High Level Synthesis Techniques for Efficient Built-In-Self-Repair

Lisa M. Guerra†, *Miodrag M. Potkonjak*‡, *Jan M. Rabaey*†
†Dept. of EECS, University of California, Berkeley, CA 94720
‡C&C Research Laboratories, NEC USA, Princeton, NJ 08540

Abstract

Built-In-Self-Repair (BISR) is a hardware redundancy fault tolerance technique, where a set of spare modules is provided in addition to core operational modules. Until now, the application of BISR methodology has been limited to situations where a failed module of one type can only be replaced by a backup module of the same type. This paper shows that in ASIC designs it is possible to enable replacement of modules of different types with the same spare units by exploiting the flexibility of high level synthesis solutions. Resource allocation, assignment and scheduling techniques that support a new BISR methodology are presented. All mentioned high level synthesis algorithms are developed on top of the HYPER high level synthesis system, using a novel statistical methodology for heuristic algorithm development and improvement. The effectiveness of the approach is verified and yield improvement data is presented for numerous real-life examples.

1.0 Introduction

BISR is a hybrid redundancy technique where a set of backup parts are provided in addition to N operating modules. As the complexity of chip designs increases, fault tolerance techniques such as BISR play an increasingly important role in reliability and yield improvement efforts. Furthermore, because of the unabating rise of semiconductor manufacturing costs, optimization of process yields becomes even more important, making process improvement techniques such as BISR crucial.

BISR sparing methodology is a conceptually simple, yet powerful technique for increasing yield by adding redundant modules to the design. If a chip is found to have defective modules, these modules can be replaced by good ones before packaging. Similarly, these same BISR methods can also be applied to improve chip reliability. Chips can be made more fault tolerant to failures that occur during operation, by automatic replacement of failed modules with spare ones, so that the overall system can continue to function correctly. This is especially important for military systems or space exploration missions where it is critical that there are no system failures, since manual replacement of failed modules is either impossible or prohibitively expensive.

The main target for BISR techniques are systems that are bit-, byte-, or digit- sliced. This application area includes memories [Moo86], which are made from a set of bit planes, and arithmetic-logic units (ALUs), assembled from ALU byte slices [Sie92]. The bit-sliced BISR in

0-8186-3502-9/93 $3.00

memories significantly increases memory production profitability. A simple, yet powerful methodology for ALU byte slices implementation was proposed by Levitt et al. [Lev68]. The shadow box fault tolerant method is another BISR technique which has recently been extended to secondary memory storage [Pat88].

Massive parallelism is another area where BISR is starting to play a crucial role. And this area will become increasingly prominent, as concurrent computations becomes more popular. For example, Griffin et al. [Gri91] recently designed an 11 million transistor neural network execution engine, which has a triple-level redundancy structure resulting in the consumption of an additional 2.8 million transistors for BISR.

In wafer scale integration, BISR also plays a prominent role. In a highly integrated ULSI system which contains both DRAM and SRAM as well as uncommitted gate array [Sat92], statistical studies showed that the BISR technique called Interchip Relief significantly improved the yield.

Finally in systolic arrays designs, the role of BISR techniques has also been analyzed, though mostly from a theoretical and statistical point of view [Lei85, Neg89]

Although BISR techniques are widely used, they have not received appropriate attention in ASIC design. This paper introduces a novel concept of BISR for ASIC designs, which is directly built on the flexibility provided by high level synthesis during design space exploration. A variety of design goals have been addressed in high level synthesis [McF90] with emphasis, however, mainly on area and speed optimization. More recently, other important goals, such as power and testability have also been addressed. High level synthesis research for fault tolerant computing and design remains limited, and has primarily concentrated on designs where self-recovery from transient faults using micro roll-back and checkpoint insertion is explored [Kar92].

2.0 Hardware Model and Problem Statement

The hardware model being considered is shown in Figure 1 [Rab91a]. To stress the importance of interconnect minimization early in the design process, this model clusters all registers in register files, and connects them only to the inputs of the corresponding execution units. We also

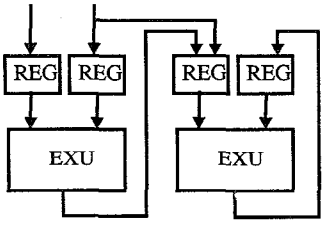

Fig. 1. Hardware Model: Interconnect-Regfile-Exu

assume that there is no bus merging, so that there exists a dedicated interconnect connecting any two units between which there are data transfers. Faults can occur in either an execution unit, a register file, or a bus. Under this hardware model and these assumptions, all faults can be classified as execution unit faults. A faulty register file prevents its corresponding execution unit from

receiving data, and thus has the same affect as a fault in the execution unit. Similarly, a faulty interconnect can be treated as a failure in the execution unit at the receiving end. Although we will concentrate on this model and show that BISR design using this hardware model results in low area overhead, the high level synthesis BISR techniques presented here are general and can be adapted to other hardware models.

The BISR synthesis problem statement can now be defined within this framework as follows:

Given a computation, an underlying hardware model, and an execution time bound t_{avail}, synthesize a minimum area semi-custom hardware implementation, so that up to K hardware units can be faulty.

If these techniques are used for fault tolerance against permanent faults, it is assumed that an error checking mechanism exists, and if they are used for yield enhancement, it is assumed that manufacturing testing will detect the faulty units. In either case, the controller is reconfigured upon detection of a fault. The controller is assumed to be reprogrammable or to lie on a chip separate from the datapath. [Che92, Yeu92] are typical examples of this type of system.

3.0 High Level Synthesis for BISR: Introduction and Motivation

One of the most straightforward approaches to BISR would be to provide a spare for each hardware instance, resulting in full duplication of the hardware. In this case, the number of additional units needed would be N, where N is the number of units required for the non-BISR implementation. However, the BISR overhead need not be so high. If the number of faulty units, K, is 1, for example, the high level synthesis assignment step provides us with the flexibility under which only 1 spare for each hardware class is necessary, as opposed to one spare per hardware instance. The operations from the failed unit will be transferred to the spare of the same type. The number of additional units needed in this case is M, where M is the number of hardware classes, $M \leq N$.

The flexibility gained through assignment clearly reduces the amount of hardware redundancy needed. Considering now the additional flexibility brought by scheduling, we can often use even fewer spares. This is possible since assignment and scheduling enable the 'replacement' of a module by a spare of a different type. When a failed unit is detected, instead of reassigning only those operations of the failed unit, we completely reassign and reschedule all the operations of the computation.

The following example (Fig. 2) illustrates the motivation and main ideas about how BISR overhead can be greatly reduced by addressing the use of alternate schedules and assignments to alleviate the need for a given failed unit. This example, for K=1, shows the imaginary part of a complex number multiplication with a constant and with a variable. Multiplication with a constant is such that it can be done using a single shift. The assumed available time is 3 control cycles, and each operation takes one control cycle. The minimum hardware required for this computation consist of 2 shifters, 1 multiplier and 1 adder. If scheduling flexibility is not exploited, the minimum BISR hardware will be 3 shifters, 2 multipliers and 2 adders. However, if we allocate only 2 shifters, 2 multipliers and 2 adders, we still can achieve a complete BISR implementation by altering the schedule. This can be verified by the schedules for all three combinations of failed units given in Table 1. An important point to note is that NO additional

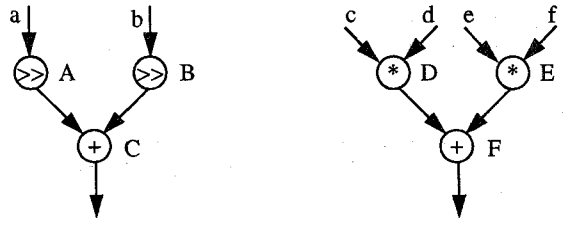

Fig. 2. Scheduling and Assignment for BISR: motivational example

shifters were needed. In the event of a shifter failure, the scheduling flexibility brought by the redundant multiplier is exploited to absorb the need for another shifter.

Failed Unit	Adder			Shifter			Multiplier		
Control Step	>>	*	+	>>	*	+	>>	*	+
1	A,B	D, E		A	D, E		A, B	D	
2			C	B		F		E	C
3			F			C			F

Table 1: Schedules for Example of Fig. 2

4.0 Allocation, Assignment, Scheduling for BISR Designs

In this section, we briefly explain the new high level synthesis algorithms for BISR. A description of standard high level synthesis notation and methodology can be found in [McF90, Cam91]. Allocation, scheduling and assignment are highly interrelated tasks which are each NP-complete. These tasks have been solved using a wide range of algorithmic strategies. The global strategy of the HYPER high level synthesis system [Pot89] is well suited for use as the starting point for the development of new high level synthesis algorithms targeting BISR. In this system, allocation first proposes a hardware solution, then assignment and scheduling are performed to check the feasibility of the solution. To take into account the peculiarities dictated by BISR requirements, it was neccessary to develop a new allocation scheme. Note that for any proposed allocation solution in BISR synthesis, it is necessary to assure that assignment and scheduling can succeed for **all** combinations of K failed units. This motivated the development of heuristics for ordering the attempted schedules and for look-ahead pruning of allocations. The pseudo-code for the global flow of the algorithm is presented below.

A sharp minimum bound, M_j, on the necessary amount of hardware of each class j is used as the initial allocation. M_j is defined as: $M_j = m_j + K$, where m_j is a minimum bound on the amount of hardware j necessary for any non fault-tolerant implementation and K is the number of faults. For each hardware class j, relaxed based scheduling techniques [Rab91b] are used to derive an estimate of m_j. The equation for M_j can be understood by observing that *any* imple-

```
GetInitialAllocation();
While TRUE {
   SortInDecreasingOrderOfStress(Ordered_HW);
   foreach j  ∈  Ordered_HW {
      Success = Assign and ScheduleWithFailedUnit(j);
      if (!Success)
        break;
   }
   UpdateStress();
   if (!Success)
       GetNewHWUnit();
   else
       RedundancyRemovalWithLookAheadPruning();
}
```

mentation requires at least m_j units, and since up to K units of type j can be bad, at least (m_j + K) units are needed.

If the initial allocation fails, the allocation expansion phase is entered, where new hardware units are added one by one until the allocation succeeds. Units are added in accordance with their area cost and likelihood that the new solution will be feasible. At the completion of the expansion phase, there is no guarantee that the feasible allocation A, is minimal. It is possible that a subset of the allocation, $A' \subset A$ is also a solution. To assure that a local minimum has been reached, it is necessary to assure that if any units are removed from the current solution, a success cannot be achieved. In general, the units with high cost and small demand, as indicated by the scheduling and assignment program, are the first to be tried for removal. Here however, to increase the efficiency of the algorithm, a look-ahead scheme is used in which data from previously failed allocations is used to prune new allocations before they are tried.

For a successful allocation, all schedules where any combination of K resources is faulty must succeed. We thus order the schedules in decreasing order of difficulty, so that we can exit as fast as possible in the event that there is an insufficient BISR resource allocation. The ordering is based on a heuristic global stress measure, where schedules for the failure of the most highly stressed units are tried first. The global stress function has three intuitively appealing, experimentally observed and statistically validated parts: (i) Resource Utilization Factor, (ii) ε-Critical Network Stress, (iii) and Scheduling Stress.

The stress functions are highest for resources which are in the highest demand. Resource Utilization Stress captures the idea that scheduling under the premise that a particular resource type will have resource utilization close to 100% is rarely feasible. The ε-Critical Network Stress expresses the observation that resources for operations that lie on the critical path of computation or on paths which are almost as critical are particularly important. The ε-critical network consists of all paths which have lengths within a small ε percentage of the critical path length. The Scheduling Stress assembles statistical data about the observed difficulty for scheduling the various types of operations during previously attempted schedules. The global stress is a nonlinear combination of these three factors, with statistically derived weighting parameters. A detailed description of all algorithms and functions is presented in [Gue93].

5.0 Experimental Results and Yield Enhancement Analysis

The BISR techniques were validated on the set of examples shown in Table 2. The table shows all relevant data for the standard and the BISR synthesis procedures. Note that although the different forms of the 8th order Avenhaus filters provide the same functionality, they have drastically different structures and sizes. The average and median BISR design area overhead over all examples was 19.1% and 16.6%. Note also that although the initial implementations of all examples had 4 different types of hardware units, an average of only 2.58 additional units were needed for the BISR designs.

Example	IU	F U	NT	Non-BISR Area	BISR Area	Area Overhead (%)
Jaumann	5	8	4	4.39	7.07	61.0
5th WDF	6	9	4	1.43	1.73	21.0
8IIR DFa	7	10	4	8.06	10.86	34.7
8IIR GMa	8	9	4	4.84	4.95	2.3
7IIRa	9	11	4	18.18	23.76	30.7
8IIR GMb	9	12	4	6.66	6.88	3.3
8IIR P	9	12	4	2.23	2.55	14.4
8IIR C	9	12	4	4.24	4.69	10.6
5IIR	11	14	4	4.55	5.56	22.2
7IIRb	17	19	4	4.47	4.92	3.1
8IIR DFb	23	26	4	19.81	21.20	7.0
Wavelet	30	32	4	22.05	26.19	18.8

Table 2: Results on 12 benchmark examples: IU - # of EXU units in non-BISR implementation; FU - # of EXU units in BISR implementation; NT - # of hardware classes; Jaumann - Jaumann LDI filter; 5th WDF - 5th order elliptical wave digital filter; 8IIR DFa & 8IIR DFb - 8th order bandpass IIR direct form filter for two different set of timing constraints; 7IIRa & 7IIRb - 7th order low pass IIR filter for two sets of timing constraints; 8IIR GMa & 8IIR GMb - 8th order bandpass IIR filter Gray Markel form for two sets of timing constraints; 8IIR P & 8IIR C - 8th order IIR filter parallel and cascade form; and Wavelet - Wavelet Transform

In Table 3, consequences on yield and productivity are presented. Yield, which is defined as the percentage of functional dies on a wafer, is an important parameter, but it is not the only indicator of wafer productivity. The redundant circuitry will in general increase the design area and thus reduce the number of chips which can be placed on a wafer. The productivity of BISR

methodology can be obtained by dividing the relative change in yield by the relative change in area. To calculate the yield increase we used the following formula [Sta92]:

$$\bar{Y}_{mn} = \binom{n}{m}\bar{Y}_1^m\left(\prod_{i=0}^{m-1}\frac{\mu+i}{\mu+i\bar{Y}_1}\right)\times(1-\bar{Y}_1)^{n-m}\left(\prod_{j=0}^{n-m-1}\frac{\mu+j\bar{Y}_1/(1-\bar{Y}_1)}{\mu+m\bar{Y}_1+j\bar{Y}_1}\right)$$

Stapper's formula calculates the probability that exactly m out of n modules operate correctly for a given value of the variability parameter μ and single module yield, $\overline{Y1}$. A slight modification is made to the formula to take into account units of largely different areas.

Although this procedure primarily targets BISR memory design, it has been demonstrated that the models can be successfully used for datapath analysis [Kor84]. The parameter μ gives an indication of the assumed probability of clustered defects, which are the most common sources of chip malfunctions. Large values of μ correspond to smaller levels of clustering, and therefore lower processing variability [Sta92]. On all examples (except the smallest one in terms of execution units), and for all values of μ, there is a clear improvement in relative productivity.

Example	Yield					Productivity				
	μ=0.5	μ=1.	μ=2.	μ=5.	μ=Inf	μ=0.5	μ=1.	μ=2.	μ=5.	μ=Inf
Jaumann	13.29	13.78	14.16	14.43	14.27	0.825	0.856	0.880	0.896	0.886
5th WDF	13.76	14.47	15.13	15.82	16.48	1.137	1.196	1.250	1.307	1.362
8IIR DFa	14.06	14.91	15.76	16.79	18.25	1.043	1.107	1.179	1.246	1.355
8IIRGMa	16.62	18.42	20.50	23.52	30.02	1.624	1.800	2.004	2.299	2.934
7IIRa	15.34	16.67	18.19	20.37	25.24	1.174	1.275	1.392	1.559	1.931
8IIRGMb	14.40	15.39	16.47	17.95	20.90	1.394	1.490	1.594	1.738	2.023
8IIR P	14.40	15.39	16.47	17.95	20.90	1.259	1.345	1.440	1.569	1.827
8IIR C	14.40	15.39	16.47	17.95	20.90	1.302	1.391	1.489	1.622	1.889
5IIR	14.54	15.61	16.80	18.56	22.74	1.190	1.277	1.375	1.519	1.861
7IIRb	15.07	16.32	17.82	20.24	28.65	1.461	1.583	1.729	1.953	2.779
8IIR DFb	14.54	15.62	16.89	18.98	27.69	1.359	1.460	1.579	1.774	2.588
Wavelet	14.67	15.76	17.08	19.30	30.50	1.235	1.327	1.438	1.625	2.179

Table 3: Yield (in %) and Relative Productivity change for examples from Table 2 for various values of the variability parameter μ. The initial yield is 10%.

The results and discussion presented here are for the number of faulty units, K, equal to 1. Handling larger values of K does not introduce any new conceptual ideas, and modifications are straightforward. Larger values will place an exponential strain on the number of different schedules to be generated. It has been shown, however, that in general as K increases, the improvement in the relative productivity can give diminishing returns, and can even produce lower

productivity [Sta92]. An interesting issue for BISR design is the selection of the value of K which gives the optimal effective yield, as a trade-off between resilience to failure and hardware overhead.

6.0 Conclusions

BISR is a powerful and widely used fault tolerance technique. It will become more important especially with the increase of massive parallelism. However, until now the scope of BISR has been restricted to the substitution of operation modules with only those of the same type. We have presented high level synthesis techniques which support a new BISR methodology for ASIC designs. This new BISR methodology exploits the flexibility of the design solution space so that resources of several different types can be backed up with the same unit. Experimental results and yield and productivity calculations indicate the strong potential of this new technique for yield and reliability improvement in future highly integrated ASICs.

7.0 References

[Cam91] R.Camposano, R.A.Walker, *A Survey of high-level Synthesis Systems*, Kluwer Academic, Boston, MA,1991.

[Che92] D. Chen, et al., "An Integrated System for Rapid Prototyping of High Performance Algorithm Specific Data Paths," *Proceedings Application Specific Array Processors*, pp. 134-148, 1992.

[Gri91] M. Griffin, et al., "An 11-Million Transistor Neural Network Execution Engine," *ISSCC-91*, pp. 180-181, San Francisco, CA, 1991.

[Gue93] L. Guerra, M. Potkonjak, J. Rabaey, "High Level Synthesis for Reconfigurable Datapath Structures", NEC Technical Report NEC-93-5510-06, 1993.

[Kar92] R. Karri and A. Orailoglu, "Transformation-Based High-Level Synthesis of Fault-Tolerant ASICs," *29th ACM/IEEE Design Automation Conference*, pp. 662-665, 1992.

[Kor84] I. Koren, M.A. Breuer, "On Area and Yield Consideration for Fault-Tolerant VLSI Processor Arrays", *IEEE Trans. on Computing*, Vol. 33, No. 1, pp. 21-27, 1984.

[Lei85] T. Leighton, C.E. Leiserson, "Wafer-scale integration of systolic arrays," *IEEE Trans. on Computers*, Vol. 34, No. 5, pp. 448-461, 1985.

[Lev68] K.N. Levitt, M.W. Green, J. Goldberg, "A Study of the Data Communication Problems in a Self-Repairable Multiprocessors," *Conf. Proc. of AFIPS*, Vol. 32, pp. 515-527, Thompson Book, Washington, DC, 1968.

[McF90] M.C. McFarland, A.C. Parker, R. Camposano, "The High-Level Synthesis of Digital Systems," *Proceedings of the IEEE*, Vol. 78, No. 2, pp. 301-317, 1990.

[Moo86] W.R. Moore, "A review of fault-tolerant techniques for the enhancement of integrated circuit yield," *Proceedings of the IEEE*, Vol. 74, No. 5, pp. 684-698, 1986.

[Neg89] R. Negrini, M.G. Sami, R. Stefanelli, *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*, MIT Press, Cambridge, MA, 1989.

[Pat88] D. A. Patterson, et al., "A case for redundant arrays of inexpensive disks (RAID)," *Proceedings SIGMOD*, pp. 109-116, 1988.

[Pot89] M. Potkonjak, J. Rabaey, "A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Flow Graphs," *26th ACM/IEEE Design Automation Conference*, pp. 7-12, 1989.

[Rab91a] J. Rabaey et al., "Fast Prototyping of Data Path Intensive Architectures," *IEEE Design & Test Magazine*, June 1991.

[Rab91b] J. Rabaey, M. Potkonjak, "Complexity Estimation for Real Time Application Specific Circuits," *Proc. IEEE ESSCIRC Conference*, Milan, Sept. 1991.

[Sat92] K. Sato et al., "A System-Integrated ULSI Chip Containing Eleven 4 Mb RAMs, Six 64kb SRAMs and an 18k Gate Array," *ISSCC-92*, pp. 52-53, San Francisco, CA, 1992.

[Sta92] C.H. Stapper, "A New Statistical Approach for Fault-Tolerant VLSI Systems", *22nd Annual FTCS*, pp. 356-365, 1992.

[Sie92] D.P. Siewiorek, R.S. Swartz, *Reliable Computer Systems: Design and Evaluation*, 2nd edition, Digital Press, Burlington, MA.

[Yeu92] A. Yeung and J. Rabaey, "A Data-Driven Architecture for Rapid Prototyping of High Throughput DSP Algorithms," *IEEE VLSI Signal Processing Workshop*, pp. 225-234, 1992.