Kasetsart J. (Nat. Sci.) 36 : 319 - 326 (2002)

Parallel Logic Synthesis Optimization for Digital Sequential Circuit

Aswit Pungsema and Pradondet Nilagupta

ABSTRACT

High-level synthesis tools are very important for designing electronic circuits. A lower level logic gates are synthesized by optimization of the circuit's combination part, which is then realized by mapping on programmable devices such as FPGAs. This synthesis process is a computation intensive task. In this paper, we propose an alternative method to synthesis a sequential logic circuit which reduces time consuming in synthesis process. First using a parallel partitioning algorithm partition the whole circuit into sub-circuits and then using parallel sub-circuit synthesis in order to reduce computation. The LGSynth'91 benchmark suite used for experiment is in net-list format. Our result shows that the number of partition is increasing whereas the synthesis time is reduced as the number of processor is increased.

Key words: parallel logic optimization, graph partitioning, logic synthesis, sequential circuit

INTRODUCTION

With general purpose parallel processing machines becoming more commonplace, parallel processing is being used extensively to solve a variety of VLSI CAD algorithms. In recent years, there has been a tremendous amount of research on algorithms for optimization problems. Many engineering design problems which are classified as optimization problems have benefited from the results of this research. In particular, design problems can be modeled mathematically and solved by mathematical programming techniques. There remain many logic synthesis optimization design problems which are considered too complex to be solved in a reasonable amount of time using single-processor computers.

Logic synthesis forms an important part of many VLSI CAD applications. In this paper we concentrate on circuits partitioning for parallel logic synthesis in which we exploit parallelism by partitioning the circuit being synthesis. The logic synthesis problem is started as follows. Given a state-transition-graph (STG), generate a logic function from STG, and then optimize the logic (logic synthesis) and finally device mapping. Logic synthesis, given a logical function $f(x_1,\ldots,x_s)$, composed of an on-set $F(x_1,\ldots,x_s)$ and a don't-care-set $D(x_1,\ldots,x_s)$, finds a sum-of-products expression (a cover of f), $G = g_1+\ldots+g_t$ with minimum t, such that $F \subseteq G \subseteq F+D$.

Many algorithms have been published for various stages of sequential synthesis. For synchronous circuits, these include methods for state assignment (Lin and Newton, 1989 and Villa and Vincentell., 1990), state minimization (Hachtel *et al*., 1991), testing (Ghosh, 1991), retiming (Leiserson *et al*., 1983.), technology mapping (Moon *et al*., 1989), verification (Bryant. 1986 and Coudert *et al*., 1989), timing analysis, and optimization across register boundaries (DeMicheli, 1991 and Malik *et al*., 1991). For asynchronous circuits, these include methods for hazard-free synthesis (Lavagno *et al*., 1991). However, no comprehensive

Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Bangkok 10900, Thailand.

evaluation of the algorithms and no complete synthesis system in which all of these algorithms are employed, has been reported to date. A complete sequential circuit synthesis system is needed, both as a framework for implementing and evaluating new algorithms, and as a tool for automatic synthesis and optimization of sequential circuits.

MATERIALS AND METHODS

At present the device density is very high, enabling the circuit designer to design a larger circuit. However, processing problems arise when the netlist is too large and consumes all of the available memory. In this paper, we propose an alternative method to synthesis a sequential logic circuit which reduces time consuming in synthesis process. First using a parallel partitioning algorithm partition the whole circuit into sub-circuits and then using parallel sub-circuit synthesis in order to reduce computation as shown in Figure 1 and Figure 2.

There are potential problems with this approach. The first problem is that the large circuit must be partitioned into N sub net-lists (define 'N' before the partition) which balance the size of each sub net-list with respect to performance of the next process. The second problem is that the connections between nodes during partitioning process must be minimal, because the network speeds is slower than CPU. The last problem is the global minimizations of synthesis results which consumes computation time.

This paper focused on circuit partitioning which is modeled as a graph. Since the next step, a parallel synthesis uses a partition circuit as an input. The graph partition algorithms we choose for circuit

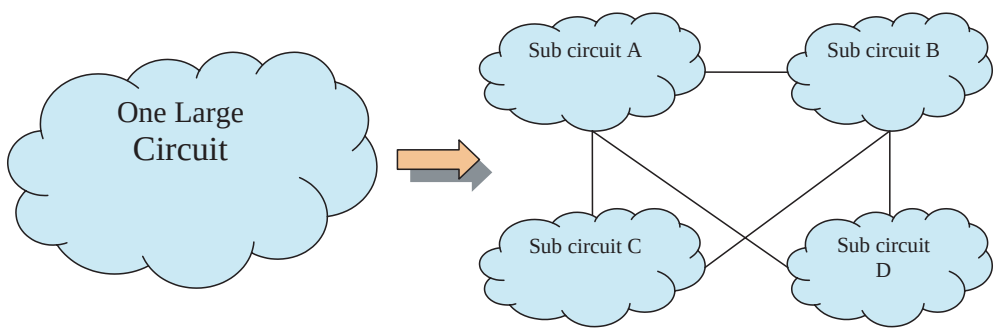

Figure 1 The first step of partitioning a circuit.

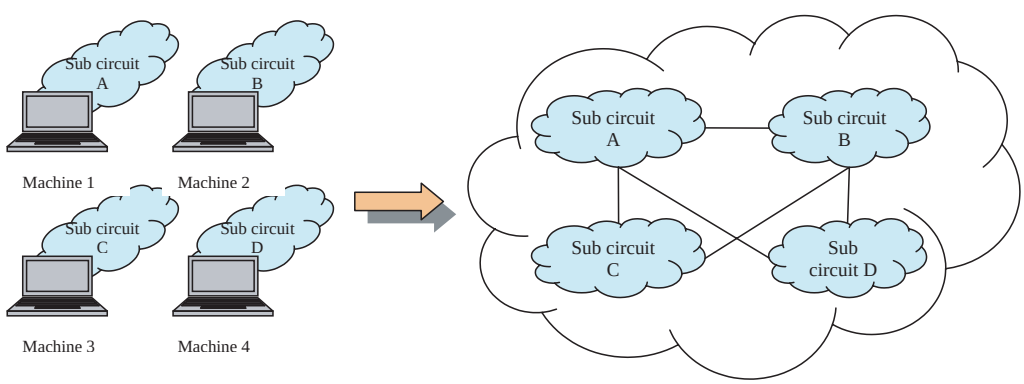

Figure 2 Thesecond step is to distribute the logic synthesis. The third step is to join the sub-circuits back into one circuit.

partitioning must be possible to use in parallel for increase speed and reduce share memory. For instance, the solutions for a linear equation ($Ax = b$) applied using iterative methods on a parallel computer gives rise to a graph partitioning problem. A key step is a multiplication of a sparse matrix and a dense vector. Graph partitioning corresponded to matrix A, is used to significantly reduce the amount of communication time. If direct parallel methods are used to solve a sparse system of equations, then a graph partitioning algorithm can be used to compute a fill reducing order that leads to a high degree of concurrency in the factorization phase. The multiple minimum degrees ordering used almost exclusively in serial direct methods, is not suitable for direct parallel methods, as it provides limited concurrency in the parallel factorization phase.

The graph partitioning problem is NP-complete. However, many algorithms have been developed that result in a reasonably good partition. Recently, a new class of multilevel graph partitioning techniques were introduced by Hendrickson and Leland (1993), and further studied by Karypis *et al.*, (1998). This multilevel algorithm provides an excellent graph partitioning and a moderate computational complexity. Even though these multilevel algorithms are quite fast compared with spectral methods, parallel formulations of multilevel partitioning algorithms are needed for the following reason. The amount of memory on serial computers is insufficient to allow the partitioning of graphs corresponding to large problems that can now be solved on massively parallel computers and workstation clusters. Subsequently, a parallel graph partitioning algorithm can take advantage of the significantly higher amount of memory available in parallel computers.

Unstructured graph partitioning algorithm

The problem is to partition the vertices of a graph into *p* roughly equal parts, such that the number of edges connecting vertices in different parts is minimized. The ***p-way*** graph partitioning problem is defined as followed : Given a graph G = (V, E) with |V| = n, partition V into p subset, V_1, V_2,…, V_p such that $V_i \cap V_j = \varnothing$ for $i \neq j$, $|V_j| = n/p$, and $\cup_j V_j = V$, consequently the number of edges of E whose incident vertices belong to different subsets is minimized. A p-way partitioning of V is commonly represented by a partitioning vector P of length n, so that for every vertex ($v \in V_1$,P[v]) is an integer between 1 and p. This indicated the partition to which the vertex (v) belongs. Given a partitioning (P), the number of degrees whose incident vertices belong to different partitions is called the *edge-cut* of the partition.

The various phases of the multilevel graph bisection are shown in Figure 3. There are three phases in multilevel graph bisection, *coarsening phase*, *initial partitioning phase*, and *uncoarsening and refinement phase*. During the coarsening phase, the size of the graph is successively decreased. During the initial partition phase, a bisection of a smaller graph is computed, and then during the uncoarsening phase, the bisection is successively refined as it is protected to a larger graph. During

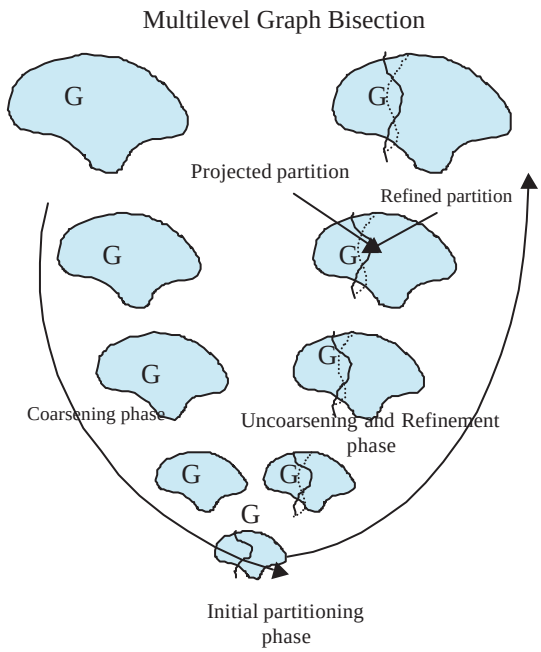

Figure 3 The multilevel graph bisection.

the uncoarsening phase a dotted lines indicate projected partition, and a solid line indicate partitions that are produced after refinement.

Parallel multilevel graph partitioning algorithms

The basic concept of parallel multilevel graph partitioning algorithms is as follows. There are two parallel methods that can be exploited in the p-way graph partitioning algorithm based on the multilevel bisection, as described in previous section. The first parallel method is due to the recursive nature of the algorithm (Karypis, 1996). Initially a single processor finds a bisection of the original graph, and then two processors find bisections of those two newly created sub-graphs and so on. However, this scheme can only use up to log p processors, and only reduces the overall run time of the algorithm by a factor of O (log p). We will refer to this parallel method as the procedure associated with the recursive step.

The second parallel method can be exploited during the bisection step. In this case, instead of performing the bisection of the graph on a single processor, we perform it in parallel. We will refer to this parallel method as the procedure associated with the bisection step. Note that if the bisection step is done in parallel, then the increase in speed obtained by the parallel graph partitioning algorithm can be higher than O(log p).

The parallel graph partitioning algorithm we have been describing in this section exploits both of these parallel methods. Initially all the processors cooperate to bisect the original graph G, into G^0 and G^1. Subsequently half of the processors bisect G^0, while the other half of the processors bisects G^1. This step creates four subgraphs G^{00}, G^{01}, G^{10}, and G^{11}. Then each quarter of the processors bisect one of these subgraphs and so on. After a log p steps, the graph G has been partitioned into p parts.

The ParMETIS library (Karypis *et al*., 1998) provides a variety of algorithms. ParMETIS is an MPI-based parallel library that implements a variety of algorithms for partitioning unstructured graphs, meshes, and for computing fill-reducing orderings of sparse matrices. A brief overview of the functionality of PARMETIS is shown in Figure 4. ParMETIS extends the functionality provided by METIS and includes routines that are especially suited for parallel AMR computations and large scale numerical simulations. We investigated some partitioning algorithm matching our problem. We

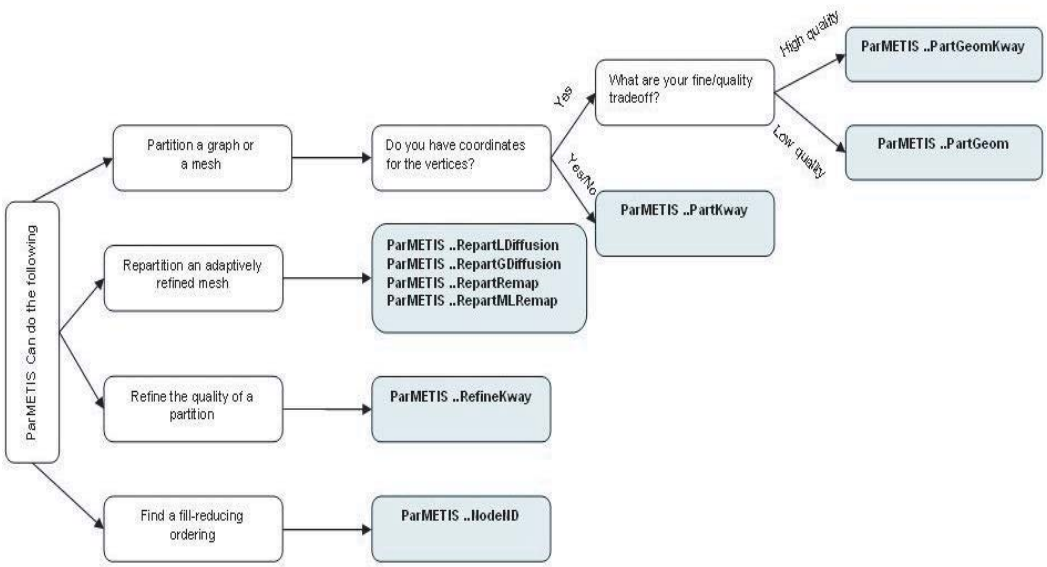

Figure 4 Abrief overview of the functionality provided by partMETIS.

started verifying those algorithms in order to find the algorithm that gives the best result respect to our scope. PARMETIS_PARTKway is based on the serial multilevel k-way partitioning algorithm describe in previous section and used the second part of the parallel method. This k-way partitioning algorithm shows that it produces a quick partition and high quality partition. The multilevel k-way partitioning algorithm works as follow. First, a graph is gradually coarsened down so it contains a few hundred vertices. Second, a k-way partition of this smaller graph is computed, and then this partitioning graph is projected back to the original graph (finer graph) by periodically refining thee partition. Since the finer graph has more degrees of freedom, such refinements improve the quality of the partition.

Implementation

We started our implementation by porting SIS version 1.2 (Sentovich *et al*., 1992) to the Linux platform in order to parse the netlist file format. After that we built the module to map the netlist in to the graph structure for processing and testing by ParMETIS for partitioning and finally we built the module to merge the sub-circuits and verify the circuit correction. The experiments on the LGSynth'91 benchmark suite (Saeyang, 1991) were performed in PIRUN cluster (Uthayopas *et al*., 2000) using 16 nodes of the Beowulf PC Cluster, Pentium III 500 MHz processors, with 128MB memory and 100Mbps Ethernet connections.

RESULTS AND DISCUSSION

The first phase of mapping netlist to graph is to analyses the characteristic of sub circuit results. This paper will present results using time and the quality of circuit output in order to select the best parameter for the next step. We focused on the boolean part of RTL circuits. The set of equations describing the circuit is represented by a boolean network. This network is a directed acyclic graph (DAG). Each node of DAG is associated to each boolean equation, and there is an arc between two nodes N_i and N_j if the output of N_i is an input to N_j.

Figure 5-7 show the partitioning results for each circuit into 2, 4, 6, 8, 10, 12, 14 and 16 partitions running on 2, 4, 8, and 16 processors (workstations). The results show time consumption will be affected by the number of processors. The network I/O is leaking the CPU time. From the partitioning algorithm, the graph image will be distributed to each process and then synchronized at a random node to perform the bisection algorithm. This process consumes a lot of network I/O of communicating to each other. The time consume for partition on each circuits trend to increase since the complexity of each circuits are increase. The time consuming for partition circuit into bigger partition is decrease i.e. partition circuit into 16 partitions is better than partition into other partition size when the number of processor is increased. However, there are some partition size is better than partition into 16 partition because of the complexity of circuit. When the complexity of a circuit is increased, the more partition divides, the time consume is better. The s38417 is more complicate than s15850 and s1488.

Table 1 and 2 show the synthesis runtime and edge-cut of sub-circuit VS. number of processors. Edge-cut is total number of links to each of the other sub-circuits. From Tables 1-2 we choose the optimum number of sub-circuits by the number of processors. The synthesis runtime is decreased when we use a processor up to 16 processors. The speed up of synthesis runtime is increased as the circuit is more complex. The speed up of circuits1488, s15850, and s38417 are 0.6, 8.2 and 7.7 respectively. The number of edge-cut is increased as the number of processor is increased. The results, as compared to sequential execution, are obtained through a complex circuit taking a long time at one processor. Parallel runtime is calculated from logic synthesis time and merge time (sequential). Since parallel runtime is calculated by

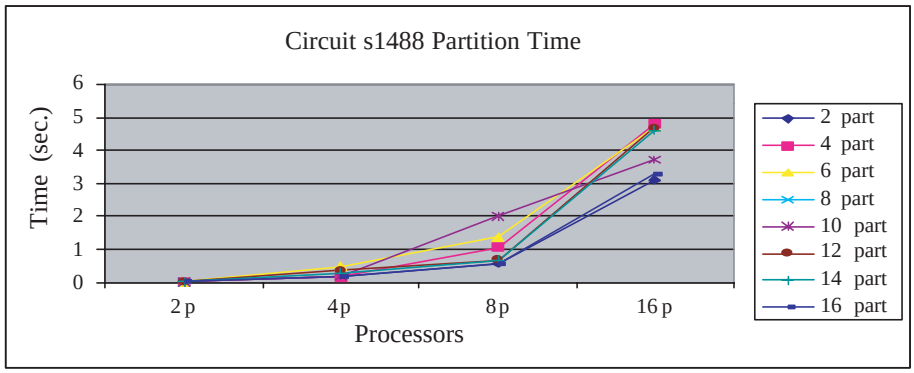

Figure 5 The time consume for partitioning circuit s1488 into 2-16 partitions VS. the number of processors.

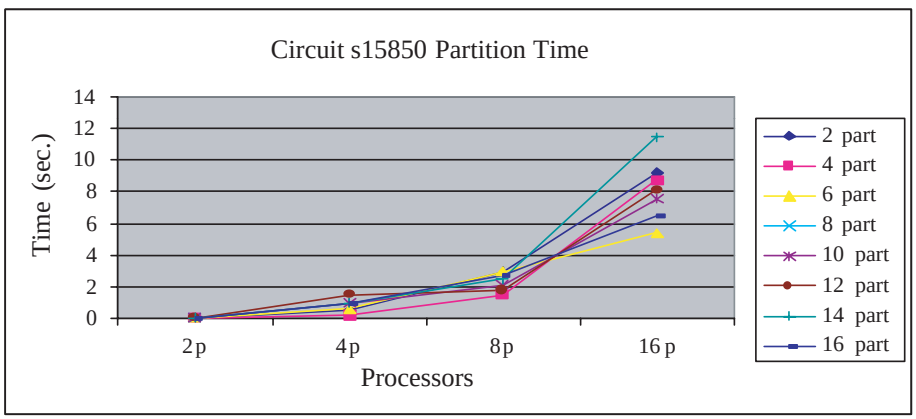

Figure 6 The time consume for partitioning circuit s15850 into 2-16 partitions VS. the number of processors.

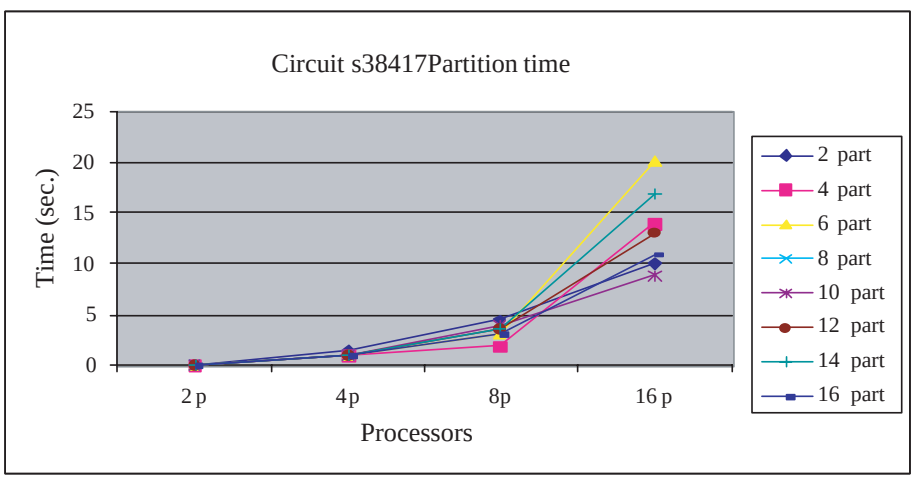

Figure 7 The time consume for partitioning circuit s38417 into 2-16 partitions VS. the number of processors.

Table 1 Synthesis runtime VS. number of processors (sec.).

Circuit	1P	2P	4P	8P	16P
S1488	52	38	42	56	76
S15850	793	435	138	111	96
S38417	953	492	314	167	125
Total	1789	965	494	334	297

Table 2 An edge-cut VS. number of processors.

Circuit	2P	4P	8P	16P
S1488	1	185	274	345
S15850	35	56	103	217
S38417	40	106	166	274

the longest processing time, the more processors we have the longer delay time the Network-File-System takes to load the sub netlist into memory. In addition, merge time is dependent on the number of circuits to merge.

CONCLUSION

We have proposed an alternative method to synthesis a sequential logic circuit which reduces time consuming in synthesis process. The LGSynth'91 benchmark suite used for experiment is in net-list format. Our result shows that the number of partition is increasing where as the synthesis time is reduced as the number of processor is increased up to 16 processors. The number of edge-cut is increased as the number of processor is increased. The speed up on each circuit is 0.6, 8.2 and 7.7 respectively. Future work, we will try on other parallel optimization which may be improved performance and also we will try to test more examples.

LITERATURE CITED

Bryant, R. 1986. Graph-based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers,C-35(8) : 667-691.

Coudert, O. C. Berthet, and J.C. Madre, 1989. Verification of Sequential Machines Based on Symbolic Execution, pp. 365-373. *In* Proc. of the Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France.

DeMicheli. G. 1991. Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization. IEEE Transactions on Computer-Aided Design, 10(1) : 63-73.

Ghosh. A. 1991 Techniques for Test Generation and Verification in VLSI Sequential Circuits. UCB PhD Thesis, University of California, Berkeley.

Hachtel, G.D. J.K. Rho, F. Somenzi, and R. Jacoby. 1991. Exact and heuristic algorithms for the minimization of incompletely specified state machines, pp. 184–191. *In* The Proceedings of the European Conference on Design Automation.

Hendrickson, B. and R. Leland. 1993. A multilevel

algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories. 14 p.

Karypis, G., K. Schloegel, and V. Kumar, 1998. ParMeTis Parallel Graph Partitioning and Sparse Matrix Ordering Library Version 2.0. University of Minnesota, Department of Computer Science / Army HPC Research Center Minneapolis.

Karypis, G., R. Aggarwal, V. Kumar, and S. Shekhar. 1999. Multilevel Hypergraph Partitioning : Applications in VLSI Domain. pp. 69-79. IEEE Trans. VLSI System, vol. 7.

Karypis, G. and V. Kumar. 1996. Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs. University of Minnesota, Department of Computer Science / Army HPC Research Center Minneapolis. 23p.

Karypis, G. and V. Kumar, 1997. A Coarse-Grain Parallel Formulation of a Multilevel k-way Graphs Partitioning Algorithm. University of Minnesota, Department of Computer Science / Army HPC Research Center Minneapolis. 12p.

Lavagno, L., K. Keutzer, and A. sangiovanni-Vincentelli. 1991. Algorithms for Synthesis of hazard-free asynchronous circuits, pp. 302–308. *In* Proceedings of the Design Automation Conference.

Leiserson, C. E., F. M. Rose, and J. B. Saxe. 1983. Optimizing synchronous circuitry by retiming, pp. 86–116. *In* Advanced Research in VLSI: Proceedings of the Third Caltech Conference.

Lin, B. and A.R. Newton. 1989. Synthesis of multiple level logic from symbolic high-level description languages, pp. 187–196. *In* Proceedings of the International Conference on VLSI.

Malik, S., E.M. Sentovich, R.K. Brayton, and A. Sangiovanni-Vincentelli, 1991. Retiming and resynthesis: Optimization of sequential networks with combinational techniques. IEEE Transactions on Computer-Aided Design 10(1). pp. 74–84.

Moon, C.W., B. Lin, H. Savoj, and R.K. Brayton. 1989. Technology mapping for sequential logic synthesis, pp. 63–73. *In* Proc.Int'l. Workshop on Logic Synthesis, North Carolina.

Saeyand. Y. 1991. Logic Synthesis and Optimization Benchmarks User Guide Version 3.0. Microelectronics Center of North Carolina, P.O. Box 12889, Research Triangle Park.

Sentovich, E. M., K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A Saldanha, H. Savoj, P.R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli 1992. SIS : A System for Sequential Circuit Synthesis. Department of Electrical Engineering and Computer Science University of California, Berkeley. 45 p.

Uthayopas, P., Sanguanpong, S., and Poovarawan, Y. 2000. Building a large scale internet superserver for academic services with Linux Cluster Technology, pp. 65-71. *In* International Workshop on Asia Pacific Advanced Network and Its Application (IWS-2000), Tsukuba.

Villa T. and A. Sangiovanni Vincentelli, 1990. NOVA: State Assignment of Finite State Machines for Optimal Two Level Logic Implementations,. *IEEE Transactions on Computer-Aided Design*, 9(9) : 905–924.

Received date : 28/01/02
Accepted date : 30/09/02