PHANTOM REDUNDANCY: A HIGH-LEVEL SYNTHESIS APPROACH FOR MANUFACTURABILITY

Balakrishnan Iyer, Ramesh Karri and Israel Koren*
Department of Electrical and Computer Engineering
University of Massachusetts at Amherst
Amherst, MA 01003

Abstract *– **Phantom redundancy**, an area-efficient technique for fabrication-time reconfigurability is presented. Phantom redundancy adds extra interconnect so as to render the resulting microarchitecture reconfigurable in the presence of any (single) functional unit failure. The proposed technique yields **partially good chips** in addition to **perfect chips**. A genetic algorithm is used to incorporate phantom redundancy constraints into microarchitecture synthesis. The algorithm minimizes the performance degradation due to any faulty functional unit of the resulting microarchitecture. The effectiveness of the technique is illustrated on benchmark examples.*

1 INTRODUCTION

A number of researchers have examined fabrication-time reconfiguration approaches to enhance the yield of ICs. Built-In-Self-Repair (BISR) is one approach to fabrication time reconfiguration. In BISR fault tolerance is realized by providing a set of spare modules in addition to the core operational modules[1]. BISR approaches have been applied mostly to regular architectures such as memory and processors[1]. BISR techniques in the context of non-regular architectures have been recently developed by[2]. However, since BISR approaches are expensive in terms of silicon area, they have been limited to bit-, byte-, or digit-sliced systems[3].

In this paper, we present ***phantom redundancy***, an alternate technique for fabrication-time repair and reconfiguration of ICs. In contrast to the previous techniques, phantom redundancy realizes fabrication-time reconfigurability by using ***redundant programmable interconnects instead of spare functional units***. Phantom redundancy, in addition to enhancing chip yield, entails minimal hardware overhead. The interconnection network can be reprogrammed in the event of a fault to reconfigure the fault-free functional units into an operational IC, albeit with a degraded performance.

We have incorporated phantom redundancy at the register transfer level within a top-down VLSI design methodology for the following reasons: (i) there is a tight interdependence between the synthesized microarchitecture without reconfigurability and the reconfigurability of such an architecture and (ii) the underlying functional fault model is at the register transfer (RT) level. We used a ***genetic algorithm*** [4] to incorporate the scheduling, allocation and reconfigurability constraints.

In a reconfigurable microarchitecture, the control unit is important since reconfiguration is affected by reprogramming the control signals[2]. We assume a microprogrammable controller where reconfigurability is achieved by loading a suitable microprogram based on the results of testing the fabricated chips. Fault tolerance of the microcontroller can be achieved by the use of standard memory fault tolerance techniques like adding spare rows and columns[3].

Related research in CAD for manufacturability is briefly outlined. Recently, Chiluvuri and Koren[5] have developed layout compaction and routing algorithms to maximize defect-tolerance. At the RT level, Guerra et. al. [2] have presented a technique for incorporating BISR to enhance the yield of VLSICs. They use redundant functional units to overcome permanent fabrication-time faults. In the area of high-level synthesis most researchers have addressed the problems of area and performance optimization[6]. More recently, other important objectives, such as power [7], testability [8], and fault tolerance [9] have also been addressed.

2 PHANTOM REDUNDANCY

Phantom redundancy yields ***gracefully degradable*** data paths with minimal hardware overhead by using additional interconnects instead of spare functional units to achieve fault-tolerance. Upon detecting a faulty functional unit at fabrication time, the interconnect is reprogrammed so as to perform the intended function on the fault-free functional units, although at a reduced throughput.

Towards illustrating and clarifying the concept of phantom redundancy, consider a CDFG consisting of six operations $(a, b, \ldots, f)$ shown in Fig. 1(a). Assuming that all operations are of the same type and no back-to-back chaining is allowed, the fastest schedule requiring two clock cycles and four functional units is shown in Fig. 1(a). A gracefully degrading microarchitecture incorporating phantom redundancy is shown in Fig. 1(b). The redundant interconnects, shown in dotted lines in the figure, render the basic architecture reconfigurable in the presence of any single functional unit failure. Upon identifying a faulty functional unit, the controller is programmed so as to operate the reconfigured microarchitecture with a degraded performance. For example, if

*Supported in part by NSF, under contract MIP-9305912.

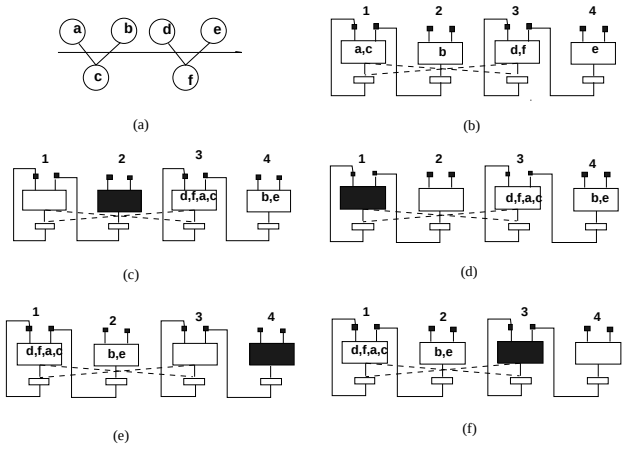

Figure 1: (a) Scheduled CDFG (b) Microarchitecture realizing the CDFG (c)–(f) Area optimal phantom redundant microarchitectures.

functional unit 2 is faulty (shown as a dark rectangle in Fig. 1(c)), operation b assigned to it can be reallocated to the remaining fault-free functional units. The reconfigured microarchitecture with a degraded performance of 4 clock cycles (as opposed to the original 2 clock cycles) is shown in Fig. 1(c). Operations $\{a, c\}$ have been assigned to functional unit 3 and operation b which was originally allocated to functional unit 2 has been assigned to functional unit 4. The reallocated and hence reconfigured microarchitectures in the event of failure of any one of the other three functional units are shown in Fig. 1(d), (e), (f), respectively.

2.1 HARDWARE AND FAULT MODELS

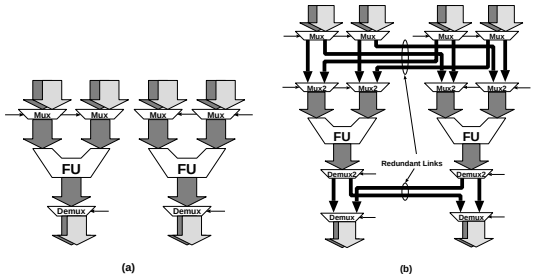

Figure 2: (a) Hardware Model without Reconfigurability (b) Hardware Model for Phantom Redundancy Showing the Redundant Links and Extra Multiplexers

The model for reconfigurable microarchitectures is shown in Fig. 2. The *redundant links* and the multiplexers labeled *mux2* and the demultiplexers labeled *demux2* constitute the reprogrammable interconnect. The functional units connected as in Fig. 2 form a *backup pair*. Within such a backup pair, if a functional unit is defective at fabrication time, the other functional unit can take over by programming the select signals of *mux2* and *demux2*. In general, to have redundancy among N functional units, at least $\lceil \frac{N}{2} \rceil$ backup pairs are necessary. The two level multiplexing and demultiplexing scheme shown in Fig. 2 may be flattened into a single level.

We also assume that at most one functional unit may be faulty. Our functional fault model is based on the observation that the critical area (i. e., the area susceptible to fabrication faults) of the functional units is much larger than that of the buses and the registers. Consequently, the probability of faults in functional units is much larger than that of the buses and register files.

3 SYNTHESIS FOR MANUFACTURABILITY

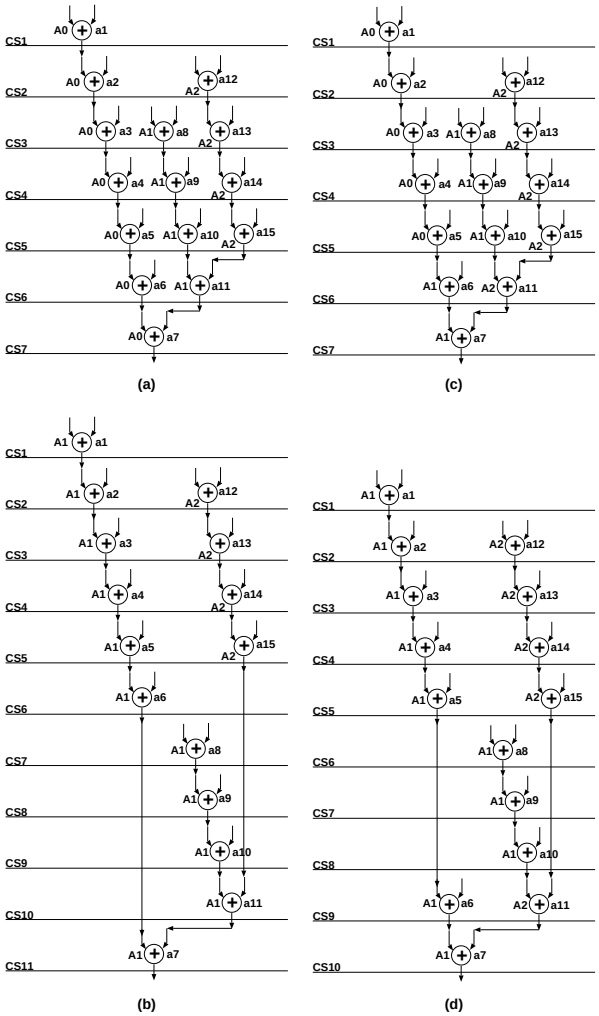

Figure 3: (a) A schedule and an interconnect optimal allocation (b) Sub-optimal reconfiguration in the presence of a faulty **A0** as a result of a two-step approach to incorporation of reconfigurability constraints (c) A global schedule, and allocation incorporating reconfigurability constraints (d) An efficient reconfiguration in the presence of a faulty **A0**

During microarchitecture synthesis, incorporation of reconfigurability following the scheduling and binding phases will yield architectures with a poorer performance. Consequently, scheduling, allocation and recon-

figurability constraints have to be considered in an integrated fashion. For example, consider a CDFG consisting of fifteen add operations (**a1, ..., a15**) as shown in Fig. 3(a).

Assuming three adders (**A0, A1, A2**), one possible schedule and an allocation that minimizes the interconnect overhead is shown in Fig. 3(a). While the horizontal lines across the CDFG identify the time step wherein a node is executed, the annotations to the left of each node determine the functional unit on which it is executed. In Fig. 3(a), node **a14** is scheduled in control step 4 and is executed on adder **A2**. Until now scheduling and allocation have not accounted for possible performance degradation in the presence of an adder failure. Reconfigurability can be achieved by identifying {**A0, A1**} and {**A1, A2**} to be the backup pairs. That is, if **A0** is faulty, adder **A1** takes over, if **A2** is faulty, **A1** takes over and if **A1** is faulty either **A0** or **A2** take over. However, this results in a worst-case performance degradation of four clock cycles. In Fig. 3(b), one such schedule (using 11 control steps) and allocation (wherein **A1** takes over from a failed **A0**) is shown. An alternate schedule and allocation resulting in a worst-case performance degradation of only three clock cycles is shown in Fig. 3(c,d).

The problem of microarchitecture synthesis with phantom redundancy constraints can be formulated as follows: Given a CDFG and a hardware model, synthesize a microarchitecture such that (1) the performance of the unimpaired system is not compromised. (2) the performance degradation in the event of a single FU failure is minimal. (3) the area overhead of reconfiguration is minimal.

To find such globally optimal solutions, we use genetic algorithms to perform scheduling, binding and partitioning of the functional units into backup pairs simultaneously. The cost function simultaneously explores the space and time dimensions. Each valid solution is encoded using four ***chromosomes (strings)***. The ***module selector string*** encodes the space dimension of the solution. The ***module allocator string*** encodes the functional unit on which an operation is carried out. The ***time stamp string*** encodes the time dimension of the scheduling and allocation problem. Finally, the ***backup pair string*** identifies the backup pairs in the reconfigured architecture. The reader is referred to [10] for a detailed discussion of the cost function, the problem specific encodings, the genetic operators and the selection scheme employed.

It is important to note that the two functional units forming a backup pair need not be identical in terms of performance although they should be capable of carrying out the same function. For example, a fast multiplier unit can have a slow multiplier as the backup unit. A high-level synthesis tool should explore these tradeoffs.

4 RESULTS

In this section we summarize the trade-offs conducted on a set of benchmark examples namely, the fifth order elliptic wave digital filter (Ex #1), an AR filter (Ex #2), a third order bilinear lossless discrete integrator filter (Ex #3), and the FIR filter (Ex #4).

Non-pipelined Functional Units: Initially, we used non-pipelined functional units with the multiplier taking 2 clock cycles and the adder taking one clock cycle. The system word length is assumed to be 24 bits and the filter coefficients are assumed to be represented in 16 bits.

The results of this experiment are summarized in Table 1. The second and the third columns indicate the number of multipliers and adders, used in the design. While the fourth column (titled *Orig*) shows the number of clock cycles required for executing the CDFG when the system is unimpaired, the fifth column gives the control steps required in the worst case to execute the CDFG (in the presence of any single functional unit failure). The next column summarizes the percentage degradation in performance. The chip area estimates for the original IC, the phantom redundant reconfigurable IC, and the chip with BISR redundancy are given in the columns titled *Chip Area Orig.*, *Chip Area Phant.* and *Chip Area BISR* respectively. All the reported chip area estimates were obtained using the HYPER [11] hardware database. These area estimates are are about 15% off the actual layout areas[12]. The BISR area was computed by assuming one redundant functional unit of each type and does not include any additional interconnect and multiplexing overheads. The percentage area overheads for the phantom redundancy and BISR techniques over that of the original chip area are indicated in the tenth and eleventh columns. The last column summarizes the percentage savings in area overhead of the proposed technique *vis-a-vis* the BISR technique.

Ex #	#		Clks		%	Chip Area			% Ovrhd		%
	Mult	Add	Orig	Deg	Degr	Orig	Phantom	BISR	Phantom	BISR	Impr
1	3	3	17	19	11.8	1.8	1.8	2.4	1.5	31.8	30.3
2	2	2	19	34	78.9	1.2	1.3	1.8	4.6	46.6	42.0
2	4	3	11	18	63.6	3.1	3.2	3.9	1.9	24.5	22.6
2	4	2	11	18	63.6	3.1	3.2	3.9	2.1	24.7	22.7
3	2	2	8	11	37.5	1.2	1.3	1.8	1.7	47.2	45.4
4	4	4	8	12	50.0	2.8	2.8	3.5	1.4	24.1	22.7
4	8	8	6	8	33.3	7.8	7.9	8.8	0.7	12.3	11.6

Table 1: Impact of phantom redundancy on designs synthesized using non-pipelined functional units.

From the table, it can be seen that the area savings due to phantom redundancy over the BISR technique is, on an average, 28.19%. While the area overhead of phantom redundancy is negligible when compared to that of original designs, the additional area required for BISR techniques corresponds to a significant proportion of the original chip area. This we believe is in general true for most designs.

In contrast to BISR, the performance degradation of these partially good chips is quite significant. In fact, the performance degradation (of over 78%) is highest for the AR filter built from 2 adders and 2 multipliers. A closer look into this synthesized design and the AR algorithm reveals that this is because of two fac-

tors. Firstly, the number of functional units of a given type are very small. Secondly, the multiplication and the addition operations in the AR filter are clustered, thereby dramatically increasing the critical paths in the algorithm.

Pipelined Functional Units: For this experiment we use a two stage pipelined multiplier with a latency of 2 clock cycles and an initiation rate of 1 clock cycle. The results are summarized in Table 2.

Ex #	# Mult	# Add	Clks Orig	Clks Degr	% Degr	Chip Area Orig	Chip Area Phantom	Chip Area BISR	% Ovrhd Phantom	% Ovrhd BISR	% Impr
1	2	3	17	19	11.7	1.3	1.3	1.9	2.3	45.5	43.2
2	2	2	13	19	46.2	1.2	1.3	1.8	4.2	46.6	42.4
2	4	3	11	13	18.2	3.2	3.2	3.9	1.5	24.5	23.0
2	4	2	11	16	45.5	3.1	3.2	3.9	1.4	24.7	23.3
3	2	2	8	9	12.5	1.2	1.2	1.8	1.3	47.2	45.9
4	4	4	7	9	28.6	2.8	2.8	3.5	1.4	24.2	22.8
4	8	8	6	7	16.7	7.8	7.9	8.8	0.6	12.3	11.7

Table 2: Impact of pipelined functional units on phantom redundancy

The best results were obtained for Ex #1, where the performance degradation is 11.76% while the area overhead incurred is only 2.29%. On the other hand, the BISR strategy leads to no performance degradation but the area overheads amount to 45.53%. The worst performance degradation occurs for the Ex #2 example with 2 multipliers and 2 adders. The large performance degradation is largely due to the availability of only one adder(multiplier) unit in the event of a failure in one of the 2 adder(multiplier) units present. The savings in area over the BISR technique is on the average 30.34%. The average degradation in performance is 25.61%. Notice the marked reduction in performance degradation when using pipelined functional units.

The performance penalties in Table 1 are quite high as compared to Table 2. This is because the use of pipelined units permits the initiation of operations at a much higher rate when one of the functional units fail. Thus operations assigned to a failed unit are scheduled in earlier steps as opposed to waiting for a multicycle operation to be completed before the next operation can be initiated. Pipelining is particularly effective when operations of the same type are clustered in the CDFG. The reduction in performance degradation will be even more significant with more deeply pipelined functional units.

Multifunctional ALUs: The results for the examples using ALU type functional units are summarized in Table 3. The average saving in area over the BISR technique is 21.22% while the average degradation in performance is 38%. The degradation values are higher in this table as compared to Tables 1 and 2 because the number of functional units used in the examples is much smaller.

Ex #	# ALUs	Clks Orig	Clks Degr	% Degr	Chip Area Orig	Chip Area Phantom	Chip Area BISR	% Ovrhd Phantom	% Ovrhd BISR	% Impr
2	3	12	15	25.0	1.7	1.8	2.3	1.9	33.1	31.1
2	4	9	14	55.6	3.1	3.1	3.8	1.9	25.2	23.3
2	6	8	10	25.0	4.5	4.6	5.3	1.3	17.0	15.7
4	4	9	13	44.4	2.7	2.7	3.4	1.3	25.0	23.7
4	8	5	7	40.0	7.5	7.6	8.5	0.6	12.8	12.2

Table 3: Results on Benchmark Examples using multifunctional ALUs

5 DISCUSSION

The effectiveness of phantom redundancy has been verified on benchmark examples. The results indicate large savings in area (of over 30%) over the BISR approach. The overheads due to phantom redundancy are negligible as compared to the original designs ($\leq 5\%$). Further, the performance degradation in pipelined architectures is significantly lower than that of non-pipelined data paths. Phantom redundancy is eminently suited for mature processes where the yields are sufficiently high to make the overheads of BISR too exorbitant. Since the BISR technique does not lead to performance degradation it may be preferable in designs with tight performance constraints. There is also an increase in the number of registers required to store the state information. The use of pipelined data path elements leads to smaller performance degradation as opposed to multicycle non-pipelined functional units.

References

[1] B.W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publishing Company, Inc., 1989.

[2] L. Guerra, M. Potkonjak, and J. Rabaey, "High Level Synthesis Techniques for Efficient Built-In-Self-Repair", *in Intl Workshop on DFT in VLSI Systems*, pp. 41–48. IEEE, 1993.

[3] W.R. Moore, "A review of fault-tolerant techniques for the enhancement of integrated circuit yield", *in Proc. IEEE*, vol. 74, pp. 684–698, May 1986.

[4] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, 1989.

[5] V. Chiluvuri and I. Koren, "Layout synthesis techniques for yield enhancement", *in IEEE Trans. on Semiconductor Manufacturing*, pp. 178–187, May 1995.

[6] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems", *Proc. IEEE*, vol. 78, pp. 301–318, 1990.

[7] A. Chandrakasan *et. al.*, "HYPER-LP: A System for Power Minimization Using Architectural Transformations", *in ICCAD*, pp. 300–303, 1992.

[8] T. Lee, W.H. Wolf, and N.J. Jha, "Behavioral Synthesis for Easy Testability in Data Path Scheduling", *in ICCAD*, pp. 616–619, 1992.

[9] A. Orailoglu and R. Karri, "Coactive Scheduling and Checkpoint Determination During High Level Synthesis of Self-Recovering Microarchitectures", *in IEEE Trans. VLSI*, pp. 429–433, Sep. 1994.

[10] B. Iyer, R. Karri, and I. Koren, "Phantom Redundancy: A High-Level Synthesis Technique For Manufacturability", *in TR-95-CSE-3, Dept of ECE, U. of Mass., Amherst 01003*, 1995.

[11] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast Prototyping of Data Path Intensve Architectures", *in IEEE Design & Test*, pp. 40–51, 1991.

[12] M.M. Potkonjak, *Personal Communication.*