High-Level Synthesis Under Multi-Cycle Interconnect Delay

Jinhwan Jeon, Daehong Kim, Dongwan Shin*, and Kiyoung Choi

School of EECS, Seoul Nat'l University, Seoul 151-742, Korea

*Dept. of ICS, University of California, Irvine, CA 92697-3425, USA

{jeonjinh, daehong, kchoi}@poppy.snu.ac.kr *dongwans@ics.uci.edu

Abstract - As process technology goes into deep sub-micron range, interconnect delay becomes dominant among overall system delay, occupying most of the system clock cycle time. Interconnect delay is now a crucial factor that needs to be considered even during high-level synthesis. In this paper, we propose a concurrent scheduling and binding algorithm that takes interconnect delay into account. We first define our distributed target architecture, which minimizes the effect of interconnect delay on clock cycle time. We no longer assume that interconnect delay between functional units is a part of one clock cycle. Interconnect delay can span over multiple clock cycles. We incorporate the concept of multi-cycle interconnect delay into scheduling and binding process, to reduce the critical path length and therefore the system latency. We show that by introducing interconnect delay, we can obtain latency improvement of up to 54 % and of 37% on the average.

I. Introduction

A number of researchers have independently reported that interconnect delay dominates logic delay in current deep sub-micron (DSM) process technology [1, 2]. In deeper sub-micron geometry in the future, which will be still with the continuous scaling of process technology and thereby generate longer interconnect delay due to RC delay, coupling noise, inductance, etc. [3, 4], the situation will be worse and worse. The interconnect delay is expected to occupy 80% or more of the overall critical path delay in the near future [5].

In conventional target architecture based on centralized register files, functional units (FU's) read their operands from centralized register files (and write their results to centralized register files) through relatively long buses [6]. Fig. 1 shows a simple centralized architecture model. In such architecture, total clock cycle time is bounded by

$$T_{CYCLE} = T_{LOGIC} + T_{GLOBAL} + T_{SETUP} + T_{REGISTER\text{-}DELAY}$$
$$T_{GLOBAL} = T_{R2FU} + T_{FU2R}$$

where T_{LOGIC}, T_{SETUP}, $T_{REGISTER\text{-}DELAY}$, T_{R2FU} and T_{FU2R} represent FU delay, register setup time, clock-to-register-output delay, FU-to-register delay and register-to-FU delay, respectively. T_{GLOBAL} represents the sum of interconnect delay between register files and functional units through relatively long global buses.

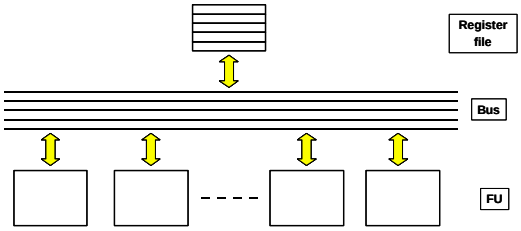

Fig. 1. Centralized register based architecture.

In current DSM technology, T_{GLOBAL} has become comparable to the rest of T_{CYCLE}. Therefore, we usually insert latches at both inputs and output of a FU to reduce the cycle time, which results in complex two-phase clocking scheme. In this case the cycle time is computed as $T_{CYCLE} = \max(T_{LOGIC}+T_{SETUP}+T_{LATCH\text{-}DELAY}, T_{GLOBAL})$. In deeper sub-micron technology, the case will be even worse because increased T_{GLOBAL} will dominate T_{CYCLE}. In that case, T_{GLOBAL} can be three or four times longer than T_{LOGIC} and existing complex clocking scheme, including two-phase clocking, will be inefficient.

In this paper, we first define our target architecture based on distributed memory architecture. The architecture minimizes the effect of interconnect delay on clock cycle time. In the target architecture, we do not regard the global data transfers among functional units to be completed within one clock cycle time. Instead, we allow global data transfers to be carried out over multiple clock cycles. We incorporate such concept of *multi-cycle interconnect delay* into scheduling and binding process, which is the extension of Continuous Time List Scheduling proposed in [8], in order to reduce the critical path length and therefore the system latency during high-level synthesis.

We will focus on adaptation of *multi-cycle interconnect delay* to scheduling and binding process to improve overall

system latency. Our target architecture and overall design flow are proposed in Section II. In Section III, we illustrate motivational example by considering the effect of multi-cycling on overall system latency and the effect of operation binding on critical path when interconnect delays, comparable to or longer than FU delays, are incorporated into scheduling and binding procedure. Performance driven scheduling algorithm with interconnect delay and its comparisons with those not considering interconnect delay are presented in Section IV and Section V. Finally conclusions and future work are given in Section VI.

II. Target Architecture and Design Flow

In this section, we propose a distributed register-based target architecture which can manage *multi-cycle interconnect delay* and compare the proposed architecture with conventional one. We also explain overall design flow for high-level synthesis under the target architecture.

In our target architecture, each FU performs computation by using the data stored in dedicated local registers (or possibly latches) and the global communication with other FU's can be carried out as a data transfer between local registers(or latches) of the source FU and the input registers(or latches) of the destination FU, respectively. Therefore, global communication delay among FU's can be separated in the computation of clock cycle time in the distributed architecture. Recall that $\mathbf{T_{GLOBAL}}$ is incorporated into $\mathbf{T_{CYCLE}}$ in the centralized architecture. In Fig. 2, a simple model of the target architecture is shown. In this architecture, we define two terms - Intra-FU cycle time ($\mathbf{T_{INTRA}}$) for the computation and Inter-FU cycle time ($\mathbf{T_{INTER}}$) for the communication – that are defined as

$$\mathbf{T_{INTRA} = T_{LOGIC} + T_{R2FU} + T_{FU2R} + T_{SETUP} + T_{REGISTER\text{-}DELAY}}$$

$$\mathbf{T_{inter} = T_{SETUP} + T_{R2R} + T_{REGISTER\text{-}DELAY}}$$

where $\mathbf{T_{R2R}}$ is the interconnect delay between dedicated registers through global buses. Note that both $\mathbf{T_{R2FU}}$ and $\mathbf{T_{FU2R}}$ are almost constant and have negligible values due to the property of distributed architecture. In DSM, $\mathbf{T_{INTER}}$ can be equal to or longer than $\mathbf{T_{INTRA}}$ due to dominating interconnect delay $\mathbf{T_{R2R}}$.

Assume that we appropriately assign operations and variables to FU's and their dedicated registers, respectively, in such a way to reduce the number of inter-FU communications. In this case, the system clock is determined by $\mathbf{T_{INTER}}$ because $\mathbf{T_{INTER}}$ dominates $\mathbf{T_{INTRA}}$ which has relatively small value. Therefore, to minimize overall system latency, the system should be able to treat the communication among FU's, with relatively long interconnect delay, as a multi-cycle operation chained from the output of a source FU to the input of a destination FU.

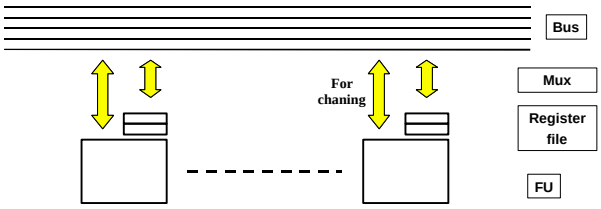

Fig. 2. Our target architecture.

Although centralized architecture is simple, it is inefficient in DSM because of long cycle time. The distributed one needs complex FU/register binding algorithm compared to centralized one. It also requires more registers in general. However, it is efficient in DSM if we manage *multi-cycle interconnect delay.*

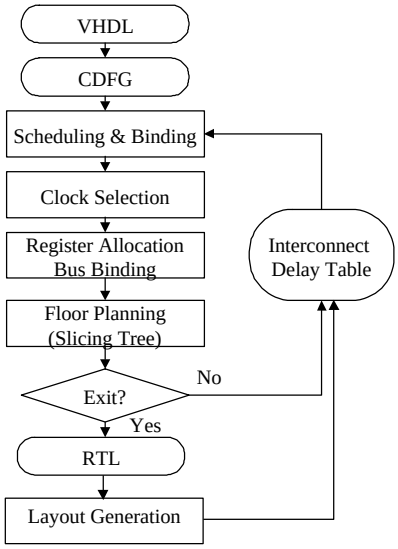

Fig. 3. Overall design flow.

Fig. 3 illustrates an overall design flow of our high level synthesis system. We iterate synthesis loop until exit condition is satisfied. Within loop, we perform scheduling, binding, clock selection, register allocation, and bus binding based on the interconnect delay information obtained in the previous iteration. At the first iteration where we do not have interconnect delay information, we set interconnect delay to zero. To estimate interconnect delay among functional units, we perform floor planning by using the slicing tree algorithm [14]. The exit condition to terminate the synthesis loop is when there is no more change in the floor planning or the iteration limit is reached. When the loop terminates by iteration limit (that is, no convergence during the synthesis loop), we return

the best solution obtained so far. We also support feedback path from the layout generation step, which performs re-synthesis using the delay information obtained from real layout.

III. Motivation

In this section, we explain the concept of *multi-cycle interconnect delay* in more detail using a simple example. It can be seen that overall system latency is improved by allowing multi-cycle global data transfer between FU's. For the multi-cycling of global data transfers, we treat a bus with multi-cycle delay like a FU with multi-cycle delay. Moreover, we explain the need for specific operation binding algorithm by showing the effect of operation binding on critical path length.

Fig. 4 (a) represents a simple example that consists of two additions and one multiplication. Multiplier1 accepts the result from the adder1 to perform its multiplication operation. Assuming that interconnect delay from adder1 to multiplier1 is 20ns, which is longer than the execution time of adder1, scheduling result of Fig. 4(b) is obtained. If we apply path-based scheduling [15] to find the schedule with the minimum number of control-steps and assume the register-delay is 3ns, we get a clock cycle of 46ns (40+3+3) and then the system latency of 92ns (46*2). Note that in this case, interconnect delay is contained within a clock cycle. On the contrary, if we set a clock cycle time as 16ns (10+3+3) through our novel scheduling and binding algorithm considering interconnect delay, we achieve the latency of 64ns (16*4) as shown in Fig. 4 (c). Fig. 4(d) is the target architecture based on the scheduling result of Fig. 4 (c), where the data transfer done is across two effective clock cycles. We call this delay *multi-cycle interconnect delay* and treat a bus with multi-cycle delay like a FU with multi-cycle delay. That is, we treat a global data transfer as an operation taking one or more clock cycles.

The multi-cycling of data transfers can reduce the system latency in two ways. First, it enables us to select smaller value of clock cycle time, which reduces the waste of clock slack induced by the difference between computation time and cycle time, thereby minimizing the system latency. Secondly, multi-cycling enables parallel execution of data transfers and functional units. While data is transferred on a global bus, we can initiate a new operation in a functional unit or we can initiate another global data transfer through a different bus, which is impossible if the concept of multi-cycle interconnect delay is not adopted. Recall that, in the conventional architecture, interconnect delay is merged with a FU delay into a clock cycle and therefore it is not allowed to initiate a new operation or data transfer.

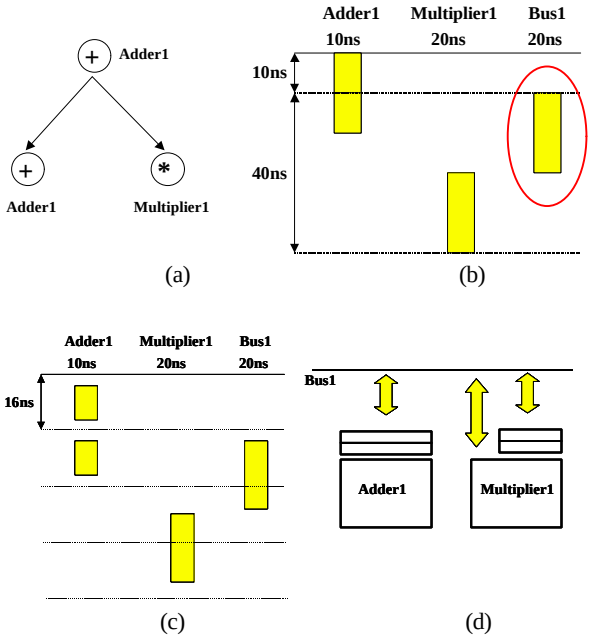

Fig. 4. Multi-cycle interconnect delay and optimal clock selection.

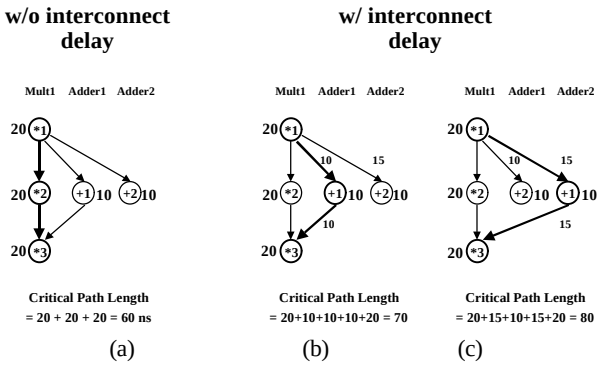

Fig. 5. Effect of operation binding on critical path when interconnect delay is considered.

For minimizing latency, we should perform careful scheduling and binding because critical path length strongly depends on the binding state when interconnect delay is considered. Figure 5 illustrates the case where critical path varies according to operation binding. Without considering interconnect delay the path *1 →*2 →*3 is the critical path with 60 ns delay. Let us introduce interconnect delay (10 ns between Mult1 and Adder1 and 15 ns between Mult1 and Adder2) to the same example. In this case, the critical path in Fig. 5 (a) is no longer critical. In Fig. 5 (b) and (c), the *1 → +1 →*3 becomes a new critical path. However, the critical path length varies from 70 to 80 ns depending on operation binding. Therefore, we need to carefully perform scheduling

and binding under the condition that interconnect delay dominates system latency.

IV. Performance Driven Scheduling with Interconnect Delay

Our scheduling and binding algorithm is based on performance driven scheduling (PDS) algorithm proposed in [8]. The PDS algorithm maximizes performance by the combination of Continuous Time List Scheduling and optimal clock selection [7]. To consider the effect of long interconnect delay, we extend PDS scheduling algorithm, which we call performance driven scheduling with interconnect delay (PDSI). The PDSI algorithm performs simultaneous scheduling and binding such that the overall system latency is minimized while considering interconnect delay. The algorithm treats a global data transfer among functional units as an operation chained between the source and the destination functional units (or registers). Therefore, we can allow a data transfer with long delay to behave like a multi-cycle functional operation. The multi-cycling of data transfer enables both data transfer and other functional units to execute in parallel thereby reducing total latency. The algorithm binds operations such that true critical path length is minimized as shown in Fig. 5 (b).

Before we perform PDSI algorithm, we assume preliminary floorplanning information is available. From the floorplanning information, we obtain interconnect delay among functional units and distributed register files. We denote $d_{i,j}$ as the global interconnect delay between functional unit f_i and f_j.

Table 1. Interconnect delay for the example of Fig. 5

$d_{i,j}$	Add1	Add2	Mult1
Add1	0	5	10
Add2	5	0	15
Mult1	10	15	0

Table 2. Critical path length for the example of Fig. 5

$cpl_{i,j}$	Add1	Add2	Mult1
*1	-	-	70
*2	-	-	40
*3	-	-	20
+1	40	45	-
+2	10	10	-

The PDSI algorithm first performs scheduling and binding in continuous time domain without fixed clock cycle time. Then we determine clock cycle time by using optimal clock selection algorithm proposed in [7] for minimizing total latency. Like conventional list scheduling, our algorithm selects operations among ready operations in descending order of path length. However, the path length strongly depends on operation binding when interconnect delay is introduced for the data transfer among functional units, as shown in Fig. 5. To consider the effect of operation binding on critical path and to handle the interconnect delay that depends on operation binding, we use the concept of dynamic critical path scheduling algorithm [10].

We define $cpl_{i,j}$ and $est_{i,j}$ as the critical path length from an operation node n_i to the end node and the earliest schedulable time of n_i, respectively when n_i is bound to a functional unit f_j. The critical path length is computed as

$$cpl_{i,j} = c_j + \max_{n_k \in succ_i} (\min_l (d_{j,l} + cpl_{k,l})) \quad (1)$$

where $succ_i$ and c_j represents the set of successors of n_i and the computation time of f_j, respectively. The critical path length is recursively computed from the end node to the start node in the graph. Table 1 and 2 show the interconnect delay and the computed critical path length for the example of Fig. 5, respectively. In the PDSI algorithm, we use $\min_j(cpl_{i,j})$ as the priority function for the selection of an operation. Among ready operations, we select the one with highest priority. After selecting an operation, we select a functional unit f_j to which the operation will be bound such that estimated latency is minimized. The estimated latency $dcpl_{i,j}$ is computed as

$$dcpl_{i,j} = est_{i,j} + cpl_{i,j} \quad (2)$$

Fig. 6 explains the procedure of our scheduling and binding algorithm.

Step	Ready operations	Priority	Selected operation	Estimated latency (M1, A1, A2)	Binding
1	*1	**70**	*1	**70**, N/A, N/A	*1 → M1
2	*2, +1, +2	40, **45**, 10	+1	N/A **70**, 75	+1 → A1
3	*2, +2	**40**, 10	*2	**70**, N/A, N/A	*2 → M1
4	+2, *3	10, **20**	*3	**70**, N/A, N/A	*3 → M1
5	+2	10	+2	N/A, 60, **55**	+2 → A2

Fig. 6. Scheduling and binding procedure.

After scheduling and binding in continuous time domain, we map a set of chained operations to control steps (i.e. discrete time domain) according to resource conflict condition explained in [11]. If there is a resource conflict for an operation, then we start a new control step at the earliest schedule time of the operation. Fig. 7 (a) and (b) illustrate the results of continuous time scheduling and control step mapping respectively for the example in Fig. 5. Note that data transfers *1 → +1 and *1 → +2 are overlapped with the computation of *2 to minimize total latency. After the mapping procedure,

each control step may have different delay. To minimize total latency by reducing clock slacks, we allow each control step to extend for more than one clock cycles. For the lack of space, we do not explain detailed algorithm for determining optimal clock cycle time. The detailed algorithm is proposed in [7]. Fig. 7 (c) illustrates the final scheduling and clock selection result by the PDSI algorithm. Note that each control step requires different number of clock cycles for the minimum latency.

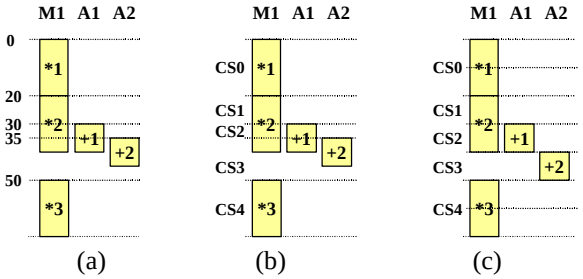

Fig. 7. Result of PDSI for the test example.

V. Experimental Results

To evaluate the proposed scheme, we performed experiments with well-known data-dominated examples from the HYPER system [13]. We assumed the delays of an adder and that of a multiplier are 4 ns and 40 ns, respectively. We assumed that $T_{SETUP}+T_{LATCH}+T_{FU2R}+T_{R2FU}$ is 2ns. To compute delays among functional units, we generated floor plan from the synthesis result obtained in the previous iteration of the synthesis loop. We limited the maximum number of iterations by 10. However, the synthesis loop terminates after 3 iterations in most cases due to the exit condition on floor planning. To show the effectiveness of the proposed algorithm, we compared three different methods running on the same examples. The first one, which is represented as LS, is the conventional list scheduling algorithm with clock optimization scheme that does not allow multi-cycling of global data transfers. The second one, which represented as Resyn, is the re-synthesis algorithm that performs re-scheduling and clock optimization as post processing after interconnect delay is added to the computation time of each operation [9]. The Resyn uses PDS[8] algorithm for minimizing latency. The last one is the proposed PDSI algorithm. For fair comparison, we assumed the same distributed-register architecture for the three different methods.

Table 3 compares the results obtained by the three different algorithms. The table shows performance improvement of up to 54% (37% on the average) over LS and improvement of up to 47% (27% on the average) over Resyn. Such significant improvement was obtained by multi-cycling of data transfers and considering the effect of interconnect delay on critical path length. Multi-cycling of data transfers enables parallel execution of functional units and data transfers thereby minimizing total latency. It also reduces slack within a clock cycle and the computation delay of a functional unit. Moreover, by considering the effect of interconnect delay during scheduling, we can find a solution with shorter latency. To see the effect of interconnect delay on the latency improvement, we experimented under different values of interconnect delay. We multiplied $d_{i,j}$ by a factor α. We experimented for three values of α: 1.0, 1.5 and 2.0. The results in the table show significant improvement in latency as interconnect delay dominates total delay. Unlike LS and Resyn, the PDSI does not degrade performance as interconnect delay increases.

The table also compares the number of buses and the number of states for three methods. Since multi-cycling of data transfers increases the number of concurrent data transfers at each control step, our method requires more buses than LS. However, the overhead is not significant compared to substantial performance improvement. Moreover, the increase in the number of buses not always results in the increase in area since the increased buses can simplify multiplexing units and localize data transfers on a bus to a smaller set of functional units thereby reducing bus length [12]. The number of states also increases because multi-cycling causes additional state in the controller. However the increase in controller area can be complemented by substantial performance improvement. The table shows trade off between the number of states and the number of buses. We obtain reduction in the number of global buses because the increased number of states divides life time of a variable into shorter fragments thereby reducing the overlap of life times of variables.

VI. Conclusions and Future Work

As the complexity of design increases and process technology runs into deep sub-micron (DSM), interconnect delay has become major bottleneck in performance. In this paper, we proposed an effective high-level synthesis algorithm considering interconnect delay during scheduling and binding to minimize overall latency. The proposed algorithm allows multi-cycling of data transfers that enables parallel execution of data transfers and functional units and minimizes clock slack, based on the target architecture. Moreover, the algorithm considers the effect of interconnect delay on critical path length, which dynamically varies depending on operation

Table 3. Comparison of synthesis results

	Resource	α	LS				Resyn				PDSI				Improvement	
			No. of states	Cycle time (ns)	No. of buses	Latency (ns)	No. of states	Cycle time	No. of buses	Latency (ns)	No. of states	Cycle time (ns)	No. of buses	Latency (ns)	PDSI / Resyn (%)	PDSI / LS (%)
fir11	+1 *3	1.0	12	6.8	2	243	16	5.4	2	203	12	6.3	3	189	**6.9**	**22.2**
		1.5	12	6.2	2	262	13	7.9	2	206	12	6.2	3	185	**10.2**	**29.4**
		2.0	12	6.6	2	276	20	5	2	211	16	6	3	187	**11.4**	**32.2**
iir7	+2 *3	1.0	14	8.1	3	317	17	6.7	3	281	15	6.1	3	227	**19.4**	**28.4**
		1.5	14	8.7	3	339	22	5.2	2	295	15	6.3	3	232	**21.4**	**31.6**
		2.0	14	9.3	3	407	19	7	3	295	17	5.2	3	230	**22.0**	**43.5**
dct	+2 *3	1.0	18	7.1	3	321	23	6.5	2	268	19	7.2	4	183	**31.7**	**43.0**
		1.5	18	6.5	3	316	18	6.2	4	285	23	6	4	180	**36.8**	**43.0**
		2.0	18	7.8	3	375	23	6.5	2	327	22	6.1	4	181	**44.6**	**51.7**
wavelet	+2 *4	1.0	26	13.3	4	517	34	7.1	2	427	21	6.2	5	317	**25.8**	**38.7**
		1.5	26	8.3	4	535	46	5.6	2	472	21	6.2	4	317	**32.8**	**40.7**
		2.0	26	21.4	4	683	41	5.5	2	595	25	6	4	312	**47.6**	**54.3**
nc	+2 *4	1.0	26	6.1	3	533	28	6.1	3	457	27	6.1	3	385	**15.8**	**27.8**
		1.5	26	5.1	3	573	32	9.1	3	539	27	6.1	3	383	**28.9**	**33.2**
		2.0	26	9.1	3	713	34	6.1	3	539	27	6.1	3	385	**29.3**	**46.6**
parallel	+2 *5	1.0	14	17.6	3	300	18	5.5	2	255	15	6.3	3	171	**32.9**	**43.0**
		1.5	14	11.7	3	327	20	5.6	2	281	18	7.2	3	179	**36.3**	**45.3**
		2.0	14	7.1	3	376	19	5.5	2	322	17	7.3	4	182	**43.5**	**51.6**
Average															**27.63**	**37.17**

binding, during scheduling and binding for minimum latency. Experimental results show performance improvement of up to 54 % over conventional method with slight area overhead due to increased number of buses and number of controller states.

We are currently working on devising an efficient floorplan algorithm, which is required to produce the initial yet quite accurate estimation of interconnect delay. We are also trying to reveal the trade-off relationship between the number of states and performance.

References

[1] J. M. Rabaey, *Digital Integrated Circuits, A Design Perspective*, Englewood Cliffs, NJ: Prentice-Hall, 1996.

[2] E. G. Friedman, "Substrate Coupling and Interconnect Noise in Mixed-Signal and High Speed Digital ICs," *IEEE CAS Workshop on Mixed-Signal Integrated Circuit Design*, Dec., 1999.

[3] D. Sylvester and K. Keutzer, "Getting to the bottom of deep submicron," *Proc. of International Conference on Computer-Aided Design*, pp. 203-211, 1998.

[4] Y. I. Ismail, E. G. Friedman, and J. L. Neves, "Figures of Merit to Characterize the Importance of On-Chip Inductance," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 4, Dec., 1999.

[5] *International Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 1999.

[6] *High Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992

[7] S. Park and K. Choi, "Latency minimisation by system clock optimisation," *IEE Electronics Letters*, vol. 34, pp. 862-864, 1998.

[8] S. Park and K. Choi, "Performance-Driven Scheduling with Bit-Level Chaining," *Proceedings of Design Automation Conference*, pp. 286-291, 1999.

[9] S. Park, K. Kim, H. Chang, J. Jeon, and K. Choi, "Backward-Annotation of Post Layout Delay Information into High-Level Synthesis Process for Performance Optimization," *Proc. of 6th International Conference on VLSI and CAD*, pp. 25-28, Oct. 1999.

[10] Y. Kwok and I. Ahmad, "Dynamic Critical-Path Schedule: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, May 1996.

[11] R. Camposano, "Path-Based Scheduling for Synthesis," *IEEE Transactions on Computer Aided Design*, pp. 154-157, 1996.

[12] S. Tarafdar, M. Leeser, and Z. Yin, "Integrating Floorplanning In Data-Transfer Based High-Level Synthesis," *Proceedings of International Conference on Computer Aided Design*, pp. 412-417, 1998.

[13] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures," *IEEE Design and Test of Computers*, pp. 40-51, 1991.

[14] M. McFarland, "A Fast Floor Planning Algorithm for Architectureal Evaluation," *Proceedings of the International Conference on Computer Design*, pp. 96-99, Oct. 1989.

[15] R. Camposano, "Path-Based Scheduling for Synthesis," *IEEE Transactions on CAD*, vol. 10, no. 1, pp. 85-93, Jan. 1991.