Distributed Synchronous Control Units for Dataflow Graphs under Allocation of Telescopic Arithmetic Units

Euiseok Kim[†, ‡] Hiroshi Saito[†] Jeong-Gun Lee[‡]
Dong-Ik Lee[‡] Hiroshi Nakamura[†] Takashi Nanya[†]

Dependable and High-Performance Computing Laboratory, RCAST, Univ. of Tokyo[†]
Ultra-Fast Fiber Optic Networks Research Center, Dept. of Info. and Comm., K-JIST[‡]
E-mail: {**eskim**, hiroshi, nakamura, nanya}@hal.rcast.u-tokyo.ac.jp & {eulia, dilee}@kjist.ac.kr

Abstract

In order to enjoy performance improvement effects of variable computation time arithmetic units in a system level, we propose a new synchronous control unit design methodology for dataflow graphs under allocation of a telescopic arithmetic unit which is one of well-known synchronous variable computation time arithmetic units. The proposed method generates an independent synchronous controller for each component arithmetic unit, and builds a distributed synchronous control unit through integrating derived controllers. The distributed structure of a final synchronous control unit maximizes performance improvement effect of telescopic arithmetic units through a complete preservation of original concurrency among operations.

1. Introduction

With the increase of interests in high-performance system design, variable computation time arithmetic units, in short VCAUs, have started to be used for implementing a variable delay datapath of a target system in pursuit of performance enhancement[1, 2, 3]. However, adoption of VCAUs requires modification of a traditional synchronous control unit design methodology since existing ones are based on the assumption that all arithmetic units of a datapath operate with their own fixed computation time.

[1, 2] proposed a telescopic arithmetic unit, in short TAU, which is one of representative synchronous VCAUs, and presented a method to build a centralized control unit for a datapath including TAUs. Although the method in [1, 2] achieved noticeable and pioneering results, it may cause some problems such as system performance degradation with the increase of the number of TAUs used in a target system. In [3], a synchronous independent controller is built for each operation, and a global control unit is realized as a set of those independent small controllers. Although the method can guarantee that resulting control units are able to preserve original concurrency among operations, resulting control units may suffer from a rapid area increase with the increase of the number of operations in system specifications.

In this paper, we propose a new synchronous control unit design method for dataflow graphs, in short DFGs, under allocation of TAUs. The proposed method generates an independent synchronous controller for each component arithmetic unit, and builds a distributed synchronous control unit through integrating derived controllers. Each independent synchronous controller activates allocated operations sequentially once its input operands and allocated arithmetic unit become available with a new clock occurrence. Therefore, the idle time of each component arithmetic unit can be minimized and the original concurrency among operations is preserved completely.

The organization of this paper is as follows; Section 2 introduces previous work and related problems. In Section 3, a new scheduling method, which is for building a distributed synchronous control unit, will be explained. Section 4 explains how to generate a set of FSMs which will be synthesized into a final distributed synchronous control unit. Section 5 and Section 6 present experimental results, discussions and future work.

2. Previous Work

2.1. Telescopic arithmetic units

A TAU[1, 2] consists of the following two parts, an arithmetic logic part and a completion signal generator as shown in Fig. 1. The arithmetic logic part of a TAU is exactly the same as general synchronous arithmetic logics. The

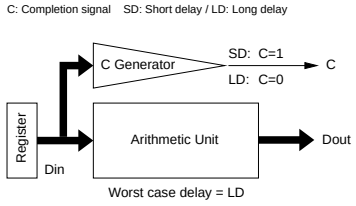


Figure 1. A structure of a TAU

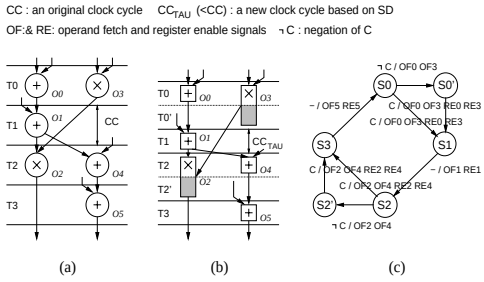


Figure 2. (a) An original DFG (b) A TAUBM DFG (c) A corresponding TAUBM FSM

completion signal generator, which is a distinctive part of VCAUs, generates a completion signal when it decides that computation for input operands is over. For an explanation, we define two variables LD(Long Delay) and SD(Short Delay). Actually, LD corresponds to the worst case delay of the arithmetic logic part. Note that the real computation time varies according to input operands although traditional synchronous system designers assume the worst and fixed computation time. Therefore, we can divide whole input operands into two groups; the first group is the set of input operands requiring computation time not larger than SD, and the second group is the set of remaining input operands not belonging to the first group. Intuitively speaking, a completion signal generator is the set of input operands belonging to the first group. Therefore, it produces '1' for input operands which can be computed within SD, and thus we can decide whether the corresponding computation is over within SD or not by checking the value of the completion signal generated from the completion signal generator [1]. An automatic generation method of the completion signal generator was also developed in [1, 2]. Note that our consideration is restricted to TAUs just for convenience of explanation. That is, the proposed method in this paper can be applied to other kinds of synchronous VCAUs in the same manner.

Contrary to synchronous system designs, VCAUs are very general in asynchronous system designs, and there exist well-known VCAUs such as dual-rail arithmetic units[4] and speculative arithmetic units[5]. However, they generate their computation completion signals in continuous time domain and thus require special asynchronous control circuits[6, 7, 8].

2.2. A synchronous control unit generation method based on TAUs

In addition to developments of TAUs, [1, 2] developed a method to modify an original FSM into a new FSM which is able to control a variable delay datapath including TAUs, and we call it TAU based methodology, in short TAUBM. Although the original TAUBM considers direct modification of an original FSM, we introduce a TAUBM DFG as an intermediate model in a control unit building procedure because the main goal of this paper is to derive a distributed synchronous control unit for DFGs. Actually, the derivation of a TAUBM DFG is trivial as follows; [step 1] If a time step $\mathrm{T_i}$ in an original DFG includes operations bound to TAUs, divide it into $\mathrm{T_i}$ and $\mathrm{T_i'}$. [step 2] For operations in a time step $\mathrm{T_i}$ of an original DFG, schedule operations bound to TAUs in both $\mathrm{T_i}$ and $\mathrm{T_i'}$, and allocate remaining operations into $\mathrm{T_i}$. Note that operations which are scheduled in $\mathrm{T_i}$ and $\mathrm{T_i'}$ can be performed within $\mathrm{T_i}$ or within $\mathrm{T_i'}$ according to their input operands. Fig. 2 (a) and (b) show an original DFG and the corresponding TAUBM DFG when we assume that a multiplier is implemented in a TAU. Gray colored boxes in Fig. 2(b) represent that corresponding time steps are not spent for input operands requiring SD. Therefore, other operations are not scheduled to time steps corresponding to gray colored boxes. A typical synchronous controller for a DFG generates predetermined control signals necessary for performing operations in each time step, and thus its corresponding FSM has a simple counter like structure. However, a TAUBM FSM receives completion signals from TAUs additionally, and select different state transitions according to their values. Fig. 2(c) shows a TAUBM FSM derived for a TAUBM DFG in Fig. 2(b). Note that states $\mathrm{S_0}$ and $\mathrm{S_2}$ have choices according to the value of a completion signal 'C', and thus a resulting system latency varies between 4 and 6 clock cycles. A typical FSM does not have $\mathrm{S_0'}$, $\mathrm{S_2'}$ and 'C'. For a detailed explanation about a TAUBM FSM derivation procedure, please refer to [1, 2].

2.3. Problems of previous work

A TAU and TAUBM proposed in [1, 2] are pioneering work which introduces the concept of VCAUs in a synchronous system design, and their method improved the performance of a synthesized target system largely. However, the method, TAUBM, performs a simple modification of an original FSM on a reduced new clock cycle. Therefore,

[1] In [1], the completion signal '1' is generated for input operands which can not be processed within SD. However, in this paper, we take an opposite way for convenience.

TAUBM may not manage complex behavior of a datapath if the number of TAUs increases. In this subsection, we point out two problems that TAUBM causes with the increase of the number of allocated TAUs. Solving those two problems are important goals of this paper.

The distinctive feature of a TAU is for it to spend one or two clock cycles selectively according to input operands. Therefore, if the ratio of input operands requiring SD, denoted in 'P', is small, performance of a resulting TAUBM FSM may get worse because of the increased number of clock cycles. With this feature of a TAU, the first problem of TAUBM is that the possibility of spending one additional clock cycle increases as more TAUs are used. For example, if only one TAU is allocated to the time step T_i, the probability that one more clock cycle is spent is 1-P. However, if n TAUs are allocated to the time T_i, the probability that one more clock cycle is spent becomes 1-P^n. Consequently, performance of a resulting control unit gets worse. In pursuit of high performance system, more and more TAUs will be used, and hence this problem becomes serious.

The second problem is that a TAUBM FSM may not preserve original concurrency among operations. This feature originates from the fact that all the operations assigned to the same time step are synchronized. For example, in a TAUBM DFG of Fig. 2(b), operation O_1 can be performed once operation O_0 is over. However, operations O_0 and O_3 are synchronized, and thus operation O_1 cannot start until operation O_3 is over although operations O_1 and O_3 can be performed concurrently. Note here that this synchronization increases idle time of arithmetic units and leads to degradation of system performance.

3. A New Scheduling Method under TAU Allocation

Synchronous scheduling is a procedure to allocate each operation in a DFG to a specific time step which is determined according to the global clock cycle. Therefore, a control unit starts and finishes each operation at predetermined time steps. This feature of general synchronous scheduling is due to the basic assumption that all the operations have fixed computation time. However, with the adoption of VCAUs such as TAUs, that basic assumption of synchronous scheduling is not true any more. For example, for a TAUBM DFG in Fig. 3(a), if we assume that two TAU multipliers and two general adders are allocated, a time step T_0' is reserved for operations O_0 and O_6 and thus operation O_4 should wait until both of operations O_0 and O_6 are over. However, operation O_4 can be activated actually if only one operation out of O_0 and O_6 is over when an operation O_3 is over. Therefore, in order for performance enhancement through adoption of TAUs, we need a new concept of scheduling. The new scheduling method is not to allocate each operation to a specific time step but to decide execution order among vertices performing the same operation under available resource allocation. Therefore, in a new scheduling method, several schedule arcs are inserted in order to make the number of concurrently performed operations not bigger than the number of allocated corresponding resources. Fig. 3(b) shows a dependency graph, where each node corresponds to a multiplication operation and each edge represents dependency relation between two operations corresponding to adjacent nodes. When we consider only solid edges, the graph has three cliques ((0-1), (4), (6-8)) minimally and this means that at least three TAU-multipliers are required. Therefore, if we assume that two TAU-multipliers are allocated, we should insert additional schedule arcs. Fig. 3(b) shows that two edges(dotted edges) are inserted and the minimal number of cliques becomes two. This kind of scheduling methods have been already developed by several research groups[9, 10] and we use them. Fig. 3(c) shows a final scheduled DFG with 4 additional schedule arcs, where (O_0, O_1), (O_6, O_4, O_8), (O_3, O_2) and (O_7, O_5) are bound to TAU multiplier-1, 2 and adder-1, 2 respectively.

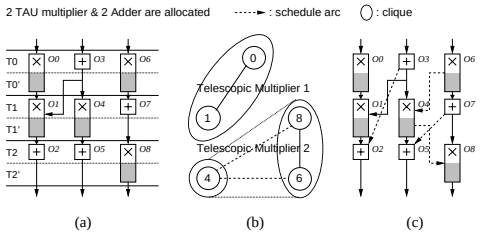


Figure 3. (a) A TAUBM DFG (b) A dependency graph for multiplications (c) A scheduled DFG

4. Derivation of a Distributed Synchronous Control Unit

4.1. A distributed synchronous control unit

FSMs for datapaths with TAUs come to have choices in next state transitions according to values of completion signals contrary to typical FSMs and this feature leads to the increase of the number of states. What is worse is that the number of states may increase exponentially with respect to the number of TAUs allocated to the same time step. Fig. 4(a) shows a part of an example FSM where two TAU multipliers, TM_1 and TM_2, are used in one time step. As we see in Fig. 4(a), a state S_i has $2^2 = 4$ state transitions, and similar choices occur in each subsequent state continuously. Note that the increase of the number of states causes circuit area overhead and hence performance degradation. If we assume a synchronization among operations in the same time step,

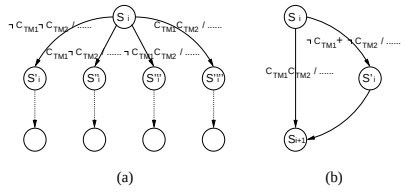


Figure 4. (a) Exponential increase of state transitions (b) A synchronized state transition('¬' means the negation)

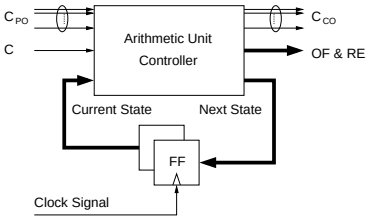


Figure 5. A basic structure of an arithmetic unit controller

we can derive a simpler FSM as shown in Fig 4(b) but the performance of a target system is degraded because of the first problem explained in 2.3.

In order to build global control units which can control a datapath with several TAUs efficiently, we propose a method to derive a set of distributed FSMs for a global control unit. In the proposed method, the granularity of distribution is an arithmetic unit. Why? A TAUBM FSM is suitable to the case that only one TAU is used and its synthesized controller can guarantee performance improvement effect of the TAU with variable computation time. Therefore, we intend to take advantage of a TAUBM FSM by restricting each FSM to one arithmetic unit. Since an independent synchronous controller is derived for an arithmetic unit, we call the controller an arithmetic unit controller. A global control unit consists of arithmetic unit controllers.

4.2. Derivation of arithmetic controllers from DFGs

Fig. 5 shows a basic structure of an arithmetic unit controller. In this paper, we denote completion signals of direct predecessor operation $\mathrm{O_i}$ and current operation $\mathrm{O_j}$ as $\mathrm{C_{PO}(i)}$ and $\mathrm{C_{CO}(j)}$ respectively. Here, $\mathrm{O_i}$ is a direct predecessor of $\mathrm{O_j}$ and $\mathrm{O_j}$ is a direct successor of $\mathrm{O_i}$ if $\mathrm{O_j}$ uses the result of $\mathrm{O_i}$ as an input operand. However, in this paper, a direct predecessor or successor relation is restricted to two operations executed on different arithmetic units since an arithmetic controller guarantees correct execution order for two operations using the same arithmetic unit automatically. As mentioned before, basically the idea of each arithmetic unit controller is the same as that of a TAUBM FSM. However, since we construct a distributed controller, where each entity corresponds to one arithmetic unit, by aggregating arithmetic unit controllers, we need to build a coordination mechanism, each arithmetic unit controller receives completion signals from the direct predecessor operations and gives completion signals to the direct successor operations in terms of given schedule. The algorithm to derive an FSM for an arithmetic unit controller corresponding to a TAU is as follows;

ALGORITHM 1 Derivation of an FSM for an arithmetic unit controller corresponding to a TAU

input: A DFG where scheduling and resource binding are completed.

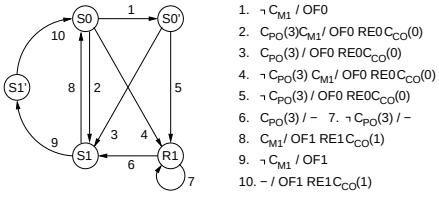
[step 1] For a TAU 'T', label operations, which are allocated to 'T', as $\mathrm{O_0}$, $\mathrm{O_1}$, .., $\mathrm{O_n}$.

[step 2] For every operation $\mathrm{O_i}$, i=0..n, generate states $\mathrm{S_i}$ and $\mathrm{S_i'}$. If there exist direct predecessor operations for $\mathrm{O_i}$, generate a ready state $\mathrm{R_i}$ additionally.

[step 3] For every operation $\mathrm{O_i}$, generate state transitions $\mathrm{S_i \rightarrow S_i'}$, $\mathrm{S_i \rightarrow S_{i+1}}$, and $\mathrm{S_i' \rightarrow S_{i+1}}$. For each state transition, corresponding input and output signals are as follows; $[\mathrm{S_i \rightarrow S_i'}]$: $\neg \mathrm{C_T}$ / $\mathrm{OF_i}$, $[\mathrm{S_i \rightarrow S_{i+1}}]$: $\mathrm{C_T}$ $\mathrm{C_{PO}}$'s / $\mathrm{OF_i}$ $\mathrm{RE_i}$ $\mathrm{C_{CO}(i)}$, $[\mathrm{S_i' \rightarrow S_{i+1}}]$: $\mathrm{C_{PO}}$'s / $\mathrm{OF_i}$ $\mathrm{RE_i}$ $\mathrm{C_{CO}(i)}$. In the second and third state transitions, input signals, $\mathrm{C_{PO}}$'s, are inserted only when there exist direct predecessor operations for $\mathrm{O_{i+1}}$.

[step 4] For every ready state $\mathrm{R_{i+1}}$, generate state transitions $\mathrm{S_i \rightarrow R_{i+1}}$, $\mathrm{S_i' \rightarrow R_{i+1}}$, $\mathrm{R_{i+1} \rightarrow S_{i+1}}$ and $\mathrm{R_{i+1} \rightarrow R_{i+1}}$. For each state transition, corresponding input and output signals are as follows; $[\mathrm{S_i \rightarrow R_{i+1}}]$: $\neg(\mathrm{C_{PO}}$'s) $\mathrm{C_T}$ / $\mathrm{OF_i}$ $\mathrm{RE_i}$ $\mathrm{C_{CO}(i)}$, $[\mathrm{S_i' \rightarrow R_{i+1}}]$: $\neg(\mathrm{C_{PO}}$'s) / $\mathrm{OF_i}$ $\mathrm{RE_i}$ $\mathrm{C_{CO}(i)}$, $[\mathrm{R_{i+1} \rightarrow S_{i+1}}]$: $\mathrm{C_{PO}}$'s / -, $[\mathrm{R_{i+1} \rightarrow R_{i+1}}]$: $\neg(\mathrm{C_{PO}}$'s) / -. $\mathrm{OF_i}$ and $\mathrm{RE_i}$ represent "operand fetch signal" and "register enable signal" for operation $\mathrm{O_i}$ respectively. For operation $\mathrm{O_n}$, $\mathrm{S_{n+1}}$ and $\mathrm{R_{n+1}}$ are $\mathrm{S_0}$ and $\mathrm{R_0}$ respectively.

In Algorithm 1, we generate two states, $\mathrm{S_i}$ and $\mathrm{S_i'}$, one corresponds to SD and the other corresponds to LD, for each operation which are bound to TAU 'T'. If an operation has direct predecessor operations, a ready state $\mathrm{R_i}$ is generated additionally in order to wait for activation of their completion signals(step 2). Then, according to the values of completion signals of direct predecessor operations and the TAU 'T', $\mathrm{C_{PO}}$'s and $\mathrm{C_T}$, several state transitions are generated with output signals such as "operand fetch signals($\mathrm{OF_i}$)", "register enable signals($\mathrm{RE_i}$)" and "current operation completion signals($\mathrm{C_{CO}(i)}$)"(step 3). For the case that completion signals from direct predecessor operations are not activated, state transitions to a ready state are generated(step 4). FSMs for non-TAU styled arithmetic units are also constructed in the similar way. We have only to remove a completion sig-



M1: TAU multiplier
OF & RE: operand fetch and register enable signals

Figure 6. A new FSM of an arithmetic unit controller for a DFG in Fig. 3(c)

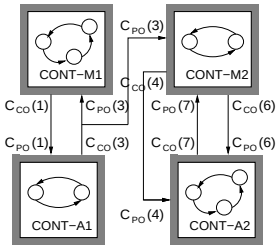


Figure 7. A distributed synchronous global control unit

nal C_T, states S_i''s, all the incoming and outgoing transitions of states S_i''s. Fig. 6 shows an FSM derived by algorithm 1. The FSM corresponds to a multiplier which is bound with operations O_0 and O_1 for a DFG in Fig. 3(c). In Fig. 6, state transitions "1, 2 & 3" and "4, 5, 6 & 7" correspond to three state transitions in step 3 and four state transitions in step 4 respectively. In S_0, multiplication O_0 can immediately start and generate operand fetch signal, OF_0, since there is no direct predecessor. If those fetched operands can be processed within 1 clock cycle, C_{M1}, $C_{CO}(0)$ and register enable signal, RE_0, are activated. Here, note that there are two possible state transitions, $S_0 \rightarrow S_1$ and $S_0 \rightarrow R_1$, since O_1 can start only after it receives a completion signal $C_{PO}(3)$. In S_0, if C_{M1} is not activated within 1 clock cycle, a state transition $S_0 \rightarrow S_0'$ occurs. Since a TAU has two delay levels in general, in S_0', $C_{CO}(0)$ and RE_0 are generated newly with OF_0 irrespective of the value of C_{M1}. Here, there are also two state transitions, and one of two is selected according to the value of the completion signal $C_{PO}(3)$ from a direct predecessor O_3. In a ready state R_1, a synthesized controller waits until O_3 finishes its operation, and goes to S_1 when the completion signal from O_3 is activated. Remaining behaviors are similar and thus we skip explaining it.

Fig. 7 shows a distributed synchronous global control unit for a DFG in Fig. 3(c) under the assumption that (O_0, O_1), (O_6, O_4, O_8), (O_3, O_2) and (O_7, O_5) are bound to TAU multiplier-1, 2 and adder-1, 2 respectively. In Fig. 7, note that several communication signals are optimized. For example, $C_{CO}(0)$ is removed since any other controllers do not receive it.

5. Experimental Results and Discussions

In this paper, we propose a method to build distributed synchronous control units for DFGs under TAU allocation. The main goal of distributed synchronous control units is to minimize idle time of component arithmetic units through complete preservation of concurrency among operations.

In order to perform area analysis of the proposed method,

Table 1. Area analysis results for TAUBM FSMs and a distributed FSM for a Diff. DFG

FSM	I/O	States	FFs	Area(Com./Seq.)
CENT-FSM	4 / 18	24	5	640 / 110
CENT-SYNC-FSM	4 / 18	7	3	96 / 66
DIST-FSM	4 / 18	20	10	251 / 220
D-FSM-M1	4 / 9	7	3	138 / 66
D-FSM-M2	3 / 9	6	3	53 / 66
D-FSM-A1	3 / 6	3	2	21 / 44
D-FSM-S1	4 / 6	4	2	39 / 44

• DIST-FSM is aggregation of D-FSM-M1, M2, A1 & S1.

we derive three types of FSMs, CENT-FSM, CENT-SYNC-FSM and DIST-FSM for the well-known high-level synthesis benchmark '*Differential Equation Solver*'(Diff.). Table 1 shows the area analysis results of control units synthesized from derived FSMs. CENT-FSM and CENT-SYNC-FSM are built based on expansion of TAUBM and DIST-FSM is derived by the proposed method. Note that CENT-FSM and CENT-SYNC-FSM are derived according to the styles shown in Fig. 4(a) and (b) respectively because original TAUBM considers allocation of only 1 TAU. Scheduling and resource binding for a DFG of Diff. are performed under allocation of two TAU multipliers(M1, M2), one adder(A1) and one subtractor(S1). Therefore, our proposed method derives 4 FSMs, D-FSM-M1, M2, A1, S1, and DIST-FSM is a set of them. As shown in Table 1, a distributed control unit(DIST-FSM) is three times larger than CENT-SYNC-FSM. This is due to the redundancy of sequential circuits and the communication signal overhead caused by the distribution of a control unit. However, as explained in section 4.1, CENT-SYNC-FSM causes performance degradation because of additional synchronization among TAUs in the same time step. CENT-FSM corresponding to the style of Fig. 4(a) guarantees performance as good as DIST-FSM. However, CENT-FSM represents complex relations among arithmetic units in one FSM, its synthesized circuit area becomes relatively large, for example about 1.6

Table 2. Latency comparison between TAUBM FSMs and new distributed FSMs

DFG	Resources	LT_{TAU}(ns)	LT_{DIST}(ns)	Performance Enhancement
3^{rd}FIR	×:2, +:1	[45][49.4, 57.1, 63.7][75]	[45][49.2, 56.2, 61.8][75]	$[0.4\%, 1.6\%, 2.9\%]$
5^{th}FIR	×:2, +:1	[75][81.9, 92.5, 99.4][105]	[75][77.9, 82.7, 86.3][90]	$[4.9\%, 10.6\%, 13.2\%]$
2^{nd}IIR	×:2, +:1	[75][80.7, 90.3, 97.5][105]	[75][77.9, 82.7, 86.3][90]	$[3.5\%, 8.4\%, 11.5\%]$
3^{rd}IIR	×:3, +:2	[75][83.1, 94.7, 101.3][135]	[75][80.6, 89.3, 95.9][135]	$[3.0\%, 5.7\%, 5.3\%]$
Diff.	×:2, +:1, -:1	[60][68.6, 82.9, 93.8][105]	[60][68.1, 80.7, 90.6][105]	$[0.7\%, 2.7\%, 3.4\%]$
AR-lattice	×:4, +:2	[120][140.6, 165.6, 176.3][180]	[120][134.2, 150.8, 160.2][165]	$[4.6\%, 8.9\%, 9.1\%]$

×: SD(×)=15ns, LD(×)=20ns / +, -: FD(+, -)=15ns

times larger than DIST-FSM, as already explained in section 4.1. Therefore, we can conclude that the distributed approach enables designers to build performance efficient synchronous control units with small additional area overhead.

For the analysis of performance effects of the proposed method, we applied it to several DFG benchmarks, and Table 2 shows analysis results. FSMs for latencies LT_{TAU} and LT_{DIST} are derived through expanded TAUBM based on the style shown in Fig. 4(b) and the proposed method respectively. Each result for latencies consists of [best case], [average cases where P is equal to 0.9, 0.7 and 0.5], [worst case]. Average latencies are acquired as changing the value of 'P' because performance effects of a TAU are sensitive to its SD occurrence ratio 'P'. The 5^{th} column in Table 2 shows performance enhancement ratios achieved by our proposed approach.

Experimental results in Table 1 and 2 show that the proposed method can guarantee performance improvement with small additional area overhead explicitly. This feature is due to the fact that the proposed method succeeds in transforming performance effects of component TAUs into a system performance under a distributed structure of a global synchronous control unit. In conclusion, the proposed distributed approach is expected to be useful for building performance efficient synchronous control units for datapaths including TAUs.

6. Future Work

Although TAUs are considered mainly in this paper, the proposed method can be applied to other types of VCAUs without special modification. Therefore, we have concentrated our effort on developing a new high-level synthesis tool which can manage various kinds of VCAUs through integration of the proposed method into an existing our high-level synthesis tool. For it, development or modification of new or existing high-level synthesis algorithms in scheduling, resource allocation and binding procedures will be required. Moreover, construction of a hardware resource library including VCAUs is one of important future work.

Acknowledgements

This work was partially supported by MEXT of JAPAN, Special Coordination Fund for Science and Technology and the Korea Science and Engineering Foundation(KOSEF) through the Ultra-Fast Fiber Optic Networks Research Center at Kwangju Institute of Science and Technology.

References

[1] L. Benini, E. Macii, M. Poncino and G. DeMicheli, "Telescopic Units: A New Paradigm for Performance Optimization of VLSI Designs," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.17, no.3, pp.220-232, Mar. 1998.

[2] L. Benini, E. Macii and M. Poncino, "Efficient Controller Design for Telescopic Units," In Proceedings of IEEE International Conference Innovative Systems in Silicon, pp.290-299, Oct. 1997.

[3] G. DeMicheli, "Synthesis and Optimization of Digital Circuits," pp.174-178, New York: McGraw-Hill, 1994.

[4] S. Hauck, "Asynchronous Design Methodologies: an Overview," In Proceedings of the IEEE, vol.83, no.1, pp.69-93, 1995.

[5] S. M. Nowick, K. Y. Yun, A. E. Dooply and P. A. Beerel, "Speculative Completion for the Design of High-Performance Asynchronous Dynamic Adders," In Proceedings of Async'97, pp.210-223, 1997.

[6] S. M. Nowick, "Automatic Synthesis of Burst-Mode Asynchronous Controllers," Ph.D. Dissertation, Stanford University, 1995.

[7] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," IEICE Trans. Inf. & Syst., vol.E80-D, no.3, pp.315-325, Mar. 1997.

[8] E. Kim, J.-G. Lee and D.-I. Lee, "Automatic Process-Oriented Asynchronous Control Unit Generation from Control Data Flow Graphs," IEICE Trans. Fund., vol.E84-A, no.8, pp.2014-2028, Aug. 2001.

[9] B. M. Bachman, H. Zheng and C. J. Myers, "Architectural Synthesis of Timed Asynchronous Systems," In Proceedings of IEEE International Conference on Computer Design, pp.354-363, Oct., 1999.

[10] E. Kim and D.-I. Lee, "Resource Constrained Asynchronous Scheduling Method through Transformation of Dataflow Graph," In Proceedings of 2001 IEEE International Symposium on Circuits and Systems, pp.V-41-V-44, 2001.