A RTL Partitioning Method with a Fast Min-Cut Improvement Algorithm

Kenichi Kawaguchi Chie Iwasaki Michiaki Muraoka
Semiconductor Research Center, Matsushita Electric Industrial Co., LTD.
3-1-1, Yagumo-Nakamachi, Moriguchi, Osaka 570 JAPAN
Tel: +81-6-906-4933
Fax: +81-6-906-3851
e-mail: kawaguti@vdrl.src.mei.co.jp

Abstract !=A design flow with register-transfer-level (RTL) partitioning and a RTL partitioning algorithm for efficient logic synthesis and layout are described in this paper. Changing the parameter of partitioning optimization dynamically, the algorithm improves an interconnection cost in a short CPU time. Experimental results on large circuits show that the algorithm partitioned circuits with the large number of RTL components in a tenth to a hundredth of conventional partitioning times.

I. INTRODUCTION

DA technology is struggling with today's complicated ASIC designs. A topdown design approach with Hardware Description Languages (HDL) and logic synthesis has reduced the design term. The approach requires circuit partitioning to change the logical or functional hierarchy of a circuit into the physical hierarchy proper for its hardware implementation.

In the design of a large digital circuit, the logical hierarchy is so deep and complicated that it is very difficult for designers to find the best hierarchy that minimizes the design terms of logic synthesis and layout. The hierarchy with large blocks is suitable for floorplanning, but causes time-consuming iterations of syntheses. Small blocks are more appropriate to synthesize than large ones, but the hierarchy with small blocks results in the large number of interconnects which makes delay and area of interconnection large. To solve this problem, the physical hierarchy should be constructed with proper size blocks and the small number of interconnections so that the blocks of the circuit are appropriate for logic synthesis, floorplanning, and layout.

Our approach is to partition circuits at the RTL so that the synthesis term and the layout term are reduced. The partitioning method reduces partitioning CPU time significantly, and minimizes the interconnects with a heuristic algorithm which estimates partitioning CPU time in advance of partitioning improvement.

Partitioning is popular at the netlist level because the quality and the CPU time of layout highly depend on that of the partitioning carried out beforehand. A circuit at the netlist level has much more objects than at the RTL, so partitioning takes much longer CPU time, in particular, in the partitioning improvement stage. Because minimum objects of a netlist circuit are cells or gates, it is difficult to partition the circuit taking functional units into consideration. For example, flip--flops of a shift-register could be assigned to different blocks. Nets of a bundle connecting a pair of functions could turn to be interconnects of different pairs of blocks.

Partitioning studies at the functional level have been reported[1],[2]. Partitioning in this early stage of a system design flow is very important for a multi-chip system, because timing and area constraints for high-level synthesis and logic synthesis highly depend on the way of chip partitioning. However, a hardware structure performing the functions is not so clear that measuring the circuit area, which plays a principal role in partitioning for logic synthesis and layout, is difficult and not accurate.

With its hardware structure clear and easy to measure, a RTL circuit is suitable for synthesis-and-layout-oriented partitioning. Fewer objects make partitioning CPU time shorter than at the netlist level. Because each object represents a functional unit, flip-flops of a shift-register come into the same block when the circuit is synthesized. Nets of a bundle handled as a minimum unit are never assigned to different blocks.

The min-cut algorithm known as the Kernighan-Lin[3] algorithm interchanges two groups of nodes between two blocks to minimize the interconnects. Fiduccia and Mattheyses[4] improved the algorithm to execute in a shorter CPU time. Since the major part of partitioning time is spent in the iterative improvement stage, it is important to reduce interconnects effectively in this stage.

This paper proposes a partitioning algorithm which estimates the number of computations for improvement, makes groups for interchange efficiently, and completes the improvement in a short CPU time. It allows designers to design and partition a circuit interactively. Experimental results show that the algorithm significantly improves partitioning CPU time with fewer interconnects.

II. RTL PARTITIONING

1) A Design Flow with RTL Partitioning

In a topdown design flow, designing at the RTL is based on the function of a circuit and makes hardwares clear. Because data transfers between registers are visible and the minimum objects of the circuit represent functions, the circuit is able to be partitioned for synthesis and layout based on the functional units. Fig. 1 shows a design flow with RTL partitioning. The circuit designed at the RTL is partitioned into some blocks with their sizes appropriate for logic synthesis and layout. Small logical blocks are merged into larger physical blocks, and large logical blocks are partitioned into

ASP-DAC '97
0-89791-851-7/$5.00

smaller physical blocks to shorten the terms of synthesis and layout. In the RTL partitioning stage, areas of the circuit and its blocks are estimated to make blocks with similar sizes. Logic synthesis of each block follows RTL partitioning.

2) Bundle-base Interconnection

Interconnects of a netlist-level circuit are nets, while those of a RTL circuit are bundles of nets. From the layout and timing point of view, the nets within a bundle should be included in the same block or connect the same pair of blocks. Fig. 2a shows a RTL circuit which has a four-bit register, a comparator, and two one-bit registers. The circuit is easily partitioned into two blocks as shown in Fig. 2b. After logic synthesis, the partition of the circuit turns to be as shown in Fig. 3a. It has four interconnects between Block 1 and Block 2. If logic synthesis comes first and partitioning second, the circuit is partitioned with three interconnects as shown in Fig. 3b. The partition in Fig. 3a has a one-larger interconnection cost than the partition in Fig. 3b. However, the layout of the circuit shown in Fig. 3a would have shorter wiring delay and smaller wiring area thanks to bundle-base interconnection.

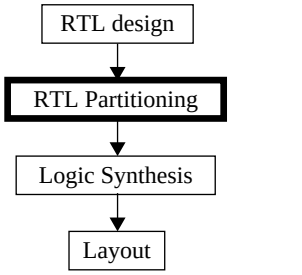

Fig. 1: Design Flow

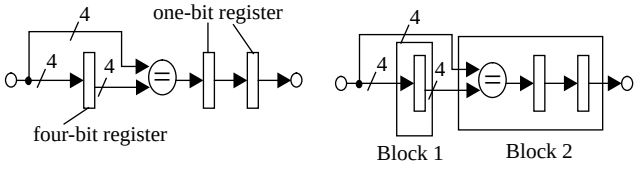

Fig. 2: RTL Circuit and Its Partition

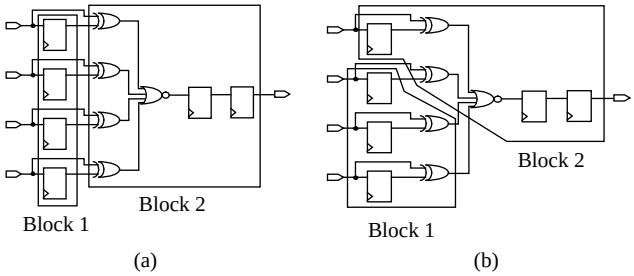

Fig. 3: Partition at Netlist Level

III. Partitioning Algorithm

1) Circuit Model

A circuit at the RTL handled in this paper consists of objects representing the functional units and data-transfers which connect some objects. Types of functional units are registers, functional modules, buses, multiplexers, submodules, and tables. In a partitioning process, the circuit is compiled as a hypergraph of which nodes and edges correspond to objects and data-transfers in the circuit, respectively. Each edge is weighted with the bit width of the corresponding data transfer.

2) Partitioning Flow

A hypergraph modeling a circuit is partitioned into some blocks. Then, the circuit is partitioned corresponding to the hypergraph partitioning.

Hypergraph partitioning consists of two processes: initial partitioning which partitions the hypergraph into the required number of blocks and partitioning improvement which reduces the interconnection cost, moving nodes from some blocks to other blocks. In each process, a gate estimator estimates the number of gates of blocks required for balancing the sizes of blocks at the netlist level.

The major part of the partitioning CPU time is spent on partitioning improvement to reduce lots of interconnects generated during initial partitioning.

3) Partitioning Improvement

The partitioning improvement algorithm reduces interconnects by moving a group of one or more nodes at a time rather than one node to avoid a local minimum of the interconnection cost. The larger the upper limit of the sizes of moving groups is, the fewer the interconnection cost is. However, the increase of the upper limit of the sizes of moving groups makes the number of the groups to compute the gains of moving larger and the CPU time for partitioning longer.

Our algorithm estimates the number of groups necessary to compute the gains with each upper limit of the size of the groups, and gets the initial upper limit from the point of view of the CPU time.

Fig. 4 shows our algorithm for partitioning improvement. A node at a border is defined as the node adjacent to at least one node of a different block. First, in Step1, it gets the initial upper limit of sizes of moving groups. Second, in Step2, it selects one of the (unselected) nodes at the border, say node A, of the blocks. Third, in Step3, finding the group that reduces the interconnection cost most among groups including node A, it moves the group. Fourth, in Step4, if some unselected nodes are left at the border, it goes back to the second step. Otherwise, it ends.

If the upper limit of sizes of moving groups is fixed regardless of the circuit structure and the algorithm selects all groups with the size equal to or smaller than the upper limit, the number of the groups grows rapidly with the increase of the number of nodes of the circuit. That leads the increase of

partitioning CPU time.

To avoid the increase of CPU time, the algorithm selects a group of nodes by picking up nodes one by one and changing the upper limit according to the number of adjacent nodes and the blocks they are assigned to.

Fig. 5 shows the algorithm of moving the group with the largest gain including a node, say node A, at the border. A selection parameter in a step is defined as the number of nodes which would be assigned to the group after the step. In this algorithm, S7 plays the crucial role to shorten partitioning CPU time and improve the connection cost. S7 increases the selection parameter by one if the number of nodes adjacent to the current node is one or two. It makes the possibility of partitioning improvement higher.

If all of the following conditions are satisfied, S7 decreases the selection parameter by one.

1. Four or more nodes outside of the selected group are adjacent to the current node.
2. The number of nodes adjacent to the current node is larger than the selection parameter.
3. All nodes adjacent to nodes of the selected group except node A are assigned to the block which the current node is assigned to.

It reduces the number of computations of gains of groups.

In Fig. 6, the circuit has two interconnects between Block1 and Block2. Assume the selection parameter is three, the current node is node n1, and node n1 is added to the selected group as the first node of the group at S3. To reduce the interconnection cost of the partitioning, the group of five nodes, {n1, n2, n3, n4, n5}, should be moved to Block1 at a time. Without S7, three groups, {n1}, {n1, n2}, {n1, n2, n3}, would be computed their gains. no matter which one is moved, the interconnection cost would never be reduced.

At S7, if the number of nodes adjacent to the current node such as node n1 is two, the selection parameter is increased by one. Therefore, at S9, when nodes n2, n3, n4, n5 are selected as the current node, the selection parameters are 3, 3, 2, 2, respectively. Thus, the group of five nodes, {n1, n2, n3, n4, n5} can be selected to compute the gain. With the largest gain, the group is moved to Block1 to reduce the interconnection cost to one.

In Fig. 7, the circuit has one interconnect. Assume the selection parameter is three at S3, the current node is node n1, and the selected group consists of a node, n1.

.

Step1 Get the initial upper limit of the size of moving groups.
Step2 Select one node at the border (say node A) of the blocks.
Step3 Move the initial group including node A.
Step4 If some unselected nodes are left at the border, go back to Step2. Otherwise, end.

Fig. 4: Partitioning Improvement Algorithm

S1 Set the upper limit of sizes of moving groups to a selection parameter.
S2 Select node A as the current node.
S3 Add the current node to the selected group.
S4 Compute the gain of the selected group and the sizes of blocks when it moves to the other block.
S5 If the sizes of blocks computed in the former step were balanced, update the best group with the largest gain.
S6 Decrease the selection parameter by one.
S7 Change the selection parameter according to the number of nodes adjacent to the current node and the blocks they are assigned to.
S8 If the selection parameter is zero, execute S10.
S9 For all nodes adjacent to the current node, set one of them as the new current node, and execute S3 to S9 one by one.
S10 If the gain of the best group is one or more, move the best group.

Fig. 5: Algorithm of Moving Group

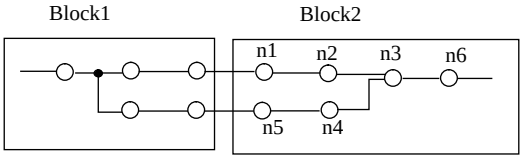

Fig. 6: Partitioning Example 1

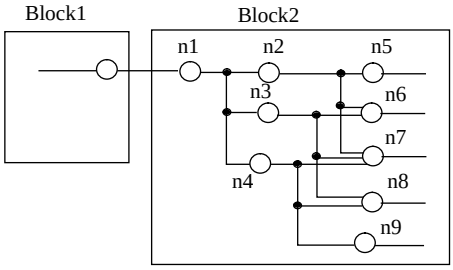

Fig. 7: Partitioning Example 2

Without S7, thirteen groups would be computed their gains. No matter which one is moved, the interconnection cost would never be reduced.

At S7 when the current node is node n2, which has six adjacent nodes, the selection parameter decreases from one to zero. The number of groups to compute the gain is four and partitioning goes faster.

4) Getting the Initial Upper Limit

Because the major part of the partitioning CPU time is spent on partitioning improvement as mentioned above, partitioning CPU time is estimated from the number of gain computations of groups. The algorithm gets the initial upper limit of sizes of groups in order to complete partitioning in time appropriate to the circuit size.

The algorithm to get the initial upper limit is shown in Fig. 8. The sub-algorithm to estimate the number of gain computations of groups for T3 is shown in Fig. 9. As you see, the algorithm in Fig. 9 looks similar to the algorithm of moving the node with the largest gain in Fig. 5. The difference is it has no steps corresponding to S4, S5, and S10.

As groups of nodes don't move in the algorithm in Fig. 9, the estimated number of computations is different from the real number. However, the difference is too small to affect getting the initial upper limit.

To estimate the number of computations, the algorithm doesn't compute gains, nor update the best selected groups, nor move them as shown in Fig. 9. Thus, it takes much smaller amount of CPU time than partition improvement does.

IV. Experimental results

To evaluate the performance of it, we implemented the partition algorithm described in this paper. The RTL circuits for evaluation were designed with our in-house ESDA tool, Bchart[5],[6]. We also implemented the Kernighan-Lin algorithm (K-L algorithm) for RTL partitioning. TABLE I shows the results of two-way partitioning.

T1 Fix the number of computations of gains according to the size of circuit.
T2 Set one to the upper limit.
T3 Estimate the number of computations of gains of groups with the upper limit.
T4 Exit if the number of computations at T3 is reached to the number fixed at T1.
T5 Increase the upper limit by one and go back to T3.

Fig. 8: Getting the Initial Upper Limit

U1 Select a node at the border of blocks as the current node.
U2 Set the upper limit of sizes of moving groups to a selection parameter.
U3 Add the current node to the selected group.
U4 Increase the estimation number by one.
U5 Exit if the estimation number has been reached to the fixed number at T1.
U6 Decrease the selection parameter by one.
U7 Change the selection parameter according to the number of nodes adjacent to the current node and the blocks they are assigned to.
U8 If the selection parameter is zero, execute U10.
U9 For all nodes adjacent to the current node, set the selected node as the new current node, and execute U3 to U9 one by one.
U10 If some unselected nodes are left at the border, it goes back to T1. Otherwise, exit.

Fig. 9: Estimation of the Number of Computations of Gains

TABLE I: Experimental results of two-way partitioning

Circuits	#objects	#data-transfers	partitioning CPU time (sec.)		interconnection costs	
			K_L	our algorithm	K_L	our algorithm
#1	595	732	8	8	35	35
#2	866	1900	217	10	215	208
#3	2243	7041	10437	90	593	207

#gates: 16305(#1), 24237(#2), 4751(#3)

The amount of reduced CPU time was balanced to the CPU time for estimation of partitioning time on circuit #1. Setting the upper limit of sizes of moving groups according to the size and construction of a circuit, our algorithm avoids the explosion of CPU time. Our algorithm reduced partitioning CPU times significantly in large circuits with fewer interconnection costs.

V. Conclusion

In this paper, a design flow with RTL circuit partitioning and a partitioning algorithm were proposed. The algorithm estimates partitioning CPU time in advance of partitioning improvement, gets the initial upper limit of the size of moving groups, and changes the upper limit dynamically according to the local structure of the circuit. Experimental results showed that the algorithm shortened partitioning CPU time with fewer interconnection cost.

The difference between the proper block sizes for logic synthesis and layout was not mentioned in this paper. In some cases, the physical hierarchy of a circuit should be changed when designing comes from logic synthesis to layout. For example, merging some blocks for logic synthesis into a block for layout reduces the layout term in the case that a layout tool handles larger blocks than logic synthesis tool does. The way to manage the hierarchies for logic synthesis and layout should be studied.

In the future, we will extend our work to the architecture or system level partitioning and complete a top-down design flow from the system level.

VI. References

[1] K.Kucukcakar and A.C. Parker, "CHOP: A constraint-driven system--level partitioner," *Proceedings of the 28th Design Automation Conf.*, pp.514-519, 1991

[2] R. Gupta and G. De Micheli, "Partitioning of functional models of synchronous digital systems," *Proceedings of the International Conf. on Computer-Aided Design*, pp. 216-219, 1990

[3] B.W. Kernighan and S. Lin., "An efficient heuristic procedure for partitioning graphs," *Bell Systems Technical Journal*, Vol.49, pp.291-307, 1970

[4] C.M. Fiduccia and R.M. Mattheyses, "A linear-time heuristic for improving network partitions," *Proc. of 19th Design Automation Conf.*, pp.175-181, 1982

[5] M.Matsumoto, Y.Takai, C.Iwasaki, and M.Muraoka, *"A functional design method based on graphical representation,"* Technical Report of IEICE, 93-ARC-98, 93-DA-65, pp. 73-80, 1993

[6] K.Nakatani, *"The evaluation of Bchart: a functional design system with a graphical environment,"* DA Symposium '93, pp. 97-100, 1993