High-Level Synthesis of Multiple-Precision Circuits Independent of Data-Objects Length*

M.C. Molina, J.M. Mendías, R. Hermida
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid
Avda. Complutense s/n, 28040 Madrid (SPAIN)
{cmolinap, mendias, rhermida}@dacya.ucm.es

ABSTRACT
This paper presents a heuristic method to perform the high-level synthesis of multiple-precision specifications. The scheduling is based on the balance of the number of bits calculated per cycle, and the allocation on the bit-level reuse of the hardware resources. The implementations obtained are multiple-precision datapaths independent of the number and widths of the specification operations. As a result impressive area savings are achieved in comparison with conventional algorithms implementations.

Categories and Subject Descriptors
B.5.2 [**Register-transfer-level Implementation**]: Design Aids – *automatic synthesis, optimization.*

General Terms
Algorithms, Design.

Keywords
High-level synthesis, multiple-precision, scheduling, allocation.

1. INTRODUCTION
Conventional high-level synthesis (HLS) scheduling algorithms set the execution cycle in which every specification operation starts its execution. They try to balance the number of operations of each different type scheduled per cycle, independently of their widths. Figure 1a) shows a possible scheduling provided by a conventional algorithm for a multiple-precision specification formed by three additions with no data dependencies. However, it is possible to obtain circuits with smaller area by uniformly distributing the number of bits calculated among the cycles. In this case, an operation may be executed during a set of non-necessarily consecutive cycles starting from the least significant bits. The carry-out produced in one cycle must be stored to be supplied as the carry-in during the next execution cycle of the operation. Figure 1b) shows another possible scheduling for the specification in which the least significant bits of the operation (*A=B+C)* are calculated in cycle *i*, and the remaining ones in cycle *i+1*.

DAC 2002, June 10-14, 2002, New Orleans, Lousiana, USA.

Conventional HLS allocation algorithms provide datapaths where every operation is executed over one functional unit (FU) of the same or greater width. The number of FUs is determined by the maximum number of operations scheduled in the same cycle, with their respective widths depending on the number of bits of the wider operations. When these wider operations are not scheduled in such *busy* cycle, a considerable waste of area may result. Figure 1c) shows the conventional solution for the scheduling in Figure 1a). However, it is possible to obtain circuits with smaller area by following other design strategies, like the simultaneous execution of several operations over one FU, or the execution of one operation over several linked FUs. Figures 1d) and 1e) show these novel solutions for the scheduling in Figure 1b).

The argumentation made for operations and FUs can be extrapolated to variables and storage resources, and to data transfers and routing resources.

In this paper we propose an algorithm able to offer solutions like the ones presented above. These scheduling and allocation methodologies lead to smaller datapaths where the number and widths of the HW resources are independent of the number and widths of the specification operations and variables.

2. RELATED WORK
Multiple-precision behaviours appear both in SW and HW, especially in the development of DSP applications.

The problem to be solved in DSP SW development consists in the transformation of the original multiple–precision specification into another one with a unique width. This obliges to use the truncation and extension operators to make uniform the operations widths, with the consequent loss of precision in the first case, and waste of HW in the second one. Recent research in multiple-precision SW derives fixed-point implementations from floating-point or infinite-precision descriptions [1] [8].

Within the HW field there has been little research in HLS for multiple-precision specifications. Up to now most systems have adopted trivial solutions, balancing the number of operations executed per cycle, and binding operations to FUs with identical widths [4] [5]. More efficient systems execute operations over wider FUs, extending the operands when necessary, and discarding some bits of the solutions produced [2] [3]. Nevertheless, some authors admit that these trivial solutions are not good enough and suggest (but not implement) other alternatives, like the execution of an operation during several cycles [3].

* Supported by Spanish Government Grant CICYT TIC-99 0474

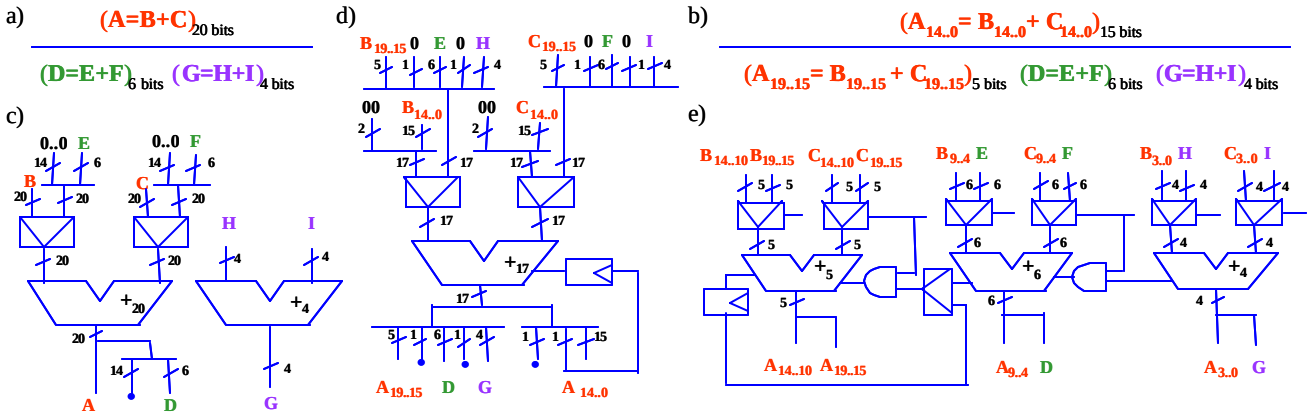

Figure 1. a) Scheduling proposed by a conventional algorithm, b) more efficient scheduling, c) implementation proposed by a conventional algorithm, d) and e) more efficient implementations for the 2nd scheduling.

3. PROPOSED ALGORITHM

Our algorithm performs the scheduling and allocation of multiple-precision specifications by using the design strategies shown in the introduction to deal with additions and those operations with an *additive* kernel. Remaining operations are treated in a conventional way.

The input is a multiple-precision specification whose *additive* operations (comparisons, maximum/minimum, subtractions, etc) are previously transformed into additions. The outputs are a datapath and a controller. In the datapaths provided the sum of all adder widths and the sum of all register widths is equal to, respectively, the maximum number of bits added and stored simultaneously in a cycle. And the sum of all multiplexer widths is less or equal to the maximum number of bits transmitted simultaneously in a cycle.

Figure 2 shows a schema of the proposed algorithm, and in the next subsections its central parts are explained.

3.1. Scheduling

We propose a variant of the popular and classical force-directed scheduling algorithm [7]. The intent of the original method is to minimize HW cost subject to a given time constraint (λ) by balancing the number of operations executed per cycle. The algorithm successively selects, among all operations and all execution cycles, an (operation, cycle) pair according to an estimation of the circuit cost called *force*.

Since the area of multiple-precision datapaths depends on the maximum number of bits computed simultaneously in a cycle, the intent of our variant is to minimize HW cost by balancing this number. Our method successively selects, among all operations and all execution cycles, a pair formed by an operation or an operation fragment, and an execution cycle according to a new *force* definition that takes into account the operations widths. As a result of the possible fragmentation an operation may be scheduled in a set of non-necessarily consecutive cycles.

In order to calculate the new *force* measure, the *time frame* of every operation (set of cycles an operation may start its execution) is calculated first:

$$TF_{op} = \left\{ c \in N \;\middle|\; \acute{o}_{ASAP}(op) \le c \le \acute{o}_{ALAP}(op) \right\}$$

Assuming that all bindings to feasible execution cycles have equal probability, operation *op* covers execution cycle *c* with probability:

$$P(op,c) = \begin{cases} \dfrac{1}{\|TF_{op}\|} & \text{if } c \in TF_{op} \\ 0 & \text{otherwise} \end{cases}$$

The *estimated operation cost* in cycle *c*, is the average value of the number of operation widths covering execution cycle *c*:

$$EOC(c) = \sum_{op \in EOP_c} \left(\text{width}(op) \cdot P(op,c) \right)$$

$$\text{where} \quad EOP_c = \left\{ op \in OP \;\middle|\; c \in TF_{op} \right\}$$

Force is a rather intuitive measure for how desirable a certain operation to cycle binding is, and its basic formula is:

$$F(op,c) = EOC(c) - \sum_{i \in TF_{op}} \frac{EOC(i)}{\|TF_{op}\|}$$

The smaller or more negative this expression becomes, the more preferable is an operation to cycle binding. Of course, an operation to cycle binding results in consequences that need to be considered. Fixing the time frame of an operation possibly affects the time frames of data dependent operations, resulting in additional changes of the distribution graph. In our complete new *force* definition this has been taken into account as in the complete original *force* definition [7].

Our algorithm binds operations (or fragments) to cycles subject to the following *upper bound*:

$$\frac{\sum_{op \in OP} \text{widths}(op)}{l} + \text{error\_margin}$$

which corresponds to the most uniform operation bits distribution among cycles plus a certain error margin. The error margin is

```
INPUT: DFG
OUTPUT: Datapath & Controller
BEGIN
  AdditiveOperations2Additions(OP);
  Unscheduled = OP;                                   Scheduling
  REPEAT                                              Phase
    F = CalculateForces(Unscheduled);
    BestOC = SelectMinForce(F);
    IF Bound < (BitsSched(BestOC.cycle)+Width(BestOC.op))
    THEN
      width1 = Bound - BitsSched(BestOC.cycle);
      width2 = Width(BestOC.op) - width1;
      Fragment(BestOC.op, width1, width2, op1, op2);
      Schedule(op1, BestOC.cycle);
      Add(Unscheduled, op2);
      FOR ope IN (Suc(BestOC.op) ∪ Pred(BestOC.op)) DO
        Fragment(ope, width1, width2, op1, op2);
        Remove(Unscheduled, ope);
        Add(Unscheduled, op1);
        Add(Unscheduled, op2);
    ELSE Schedule(BestOC.op, BestOC.cycle);
    END IF;
    Remove(Unscheduled, BestOC.op);
    UpdateDependencies(Unscheduled);
  UNTIL Unscheduled = Ø;

  REPEAT                                              Allocation
    C = CalculateCandidates(REG);                     Phase
    VL = SelectValidLocations(C, FU, REG);
    IS = CalculateIS(C, VL, FU, REG);
    BestCL = SelectBestCandidate(C, VL, IS);
    Allocate(BestCL, FU, REG);
    Remove(C, BestCL);
  UNTIL C = Ø;
  CompactDatapath(FU, REG);
  CreateInterconnectio ns(FU, REG, Datapath);
END;
```

Figure 2. Algorithm to perform the HLS of multiple-precision specifications.

necessary because due to the data dependencies this perfect distribution is not always reachable. Experimental results have measured this margin in the 15% to 20% increase range.

The scheduling algorithm consists in a loop which finishes when all the operations and all the operation fragments have been scheduled. In every step an operation and one of its time frame cycles are selected using the *force* measure. If the number of bits already scheduled in the selected cycle plus the width of the selected operation does not reach the *upper bound*, then the operation is scheduled in the cycle. Otherwise the operation is fragmented into two new operations. The width of one of these fragments is *upper bound* minus number of bits already scheduled in the selected cycle. This fragment is scheduled in the cycle, and the other one remains unscheduled with its original time frame. In order to avoid reductions in the time frames of the predecessors and successors of the fragmented operation, they are also fragmented. After scheduling an operation, or an operation fragment, the data dependencies of its successors and predecessors need to be updated.

When an operation is scheduled in several cycles, the result is calculated starting in the earliest cycle from the least significant bits, and storing the partial results until the latest execution cycle. An additional 1-bit register is also needed to store the partial carry signal.

3.2. Allocation

This phase is performed in three steps, first a virtual resources selection and binding of operations and variables take place; next, the partial datapath is compacted; and finally, it is completed with the routing resources. It should be highlighted that the set of operations and variables the algorithm allocates is not the original one, but the obtained after the scheduling phase.

The aim of the first step is to construct a datapath of as many 1-bit adders and 1-bit registers as, respectively, the maximum number of bits added and stored simultaneously in a cycle.

During this phase the algorithm works with a virtual array of linked 1-bit adders. The carry-out of each adder is connected, through an *and* gate, to the carry-in of the one on its left. These *and* gates are used to enable the carry propagation when necessary.

Before explaining this phase of the algorithm some concepts need to be defined:

Candidate: set of operations scheduled in different cycles and aligned in a particular way (different alignments result in different candidates). Figures 3a) and 3b) show respectively, a scheduling of a multiple-precision specification, and all the possible *candidates* formed by two of its operations.

Location of a candidate: set of contiguous 1-bit FUs which are not busy during the cycles in which the *candidate* operations are scheduled. Figure 3c) shows the two possible *locations* of one of the *candidates* drawn in Figure 3b).

Valid location: *location* that leaves enough contiguous regions of 1-bit FUs to allocate the still not allocated operations. Only the first *location* shown in Figure 3c) is a *valid location*.

Interconnection saving: cost function used to select the best pair (*candidate*, *location*). The *interconnection saving* of allocating a *candidate* in a *location* takes into account the number of bits of the *candidate* operands and results which can be stored in the

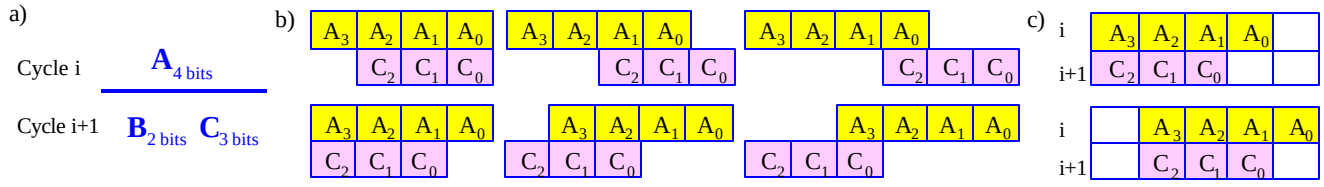

Figure 3. a) Possible scheduling for a multiple-precision specification, b) all possible *candidates* formed by operations *A* and *C*, c) all possible *locations* of one of these *candidates*.

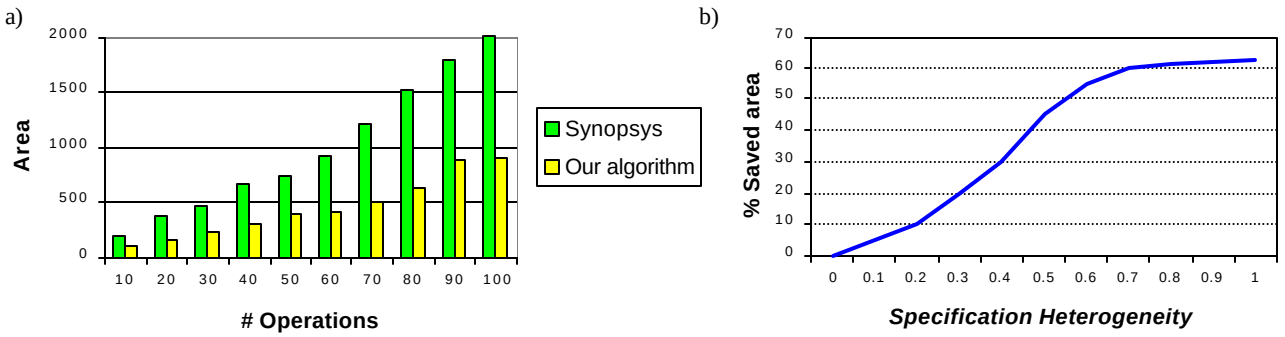

Figure 4. a) Average area of some circuits synthesized by Synopsys and our algorithm, b) percentage of area saved by the proposed algorithm in comparison with Synopsys implementations.

same registers, and the number of bits of the *candidate* operands and results which can be stored in the registers used by the operations already allocated in the selected *location*.

This phase of the algorithm consists in a loop which finishes when all the operations and all the variables have been allocated. In every step a subset of the *candidates* and a subset of their *valid locations* are created. The criteria to select these subsets are explained in [6]. Next the algorithm calculates the *interconnection saving* of each *candidate* in every *valid location*, and finally, the *candidate* with the maximum *interconnection saving* is allocated.

The allocation of a *candidate* consists in the individual allocation of each of its constituent operations to a set of contiguous 1-bit FUs, and the allocation of their respective operands and results to a set of 1-bit registers (not necessarily contiguous). In order to reduce the interconnection area, the algorithm tries to allocate all the *candidate* right operands to the same set of registers, and the same applies to the left operands and the results. It also tries first the registers used to store the variables of the operations already allocated in the selected *location*.

Once all the specification operations have been allocated, the algorithm joins those 1-bit-width FUs which must remain linked to propagate the carry signal during all the cycles to form the final FUs. And those 1-bit-width registers with compatible load signals are also joined to form wider ones.

4. EXPERIMENTAL RESULTS

In order to measure the quality of the implementations obtained by our algorithm we have compared them with the ones proposed by Synopsys Behavioral Compiler. We have synthesized a wide collection of randomly generated multiple-precision specifications, which ranges from 10 to 100 *additive* operations with latencies from 4 to 30 cycles. Results show that the area of the implementations obtained by our approach is always smaller than the area of the circuits proposed by the commercial tool. For the circuits synthesized the average saved area ranges between 45% and 55%. Figure 4a) shows graphically some of these results.

In order to explain these results it is necessary to identify the factors which influence the quality of the implementations obtained by both methods. The area of our implementations only depends on the specification size, the data dependencies, and the given time constraint. However, the area of the implementations obtained by conventional algorithms also depends on the number and widths of the specification operations. Figure 4b) shows the amount of area saved by our algorithm in function of the *specification heterogeneity* (ratio of the number of different widths in the specification to the number of operations).

5. CONCLUSION

This paper presents novel scheduling and allocation algorithms for multiple-precision specifications, which balance the number of bits calculated per cycle and guarantee a maximum bit-level reuse of adders and registers. The implementations obtained are multiple-precision datapaths, where the number and widths of the resources are independent of the specification operations.

Experimental results show that the implementations given by our algorithm have smaller area than the ones offered by commercial tools, and also that the amount of saved area grows with the *specification heterogeneity.*

Future work will include the generalization of the algorithm to increase the reuse of all kinds of FUs, and the integration of the scheduling and allocation phases, what we expect could even report better results.

6. REFERENCES

[1] Cmar, R., Rijnders, L., Schaumont, P., Vernalde, S., and Bolsens, I. A methodology and design environment for DSP ASIC fixed point refinement. In Proceedings of DATE, 1999.

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

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

[4] Huang, C., Chen, Y., Lin, Y., and Hsu, Y. Data path allocation based on bipartite weighted matching. In Proceedings of DAC, 1990.

[5] Küçükçakar, K., and Parker, A. Data Path tradeoffs using MABAL. In Proceedings of DAC, 1990.

[6] Molina, M.C., Mendías, J.M., and Hermida, R. Multiple-Precision Circuits Allocation Independent of Data-Objects Length. In Proceedings of DATE, 2002.

[7] Paulin, P.G., and Knight,J.P. Force-directed scheduling for the behavioral synthesis of ASICS. IEEE Trans. on CAD, 1989.

[8] Willens, M., Bürsgens, V., Keding, H., Grötker, T., and Meyer, M. System-level fixed-point design based on an interpolative approach. In Proceedings of DAC, 1997.