A Hierarchical Approach Towards System Level Static Timing Verification of SoCs

Rupsa Chakraborty and Dipanwita Roy Chowdhury
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, INDIA - 721302
rupsac@cse.iitkgp.ernet.in, drc@cse.iitkgp.ernet.in

Abstract— **The high complexity and the core diversities make timing verification of an entire flattened SoC design a tedious process. In this paper, at first the various timing issues related to modular SoC verification have been investigated and then a bottom-up hierarchical approach of verifying the system level timing of an SoC, is presented. The timing abstractions of the cores are assumed to be provided by the core vendors. The interconnection delays of the SoC may be extracted from the SDF file generated after post layout simulation. The hierarchical approach provides a fast and systematic way of timing verification, as opposed to the flattened approach. Experiments were conducted on synthetic SoCs, using ISCAS benchmark circuits as cores. Results validate the claim of the proposed approach.**

I. Introduction

In a system-on-chip (SoC) technology, the time to design sign-off is reduced by using pre-designed, pre-verified re-usable IP-blocks called *cores* [1]. The cores may be designed either in-house or externally by various commercial vendors. The cores are pre-verified by the core-vendors, on timing constraints in accordance with the core specification. However, since the SoC environment in which the cores are to be integrated varies, and also since the cores are designed at various abstraction levels, the integrated system still remains to be validated. In fact, it is the task of the SoC integrator to verify that the complete SoC, after integration of all the cores, meets its timing closure in accordance with the SoC specification.

One way of tackling this problem is by completely flattening the final SoC design and then running the standard, commercially available, static timing analysis (STA) tools. Another approach is by conserving the core level hierarchy, inherent in SoC designs, through using efficient timing models or abstractions of the corresponding cores. Fortunately, today, there already exists a number of quality works on the generation of timing abstraction of IP blocks or macros. Some notable works are reported in [2]-[8].

The timing abstraction of a core contains a highly reduced information set with respect to the original model, as the structural information is removed and replaced by timing arcs corresponding to the ports around the core boundaries. A black-box timing model for a flip-flop based synchronous sequential core is shown in Fig. 1. Additionally, by hiding the internal details, the black-box model provides IP protection. A disadvantage with the black-box model is that it cannot

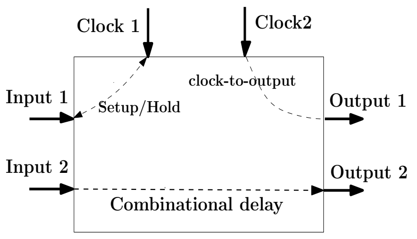

Fig. 1. A black-box model [2] of timing abstraction.

effectively model the complete latch behavior. It can do so only for a given clock waveform. Once the waveform changes, the model needs to be changed. Alternatively, a gray-box model may be used to tackle latch-based macros. A gray-box model extraction by graph reduction method has been proposed by Moon et al. in [6]. This method may sometimes need re-verification of the cores after integration. Recently, an advanced black box model has been proposed by Do et al. in [2], which can effectively model the timing of latch controlled systems without the need for re-verification after integration.

Given a valid timing model for the IP blocks, the next stride would be to find an efficient method of validating the timing closure for the complete system. A system level timing verification method, using the timing models of the IP cores, has been recently proposed in [10]. In this method, the SoC has been transformed into a timing graph, where the nodes represents the cores and edges represents the critical path between the cores. The back-annotated delays in the interconnects, obtained after post-layout synthesis, are validated with the specified waveform to check the timing. A problem with this approach is that the complexity increases for SoCs having a large number of cores. The method also uses a highly simplified SoC environment, with very small number of core boundary timing issues.

In order to tackle the complexity problem, a hierarchical approach may be employed. It is a divide and conquer approach which often leads to a much faster system level validation. Scheffer, in [11], has shown that if proper grouping can be made, then the complexity of a timing verification algorithm can be brought down to $O(N)$, from the complexity of at least $O(NlogN)$, where N is the size of the flattened hierarchy. Moreover, hierarchical timing

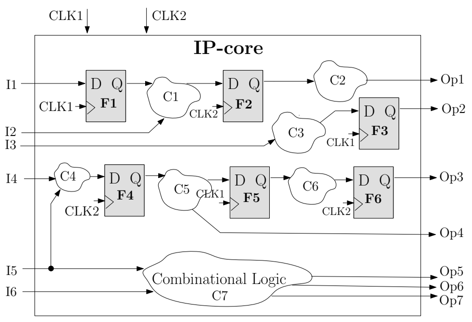

Fig. 2. IP core internal - registers and combinational paths.

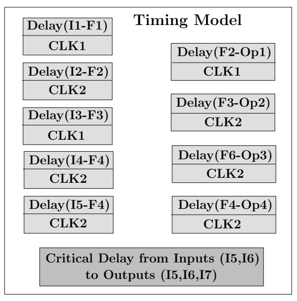

Fig. 3. Black-box timing model for the IP-core of Fig. 2.

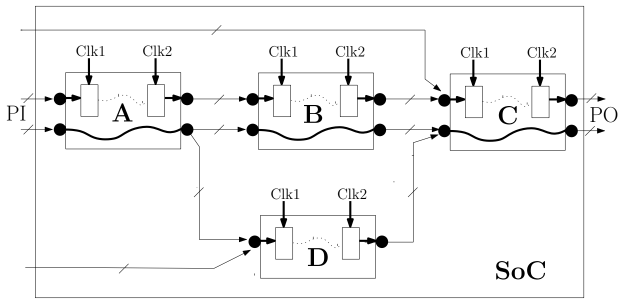

Fig. 4. Integration of cores in an SoC.

analysis of combinational circuits has been shown to be faster than gate-level analysis in some previous works [9].

In this paper, at first, the requirements for a hierarchical system level timing verification of a core-based design $-$ such as an SoC $-$ have been explored, and then an algorithm for hierarchical timing verification is proposed. The timing abstractions of the cores are assumed to be given by core vendors. The interconnection delays of the SoC may be extracted from the timing information file generated after post layout simulation. In the lowest level of the hierarchy are the cores and the glue logic. The glue logic is broken down into one or more modules according to mutual data propagation distance. Timing abstractions are generated for each such module. The cores and glue-logic modules are then considered to be black boxes and are further clustered according to data-propagation distance. Each of these clusters are timing verified concurrently. This continues till clustering results in a single module. This is the top level module. Verifying this top level module for timing results in the verification of the entire SoC for timing closure. Such concurrent verification approach results in a reduction of verification time by factors, as has been shown by experimental results.

The organization of the paper is as follows. Section II discusses the motivation and the timing issues related to system level timing verification of SoC. In Section III, the problem of timing verification in SoC is formulated. Section IV discusses the method of forming the clusters for hierarchical verification. Section V presents the timing verification algorithm. Experimental results have been provided in Section VI. Section VII concludes the paper.

II. Motivation and the SoC Timing Issues

In hierarchical timing verification of SoCs, the timing abstractions in the first level of the hierarchy, clearly, corresponds to that of the IP cores and glue logic. Glue logic is the user defined logic added to the SoC design by the SoC integrator, and not imported as cores. Assuming that the black-box timing models for each IP-core are provided, it now remain to be checked if, using the given models, the entire SoC design can be timing verified correctly. In other words, the results of timing verification of a completely flattened SoC should be equivalent to that of the results of hierarchical timing verification of the SoC using the timing models of the cores.

Fig. 2 shows the paths between the input pins, output pins and the registers within an IP-core. The flip-flops are driven by two clocks, $CLK1$ and $CLK2$. The irregular blobs denote combinational circuitry. Noticeably, there are the following kinds of paths (PT denotes path type) from input pins to the output pins of the IP core. $PT1$: Directly from input pin to register, without passing through any combinational circuitry (eg. $I1$ to $F1$). $PT2$: Input pin to register, with intermediate combinational path (eg. the path $I2-C1-F2$). $PT3$: Directly from register to output pins, without passing through any combinational circuitry (eg. $F3$ to $Op2$). $PT4$: Register to output pin, with intermediate combinational path (eg. the path $F4-C5-Op4$). $PT5$: Register to register, with or without passing through any combinational circuitry (eg. $F5$ to $F6$). $PT6$: Pure combinational paths from the input pin to the output pin (eg. $I5$ to $Op7$).

Fig. 3 shows the information contained in a viable black-box timing model for the IP-core of Fig. 2. The model accommodates all the pin-to-register, register-to-pin delays, and also the critical path delay of the purely combinational paths through $C7$. If there are wires connecting the input pins directly to the output pins, without passing through any combinational or sequential circuitry, then the estimated interconnection delays of each such wire would be provided in the model. Associated with the delay information of each register, is the clock waveform; since there may be more than a single clock driving the different registers. It is to be noted that there are no information related to the intermediate register-to-register paths, having no direct or combinational connectivity with the pins at the boundary (eg. path $F5$ to $F6$).

Fig. 4 shows an example SoC with four cores (A,B,C,D) and their interconnections. It is time, now, to investigate the timing issues related to the connectivity of the cores in the SoC with relation to their timing models. A timing verifier checks if signals reach the destinations in the desired time, so that the functionality of the circuit remains intact. Usually, the timing check-points are at the registers. At every register it is checked if the correct signal reaches the register in the right time. In case of an SoC too, this must be true for all its registers, during either flattened or hierarchical verification modes. Since, at the time of hierarchical verification, the register-to-register paths internal to the cores are already verified, the checking is left only for registers present at the boundary of the cores.

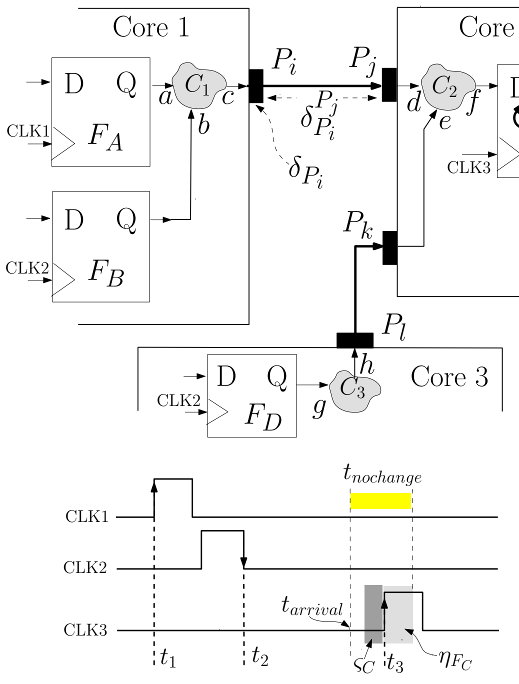

Fig. 5. Integration of cores in an SoC.

Fig. 5 shows a connectivity among four boundary flip-flops in three cores within an SoC. The flip-flops have three different clocks, *CLK*1, *CLK*2 and *CLK*3. From the waveforms shown in the figure, the clocks visibly hold the following properties:

- Three clocks are in three separate phases with respect to each other.
- Clocks *CLK*1 and *CLK*3 are positive edge triggered, while *CLK*2 is negative edge triggered.
- The clocks should be synchronized so that there is no setup or hold violations (discussed later in detail).

The timing models of all the three cores are assumed to be provided. In such case, the following information are known.

FROM THE GIVEN TIMING MODEL OF *CORE*1: delay δ, associated with output pin P_i, δ_{P_i}, is provided. This is the time a valid data takes to reach pin P_i from the immediate registers F_A and F_B.

$$\delta_{P_i} = Max.\{\delta^a_{Q_{F_A}}, \delta^b_{Q_{F_B}}\} + \delta^{C_1}_{critical} + \delta^{P_i}_c + (t_2 - t_1) \qquad (1)$$

Notations:

$\delta^a_{Q_{F_A}}$:	delay from the Q output of flip-flop F_A to input a of combinational circuit C_A.
$\delta^b_{Q_{F_B}}$:	delay from the Q output of flip-flop F_B to input b of combinational circuit C_A.
$\delta^{P_i}_c$:	delay from the output c of combinational circuit C_A to the pin P_i.
$\delta^{C_1}_{critical}$	Critical delay of the combinational path in C_1.
$t_2 - t_1$:	phase difference between clocks $CLK1$ and $CLK2$.

FROM THE GIVEN TIMING MODEL OF *CORE*2: delay,δ_{F_C} ,associated with the input D of flip-flop F_C. It is the time taken by valid data to enter the D-input, D_{F_C}, from the input pins, P_j and P_k.

$$\delta_{F_C} = Max.\{\delta^d_{P_j}, \delta^e_{P_k}\} + \delta^{C_2}_{critical} + \delta^{D_{F_C}}_f + \varsigma_C \qquad (2)$$

Notations:

$\delta^{D_{F_C}}_f$:	delay from output f of combinational path at C_2 to the D input of flip-flop F_C.
ς_C:	the rated *setup time* of flip-flop F_C.

Rest have similar meaning as mentioned for Equation 1.

FROM THE GIVEN TIMING MODEL OF *CORE*3: delay δ, associated with output pin P_l, δ_{P_l}, is provided. This is the time a valid data takes to reach pin P_l from its immediate register F_D. Notations have similar meaning as in Equations 1 and 2.

$$\delta_{P_l} = \delta^g_{Q_{F_D}} + \delta^{C_3}_{critical} + \delta^{P_l}_h \qquad (3)$$

Now, it is required to be validated if valid data reaches the D input of flip-flop F_C at each rising edge of the clock *CLK*3. Clearly, after integration, the delay at the D input of F_C changes from, δ_{F_C}, which is given in the model for *Core*2. After integration, the following points need to be considered as well.

- The external delay just behind the core input pins, leading to the objective flip-flop. In this case, it is the delay up to the pins P_j and P_k for the flip-flop F_C.
- Phase differences between the clocks in different cores from which there is a path to the objective flip-flop. In this case, we need to consider the phase differences between the clocks *CLK*1, *CLK*2, and *CLK*3.

Validation of Data Arrival Time. In order to have the valid data reach D_{F_C} at the right time, the following relation should hold:

$$Max.\{t_1 + \delta_{P_i} + \delta^{P_j}_{P_i} + \delta_{P^d_j},\ t_2 + \delta_{P_l} + \delta^{P_k}_{P_l} + \delta^e_{P_k}\} + \delta^{C_2}_{critical} + \delta^{D_{F_C}}_f \leq t_3 - \varsigma_C \qquad (4)$$

Validation of Data Hold Time. The data arriving at D_{F_C} should be held there for until the arrival of the rising edge of *CLK*3 plus the rated *hold time*, η_{F_C}, of the flip-flop.

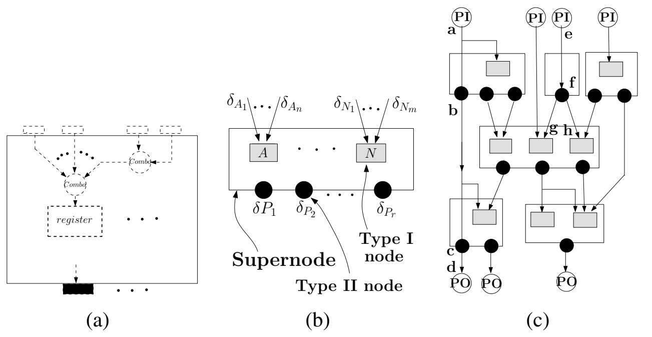

Fig. 6. (a) A single core data. (b) A supernode, (c) The directed acyclic graph at Level-1.

Therefore, synchronization should be guaranteed such that data do not change within this time. If data should remain unchanged for time $t_{nochange}$, and $t_{arrival}$ is the data arrival time at D_{F_C} then,

$$t_{nochange} \geq t_3 - t_{arrival} + \eta_{F_C} \qquad (5)$$

So long verification of timing for only F_C was dealt with. For a complete SoC timing verification, similar timing check is required at every flip-flop inputs at the core boundaries, and matched with the given SoC timing specification. Any violation must be reported. The timing verification algorithm proposed in this paper, takes care of the above criteria on all the receiving flip-flops at the core boundaries. For this, the entire SoC is modeled as a graph, and timing checks are done at the input of each node of the graph.

III. Problem Formulation

The least information that could be useful from the timing model of each core is when the core consists entirely of combinational circuits. In such case, only the delays associated with the core outputs will be required. The core model in Fig. 6(a) illustrates it by showing that the register information at the core input boundary may or may not exist, but there must exist at least one output port delay information.

In the proposed method, the SoC is modeled as a directed acyclic graph (DAG), $G = \{S, E, W\}$, as shown in Fig. 6(c). Acyclic, since, feedback paths among cores have not been considered. The graph G consists of a set of supernodes S, a set of edges E, and a set of weights W associated with the edges and node, and also the primary input nodes PI and the primary output nodes PO. A supernode, further, consists of two sets of node, $TypeI$ nodes, and $TypeII$ nodes (see, Fig. 6(b)). Thus, $S = \{TypeI, TypeII\}$. The elements of the $TypeI$ set are the receiving registers at the core input. The elements of the $TypeII$ set are the ports at the output of each core. The $TypeI$ set can be empty, but the $TypeII$ set is never empty. The set PI consists of the nodes denoting the primary inputs of the SoC. The set PO consists of the nodes denoting the primary outputs of the SoC. The set W further consists of two sets, W_E and W_N, i.e., $W = \{W_E, W_{N^I}, W_{N^{II}}\}$. The set elements of the set W_E are delays associated with each edge starting from a core output port and leading to the combinational circuitry present before the receiving flip-flop − if there is one; otherwise leading to the D-input of the receiving flip-flop. For $TypeI$ category of nodes (flip-flops), the associated delays are contained in the set W_{N^I}. If the D-input to the corresponding flip-flop is driven by a combinational logic then the delay in W_{N^I} for the flip-flop is a summation of the critical delay of the logic, the delay of the wire leading to the D-input and the rated setup time of the flip-flop. If there is no combinational logic before the $D-$input then the delay value is just the rated setup time of the flip-flop. The delays associated with the $TypeII$ category of nodes (output ports) are contained in the set $W_{N^{II}}$. This delay is the delay a valid data takes to reach the output port either from the input port of the core (for purely combinational cores), or from one or more registers driving that port. Each element in W_{N^I} and W_{N^II} also contains the clock information of flip-flops associated with those nodes.

After the DAG has been built, the problem of timing verification of the entire SoC reduces to checking if every edge in the graph meets the timing requirement of the SoC timing specification. Any mismatch has to be reported as an error.

At this point, it should be noted that all the $TypeI$ nodes can be verified concurrently. The purely combinational edges, that is, edges connected by $TypeII$ nodes must be verified together, since there is time dependencies between these edges. For instance, in Fig. 6(c), the edges {ab, bc, cd} have to be verified together since the delay from each edge is added up with the next edge to get the final delay. Other edges for sequential verification are {ef, fg}, {ef, fh}. The rest of the edges can be tested in parallel. Because parallel testing is possible, as a step towards further reduction of the verification time, the graph may be divided into clusters and each such cluster may be verified concurrently.

IV. Building the Hierarchy

The proposed hierarchical verification method proceeds as follows:

- Construct a DAG, namely Level-1 graph, storing the timing models corresponding each IP-core, interconnecting lines, and any feedbackless combinational or sequential path between them, see Fig. 6(c).
- If there is an input-to-output combinational path in any IP-core timing module, then a separate edge is added for the input-to-output line; and this delay is removed from the original timing model. The weight of the new edge is a summation of the preceding and following edge delays, and also the input-to-output combinational delay.
- Remove the supersets and construct a weighted forest, namely Level-2 forest, representing the SoC with nodes corresponding to flip-flops and edges corresponding to critical delay paths leading to the flip-flops from a driving flip-flop (attached to an output port) or a primary input. The critical delay, in the form of weight, is associated with each edge.
- Form various clusters out of the trees in the forest.

- All trees residing in the in the same cluster will be verified sequentially.
- Trees residing in different clusters will be verified in parallel.
- There is a constraint on the number of clusters that can be formed.
- The motive is to reduce the number of modules to be verified sequentially and, in effect, the total verification time.

• Do timing verification.

A. Transformation of the Glue Logic

The components of an SoC just before constructing the Level-0 graph are as follows:

1. Timing abstraction of each IP-core.
2. Interconnections between the IP-cores.
3. Glue logic. This is the extra logic circuitry added into the SoC by the SoC integrator for completeness of SoC functionality.

In order to have a completely hierarchical verification, the glue logic is processed separately and its timing model is extracted. Then this logic is replaced by the corresponding timing model in the Level-0 graph, as shown in Fig. 7.

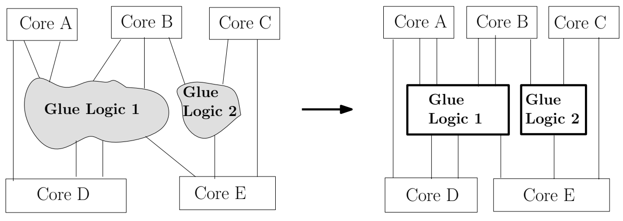

Fig. 7. Conversion of glue logic to a timing abstraction.

B. Construction of Level-1 Graph

The SoC is modeled as a directed acyclic graph $G = \{S,E,W\}$, as shown in Fig. 6(c). Each of S, E, and W has been defined explicitly in the previous section of problem formulation. A weight is associated with each edge denoting its delay. The delay of the edges have been generated by extracting circuit information from the post-layout timing information. The post-layout synthesis of the SoC design can be done using a commercial front-end and back-end synthesis tool. Each core during synthesis have been kept 'untouched' so that their modular properties are retained. The post-layout timing information file produces all the pin-to-pin delays. The delays from the core boundaries were used to construct the black-box timing models of the cores. The delays at the core interconnects were used in forming the delay associated with edges.

C. Clustering - Formation of the Hierarchy

At any level in the hierarchy, there is a fixed constraint, N_c, on the number of clusters to be made, denoting the degree of parallelism in the algorithm. This constraint can be due to restriction in the number of threads of the verification software. At the same time, the target should be to maximize

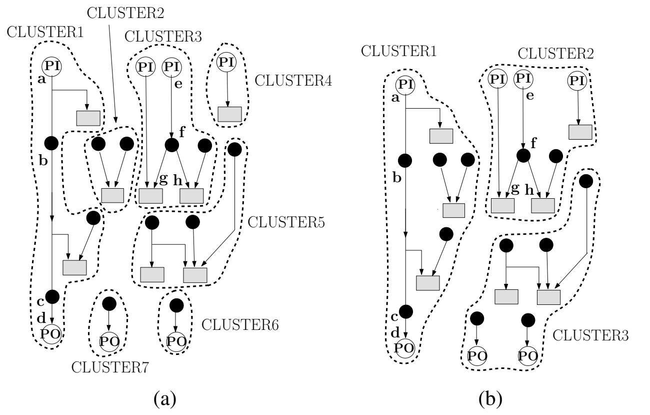

Fig. 8. (a) After pass-1 of clustering the Level-2 forest (b) After pass-2 clustering the Level-2 forest, restricted to three clusters.

the number of clusters within this constraint. By increasing the number of clusters, the timing verification process could be made faster. The clustering process tries to equalize the number of timing paths to be verified in each cluster. This is to guarantee that each process needs to utilize similar amount of time in verification; and no process needs to wait too long for another process to finish. Once clustering is completed, the clusters are scheduled for verification, concurrently.

Fig. 8(a) shows a clustering on the DAG of Fig. 6(c). Clustering algorithm works in two passes. In the first pass, all the connected modules are kept in the same cluster. So, each cluster, in the first round, consists of a single connected module. The connected clusters may be obtained just by removing the boundary created by the core modules (supernodes, shown in Fig. 6(b) as the supernode box). Each module is a tree. Thus what remains after the superset boundaries are removed is a forest. This leads to the formation of the Level-2 forest.

Since there is a constraint on the number of clusters, in the second pass of the algorithm, the trees in the Level-2 forest are merged together into a single module. The merging is done in such a fashion that the total number of edges in each cluster is almost the same. The merging is done until the number of clusters equals the given constraint, see Fig. 8(b).

V. The Hierarchical Timing Verification Algorithm

Once the cluster formation is complete, now its turn for the timing verification of the entire SoC. This is done by concurrently validating each of the clusters. The validation process for each cluster is the same, and hence the following discussion will proceed by taking into account a general cluster. The pseudo-code explaining the process of timing verification is given in **Algorithm**_*TimingVerfication*.

For each *TypeI* node (flip-flops in the inputs), all edges converging to it is found. Each *TypeI* node has a delay associated with it, stored in its corresponding element in W_{NI}. This edge is the summation of the combinational path delay combinational block input to its *D*-input, plus its rated setup time delay. The maximum delay of all lines leading

TABLE I

COMPARISON OF RUN TIME WITH 10 CLUSTERS, AND WITHOUT CLUSTERS.

SoC Name	ISCAS Cores (how many used)	Run time (sec.) no cluster	Run time 10 clusters	%age reduction in program run time
SoC1	c432(6),c1908(2),s444(2),s546(1)	38	5	86.84
SoC2	c499(1),c1355(2),s820(2)	22	3	86.36
SoC3	c2670(1),c5315(1),s1196(1)	41	7	82.92
SoC4	c432(2),c6288(1), s1494(1)	49	7	85.71
SoC5	c880(1), s444(1), s820(1)	33	4	87.87

```
Algorithm_TimingVerification
  Input: Cluster generated after pass-2.
  Output: A verification report.

  BEGIN
  1. for($n_i \in TypeI$, $i = 1, ..., |TypeII|$) do
  2.     $E_{n_i} \leftarrow edges\_converging\_to\_ni()$
  3.     $\delta_{max} \leftarrow MAX.\{e_k\}$,
         such that, $e_k \in E_{n_i}, k = 1...|E_{n_i}|$
  4.     $D \leftarrow \delta_{max} + d_{n_i}$, such that, $d_{n_i} \in W_{N^I}$
  5.     if $eqn.(4)$ is violated then
  6.         report setup violation
  7.     if $eqn.(5)$ is violated then
  8.         report hold violation
     endfor
  END
```

to this flip-flop is found and this delay is added to the node delay. Now, Equation 4 is checked. If this equation is not satisfied then a setup time violation is reported. Similarly, checking with Equation 5 leads to the detection of any hold violation. Similar checks are done to all nodes belonging to the set *OP*. This leads to the detection of any critical delay violation in the combinational path, with respect to the SoC timing specification.

VI. EXPERIMENTAL RESULTS

Experiments were conducted on synthetically generated SoCs using ISCAS'85 combinational and ISCAS'89 sequential benchmark circuits as cores. The algorithm *timingVerification* was implemented in C programming language and run on Intel(R) Core(TM)2 processor with 1.66GHz speed, 1.5GB of RAM. The run times of the program on various SoCs have been listed in Table I. In column 2, the ISCAS circuits are listed. The numbers in brackets denoted the number of times a particular benchmark circuit is repeated for the SoC design. The gate and interconnection delays were extracted from the post-layout timing information file generated after post-layout synthesis of the SoCs, using a commercial synthesis tool. In column 3 and 4, the run times are measured in seconds. Column 3 lists the run times when no clustering is done for verification, and column 4 list the run times when the number of clusters are restricted to 10.

In all cases, it can be observed that around 85% reduction in the program run time is observable. Note that, for results on no clustering, the SoC was not completely flattened. The basic core-based hierarchy was restored. If the program is run on a completely flattened SoC, then definitely the program run times would be higher. From the results, it can be concluded that considerable amount of program run time can be saved using a hierarchical timing verification approach.

VII. CONCLUSION

In this paper, a hierarchical timing verification approach has been proposed, based on timing abstractions on IP-cores and subsequent clustering. The proposed method checks the timing closure of the entire SoC design, in its system level. Various timing issues have been investigated, and a method of modeling the SoC timing criteria has been discussed. The hierarchical approach leads to a lowered verification time. Experimental results on synthetic benchmark SoCs built out of ISCAS'85 and ISCAS'89 benchmark circuits validate the proposed approach.

REFERENCES

[1] M. Keating, P. Bricaud, "Reuse Methodology Manual for System-on-a-Chip Designs", *Springer*, Edition 3, 2002

[2] K. T. Do, Y. H. Kim, H. S. Son, "Timing modeling of latch-controlled sub-systems", *Integration, the VLSI Journal*, Vol. 40, Issue 2, pp.62-73, Feb. 2007

[3] S. Zhou, Y. Zhu, Y. Hu, R. Graham, M. Hutton, C.-K. Cheng, "Timing Model Reduction for Hierarchical Timing Analysis", *IEEE/ACM International Conference on Computer-Aided Design, ICCAD'06*, pp. 415-422, Nov. 2006

[4] R. Chen, H. Zhou, "Timing macro-modeling of IP blocks with crosstalk", *IEEE/ACM International Conference on Computer-Aided Design, ICCAD'04*, pp. 155-159, 2004

[5] A. J. Daga, L. Mize, S. Sripada, C. Wolff, Q. Wu, "Automated timing model generation", *Proc. 39th Design Automation Conference, DAC'02*, pp. 146-151, June 2002

[6] C. W. Moon, H. Kriplani, K. P. Belkhale, "Timing model extraction of hierarchical blocks by graph reduction", *Proc. 39th Design Automation Conference, DAC'02*, pp. 152-157, June 2002

[7] M. Foltin, B. Foutz, S. Tyler, "Efficient stimulus independent timing abstraction model based on a new concept of circuit block transparency", *Proc. 39th Design Automation Conference, DAC'02*, pp. 158-163, June 2002

[8] S. V. Venkatesh, R. Palermo, M. Mortazavi, K. A. Sakallah, "Timing abstraction of intellectual property blocks", *Proc. of IEEE Custom Integrated Circuits Conference*, pp. 99-102, May 1997

[9] H. Yalsin, J. P. Hayes, "Hierarchical timing analysis using conditional delays", *IEEE/ACM International Conference on Computer-Aided Design, ICCAD'95*, pp. 371-377, Dec. 1995

[10] R. Chakraborty, D. Roy Chowdhury, "Raising the Level of Abstraction for the Timing Verification of System-on-Chip",*Proc. IEEE Computer Society Annual Symposium on VLSI, ISVLSI'08*, pp. 459-462, 2008

[11] L. K. Scheffer, "Some conditions under which hierarchical verification is O(N)", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, Issue 5, pp. 643-646, May 2003