Heterogeneous BISR Techniques for Yield and Reliability Enhancement using High Level Synthesis Transformations

Miodrag M. Potkonjak
C&C Research Laboratories, NEC USA, Princeton, NJ

Lisa M. Guerra, Jan M. Rabaey
Dept. of EECS, University of California, Berkeley, CA

Abstract

Built-In-Self-Repair (BISR) is a fault tolerance technique against permanent faults, where in addition to core operational modules, a set of spare modules is provided. If a faulty core module is detected, it is replaced with a spare module. The BISR methodology has been used only in situations where a failed module of one type can only be replaced by a backup module of the same type. We propose a new BISR approach for ASIC design which removes this constraint and enables replacement of modules of different types with the same spare units by exploiting the design space exploration abilities provided by the use of transformations in high level synthesis. Fast and efficient high level synthesis algorithms which take into account peculiarities of transformation-based design for BISR are presented. The potential of the approach is demonstrated on a set of benchmark examples by showing significant yield and relative productivity improvements which are calculated using state-of-the-art yield modeling techniques.

1.0 Introduction

As the cost of semiconductor manufacturing increases, it becomes imperative to improve process yields as fast as possible. Process improvement techniques such as BISR therefore become very important.

BISR is a hybrid redundancy technique where in addition to N core operational modules, a set of spare modules is provided. If a chip is found to have defective modules, these modules can be replaced by good ones after wafer testing. Similarly, 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 in military systems and space exploration missions where it is critical that there are no system failures, even in the face of errors, or where manual replacement of failed modules is either impossible or prohibitively expensive.

BISR techniques are regularly used during the development and operation of primary and secondary memories [Moo86, Sie92, Pat88] and sometimes in general purpose bit-sliced execution units [Sie92]. They have not received appropriate attention in ASIC design, but the ever increas-

0-8186-3492-8/93 $3.00 © 1993 IEEE

ing level of integration should soon make them an important methodology for ASIC yield improvement.

This paper introduces a novel concept of BISR for ASIC designs, which can be used for yield improvement or fault-tolerance against permanent faults. The concept is directly built on the flexibility provided by transformations during design space exploration. The rest of the paper is organized in the following way. After a survey of the previous BISR efforts in several areas and a precise problem formulation, very simple, yet real-life examples are used to introduce the main ideas. Next, a new high level synthesis transformation algorithm which minimizes hardware overhead for BISR is discussed. The paper concludes by presenting experimental results on a variety of real-life DSP examples and a brief outline of directions for future research.

2.0 Previous Related Work

High level synthesis provides the flexibility of design space exploration so that a variety of design goals can be addressed [McF90, Cam91]. The same references survey a variety of high level synthesis algorithms and provide an extensive high level synthesis bibliography. Most of these works, however, have targeted the optimization of area and speed (throughput). More recently, other important goals, such as power [Cha92], testability [Lee92], and fault tolerance [Rag91, Kar92] have also been addressed. Relatively little work has been done on high level synthesis techniques for fault tolerant design. Raghavendra and Lursinsap [Rag91] concentrated on designs with self - recovery from transient faults using micro roll-back and checkpoint insertion. Karri and Orailoglu [Kar92] presented a transformation-based method for minimizing hardware overhead while achieving a certain level of fault tolerance for common mode failures. While all previous high level synthesis methods for enhancing fault tolerance have addressed intermittent and transient faults [Sie92], this work concentrates on permanent faults, where fault tolerance is used for yield enhancement.

The main target for BISR techniques are systems that are bit-, byte-, or digit- sliced. This area includes SRAM and DRAM memories [Moo86], which are made from a set of bit planes and arithmetic-logic units (ALUs), assembled from ALU byte slices [Sie92]. By far the most important use of bit-sliced BISR is in SRAM and DRAM circuits. This is regularly used in almost all current day memory designs. The bit-sliced BISR in memories significantly increases memory production profitability. A simple, yet powerful methodology for implementation of ALU byte slices was proposed by Levitt et al. [Lev68]. Another important technique for preserving data through a failure occurrence in primary storage systems was proposed by Arulpragasm and Swartz [Aru80]. The concept is based on the use of a shadow box, a spare memory box which is identical to the other M operating memory boxes. A word stored at address j is the XOR of the words stored at location j in the other M operating boxes and has to be updated after each write to the memory system. In this reliability scheme, the content of a lost box can be reconstructed from the operating boxes and the shadow box by XORing values at corresponding locations. The shadow box technique has been recently extended to secondary memory storage [Pat88]. It is conceptually similar to the technique described in [Aru80], but makes updates on either the word or page basis. Several other algorithms describing methods to recover lost information content have been proposed [Sto90] and the performance and reliability of the methodology have been analyzed [Bit88, Gib89].

Massive parallelism is another area where BISR is starting to play a crucial role, which will become increasingly prominent with greater use of concurrent computations. 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 the 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 improves the yield. Several authors have also discussed and analyzed the role of BISR techniques in systolic arrays designs, though mostly from a theoretical and statistical point of view [Lei85, Neg89].

It is also interesting to note that BISR methodology is not limited to memory and execution units. Lewis [Lew79], for example, proposed the use of a backup fault tolerant clock.

3.0 Hardware Model and Problem Statement

We will first introduce a number of assumptions. The algorithm to be implemented is represented as a hierarchical Control-Data Flow Graph G (N, E, C), (or CDFG), with nodes N representing the flow graph operations, and the edges E and C respectively the data and control dependencies between the operations. The control dependencies are used to express relations between operations, which are not imposed by the data precedence relations [Rab91].

The hardware model being considered is shown in Figure 1 [Rab91]. To stress the importance of interconnect minimization early in the design process, this model clusters all registers in register files, connected only to the inputs of the corresponding execution units. We also assume

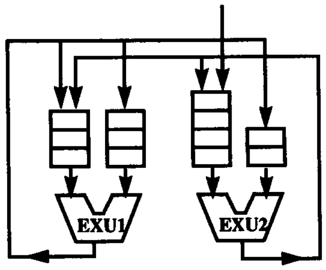

FIG. 1. Hardware Model: Interconnect-Regfile-EXU

that there is no bus merging, so there exists a dedicated bus 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 the just mentioned 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 bus can be treated as a failure in the execution unit at its receiving end. All other high level synthesis hardware models, can be addressed using the methodology presented here, of course, with proper modification of the algorithms.

The high level synthesis BISR process can be defined within this framework:

Given a hierarchical flow graph G(N,E,C), an underlying hardware model H and an execution time bound t_{avail}, synthesize a minimum area design, 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 either reprogrammable or to lie on a separate chip than the datapath. [Che92] and [Yeu92] are typical examples of this type of system.

4.0 Transformations for BISR

Probably the most straightforward approach to BISR is 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 W, where W is the number of units required for the non-BISR implementation. With the detection of a faulty unit, reconfiguration takes place to initiate use of its spare. This reconfiguration is conceptually a switch that passes control from the failed to the backup unit, or a reassignment of operations.

Fortunately, 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 it is clear that 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, and $M \leq W$.

The flexibility gained through assignment clearly reduces the amount of hardware redundancy needed. However, by considering the additional flexibility brought by use of transformations, we can often use significantly fewer spares. This is possible since transformations 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 transform, reassign and reschedule all operations of the CDFG. The specific goal addressed can now be restated as follows: *find the minimum area solution, for which the CDFG can be transformed, reassigned and scheduled in t_{avail}, even when as many as K units are faulty.*

4.1 Transformations in High Level Synthesis

Transformations are alterations in the computational structure such that the behavior (the relationship between output and input data) is maintained. They are used extensively in several computer science and CAD areas, most often in compilers [Fis88] and high level synthesis [Pot92]. Transformations have been successfully applied for the optimization of a variety of high level synthesis goals: area, speed [Pot92], power [Cha92], and run-time fault tolerance [Kar92]. This section shows how transformations, using specifically tailored optimization techniques, can significantly reduce the area of implementation for designs with BISR requirements.

4.2 Key Ideas: Motivational Examples

The basic idea behind the application of transformations in high level synthesis based BISR methodology is to transform the computation in several different ways according to the needs imposed by the available hardware, for each possible scenario of failed units. The simple example in Figure 2 will be used to illustrate this idea. In all the examples in this section, assume that each operation takes one control cycle. HYPER simulation tools can be used to verify that important numerical properties (e.g. numerical stability and overflow control) are maintained in all transformed designs. The assumed available time for the first example is 2 control cycles. The following identity is used to transform 2a into 2b:

$$(a + b) - (c + d) \quad = \quad (a - c) + (b - d)$$

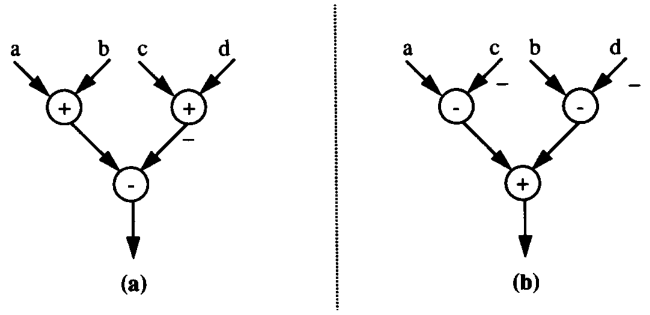

FIG. 2. Transformations for BISR: Motivational Example

It is easy to verify that both implementations calculate the same output for the same set of inputs. If we consider only implementation 2a, and assume that any unit can fail, then 3 adders and 2 subtractors are needed, since 2 adders and 1 subtractor were needed for the non-BISR implementation. However, if we consider both implementations, only 2 subtractors and 2 adders are needed. If the subtractor fails, we can use implementation 2a which needs 2 adders and 1 subtractor, and when the adder fails we can use implementation 2b which needs 2 subtractors and 1 adder.

In general, there exist a large variety of transformations, each of which reduces a computation in different ways. The transformations to reduce BISR overhead, however, can be classified into two classes: (1) Transformations to increase the chance for high resource utilization (and therefore reduced need) of the units of the same type as the failed EXU, and (2) Transformations to reduce the number of operations of the same types as the failed resources.

Some transformations can be used for both strategies simultaneously (e.g. inverse element law, distributivity, loop fusion and loop blocking), while others are specific to only one. The former group, for example, includes retiming (and functional pipelining), associativity, and loop permutation, while the latter group includes strength reduction (i.e. substitution of multiplication with constant by shifts and additions), constant propagation, dead code elimination and common subexpression elimination.

The remainder of this section illustrates how three important and powerful transformations, associativity, inverse element law, and retiming, can be used for high level synthesis based BISR. Note that although it is not explicitly stated, it is implied that transformations in the explanatory examples and in the final software application are supported by the commutativity transformation.

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

Table 1: Potential Schedules for Example of Fig. 3

Figure 3 shows the application of associativity for BISR. Notice that the only difference between Figures 3a and 3b is that associativity is applied so that shift A on Figure 3a and multiplication B on Figure 3b are the only operations which are not on the critical path. It is easy to figure out that the minimum hardware configuration, for the computation of Figure 3a requires 2 adders, 2 multipliers and 1 shifter. Associativity reduces the minimum BISR overhead, so that only one additional adder and one additional shifter are needed. Table 1 shows the feasible schedules when 3 adders, 2 multipliers and 2 shifters are available for various scenarios of unit failures. When a shifter fails, the implementation from Figure 3a is used, when a multiplier or adder fail the implementation of Figure 3b is used. (Actually, either 3a or 3b can be used when an adder fails).

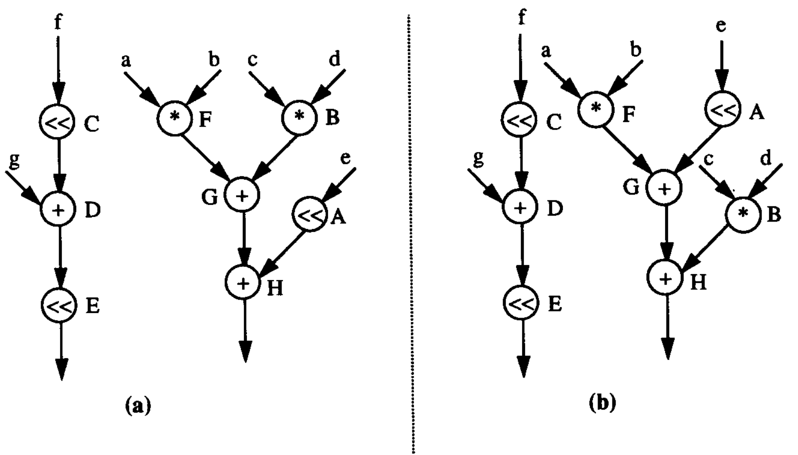

FIG. 3. Associativity for BISR: CDFG before (a) and after (b) application of associativity

The inverse element law transformation is used in the example shown in Figure 4. For an available time of 2, the non-BISR design can be implemented with 1 adder and 1 subtractor.

The inverse element law (combined with the enabling transformations of distributivity and associativity), gives the following two identities which can be used to transform 4a into 4b:

a - (b + c) = (a - b) - c

f + (d - e) = d - (e - f)

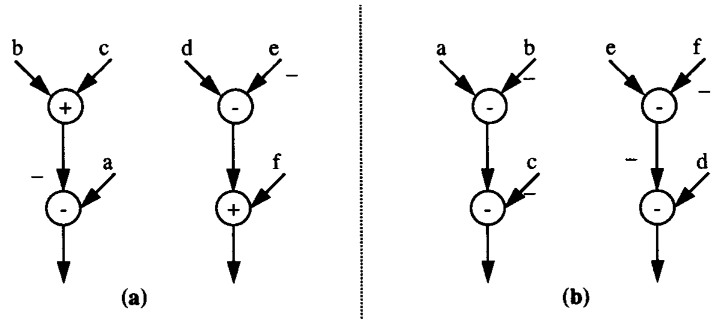

FIG. 4. Inverse Element Law for BISR

By using this transformation, just two subtractors and one adder are sufficient to enable the implementation of the required functionality, regardless of which unit is detected as faulty. If one of subtractors fails, the structure on the left side of Figure 4 is used; if the adder fails then the structure on the right side is used.

Note that it is sometimes possible to totally eliminate the need for a particular type of unit. The computation of Figure 4, for example, could be implemented for BISR using only three subtractors. This BISR scheme is not preferred however, since a subtractor is slightly more expensive than an adder. Also notice that this particular application of the inverse element law can similarly be applied to pairs of multiplications and divisions. In such a case, however, it would be significantly more efficient to use the BISR scheme which uses two dividers and one multiplier instead of the solution where three dividers are used.

Using a larger set of transformations (to include algebraic and redundancy manipulations) brings more options for the trading of operations. For example, $x^2 - y^2$ can be implement in two ways, either as $(x \times x) - (y \times y)$ or as $(x - y) \times (x + y)$. In this identity one multiplication can be traded for an adder.

Finally, Figure 5 shows how retiming can be used for high level synthesis BISR. The available time in this example is two control cycles.

Although retiming cannot, in this case, change the slacks [Cam91] on various operations, it can reshuffle the operation overlaps. This redistribution is done such that operations competing for a faulty unit are no longer bound to happen in the same control step. It is easy to figure out by analyzing the various schedules, that for the final BISR implementation, 3 subtractors, 2 adders and 2 shifters are sufficient. This results once again in a lower overhead than that achievable using only allocation and assignment.

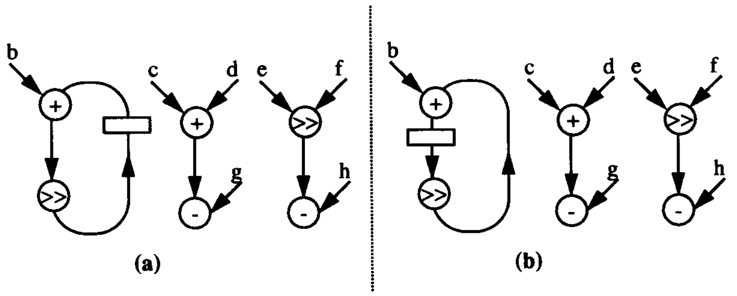

FIG. 5. Retiming for BISR

4.3 Optimization Algorithm for Transformation-Based BISR Hardware Minimization

The global strategy for transformation-based BISR design is illustrated in Figure 6. Note that transformations must be able to modify the graph so that a successful reassignment and scheduling can be obtained for all scenarios of failed units.

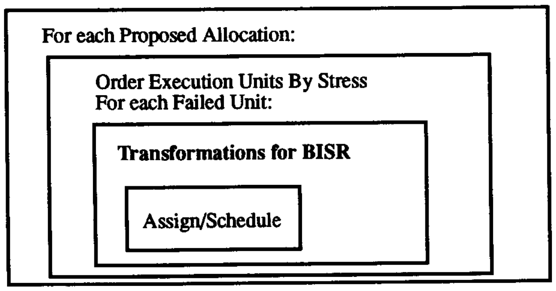
For each Proposed Allocation:
Order Execution Units By Stress
For each Failed Unit:
Transformations for BISR
Assign/Schedule

FIG. 6. Transformation Based Optimization: Global Strategy

As the basis for the BISR transformation-based optimization algorithm, a probabilistic sampling algorithm is used [Pot91]. The algorithm applies two types of basic moves: retiming and generalized associativity. Generalized associativity is a transformation that combines associativity moves with inverse element law and commutativity moves. The algorithm has two phases. The first phase is a global search using probabilistic sampling, where the design space is probabilistically evaluated to detect the k most promising starting points (m is a small integer which is a function of the number of nodes in the computation). A value of m=5 was used for the experimental results which are presented in the next section. In generating the starting points, we also change the number of operations of various types (e.g. subtraction vs. addition) using generalized associativity moves to trade off between the number of those operations.

The second, local optimization phase, uses the basic steepest descent approach to locally maximize these starting points. After each move, the objective function is evaluated, to get an estimate of the final area (execution units, interconnect, and registers) expected from the system.

This objective function is composed of 3 key parts, all of which are strongly correlated to the final area: the critical path, the number of delays, and a measure of the expected resource utilization of each hardware type (the overlap component). The objective function has been statistically validated on a set of 25 real-life examples, by showing the high correlation between it and the implementation cost of the final implementation [Poi91]. During the local optimization phase, the overlap components of the objective function are normalized by the available number of resources of each hardware type. When a unit is in short supply due to failure, the overlap component for the resource is large, and thus the algorithm will transform the graph in such a way that the need for this unit is alleviated.

When this algorithm is used for the optimization of BISR overhead, the proposed allocation is changed, by assuming that various units, one at time, have failed. Transformations are tried for the various scenarios in decreasing order of estimated difficulty. A heuristic global stress function is applied on the initial configuration and after each attempted scenario, to determine the estimated difficulty. 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 paths which are almost 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 on already transformed designs. The global stress is a nonlinear combination of these three factors, with statistically derived weighting parameters. The detailed description of all algorithms and functions, is presented in [Gue93].

Notice that both classes of transformations for BISR are utilized: (1) transformations to increase the chance for high utilization (and therefore reduced need) of the units of the same type as the failed EXU, and (2) transformations to reduce the number of operations of a failed type by trading operations of that type for other operations.

5.0 Experimental Results and Yield and Relative Productivity Analysis

Table 2 shows physical characteristics of several examples designed using the transformation-based methods of BISR design. The average area increase is only 9.7%, and an average of only 1.57 additional hardware units were needed. Note that an average of 3.71 additional hardware units would be used in a BISR implementation that exploited only assignment flexibility.

The consequences on yield and productivity can be calculated for all examples, and in Table 3, we present these values. For the yield and productivity calculations, we followed the state-of the-art procedure presented in [Sta92]. Over the last three decades, Stapper has presented a number of yield and productivity models. This effort started with a negative binomial distribution model for fault distribution. He then improved it by using a "unified negative binomial" model.

Example	IU	FU	NT	Non-BISR Area	BISR Area	Area Overhead (%)
11 FIR	8	9	4	5.45	6.5	19.3
7 IIR	7	9	4	9.27	9.92	7.0
35 FIR	7	8	4	12.31	13.34	8.4
55 FIR	14	16	4	20.77	23.44	12.9
8 IIR	16	18	4	24.85	27.04	8.8
LIN3	18	19	3	33.06	34.52	4.4
LIN4	21	23	3	36.00	38.49	6.9

Table 2: Physical characteristics of examples used during validation of transformations for BISR: IU - # of EXU units in non-BISR implementation; FU - # of EXU units in BISR implementation; NT - # of hardware classes; 11 FIR - 11th order high pass FIR filter; 7 IIR - 7th order low pass IIR filter; 35 FIR - 35th order Butterworth Flat Low Pass FIR filter; 55FIR - 55th order multiband FIR filer; 8IIR - 8th order IIR Avenhaus direct form II filter; LIN3 and LIN4 - 2 different 5-state linear controllers.

Recently, he has presented an even better model based on the combination of the binomial and beta distributions. The high accuracy of this model has been demonstrated on a variety of real-life production designs. At the same time the model is general enough to encompass even designs where limited repairability (e.g. using laser intervention) is used. Although Stapper's procedures primarily target BISR memory design, they have been regularly and successfully used in BISR for both dedicated [Kor84] and programmable [Pat89] datapath analysis.

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, Y1. A slight modification is made to the formula to take into account units of largely different areas. 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.

The initial yield was assumed to be 10%, as was assumed in Stapper's paper. Similar data was also used in [Kor84] and [Pat89]. We calculated changes in both yield and productivity, for various values of the variability parameter μ. This parameter 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]. For all examples, a significant improvement in relative productivity is apparent for all values of μ.

Example	Yield					Productivity				
	μ=0.5	μ=1.0	μ=2.0	μ=5.0	μ=Inf	μ=0.5	μ=1.0	μ=2.0	μ=5.0	μ=Inf
11 FIR	16.62	18.42	20.50	23.52	30.02	1.393	1.544	1.718	1.971	2.531
7 IIR	15.30	16.60	18.00	19.89	23.34	1.430	1.551	1.682	1.859	2.181
35 FIR	16.82	18.69	20.80	23.78	29.63	1.552	1.719	1.919	2.199	2.733
55 FIR	15.18	16.48	18.02	20.46	27.82	1.345	1.460	1.596	1.812	2.464
8 IIR	15.11	16.38	17.88	20.32	28.42	1.389	1.506	1.643	1.868	2.612
LIN3	15.49	16.89	18.58	21.37	31.62	1.484	1.618	1.780	2.047	3.029
LIN4	14.92	16.13	17.56	19.93	29.49	1.396	1.509	1.643	1.864	2.759

Table 3: Yield (in %) and Relative Productivity change due to use of transformations for BISR for 7 examples from Table 2 for various value 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 is increased, 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 actual 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

High level synthesis of datapaths has traditionally concentrated on synthesizing a specific implementation for a given computational problem. This paper has presented new techniques to compose a BISR implementation with a minimum amount of area overhead. BISR is an efficient yield, productivity, and reliability fault tolerance improvement technique, which will continue to gain importance especially with the increase in commercial significance of massive parallelism. However, until now the BISR scope has been restricted to the substitution of operation modules with only those of the same type. This paper has presented novel transformation techniques which support a new heterogeneous BISR methodology for ASIC designs. These methods are based on the flexibility of the design solution space and the exploration potential of transformations to find designs where resources of several different types can be backed up with the same unit.

High Level Synthesis for BISR is a new and unexplored field, which opens many new venues of research. Future research topics includes the extension of current ideas to support the use of other hardware models, the exploration of other high level synthesis tasks (e.g. module selection and partitioning), and algorithm selection (e.g. one of several types of filters with the same transfer function are selected depending on the available resources).

7.0 References

[Aru80] J.A. Arulpragasm, R.S. Swartz, "A Design for Process State Preservation on Storage Unit Failure," *10th Int. Symp. on Fault-Tolerant Computing*, pp. 47-52, 1980.

[Bit88] D. Bitton, J. Gray, "Disk Shadowing," *14th Conference on Very Large Data Bases*, pp. 331-338, 1988.

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

[Cha92] A. Chandrakasan, et al., "HYPER-LP: A System for Power Minimization Using Architectural Transformations," *IEEE ICCAD-92*, pp. 300-303, 1992.

[Che92] D. Chen, J. Rabaey, "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Real-Time Data Paths", ISSCC DIGEST of TECHNICAL PAPERS, pp. 74-75, 1992.

[Fis88] C.N. Fischer, R.J. LeBlanc, Jr., *Crafting a Compiler*, The Benjamin/Cummings, Menlo Park, CA, 1988.

[Gib89] G. Gibson, L. Hellerstein, R. Karp, R, Katz, D. Patterson, "Error Correction in Large Disk Arrays," *ASPLOS III*, pp. 123-132, ACM, 1989.

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

[Gue93] L. Guerra, M. Potkonjak, J. Rabaey: "High Level Synthesis for Reconfigurable Datapath Structures', Technical Report 93-C107-4-5510-9, NEC USA, Princeton, NJ, 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. Breuer, "On Area and Yield Considerations for Fault-Tolerant VLSI Processor Arrays," *IEEE Transactions on Computers*, Vol. C-33, No. 1, pp. 21-27, Jan. 1984.

[Lee92] T. Lee, W. H. Wolf, N.J. Jha, "Behavioral Synthesis for Easy Testability in Data Path Scheduling," *IEEE ICCAD-92*, pp. 616-619, 1992.

[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.

[Lew79] D.W. Lewis, "A Fault-Tolerant Clock Using Standby Sparing," *9th Int. Conf. on Fault Tolerant Computing, IEEE Computer Society*, Madison, WI, pp. 33-40, 1979.

[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, G. Gibson, R.H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *Proceedings SIGMOD*, pp. 109-116, 1988.

[Pat89] D.A. Patterson, J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[Pot92] M. Potkonjak, J. Rabaey, "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations," *IEEE ICCAD-92*, pp. 304-308, 1992.

[Rab91] J. Rabaey et al., "Fast Prototyping of Data Path Intensive Architectures," *IEEE Design & Test Magazine*, June 1991.

[Rag91] V. Raghavendra and C. Lursinsap, "Automated Micro-Roll-Back Self Recovery Synthesis," *28th ACM/IEEE Design Automation Conference*, pp. 385-390, 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.

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

[Sta92] C. H. Stapper, "A New Statistical Approach for Fault-Tolerant VLSI Systems," *The 23rd Annual International Symposium on Fault-Tolerant Computing*, pp. 356-365, Boston, MA, 1992.

[Sto90] M. Stonebraker, G. Scholss, "Distributed RAID - A New Multiple Copy Algorithm," *Conference on Data Engineering*, pp. 430-437, 1990.

[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.