システムＬＳＩ設計技術　107－17
(2002. 11. 28)

非同期回路におけるデータパス遅延情報を用いた制御信号共有化手法

齋藤　寛† EuiseokKim† 今井　雅† NatthaSretasereekul† 中村　宏†
南谷　崇†

† 東京大学先端科学技術研究センター　〒153-8904 東京都目黒区駒場 4-6-1
E-mail: †{hiroshi,eskim,miyabi,nattha,nakamura,nanya}@hal.rcast.u-tokyo.ac.jp

あらまし　合成における状態爆発問題のために、現存する非同期 CAD ツールのほとんどは大きな設計仕様を扱うことができないといった問題点がある。このような問題点を解決する一つの手法として、本報告ではデータパス遅延情報を用いた２つの制御信号共有化手法を提案する。合成の際に生ずる状態の数は信号の数に指数的増大なので、共有化による信号数の削減は状態数の削減に直結することができる。共有化自体は、コントロールフローグラフ上でのノード共有化によって行われる。実験の結果より、多数の信号が共有化されたと言う点で本手法の有効性を示すことができた。

キーワード　コントロールフローグラフ, ノード共有化, start and end time requirements, 信号遷移グラフ

Control Signal Sharing of Asynchronous Circuits Using Datapath Delay Information

Hiroshi SAITO†, Euiseok KIM†, Masashi IMAI†, Nattha SRETASEREEKUL†, Hiroshi NAKAMURA†, and Takashi NANYA†

† Faculty of Research Center for Advanced Science and Technology, University of Tokyo
Komaba 4-6-1, Meguro-ku, Tokyo, 153-8904
E-mail: †{hiroshi,eskim,miyabi,nattha,nakamura,nanya}@hal.rcast.u-tokyo.ac.jp

Abstract　Due to state explosion problem in the synthesis, most of asynchronous CAD tools based on graph models cannot handle large specifications. To overcome this problem, in this paper, we propose two control signal sharing methods by using delay information of datapath circuits. Since the number of states in the synthesis is exponential with respect to the number of signals, the reductions of signals by sharing may reduce the number of states significantly. They are carried out at control flow graph (CFG) descriptions in terms of node sharing. Experimental results are encouraging in that a number of control signals are shared.

Key words　control flow graph, node sharing, start and end time requirements, signal transition graph

1. Introduction

There are several CAD tools for asynchronous circuit synthesis [2], [5]. However, due to state explosion problem in the synthesis, most of asynchronous CAD tools based on graph models cannot handle large specifications although they have good optimization heuristics.

In this work, we suggest control signal sharing methods by using delay information of datapath circuits. Since the number of states in the synthesis is exponential with respect to the number of signals, the reductions of signals by sharing may reduce the number of states significantly. Delay information of datapath circuits is derived if one tries scheduling and allocation for datapath operations using pre-designed datapath components because they are parameterized by execution delays.

Control signal sharing is realized as follows. As an initial description, a control and data flow graph (CDFG) after scheduling and allocation for datapath operations is used. From it, delay information for datapath circuits is derived. Using this delay information, control signals are shared without changing the total delay in detapath circuits if several

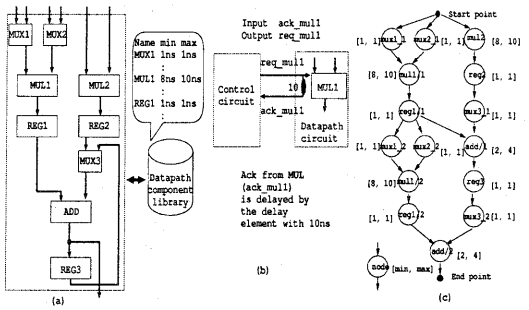

Fig. 1 (a) a datapath circuit with a component library, (b) a bundled data implementation, (c) a control flow graph for (a)

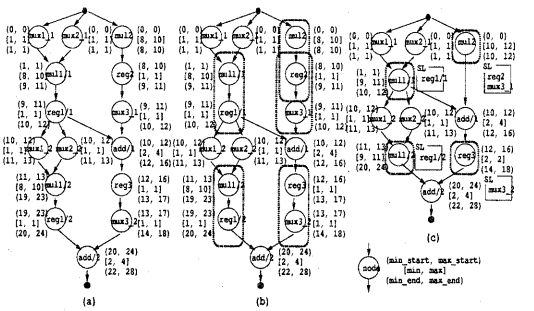

Fig. 2 (a) Calculations of start and end times, (b) Possibilities of sequential node sharing in (a), (c) A reduced CFG

timing requirements are satisfied. This is considered at a control flow graph (CFG) which is a part of CDFG in terms of CFG node sharing. Finally, low level control circuit descriptions are derived from reduced CFGs and synthesized by asynchronous logic synthesis tools.

2. Basic Notions

2.1 Datapath Circuit Model

Figure 1.a shows an example of datapath circuits in register transfer level where *FUs, MUXs, REGs*, mean functional units, multiplexors, and registers. Due to the uses of predesigned component libraries, each component has parameterized execution delays denoted by [`min`, `max`] where `min` means minimal delay and `max` means maximal delay.

In asynchronous datapath circuits, each datapath component is controlled by a pair of handshake signals, a *request (req)* which initiates the component and an *acknowledgment (ack)* which indicates the completion. In this work, we assume bundled data type implementations [4], where *req* signals are generated from control circuits and *ack* signals are *req* signals delayed by the delay elements which correspond to `max` delays of datapath components plus margins. Figure 1.b shows the control of *MUL1* with this implementation. In another datapath style so-called dual-rail implementations [4] where execution delay ranges between `min` and `max` delays depending on input data, our proposed sharing methods may work well due to the considerations of `min` and `max` delays.

2.2 Control Flow Graph

Different from datapath circuits, control circuits for datapath circuits must be synthesized. In this work, since we use a scheduled and allocated control data flow graph (CDFG) description, a control part of CDFG called control flow graph (CFG) is derived to synthesize control circuits. Figure 1.c shows a CFG of Figure 1.a where nodes represent the controls of datapath components, arcs represent causal relations among nodes, and labels represent node names weighted by `min` and `max` delays. If a label is indexed with i (e.g., mul1/1), it means the i-th instance of the node. Note, in this work, *MUXs* have a one-hot control signal per input, e.g., mux1_1, mux1_2, and mux1_3 are control signals for *MUX1*, since it has three inputs. In addition, we introduce two reference points, *start point* and *end point* where start point means the start of datapath operations and end point means the end of datapath operations.

In this work, control signal sharing is realized at CFGs by using the relations of `min`, `max` delays, and other detapath delay information.

2.3 Start and End Times

We introduce *start* and *end* times for each CFG node, which are used during control signal sharing. Start time means when the corresponding operation starts and end time means when the operation completes. Since we are introducing both `min` and `max` delays for each CFG node, start time is denoted by (*min_start*, *max_start*) and end time is denoted by (*min_end*, *max_end*). Both *min_start* and *max_start* of each CFG node come from maximum values for each *min_end* and *max_end* of its predecessors except that *min_start* and *max_start* of the nodes immediately after start point are assumed to 0. On the other hand, both *min_end* and *max_end* are calculated by the summations of *min_start* and `min` delay for *min_end* and *max_start* and `max` delay for *max_end*.

For example in Figure 2.a, *min_start* and *max_start* of a node reg1/1 are (9, 11), whereas *min_end* and *max_end* of the node are (10, 12).

3. Control signal sharing

In this work, two kinds of control signal sharing methods, *sequential node sharing* and *concurrent node sharing*, are presented. They are carried out on a given CFG without sacrificing the performance of datapath circuits.

3.1 Sequential Node Sharing

In a given CFG, suppose nodes a and b are directly ordered in the sequence of $a \rightarrow b$ (e.g.,mul1/1→reg1/1 in Figure 2.a).

The conditions of sharing for nodes a and b are classified into three cases by which datapath components are controlled by nodes a and b.

Case 1: Both a and b are a node for *FUs* or *REGs*. Nodes a and b are shared if

- $\forall i(i = 1 \ldots n), \exists j(j = 1 \ldots m)$, a/i has only one successor node b/j and b/j has only one predecessor node a/i, and
- $\forall j(j = 1 \ldots m), \exists i(i = 1 \ldots n)$, b/j has only one predecessor node a/i and a/i has only one successor node b/j

In this case, a and b can be shared in all instances.

For example in Figure 2.a, reg1/1 is shared by mul1/1 and reg1/2 is shared by mul1/2 because in all instances of mul1 and reg1 they satisfy the above conditions. In addition, mul2 and reg2 are also shared.

Case 2: a is a node for FUs or REGs and b is a node for MUXs. Nodes a and b are shared if

- $\forall i(i = 1 \ldots n), \exists j(j = 1 \ldots m)$, a/i has only one successor node b/j and b/j has only one predecessor node a/i

Different from Case 1 where all of the instances of a and b are considered, Case 2 may consider all of the instances of a only. This is because in datapath circuits b is just used to multiplexing the output of a FU which is controlled by a. Hence, b is not used independently from a.

For example in Figure 2.a, mux3_1 is shared by reg2 and mux3_2 is shared by reg3 because they satisfy the Case 2 conditions.

Case 3: a is a node for MUXs and b is a node for FUs or REGs. Nodes a and b are shared if

- $\forall j(j = 1 \ldots m), \exists i(i = 1 \ldots n)$, there is a predecessor node p/i before b/j, which controls the same MUX as a, and p/i has only one successor b/j and b/j has only one predecessor p/i

Case 3 is different from Case 1 and Case 2, since MUX control signals before *REGs* or *FUs* can have different node names in all instances of *REGs* or *FUs* due to the use of a control signal for each *MUX's* input. Therefore, in all instances of b/j , the predecessor node p/i can be shared with b/j if p/i always controls the same MUX as a and they are directly ordered.

Figure 2.b shows all of the possibilities of sequential node sharing with respect to Figure 2.a (surrounded by dotted cycles).

Effects of sequential node sharing. These three conditions are checked for all of the nodes in a given CFG. If a pair of nodes a and b can be shared, the predecessor node a is remained where `min` and `max` delays are modified to the summations of nodes a and b. In addition, to identify which node is shared by which, the predecessor node a keeps the successor node b in the *seq_list (SL)*. Figure 2.c shows a reduced CFG obtained by sequential node sharing of Figure 2.a. Figure 3.a shows the effect of sequential node sharing where *req* of *REG1* is the delayed req_mul1 and *ack* of REG1 is returned to the control circuit. Note since *MUXs* use one-hot control signals, *acks* are generated ORing them (see Figure 3.b).

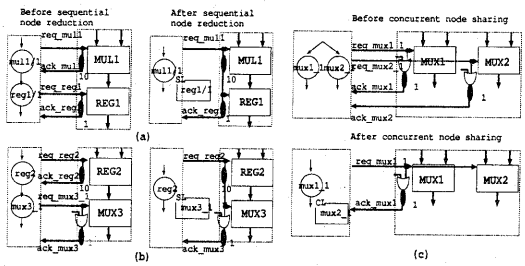

Fig. 3 Effects of control signal sharing

3.2 Concurrent Nodes Sharing

Suppose nodes a and b are concurrent in a given CFG. Node b is shared by node a if a satisfies both the start and the end time requirements of b. However, this may be very restrictive. Intuitively, suppose that the start times of both nodes a and b are the same. In this situation, although node b can be shared by node a if `min` delay of a is larger than `max` delay of b, it may not be realized because the end time of a will be over the end time of b. Therefore, we introduce *slack* for each node to explore much possibility of sharing.

3.2.1 Calculations of Slack

Slack is defined for each start and end time where the slack of start time means the latest possible start time denoted by (sl_min_start, sl_max_start) and the slack of end time means the latest possible end time denoted by (sl_min_end, sl_max_end).

Slack is calculated in opposite way in contrast to start and end times, i.e., the calculation starts from end point until start point. sl_min_end and sl_max_end directly come from the minimum values of sl_min_start and sl_max_start of the successor nodes. sl_max_start is calculated by subtracting `max` delay from sl_max_end and sl_min_start is calculated by subtracting `max` delay (not `min` delay) from sl_min_end. From the slack calculation, we can identify the nodes in *critical paths* of datapath circuits. In these nodes, two pairs of *max_start* and sl_max_start and *max_end* and sl_max_end are equal to each other. We consider the nodes with their slack if their *min_start* is less than or equal to sl_min_start. Otherwise, the introduction of slack is meaningless.

Figure 4.a shows the slack for Figure 2.c, where nodes mux1_1, mux2_2, mul1/1, mux1_2, mux2_2, mul1/2, and add/2 are the nodes in the critical paths. Therefore, the rest of the nodes, mul2, add1, and reg3, are considered with

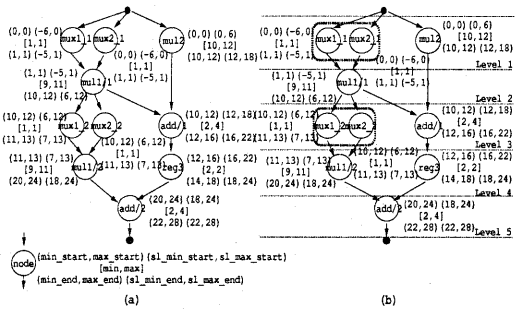

Fig. 4 (a) Calculations of slack time in Figure 2.c, (b) Possibilities of concurrent node sharing in Figure 2.c

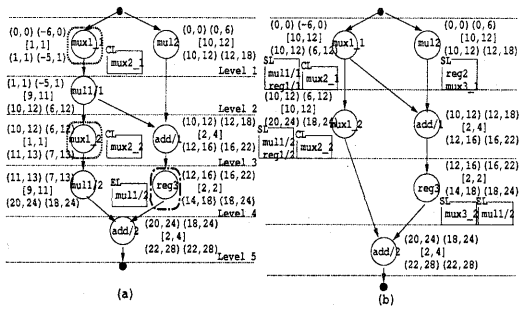

Fig. 5 (a) A reduced CFG after concurrent node sharing in Figure 4.b, (b)The finally reduced CFG in Figure 2.a

their slack.

3. 2. 2 Start and End Time Requirements

In this sub-section, we explain start and end time requirements which must be satisfied to share nodes a and b. Suppose that $a(time)$ denotes *time* of node a, where *time* comes from *min_start*, *min_end*, sl_max_end, and so on. The formal start time requirements of a with respect to b are:

- $b(min\_start) = a(min\_start)$ and $b(max\_start) = a(max\_start)$ if b(sl_min_start) $< b(min\_start)$
- $b(min\_start) \leqq a(min\_start) \leqq b$(sl_min_start) and $b(max\_start) \leqq a(max\_start) \leqq b$(sl_max_start) if b(sl_min_start) $\geqq b(min\_start)$
- $b(max\_start) \leqq a(min\_start)$ if both a and b do not have the same predecessors

The first requirement is required if the slack time of node b is meaningless, i.e., sl_min_start is less than *min_start*. If it is not the case, the second requirement is checked. The second one just checks whether the start time of b is delayed to the start time of a if b starts before a. The last condition guarantees that the predecessor of b always complete before the start of a.

For example in Figure 4.a, since mux1_1 and mux2_1 have the same predecessor (i.e., start point) and the same start time (the first and the last requirements are satisfied), we find out that a satisfies the start time of b and vice versa.

In addition to the start time requirements, the end time requirements must be checked to share node b by a. The formal end time requirements of a with respect to b are as follows;

- b(max) $\leqq a$(min)
- $a(min\_end) = b(min\_end)$ and $a(max\_end) = b(max\_end)$ if b(sl_min_end) $< b(min\_end)$
- $a(min\_end) \leqq b$(sl_min_end) and $a(max\_end) \leqq b$(sl_max_end) if b(sl_min_end) $\geqq b(min\_end)$

The first requirement guarantees that node b always completes before node a. In addition, since the end time of node b affects the start time of successor nodes of node b, the sharing may lead performance degradation if node b is a node in the critical paths. Therefore the second requirement is required. On the other hand, the last requirement guarantees that node b can be delayed until the latest possible end time (slack). Finally, in addition to the first requirement, one of the last two requirements must be satisfied.

For example in Figure 4.a , since min delay of mux1_1 and max delay of mux2_1 are equal to 1 and they have the same *min_end* and *max_end* , the first two requirements are satisfied. It means that mux1_1 satisfies the end time requirements of mux2_1.

3. 2. 3 Applications of Concurrent Node Sharing

Concurrency between nodes in a given CFG can be captured by the introduction of *levels*. The level for each node is defined as follows; The nodes immediately after start point are level 1. The nodes immediately after level n-1 are level n (see Figure 4.b).

After introducing levels, for each level, concurrent node sharing between nodes a and b is carried out by checking start and end time requirements. This must be satisfied in all of the instances if nodes a and b have multiple instances.

In Figure 4.b, since mux1 satisfies both the start and the end time of mux2 in all instances (i.e., in level 1 and level 3), they are shared (denoted by dotted circles). In this case, mux1_1 and mux1_2 are remained in the graph while mux2_1 and mux2_2 are put into *conc_list (CL)* of mul1 (Figure 5.a).

Effects of concurrent node sharing. Figure 3.c shows the effect of concurrent node sharing where *MUX2* is controlled by the signals of *MUX1*.

Satisfying end time requirements. During concurrent node sharing, there may exist a node a which satisfies both the start and the end time requirements of a node b but fails to sharing because of the violations in different instances. In this case, only in the instance where node a satisfies both the start and the end time of node b, node b keeps the node name of node a into *end_list (EL)* of node b. Satisfying the end time by node a with respect to node b implies that *ack*

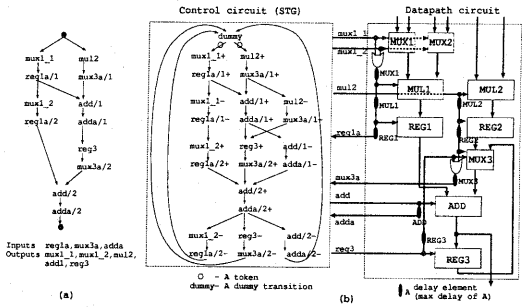

Fig. 6 STGs of Figure 5.b: (a) an intermediate form, (b) a complete one

Table 1 Results of control signal sharing

	# of nodes			# of signals			
name	ori	redc	ratio1	ori	redc	ratio2	lits
3FIR	15	6	60%	21	7	66%	43
5FIR	33	17	48%	41	21	49%	136
2IIR	29	16	45%	37	22	40%	146

signal for node b can be shared by *ack* signal for node a. For example in Figure 5.a, since mul1/2 satisfies the end time of reg3, reg3 keeps mul1/2 into its *EL* (see Figure 5.a).

4. Experimental Results

4.1 Derivations of Signal Transition Graphs

Sequential and concurrent node sharing methods are applied until there is no possibility of sharing in reduced CFGs. Figure 5.b shows the reduced CFG after further sequential node sharing in Figure 5.a which is the finally reduced CFG of Figure 2.a. The important point is that both sharing methods do not change the total execution delay in datapath circuits.

We use signal transition graphs (STGs) [1] as the inputs of logic synthesis tools. They are derived from reduced CFGs while inserting handshake signals (*req* and *ack*) for each CFG node and four phase signaling protocol [3].

Figure 6.a shows a partial STG of Figure 5.b after handshake signal insertions. The final STG of Figure 5.b is shown in the left side of Figure 6.b. The overall circuit structure of Figure 1 with bundled data implementation is shown in Figure 6.b.

4.2 Experiments on Filter Circuits

In the experiments, we applied control signal sharing methods to well known *filter* circuits in the order of sequential, concurrent, sequential, concurrent node sharing. The derived STGs were synthesized by using asynchronous logic synthesis tool *Petrify* [2]. To investigate the efficiency of control signal sharing, we show how many nodes in the original CFGs are reduced by sharing methods.

Table 1 shows the results of control signal sharing and the synthesis results by *Petrify*. Column "name" means the names of filter circuits. The second and the third columns represent the number of nodes in the original CFGs and the reduced CFGs. The fourth column "ratio1" shows the reduction ratio of CFG nodes due to control signal sharing methods. The fifth and sixth columns show the number of signals in the STGs derived from the original CFGs and the reduced CFGs. The seventh column "ratio2" represent the reduction ratio of signals due to control signal sharing. The last column shows the number of literals for logic functions produced by *Petrify* with respect to the STGs derived from the reduced CFGs. Note "lits" also includes OR gates for *MUXs*.

Observing the experimental results, we found out that our control signal sharing methods significantly reduce a number of signals in STGs derived from the reduced CFGs (about half of the original). From this reason, we believe that control signal sharing gives better perspective for large asynchronous circuit synthesis because it may significantly reduce the size of control circuits.

5. Conclusions

In previous works, graph-based synthesis methods for asynchronous circuits cannot handle large specifications because of state explosion problem. To overcome this problem, we have proposed two control signal sharing methods at control flow graph (CFGs) descriptions. Since sharing is realized by using delay information of datapath circuits, we formally defined the requirements of sharing. Experimental results were encouraging in that a number of control signals are shared (about half of the original). From these results, control signal sharing may give better perspective for large asynchronous circuit synthesis.

References

[1] T. A. Chu. Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications. PhD thesis, MIT Laboratory for Computer Science, June 1987.

[2] J. Cortadella, M. Kishinevsky, A.Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Automatic handshake expansion and reshuffling using concurrency reduction. In *Proc. of the Workshop Hardware Design and Petri Nets* , pages 86–110, June 1998.

[4] A. Davis and S. M. Nowick. Asynchronous circuit design: Motivation, background, and methods. In Graham Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 1–49. Springer-Verlag, 1995.

[5] K. Yun. *Synthesis of Asynchronous Controllers for Heterogeneous Systems*. PhD thesis, Stanford University, August 1994.