Behavioural Bitwise Scheduling Based on Computational Effort Balancing[*]

M.C. Molina, R. Ruiz-Sautua, J.M. Mendías, R. Hermida
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid
{cmolinap, mendias, rhermida}@dacya.ucm.es, rsautua@fdi.ucm.es

Abstract

Conventional synthesis algorithms schedule multiple precision specifications by balancing the number of operations of every different type and width executed per cycle. However, totally balanced schedules are not always possible and therefore some hardware waste appears. In this paper a heuristic scheduling algorithm to minimize this hardware waste is presented. It successively transforms specification operations into sets of smaller ones until the most uniform distribution of the computational effort of operations among cycles is reached. In the schedules proposed some operations are executed during a set of non- consecutive cycles.

1. Introduction

In order to reduce the datapath area, conventional scheduling algorithms try to balance the number of operations of every different type executed per cycle [1], and allocation ones try to maximize the reuse of datapath hardware (HW) resources. When synthesizing mono precision specifications (those formed by operations of equal width), sometimes it is not possible to schedule the same number of operations of every different type per cycle. Thus, conventional allocation algorithms produce implementations in which some functional units (FUs) are unused during some cycles.

This HW waste is dramatically incremented when multiple precision specifications (those formed by operations of different widths) are synthesized using conventional allocation algorithms [2-4]. In this case, some datapath FUs are partially wasted due to the allocation of some operations to wider FUs.

The scheduling algorithm in [6] transforms every specification operation into additions, and afterwards performs jointly their scheduling. In combination with bit-level allocation algorithms [5], it produces datapaths with a unique FU type (adders), and where all complex operations have been fragmented into a set of simpler ones. Thus unnecessary fragmentation occurs in most cases. The proposed algorithm overcomes these disadvantages by performing a selective transformation of the specification operations. It minimizes HW waste by balancing the computational effort of operations executed per cycle, and produces substantial area reductions in comparison to other known approaches.

2. Scheduling algorithm

In order to minimize the HW waste of the synthesized circuits, the proposed algorithm assigns operations to cycles trying to balance the computational effort of the operations executed in every cycle. With this aim, it successively transforms specification operations into sets of simpler ones. The type and width of the new operations may be different from those of the original one, and are scheduled independently. In consequence, one specification operation may be executed during a set of non-consecutive cycles.

The algorithm is divided in the next three phases:

1) *Kernel extraction.*

To increase the number of operations which may share one FU, the specification multiplicative and additive operations are transformed into multiplications, additions, and some glue logic [6]. And also, signed operations are transformed into unsigned ones.

2) *Multiplication scheduling*

The algorithm selects for every multiplication a set of cycles included in its mobility (set of cycles in which an operation may start its execution). And it also fixes, for those operations scheduled in several cycles, the exact portion of every multiplication executed in each cycle.

3) *Addition scheduling*

The addition scheduling is performed using the algorithm presented in [6]. It produces schedules with the most uniform addition bits distribution among cycles reachable in every case.

In order to produce implementations with smaller area than those obtained by conventional synthesis algorithms, the schedules proposed by our approach must be allocated using bit-level allocation algorithms like [5]. They allow the execution of one operation over several linked FUs, whose types and widths may be different from the specification operation.

[*] This work has been supported by the Spanish Government under research grant CICYT TIC-2002/750

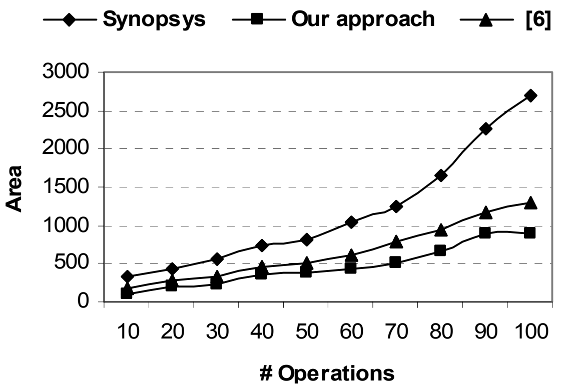

Fig.1. Average area of some implementations proposed by Synopsys, our approach, and [6].

2.1. Multiplication scheduling

This scheduling algorithm is a variant of the classical *force-directed algorithm* [1]. It takes into account operations widths in order to reduce the datapath area due to FUs either partially used, or unused during some cycles.

In every step of the algorithm, one multiplication and one of its mobility cycles are selected, in accordance to a redefinition of the *force* measure. The operation is then scheduled if the assignment does not unbalance the most uniform multiplication bits to cycles distribution reachable at the moment. Otherwise, it is fragmented to obtain one multiplication fragment of a certain *size* which will be scheduled in the selected cycle. The width $k \times p$ of this fragment must satisfy the next equation:

$$(k \leq m) \wedge (p \leq n) \wedge (size = k \times p)$$

being $m \times n$ ($m \geq n$) the original multiplication width

To avoid the high cost of solving this equation, our algorithm tries first the width of the already scheduled multiplications. If none of these operations widths solves the above equation, then it tries those fragmentations of the multiplication that produce two multiplication fragments (being one of the desired *size*) and one addition. Finally, if no solution is found, the algorithm transforms the selected operation to obtain two multiplication fragments whose sum of widths equals the desired fragment *size*. This is performed even when a possible fragmentation to obtain a fragment of the desired *size* exists. In this case, the widths $k \times n$ and $r \times 1$ of the multiplications fragments are calculated using the next equation:

$$(k \leq m) \wedge (r \leq n) \wedge (k = \lfloor size / n \rfloor) \wedge (size = (k \times n) + r)$$

Once the width of the multiplication fragment is selected, among all the different ways of fragmenting the rest of the operation, only those which produce the minimum number of operation fragments are considered. The algorithm selects among them, the one which requires the least cost in adders and produces the maximum number of operations of the same width as other specification operations.

After scheduling one multiplication fragment, some of the successors and predecessors of the scheduled operation must also be fragmented to avoid reductions in their mobilities.

3. Experimental results

The implementations obtained by our algorithm have been compared to those proposed by both Synopsys Behavioral Compiler, and the approach presented in [6].

We have synthesized a wide collection of synthetic specifications formed by multiplications and additive operations. Specifications sizes range from 10 to 100 operations (about 40% are multiplications), and latencies from 4 to 30 cycles. Circuit areas in the three cases are measured in number of equivalent gates (including FUs, storage and routing units, and controller).

Our approach saves up to 70% in area compared to Synopsys and up to 40% compared to [6]. Figure 1 shows the average area of the implementations obtained by the three algorithms grouped by the number of specification operations. The amount of area saved by the algorithm grows, in general, with the number of different operation widths and types present in the specification.

4. Conclusion

This paper presents a heuristic scheduling algorithm to perform the high level synthesis of behavioural specifications. Big datapath area reductions are achieved by distributing uniformly the operation computational cost among cycles. In order to increase the FUs reuse the algorithm extracts the common operative kernel of specification operations.

Experimental results show that the implementations obtained by our algorithm in combination with [5] have considerably smaller area than those proposed by other known approaches.

References

[1] P.G. Paulin, and J.P. Knight. "Force-Directed Scheduling for the Behavioral Synthesis of ASICS". *IEEE Transactions on CAD, 1989.*

[2] G.A. Constantinides, P.Y.K. Cheung, and W.Luk, "Heuristic datapath allocation for multiple wordlength systems". *In Proceedings of DATE, 2001.*

[3] M. Ercegovac, D. Kirovski, and M. Potkonjak, "Low-power behavioural synthesis optimization using multiple precision arithmetic". *In Proceedings of DAC, 1999.*

[4] B. Landwehr, P. Marwedel, and R. Dömer, "OSCAR: Optimum simultaneous scheduling, allocation and resource binding based on integer programming". *In Proceedings of EDAC, 1994.*

[5] M.C. Molina, J.M. Mendías, and R. Hermida, "High-level Allocation to Minimize Internal Hardware Wastage". *In Proceedings of DATE, 2003.*

[6] M.C. Molina, J.M. Mendías, and R. Hermida, "Bit-level Scheduling of Heterogeneous Behavioural Specifications". *In Proceedings of ICCAD, 2002.*