Flexible Partitioning of CDFGs for Compact Asynchronous Controllers

Nattha Sretasereekul† Yuichi Okuyama‡ Hiroshi Saito† Masashi Imai† Kenichi Kuroda‡ Takashi Nanya†
†The University of Tokyo ‡The University of Aizu

Abstract— Asynchronous circuits have the potential to solve the problems related to parameter variations such as gate delays in deep sub-micron technologies. However, current CAD tools for large-scale asynchronous circuits partition specification irrelevantly, because these tools cannot control the granularity of circuit decomposition. In this paper, we propose a hierarchical Control/Data Flow Graph (CDFG) containing nodes that are flexibly partitioned or merged into other nodes. We show a partitioning algorithm for such CDFGs to generate *handleable Signal Transition Graphs* (STGs) for asynchronous synthesis tools. The algorithm allows designers to assign the maximum number of signals of partitioned nodes considering optimality. From an experiment, this algorithm can flexibly partition and result in more compact asynchronous controllers.

I. Introduction

Recently, gate delay has been reduced by technological improvements in deep sub-micron technologies. As a result, the relationship between gate delay and wire delay has been reversed, and wire delay has become dominant over gate delay, making it difficult to execute all the register transitions synchronously. Using asynchronous circuits becomes a potential solution for this problem.

Current asynchronous CAD tools have limitations to deal with large-scale asynchronous circuits. Partitioning of such circuits into small sub-circuits improves the synthesis and verification quality. Moreover, synthesizing from small partitioned specifications tends to get more efficient circuits than large centralized specifications [1, 2, 3, 4].

Control/Data Flow Graphs (CDFGs) [5] are a well-known register transfer level (RTL) specification for processors. CDFGs are inherently in a hierarchical structure. Based on them, there are several works previously solved for asynchronous circuit synthesis called hardware-oriented and process-oriented partitioning. In hardware-oriented partitioning, a centralized controller is separated into small control nodes. A sub-controller corresponds to a controller of a data-path block (i.e. functional units, multiplexers, and registers) [1, 3]. In process-oriented partitioning, a control node corresponds to the controller of an operation node (such as +, -, *, <, etc.) [2]. These techniques are systematic and allow automatic partitioning. However, they are not flexible. Controllers for functional units are sometimes so large that they cannot be handled by CAD tools, and controllers for operation nodes are so small that they need large protocols between them instead.

In this paper, we propose a hierarchical CDFG containing nodes that are flexible to be partitioned or to be merged into other nodes. We focus on only the control part because the data path part can use library modules. We show an algorithm for partitioning such CDFGs to *generate handleable STGs (Signal Transition Graphs)* [6] for asynchronous synthesis tools. The STGs are then synthesized by Petrify [8] to obtain hazard-free asynchronous circuits. Our algorithm allows designers to assign the maximum number of signals of partitioned nodes. From many synthesis results, the designer can choose the optimum one. From an experiment, this algorithm can flexibly partition and result in more compact asynchronous controllers.

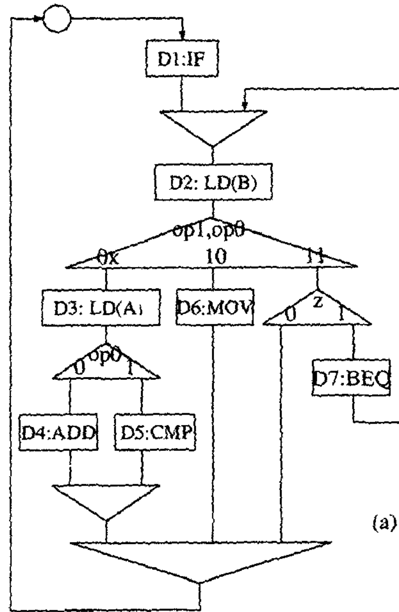
II. Hierarchical CDFG

CDFGs are commonly used as an intermediate form to represent the control and data flow of target systems. A leaf node of the structure may be the controller of a process or a functional unit. Different from [1, 3] and [2], where a functional unit controller is too large and a process controller is too small, in this paper, a leaf node represents a group of processes.

An example of our CDFGs is shown in Fig.1(a) , and its corresponding hierarchical structure in Fig.1(b). Here, we consider a processor called Sim1 which is a memory-based processor with two temporary registers. It has four instructions. The data path consists of multiplexers, an instruction register, a program counter, temporary registers A and B, ALU, and a flag register.

Each leaf node corresponds to a Data Flow Graph (DFG) describing a data flow such as instruction fetch, instruction decode, or execution. If a leaf node is too large for synthesis, it can be further partitioned in the same way as a CDFG is partitioned to have the controller corresponding to each leaf node.

Introducing this hierarchical structure allows the CDFG to be flexibly partitioned within the defined primitive nodes (seq: sequential node, par: parallel node, choice: choice node, while: loop node, and main node) and enables optimum output STGs for synthesis. Note that the



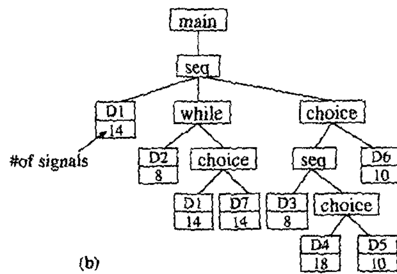


Fig. 1. A CDFG of processor Sim1 (a) and its hierarchical structure (b)

given CDFGs should be already scheduled and derived from specifications such as HDL.

III. STG Generation

An asynchronous controller mainly generates control signals for data paths and other controllers. Two signals, namely, request (req) and acknowledgment (ack) signals are distributed to each module to control computation and communication. In this paper, these control signals are executed by using a four-phase protocol [7].

STGs effectively represent the behavior of asynchronous controllers containing concurrent events. However, synthesizing asynchronous controllers from STGs may suffer from the state explosion problem. An existing tool, Petrify [8], can synthesize an STG that contains only about 20 signals.

To obtain hazard-free asynchronous controllers, the STGs should be derived satisfy properties: boundedness, commutativity, consistency, persistency, and complete state coding (CSC) [8]. After partitioning, CDFGs are translated into handleable STGs satisfying these properties. Fig.2 shows the STGs corresponding to each primitive node.

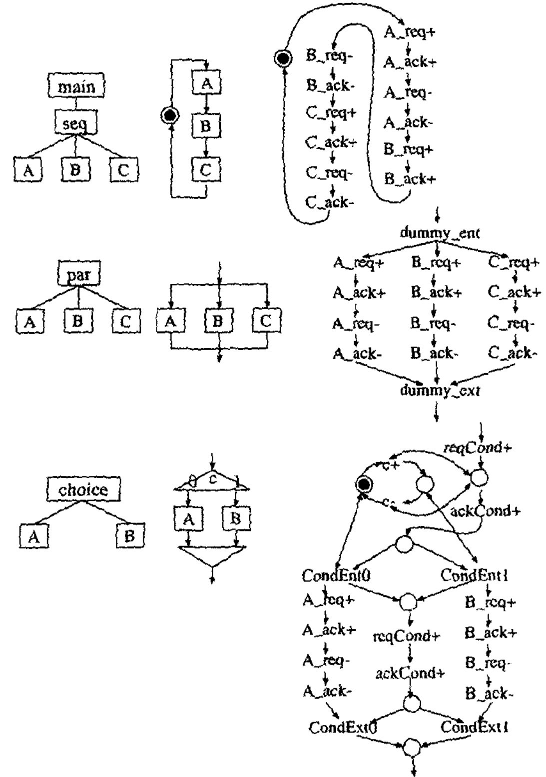


Fig. 2. STGs of CDFG primitive nodes.

IV. Partitioning algorithm

The structure of our hierarchical CDFG can provide STGs that can be synthesized by Petrify. Conditions under which circuits can be synthesized by Petrify are not specifically identified because they are strongly influenced by the given STGs. Therefore, in most cases, the number of input and output signals in the STG is used as a basis to verify whether the generated circuits can be synthesized. The largest number of signals allowed is represented by n, which is regarded as the sufficient condition.

To provide an STG with signals limited to n, the given CDFG and its tree structure must be partitioned. Then we add handshake signals between the partitioned controllers. The number of signals in each partitioned controller must not exceed n. These additional handshake signals are an overhead. However, since less CSC conflicts left in each STG, the result partitioned circuits often smaller than the centralized one.

Our algorithm is described below.

Algorithm process

1. A sub-tree that does not include control nodes is referred to a DFG, and a group of nodes in a DFG is removed from the list of candidates for partitioning.

2. Control nodes ("choice" and "loop" nodes) are separated from the other nodes. A group of separated control nodes is S'.
3. The following process is applied to each CDFG of the separated control nodes in S' according to the results of the breadth-first search:
   (a) If a node to be merged exceeds n, partitioning of this node is performed.
   (b) If "total + 2 < n", the two nodes are merged.
   (c) If not, the node is partitioned and the resulting sub-tree is added to S'.

First, nodes that include a conditional operation are referred to control nodes, while all the other nodes are referred to Data Flow Graph (DFG) nodes. The DFG node group obtained after the execution of a conditional signal could generate STGs capable of synthesizing.

Next, partitioning related to conditional behavior is performed. The STGs generated from "choice" and "loop" nodes, which contain conditional behavior, tend to be extremely complicated; therefore, only one conditional node should be allowed in each partitioned CDFG.

The "seq" nodes that appear inside "choice" nodes may cause complicated CSC conflicts. Therefore, partitioning of "choice" nodes is performed on the "seq" node level. In the other word, the "choice" node to be partitioned is allowed to include only one level of a "seq" node. As shown in Fig.4(a), node D3 is not merged into the upper "seq" node. This restriction results in less complicated CSC conflicts than merging D3 into the upper "seq" node as shown in Fig.4(b).

In addition, when the hierarchical CDFG is used to represent a certain circuit that has been represented by a CDFG, several DFGs of the same function could appear in the hierarchical CDFG. In such a case, only one of them is used to synthesize a control circuit. For example, node D1 in Fig.1 is shared by the seq and choice nodes. The overall control circuits generate circuits that meet the specifications of the hierarchical CDFG by activating the synthesized circuits using handshake pairs from several parent CDFGs.

Finally, when a given single node ("seq", "par", "choice") is more complicated than the S' set obtained by partitioning the control nodes, the node will be partitioned rather than being merged into the nodes in S'. After that, connections among nodes are searched out. Then the primitive nodes continue to be merged inside the CDFG until the largest number of signals allowed is reached. The result is a sub-graph that can be synthesized. This algorithm performs a breadth-first search against the given tree structure. The searched-out nodes are checked for the number of signals. Then the number of signals after merging is calculated on the premise that these nodes are merged with additional ones. If the calculated number exceeds the largest number of signals allowed (n), merging will not be performed but the node will be partitioned. By repeating the same procedure on the partitioned sub-trees, a hierarchical CDFG with signals within the largest number of signals allowed is generated.

V. Glue Logics

As a result of partitioning, one data path block or a controller could sometimes receive request signals from more than one controller. Therefore, to control or arbitrate signals, we need a special kind of circuits called "glue logics".

Glue logics are likely to increase the controller area. However, since a data path block is often much larger than its controller, using glue logics could reduce the redundant data path blocks.

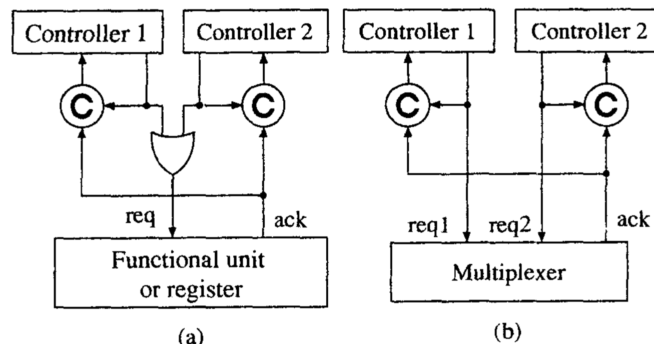


Fig. 3. Glue logics

As hypothesized in this paper, the given CDFGs are all scheduled. Therefore, partitioned controllers never transmit request signals to modules simultaneously, and the glue logic to be inserted can easily be determined.

For example, Fig.3(a) illustrates the circuits in which one controller receives request signals from two controllers. In this circuit, when a request signal is generated from one of controllers, the C-elements are used to determine for which controller should be returned the acknowledgment signal. The other glue logic is shown in Fig.3(b). When an operation that involves a multiplexer is being performed, the request signals directly come from each controller, and the acknowledgment signal is returned to the controller requesting to use the multiplexer.

VI. Experimental Result

As an example, we make an experiment on processor Sim1 described previously. An evaluation was conducted based on the following three results of partitioning:

1. Partitioning using the proposed algorithm → Fig.4(a).
2. Partitioning to obtain the finest grain (fine-grain partitioning) → Fig.1(b).
3. Partitioning to minimize the number of nodes (coarse-grain partitioning) → Fig.4(b).

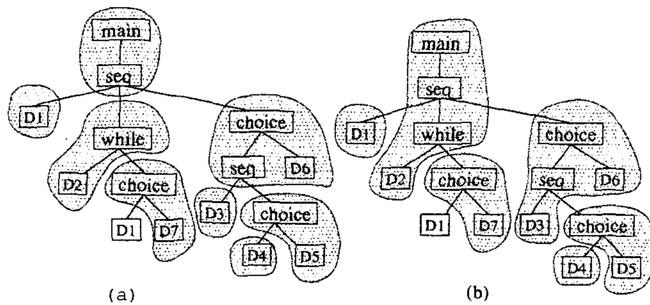


Fig. 4. CDFG nodes of processor Sim1 partitioning: proposed partitioning (a), coarse grain partitioning (b).

TABLE I
SYNTHESIS RESULTS OF PROCESSOR SIM1.

processor Sim1 (♯ of CDFGs)	suggested 8	fine 13	coarse 6
non-tm CDFG area	2224	2376	2603
ratio	1.00	1.04	1.06
CB-C10 CDFG area	631	662	-
ratio	1.00	1.03	-
Petrify CDFG area	3832	4088	-
ratio	1.00	1.04	-

The number of the signals is limited to 22 (n=22).

Table I shows the results of synthesizing these CDFGs. Technology-mapping of the coarse-grain partitioning was unable to be mapped to CB-C10 [9] and Petrify libraries, so no results for coarse-grain partitioning are shown.

From these results, we can observe that the proposed algorithm provides 4% decrease in circuit size compared to the fine-grain partitioning. This indicates that partitioning CDFGs into too small nodes causes the area overhead on handshake signals for communicating the partitioned nodes.

The proposed algorithm also provides 6% better area than the coarse-grain partitioning. This indicates that restriction on partitioning a series of "seq" nodes can reduce the number of state explosion, resulting in fewer CSC signal insertions during synthesis. Therefore, considering the synthesis efficiency of each partitioned CDFG node, fine-grain partitioning of nodes is considered more desirable.

The above two evaluations are obviously contradictory, and a trade-off in the granularity of partitioning is suspected. The results of this experiment suggest that, with respect to generated circuit size, the proposed algorithm is more capable of achieving optimum granularity compared to fine-grain and coarse-grain partitioning.

VII. Conclusions

We have proposed a hierarchical CDFG and an algorithm for partitioning asynchronous controllers. The structure of our hierarchical CDFG allows flexible partitioning regardless of data path structures. The given CDFG was partitioned according to the largest number of signals allowed so that circuits could be synthesized from the partitioned CDFGs. The evaluation of the algorithm indicated its capability to generate asynchronous circuits efficiently.

Acknowledgments

This work was supported in part by Information-technology Promotion Agency (IPA) Japan. We would like to thank Prof. Hiroshi Nakamura for his helpful comments on this work.

References

[1] Jordi Cortadella, Rosa Badia, Enric Pastor, and Abelardo Pardo, "Achilles: A High-Level Synthesis System for Asynchronous Circuits", 6th International Workshop on High-Level Synthesis, pp. 87-94, 1992.

[2] Euiseok Kim, Jeong-Gun Lee and Dong-Ik Lee, "Automatic Process-Oriented Control Circuit Generation for Asynchronous High-Level Synthesis", *in Proceedings of Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems(ASYNC2000)*, pp. 104-113, April 2-6, 2000.

[3] Michael Theobald and Steven M. Nowick, "Transformations for the Synthesis and Optimization of Asynchronous Distributed Control", *in Proceedings of 38th Design Automation Conference (DAC)* pp. 263-268, June 18-22, 2001.

[4] Prabhakar Kudva, Ganesh Gopalakrishnan, and Hans Jacobson, "A Technique for Synthesizing Distributed Burst-mode Circuits", *In proceedings of 33rd Design Automation Conference*, 1996.

[5] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[6] Tam-Anh Chu, "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications", *Ph.D Thesis, Massachusetts Institute of Technology*, June, 1987.

[7] Al Davis and Steven M. Nowick, "An Introduction to Asynchronous Circuit Design", *The Encyclopedia of Computer Science and Technology*, Vol. 38, Marcel Dekker, New York, February 1998.

[8] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers", *XI Conference on Design of Integrated Circuits and Systems*, November 1996.

[9] NEC Corporation, "CB-C10 Family 0.25-μm CMOS Cell-Based IC (2.5V) Library", July 1997.