

Optimising High-Level Synthesis for Self-Checking Arithmetic Circuits

A. Antola, V. Piuri, MG. Sami
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. Da Vinci 32, 20133 Milano, Italy
e-mail: antola,piuri,sami@elet.polimi.it

Abstract

Introduction of self-checking capacity in arithmetic systems since the initial steps of high-level synthesis is taken into account, as an alternative to conventional solutions that adopt ad-hoc coding or comparable techniques after the register-level architecture has been fully defined. A technique based on initial partitioning of the Data Flow Graph into detectable subgraphs is proposed, by which all single errors appearing within one subgraph are detected; an algorithm leading to optimize resource sharing (allocation and binding of both functional units and registers) while keeping minimum latency and optimum number of checkers is discussed.

1. Introduction

High-level synthesis has been a subject of research and development since several years now; starting from an algorithmic description of a specific device's computation, techniques have been defined to derive an intermediate graph description (usually consisting of a *Control Flow Graph* - CFG - and a *Data Flow Graph* - DFG) used to obtain optimum scheduling, allocation and binding [1,2]. Main figures of merit are *latency*, *throughput* and *resources*; the final product of high-level synthesis consists of an RT-level datapath and of a transition table for the control FSM, and the above figures of merit are evaluated in a preliminary fashion on the basis of parameters characterising the library cells used for the subsequent architectural synthesis.

A particular problem, not usually taken into account, concerns design of *self-checking systems*; in fact, self-checking capability is usually accounted for at a lower level, e.g., by introducing suitable modifications in the state table and using particular coding techniques in the state assignment phase of an FSM [3,4] or by adopting coded data and related self-checking operating units in the datapath [5]. Such solutions are introduced when both datapath architecture and FSM state table have been completely defined, i.e., after high-level synthesis has been carried out; as a consequence, possible optimisation may be performed only in the logic synthesis phase, without taking into account behavioural characteristics of the algorithm implemented by the ASIC.

Since the number of states for the controlling FSM is directly determined by the number of control steps over which the DFG is scheduled, while width of the related output word derives from allocation and binding of the datapath, it follows that datapath synthesis is the first problem to be approached.

Conventional solutions tend to protect individual operations rather than to exploit the relationships between application-specific algorithm and detection capacity of the coding techniques; typically, checking will be associated with each individual functional unit (operator) introduced during high-level synthesis (and, possibly, with individual registers as well); this easily results in excessive area overhead. For the same reason, there may be also a deterioration in both latency and throughput, since the control step must now accommodate checking as well as nominal operation. We suggest as an alternative to introduce self-checking requirements already in the initial steps of high-level synthesis, and to modify the synthesis approach so as to optimise area and performances while granting autonomous error detection.

For the reasons already suggested, we concentrate on data path synthesis, considering, in particular, synthesis of algorithms consisting of *arithmetic operations* only. This last-named restriction, while not forcing basic limitations of our approach, allows us to adopt a single and cost-effective coding solution throughout the whole circuit and thus to avoid insertion of transcoding operations in the algorithmic flow. We make use of *arithmetic codes*, e.g., AN codes or residue codes [5]; for such codes it has been already proved that - for some specific computations such as convolution - it is possible to drastically reduce the number of checkers without impairing the detection capability [6,7]. Our approach can be briefly summarised as follows:

- a) the Data Flow Graph (DFG) is analysed, so as to extract all *Maximum Detectable Subgraphs (MDS's)*, an *MDS* being defined as a subgraph provided with k *checked inputs* and one *checked output* o_j , such that any single error within the subgraph is detected by checking the output o_j , and that the subgraph itself is not wholly contained within any other proper subgraph of the DFG having the same properties;
- b) a minimum cover of the DFG consisting of *MDS's* is determined: such cover corresponds to the minimum number of checking points that grant complete detectability without aliasing (always assuming the single-error model within each *MDS*);
- c) to avoid conflicts or inconsistencies in scheduling and allocation, each operation (node of the DFG) must be checked only once, i.e., the DFG must actually be partitioned into a set of disjoint (possibly non maximum) Detectable Subgraphs; the cover previously identified provides a guide for such partitioning;
- d) scheduling and resource sharing are performed so as to minimise hardware costs while keeping minimum latency and granting single-error detectability.

Definitions and properties of *MDS's* and of detectable covers have been presented in [9], where a first time-constrained scheduling technique reducing the number of checker required while limiting operator cost was presented. These preliminary results are recalled in Section 2. We consider then in particular the problem of resource sharing (i.e., "allocation and binding") for functional units and for registers; solutions based on immediate use of *MDS's* may lead to increase the number of distinct functional units and, even more (even though less immediately noticeable) the number of registers. Heuristics allowing to select a "better" detectable cover of the DFG and to partition *MDS's* so as to reduce costs in terms of functional units and registers without increasing the number of checkers will be presented in Section 3. A solution that introduces allocation as a guideline already during identification of a detectable cover, and in the subsequent scheduling phase as well, is outlined in Section 4.

2. The basic scheduling technique.

The basic scheduling approach presented in [9] has the aim of reducing the number of checkers inserted in the datapath while keeping overall single-error detection capacity; in fact, single-error detection is granted locally to subsets of the datapath, so that a measure of multiple-error detection capacity is also allowed.

Conditions for aliasing are first identified: aliasing in a network of arithmetic devices based on single-error-detecting codes, occurs whenever, in the presence of a single fault, the error syndrome for a result affected by the error is divisible by 3: this may happen either if one input to a multiplier is faulty or if reconvergent paths in the DFG lead to a final error syndrome divisible by 3.

Based on the above considerations, an algorithm (with $O(n)$ complexity, if n is the number of nodes in the DFG) allows to detect all *MDS*'s in a DFG. A detectable cover of the DFG being defined as a set of detectable subgraphs such that each node in the DFG belongs to (at least) one *DS* in the cover, definition of *minimum detectable cover* is straightforward.

It has been proved that a cover C of the DFG consisting of a set of *DS*'s allows detection with respect to any single error within any *DS*, if a checker is activated at each *DS*'s checking point, and that, if $DS(o_1)$ and $DS(o_2)$ are two *DS*'s such that $DS(o_1) \cap DS(o_2) \neq \emptyset$, considering $DS^* = DS(o_1) - DS(o_1) \cap DS(o_2)$, the following properties hold:

- 1) the inputs to DS^* coming from $DS(o_1) \cap DS(o_2)$ are implicitly checked,
- 2) DS^* is a detectable subgraph provided $DS(o_2)$ (or a *DS* completely including it) is checked as a whole.

This allows to partition the DFG into *disjoint checked DS*'s. Note that the single-error assumption concerning each *DS* implies that:

- all operations within the *DS* must be mapped onto distinct operators,
- all variables within the *DS* must be mapped onto distinct registers.

Given all the above, in [9] a scheduling approach was presented, based on the *DS* partitioning, that allowed to keep minimum latency as the main requirement and to reduce the area requirements consisting of operators as well as of checkers as a secondary figure of merit. The area cost factor was evaluated by comparison with an upper bound by which each operator adopted for the final mapping was provided with an individual checker¹; in such case, the number of checkers directly corresponds to the area increase. A less pessimistic upper bound can be identified on the optimum non-checked scheduling of the same DFG, by considering the maximum number of operators within any control step and allowing checkers to be multiplexed for different data in the various control steps (as it is done for checkers used in our optimised approach).

A modified time-constrained scheduling technique is adopted, based on the fact that - to account for the single-error assumption within each *DS* belonging to the detectable cover - operations within one *DS* must all be performed by distinct functional units (here called operators). Thus, even *mobility* of operations for scheduling purposes should be evaluated "across" different *DS*'s, while it is not a useful parameter (as far as resource minimization is concerned) within a single *DS*. Consider the sample DFG in figure 1, where a first partitioning into a minimum number of disjoint *DS*'s is presented (the original

¹ We assume that checking is performed on the contents of the register in which operation's results are saved; checking happens in parallel with use of such data, in the control step following the one corresponding to production of the results. This solution is adopted to avoid increasing the length of the control step.

minimum cover based on *MDS*'s would consist of the following *MDS*'s: *A*:{2*,6,9+}, *B*:{2*,3*,6,10,}, *C*:{4,7+,11,}, *D*:{1*,2*,5+,8+,12+,14+}, *E*:{4,7+,13+,15+}), and the mentioned scheduling technique has been applied. Operations 5 and 8, even though scheduled in two different control steps, will have to be allocated onto two distinct add operators, since they both belong to *DS D* and are thus both checked on *D*'s checking point. In other words, resource sharing may occur only across *DS*'s, never within one *DS*. For the example in figure 1, the total cost in terms of operators amounts to 2 multipliers, 2 subtractors, 3 adders and 2 checkers, while the theoretical minimum for the non-checked solution would have required 2 multipliers, 2 adders and 1 subtractor, to which 5 checkers should be added to provide for individual checking of each operator's results. As a first approximation, we assume identical cost for all types of operators, as well as for checkers and for registers - adoption of more realistic cost figures is immediate.

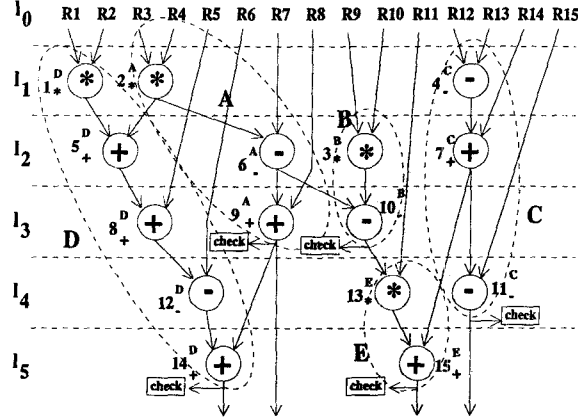


Figure 1: Sample scheduled DFG.

As seen in the example, straightforward *DS*-based scheduling can lead to increase the number of operators with respect to the non-checked optimum: the problem becomes even more relevant when register sharing is taken into account, since no register can be reused for two (or more) variables belonging to the same *DS*. In the next section, the modified allocation algorithms accounting for single-error assumptions will be outlined: it will then be seen how resource sharing can be used as a guiding constraint in *DS* partitioning, so as to reduce the number of operators and registers without increasing the number of checkers.

3. The allocation and sharing problem.

Allocation of operators can be performed by a straightforward extension of a conventional technique (see [2]) whereby operations of the same type belonging to the same *DS* are considered "not compatible" as far as mapping onto one single operator is concerned (the connectivity of the compatibility graph is correspondingly reduced). As an example, denote by *A1*, *A2*, *A3* the three adders required by the self-

checking schedule of the DFG in figure 1: the corresponding compatibility graph is shown in figure 2, whereby the following resource sharing can be determined:

$\{5,15\} \rightarrow A1$ $\{7,8\} \rightarrow A2$, $\{9,14\} \rightarrow A3$

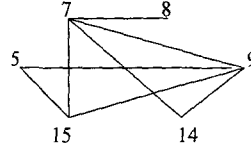


Figure 2: compatibility graph for adder sharing

Of the various possible bindings, the one balancing multiplexer complexity has been chosen.

While the number of distinct operators is already determined when scheduling is completed, only the actual sharing being then determined as above, register sharing (i.e., allocation and binding of variables to registers) has to be performed completely, since scheduling by itself does not directly provide the optimum number of registers required. Allocation of variables to registers is usually performed taking into account as the initial constraint the variables' lifetime: a register R_k , in which variable v_i has been stored, will be re-usable for storing a different variable only after the lifetime of v_i has expired. Register sharing can, again, be determined by creating a compatibility graph related to all variables in the DFG, two variables being compatible iff their lifetimes do not (even partially) overlap. As before, the minimum number of registers coincides with the minimum number of disjoint cliques covering the compatibility graph [10].

In our case, besides the variables' lifetime, additional constraints for register sharing arise whenever two variables belong to the same *DS*, in which case they must be allocated onto different registers to provide for the single-error model within each *DS*. The register-sharing algorithm must be consequently modified so as to take into account this further incompatibility between variables (two variables whose lifetimes do not overlap may share a register if and only if they belong to different *DS*'s). Large *DS*'s, as a consequence, may cause an undue increase in the number of registers whenever the *DS* extends over several control steps and involves variables whose lifetime would otherwise be short; let us initially introduce these aspects in an intuitive way, by referring to the example in figure 1, for which the non-checked optimum allocation would require *five* registers. Optimum register sharing for the self-checking solution, based on the partitioning and scheduling choice shown in figure 1, would require *six* registers. The main problem for this increase derives from *DS D*, that spans all control steps and includes a number of variables whose intrinsic lifetime would be just one control step but that cannot share any register among themselves due to the single-error assumption. A guideline for choice of *DS*'s could be already derived from this consideration, suggesting to partition overlapping *DS*'s so as to prefer *DS*'s extending over a reduced number of subsequent control steps and including as many operators and variables as possible in the chosen control steps. This may not offer a conclusive solution: an alternative partitioning to the one adopted in figure 1 is first modified by considering the initial *MDS*'s and selecting a different set of disjoint *DS*'s (precisely: **D**: {1, 2, 5, 8, 12, 14}, **A**: {6, 9}, **B**: {3, 10}, **C**: {4, 7, 11}, **E**: {13, 15}). This choice does not impact on number of checkers or on schedule: in fact, the register problem is not modified since **D** again spans all control steps. We now recall that it is possible (without impairing the checking capability of the whole cover) to split a *DS* into two disjoint ones, inserting a checker on lines that are not now internal to a *DS* and would not be implicitly checked by other *DS*'s.

We split therefore **D** into **D1** and **D2**, where the cut point has been chosen so as to split "lengthwise" the *DS* while inserting a checking point in a control step where no other checking points are scheduled (or, anyway, in a step where less than the total number of checking points required by the schedule are present). We introduce therefore the alternative cover in figure 3, which is not any more *minimal* (it requires six *DS*'s instead than five) but does not increase the number of checkers.

Incompatibility constraints imposed by variables belonging to a single *DS* and whose lifetimes would otherwise be disjoint have been relaxed: **D1** includes two variables that are alive in the same control step and would therefore be anyway incompatible, and even the relatively "long" **D2** extends over three control steps only. Optimum register allocation allows now to use only *five* registers, i.e., the theoretical minimum even for the non-checked solution: actually, even operator sharing is enhanced, such that the new allocation scheme requires *two* adders instead of the previous three.

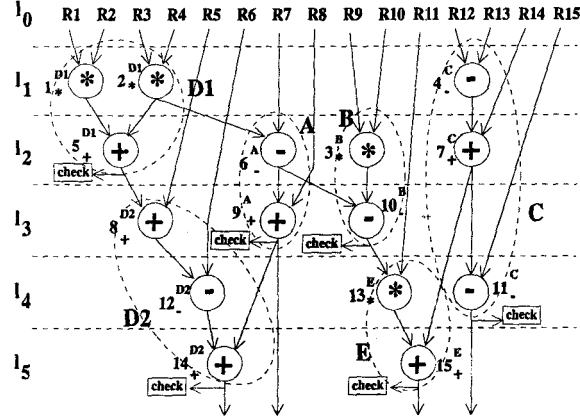


Figure 3: Modified partitioning of the DFG to optimise register sharing.

Let us now formalise the approach leading to better resource sharing: to this end, we introduce some preliminary notations:

- N_c is the maximum number of checking points present in any control step of the scheduled DFG (i.e., the number of foreseen checkers),
- $I'(DS^*)$ is the first control step in which a node belonging to DS^* is scheduled
- $I''(DS^*)$ is the last control step in which a node belonging to DS^* is scheduled.

The algorithm can then be described as follows:

- 1) A minimum cover based on *MDS*'s is determined; whenever necessary, a set of disjoint *DS*'s is extracted, without increasing the cardinality of the cover (i.e., without increasing the number of checking points required by the architecture to grant single-error coverage). In the presence of two

MDS 's $M1, M2$ such that $M1 \cap M2 \neq \emptyset$, selection of disjoint MDS 's is performed by attributing the shared subgraphs so as to:

- reduce the maximum path length in either DS , and
 - balance the complexity of both DS 's (evaluated in terms of operations, for each different operation type present in the DS 's). (The first above rule is given priority over the second one, since it impacts on register number as well as on operator number.)
- 2) An optimum time-constrained scheduling is effected on the DFG thus partitioned, minimising the number of distinct operators and checkers present in the final circuit (see [9]); scheduling allows to determine the control steps in which the checkers are active (it "freezes" the positions of checking points in the scheduled graph), as well as to compute the number of checkers required. Operator sharing is performed by the modified compatibility criterion, as shown in the example.
 - 3) An approximate lower bound to the number of distinct registers is evaluated by computing the maximum number of distinct variables simultaneously alive in any control step. (Alternatively, an exact lower bound can be determined by performing register, as well as operator, sharing on the unchecked DFG: this, anyway, could be excessively time-consuming for complex DFG's). *Register sharing* is then performed by the modified compatibility criterion.
 - 4) Whenever the solution achieved in steps 2 and 3 is not satisfactory - i.e., there is an excessive cost increase with respect to either the approximate or the exact lower bounds determined for the same unchecked scheduling - a different partitioning of the DFG is attempted, *based on the same set of MDS's initially used*: if an alternative identification of disjoint DS 's is available, not worse than the one previously adopted on the basis of the criteria named in 1), this alternative is chosen and resource sharing is repeated. Otherwise, the schedule is scanned to determine which (if any) control steps include less than N_c checking points. Let I_k be one such control step, including n_c checking points ($0 \leq n_c < N_c$). For every DS^* active in I_k , determine whether $I'(DS^*) < I_k < I'(DS^*)$; select among the ones satisfying this condition the DS maximising both distances (or at least maximising one distance and best balancing the other one) and split it so as to insert a checking point on one of the outgoing edges traversing I_k and to grant implicit checking for all other outgoing edges. This grants that the number of checkers will not be increased with respect to the previous solution. If necessary, repeat this operation until the number of checking points in I_k reaches N_c .
 - 5) The resource and register sharing phase is repeated on the modified cover obtained in step 4. (Obviously, step 4 can be repeated on *all* control steps for which $n_c(k) < N_c$).

In the next section, it will be seen how the allocation problems here discussed can be taken into account even before scheduling is performed, by guiding in the selection of the detectable cover itself.

4. Allocation-guided choice of a detectable cover.

From section 3, it follows that the initial choice of the MDS 's constituting the detectable cover upon which scheduling and allocation will be performed is a critical one; while subsequent partitioning into smaller DS 's has been proved possible, it is clearly preferable to adopt from the start the figures of merit related to subsequent allocation as a guide for identification of a "better" detectable cover. To this end, we operate - on the leveled, unchecked DFG - the two ASAP and ALAP schedulings, and immediately evaluate *mobility* associated with each operator. Then, identification of all MDS 's in the DFG is performed. At this point, we associate with *each MDS*:

- 1) the sequence of levels upon which the MDS spans;
- 2) the *mobility* of the MDS , evaluated as the minimum mobility of any of its nodes.

The usual covering table is then built and *essential MDS's* are immediately selected. Among all non-essential *MDS's*, the one spanning the largest number of levels is deleted (if two or more *MDS's* share this property, one is chosen at random); the operation is iterated until no other *MDS* can be deleted without leaving some node uncovered. At this point, a detectable cover is determined over the simplified covering table.

Having determined the minimum number of checking points, time-constrained scheduling is performed; in this phase, the *MDS* mobility is taken into account, suggesting - insofar as possible - that within any *MDS* no node be allowed to move on a number of levels larger than the mobility of the whole *MDS*; this will limit the lifetime of internal variables, and therefore the number of distinct registers that will have to be allocated to the *MDS* itself. From here on, the rules defined in sections 2 and 3 are applied (among other things, for identification of the disjoint *DS's* that are actually used for scheduling and allocation).

As an example, refer to the DFG in figure 4:

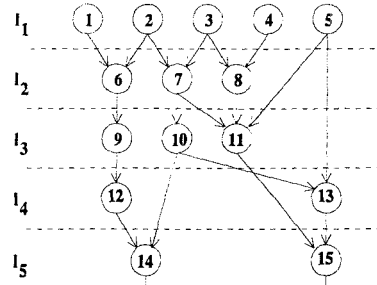


Figure 4: sample DFG for allocation-guided covering choice

Assume that all operations are identical, and that the following *MDS's* have been identified: **A**: {1,2,6,9,12,14}, **B**: {1,2,3,5,6,7,10}, **C**: {9,10,12,14}, **D**: {3,4,5,7,8,11}, **E**: {5,11,13,15}. As a consequence, checking point are respectively on the outputs of operations 14, 10, 11, 15. **A** spans *five* levels, while all other *MDS's* are seen to span a lower number of levels; the mobility of all *MDS's* is seen to be 0.

From the covering table, **A** is seen not to be essential; it is therefore deleted from the table itself. All remaining *MDS's* are now essential, so that they constitute the detectable cover for the DFG. Given their mutual overlapping, the following set of disjoint *DS's* is chosen: **B'**: {1,2,6,10}, **C'**: {9,12,14}, **D'**: {3,4,7,8,11}, **E'**: {5,13,15}. The optimum un-checked schedule requires *four* operators (and no less than four registers); by using the chosen *DS's* and the suggested guidelines for allocation-oriented scheduling, the checked solution requires a total of *five* operators, *two* checkers and *five* registers.

References

- [1] D.Gajski, N.Dutt, A.Wu and S.Lin: *High-Level Synthesis*, Kluwer Academic Publishers, Boston, MA, 1992
- [2] N.de Micheli: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, NY, 1994
- [3] N.K.Jha, S-J.Wang: "Design and Synthesis of Self-Checking VLSI circuits", *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 12, No.6, pp.879-887, June 1993
- [4] C.Bolchini, R.Montandono, F.Salice, D.Sciuto: "Self-checking FSM's based on a constant-distance state encoding", *proc. IEEE DFT 95*, Lafayette, USA, November 1995
- [5] E.Rao: *Error coding for arithmetic processors*, Academic Press, NY, 1974
- [6] V.Piuri: "Fault-tolerant systolic arrays: an approach based upon residue arithmetic", *Proc. IEEE ARITH-8*, Como, Italy, May 1987
- [7] V.Piuri: "Fault-tolerant array processors: an approach based upon AIN codes", *Proc. IEEE ISCAS88*, Helsinki, Finland, June 1988
- [8] M.Annaratone, R. Stefanelli: "A multiplier with multiple error correction capability", in *Proc. IEEE Arith 6*, 1983
- [9] A.Antola, V.Piuri, M.G.Sami: "A High-Level Synthesis Approach to Optimum Design of Self-Checking Circuits", *proc. EURODAC 96*, Geneva, September 1996
- [10] F.Kurdhal, A.Parker: "REAL: a program for register allocation", *Proc. DAC 87*, pp. 210-215, 1987