2005－SLDM－122 (11)
2005／12／1

実行可能スケジュール・資源割当のための極小生存期間重なり集合

金子 峰雄†

† 北陸先端科学技術大学院大学 情報科学研究科 〒923–1292 石川県能美市旭台 1-1
E-mail: †mkaneko@jaist.ac.jp

あらまし 多くの高位合成システムでは，演算スケジュールの終了後に資源割り当てを行うため，資源割り当て時に生存期間が全て確定しており，衝突のない資源共有が容易に行われる．これに対して，資源割り当てをスケジュールに先行させる，あるいは資源割り当てとスケジュールを同時進行的に最適化する合成手法においては，資源割り当て時にデータや演算の生存期間が確定しておらず，不用意な資源共有はスケジュール不能な解を生成してしまう．この論文では，スケジュール解の存在を保証する資源割り当てについて考察を行っている．ここでは essential lifetime overlap (ELO) と呼ばれる不可避な lifetime overlap を定義・導入し，それらの極小集合である MinELO が単一プロセッサスケジュール解から抽出できるとを明らかにした．実際の資源割り当てにあたっては，この MinELO だけに注意して資源共有を行うことで，スケジュール可能性を保証できる．

キーワード 高位合成，データパス合成，スケジュール，資源割当，演算順列

Minimal Set of Essential Lifetime Overlaps for Exploring 3D Schedule

Mineo KANEKO†

† Graduate School of Information Science, Japan Advanced Institute of Science and Technology
1–1 Asahidai, Nomi-shi, Ishikawa 923–1292, Japan
E-mail: †mkaneko@jaist.ac.jp

Abstract In many synthesis approaches, scheduling is completed before resource binding, and the lifetime of each of operations and data is firmly fixed when we start binding. So, the conflict-free resource sharing is easily guaranteed. Binding centric approaches and simultaneous scheduling and binding approaches such as 3D scheduling are alternative approaches to the high level synthesis regarding connectivity-related metrics. In those approaches, binding is often performed before completion of scheduling. Since a thoughtless binding often results in an infeasible solution, to identify and characterize a valid binding will be an important subtask. The objective of this paper is to identify and characterize the resource binding which guarantees schedulability. We introduce the concept of "essential lifetime overlap" which is an unaviodable lifetime overlap for a valid schedule. We also introduce the concept of "minimal set" of essential lifetime overlapping pairs (MinELO), and show that MinELO can be characterized by a single processor schedule. By only taking care of MinELO, a schedulable resource binding is always guaranteed.

Key words High level synthesis, datapath synthesis, schedule, binding, operation sequence

1. Introduction

High level synthesis is the task to transform the behavioral description in algorithm level into the architectural and behavioral descriptions in the register transfer level, and it includes scheduling and resource binding as its major subtasks. The objective of this article is to discuss interactions between scheduling and resource binding, and to identify and characterize the resource binding which guarantees schedulability

In many synthesis approaches [1], scheduling is completed before resource binding, and the lifetime of each of operations and data is firmly fixed when we start binding. The possibility of resource sharing among operations and data is easily tested, and conflict-free resource sharing is guaranteed. However, if we need to care about the wire complexity, wire delay, power consumed at wires, etc. in the high level synthesis, it would be hard to optimize schedule independently of resource binding.

Binding centric approaches [2] and simultaneous scheduling and

binding approaches such as 3D scheduling [3] are alternative approaches to the high level synthesis regarding connectivity-related metrics. In those approaches, binding is often performed before completion of scheduling, and only partial or no information on lifetimes of operations and data is available, when we bind someone to some resource. Since a thoughtless binding often results in an infeasible solution, to identify and characterize a valid binding will be an important subtask.

In this paper, we will introduce the concept of "essential lifetime overlap" which is an unaviodable lifetime overlap for a valid schedule. It is interesting that some of essential lifetime overlapping pairs are drawn inherently, and the other are drawn by circumstances. We will also introduce the concept of "minimal set" of essential lifetime overlapping pairs (MinELO), and show that MinELO can be characterized by a single processor schedule. By only taking care of MinELO, a schedulable resource binding is always guaranteed.

This paper is organized as follows. In section 2, we will briefly summarize several different synthesis problems with mainly focusing on lifetime models. Section 3 is devoted to the definitions of essential lifetime overlap and several related definitions. Our main theorem is described in section 4. Some additional comments on the extentions of the main theorem and conclusions are shown in section 5, and 6.

2. Synthesis Problems

The discussions, which we will have in this paper, will cover from the high level (datapath) synthesis to system level synthesis including reconfigurable system synthesis.

2.1 Fine grain synthesis

For convenience's sake, we will use a dependence graph $G(O \cup D, A)$ for representing an application algorithm to be implemented, where O is a set of operations, D is a set of data, and A is a set of arcs representing data generation by an operation and feeding data to an operation. (For a compact representation, a data vertex, not a primary input data vertex, may often be omitted and a generating operation and a consuming operation are directly connected by an arc.)

There are two major models of timing issue. One is based on execution time of each operation, and the schedule is considered as the assignment of operations to control steps. The other is based on a path delay from a source register to a destination register (or from a multiplexer to a destination register), and the schedule is considered as the assignment of control signals to control steps. We will adopt the former model in this paper, and it is assumed that the execution times of operations $e : O \rightarrow \mathbb{Z}_+$ is given as a part of input instance. Following the concept of the timing model, the operation schedule is defined as the mapping σ from O to $\mathbb{N}$.

$$\sigma : O \rightarrow \mathbb{N}$$

There may be also several different definitions on the lifetimes of operations and data, here we will follow the simplest definition. That is, for the lifetime τ of an operation $o_i \in O$,

$$\tau(o_i) = [\sigma(o_i), \sigma(o_i) + e(o_i) - 1]$$

where $[i, j]$ represents all integers from i to j. For a data d_i which is generated by o_i and is used by $o \in S(d_i)$, its lifetime $\tau(d_i)$ is given as

$$\tau(d_i) = \left[\sigma(o_i) + e(o_i), \max_{o \in S(d_i)} \{\sigma(o) + e(o) - 1\}\right]$$

where $S(d_i)$ is a set of all immediate successors of d_i in $G(O \cup D, A)$.

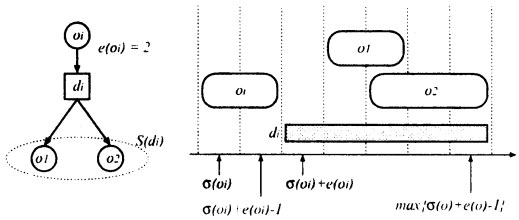


Fig. 1 Time chart demonstrating lifetime model.

On the other hand, resource binding is the mapping ρ from $O \cup D$ to the power set of hardware components $\mathcal{C}$.

$$\rho : O \cup D \rightarrow 2^{\mathcal{C}}$$

The above definition looks too periphrastic. It would be the conventional case that the hardware resource is a set of isolated functional units and registers, and the resource binding is reduced to

$$(\rho_O : O \rightarrow \mathcal{F}) \cup (\rho_D : D \rightarrow \mathcal{R})$$

where $\mathcal{F}$ adn $\mathcal{R}$ are a set of functional units and a set of registers, respectively. $\rho : O \cup D \rightarrow 2^{\mathcal{C}}$ will make sense, for instance, for a reconfigurable LSI in which cell units can form into a functional unit and a register from time to time.

Finally, the synthesis problem is to find σ and ρ which satisfy (1) dependency constraints specified by $G(O \cup D, A)$, and (2) conflict free constraint, i.e., for every pair $x, y \in O \cup D$,

$$\tau(x) \cap \tau(y) = \emptyset$$

or

$$\rho(x) \cap \rho(y) = \emptyset$$

2.2 Coarse grain synthesis

A coarse grain synthesis will be illustrated for the case of a reconfigurable system. The design flow of a reconfigurable system will include

(1) partitioning of input behavioral description of a system to be implemented into behavioral descriptions of sub systems, which are considered as units of configured blocks,

(2) for each sub–system, design a hardware module which implements the specified behavior of the sub–system, and

(3) design a reconfiguration and execution schedule of modules in the three–dimensional space (two dimensions for specifying positions on a reconfigurable VLSI chip, and one dimension for specifying timing of reconfiguration and execution).

Partitioning problems arise in various design strategies, and have been studied intensively [4] [5] [6]. The typical objectives of partitioning problems would be the size of each partition, the number of connections between partitions, the number of terminals of each partition, etc, and most of the conventional partitioning problems do not consider the fitness of the contents of each partition as a subject of hardware (or software) implementation. Automatic partitioning for a reconfigurable system remains as an open problem.

It is natural that partitioning is defined as the partitioning of O, that is, $O_1, \cdots, O_K$, where $\bigcup_{k=1}^{K} O_k = O$, and a sub-system #k is defined as the subgraph $G_k = (O_k \cup D_k, A_k)$ of G induced by the vertex set O_k. Furthermore, we will introduce a skeleton hyper graph $G_S = (P \cup I, E)$, where each element of P corresponds to one of partitions, and E is a set of global data dependencies. Furthermore we will define the cost of each arc $c : E \rightarrow \mathbb{Z}$.

Once a partitioning is given, first we need to analyze data dependency between sub-systems, enumerate hyper edges of the skeleton graph G_S, and compute the cost of each hyper edges by collecting global data having identical set of terminal partitions (which are vertices in the skeleton graph) and summing up word-lengths for these data. For convenience's sake, the number of global data assigned to a hyper edge is called multiplicity of the edge.

Fig. 2 shows a simple example of partitioning and its corresponding skeleton graph. A circle vertex in the dependence graph represents an elementary operation, and a box vertex does a primary input variable. Note that, according to the partitioning P_1, P_2, P_3, primary input data have been also partitioned into three subsets by the difference of patterns which partitions the input is fed to. In the skeleton graph, a small oval on arcs indicates that those arcs forms one hyper edge, and the number associated with such oval denotes the multiplicity of the hyper edge. For example, the hyper edge $\{P_1, P_3\}$ is the result of the original arcs g and j, and hence its multiplicity becomes 2. On the other hand, the hyper edge $\{P_1, P_2, P_3\}$ is the result of the original arc h alone, and hence its multiplicity is 1.

In an implementation of the input behavioral description on a reconfigurable system, each vertex P_k in P, which corresponds to the subgraph G_k, is associated with a computational module M_k which is designed specialized for the behavior G_k, and each hyper edge E_ℓ in E is associated with a register module R_ℓ which consists of a number of registers (the number is the same with the multiplicity of the edge).

So now the subjects to be configured and scheduled are set of computational modules $\{M_1, M_2, \cdots, M_K\}$ and set of register modules $\{R_1, R_2, \cdots, R_L\}$.

When we consider 3-D scheduling, our coarse grain model does not identify individual lifetime of each data which is bound to the same hyper edge (that is, bound to the same register module). If one want to refine the implementation result by using aggressive register sharing, one need to split one hyper edge in the skeleton graph into several hyper edges and treat each of them separately.

When we have finished to implement each computational module M_k, we can identify the timing when input data is used in the module, and also the timing when output is generated in the module, where each timing is measured (for example, but not limited to) from the start of the execution of the sub-system.

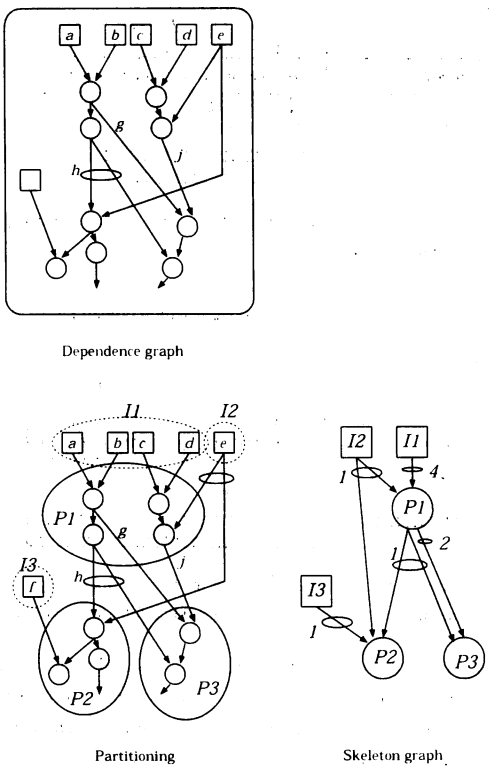


Fig. 2 Partitioning and skeleton graph.

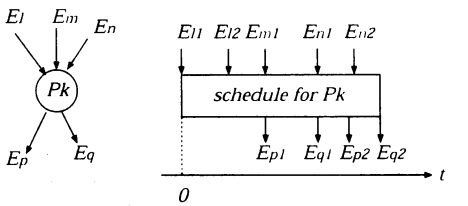


Fig. 3 Illustration of relative timing of data consumption and data generation.

The register module lifetime $\tau(R_\ell)$ for a register module R_ℓ will be computed as follows (a fine-grain lifetime model). Now let E_ℓ be the corresponding hyper edge in the skeleton graph, and $\{E_{\ell 1}, E_{\ell 2}, \cdots, E_{\ell m}\}$ be data bound to the hyper edge E_ℓ.

$$\tau(R_\ell) = \left[\min_{j=1}^{m} \{\text{generation timing of } E_{\ell j}\}, \max_{j=1}^{m} \{\text{latest consuming timing of } E_{\ell j}\} \right]$$

Another model for register module lifetime (fully-overlapping lifetime model) is:

$$\tau(R_\ell) = \left[\min_{j=1}^{m} \{\text{start of the subsystem which generates } E_{\ell j}\}, \right.$$

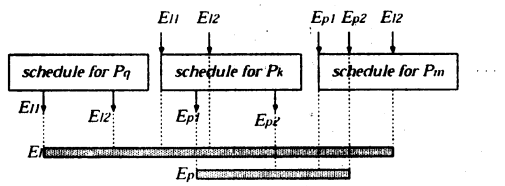


Fig. 4 A coarse grain model I of register module lifetimes (a fine-grain lifetime model).

$$\left[\max_{j=1}^{m}\{\text{end of the subsystem which consumes } E_{\ell j}\}\right]$$

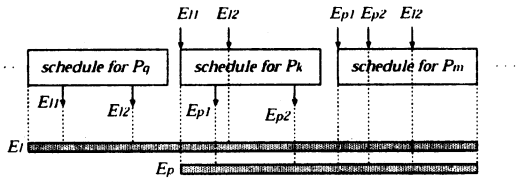


Fig. 5 A coarse grain model II of register module lifetimes (fully overlapping lifetime model), which can be applied fully top-down design.

Comparing the fully-overlapping lifetime model with the fine-grain lifetime model, the fine-grain model is more helpful for aggressive register module sharing. However, the fine-grain model requires detailed analysis (that is, fine grain description) of individual computation module in terms of the timing of data input and data output.

On the other hand, fully-overlapping lifetime model requires no information about the behavior of individual computation module. Hence, it can be easily applied to fully top-down design approach.

Fig.6 shows a small example of a complete design of a reconfigurable system based on the fully-overlapping lifetime model.

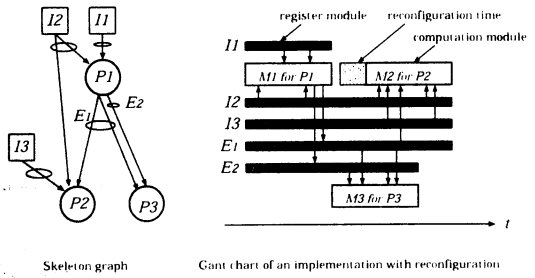


Fig. 6 A simple example of a complete design of a reconfigurable system base on fully-overlapping lifetime model.

Finally, our reconfigurable system synthesis problem is; given a skeleton graph $G_S(P \cup I, E)$, computational modules $M = \{M_1, M_2, \cdots, M_K\}$ (correspond to P in G_S), and register modules $R = \{R_1, R_2, \cdots, R_L\}$ (correspond to E in G_S), find the schedule

$$\sigma : M \to \mathbb{N}$$

and the resource binding

$$\rho : M \cup R \to 2^{\mathcal{C}}.$$

3. Essential Lifetime Overlaps

If $\tau(x) \cap \tau(y) \neq \emptyset$ for $x, y \in O \cup D$, x and y are called "lifetime overlapping pair", or we say "x and y have lifetime overlap".

[Definition 1] (Essential lifetime overlap (ELO))

If we can not find any valid schedule satisfying $\tau(x) \cap \tau(y) = \emptyset$, we call the lifetime overlap $\tau(x) \cap \tau(y) \neq \emptyset$ as "essential".

Two input data of one operation is a simplest example of the essential lifetime overlapping pair.

If x and y have the essential lifetime overlap, we must bind so that $\rho(x) \cap \rho(y) = \emptyset$. The essential lifetime overlap is defined from the view point of resource binding.

[Definition 2] (Essential resource disjoint-ness (ERD))

If $\rho(x) \cap \rho(y) = \emptyset$ is mandatory for a valid schedule, we say "x and y have the essential resource disjoint-ness". If x and y have the essential resource disjoint-ness, x and y also have the essential lifetime overlap.

[Definition 3] (Minimal set of ELO (MinELO))

Let $\mathcal{E}$ be a set of essential lifetime overlapping pairs, i.e.,

$$\mathcal{E} = \left\{ \{x, y\} \;\middle|\; \begin{array}{l} x, y \in O \cup D, \text{ a pair } x \text{ and } y \text{ have} \\ \text{the essential lifetime overlap} \end{array} \right\}$$

$\mathcal{E}$ is called "minimal" if

1. *$G(O \cup D, A)$ is schedulable under $\rho(x) \cap \rho(y) = \emptyset$ for all pair $\{x, y\} \in \mathcal{E}$, and $\rho(x') \cap \rho(y') \neq \emptyset$ for all $\{x', y'\} \in \overline{\mathcal{E}}$.*
2. *$G(O \cup D, A)$ is fail to be scheduled under $\rho(x'') \cap \rho(y'') \neq \emptyset$ for any one pair $\{x'', y''\} \in \mathcal{E}$ and $\rho(x') \cap \rho(y') \neq \emptyset$ for all $\{x', y'\} \in \overline{\mathcal{E}}$.*

("$\rho(x') \cap \rho(y') \neq \emptyset$ for all $\{x', y'\} \in \overline{\mathcal{E}}$" seems a literary description. In a practical situation, it can be interpreted as "we can assign $\tau(x') \cap \tau(y') = \emptyset$ or $\rho(x') \cap \rho(y') = \emptyset$ arbitrary for each $\{x', y'\} \in \overline{\mathcal{E}}$.")

The problem discussed in this paper is; how to identify MinELO and how to characterize MinELO.

4. Main Theorem under Fully Overlapping Model

Here we limit our discussion to the fully overlapping data lifetime model (Fig. 5), that is, the lifetime of a data begins at the beginning of the operation which generates the data, and ends at the end of the operation which uses the data as an input.

$$\tau(d_i) = \left[\sigma(o_i),\ \max_{o \in S(d_i)} \{\sigma(o) + e(o) - 1\}\right]$$

[Theorem 1]

Under the fully overlapping data lifetime model, a MinELO is given by the set of lifetime overlapping pairs in a single processor schedule.

If $G(O \cap D, A)$ is a directed acyclic graph, a single processor schedule is considered as a topological order of operations.

Sketch of proof:

Let $(\sigma_0 : O \to \mathbb{N}, \rho : O \cup D \to 2^{\mathcal{C}})$ be an arbitrary valid pair of

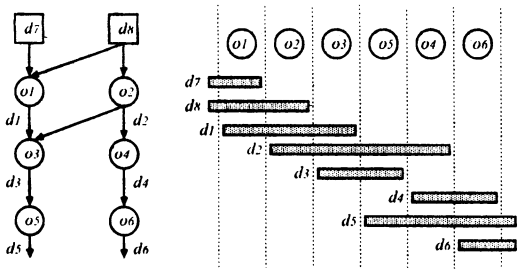


Fig. 7 Examle of a single processor schedule and MinELO extraction. MinELO given by this single processor schedule is $\{\{d_7, d_8\}, \{d_7, d_1\}, \{d_8, d_1\}, \{d_8, d_2\}, \{d_1, d_2\}, \{d_1, d_3\}, \{d_2, d_3\}, \{d_2, d_4\}, \{d_2, d_5\}, \{d_2, d_6\}, \{d_3, d_5\}, \{d_4, d_5\}, \{d_4, d_6\}, \{d_5, d_6\}\}$

schedule and resource binding. Let DJ_0 be the set of all resource disjoint pairs, and let OL_0 be the set of all lifetime overlapping pairs, both on (σ_0, ρ). Clearly,

$$DJ_0 \supseteq OL_0,$$

because $\rho(x) \cap \rho(y) = \emptyset$ must hold for all pair $\{x, y\} \in OL_0$.

Now we choose one operation $A \in O$ whose lifetime is not properly included by other operation's lifetime. We then split the schedule σ_0 into three parts, the first is the schedule of operations which begins earlier than the operation A, the second is the operation A alone, and the third is the schedule of operations which begins no earlier than A. Generate the schedule σ_1 in the following way; the schedule of the first part remains the same, the start of A is shifted after ends of all operations in the first part, and the schedule of the third part is further shifted after the end of A.

σ_1 is again a valid schedule under the initial resource binding ρ. We let OL_1 be the set of all lifetime overlapping pairs in σ_1. Then we have

$$OL_0 \supseteq OL_1$$

We can repeat the above transformation until all operations are arranged in a line, and we finally have

$$DJ_0 \supseteq OL_0 \supseteq OL_1 \supseteq \cdots \supseteq OL_m$$

where OL_m is the set of all lifetime overlapping pairs of a single processor schedule.

Finally, we can easily verify that OL_m satisfies the condition of MinELO. □

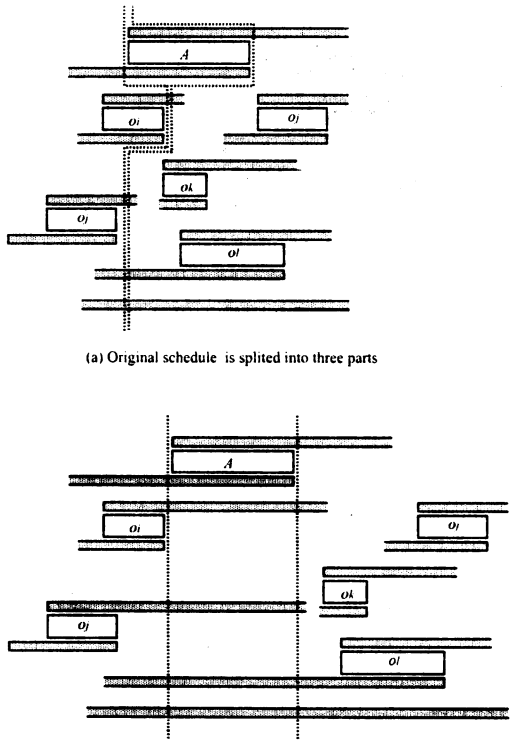
[Corollary 1]

The number of different MinELO is no larger than the number of different topological order of operations.

5. Comments on MinELO under Different Lifetime Models

There are several different data lifetime (resource occupation time) models, and the characterization of MinELO would depend on a data lifetime model.

A. Partly overlapping model



(a) Original schedule is splited into three parts

(b) Schedule is expanded

Fig. 8 Illustration of schedule splitting and expansion.

When we adopt the fine-grain lifetime model with overlapping input/output data; lifetimes of an input data and an output data of an operation overlap partly, a single processor schedule is again used to extract a minimal set of essential lifetime overlapping pairs (Fig.9).

B. No overlapping model

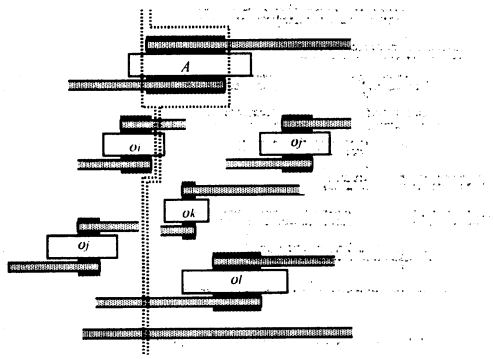
When we adopt the fine-grain lifetime model with no-overlapping input/output data; lifetimes of an input data and an output data of an operation do not overlap (see Fig.1), a simple single processor schedule is no longer valid, since there is a feasible implementation which can not be represented by a single processor schedule (Fig.10).

One possible modification for characterizing MinELO under such no-overlapping model would be the split of an operation into two operations, one of which works for designating the end of an input data, and the other works for the start of an output data. Then a single processor schedule can be used again for MinELO (Fig.11).

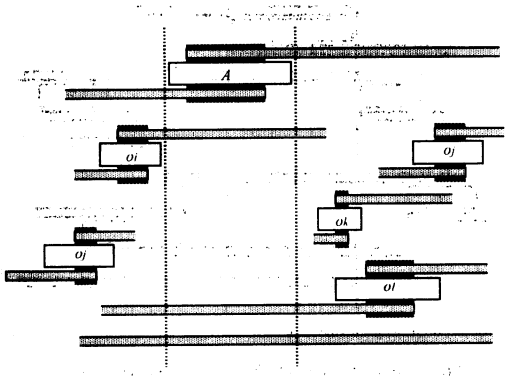
Detailed discussions will be reported in future.

6. Conclusions

In this paper, we haveintroduced the concept of "essential lifetime overlap" which is an unaviodable lifetime overlap for a valid schedule. It is interesting that some of essential lifetime overlapping pairs are drawn inherently, and the other are drawn by circumstances. We have also introduced the concept of "minimal set" of essential lifetime overlapping pairs (MinELO), and show that MinELO can be



(a) Original schedule is splited into three parts



(b) Schedule is expanded

Fig. 9 Illustration of schedule splitting and expansion for partial overlapping input/output data lifetime model.

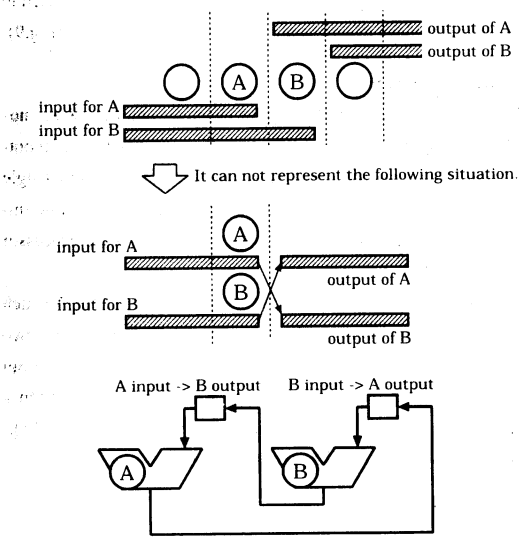


Fig. 10 Feasible implementation which can not be represented by a single processor schedule.

characterized by a single processor schedule. By only taking care of MinELO, a schedulable resource binding is guaranteed.

Comments for the case of the partly overlapping input/output data lifetime model and for the case of no overlapping input/output data

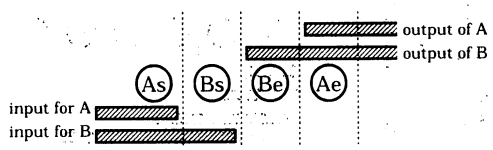


Fig. 11 Split an operation into its start and end vertices for adapting no-overlapping input/output data lifetime model. By doing this, neither pair of input of A and output B nor the pair input B and output A is an essential lifetime overlapping pair, and the input of A and the output of B can share a register while the iput of B and the output of A share another register.

lifetime model have been shown. However they are still future works. Discussions on an input algorithm including conditional branches and a datapath allowing data duplicate remain as future works. Interaction between min/max path delay based control signal schedule and resource binding is alo an interesting problem needed to be tackled in future.

References

[1] Petra Michel, Ulrich Lauther, Peter Duzy, "The synthesis approach to digital system design", Kluwer Academic Publishers, 1992.

[2] Toshiyuki Yorozuya, Koji Ohashi, Mineo Kaneko, "Assignment-Driven Loop Pipeline Scheduling and Its Application to Data-Path Synthesis", IEICE Trans. Fundamentals, Vol.E85-A, No.4, pp.819-826, 2002.

[3] Mineo Kaneko, Jun'ichi Yokoyama and Satoshi Tayu, "3D Scheduling Based on Code Space Exploration for Dynamically Reconfigurable Systems," Proc. IEEE 2002 International Symposium on Circuits and Systems, Vol.V, pp.465-468, May 2002.

[4] Eduardo Sanchez, Moshe Sipper, Jacques-Olivier Haenni, Jean-Luc Beuchat, Andre Stauffer and Andres Perez-Uribe, "Static and Dynamic Configurable System," *IEEE Transaction on Computer*, Vol.48, No.6, pp.556-564, 1999.

[5] Douglas Chang and Malgorzata Marek-Sadouska, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs," *IEEE Transaction on Computer*, Vol.48, No.6, pp.565-578, 1999.

[6] Karthikeya M. Gajjala Purna and Dinesh Bhatia, "Temporal Partitioning and Scheduling Data Flow Graphs for Reconfigurable Computer," *IEEE Transaction on Computer*, Vol.48, No.6, pp.579-590, 1999.