

# A Scheduling Method for Asynchronous Bundled-Data Implementations Based on The Completion of Data Operations

Hiroshi Saito<sup>1</sup>, Nattha Jindapetch<sup>2</sup>, Tomohiro Yoneda<sup>3</sup>, Chris Myers<sup>4</sup>, Takashi Nanya<sup>5</sup>

<sup>1</sup> The University of Aizu, Japan, <sup>2</sup> Prince of Songkla University, Thailand, <sup>3</sup> National Institute of Informatics, Japan,

<sup>4</sup> The University of Utah, USA, <sup>5</sup> The University of Tokyo, Japan,

E-mail: [hiroshis@u-aizu.ac.jp](mailto:hiroshis@u-aizu.ac.jp), [nattha.s@psu.ac.th](mailto:nattha.s@psu.ac.th), [yoneda@nii.ac.jp](mailto:yoneda@nii.ac.jp), [myers@vlsi@group.ece.utah.edu](mailto:myers@vlsi@group.ece.utah.edu), [nanya@hal.rcast.u-tokyo.ac.jp](mailto:nanya@hal.rcast.u-tokyo.ac.jp)

**Abstract:** This paper presents the MFDS algorithm for scheduling of asynchronous bundled-data implementations. This algorithm is shown to produce nearly the same quality of circuits as the FDS algorithm but in a much shorter time.

## 1. Introduction

In this paper, we propose a scheduling method for asynchronous bundled-data implementations. The proposed method is based on a traditional scheduling method [1] used for synchronous circuit designs.

The main idea of the traditional method is to divide a specified timing constraint into several control steps and to schedule operations to a control step so that the number of required resources is minimum. However, arbitrary decided control steps which imply clock cycles do not fit to scheduling of clock-less asynchronous circuits.

To decide an optimum schedule for asynchronous bundled-data implementations efficiently, we propose a calculation method of control steps based on the completion of operations. Then, we modify the traditional scheduling method so that the schedule of operations is decided by using the control steps calculated in this way. By using these methods, we can obtain a schedule with more or less the same quality of circuits (i.e., the number of resources) in short time compared to the traditional method.

## 2. Basic Notions

**Data Flow Graph:** A data flow graph (DFG) defined by  $G = \langle N, E \rangle$  is the input description for our proposed scheduling algorithm. The DFG represents a data flow in an application's behavioral description. A node  $n_i$  ( $i = 1, \dots, m$ ) in the node set  $N$  represents a data operation. An edge in the edge set  $E$  represents a data dependency between nodes. Each node is labeled by the maximum execution delay (denoted by  $\text{delay}_i$ ) required to execute the corresponding operation. By using data-path resources which are already designed and parameterized by the delay and the area,  $\text{delay}_i$  for each  $n_i$  is obtained. Fig.1(a) shows a DFG.

**Target Architecture:** The target architecture called the bundled-data implementation is shown in Fig.1(b). In the bundled-data implementation, a set of data-path resources to execute the operation for node  $n_i$  is controlled by the control circuit with a pair of handshake signals ( $\text{req}_i$  and  $\text{ack}_i$ ) and selection signals ( $\text{sel}_{i,j}$ ,  $j = 0, \dots, l$ ). To identify the completion of the operation as  $\text{ack}_i$ , a delay element called "matched delay" is put on the wire of signal  $\text{req}_i$ . Interaction with other control circuits is controlled by pairs of handshake signals  $\text{Req}_i$  and  $\text{Ack}_i$ .

## 3. Calculation of Control Steps Based on The Completion of Data Operations

In bundled-data implementations, a data operation is started after the completion of preceding operations. Therefore, the operation is triggered by at least one preceding operation called "trigger operation". To preserve this property, we calculate control steps

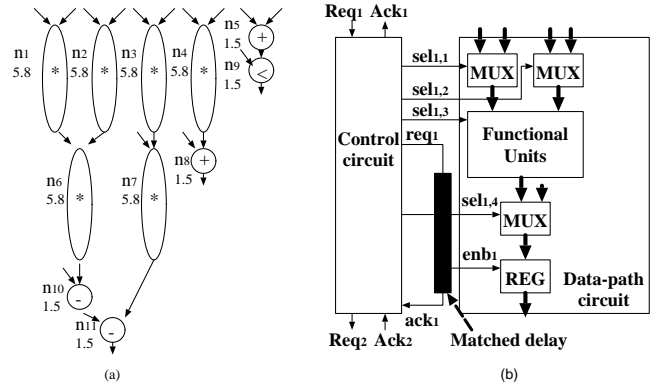


Figure 1. (a) DFG. (b) Target architecture.

for scheduling based on the completion of data operations as follows:

1. Calculation of the as soon as possible (ASAP) and the as late as possible (ALAP) schedules.
2. Estimation of start time and completion time candidates for node  $n_i$  with the identification of trigger nodes.
3. Extraction of control steps.

For a given DFG, the ASAP schedule where operations are executed as soon as possible and the ALAP schedule where operations are executed as late as possible are calculated to identify the critical path delay and operations which can be changed the schedule without changing the critical path delay. For convenience, we denote the ASAP start time, the ASAP completion time, the ALAP start time, and the ALAP completion time of node  $n_i$  as  $\text{asaps}(n_i)$ ,  $\text{asapc}(n_i)$ ,  $\text{alaps}(n_i)$ , and  $\text{alapc}(n_i)$ , respectively (see Fig.2(a)).

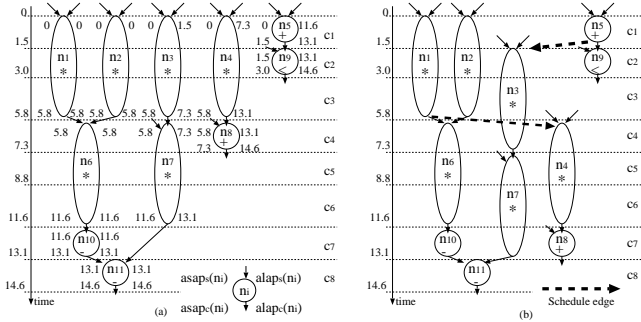
Then, the set of start time candidates  $S_{n_i}$  and the set of completion time candidates  $C_{n_i}$  for each node  $n_i$  are estimated. A start time candidate is denoted by  $\langle \text{stu}, T_u \rangle$  ( $u = 1, \dots, v$ ).  $\text{stu}$  represents a start time for node  $n_i$  and  $T_u$  represents the set of trigger nodes to start the operation of node  $n_i$  from  $\text{stu}$ . A completion time candidate is denoted by  $\text{ctu}$ . There are two ways for the estimation of candidates:

**1. Estimation from  $\text{asapc}(n_j)$  of all nodes  $n_j$  except node  $n_i$ ,  $n_i$ 's transitive predecessors, and  $n_i$ 's transitive successors:** If  $\text{asaps}(n_i) \leq \text{asapc}(n_j) \leq \text{alaps}(n_i)$  is true,  $\text{asapc}(n_j)$  can be regarded as an  $\text{stu}$  for node  $n_i$ . Node  $n_j$  is a trigger node for node  $n_i$  to start from  $\text{asapc}(n_j)$  (i.e.,  $\langle \text{asapc}(n_j), \{n_j\} \rangle$ ).

**2. Estimation from the direct predecessors  $n_j$  of node  $n_i$ :** All  $\text{ctu} \in C_{n_j}$  of direct predecessors  $n_j$  of node  $n_i$  can be regarded as  $\text{stu}$  for node  $n_i$  if  $\text{asaps}(n_i) \leq \text{ctu} \leq \text{alaps}(n_i)$  is true. Node  $n_j$  is a trigger node for node  $n_i$ .

**Table 1. Experimental results.**

Name	N (SN)	Lat.[ns]	FDS			MFDS			
			S	Res. a, m, d	T [s]	S	Res. a, m, d	T [s]	
DIFFEQ	11 (6)	14.6	146	2, 3, 0	1.04	6 – 8	2, 3, 0	0.00	
AR	28 (6)	24.9	249	2, 6, 0	12.19	10 – 13	2, 7, 0	0.04	
EWF	24 (10)	33.9	339	3, 4, 0	4.99	17 – 18	3, 4, 0	0.06	
SSH	21 (9)	46.1	461	1, 2, 2	36.03	16 – 22	2, 2, 2	0.06	
SSV4	76 (44)	160.4	1604	1, 3, 2	2168.02	56 - 186	2, 2, 2	25.09	



**Figure 2. (a) Calculated control steps. (b) Scheduled DFG.**

Finally, control steps denoted by  $cs$  ( $s = 1, \dots, t$ ) are extracted from  $S_{ni}$  of all  $ni$ . Fig.2(a) shows extracted control steps for the DFG shown in Fig.1(a).

Different from control steps used in the scheduling of synchronous circuits where the same fixed time interval is assumed, control steps calculated in this method have different time intervals (i.e.,  $c3$  and  $c6$  have different time interval from others).

#### 4. Force-Directed Scheduling Algorithm for Bundled-Data Implementations

In this section, the overview of our proposed scheduling method for bundled-data implementations is described. For details, please refer to [2].

The proposed method is based on the Force-Directed Scheduling (FDS) algorithm [1] developed by Paulin et al., which is used for synchronous circuit designs. In the FDS algorithm, operations are scheduled so that the number of resources is minimum under a timing constraint (e.g., the critical path delay). Different from other timing constraint scheduling algorithms, the FDS algorithm determines the schedule of operations so that the usage of resources is balanced. It is evaluated by calculating "self force" (i.e., a cost function) for each operation.

In our modified FDS (MFDS) algorithm, control steps decided by the method described in Section 3 are used. Then, for each  $\langle stu, Tu \rangle \in S_{ni}$  of node  $ni$ , the self force is calculated. Note that control step  $cs$  is used instead of  $stu$ . Then, the operation which has the minimum force is scheduled to control step  $cs$ . This operation is repeated until all operations are scheduled.

Fig.2(b) shows a scheduled DFG obtained by our MFDS algorithm where two schedule edges are newly introduced. A schedule edge is newly inserted to a given DFG if there is no data dependence between a node and its trigger node.

#### 5. Experimental Results

In this experiment, we have compared the number of used data-path resources and the scheduling time between our MFDS algorithm and the FDS algorithm shown in [3] where the FDS algorithm is well formalized. We assume that the interval of control steps used for the FDS algorithm is 0.1 ns. This is because

to have the same performance in scheduling results. If we assume a longer time for the interval, the scheduling time will be reduced. However, a performance degradation happens. For example, if we assume that the time interval is 1.5 ns (the execution delay of an adder) and the execution delay of a multiplier is 5.8 ns, 0.2 ns ( $1.5 * 4 - 5.8$ ) is lost. The degradation will be significant when a large number of operations have a different execution delay.

Table 1 shows the experimental result. The column N represents the number of nodes. SN represents the number of scheduled nodes. The column Lat. represents the performance of resulting circuits. The columns S represent the number of control steps. The columns Res. represent the number of data-path resources (a - alu, m - mul, d - div). The columns T represent the scheduling time.

From this result, we can know that in all cases our MFDS algorithm can decide a schedule in a short time. This is because the number of control steps used in the MFDS algorithm is less. Although the number of data-path resources required in resulting circuits is different in several cases, the difference is not so big. As a result, we can conclude that our MFDS algorithm can decide a schedule with nearly the same quality of circuits in a shorter time compared to the FDS algorithm.

#### 6. Conclusion

In this work, we proposed a scheduling method for asynchronous bundled-data implementations to determine the schedule of operations efficiently.

#### Acknowledgment

This work is supported by Grant-in-Aid for Scientific Research from Japan Society for the Promotion of Science (#16700050).

#### References

- [1] P.Paulin and J.Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's", IEEE Transactions on Computer-Aided Design, pp.661--679, vol.8, no.6, 1989.
- [2] H.Saito, N.Jindapetch, T.Yoneda, C.Myers, and T.Nanya, "A Scheduling Method for Asynchronous Bundled-Data Implementations", International Workshop on Logic and Synthesis, 2005.
- [3] W.Verhaegh, E.Aarts, J.Korst, and P.Lippens, "Improved Force-Directed Scheduling", European Design Automation Conference, pp.430--435, 1991.