Mathematical Problems of Computer Science 33, 196–200, 2010.

Developing Game Model for Migration Process of Static Timing Analysis Tools

Tigran Sargsyan

State Engerineering University of Armenia
e-mail: sartig17@yahoo.com

Abstract

The aim of the work is to find an adequate game model for Static Timing Analysis (STA) migration process and develop method for evaluating the efficiency of actions on each step of migration. For comparing timing models generated by STA tool an automated comparison utility was created which should be used for finding the best strategy in each cycle of migration process

1. Introduction

Static Timing Analysis (STA) is a method of computing the expected timing of a digital circuit without requiring simulation. Figure 1 represents the input and output of STA tool. It is needed to mention that this diagram is a simplified model of STA tool.
The input of STA tool are device SPICE models, circuit netlist in SPICE or Verilog format, circuit parasitic in DSPF or SPEF format and configuration data.
The output of STA is a timing model of analysed circuit, which is often called '.lib'.It contains information on latch setup and hold, clock-gating setup and hold, minimum pulse width, domino precharge, and user-specified checking.
Lets discuss the migration from one STA tool to another. In this process Design Data remains unchanged, only Configuration data is being modified.
STA tool can be represented as a black-box model with Design Data and Configuration data inputs and the .lib(timing model) output. Modification of Configuration data affects the output data.(Figure 2)
From this point of view the the migration objective would be implemented if the structure of generated .lib("migrate to" tool) matches the structure of reference .lib("migrate from" tool). Statistics shows that the timing models(.libs) generated for the same circuit, using different STA tools with sufficient configuration do not match

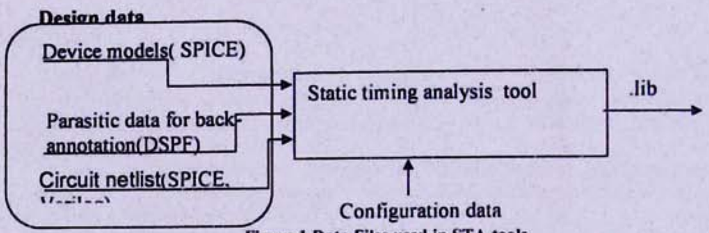

Figure 1 Data Files used in STA tools

completely, the average matching is 80%, Even in this case generated .libs represent adequate timing model of circuit being analized.

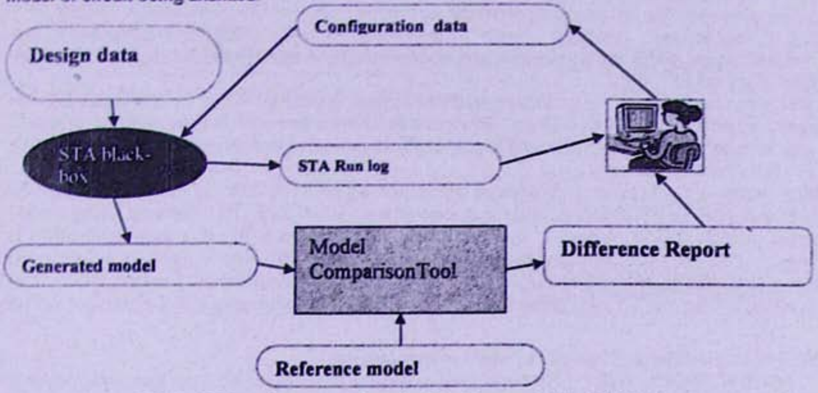

Figure 2 Cicle of Migration process by using black-box model for STA tool

Method

The process migration can be represented as a combinatorial game tree model. The game tree consists of H, T nodes (or vertices), which are points at which players can take actions, connected by edges, which represent the actions that may be taken at that node. Each H node represents the state at which the engineer(person performing migration) can take action, and T node represents the state at which STA black-box can run. Each transition from T to H is equivalent to transition of STA black-box output data to 'model comparison tool', and transition of 'difference report' to Engineer. Each transition from H to T is equal to run of STA black-box with 'configuration data' updated by engineer. An initial (or root) node represents the first decision to be made(by engineer). Every set of edges from the first node through the tree eventually arrives at a terminal H node, representing an end to the game. Each H node is labeled with the payoffs(P0...Pn) earned by engineer if the game ends at that node. The payoff value is the number of mismatches between timing model generated at current cycle of migration and the reference timing model, the aim of the game is to reduce this value to minimum. The calculation of payoff is beign calculated by 'model commaprison tool'.

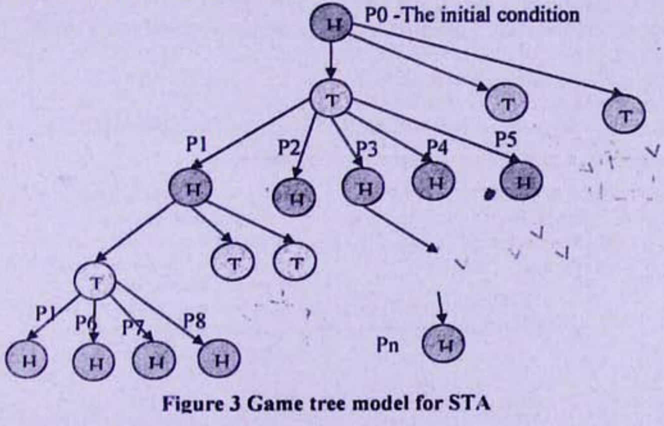

Figure 3 Game tree model for STA

.lib Timing Model Comparison Tools

The existing solutions for .lib comparing problems are
1.Text comparison utilities such as diff, cmp, tkdiff, Kompare etc.
2. 'versus' scripts which produce comparison between multiple data formats(Verilog, Library Exchange Format(LEF), .lib)
The first type of solution can not provide reasonable comparison accuracy.When .libs are generated with different configuration of STA tool, blocks which contain timing information may be disposed diversely and may have different timing attributes.The only exact diagnose that text comparison utilities can make is if two text files are equal or no.
Existing 'versus' tools do not provide detailed comparison between two .libs, only comparison of ports, and port directions. So it is needed develop an automated comparison utility for comparing timing models generated in each step of migration, and calculating payoff for each step.If a deep comparison is performed, then the list of mismatches generated by omparing utility can be used for planning the upcoming strategies. In this term strategy means the action to be taken(update in **Configuration data**) for fixing the mismatches between generated timing model and reference timing model in next run if STA tool.

1. Split input and Reference into blocks which contain only one " PORT_START(*)...PORT_END" statement and place them to respectively @in_port_blocks to @ref_port_blocks .
   Remove line carry symbols from elements of @in_port_blocks, @ref_port_blocks.
2. Each element @in_port_blocks is being passed to function Dissolve_port_block as an input argument. Empty hash array %in_hash is being created.

Function Dissolve_port_block
This function accepts a string as an input argument. Input string must contain only one "ORT_START(*)...PORT_END" statement. Dissolve_port_block creates an associative array with the following keys(table 1).
After creating all needed %port_inf entries Dissolve_port_block returns %port_inf hash array and stops working.
Returned hash array is being merged with %in_hash.

3. Same as in step 3. Returned hash array is being merged with %ref_hash.
4. For each key of %ref_hash check if key exists in %in_hash
   In case if key was found in both hash array calculate the difference between key values from %ref_hash and %in_hash and delete these entries from both hash arrays.
   If key is not found in %in_hash report a mismatch and print a mismatch report to output report file.
5. Add a "from reference file" tag to all values of %ref_hash
   Add a "from input file" tag to all values of %in_hash.
   Create empty hash array % unmatched_hash and add %ref_hash , %in_hash to this hash.
6. Sort keys of %unmatched_hash starting from the highest priority mismatches.
7. Print the list of unmatched keys to output report and exit.

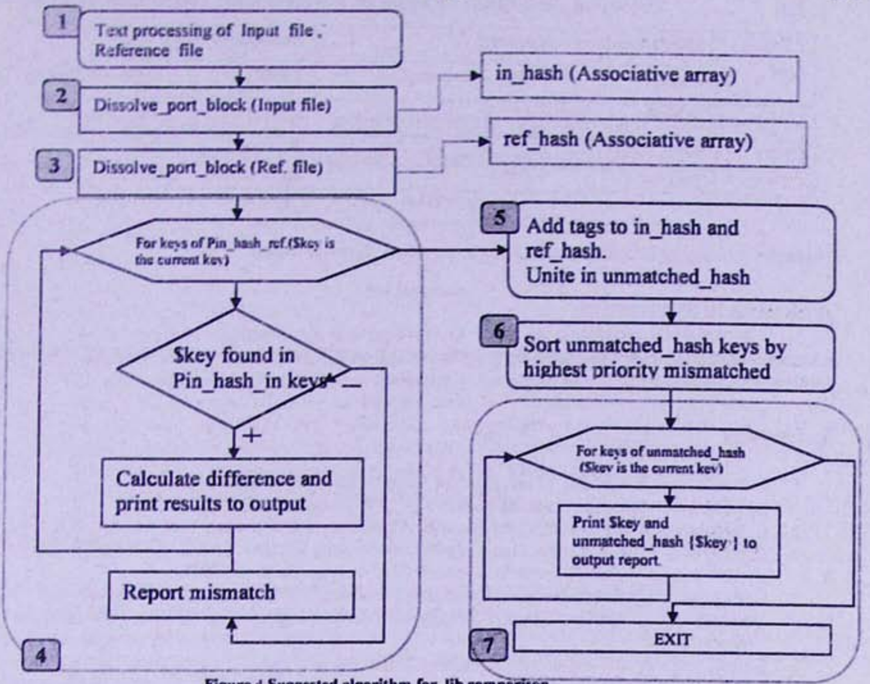
1
Text processing of Input file , Reference file
2
Dissolve_port_block (Input file)
in_hash (Associative array)
3
Dissolve_port_block (Ref. file)
ref_hash (Associative array)
For keys of Pin_hash_ref.($key is the current key)
$key found in Pin_hash_in keys
+
—
Calculate difference and print results to output
Report mismatch
4
5
Add tags to in_hash and ref_hash. Unite in unmatched_hash
6
Sort unmatched_hash keys by highest priority mismatched
For keys of unmatched_hash ($key is the current key)
Print $key and unmatched_hash {$key } to output report.
7
EXIT

Figure 4 Suggested algorithm for .lib comparison

Keys of %port_inf	Values of %port_inf
inv_out=>cap	0.17
inv_out=>dir	Out
inv_out=>timing=>start	inv_in
inv_out=>timing=>type	combinational
inv_out=>timing=>logic	inverse
inv_out=>timing=>inv_in=>combinational=>inverse=>rise_dl	0.1212, 0.2343, 0.3432, 0.4432
inv_out=>timing=>inv_in=>combinational=>inverse=>fall_dl	0.2432, 0.3453, 0.3785, 0.8762
inv_out=>timing=>inv_in=>combinational=>inverse=>rise_tr	0.0800, 0.0861, 0.0867, 0.0876
inv_out=>timing=>inv_in=>combinational=>inverse=>rise_tr	0.0800, 0.0861, 0.0867, 0.0876

Table 1 The syntax of keys in associative array

Conclusion

The developed method was tested for migration process from Pathmill to NanoTime.
Migration was performed for ISCAS85 C432, C880, C2670 combinational circuits.
Migration was done

1. Using Tkdiff for comparing libs.
2. New .lib comparison tool, with best strategy planning (RED)

The maximal matching between compared .libs was 71%.
Test results are presented in table 2.

Circuit type	*Migration time (minutes)*		*Number of iterations*	
	Tkdiff	**New .lib comp.**	**Tkdiff**	**New .lib comp.**
C432	165	112.5	23	14
C880	575	411	29	16
C2670	2460	1720	40	21

Table 2

Migration is being performed about by 31 % faster when "New .lib comp" is used.

Acknowledgements

I would like to express my gratitude to Professor Edward Pogossian for supervision of the work, constructive comments and support.

References

1. J. Bhasker and R. Chadha «Static Timing Analysis for Nanometer Designs»
   Janet Shapiro «Strategic Planning Toolkit» http://www.civicus.org
2. E. Pogossian, Adaptation of Combinatorial Algorithms, (in Russian), Yerevan., pp. 293, 1983.
3. E. Pogossian: "Combinatorial Game Models For Security Systems", *NATO ARW on "Security and Embedded Systems", Porto Rio, Patras, Greece*, Aug., pp. 8-18, 2005.
4. E. Pogossian, "On measures of performance of functions of human mind", *6th International Conference in Computer Science and Information Technologies*, CSIT2007, pp. 149-154, Yerevan, 2007.
5. Manual page for diff : http://ss64.com/bash/diff.html
6. Example of STA tool:
   http://www.synopsys.com/Tools/Implementation/SignOff/Pages/NanoTime.aspx
7. Information on Liberty format: http://www.opensourceliberty.org/
   http://www.synopsys.com/community/interoperability/pages/libertylibmodel.aspx
8. Lectures on STA: http://www.csee.umbc.edu/~cpatel2/links/641/slides/lect05_LIB.pdf

Ստատիկ ժամանակային վերլուծության գործիքների միգրացիայի պրոցեսի խաղային մոդելի մշակում

Տ. Սարգսյան

Ամփոփում

Այս աշխատանքի նպատակներն են՝ Ստատիկ Ժամանակային Վերլուծության գործիքների միգրացիայի պրոցեսի ադեկվատ մոդելի ստեղծումը և միգրացիայի յուրաքանչյուր քայլում ձեռնարկված գործողությունների արդյունավետության գնահատման մեթոդի մշակումը:
Ստեղծվել է ժամանակային մոդելների(Liberty ֆորմատով) համեմատության ծրագրային գործիք, որն օգտագործվելու է միգրացիայի յուրաքանչյուր փուլում հրամանների ճշտգրտման լավագույն ստրատեգիայի փնտրման համար: