Multiple-Precision Circuits Allocation Independent of Data-Objects Length*

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

Abstract

This paper presents an heuristic method to solve the combined resource selection and binding problems for the high-level synthesis of multiple-precision specifications.

Traditionally, the number of functional (and storage) units in a datapath 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, this way of acting could produce a considerable waste of area.

To overcome this problem, we propose the selection of the set of resources taking into account the only truly relevant aspect: the maximum number of bits calculated and stored simultaneously in a cycle. The implementation obtained is a multiple-precision datapath, where the number and widths of the resources are independent of the specification operations and data objects.

1. Introduction

Some fragments of a multiple-precision specification are shown in Fig. 1a, and a possible scheduling in Fig. 1b. Traditional high–level synthesis (HLS) allocation algorithms provide solutions where each operation is implemented over a unique functional unit of the same or greater width. This intuitive solution is shown in Fig. 1c. However, it is possible to obtain circuits with smaller area, following different design strategies.

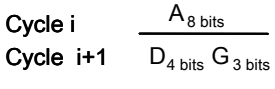
Two additions scheduled in the same cycle can be executed simultaneously over the same functional unit, linking them together and avoiding the carry signal propagation affect the results. In order to cut the carry chain the operands should simply be separated with a 0, as it is shown in Fig. 1d.

An addition can be allocated to a set of functional units, if these are linked to propagate the carry signal. The carry out of the functional unit which calculates the least significant bits should be supplied as the carry in of the functional unit which calculates the most significant ones. In Fig. 1e, the *and* gate is used to allow the carry propagation during the first cycle, and to avoid it during the second one. The control line used to permit or not the carry propagation is the same used by the multiplexer to select the operands.

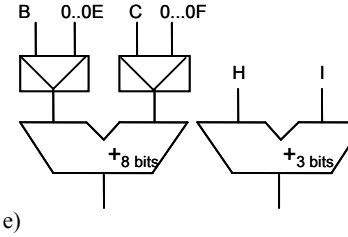
A similar problem appears in the storage units selection and binding processes. Traditional HLS systems are able to reuse registers to store variables of the same or less width. However, if some variables are divided into several fragments and stored each of them in a different register, or different variables are stored simultaneously in the same register, it is possible to obtain notorious area savings.

a)
```
...
main: process
...
variable A, B, C :  std_logic_vector (7  downto 0);
variable D, E, F :  std_logic_vector (3  downto 0);
variable G, H, I :  std_logic_vector (2  downto 0);
...
begin
...
A := B + C ;
D := E + F ;
G := H + I ;
...
end process main;
```

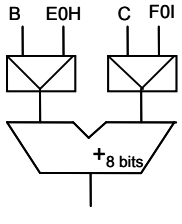
b)



c)



d)



e)

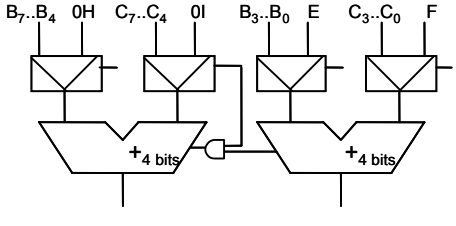


Fig.1. a) fragments of a VHDL multiple-precision specification, b) possible scheduling, c) implementation proposed by more traditional algorithms, d) and e) more efficient implementations.

* Supported by the Spanish Government Research Grant TIC99-0474

Additionally, it must be highlighted that, in general, the use of any of the cited design strategies does not imply the use of additional multiplexers.

In this paper we propose an algorithm able to offer solutions like the presented before. This allocation style leads to datapaths where the number and widths of the resources are independent of the specification operations and variables, and the amount of area due to the functional and storage units is the minimum necessary. The algorithm performs simultaneously the functional and storage units selection and binding, guided by the main objective of saving the maximum amount of interconnection area.

2. Related work

The multiple-precision problem appears both in software and hardware, specially in the development of DSP applications.

The software DSP computation model consists of a set of predesigned fixed-wordlength computational units responsible of implementing all the operations. So, the problem to be solved consists in the transformation of the original multiple–lenght specification into another one with a unique and uniform wordlength. This obligates to use the truncation and extension operators to adapt the operations widths, with the consequent loss of precision in the first case, and the resulting waste of HW in the second one. Recent research in multiple-precision software problems takes fixed-point implementations to be derived both from floating-point or infinite-precision descriptions. Some examples of these are [1], [2], [3], and [4]. In [5] the authors present a method to find the minimum number of bits for both integer and floating point variables, used in high level signal and image processing algorithms described in MATLAB.

Within the hardware field there has been little research in HLS for multiple-precision specifications. Up to now, most systems have adopted trivial solutions, as treating separately every different precision of the original specification; they bind operations to functional units with identical widths. Some examples are presented in [6], [7], and [8].

More efficient systems are able to implement operations over wider functional resources, filling in the operands with 0's, and discarding some bits of the solution produced. Some of them are the ones proposed in [9], [10], [11], and [12]. In [12] the authors also take into account the operation widths during the scheduling phase, which is performed as the same time as the resource selection and binding.

Nevertheless, some authors admit that these trivial solutions are not good enough and suggest (but not implement) another alternatives, like the execution of an operation during several cycles [11], or the execution of two multiplications over a unique multiplier during the same cycle [10]. In the first case, the least significant bits of an operation can be calculated in a cycle, and the remaining ones in subsequent cycles, by storing not only the partial results but also the partial carries. In the second case, the authors study FPGA's implementations of multipliers structures with more than two operand entry points.

3. Proposed algorithm

The actual version of our algorithm performs the resource selection and binding of multiple-precision specifications. The algorithm treats in a special way all the operations with an *additive* kernel (comparisons, maximum/minimum, subtractions, ...), and uses a classical functional unit selection and binding algorithm (not shown in this paper) to deal with non *additive* operations.

It takes as inputs both a circuit specification and a scheduling given by any HLS tool. The output is a complete multiple-precision datapath and a controller. The datapath is composed of a set of adders, and other *non–additive* functional units, some glue logic to link carry chains and to execute additive operations over adders, a set of storage units, and a set of multiplexers.

The datapaths provided have always two features:

1) The sum of all *additive* functional units widths is equal to the maximum number of bits computed simultaneously in a cycle.

$$\sum_{i=1}^{num\_fu} width\_fu(fu_i) = \max_{j=1}^{num\_cycles} \left(\sum_{k=1}^{num\_ope(j)} width\_ope(ope_k) \right)$$

2) The sum of all registers widths is equal to the maximum number of bits stored simultaneously in a cycle.

$$\sum_{i=1}^{num\_reg} width\_reg(reg_i) = \max_{j=1}^{num\_cycles} \left(\sum_{k=1}^{num\_\mathrm{var}(j)} width\_\mathrm{var}(\mathrm{var}_k) \right)$$

These novel features result in datapaths where the resources widths are, in general, independent of the circuit specification, that is, of the operations and data objects used.

The algorithm is divided into four phases. During the first one all the *additive* operations of the original specification are transformed into additions. In the second phase an allocation, based on the binding of sets of operations to a single functional unit, is performed. During the third phase the circuit obtained is transformed into a more compact one, and the number of control signals is reduced. The selection and binding of the routing resources take place during the fourth and last phase.

The figure Fig. 2 shows a schema of the algorithm, in which each phase has been separated of the others by a dotted line. The next subsections explain in detail the central phases of the algorithm proposed.

```
INPUTS:
Scheduled DFG
OUTPUTS:
Complete Datapath and Controller
BEGIN
    Preprocessing (Ope)
    REPEAT
        C = Calculate_Candidates (Reg)
        L = Select_Possible_Locations (C, FU, Reg)
        IS = Calculate_Interconnect_Savings (C, FU, Reg)
        Best = Select_Best_Candidate (C, L, IS)
        Allocate (Best, FU, Reg)
        Remove (C, Best)
    UNTIL C = 0
    Compact_Datapath (FU, Reg)
    Create_Interconnections (FU, Reg, Datapath)
END
```

Ope: List of specified operations. **C**: List of candidates, **L**: list of locations, and **IC** : list of interconnections costs.
Reg: Array of size *n* x *m*, with *n* = maximum number of bits stored simultaneously in a cycle, and *m* = number of cycles. A 1-bit variable stored during cycle *x* in the 1-bit register *y* occupies the position [*y*, *x*] of the array.
FU: Array of size *n* x *m*, with *n* = maximum number of bits calculated simultaneously in a cycle, and *m* = number of cycles. A 1-bit operation scheduled in cycle *x*, and implemented over the 1-bit adder *y* occupies the position [*y*, *x*] of the array.
Best: Candidate with maximum interconnection saving.
Datapath: Final implementation of the problem.

Fig. 2. Algorithm to solve the combined resource selection and binding problems of multiple-precision specifications.

3.1. Original specification transformation

Since our algorithm optimises the adder usage, the preprocessing phase transforms all the specification operations with an *additive* kernel into additions. For this purpose we use a set of equations whose correctness has been previously proven. The set of equations applied, as well as the way they are applied is explained in [13].

3.2. Functional and storage resources selection and binding

The objective of this phase is to construct a datapath with a minimum interconnection area, composed of as many 1-bit adders and 1-bit registers as, respectively, the maximum number of bits calculated and stored simultaneously in a cycle. Some of these 1-bit resources will be compacted in the next phase to form wider functional units and registers.

During this phase the algorithm works with an 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, as it is shown in Fig. 3. An operation can only be allocated to a contiguous portion of this array, because a less restrictive allocation policy would not result in higher HW savings and could create combinational loops.

Next some concepts will be defined in order to ease the explanation of this phase of the algorithm. With the same purpose we will use a simple example , whose circuit specification and scheduling are shown in Fig. 4a and Fig. 4b respectively.

Candidate: set of operations with aligned operands scheduled in different cycles (different alignments result in different candidates). Since the creation of all these sets requires exponential time, the algorithm only calculates a limited number of them. These are the ones made up of a minimum of one operation, and a maximum of one operation per cycle:whose left operands have a maximum number of bits in common (because either they belong to the same variable, or they come from the same previously allocated HW resource).

1) whose right operands have a maximum number of bits in common.
2) whose left operands have the same size, and not overlapped lifetimes.
3) whose right operands have the same size, and not overlapped lifetimes.

In the example the creation of all *candidates* amounts a total of 36 possibilities. Fig. 4c shows all the possible *candidates* made of operations *D* and *G*, corresponding to 3 different operands alignments.

Supposing that variables *B(3 downto 1)* and *K* are stored in the same register, and that operations *D* and *G* have not overlapped operands lifetimes, the algorithm only calculates 7 of the 36 previously cited *candidates*. These ones are shown in Fig. 4d.

Location of a candidate: set of contiguous 1-bit functional units which are not busy during the cycles in which the *candidate* operations are scheduled. The creation of all of them is again infeasible, so the algorithm only calculates the next ones:

1) those inside the array of functional units where there are operations allocated which have some operands bits in common with any of the *candidate* operations.
2) the *location* situated the nearest to the left edge of the array of functional units.

Fig. 3. Array of 1-bit adders linked by 2-input and gates.

3) the *location* situated the nearest to the right edge of the array of functional units.

The *locations* 2) and 3), have been taken into consideration because they fragment minimally the array of functional units, leaving wider contiguous regions of 1-bit functional units to allocate the remaining operations. These two *locations* are the ones with the best possibilities of allowing the algorithm to reach valid solutions.

In the example, if there is not any operation allocated yet to the array of functional units, there are 4 possible *locations* for the *candidate* formed by *A* and *J*. Nevertheless the algorithm only calculates 2 of them, which are shown in fig. 4e.

Valid location: *location* that leaves enough contiguous regions of 1-bit functional units to allocate the still not allocated operations. The *locations* shown in fig. 4e are not valid ones because they make impossible the allocation of the operations scheduled in cycle i+1.

Interconnection saving:

IS(C,L) = BitsOpe(C) + Bits_Res(C) + BitsLoc(C,L)

IS(C,L): interconnection saving of the *candidate* C in the *location* L, being:

BitsOpe(C): number of bits of the *candidate* C left and right operands which are able to come from the same sources.

BitsRes(C): number of bits of the *candidate* C results which can be stored in the same register.

BitsLoc(C,L): number of *candidate* C operands bits which can be stored in the same registers as the ones used by the operations already allocated in *location* L.

This phase of the algorithm consists of a loop. In each iteration the *candidates* and their *locations* are created. When a *candidate* has no any *location*, the algorithm determines the operations responsible for it. If the problem can be solved changing these operations for another ones, in accordance with *candidates* definition, then they are changed. If not, these operations are removed from the *candidate*. In both cases, the algorithm recalculates the set of *locations*.

After creating the *candidates* and selecting the *valid locations*, the algorithm calculates, for each *location* of every *candidate*, the *interconnection saving*. And finally, the *candidate* with the maximum *interconnection saving* is allocated.

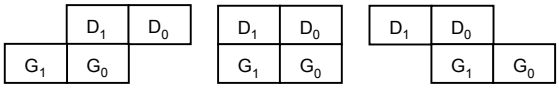
The allocation of a *candidate* consists in the individual allocation of each of its constituent operations to a set of contiguous 1-bit functional units, and the allocation of their respective operands and results to a set of 1-bit registers (not necessary contiguous). In order to reduce the interconnection area, the algorithm always tries to allocate all the right operands of the *candidate* operations to the same set of registers, all the left operands to the same set of registers, and the results to the same set of registers. If there were other operations already allocated to the *location* selected, then the algorithm would try to use the same registers used by these operations to store the *candidate* operands. This is possible if both the lifetimes of the variables already stored in the registers and those of the *candidate* operands do not overlap. If not, the algorithm selects a set of registers able to store the *candidate* variables. The loop finishes when all the operations and variables have been allocated.

a)

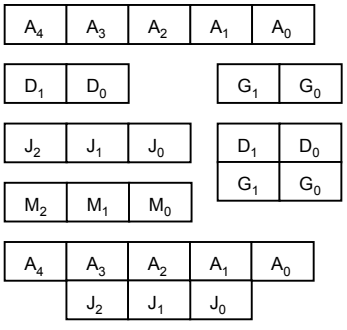
```
...
main: process
   variable A, B, C: std_logic_vector (4 downto 0);
   variable D, E, F: std_logic_vector (1 downto 0);
   variable G, H, I: std_logic_vector (1 downto 0);
   variable J, K, L: std_logic_vector (2 downto 0);
   variable M, N, O: std_logic_vector (2 downto 0);
begin
   A := B + C;
   D := E + F;
   G := H + I;
   J := K + L;
   M := N + O;
end process main;
...
```

b)

c)



d)



e)

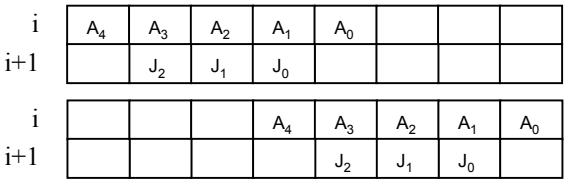


Fig. 4. a) Example of multiple-precision specification, b) scheduling proposed by a HLS tool, c) and d) some candidates, e) locations of a candidate.

3.3. Compacting the datapath

Once all the operations of the original specification have been allocated, the algorithm joins those 1-bit-width functional units which must remain linked to propagate the carry signal during all the cycles of the given scheduling, to form the definitively functional units. Every 1-bit-width register with compatible load signals are also joined to form greater ones, reducing in this case the number of control signals which the controller should provide.

4. Experimental results

In this section we have measured the quality of the solutions given by the proposed algorithm in terms of those proposed by Synopsys Behavioral Compiler. With that purpose we have synthesised a collection of 10 random multiple-precision specifications, with the characteristics shown in Table 1.

Table 1 shows the results given by Synopsys Behavioral Compiler, our proposed algorithm and the percentage of area saved in columns named S, A and % respectively. For each example it has been specified the number of operatios (Num. Oper.), the number of different widths in the specification (Num. Width.), and the sum of adder and register widths used in the circuit implementation (Adder Bits, and Register Bits respectively).

Results show that the area of the implementations obtained by the algorithm is always smaller than or equal to the area of the circuits proposed by Synopsys. The area savings depend neither on the number of operations of the synthesized specifications nor on the latency. The unique aspect with real influence in the area of the final implementation is the homogeneity of the operations widths scheduled in every cycle.

Num. Oper.	Num. Width.	Latency	Adder Bits			Register Bits		
			S	A	%	S	A	%
8	7	3	20	18	10	30	23	10
10	4	3	19	15	21	26	24	8
12	6	3	26	22	15	38	32	16
15	5	4	29	20	31	32	28	13
16	7	4	53	23	56	31	24	23
21	5	4	23	18	21	29	25	14
25	8	5	38	16	57	27	21	22
30	7	7	27	21	22	33	27	18
35	6	8	53	26	50	41	35	15
40	10	10	30	22	26	38	30	21

Table 1. Experimental results.

5. Conclusion

This paper presents a novel allocation algorithm for multiple-precision specifications, which guarantees a maximum reuse of adders and registers, and at the same time it also tries to reduce the interconnection area. The circuit implementations obtained are multiple-precision datapaths, where the number and widths of the resources are independent of the circuit specification.

Experimental results obtained show that the implementations given by the algorithm have smaller area than the ones offered by commercial tools, and that the more heterogeneous the operations widths scheduled in every cycle are, the more efficient in terms of the amount of area saved our algorithm becomes.

Future work will include the generalization of the algorithm to increase the reuse of all kinds of functional units, not only the *additive* ones.

References

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

[2] G.A. Constantinides, P.Y.K. Cheung, W.Luk. "Multiple precision for resource minimization". *Proc. IEEE Symposium on FCCM, 2000.*

[3] K. Kum, and W. Sung. "Word-length optimization for high-level synthesis of digital signal processing systems". *Proc. IEEE Int. Workshop on Signal Processing Systems SIPS'98.*

[4] M. Willens, V. Bürsgens, H. Keding, T.Grötker, and M. Meyer. "System-level fixed-point design based on an interpolative approach". *Proc. ACM/IEEE DAC 1997.*

[5] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee. "Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGA's". *Proc. DATE, 2001.*

[6] C. Huang, Y. Chen, Y. Lin, and Y. Hsu. "Data path allocation based on bipartite weighted matching". *Proc. 27th ACM/IEEE DAC, 1990.*

[7] K. Küçükçakar, and A. Parker. "Data Path tradeoffs using MABAL". *Proc. ACM/IEEE DAC, 1990.*

[8] F. Tsai, and Y. Hsu. "Data path construction and refinement". *Proc. ICCDCS, vol. CAD-5, nº3, July 1986.*

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

[10] G.A. Constantinides, P.Y.K. Cheung, and W.Luk. "Multiple-wordlength resource binding". *In H. Gruenbacher and R. Hartenstein, editors, Field-Programmable Logic: The Roadmap to reconfigurable systems, LNCS, 2000.*

[11] M. Ercegovac, D. Kirovski, and M. Potkonjak. "Low-power behavioural synthesis optimization using multiple precision arithmetic". *Proc. ACM/IEEE DAC, 1999.*

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

[13] J.M. Mendías, R. Hermida, M. Fernández. "Formal techniques for hardware allocation". *Proc. International Conference on VLSI Design, VLSI'97, Jan. 1997.*