PAPER

Statistical Analysis Driven Synthesis of Application Specific Asynchronous Systems

Koji OHASHI[†] ***and*** **Mineo KANEKO**[†a)]**,** ***Members***

SUMMARY In this paper, we propose an effective asynchronous datapath synthesis system to optimize statistical performance of asynchronous systems. The proposed algorithm is a heuristic method which simultaneously performs scheduling and resource binding. During the design process, decisions will be made based on the statistical schedule length analysis. It is demonstrated that asynchronous datapaths with the reduced mean total computation time are successfully synthesized for some datapath synthesis benchmarks.

key words: *asynchronous system, scheduling, binding, statistical analysis*

1. Introduction

As a VLSI system becomes larger and a clock period becomes shorter, it becomes difficult to control a digital circuit by a global clock under the fluctuation of datapath delays and clock skew. Asynchronous design is considered as a promising alternative, since it is free from such a global clock. Also it has the potential to achieve low power consumption, higher average-case performance, and higher reliability [1]. To design a cost effective high performance asynchronous system for a specified application, optimization of a datapath in the register transfer level is an important design step. Scheduling and resource binding (assignment) are major subtasks in datapath synthesis not only for synchronous systems but also for asynchronous systems [2]–[4].

Mercury [2] is known as a synthesis system of asynchronous datapaths. Since the execution time of each operation in an asynchronous system is not fixed and must be treated as a variable, scheduling is defined as a task to determine the sequence of operations by adding "resource edges" to a data dependency graph. In [2], schedule (i.e. set of resource edges) is explored in branch-and-bound fashion, and Asynchronous-Left-Edge algorithm is applied to each schedule to obtain resource binding. In [3], a new filter (called "one direction") has been proposed and applied to Mercury. One direction is a heuristic pruning based on the sequentialization of ALAP schedule. By using this filter, the design space explored by Mercury is reduced, and datapaths are synthesized efficiently. However the computation time is still exponential to the input size because of the nature of the branch-and-bound method. In [4], the assignment space exploration based synthesis system has been proposed. It explores the assignment solution space using Simulated Annealing (SA), and each solution visited in SA is evaluated by its schedule length obtained by assignment constrained scheduling. Similar to [2], the execution time of each operation is treated as a variable, and the scheduling problem is transformed into the problem to decide the sequence of operations and data assigned to the same resource by adding "disjunctive arcs" to a dependence graph. The system in [4] also takes a long time for larger instances because of the nature of SA based exploration of the solution space.

On the other hand, as the feature size becomes smaller, the density of devices and wires becomes higher and the operation speed becomes higher, delay variations due to local supply noises, local variations of temperature, crosstalks between wires, local manufacturing imperfections, etc., cannot be neglected. For a synchronous system, the effect of delay variations is masked by the clock period and delay margins. For an asynchronous system, it affects the total computation time of an application directly.

In this paper, we propose a new asynchronous datapath synthesis system to synthesize asynchronous datapaths having the minimum statistical execution time of a specified application. The important features of this work include the following points. (i) Statistical performance (statistical execution time), instead of deterministic performance, is chosen as the objective for datapath optimization, and a synthesis algorithm is developed toward this purpose. (ii) Existing methods [2]–[4] are based on the branch-and-bound exploration or the SA based exploration of a solution space, and hence they consume too much time to apply to large instances. To apply to large applications and to find solutions successfully, we develop a heuristic algorithm which can work in polynomial time of the input size.

The rest of this paper is organized as follows. Section 2 is devoted to an overview of asynchronous schedule. Section 3 shows the statistical schedule length analysis. Section 4 presents our asynchronous datapath synthesis system with the statistical makespan analysis. Section 5 shows experimental results, and finally Sect. 6 gives conclusions.

2. Preliminaries

2.1 Asynchronous Datapath

In a synchronous system, the start of each operation is controlled by a FSM, in which the transition of a state is trig-

Manuscript received July 3, 2006.
Final manuscript received November 24, 2006.

[†]The authors are with the School of Information Science, Japan Advanced Institute of Science and Technology, Nomi-shi, 923-1292 Japan.
a) E-mail: mkaneko@jaist.ac.jp

DOI: 10.1093/ietfec/e90–a.3.659

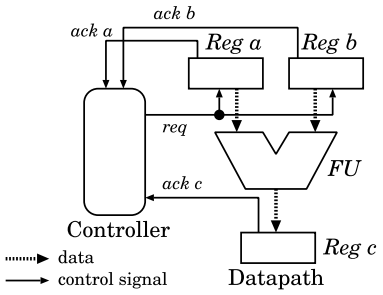


Fig. 1 Handshaking between a controller and a datapath.

gered by a clock signal. Hence, we can control the start of each operation by assigning each operation to one of the states of a FSM, and a schedule is usually considered as a mapping from operations to $\mathbb{Z}$ (integers).

In contrast, in an asynchronous system, no clock signal is used, and each operation starts under a handshake protocol between a controller and a datapath. Figure 1 shows an example of handshaking between a controller and a datapath. The controller knows, by receiving control signals *ack a* and *ack b*, that necessary data have been prepared. A control signal *req* is sent from a controller to registers *Reg a* and *Reg b*, and the data are sent out from the registers to the functional unit FU. The data are then processed in FU, and the output data is written in the register *Reg c*. Then a control signal *ack c* is sent from the register to the controller.

In the above case, the start of an operation is triggered by the event that all required input data have been prepared. Note that, generally, we can add some other events besides data dependency to trigger the execution of an operation if it is necessary. Thus, in such an asynchronous framework, synchronous scheduling, such as Force-directed scheduling [5], List scheduling [6], and improved synchronous scheduling [7] (i.e., start time assignment) cannot be used.

2.2 Scheduling Graph and Scheduling Problem

In this subsection, we will briefly summarize representation models and asynchronous schedule.

A target application algorithm to be implemented is first transformed into a directed graph $G = (V_G, A_G)$, which is called a "dependence graph." V_G is the union of the set V_O of operations and the set V_D of data, and A_G is the union of the set A_O of arcs from operations to data and the set A_I of arcs from data to operations. Auxiliary operations o_{init} and o_{quit} are introduced for identifying primary input and primary output data explicitly. Figure 2(a) shows an example of the dependence graph.

A graph which represents precedence constraints between operations will be called a "scheduling graph" $G_S = (V_S, A_S)$. The precedence relations in G_S come from two different sources, one is the mandatory precedence relation specified by the structure of a target application algorithm, and the other is the optional precedence relation brought by scheduling. The arcs representing the latter are called "disjunctive arcs" in this paper. V_S is the set of the starts and the ends of operations, and we denote the start node and the end node of each operation o_i as o_i^s and o_i^e, respectively. A_S is the set of arcs which represent precedence constraints between nodes in V_S. The weight of an arc (o_i^s, o_i^e) corresponds to the execution time of operation o_i, which is denoted by $\varepsilon(o_i)$. For the other arcs, the weights of them are 0. Figure 2(b) shows a scheduling graph constructed from the dependence graph in Fig. 2(a).

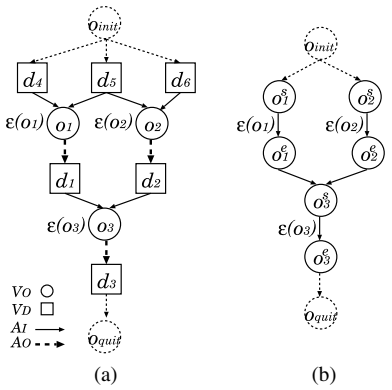


Fig. 2 Dependence graph (a) and its scheduling graph (b).

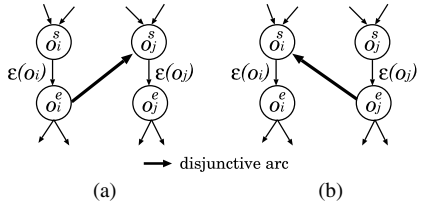


Fig. 3 Disjunctive arcs induced by functional unit assignment.

The disjunctive arcs are introduced in G_S to avoid a collision between operations or between data. That is, if two operations o_i and o_j are assigned to the same functional unit, then one lifetime precedes the other. This constraint can be represented by adding either an arc (o_i^e, o_j^s) (Fig. 3(a)) or an arc (o_j^e, o_i^s) (Fig. 3(b)) to G_S. Similarly if two data d_i and d_j, which are generated by operations o_i and o_j, respectively, are assigned to the same register, then one lifetime precedes the other. The constraint reflecting this register assignment can be represented by adding either arcs $(s_x(o_i)^e, o_j^e)$ for every operations $s_x(o_i)$ (Fig. 4(a)) or arcs $(s_y(o_j)^e, o_i^e)$ for every operations $s_y(o_j)$ (Fig. 4(b)) to G_S, where $s_x(o_i)$ and $s_y(o_j)$ are operations using data d_i and data d_j as their inputs, respectively.

As a result, a set of disjunctive arcs (or a resultant scheduling graph itself in a wide sense), which can resolve constraints imposed from resource assignment, is considered as an asynchronous schedule, because, by designing a controller according to the resultant scheduling graph, operations can be executed correctly without the conflict of lifetimes in as soon as possible manner under precedence relations given by the scheduling graph.

Note that, for every pair of operations o_i and o_j which are assigned to the same functional unit, we need to decide

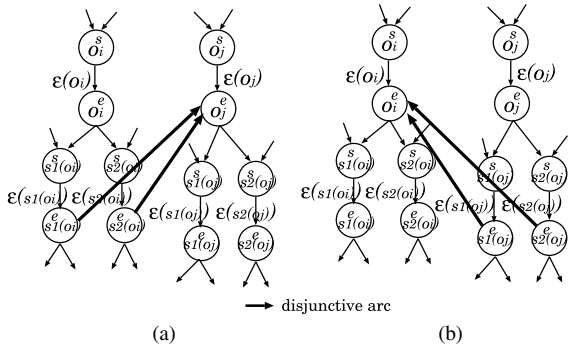


Fig. 4 Disjunctive arcs induced by register assignment.

which one of the alternative disjunctive arcs is added to G_S. Also for every pair of data d_i and d_j which are assigned to the same register, we need to decide which one of the alternative sets of disjunctive arcs is added to G_S. Different decisions yield different sets of disjunctive arcs (i.e., different schedules), and to find an optimum set is our assignment constrained scheduling problem. The optimality of the schedule depends on the evaluation of the schedule. Statistical schedule length as an evaluation method of the schedule will be described in Sect. 3, and a scheduling method will be shown in Sect. 4.3.

3. Statistical Schedule Length Analysis

Not only to evaluate a final schedule but also to make decisions in a scheduling process, we need to evaluate a schedule (equivalently, a scheduling graph) in some way. In the conventional evaluation phase [2]–[4], they introduce a specious constant delay for the execution time of each operation, such as typical delay, maximum delay or minimum delay, and compute typical (maximum or minimum) makespan by using typical (maximum or minimum) execution delay. If there is no random delay variation between modules and delays of all modules vary uniformly, the above constant delay model seems to be an acceptable way to handle delay variations. However, due to local supply noise, local variations of temperature, crosstalks between wires, local manufacturing imperfections, etc., functional delay and transmission delay can vary easily, and the calculation of minimum, typical, and maximum total computation time based on the constant delay model is unacceptable for those random delay variations.

In [8], a statistical schedule length analysis for evaluating a schedule during asynchronous datapath synthesis has been proposed. The statistical delay analysis handles correlations induced by three different sources; (1) correlation between delays on different modules and nets, (2) reconvergent paths in a scheduling graph, and (3) resource sharing (depends on resource binding).

In this paper, the execution time $\varepsilon(o_i)$ is modeled by a stochastic variable having normal distribution as done in [8]–[10], and the schedule length of a schedule is also considered to be distributed randomly. In what follows, the weight of each arc e in A_S is denoted by $w(e)$, and the normal distribution of $w(e)$ is denoted by $N(\mu(e), \sigma^2(e))$, where $\mu(e)$ and $\sigma^2(e)$ are the mean and the variance of the distribution, respectively. For two arcs e_i and e_j in A_S, the correlation coefficient between $w(e_i)$ and $w(e_j)$ is denoted by $\rho(e_i, e_j)$.

The statistical schedule length analysis in [8] is summarized as follows, in which $l_i^t(v)$ denotes the longest path length from o_{init} to v passing through the ith incoming arc of v, $l_i(v)$ denotes the longest path length from o_{init} to v passing through one of 1st to ith incoming arcs of v, and $l(v)$ denotes the longest path length from o_{init} to v, that is $l(v) = l_{f(v)}(v)$, where $f(v)$ is the in-degree of v.

Algorithm: Statistical Schedule Length Analysis

Step 1: Depending on the binding, $\mu(e)$ and $\sigma^2(e)$ are assigned to each arc e in G_S. In addition, $\rho(e_i, e_j)$ is assigned to every pair of arcs e_i and e_j.

Step 2: All nodes in G_S are sorted in topological order.

Step 3: If $l(o_{quit})$ is computed, then the mean $E\left[l(o_{quit})\right]$ and the variance $V\left[l(o_{quit})\right]$ are outputted. Otherwise, a node v is selected from G_S in topological order.

Step 4: If $v = o_{init}$, we set $E[l(v)] = V[l(v)] = 0$ and the correlation coefficients $R[l(v), l(v)] = 1$ and $R[l(v), w(e)] = 0$ for each arc $e \in A_S$, respectively, and go to Step 3. Otherwise, for each incoming arc of v $e_i = (u_i, v)$ $(i = 1, 2, \cdots, f(v))$, we compute $l_i^t(v)$. Using the probabilistic equivalent of $l_i^t(v) = l(u_i) + w(e_i)$ ("add" operation), we calculate the mean $E[l_i^t(v)]$ and the variance $V[l_i^t(v)]$ of $l_i^t(v)$ as follows.

$$
\begin{aligned}
E\left[l_i^t(v)\right] &= E[l(u_i)] + \mu(e_i), \\
V\left[l_i^t(v)\right] &= V[l(u_i)] + \sigma^2(e_i) \\
&\quad + 2\sqrt{V[l(u_i)]\sigma^2(e_i)}R[l(u_i), w(e_i)].
\end{aligned}
$$

Step 5: For each node $x \in V_S$ whose $l(x)$ has been already computed, we calculate correlation coefficient $R\left[l_i^t(v), l(x)\right]$ between $l_i^t(v)$ and $l(x)$ as follows.

$$
\begin{aligned}
&R\left[l_i^t(v), l(x)\right] \\
&= \frac{\sqrt{V[l(u_i)]}R[l(x), l(u_i)] + \sigma(e_i)R[l(x), w(e_i)]}{\sqrt{V\left[l_i^t(v)\right]}}.
\end{aligned}
$$

Note that we set $R[l(x), l(u_i)] = 1$, if $x = u_i$.

Similarly, for each arc $y \in A_S$ we calculate correlation coefficient $R\left[l_i^t(v), w(y)\right]$ between $l_i^t(v)$ and $w(y)$ as follows.

$$
\begin{aligned}
&R\left[l_i^t(v), w(y)\right] \\
&= \frac{\sqrt{V[l(u_i)]}R[l(u_i), w(y)] + \sigma(e_i)\rho(e_i, y)}{\sqrt{V\left[l_i^t(v)\right]}}.
\end{aligned}
$$

where $\rho(e_i, y)$ is the correlation coefficient between $w(e_i)$ and $w(y)$ of arcs e_i and y in A_S.

Step 6: Instead of $l_i(v) = \max[l_1^t(v), l_2^t(v), \cdots, l_i^t(v)]$, we recursively calculate $l_i(v) = \max\left[l_{i-1}(v), l_i^t(v)\right]$ for $i \geq 2$

and $l_1(v) = l_1^t(v)$. Using the probabilistic equivalent of $l_i(v) = \max\left[l_{i-1}(v), l_i^t(v)\right]$ ("max" operation), the mean $E[l_i(v)]$ and the variance $V[l_i(v)]$ of $l_i(v)$ are calculated as follows.

$$\begin{aligned}
&E[l_i(v)] \\
&= E[l_{i-1}(v)]\Phi(\alpha) + E[l_i^t(v)]\Phi(-\alpha) + a\varphi(\alpha), \\
&V[l_i(v)] \\
&= \left(E^2[l_{i-1}(v)] + V[l_{i-1}(v)]\right)\Phi(\alpha) \\
&\quad + \left(E^2[l_i^t(v)] + V[l_i^t(v)]\right)\Phi(-\alpha) \\
&\quad + \left(E[l_{i-1}(v)] + E[l_i^t(v)]\right)a\varphi(\alpha) - E^2[l_i(v)], \\
&\alpha = \frac{E[l_{i-1}(v)] - E[l_i^t(v)]}{a}, \\
&a = \sqrt{\begin{array}{l} V[l_{i-1}(v)] + V[l_i^t(v)] \\ -2\sqrt{V[l_{i-1}(v)]}\Big(\sqrt{V[l(u_i)]}R[l_{i-1}(v), l(u_i)] \\ +\sigma(e_i)R[l_{i-1}(v), w(e_i)]\Big) \end{array}},
\end{aligned}$$

where $\varphi(x)$ and $\Phi(x)$ are the probability density function and the cumulative distribution function of a stochastic variable x having the normal distribution, respectively.
In addition, the correlation coefficients $R[l_i(v), l(x)]$ and $R[l_i(v), w(y)]$ between $l_i(v) = \max\left[l_{i-1}(v), l_i^t(v)\right]$ and $l(x)$ of each node x whose $l(x)$ has been already computed and $w(y)$ of each arc y, respectively, are calculated as follows.

$$\begin{aligned}
&R[l_i(v), l(x)] \\
&= \frac{\begin{array}{l}\sqrt{V[l_{i-1}(v)]}R[l_{i-1}(v), l(x)]\Phi(\alpha) \\ \quad + \sqrt{V[l_i^t(v)]}R[l_i^t(v), l(x)]\Phi(-\alpha)\end{array}}{\sqrt{V[l_i(v)]}}, \\
&R[l_i(v), w(y)] \\
&= \frac{\begin{array}{l}\sqrt{V[l_{i-1}(v)]}R[l_{i-1}(v), w(y)]\Phi(\alpha) \\ \quad + \sqrt{V[l_i^t(v)]}R[l_i^t(v), w(y)]\Phi(-\alpha)\end{array}}{\sqrt{V[l_i(v)]}}.
\end{aligned}$$

By computing $E[l_i(v)]$, $V[l_i(v)]$, $R[l_i(v), l(x)]$ and $R[l_i(v), w(y)]$ recursively, we have the mean $E[l(v)] = E[l_{f(v)}(v)]$ and the variance $V[l(v)] = V[l_{f(v)}(v)]$ of $l(v)$, correlation coefficients $R[l(v), l(x)] = R[l_{f(v)}(v), l(x)]$ between $l(v)$ and $l(x)$ of each node x whose $l(x)$ has been already computed, and correlation coefficients $R[l(v), w(y)] = R[l_{f(v)}(v), w(y)]$ between $l(v)$ and $w(y)$ of each arc y. Go to Step 3. ■

This algorithm selects a node v in topological order of the precedence relation given by G_S, and computes the mean $E[l(v)]$ and the variance $V[l(v)]$ of the longest path length $l(v)$ using "add" and "max" operations. In Step 6, to obtain a, we need the correlation coefficient $R\left[l_{i-1}(v), l_i^t(v)\right]$ between $l_{i-1}(v)$ and $l_i^t(v)$ as follows.

$$R\left[l_{i-1}(v), l_i^t(v)\right] = \frac{\begin{array}{l}\sqrt{V[l(u_i)]}R[l_{i-1}(v), l(u_i)] \\ \quad +\sigma(e_i)R[l_{i-1}(v), w(e_i)]\end{array}}{\sqrt{V[l_i^t(v)]}}.$$

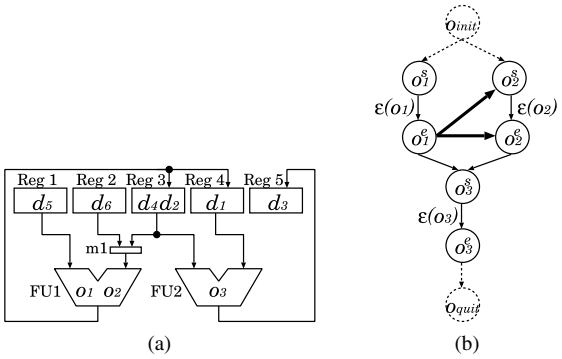


Fig. 5 Datapath (a) and scheduling graph (b).

Thus correlation coefficients $R\left[l_i^t(v), l(x)\right]$ and $R\left[l_i^t(v), w(y)\right]$ in Step 5 and $R[l_i(v), l(x)]$ and $R[l_i(v), w(y)]$ in Step 6 are also computed.

As a demonstrative example, consider the mean $E[l(o_{quit})]$ and the standard deviation $\sqrt{V[l(o_{quit})]}$ of the longest path length $l(o_{quit})$ of the sink o_{quit} in Fig. 5(b). Figure 5(a) shows an example of a datapath, i.e., three operations and six data in Fig. 2(a) are assigned to two functional units FU1 and FU2, and five registers Reg1, Reg2, Reg3, Reg4, and Reg5, respectively. Figure 5(b) shows an example of the scheduling graph for the datapath in Fig. 5(a), which is obtained by adding two disjunctive arcs (o_1^e, o_2^s) and (o_1^e, o_2^e) to the scheduling graph in Fig. 2(b). The delays of functional units FU1 and FU2 in Fig. 5(a) are modeled by the normal distribution $N(9, 13.44)$. Now operations o_1 and o_2 are assigned to the same functional unit FU1, and we set the correlation coefficient $\rho(e_1, e_2)$ between $w(e_1)$ and $w(e_2)$ for arcs $e_1 = (o_1^s, o_1^e)$ and $e_2 = (o_2^s, o_2^e)$ to 1. The other correlation coefficients are set to 0. Then, $E[l(o_{quit})]$ and $\sqrt{V[l(o_{quit})]}$ obtained by the Monte Carlo simulation are 26.96 and 8.12, respectively. The results of the statistical delay analysis are 27.02 and 8.15, respectively, and they are close to the results of the Monte Carlo simulation. On the other hand, if the above correlations are ignored, the results are 27.23 and 6.09, respectively. Thus, for accurate computation, we cannot ignore these correlations even for such a small instance.

By using this statistical delay analysis, in this paper, we define statistical ASAP instantiation $\mathcal{S}_{\mathrm{ASAP}}(v)$ as the mean $E[l(v)]$ of the longest path length $l(v)$ from a source node o_{init} to a node v. On the other hand, statistical ALAP instantiation $\mathcal{S}_{\mathrm{ALAP}}(v)$ is defined as a constant T minus the mean $E[l(v)]$ of the longest path length $l(v)$ from a sink node o_{quit} to a node v, where the directions of arcs in G_S are reversed.

4. Asynchronous Datapath Synthesis with Statistical Makespan Analysis

4.1 Motivation

The statistical schedule length depends not only on delay variations of modules and the correlations induced by various sources but also on resource assignment and schedule (a

selection of disjunctive arcs) strongly. Figure 6 shows a simple example of the statistical schedule length, which varies depending on functional unit assignment. The scheduling graph under the assumption that operations o_1 and o_2 are assigned to FU1 and FU2, respectively, is shown in Fig. 6 (a). Now we will consider functional unit assignment of o_3 to one of the allocated modules (i.e., FU1 and FU2). Figure 6 (b) shows the scheduling graph having the unambiguous sequence o_1o_3 on FU1 in the case where o_3 is assigned to FU1. Similarly, Fig. 6 (c) shows the scheduling graph having the sequence o_2o_3 on FU2 in the other case. Then, these statistical schedule length $l(o_{quit})$ in Figs. 6(b) and (c) are $N(18.57, 42.65)$ and $N(20.07, 36.03)$, respectively. (We have used the same delays and correlation coefficients with the example shown in Sect. 3.) The mean total computation time $E[l(o_{quit})]$ varies from 18.57 to 20.07 depending on the functional unit assignment. Note that, under the constant delay model, both of the typical total computation times in Figs. 6(b) and (c) are 18. Therefore we need to perform carefully resource assignment and scheduling for statistical performance optimization.

For statistical performance optimization problems, our basic strategy is to simultaneously perform scheduling and resource assignment, which is based on assignment driven approach. Roughly speaking, existing methods [2]–[4] are also categorized into "simultaneous scheduling and resource assignment" with constant delay analysis, but they consume much time because of their branch-and-bound based or simulated annealing based exploration of solution space. If statistical delay analysis shown in Sect. 3, instead of constant delay analysis, is incorporated into the conventional asynchronous datapath synthesis systems, they consume more time than the original systems (the time complexity of the statistical delay analysis algorithm proposed in [8] is $\Theta(|A_S|^2)$ while the complexity of constant delay analysis is $O(|A_S|)$). Thus we propose a heuristic method so that it can generate a solution quickly even if the statistical delay analysis is incorporated.

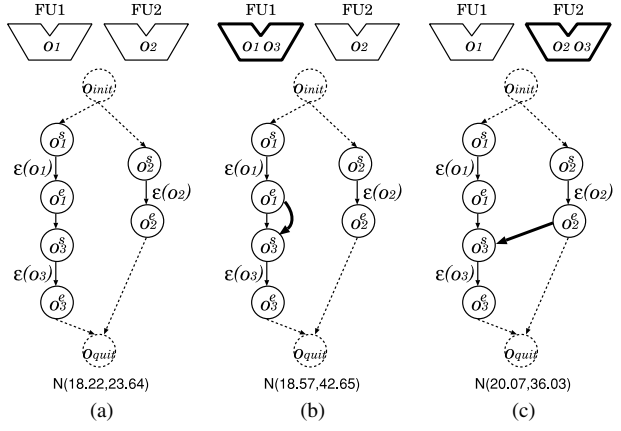


Fig. 6 The statistical schedule length affected by functional unit assignment.

4.2 Proposed Algorithm

In this paper we propose a synthesis system, which can generate asynchronous datapaths having good statistical performances. Now, we consider the problem to find a datapath and a schedule with minimum mean total computation time (i.e., minimum $E[l(o_{quit})]$) with maximum total computation time $T_{max} \leq T_M$ under given set of available modules and constant T_M (the upper bound of maximum total computation time T_{max}). Note that T_{max} is computed using maximum execution time of each operation. Note also that the choice of $E[l(o_{quit})]$ is not essential, but the choice of statistical measure is. Actually, other performance measures, such as $E[l(o_{quit})]+3\sqrt{V[l(o_{quit})]}$, $E[l(o_{quit})]$ plus interconnection complexity between modules, etc., would be candidates for the objective function.

Figure 7 shows the proposed algorithm (SAS). Basically, one operation (data) which is not assigned yet, and one functional unit (register) from given available modules are selected, and the operation (register) is assigned to the functional unit (register). Then the unambiguous sequences of operations and data, which are induced by this resource assignment, are fixed. The above procedure will be repeated until all operations and data are assigned to functional units and registers. Finally all unresolved disjunctive arcs are

Algorithm: Statistical Analysis driven Synthesis (SAS)

Input: A dependence graph G, a set of available modules, and T_M.

Output: A datapath and a schedule.

Step 1: Construct an initial scheduling graph G_S from a dependence graph G.

Step 2: Compute statistical ASAP instantiation $\mathcal{S}_{\mathrm{ASAP}}$ and statistical ALAP instantiation $\mathcal{S}_{\mathrm{ALAP}}$, and calculate the statistical range $\mathcal{S}_{\mathrm{ALAP}}(o_i^s) - \mathcal{S}_{\mathrm{ASAP}}(o_i^s)$ of each operation o_i and the statistical range $\mathcal{S}_{\mathrm{ALAP}}(o_j^e) - \mathcal{S}_{\mathrm{ASAP}}(o_j^e)$ of each data d_j which is generated by the operation o_j.

Step 3: Find the operation having the smallest statistical schedule range among operations which are not assigned to functional units yet. If all operations are assigned to the functional units, find the data having the smallest statistical schedule range among data which are not assigned to registers yet.

Step 4: For the selected operation (data), evaluate the mean total computation time $E[l(o_{quit})]$ and T_{max} for each possible assignment of the operation (data) to one of available functional units (registers), and find the best functional unit (register) which achieves minimum $E[l(o_{quit})]$ while $T_{max} \leq T_M$ is maintained.

If the best functional unit (register) achieves minimum $E[l(o_{quit})] = \infty$, output "FAIL."

Step 5: The operation (data), which is selected in Step 3, is assigned to the functional unit (register) which is selected in Step 4. Then, the unambiguous sequences of operations and data, which are induced by this resource assignment, are fixed, and the corresponding disjunctive arcs are added to G_S.

If all operations and data are assigned to functional units and registers, respectively, then go to Step 6. Otherwise go to Step 2.

Step 6: Fix all unresolved disjunctive arcs by assignment constrained scheduling, and then output a datapath and a schedule.

Fig. 7 Statistical analysis driven synthesis algorithm.

fixed.

4.3 Module Assignment and Assignment Constrained Scheduling

As shown in Fig. 6, the statistical schedule length strongly depends on resource assignment and schedule (a selection of disjunctive arcs). In Step 4 of SAS, we will repeatedly evaluate the statistical schedule length for different assignment of a selected operation (data) to an available functional unit (register), and find the best module. During this process, we use the concept of assignment constrained scheduling. That is, the evaluation of $E[l(o_{quit})]$ is done by temporarily solving the assignment (partial assignment) constrained scheduling problem, which is described as follows. Given G_S, T_M, and (partial or full) resource assignment, find a set of disjunctive arcs with respect to all pairs of operations assigned to the same functional unit and all pairs of data assigned to the same register such that the resultant scheduling graph achieves the minimum $E[l(o_{quit})]$ and $T_{max} \leq T_M$ over all feasible sets of disjunctive arcs.

In order to solve this problem, we use an extended version of the assignment constraint scheduling in [4]. The assignment constrained scheduling algorithm is shown in Fig. 8. The scheduling algorithm incrementally adds disjunctive arcs induced from specified resource assignment to G_S. The choice of disjunctive arcs is done in two ways, one is mandatory and the other is heuristic.

(A) Unambiguous sequence

With respect to a pair of operations o_i and o_j which are assigned to the same functional unit, if addition of the disjunctive arc (o_j^e, o_i^s) forms a positive cycle in the scheduling graph, we call $o_i o_j$ as the unambiguous sequence. If $o_i o_j$ is the unambiguous sequence, we choose the disjunctive arc (o_i^e, o_j^s) and discard (o_j^e, o_i^s).

The unambiguous sequence of data is defined in a similar way, and if $d_i d_j$ is the unambiguous sequence of data d_i and d_j, we choose $(s_x(o_i)^e, o_j^e)$ and discard $(s_y(o_j)^e, o_i^e)$, where o_i generates d_i, o_j generates d_j, $s_x(o_i)$ uses d_i as input, and $s_y(o_j)$ uses d_j as input.

(B) Heuristics

If neither $o_i o_j (d_i d_j)$ nor $o_j o_i (d_j d_i)$ is the unambiguous sequence, we choose disjunctive arcs in the following heuristic way.

First we compute $\mathcal{SL}(o_i, o_j)(\mathcal{SL}(d_i, d_j))$ as follows.

$$\begin{aligned} \mathcal{SL}(o_i, o_j) &= \sigma_{\mathrm{ALAP}}(o_j^s) - \sigma_{\mathrm{ASAP}}(o_i^e), \\ \mathcal{SL}(d_i, d_j) &= \sigma_{\mathrm{ALAP}}(o_j^e) - \max_x \left[\sigma_{\mathrm{ASAP}}(s_x(o_i)^e)\right], \end{aligned}$$

where o_j generates d_j and $s_x(o_i)$ uses d_i as input. Note that ASAP instantiation $\sigma_{\mathrm{ASAP}}(v)$ is defined as the longest path length from o_{init} to v under the maximum execution time for each operation. On the other hand, ALAP instantiation $\sigma_{\mathrm{ALAP}}(v)$ is defined as a constant T minus the longest path length from o_{quit} to v under the maximum execution time for each operation, where the directions of arcs in G_S are reversed. (Use of $\mathcal{S}_{\mathrm{ASAP}}$ and $\mathcal{S}_{\mathrm{ALAP}}$ instead of σ_{ASAP} and $\sigma_{\mathrm{ALAP}}(v)$ is also possible. Here we use σ_{ASAP} and $\sigma_{\mathrm{ALAP}}(v)$ because of its low computational complexity and the request of $T_{max} \leq T_M$.)

Then we find minimum $\mathcal{SL}(a, b)$ and fix the sequence ba. That is, we add the corresponding disjunctive arcs to G_S.

Algorithm: Assignment Constrained Scheduling

Input: A scheduling graph G_S, (partial or full) resource assignment, and T_M.

Output: $E[l(o_{quit})]$ and the list of disjunctive arcs when it is specified.

Step 1: Calculate maximum total computation time T_{max} on G_S. If a positive cycle is detected or $T_{max} > T_M$, output $E[l(o_{quit})] = \infty$ and terminate.

Step 2: If disjunctive arcs between every pair of operations (data) assigned to the same functional unit (register) are fixed, compute $E[l(o_{quit})]$ on G_S and terminate with output $E[l(o_{quit})]$. When it is specified, output also the list of disjunctive arcs.

Step 3: If there exist the unambiguous sequences of operations (data) assigned to the same functional unit (register), then add the corresponding disjunctive arcs to G_S and go to Step 1.

Step 4: Based on the heuristic rule, fix the sequence of one pair of operations (data) which are not scheduled. Then add the corresponding disjunctive arcs to G_S and go to Step 1.

Fig. 8 Assignment constrained scheduling algorithm.

Once disjunctive arcs between every pair of operations (data) assigned to the same functional unit (register) are fixed, then $E[l(o_{quit})]$ on G_S is computed by the statistical schedule length analysis.

When the assignment constrained scheduling is called at Step 4 in the first loop of SAS, it receives an initial scheduling graph constructed from a given dependency graph without any disjunctive arcs, and incrementally adds disjunctive arcs induced from specified resource assignment to the scheduling graph. When the scheduling algorithm is called in the following time of Step 4 or in Step 6, it starts with the scheduling graph which has several disjunctive arcs added in Step 5 of SAS. These disjunctive arcs (i.e., a partial schedule) mean the unambiguous sequences of operations and data induced by resource assignment which has been already fixed in SAS. The scheduling algorithm called in SAS takes a previously fixed partial schedule into account, and it runs more efficiently than a scheduling algorithm without taking any partial schedule into account. Note that G_S in the assignment constrained scheduling is local data, and the modification on G_S done in the assignment constrained scheduling does not affect the scheduling graph G_S in SAS.

The time complexity of the scheduling algorithm is evaluated as follows. Step 2 needs $O(|V_G|^3)$ because the longest past length of every pair of nodes is computed using Floyd's algorithm in order to detect positive cycles and

the unambiguous sequences. Step 4 and 5 are computed in $O(|V_G|^2)$. Since these processes are repeated until all operations and data are scheduled (at most $O(|V_G|^2)$), the total time complexity is $O(|V_G|^5)$.

4.4 Computational Complexity

The computational complexity of the proposed algorithm is evaluated as follows. To select one operation (data) in Step 2, the statistical range of operations and data are computed by the statistical delay analysis, and the computation time is $O(|A_G|^2)$. To select one module in Step 4, the statistical schedule length for different assignments (at most M) of a selected operation (data) is evaluated by assignment constrained scheduling, where M is the number of given available modules, and the time complexity of the assignment constrained scheduling is $O(|V_G|^5)$. Hence the computation time is $O(M \cdot |V_G|^5)$. Step 5 needs $O(|V_G|^5)$ because the assignment constrained scheduling is partly used. Since these processes are repeated until all operations and data (at most $|V_G|$) are assigned, the total time complexity is $O(M \cdot |V_G|^6)$.

5. Experiments

The proposed algorithm is implemented using C program language on a 1 GHz Pentium III personal computer. The algorithm is applied to three datapath synthesis benchmarks; four-order all-pole lattice filter (which contains 11 additions, 4 multiplications, 16 variables, and 4 constants), fifth-order elliptic wave filter (which contains 26 additions, 8 multiplications, 35 variables, and 8 constants), and twice unfolded fifth-order elliptic wave filter in which two original iterations are combined as a new single iteration (which contains 52 additions 16 multiplications, 69 variables, and 16 constants).

For comparison purposes, datapaths and schedules are also synthesized by using asynchronous datapath synthesis systems in [2]–[4] and [8]. Systems in [2]–[4] find datapaths and schedules with minimum typical total computation time T_{typi} with $T_{max} \leq T_M$. On the other hand, the system in [8] synthesizes datapaths and schedules with minimum mean total computation time $E[l(o_{quit})]$ with $T_{max} \leq T_M$ by SA based exploration of assignment solution space with the statistical delay analysis. All synthesis results are evaluated by the Monte Carlo simulation, and we obtain the mean of the total computation time.

5.1 Results

Tables 1, 2, and 3 show the results (column E) obtained with several different specifications of the numbers of adders (Add1 and Add2), multipliers (Mul1 and Mul2) and registers (Reg) and the upper bound of maximum total computation time T_M for each benchmark. T_M is set to the minimum T_{max} under given available modules for the four-order all-pole lattice filter and the fifth-order elliptic wave filter. For the twice unfolded fifth-order elliptic wave filter, we could not find (guarantee) the minimum T_{max}, and we use the smallest T_{max} found so far as T_M. The column of "time" shows runtime in seconds. We use the module library in [10], and model the delays of Add1, Add2, Mul1, and Mul2 with $N(7.5, 0.69)$, $N(15, 2.78)$, $N(16, 2.78)$, and $N(43.5, 23.36)$, respectively. If two operations o_i and o_j are assigned to the same functional unit, we set the correlation coefficient $\rho(e_i, e_j) = 1$ between $w(e_i)$ and $w(e_j)$ for two arcs $e_i = (o_i^s, o_i^e)$ and $e_j = (o_j^s, o_j^e)$ in a scheduling graph G_S. Otherwise the correlation coefficient is set to 0.0, 0.3, 0.6, or 0.9, which is shown in the column of "Corr."

As we can see from the column E, our system always provides better solutions than [2]–[4]. Unfortunately, our

Table 1 Experimental results for four-order all-pole lattice filter. Fast module set Add1: $N(7.5, 0.69)$, Mul1: $N(16, 2.78)$. Slow module set Add2: $N(15, 2.78)$, Mul2: $N(43.5, 23.36)$. [2] and [3] don't use the hierarchical approach. [2], [3] and [4] generate optimal solutions in terms of the minimization of T_{typi}.

Add1	Mul1	Reg	T_M [ns]	Corr	[2]			[3]			[4]			[8]		Ours	
					T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
4	2	6	82	0.0	62	63.90	≤1	62	63.90	≤1	62	65.25	≤1	63.63	1	63.63	≤1
				0.3		63.60			63.60			64.72		63.39	1	63.39	≤1
				0.6		63.60			63.60			64.72		63.39	1	63.39	≤1
				0.9		62.62			62.62			63.05		62.55	1	62.55	≤1
2	2	6	92	0.0	69.5	72.41	≤1	69.5	72.41	≤1	69.5	72.88	≤1	70.89	1	71.69	≤1
				0.3		71.96			71.96			72.32		70.65	1	71.33	≤1
				0.6		71.32			71.32			71.62		70.37	1	70.87	≤1
				0.9		70.40			70.40			70.55		69.92	1	70.17	≤1
Add2	Mul2	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
4	2	6	196	0.0	147	151.46	≤1	147	151.46	≤1	147	154.90	≤1	151.07	1	151.07	≤1
				0.3		150.75			150.75			153.63		150.44	1	150.44	≤1
				0.6		149.86			149.86			152.03		149.55	1	149.62	≤1
				0.9		148.47			148.47			149.56		148.24	1	148.35	≤1
2	2	6	216	0.0	162	169.45	≤1	162	169.45	≤1	162	170.39	≤1	166.99	1	166.99	≤1
				0.3		168.21			168.21			169.00		163.53	1	166.15	≤1
				0.6		166.67			166.67			167.27		163.12	1	165.11	≤1
				0.9		164.30			164.30			164.60		162.83	1	163.52	≤1

Table 2 Experimental results for fifth-order elliptic wave filter. Fast module set Add1: $N(7.5, 0.69)$, Mul1: $N(16, 2.78)$. Slow module set Add2: $N(15, 2.78)$, Mul2: $N(43.5, 23.36)$. [2] and [3] use the hierarchical approach, and set the maximum block size at 15. For two instances (Add1(Add2)=3, Mul1(Mul2)=3, Reg=13, T_M=174(412)), [2] and [3] generate optimal solutions in terms of the minimization of T_{typi}. For the other instances, all solutions generated by [2] or [3] within 24 hours are not comparable ones to ours. For the latter set of instances, [2] and [3] don't halt within 24 hours if the maximum block size is greater than 15. [4] generates optimal solutions in terms of the minimization of T_{typi} for all instances.

Add1	Mul1	Reg	T_M [ns]	Corr	[2]			[3]			[4]			[8]		Ours	
					T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	174	0.0	131.5	134.68	22	131.5	134.68	11	131.5	137.24	19	132.32	72	132.42	7
				0.3		134.01			134.01			136.22		132.20	56	132.22	7
				0.6		133.20			133.20			134.97		131.92	93	132.00	7
				0.9		133.20			133.20			133.07		131.60	59	131.72	7
2	2	13	184	0.0	-	-	1941	-	-	386	131.5	137.20	26	136.08	200	136.51	8
				0.3		-			-			136.10		135.19	225	135.45	8
				0.6		-			-			134.75		133.75	258	134.55	8
				0.9		-			-			132.83		132.33	170	132.63	8
2	1	13	218	0.0	-	-	> 24h	-	-	> 24h	165.5	168.75	31	167.71	200	167.71	7
				0.3		-			-			168.01		167.23	786	167.23	8
				0.6		-			-			167.14		166.77	93	166.77	8
				0.9		-			-			166.02		165.95	122	165.96	8
Add2	Mul2	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	412	0.0	309	311.59	21	309	311.59	13	309	319.65	16	310.53	68	310.53	7
				0.3		311.11			311.11			317.65		309.83	59	309.87	7
				0.6		310.73			310.73			315.34		309.55	59	309.70	7
				0.9		310.41			310.41			312.18		309.40	60	309.55	7
2	2	13	432	0.0	-	-	1924	-	-	384	309	318.33	14	316.02	225	318.03	9
				0.3		-			-			316.69		313.68	119	315.97	8
				0.6		-			-			314.73		314.20	182	314.21	8
				0.9		-			-			311.80		310.66	150	311.72	8
2	1	13	564	0.0	-	-	> 24h	-	-	> 24h	423	425.84	33	425.80	185	425.80	9
				0.3		-			-			425.28		425.21	132	425.26	8
				0.6		-			-			424.67		424.67	130	424.67	8
				0.9		-			-			423.79		423.79	98	423.79	7

results cannot be compared with the results of [2], [3] for larger benchmarks, since [2] and [3] do not produce comparable solutions within 24 hours on our PC environment. On the other hand, [4] generates datapaths and schedules with minimum T_{typi} for four-order all-pole lattice filter and fifth-order elliptic wave filter.

For the case of small target algorithms or small number of functional units, the performance difference between datapaths designed by our system and a conventional one seems not so large. However, our system tends to generate better solutions than a conventional one in the mean total computation time, when the size of a target algorithm becomes larger, or the number of functional units becomes larger. Especially, Table 3 shows solutions with the maximum 2.14% reduced mean total computation time by comparison with [4]. Also it shows our system runs about 35.12 ∼ 147.57 times faster than [4].

The effect of the variance of the delay is tested in the next experiment. Now we use Add3: $N(15, 1.34)$, Mul3: $N(43.5, 11.68)$ as a small variance module set, where each variance is set as half of the variance of Add2 or Mul2. We also use Add4: $N(15, 5.46)$ and Mul4: $N(43.5, 46.72)$ as a large variance module set, where each variance is set as double of the variance of Add2 or Mul2. Tables 4, 5, and 6 show the results for each benchmark. Our system tends to generate better solutions in the mean total computation time, when the variance of the delay of each module becomes larger. Table 5 shows solutions with the maximum 4.57% reduced mean total computation time by comparison with [4].

On the other hand, our system always provides comparable solutions to [8], and [8] cannot produce any solution for larger benchmarks within 24 hours on our PC environment. Thus our system runs faster than existing synthesis systems of asynchronous datapath, and it can find near optimum solutions in most cases.

5.2 Comments on Synthesis Systems in [2] and [3]

To obtain datapaths and schedules with minimum T_{typi}, [2] uses redundant edge filter and minimum latency filter, and [3] uses these filters and one direction filter. However, by only those filters, we cannot find a feasible solution within 24 hours for the fifth-order elliptic wave filter and the twice unfolded fifth-order elliptic wave filter. In addition to those filters, the hierarchical approach in [2] is a powerful technique for reducing the solution space. The hierarchical approach randomly groups operations of the same type into

Table 3 Experimental results for twice unfolded fifth-order elliptic wave filter. Fast module set Add1: $N(7.5, 0.69)$, Mul1: $N(16, 2.78)$. Slow module set Add2: $N(15, 2.78)$, Mul2: $N(43.5, 23.36)$. [2] and [3] use the hierarchical approach, and set the maximum block size at 15. For two instances (Add1(Add2)=3, Mul1(Mul2)=3, Reg=13, T_M=348(824)), [2] and [3] halt within 2800[s] and 1200 [s], respectively, but they did not generate comparable solutions to ours. For the other instances, [2] and [3] don't halt within 24 hours on our PC environment. [2] and [3] don't halt within 24 hours for all instances if the maximum block size is greater than 15. Also [8] don't halt within 24 hours for all instances. So, we omit the columns for [2], [3] and [4].

					[4]			Ours	
Add1	Mul1	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	348	0.0	254.5	266.42	3042	260.20	103
				0.3		264.46		259.33	105
				0.6		262.04		258.27	105
				0.9		258.46		256.71	103
3	2	13	358	0.0	263	274.30	4704	269.12	106
				0.3		272.44		268.23	108
				0.6		270.15		267.12	108
				0.9		266.82		265.48	106
2	2	13	368	0.0	269.5	283.27	10365	281.46	138
				0.3		280.97		279.45	141
				0.6		278.16		276.98	141
				0.9		274.04		273.41	141
Add2	Mul2	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	824	0.0	589.5	611.58	3652	598.79	103
				0.3		607.72		596.86	103
				0.6		603.18		595.28	103
				0.9		596.78		592.84	103
3	2	13	844	0.0	618	640.41	4277	631.30	104
				0.3		636.90		629.56	104
				0.6		632.51		626.97	104
				0.9		625.73		622.94	104
2	2	13	864	0.0	619.5	650.27	20365	645.23	138
				0.3		645.22		641.00	138
				0.6		638.96		635.78	138
				0.9		629.40		626.03	138

blocks of a given maximum size, and then the solution space exploration is done separately for each block. In our experiments, the hierarchical approach is also applied to the fifth-order elliptic wave filter and the twice unfolded fifth-order elliptic wave filter. The maximum block size is set to 15, which is as large value as possible for the system to halt within 24 hours on our PC environment. Note that we also tested smaller maximum block sizes from 2 to 14. Their runtimes spread from 1 second to more than 24 hours on our PC environment. However, all of their solutions generated within 24 hours are not comparable to ours.

The runtimes of [2] and [3] increase when T_M becomes larger. This is because the chance of bounding by the minimum latency filter becomes smaller. As for the hierarchical approach, the runtime and the quality of solutions depend on not only the maximum block size but also the grouping of operations into blocks. Thus, if the grouping is not adequate, no feasible (or good) solutions will be generated even if a larger maximum block size is used. Unfortunately, [2] didn't discuss how to group operations into blocks more than random grouping. In our experiments, we also used random grouping.

5.3 Comments on Differences between Previous Systems and Ours

Finally, we briefly discuss differences between [2], [3] (Mercury and Mercury based synthesis Systems, in the following MS in short) and [4], [8] and our system (Assignment driven synthesis Systems, AS in short) with regard to the efficiency as a synthesis system.

1. The central optimization procedure of MS is the branch-and-bound search of the schedule solution space, and that of AS is the simulated annealing search ([4], [8]) or the constructive search (this paper) of the assignment solution space. The performance difference especially the difference in the computation time would be caused partly by the difference of the underlying exploration method.

2. The central optimization procedure of MS is the branch-and-bound search of the schedule solution space, and that of AS is the simulated annealing search ([4], [8]) or the constructive search (this paper) of the assignment solution space. As a result, the assignment is applied to a generated schedule in MS, while the scheduling is applied to a generated assignment in AS. In our experiments done in this paper, the numbers of functional units and registers are specified as design constraints. In AS, we can efficiently utilize those resource constraints for exploring the assignment solution space. However, in MS, those resource constraints can hardly help the efficiency of the branch-and-bound search of the schedule solution space, since the more disjunctive arcs are (i.e., the deeper the branches proceed), the higher the chance of resource sharing is. Thus, it is difficult to generate solutions satisfying resource constraints efficiently.

6. Conclusions

In this paper, we have proposed an effective asynchronous datapath synthesis system to optimize statistical schedule length using statistical schedule length analysis. The proposed algorithm is a heuristic method which simultaneously performs scheduling and resource assignment. During the design process, decisions have been made based on the statistical schedule length analysis. Our system generates schedules and datapaths having higher statistical performances, which are not synthesized by using conventional ones.

The normal distribution is not always adequate for the delay analysis of asynchronous datapaths. Development of a proper model and algorithms to compute statistical parameters, which reflect practical random delay variations, are left for future work.

In this paper, operation delays are mainly considered,

Table 4 Experimental results for four-order all-pole lattice filter. Small variance module set Add3: $N(15, 1.34)$, Mul3: $N(43.5, 11.68)$. Large variance module set Add4: $N(15, 5.46)$, Mul4: $N(43.5, 46.72)$. [2] and [3] don't use the hierarchical approach. [2], [3] and [4] generate optimal solutions in terms of the minimization of T_{typi}.

Add3	Mul3	Reg	T_M [ns]	Corr	[2]			[3]			[4]			[8]		Ours	
					T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
4	2	6	176	0.0	147	150.14	≤1	147	150.14	≤1	147	152.57	≤1	149.87	1	149.87	≤1
				0.3		149.64			149.64			151.67		149.43	1	149.43	≤1
				0.6		149.01			149.01			150.55		148.85	1	148.85	≤1
				0.9		148.03			148.03			148.80		147.95	1	147.95	≤1
2	2	6	194	0.0	162	167.26	≤1	162	167.26	≤1	162	167.91	≤1	165.51	1	165.51	≤1
				0.3		166.38			166.38			166.94		164.92	1	164.92	≤1
				0.6		165.30			165.30			165.72		164.19	1	164.19	≤1
				0.9		163.63			163.63			163.83		163.07	1	163.07	≤1
Add4	Mul4	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
4	2	6	216	0.0	147	153.28	≤1	147	153.28	≤1	147	158.13	≤1	152.74	1	152.74	≤1
				0.3		152.28			152.28			156.34		151.85	1	151.85	≤1
				0.6		151.02			151.02			154.09		150.70	1	150.70	≤1
				0.9		149.06			149.06			150.60		148.90	1	148.90	≤1
2	2	6	238	0.0	162	172.51	≤1	162	172.51	≤1	162	173.83	≤1	169.20	1	169.20	≤1
				0.3		170.76			170.76			171.87		167.84	1	167.84	≤1
				0.6		168.59			168.59			169.43		166.38	1	166.38	≤1
				0.9		165.25			165.25			165.67		164.14	1	164.14	≤1

Table 5 Experimental results for fifth-order elliptic wave filter. Small variance module set Add3: $N(15, 1.34)$, Mul3: $N(43.5, 11.68)$. Large variance module set Add4: $N(15, 5.46)$, Mul4: $N(43.5, 46.72)$. [2] and [3] use the hierarchical approach, and set the maximum block size at 15. For two instances (Add3(Add4)=3, Mul3(Mul4)=3, Reg=13, T_M=370(454)), [2] and [3] generate optimal solutions in terms of the minimization of T_{typi}. For the other instances, all solutions generated by [2] or [3] within 24 hours are not comparable ones to ours. For the latter set of instances, [2] and [3] don't halt within 24 hours for all instances if the maximum block size is greater than 15. [4] generates optimal solutions in terms of the minimization T_{typi} for all instances.

Add3	Mul3	Reg	T_M [ns]	Corr	[2]			[3]			[4]			[8]		Ours	
					T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	370	0.0	309	310.76	23	309	310.76	10	309	316.06	18	309.62	59	309.69	7
				0.3		310.61			310.61			314.82		309.53	48	309.88	7
				0.6		310.48			310.48			313.38		309.44	49	309.72	7
				0.9		310.29			310.29			311.24		309.28	50	309.28	7
2	2	13	388	0.0	-	-	1945	-	-	378	309	315.48	25	312.90	151	315.31	8
				0.3		-			-			314.38		312.90	146	314.23	8
				0.6		-			-			313.04		312.68	172	312.68	7
				0.9		-			-			310.97		310.16	126	310.79	7
2	1	13	622	0.0	-	-	> 24h	-	-	> 24h	423	424.94	32	424.94	181	424.94	7
				0.3		-			-			424.43		424.34	217	424.38	7
				0.6		-			-			424.38		424.33	246	424.36	7
				0.9		-			-			423.56		423.49	150	423.56	7
Add4	Mul4	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	454	0.0	309	313.27	20	309	313.27	11	309	325.25	17	310.91	75	311.05	7
				0.3		311.65			311.65			322.10		310.16	75	310.49	7
				0.6		310.36			310.36			318.41		309.81	95	310.02	7
				0.9		310.36			310.36			313.50		309.21	47	309.60	7
2	2	13	476	0.0	-	-	1965	-	-	387	309	322.80	15	316.16	252	322.12	7
				0.3		-			-			322.17		315.63	236	319.71	8
				0.6		-			-			317.16		313.96	189	316.90	8
				0.9		-			-			312.95		312.58	173	312.58	7
2	1	13	622	0.0	-	-	> 24h	-	-	> 24h	423	427.50	32	427.14	110	427.14	7
				0.3		-			-			426.43		426.34	217	426.38	7
				0.6		-			-			425.38		425.35	246	425.36	7
				0.9		-			-			424.11		424.11	184	424.11	7

Table 6 Experimental results for twice unfolded fifth-order elliptic wave filter. Small variance module set Add3: $N(15, 1.34)$, Mul3: $N(43.5, 11.68)$. Large variance module set Add4: $N(15, 5.46)$, Mul4: $N(43.5, 46.72)$. [2] and [3] use the hierarchical approach, and set the maximum block size at 15. For two instances (Add3(Add4)=3, Mul3(Mul4)=3, Reg=13, T_M=740(908)), [2] and [3] halt within 2800[s] and 1200[s], respectively, but they did not generate comparable solutions to ours. For the other instances, [2] and [3] don't halt within 24 hours on our PC environment. [2] and [3] don't halt within 24 hours for all instances if a maximum block size is greater than 15. Also [8] don't halt within 24 hours for all instances. So, we omit the columns for [2], [3] and [8].

					[4]			Ours	
Add3	Mul3	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	740	0.0	589.5	604.59	3052	595.76	106
				0.3		602.15		594.80	103
				0.6		599.23		593.69	108
				0.9		594.89		592.09	107
3	2	13	758	0.0	618	634.08	4715	627.90	105
				0.3		631.61		626.45	106
				0.6		628.53		624.63	106
				0.9		623.74		621.80	107
2	2	13	776	0.0	619.5	641.31	11365	637.75	138
				0.3		637.78		634.80	138
				0.6		633.39		631.16	139
				0.9		626.71		623.51	139
Add4	Mul4	Reg	T_M [ns]	Corr	T_{typi} [ns]	E [ns]	time [s]	E [ns]	time [s]
3	3	13	908	0.0	589.5	622.22	3665	603.75	104
				0.3		616.27		601.01	105
				0.6		609.20		597.88	104
				0.9		599.45		593.91	104
3	2	13	930	0.0	618	649.40	4727	636.91	105
				0.3		644.31		633.94	106
				0.6		638.06		630.26	105
				0.9		628.48		624.59	106
2	2	13	952	0.0	619.5	662.91	20335	655.97	140
				0.3		655.67		649.79	138
				0.6		646.77		642.29	136
				0.9		628.06		623.20	141

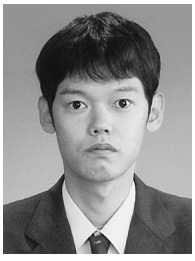
and we propose a framework of statistical delay variation-aware asynchronous datapath synthesis. Currently, the other delays, such as wire delay, multiplexer delay, and controller delay, are included in a limited way, that is, $\varepsilon(o_i)$ is redefined as the sum of operation delay, wire delay, and multiplexer delay. However, more precise treatment of those delays is also left for future work.

Acknowledgments

The authors would like to thank reviewers for their helpful suggestions.

References

[1] J.A. Brzozowski and C.H. Seger, Asynchronous circuits, Springer-Verlag, 1994.
[2] B.M. Bachman, H. Zheng, and C. Merys, "Architectural synthesis of timed asynchronous systems," Proc. International Conf. Computer Design, pp.354–363, 1999.
[3] M. Kawanabe, H. Saito, M. Imai, H. Nakamura, and T. Nanya, "Design space reduction filter in asynchronous data-path synthesis," IEICE Technical Report, SLDM2003-112, 2003.
[4] K. Ohashi and M. Kaneko, "Asynchronous datapath synthesis based on binding space exploration," Proc. 17th Workshop on Circuits and Systems in Karuizawa, pp.549–554, 2004.
[5] P.G. Paulin and J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.8, no.6, pp.661–679, 1989.
[6] R. Camposano and W. Rosenstiel, "Synthesizing circuits from behavioral descriptions," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.8, no.2, pp.171–180, 1989.
[7] H. Saito and T. Yoneda, "Asynchronous data-path circuit synthesis by using force-directed scheduling algorithm and consideration to improve efficiency," IEICE Technical Report, VLD2004-80, 2004.
[8] K. Ohashi and M. Kaneko, "Statistical schedule length analysis in asynchronous datapath synthesis," Proc. IEEE International Symposium on Circuits and Systems, pp.700–703, 2005.
[9] S. Tsukiyama, "Statistical timing analysis: A survey," Proc. 18th Workshop on Circuits and Systems in Karuizawa, pp.533–538, 2005.
[10] H. Tomiyama and H. Yasuura, "Module selection using manufacturing information," IEICE Trans. Fundamentals, vol.E81-A, no.12, pp.2576–2584, Dec. 1998.



Koji Ohashi received the B.E. degree in Electronic engineering from Nagaoka University of Technology in 1998. He received the M.E. and Ph.D. degrees in Information Science from Japan Advanced Institute of Science and Technology (JAIST) in 2000 and 2003, respectively. Since 2003 he has been a researcher in School of Information Science, JAIST. His research interests include CAD for VLSIs and high level synthesis.

Mineo Kaneko received Bachelor of Engineering, Master of Engineering, and Doctor of Engineering degrees in electrical and electronic engineering from Tokyo Institute of Technology in 1981, 1983, and 1986, respectively. From 1986 to 1996, he worked at Tokyo Institute of Technology as a research associate, a lecturer, and an associate professor. In 1996, he transferred to Japan Advanced Institute of Science and Technology (JAIST), and currently he is a professor in the Graduate School of Information Science, JAIST. His research interests include circuit theory, CAD for VLSIs, and signal processing. He is a member of IEEE and ACM.