Performance Optimization of Synchronous Control Units for Datapaths with Variable Delay Arithmetic Units

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

RCAST, Univ. of Tokyo[†]
Tel: +81-3-5452-5160
Fax: +81-3-5452-5161

UFON, K-JIST[‡]
Tel: +82-62-970-2267
Fax: +82-62-970-2204

e-mail: {**eskim**, hiroshi, nakamura, nanya}@hal.rcast.u-tokyo.ac.jp & {eulia, dilee}@kjist.ac.kr

Abstract— Nowadays, variable delay arithmetic units have been used for implementing a datapath of a target system in pursuit of performance improvement. However, adoption of variable delay arithmetic units requires modification of a typical synchronous control unit design methodology. A telescopic arithmetic unit based methodology is one of representative methodologies to design synchronous control units for variable delay datapaths. In this paper, we propose two optimization methods for it. Proposed optimization techniques will be analyzed in order to show their performance improvement effects explicitly.

I. Introduction

Although synchronous system designers still assume that all the component arithmetic units of a datapath operate with their own fixed delays, variable delay arithmetic units, in short VDAUs, have started to be implemented and used in pursuit of performance improvement of target systems[1, 2, 3]. However, adoption of VDAUs requires modification of a typical synchronous control unit design methodology.

[1, 2] proposed how to synthesize VDAUs, which were called telescopic arithmetic units, in short TAUs, automatically, and how to modify an original finite state machine, in short FSM, into a new FSM which can control a variable delay datapath with TAUs respectively. Although they achieved noticeable and pioneering results, their work is not optimized enough in following two aspects. The first, concurrent execution among VDAUs are not supported enough and unnecessary synchronizations are required. The second, selection of proper clock cycles is restricted. In [3], a synchronous independent controller is built for each operation, and all the controllers are integrated into a global control unit. 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 two optimization methods in order to ameliorate above two problems of the previous approach[1, 2]. The core features of our proposed new optimization methods are "to support concurrent executions among VDAUs completely" and "to reduce idle time of VDAUs by lowering the lower bound for selection range of clock cycles".

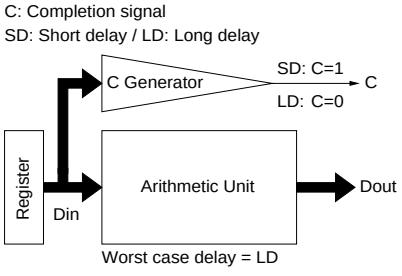


Fig. 1. A structure of a telescopic arithmetic unit(SD: short delay / LD: long delay)

II. Preliminaries

A. Variable delay arithmetic units

A telescopic arithmetic unit[1, 2], in short TAU, consists of the following two parts, an arithmetic unit and a completion signal generator as shown in Fig. 1. The arithmetic unit part of a TAU is exactly the same as general synchronous arithmetic units. The completion signal generator, which is a distinctive part of VDAUs, generates a completion signal when it decides that computation for input operands is over. For convenience of an explanation, we define two variables LD(Long Delay) and SD(Short Delay). Actually, LD corresponds to the worst case delay of the arithmetic unit. Note that the real computation time varies according to input operands although general synchronous arithmetic units are assumed to have the worst and fixed computation time for easy design. 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

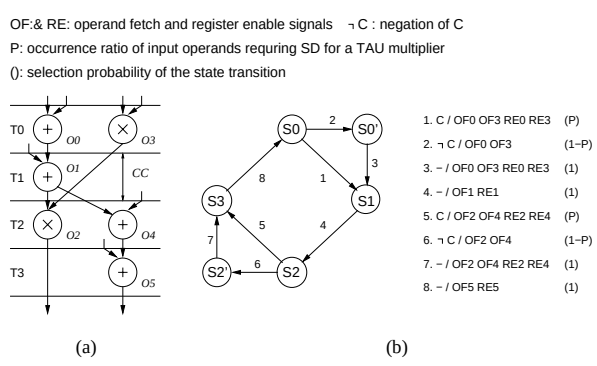


Fig. 2. (a) An original dataflow graph (b) A corresponding TAUBM FSM('C': a completion signal from a TAU styled multiplier)

know 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 synthesis method of the completion signal generator was also developed in [1, 2].

B. A synchronous control unit generation method based on TAUs

In addition to development 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. we call it TAU based methodology, in short TAUBM. Although the initial TAUBM was developed to be applied to the modification of any given FSM, we restrict our discussion to FSMs corresponding to controllers for dataflow graphs, in short DFGs because we intend to implement the proposed method under our high-level synthesis tool. An FSM modification method based on TAUBM is as follows[1, 2]; [step 1] If a time step $\mathrm{T_i}$ includes TAU operations, divide the corresponding state $\mathrm{S_i}$ into $\mathrm{S_i}$ and $\mathrm{S_i'}$. Here, a TAU operation represents an operation to which a TAU is allocated. [step 2] For the case where the TAU consumes LD, state transitions, $\mathrm{S_i} \rightarrow \mathrm{S_i'} \rightarrow \mathrm{S_{i+1}}$, are generated. For the other case where the TAU requires SD, a state transition, $\mathrm{S_i} \rightarrow \mathrm{S_{i+1}}$, is generated. In $\mathrm{S_i} \rightarrow \mathrm{S_i'}$, register enable signals are not generated because allocated TAUs do not finish their corresponding TAU operations. In order to make a decision between $\mathrm{S_i} \rightarrow \mathrm{S_{i+1}}$ and $\mathrm{S_i} \rightarrow \mathrm{S_i'}$, a completion signal 'C' from the TAU is considered in the input set of an FSM. We call the finally modified FSM a TAUBM FSM. Fig. 2 shows an example DFG and a corresponding TAUBM FSM respectively. For the more detailed explanation about a modification procedure, please refer to [1, 2]. Note that execution latencies of TAUBM FSMs vary because different state transition paths are selected according to the values of completion signals from TAUs. For example, execution latency of TAUBM FSM in Fig. 2(b) varies between 4 and 6 clock cycles according to the value of a completion signal 'C'.

[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.

III. Optimization of TAUBM Synchronous Control Units for Variable Delay Datapaths - I

The most distinctive feature of TAUBM is that TAUs constituting a variable delay datapath spend 1 or 2 clock cycles selectively according to their input operands. Although only TAUs spend variable computation time, original TAUBM often require unnecessary synchronizations with TAUs for other arithmetic units. If the ratio of input operands requiring SD for a TAU, 'P', is big, the synchronization may not be critical. However, otherwise, performance degradation due to the unnecessary synchronization is not negligible. You should note that 'P' is not the variable that system designers can control. That is, the value of 'P' depends on the set of input operands entirely. Therefore, in the optimization method I, we remove all the unnecessary synchronizations and reduce bad effects of low 'P' by supporting concurrent executions among arithmetic units completely. The following is a simple algorithm to derive a new FSM through the optimization method I.

ALGORITHM 1 Derivation of a new FSM through the optimization method I

DerivationOfFsmInOptimization-I() {
  Generate an initial state S_0*;*
  Cstates = RecursiveGenerateAllChildStates(S_0*);*}
RecursiveGenerateAllChildStates(S_i*)* {
  Generate the set of child states, Cstates, for S_i*;*
  Forall Cstates(i)
    RecursiveGenerateAllChildStates(Cstates(i)); }

The function $RecursiveGenerateAllChildStates(S_i)$ in above algorithm generates all successor states which can be reachable from the state $\mathrm{S_i}$ by activating all operations whose input operands and allocated arithmetic unit is available. Therefore, FSMs derived in Algorithm 1 generate all the reachable states according to the delays of TAUs. For better understanding of the concept of the optimization method I, compare two FSMs in Fig. 2(b) and Fig. 3. They are FSMs, which are derived by applying TAUBM and the optimization method I respectively, for a DFG in Fig. 2(a). Here, we assume that a TAU styled multiplier is allocated to operations 2 and 3. In TAUBM, since the 2^{nd} stage of a TAU styled multiplier is spent selectively according to input operands, we should not allocate some operations to the same time interval the 2^{nd} stage of the TAU is allocated to. Therefore, operation 1, which is a successor of operation 0, should be delayed until operation 3 is over, although it can be started once operation 0 is over as shown in Fig. 2(a). However, the optimization method I enables the corresponding FSM to activate operations as soon as possible because it explores all the reachable states according to the delays of TAUs. For example, in Fig. 3, operation 1 is always activated in the 2^{nd} time step and operations 0, 1 and 3 can be performed in 2 clock cycles irrespective of the delay of TAU operation 3 in both state transition paths $\mathrm{S_0} \rightarrow \mathrm{S_2} \rightarrow \mathrm{S_3}$ and $\mathrm{S_0} \rightarrow \mathrm{S_1} \rightarrow \mathrm{S_3}$, while they spend 2 or 3 clock cycles in TAUBM. Operations 2, 4 and 5 are also same.

For performance analysis, we define the execution latency of a TAUBM FSM as $LT_{TAU} = \sum_{i=0}^{i=N-1} CC \cdot P^{k(i)} + 2 \cdot CC \cdot$

TABLE I
Comparison between TAUBM FSMs and new FSMs obtained by the optimization method I

DFG	Resources	States(FF)	$\mathrm{LT_{TAU}}$(ns)	States(FF)	$\mathrm{LT_{OPT1}}$(ns)	Performance Enhancement
3^{rd}FIR	×:2, +:1	6(3)	[45][49.4, 57.1, 63.7][75]	10(4)	[45][49.2, 56.2, 61.8][75]	[0.4%, 1.6%, 2.9%]
5^{th}FIR	×:2, +:1	7(3)	[75][81.9, 92.5, 99.4][105]	19(5)	[75][77.9, 82.7, 86.3][90]	[4.9%, 10.6%, 13.2%]
2^{nd}IIR	×:2, +:1	7(3)	[75][80.7, 90.3, 97.5][105]	15(4)	[75][77.9, 82.7, 86.3][90]	[3.5%, 8.4%, 11.5%]
Diff.	×:2, +:1, -:1	7(3)	[60][68.6, 82.9, 93.8][105]	24(5)	[60][68.1, 80.7, 90.6][105]	[0.7%, 2.7%, 3.4%]

×: SD(×)=15ns, LD(×)=20ns / +, -: FD(+, -)=15ns / FF: the number of flip-flops

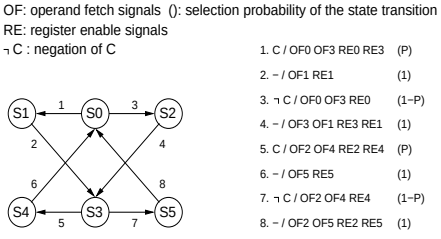


Fig. 3. A new FSM obtained by the optimization method I for Fig. 2(a)

$(1 - P^{k(i)})$. Here, CC, k(i) and N represent a new clock cycle selected according to the value of SD, the number of TAUs in the i-th time step, and the number of time steps. Note that $\mathrm{LT_{TAU}}$ varies according to the value of 'P'. For example, in the case of Fig. 2(b), the execution latency is (6-2P)·CC. In the worst case, 'P' is 0, $\mathrm{LT_{TAU}}$ is 6·CC, but in the best case, 'P' is 1, $\mathrm{LT_{TAU}}$ is 4·CC. Contrary to TAUBM, new FSMs derived in the optimization method I activates operations, whose input operands and allocated arithmetic unit is available, as soon as possible in all the state transition paths and thus they can activate some operations at the same time step which 2^{nd} stages of TAUs are allocated to. Therefore, new FSMs always guarantee performance as good as corresponding TAUBM FSMs at least, and improve average performance. For example, a new FSM in Fig. 3 always spend 4 clock cycles irrespective of the various delays of the TAU.

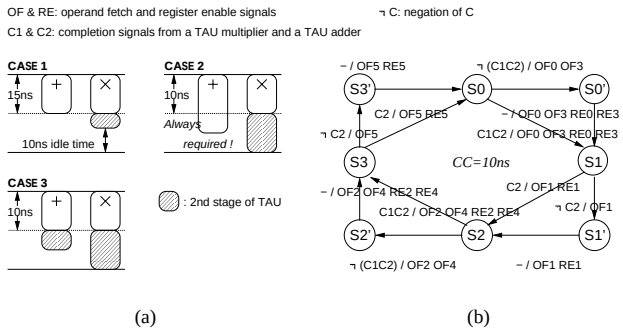
In order to check the effects of the optimization method I in aspects of performance and area, we derived FSMs for several DFG benchmarks through TAUBM and the optimization method I respectively, and Table I shows the analysis results. In the aspect of area, it seems that the optimization method I requires larger circuit areas because the number of states(5^{th} column) for corresponding FSMs is more than the number of states(3^{rd} column) for TAUBM FSMs. This is due to the fact that new FSMs include all reachable states. However, the goal of the adoption of VDAUs such as TAUs is to improve the performance of targeted systems. Latencies in 4^{th} column and 6^{th} column are latencies for FSMs obtained by TAUBM and the optimization method I. Each result for latencies consist of [best case], [cases where P is equal to 0.9, 0.7 and 0.5], [worst case]. The last column shows the resulting performance enhancement ratios. As you can see, the optimization method I can guarantee performance improvement in spite of additional area overhead, and thus we can conclude that it is useful for the main goal of adoption of VDAUs such as TAUs.

IV. Optimization of TAUBM Synchronous Control Units for Variable Delay Datapaths - II

Although the selection of a small sized clock cycle presents larger flexibility for scheduling and corresponding controller generation[6, 7], a small sized new clock cycle, which is selected according to the value of SD of a TAU, does not always guarantee good performance in TAUBM because wrong selection of SD may make operations spend multiple time steps counterbalancing performance benefits of TAUs. Therefore, although the value of SD can be selected freely and the corresponding TAU can be synthesized according to the selected SD automatically[1], the selection of SD should be performed very carefully.

When we select the value of SD, we should consider following two things; the first is that SD should be larger than the fixed delay of other arithmetic units excepts TAUs. Otherwise, arithmetic units whose fixed delays are larger than SD cannot finish their operations within SD and 2^{nd} stages of some TAUs will be always required. The second is that 2·SD is larger than LD because TAUs spend 2 time steps at most. From above two facts, the following selection range of SD can be derived; $Max(LD/2, Max(FD_0, FD_1, ..)) \leq SD \leq LD$. Here, $Max(FD_0, FD_1, ..)$ represents maximum fixed delay among other arithmetic units except TAUs. Note here that it is not good to select SD similar to LD since (2· SD)-LD means the idle time of the corresponding TAU. Therefore, it is better to select SD near to LD/2 in order to minimize the idle time. However, $Max(FD_0, FD_1, ..)$ is a direct hurdle to the selection of SD near to LD/2. In order to remove the hurdle , we can consider following two approaches. The first one is to adopt the new arithmetic units with smaller fixed delays. The second one is to replace the current non TAU styled arithmetic units with corresponding TAUs. The first approach is actually trivial. Therefore, in this section, we consider the second approach and we call it the optimization method II.

We would like to explain how to apply the optimization method II through a simple example. For a DFG in Fig. 2(a), we assume that a TAU styled multiplier 'M', whose short delay, SD(M) and long delay LD(M) are 15ns and 20ns, and an adder 'A', whose FD(A) is 15ns, are allocated. In this case, if a clock



(a) (b)

Fig. 4. (a) Three cases according to clock cycles and TAU allocation (b) A new FSM obtained by the optimization method II

TABLE II
Latency comparison between TAUBM FSMs and new FSMs optimized by the optimization method II

DFG	Resources	$\mathrm{LT_{TAU}}$(ns)	$\mathrm{LT_{OPT2}}$(ns)
3^{rd}FIR	×:2, +:1	[45][53.4][75]	[30][37.6, 41.3, 44.4][60]
5^{th}FIR	×:3, +:1	[75][87.7][105]	[50][61,1, 65.1, 67.8][100]
2^{nd}IIR	×:2, +:1	[75][85.8][105]	[50][59.6, 62.1, 66.8][100]
Diff.	×:2, +:1, -:1	[60][76.2][105]	[40][54.4, 60.6, 65.2][70]

×: SD(×)=15ns→ SD(×)=10ns, LD(×)=20ns
+, -: FD(+, -)=15ns, SD(+, -)=10ns, LD(+, -)=15ns

cycle is set to 15ns according to the value of SD(M), idle time as much as 10ns occurs whenever the TAU styled multiplier requires LD(M) as shown in CASE 1 of Fig. 4(a). Although designers hope to minimize idle time by selecting smaller SD(M), it always does not lead to performance improvement because of FD(A). That is, if we select new SD(M) smaller than FD(A), the corresponding adder always spends 2 time steps because a clock cycle is adjusted to new SD(M) as shown in CASE 2 of Fig. 4(a). However, assume that we replace the current fixed delay adder with a TAU styled adder 'A', whose SD(A) and LD(A) are 10ns and 15ns respectively. Then, things are changed; addition operations also spend 1 or 2 clock cycles selectively according to input operands instead of spending 2 time steps as shown in CASE 3 of Fig. 4(a). As a consequence, selection range of SD for a TAU styled multiplier changes into $\mathrm{Max(LD(M)/2, SD(A))} \leq \mathbf{SD(M)} \leq \mathrm{LD(M)}$ from $\mathrm{Max(LD(M)/2, FD(A))} \leq \mathbf{SD(M)} \leq \mathrm{LD(M)}$, and thus additional minimization of idle time becomes possible with the adoption of a TAU styled adder. Fig. 4(b) shows a new FSM obtained by the optimization method II when we assume that a TAU styled multiplier and a TAU styled adder are allocated.

For the analysis of effects of the optimization method II, we applied it to several DFG benchmarks, and Table II shows analysis results. For $\mathrm{LT_{TAU}}$ in 3^{rd} column, each latency consists of [best case], [average case] and [worst case]. Average latency is obtained under the assumption SD ratio 'P' is 0.8. For $\mathrm{LT_{OPT2}}$ in 4^{th} column, we assume that TAU styled adder and subtractor, whose SDs are 10ns, are allocated. Therefore, we should resynthesize TAU styled multipliers so that their SD may be 10ns, and thus SD ratio 'P' is changed. In our experiment, we assume that 'P' is reduced to '$0.9 \cdot \mathrm{P}$', '$0.7 \cdot \mathrm{P}$' and '$0.5 \cdot \mathrm{P}$'. Under the assumption, Table II shows that selection of new SD under adoption of additional kinds of TAUs can lead to the performance improvement.

V. Conclusions and Future Work

In this paper, we propose two optimization methods for TAUBM. The core features of them are "to support concurrent executions among VDAUs completely" and "to reduce idle time of VDAUs by lowering the lower bound for selection range of clock cycles". Experimental results show that proposed optimization techniques are effective in the aspect of performance.

In order to take advantage of VDAUs such as TAUs in practical CAD environment, we concentrate our efforts on automating the proposed methods and integrating them into our high-level synthesis tool. Moreover, establishment of hardware resource library including VDAUs have been performed.

Acknowledgements

This work has been supported in part by Research Center for Advanced Science and Technology at University of Tokyo 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," 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. Chaudhuri, S. A. Blthye and R. A. Walker, "A Solution Methodology for Exact Design Space Exploration in a Three-Dimensional Design Space," IEEE Trans. on Very Large Scale Integration Systems, vol.5, no.1, pp.69-81, Mar. 1997.

[7] S. Narayan and D. D. Gajski, "System Clock Estimation Based on Clock Slack Minimization," In Proceedings of DAC'92, pp.66-71, Sep. 1992.