the design's behavior and characteristics, including input stimuli and expected responses. Synthesis produces the netlist, which is an interconnection of electronic components that meets the specification. Logic simulation is used to simulate the circuit for the input stimuli provided from specification. By comparing the simulated responses with the expected responses provided by specification, the design is verified. If errors are found from verification, then the netlist is modified, and verification begins again [2].

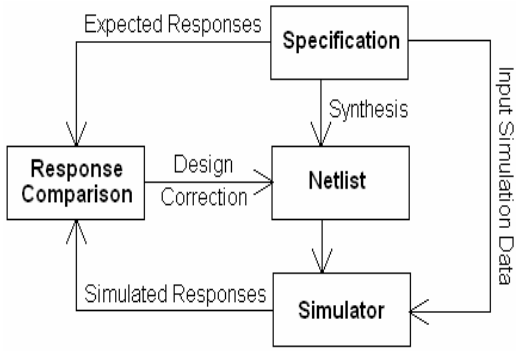

Figure 1: Basic design process

This paper will present a description of HillSIM (Hill-*ary's* SIM-*ulator*), which is a logic simulator and verification program for zero-delay combinational circuits with boolean gates using two states (0 and 1). HillSIM accepts a hierarchical bench netlist input for the circuit description, and verifies the circuit's logic behavior with a list of input vectors and expected output responses. Section 2 of this paper will discuss user information for HillSIM, including a description of the hierarchical bench netlist circuit description. Section 3 will describe data structures and algorithms used in implementing HillSIM, and section 5 will discuss the performance of HillSIM on the ISCAS 85 benchmark circuits and a four bit adder circuit. Section 6 will conclude the paper.

2. User Information

HillSIM prompts for two required input text files when executed to perform circuit simulation and verification. The first input file contains the circuit description in a hierarchical bench format, and the second input file contains simulation data (inputs and expected outputs). During execution, HillSIM simulates the specified circuit for each input vector in the simulation input file. Circuit verification is done by comparing the outputs obtained via simulation with the expected outputs specified in the simulation input file. If simulation of every vector produces the proper outputs, then HillSIM's output is simply "Circuit Verified", and the execution time is printed to the screen. If circuit simulation fails for an input vector, then HillSIM's output specifies which vector failed, the primary outputs where errors are observed, and diagnosis information to help the user diagnose the problem.

2.1 Hierarchical Bench Input File

The first input file is the circuit description in a hierarchical bench format, in which each block is separated into its own section with a beginning and an ending

statement. Boolean gates supported include NANDs, ANDs, ORs, NORs, NOTs, and BUFFs. The block begins with "BLOCK *blockname*", and ends with an "END" statement. When the block is used within a larger block within the hierarchy, the output signals are listed first, then an "=", followed by the block name and input signals. The input signals and output signals are listed in the same order as they are declared inside the block. For example, we can define an XOR block, then a Half_Adder block using the previously defined XOR:

```
BLOCK  XOR
        INPUT(A)
        INPUT(B)
        OUTPUT(Y)
        X1=NAND(A,B)
        X2=NAND(X1,A)
        X3=NAND(X1,B)
        Y=NAND(X2,X3)
END

BLOCK Half_Adder
        INPUT(in_1)
        INPUT(in_2)
        OUTPUT(Sum)
        OUTPUT(Cout)
        Sum=XOR(in_1, in_2)
        Cout=AND(in_1, in_2)
END
```

In the statement "*Sum=XOR(A,B)*" inside the Half_Adder block, *Sum* corresponds to the XOR's "*Y*" output, and "*in_1*" and "*in_2*" correspond to the XOR's "*A*" and "*B*" inputs.

2.2 Simulation Input File

The second input file specifies a set of input vectors and expected output responses to be used during circuit verification. Each line of this file contains one input vector and one output response separated by a space or tab. For example, if we wanted to simulate the Half_Adder circuit shown above with an exhaustive set of inputs, our simulation input file would contain the following:

```
# Half_Adder Exhaustive Inputs and Expected
        Outputs:
        00      00
        01      10
        10      10
        11      11
```

Each input (first column) and output (second column) is specified in the same order they are declared inside the Half_Adder block declaration. The third line contains "10 10", so *in_1* = 1, *in_2* = 0, *Sum* = 1 and *Cout* = 0.

3. Implementation

HillSIM is implemented using the C programming language to run in windows under the cygwin environment. Execution begins by reading the hierarchical bench input circuit description and storing the circuit description in memory using the various data structures discussed in the next section. After the circuit is stored in memory, execution then reads the simulation input file line by line. For each line, the circuit is simulated and its outputs are verified. If verification fails, diagnosis information is printed.

3.1 Data Structures

In order to represent the circuit in memory, HillSIM uses the following structures for a signal, gate, and block:

```
struct SIGNAL{
        int             value;
        int             level;
        char            *name;
        struct SIGNAL   *ptrToInput;
        struct SIGNAL   *ptrToOutput;
        struct SIGNAL   *next;
};
struct GATE{
        int             level;
        int             faninCount;
        char            *name;
        char            *type;
        struct SIGNAL   *inputSignals;
        struct SIGNAL   *outputSignals;
        struct GATE     **faninArray;
};
struct BLOCK{
        int             gateCount;
        int             numLevels;
        char            *name;
        struct SIGNAL   *inputSignals;
        struct SIGNAL   *outputSignals;
        struct GATE     **gateArray;
};
```

Signals are stored in a linked list data structure, in which each each element contains a pointer to the next [1], illustrated in figure 2. Both gates and blocks are stored in a dynamically allocated

array of pointers. In a dynamically allocated array, memory is allocated when it is needed [1], so to add a block or gate, memory is allocated for a new pointer in the array, and also for the new block or gate structure during HillSIM's execution. Each block is contains its name, a list of its input signals, a list of its output signals, and an array of gates associated with that block. Each gate contains a unique name, gate type, gate level, a list of input signals, a list of output signals, and an array of pointers to fan in gates. Each signal contains a unique signal name, the signal's value, the signal's level, and a pointer to the next signal in the list. A signal also contains two extra pointers: ptrToInput, and ptrToOutput. If the signal is an input to a gate, and is also a primary input to the block, ptrToInput points to the associated signal in the block's input signal list. Similarly, ptrToOutput points to the appropriate signal in the block's output signal list if that signal is also a primary output of the block. If both ptrToInput and ptrToOutput are null pointers, the signal is an internal signal inside the block.

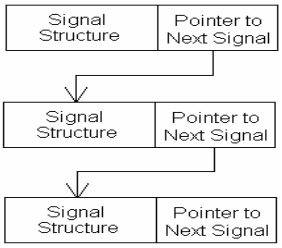

Figure 2: Signals Stored in a Linked List Data Structure.

After HillSIM reads the entire hierarchical bench input file, the block array contains one element for each defined block. When building each block's gate array, if a signal assignment is read in for a boolean gate (*X2=NAND(X1,A)*), a gate of the same type with a unique name is added to the array and its associated input and output signal lists are assigned to the new gate. If a signal assignment is read in for a previously defined block (*Sum=XOR(in_1, in_2))*, that block is found from previous elements in the block array, and its gates are inserted in the gate array with unique gate names, illustrated below:

XOR BLOCK Description:
Inputs: A, B
Output: Y
X1=NAND(A,B)
X2=NAND(X1,A)
X3=NAND(X1,B)
Y=NAND(X2,X3)

XOR Block's gate array contains:
g0: NAND — *Inputs: A, B* — *Output: X1*
g1: NAND — *Inputs: X1, A* — *Output: X2*
g2: NAND — *Inputs: X1, B* — *Output: X3*
g3: NAND — *Inputs: X2, X3* — *Output: Y*

The Statement "OUT = XOR(IN_1, IN_2)" inserts the following gates into the gate array:

XOR0.g0: NAND
Inputs: IN_1, IN_2
Output: XOR0.X1

XOR0.g1: NAND
Inputs: XOR0.X1, IN_1
Output: XOR0.X2

XOR0.g2: NAND
Inputs: XOR0.X1, IN_2
Output: XOR0.X3

XOR0.g3: NAND
Inputs: XOR0.X2, XOR0.X3
Output: OUT

By inserting the appropriate gates while building the block array, each block is stored in a "flattened" (non-hierarchical) manner. The last element in the block array corresponds to the highest level in the hierarchy, and contains the circuit to be simulated and verified using the simulation input file.

3.2 Verification Algorithm

HillSIM's verification algorithm begins by levelizing the circuit's gate array. Levelization begins by assigning the level zero to all primary input signals to the block, then each gate is visited in increasing order (starting with the first gate added). In this order, since all input signals of a gate are either primary inputs to the block or the outputs of previous gates in the gate array, all inputs to each visited gate already have their assigned level. If an input signal is not a primary input, then the previous gates in the gate array are searched for fan in gates to add to the gate's fan in array, and the level of the gate is set to the maximum level out of all input signals. The level of the gate's output signal is then assigned one more than the gate's level.

After the gate array is levelized, simulation can begin. The simulation input file is read in line by line, and for each line the circuit is simulated. Primary inputs are first assigned their appropriate values from the simulation input file, and then each gate is visited in increasing order. Using fan in gate information and primary input signal values, each gate's output signal is assigned the appropriate value given the type of gate and the values of its input signals.

After visiting all gates in the gate array, the output signals contain the simulated responses to be verified with the expected responses. If both responses match, then the circuit is verified for that vector, and simulation begins for the next vector from the simulation input file. If the responses do not match, then the failing input vector, the failing output signals, and diagnosis information is printed to help the designer locate mistakes in the netlist.

HillSIM's diagnosis begins at each failing primary output, and uses fan in information to visit each gate along the path to the failing output. Information is printed for each visited gate, which includes the gate's name, type, input signal values, and output signal values. This type of diagnosis has a very large diagnostic resolution, but does provide some information for the designer to use when correcting the netlist. The next section shows an example where diagnostic information is printed for a failing vector when verifying the c17 benchmark circuit.

4. Performance

4.1 Four Bit Adder

```
BLOCK XOR
        INPUT(A)
        INPUT(B)
        OUTPUT(Y)
        X1=NAND(A,B)
        X2=NAND(X1,A)
        X3=NAND(X1,B)
        Y=NAND(X2,X3)
END
BLOCK Half_Adder
        INPUT(A)
        INPUT(B)
        OUTPUT(Sum)
        OUTPUT(Cout)
        Sum=XOR(A,B)
        Cout=AND(A,B)
END
BLOCK Full_Adder
        INPUT(A)
        INPUT(B)
        INPUT(Cin)
        OUTPUT(Sum)
        OUTPUT(Cout)
        X1=XOR(A,Cin)
        X2=AND(A,Cin)
        X3=AND(X1,B)
        Sum=XOR(X1,B)
        Cout=OR(X3,X2)
END
BLOCK 4_Bit_Adder
        INPUT(A3)
        INPUT(A2)
        INPUT(A1)
        INPUT(A0)
        INPUT(B3)
        INPUT(B2)
        INPUT(B1)
        INPUT(B0)
        OUTPUT(Y3)
        OUTPUT(Y2)
        OUTPUT(Y1)
        OUTPUT(Y0)
        OUTPUT(Cout)
        Y0,X1=Half_Adder(A0,B0)
        Y1,X2=Full_Adder(A1,B1,X1)
        Y2,X3=Full_Adder(A2,B2,X2)
        Y3,Cout=Full_Adder(A3,B3,X3)
END
```

Simulation and verification of the four bit adder circuit shown above is used here to illustrate HillSIM's performance. The circuit description contains four blocks, an XOR, a Half_Adder, a Full_Adder, and a 4_Bit_Adder. After reading the circuit description, the four bit adder block's gate array, which is to be simulated and its outputs analyzed, contains the gates shown below:

```
Half_Adder0.XOR0.g0:    type-NAND
Half_Adder0.XOR0.g1:    type-NAND
Half_Adder0.XOR0.g2:    type-NAND
Half_Adder0.XOR0.g3:    type-NAND
Half_Adder0.g4:         type-AND
Full_Adder1.XOR0.g0:    type-NAND
Full_Adder1.XOR0.g1:    type-NAND
Full_Adder1.XOR0.g2:    type-NAND
Full_Adder1.XOR0.g3:    type-NAND
Full_Adder1.g4:         type-AND
Full_Adder1.g5:         type-AND
Full_Adder1.XOR1.g0:    type-NAND
Full_Adder1.XOR1.g1:    type-NAND
Full_Adder1.XOR1.g2:    type-NAND
```

Full_Adder1.XOR1.g3: *type-NAND*
Full_Adder1.g10: *type-OR*
Full_Adder2.XOR0.g0: *type-NAND*
Full_Adder2.XOR0.g1: *type-NAND*
Full_Adder2.XOR0.g2: *type-NAND*
Full_Adder2.XOR0.g3: *type-NAND*
Full_Adder2.g4: *type-AND*
Full_Adder2.g5: *type-AND*
Full_Adder2.XOR1.g0: *type-NAND*
Full_Adder2.XOR1.g1: *type-NAND*
Full_Adder2.XOR1.g2: *type-NAND*
Full_Adder2.XOR1.g3: *type-NAND*
Full_Adder2.g10: *type-OR*
Full_Adder3.XOR0.g0: *type-NAND*
Full_Adder3.XOR0.g1: *type-NAND*
Full_Adder3.XOR0.g2: *type-NAND*
Full_Adder3.XOR0.g3: *type-NAND*
Full_Adder3.g4: *type-AND*
Full_Adder3.g5: *type-AND*
Full_Adder3.XOR1.g0: *type-NAND*
Full_Adder3.XOR1.g1: *type-NAND*
Full_Adder3.XOR1.g2: *type-NAND*
Full_Adder3.XOR1.g3: *type-NAND*
Full_Adder3.g10: *type-OR*

The simulation input file contains 256 lines for the exhaustive set of inputs (8 primary inputs), which is replicated to 1,000 to 10,000 lines to simulate a high number of vectors in order to measure execution time in milliseconds. Figure 3 shows the results in execution time vs. number of vectors simulated (number of lines in the simulation input file).

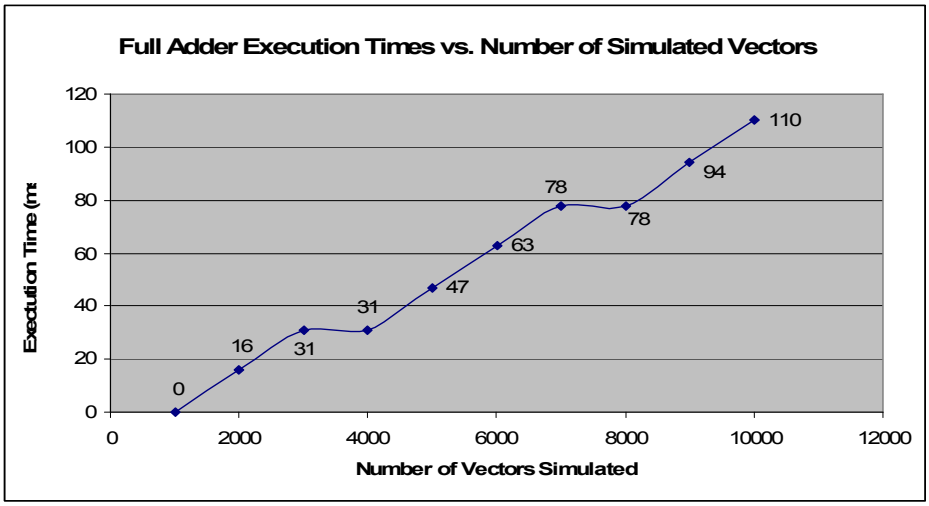

Figure 3: Execution Time (msec) vs. Number of Simulated Vectors (running on a Mobile AMD Athlon at 2.2GHz)

4.2 ISCAS 85 Circuits

In order to verify the ISCAS 85 combinational benchmark circuits using HillSIM, the bench circuit descriptions had to be modified. A beginning and end statement was added to each circuit description to produce a hierarchical bench circuit description. For example, the statement "BLOCK c17 was added to the beginning of c17.bench, and "END" was added at the end of the description. If the circuit contains XOR gates, an XOR block was added before the added "BLOCK *circuit*" statement, because HillSIM does not support XOR gates.

Execution times for each of the ISCAS benchmark circuits are show in Table 1 and Figure 4. We can see from figure 4 that the time complexity of HillSIM is slightly more than linear, and increases with both the number of gates and the number of inputs and outputs.

Circuit	Number of Gates	Number of Inputs	Number of Outputs	Execution Time (msec)
c17	6	5	2	0
c432	120	36	7	63
c499	162	41	32	141
c880	320	60	26	110
c1355	506	41	32	172
c1908	603	33	25	250
c2670	872	233	140	438
c3540	1179	50	22	485
c5315	1726	178	123	797
c6288	2384	32	32	781
c7552	2636	207	108	1125

Table 1: Execution Times for the ISCAS 85 Benchmark Circuits

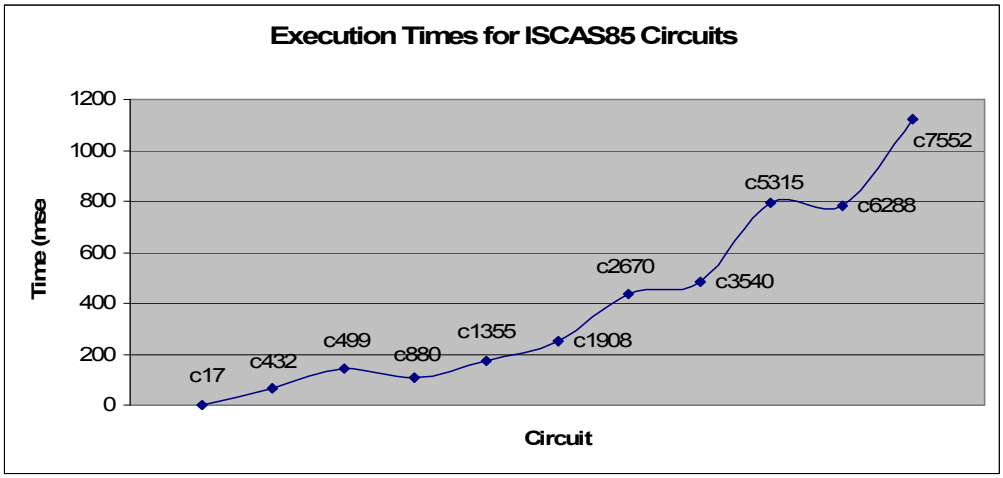

Figure 4: Execution Times for the ISCAS 85 Benchmark Circuits

To illustrate HillSIM's diagnosis, a gate was changed from NAND to NOR in the c17 benchmark circuit, and the lines "*10100 10*" and "*11000 11*" were used as inputs from the simulation input file. HillSIM produced the following diagnostic information:

```
ERROR DURING SIMULATION
  Input vector 10100 failed.
  Error observed on primary
output 23
  Gate Information along paths
leading to failing PO:
   Gate: g1 Type: NAND
    Input Signals: 3[1] 6[0]
    Output Signals: 11[1]
   Gate: g2 Type: NAND
    Input Signals: 2[0] 11[1]
    Output Signals: 16[1]
   Gate: g3 Type: NOR
    Input Signals: 11[1] 7[0]
    Output Signals: 19[0]
   Gate: g5 Type: NAND
    Input Signals: 16[1] 19[0]
    Output Signals: 23[1]
This program took 0 milliseconds
to execute.
```

This output shows that input vector 11000 passed, and input vector 10100 failed to produce the correct response on output 23 from the c17 circuit. Four gates are printed as suspected gates with input and output signal names and values specified. We can see that gate g3 of type NOR is printed, which is the same gate we modified to introduce an error in the circuit.

5. Conclusion

HillSIM provides a way to verify the logic behavior of a zero-delay combinational circuit described in the hierarchical bench format described in section 2.1. If verification fails, diagnostic information is printed to aid in design correction. HillSIM's diagnostic algorithm needs revision to narrow the diagnostic resolution. One idea to improve diagnosis is to store an array of pointers to each gate on the path to a failing output for each failing input vector. Gates that appear in every case can be printed to increase the resolution, and information of gates leading to correct outputs could be used to further increase diagnostic resolution by eliminating "good gates" from the suspected gates.

6. References

[1] H. Schildt, "C: The Complete Reference, Fourth Edition", Osborne/McGraw-Hill, 2000.

[2] M. L. Bushnell & V. D. Agrawal, "Essentials of Electronic Testing", Textbook, Kluwer Academic Publishers, 2000.