Phantom Redundancy: A Register Transfer Level Technique For Gracefully Degradable Data path Synthesis

Ramesh Karri[1], Balakrishnan Iyer and Israel Koren
Polytechnic University, Brooklyn, New York, NY 11201[1]
University of Massachusetts, Amherst, MA 01002

Abstract

In this paper we present an area-efficient register transfer level technique for gracefully degradable data path synthesis called **phantom redundancy**. In contrast to spare-based approaches, phantom redundancy is a recovery technique that does not use any standby spares. Phantom redundancy uses extra interconnect to make the resulting data path reconfigurable in the presence of any (single) functional unit failure. When phantom redundancy is combined with a concurrent error detection technique, concurrent error detection followed by reconfiguration is automatic.

We developed a register transfer level synthesis algorithm that incorporates phantom redundancy constraints. There is a tight interdependence between reconfiguration of a (faulty) data path and scheduling and operation-to-operator binding tasks during register transfer level synthesis. We developed a genetic algorithm based register transfer level synthesis approach to incorporate phantom redundancy constraints. The algorithm minimizes the performance degradation of the synthesized data path in the presence of any single faulty functional unit. The effectiveness of the technique and the algorithm are illustrated using high level synthesis benchmarks.

Contact information: ramesh@india.poly.edu; **tel: 718 260 3596; fax: 718 260 3906**

1.0 Introduction

Advances in VLSI have made it possible to implement complex algorithms on a single integrated circuit (IC) with the attendant advantages of reduced power consumption, higher reliability and reduced size and weight. While increasing device densities have made it possible to implement such complex VLSI systems, they have also rendered the ICs highly susceptible to a variety of fabrication-time fault mechanisms. In many VLSI applications, it is not uncommon to experience circuit yields on the order of 10% or even less thereby increasing the cost of manufacturing the circuit.

A number of researchers have examined fabrication-time reconfiguration approaches to enhance the yield of ICs. These techniques identify failed functional units in a fabricated IC and program the wires to reconfigure the fault free functional units into a working IC. Built-In-Self-Repair (BISR) is a popular reconfiguration technique. BISR approaches have been applied mostly to regular architectures such as memory [1]. In BISR, reconfiguration is realized by providing a set of spare modules in addition to the core operational modules [1].

In this paper we present a register transfer level technique for reconfiguration of ICs called **phantom redundancy** that does not use spare modules. Rather, phantom redundancy uses redundant programmable interconnect. When a functional unit is faulty, the interconnection network in the data path is reprogrammed to configure the fault-free functional units into an operational data path albeit with a degraded performance. Phantom redundancy is applicable to both regular and non-regular data paths and entails small area overhead. Phantom redundancy does not perform CED. When combined with a concurrent error detection (CED) and faulty unit location technique such as

introspection [6], phantom redundancy can be used for dynamic reconfiguration in the field.

1.1 Related Research

VLSI reconfiguration techniques have been developed to make regular processor arrays tolerant to faults occuring during operation. Using a spare row (column) of processing elements, Negrini *et. al.* developed a rippling replacement strategy [14]. A faulty module is replaced with its neighbor in the same row (column). When both a spare row and a spare column are available the fault stealing strategy can be used. In fault stealing, a faulty module is replaced with a neighbor either in the same row or in the same column [14]. When multiple spare rows and columns are present a repair-most strategy can be used [15]. Repair-most strategy is based on a graph theoretic formulation and bipartite matching approach. An RT level reconfigurable data path synthesis technique based on spare functional units called built-in-self-repair (BISR) has been proposed by Guerra *et. al.* [3]. Instead of one spare module for each active module, BISR uses one spare module for each module type. All of these approaches use spare modules.

Tolerance to IC fabrication process related defects can be improved using two techniques. Tuning the process parameters can reduce such fabrication time defects in the device [19]. However, such process yield maximization does not totally eliminate the fabrication-related defects. Along an orthogonal dimension, defect-tolerant circuit design and layout techniques can maximize the circuit yield. While Chiluvuri and Koren [20] developed layout compaction algorithms to maximize defect-tolerance, Allan *et. al.* [21] proposed selective relaxation of the layout design rules. Phantom redundancy complements these layout level defect-tolerance.

RT level synthesis techniques for area optimal [8,9], performance optimal [10,11] and power optimal data path design have been explored [22,24]. RT level data path synthesis targeting off-line testability [23,26] and on-line testability [2,3,4,5,7,12] has also been addressed. In [2,5] it has been shown that recovery from transient faults can be done efficiently at RT level by checkpointing and roll back in hardware. Before, rollback based recovery or reconfiguration can be carried out, the faulty unit should be identified. Concurrent error detection (CED) and faulty unit location are hence important. Straightforward duplication entails significant area overhead. RT level techniques for area-efficient CED based on fault security were developed in [4,7]. RT level techniques using spare capacity in a design have also been proposed [6]. RT level reconfigurable data path synthesis technique using spare functional units has been proposed by Guerra *et. al.* [3]. On-line testable controller unit synthesis has been reported in [12]. The proposed technique can be used in combination with these CED techniques.

1.2 Issues in Gracefully Degradable Data path Synthesis

1.2.1 The Design Methodology

We propose to incorporate phantom redundancy reconfiguration constraints within a top-down VLSI design methodology. From among the various levels of abstraction in such a VLSI design methodology, the register transfer (RT) level is the right abstraction at which to incorporate phantom redundancy. This is because:

1. there is a tight interdependence between the synthesized data path and the reconfiguration of such a data path,
2. the fault model is at the RT level of functional units, and

3. data for reconfiguration such as the clock-by-clock schedule and operation-to-operator binding can be easily obtained at the RT level.

RT level synthesis involves (I) translation of a high-level algorithmic description into an intermediate representation called the Control Data Flow Graph (CDFG), (ii) assignment of operations in the CDFG to clock cycles (scheduling), (iii) mapping the scheduled operations onto available functional units (binding) and (iv) synthesis of the control unit.

It has been shown that scheduling and binding are NP-hard [13]. Besides, scheduling and binding are interdependent. Hence numerous heuristics have been proposed to solve these problems [9]. Most RT level synthesis systems solve scheduling and binding independent of each other. Since the synthesized architecture profoundly influences its reconfigurability, it should be integrated manner with the other synthesis tasks.

In this paper we developed a genetic algorithm [18] based technique to solve the simultaneous scheduling, binding and reconfiguration problem. The schedule and binding in the presence of any single functional unit failure is constructed simultaneously. This yields an RT level data path with a minimal degradation in performance.

1.2.2 Controller Issues

In a gracefully degradable data path the control unit is important since it orchestrates the reconfiguration. There are two viable options for designing a controller for reconfiguration.

1. Programmable Controllers: Although programmable controllers suffer from the disadvantage of slightly larger silicon area for implementation and a slightly lower performance, they have a major advantage in terms of ease of reconfiguration. Even in the absence of faults in the system the extra interconnect and the controller

programmability gives the user the option to implement new CDFGs on the architecture much more efficiently.

2. Composed Controllers: The controller for operating the fault free data path is composed with the controllers for each of the single unit failure scenarios. Although these composed controllers are smaller in size and faster they are hardwired.

1.3 Research Contributions

The important contributions of this paper are:

1. **Phantom Redundancy:** we present an area efficient technique for data path reconfiguration. Phantom redundancy adds extra programmable interconnect to make the resulting data path reconfigurable in the presence of functional unit failures.
2. **Integrating reconfiguration constraints with scheduling and binding:** We developed a genetic-algorithm-based global optimization approach for the synthesis of area-efficient gracefully degradable data paths. This is because the problem of reconfiguring a data path with minimal area overhead strongly depends on the original data path. The algorithm performs simultaneous scheduling, binding and reconfiguration to minimize the performance degradation in the presence of a functional unit failure. The reported technique is applicable to regular array architectures and non-regular data path based designs.

2.0 Phantom Redundancy

Phantom redundancy is an area-efficient approach to implement gracefully degradable data paths. Phantom redundancy uses additional interconnections and yields **gracefully degradable** data paths with low hardware overhead. Upon detecting a faulty functional unit, the interconnection network is programmed to perform the intended function on the

fault-free functional units albeit at a reduced throughput. Phantom redundancy can be used for fabrication-time and real-time reconfiguration of data paths. This capability is crucial in military and space applications where replacement of a faulty module is either impossible or prohibitively expensive.

Towards illustrating and clarifying the concept of phantom redundancy, consider a CDFG consisting of six operations a, b, c, d, e, f shown in Figure 1. 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 Figure 1 (a). The redundant interconnect shown as dotted lines in Figure 1 (b) make this data path gracefully degradable in the presence of any single functional unit failure.

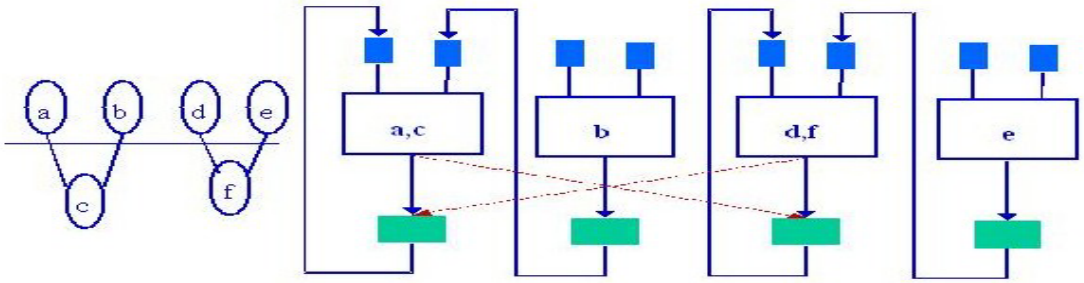

Figure 1: (a) Scheduled CDFG. (b) Data path implementing the CDFG. Two additional point-to-point links shown as dotted lines make the data path reconfigurable.

Upon identifying a faulty functional unit, the controller can be reprogrammed to operate the reconfigured data path with a degraded performance. For example, if functional unit F1 is faulty, operations a, c bound to it in the original data path are mapped to the fault-free functional unit F3 as shown in Figure 2 (a). Further, operation b is remapped to functional unit F4. This reconfigured data path operates at a degraded performance of 4 clock cycles (as opposed to 2 clock cycles in the fault-free data path).

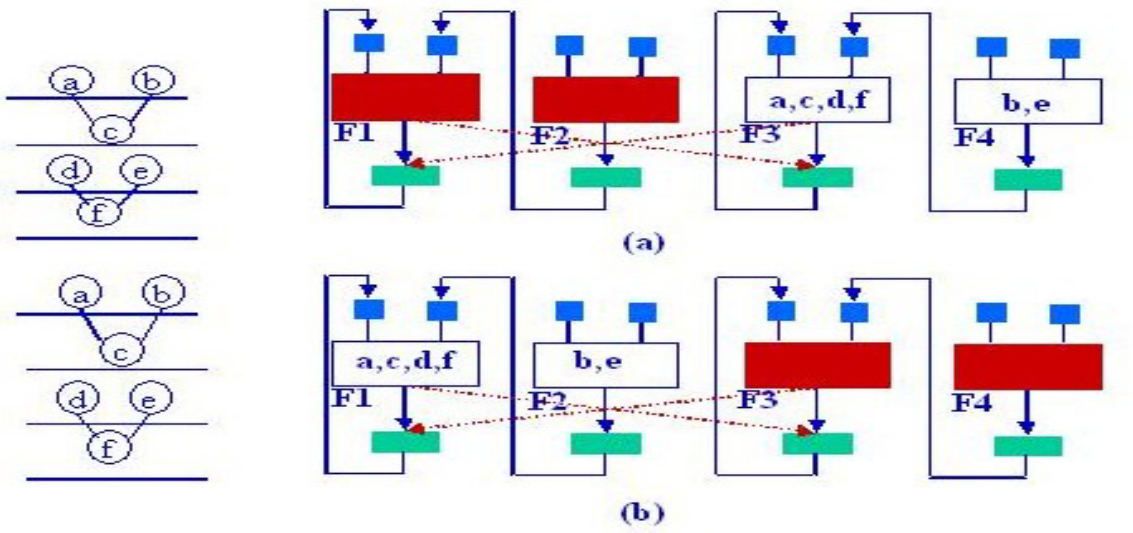

Figure 2: reconfiguration in the presence of faulty functional units

The corresponding schedule is also shown in Figure 2 (a). This data path can tolerate all single functional unit failures (see Figure 2 (a,b)). This data path does not use any spare modules but uses two additional interconnections. This data path can also tolerate 50% of all two-unit faults ((F1, F2) and (F3, F4)). For all these scenarios the reconfigured data path consumes twice as many clock cycles as the fault-free data path.

Consider another CDFG consisting of fifteen add operations (a1,..,and a15) as shown in Figure 3 (a). The schedule shown here uses three adders (A0, A1, A2). One possible operation-to-operator binding is shown in Figure 3. The functional unit on which an operation is carried is shown in capital letters. In Figure 3 (a), node a14 is scheduled in clock cycle 4 and is executed on adder A2. Until now scheduling and binding did not account for possible performance degradation in the presence of an adder failure. Arbitrarily reallocating the responsibilities of the failed unit among the fault free units increases the complexity of the interconnection network. **Consequently, we propose to reallocate the responsibilities of a failed unit to a single backup unit.**

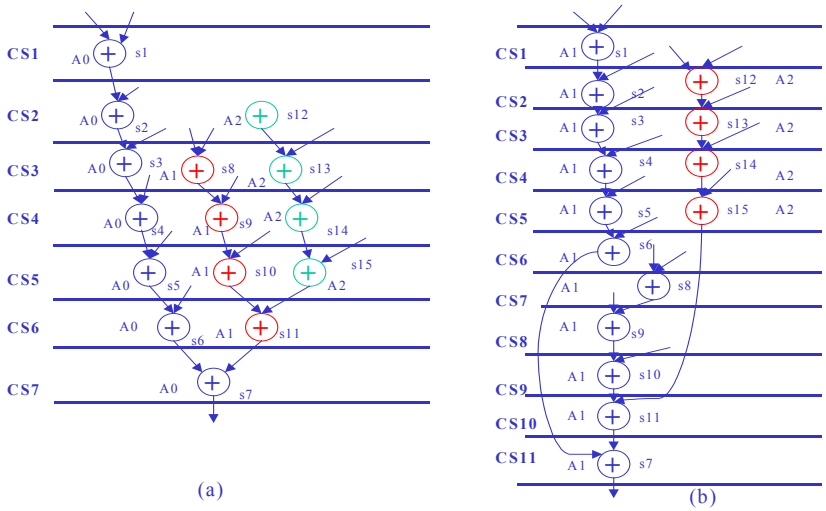

Figure 3: (a) Example schedule and binding that uses 3 adders (b) a reconfigured schedule in the presence of a faulty A0.

In Figure 3 reconfiguration can be achieved by identifying (A0, A1) and (A1, A2) to be two **backup pairs**. If A0 is faulty A1 takes over, if A2 is faulty A1 takes over and if A1 is faulty either A0 or A2 take over. This backup pair assignment entails a performance degradation of four clock cycles in the worst case. One such schedule that requires 11 clock cycles when A0 is faulty is shown in Figure 3 (b). In this reconfigured data path A1 takes over from a failed A0.

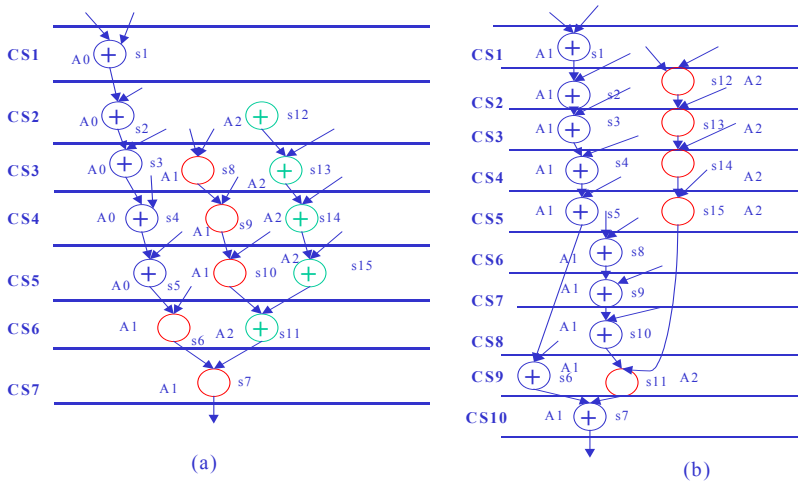

Figure 4: (a) A schedule and allocation obtained by simultaneously considering the performance of the primary data path and reconfigured data path (b) A reconfigured scheduled using 10 clock cycles in the presence of a faulty A0.

A data path can be made gracefully degradable by determining backup pairs following the scheduling and binding phases of RTL synthesis. However, this will yield architectures with a poor performance. Consequently, scheduling, binding and reconfiguration constraints will be considered in an integrated fashion. In Figure 4 an alternate schedule and binding with a worst-case performance degradation of only three clock cycles is shown. To find such solutions, we use genetic algorithm based approach to perform simultaneous scheduling, binding and creation of backup pairs.

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 its backup. It is the task of the RT level synthesis algorithms to explore these tradeoffs.

3.0 Gracefully Degradable Data path (GDD) Synthesis

3.1 Architectural Model

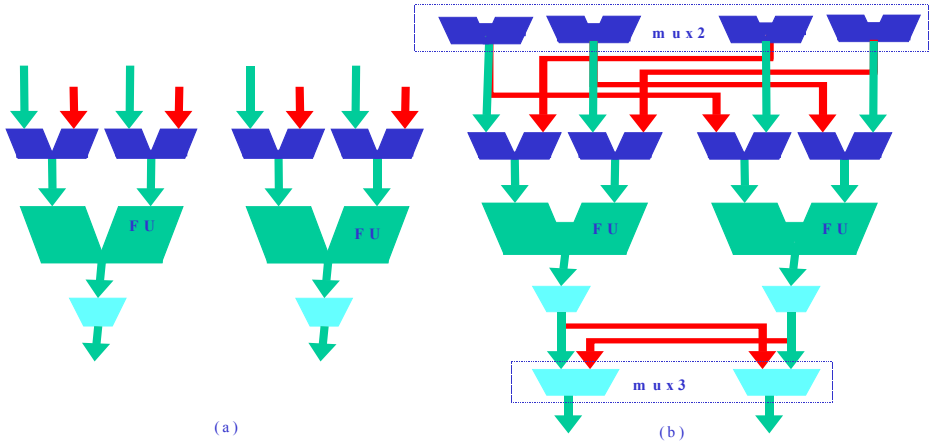

Figure 5: (a) Hardware model for basic data path synthesis. (b) Hardware model for gracefully degradable data path synthesis using phantom redundancy. Extra interconnections and extra layers of multiplexers and demultiplexers are used to form the backup pairs.

The hardware models for the basic and gracefully degradable data path synthesis are shown in Figure 5. Extra levels of multiplexing (mux2) at the input and at the output (mux3) together with the additional links combine two similar functional units in the basic data path into a backup pair. Within such a backup pair, if one of the functional units is faulty, its functionality is switched to the other functional unit by programming the select signals of mux2 and mux3. In general, to have redundancy among N functional units of a given type, at least $\left\lceil \frac{N}{2} \right\rceil$ backup pairs are necessary. In the final implementation, the two level multiplexers shown in Figure 5 (b) can be flattened into a single level. Observe that adding these extra multiplexors and registers will degrade the performance of the datapath circuit even if there is no faulty unit. The backup pair based hardware model with the additional level of multiplexes increases the clock cycle duration. Although this backup pair hardware model guarantees single unit fault

tolerance, some multiple faults can be tolerated. The data path in Figure 2 (b) can tolerate 50% of all two-unit faults ((F1, F2) and (F3, F4)). Although this backup pair model can be extended to tolerate all multiple functional unit failures, the number of links required in this case would be larger and the performance degradation would also be greater. Further, it has been shown that single unit tolerance is sufficient in most cases.

3.2 RT Level fault model

Our functional fault model is based on the observation that the critical area (i.e., the area susceptible to 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 in the buses and register files. Hence, we initially target single functional unit failures only. Faults in a bus or a register file results in faulty data being fed to all the units that are connected to it. Hence, fault in a bus or a register file is equivalent to multiple functional unit failures. Faults in those buses and register files that feed into a single functional unit can be targeted using phantom redundancy. The controller fault-tolerance can be implemented using the technique presented in [12] or by straightforward duplication.

3.3 Genetic Algorithm based Gracefully Degradable Data path Synthesis

We will outline a genetic algorithm [18] based approach to synthesizing gracefully degradable data paths. GAs have been used in a wide variety of optimization tasks, including the traveling salesman problem, circuit design and job shop scheduling [27,28]. A genetic algorithm is based on the principles of the evolution via natural selection. It employs a population of individuals that undergo selection in the presence of mutation and recombination (i.e. crossover) operators. These two operators introduce variation into

the individuals in a population. A fitness function is then used to evaluate individuals, and reproductive success varies with this fitness.

An initial population M(0) is randomly generated. The fitness f(i) for each individual i in the current population M(t) is computed. Selection probabilities p(i) for each individual i in M(t) are defined such that p(i) is proportional to f(i). Population M(t+1) is generated by probabilistically selecting individuals from M(t) and combined to produce offspring via the mutate and crossover genetic operators. Applying crossover and mutation probabilistically modifies the individuals in a population. The crossover and mutation operations depend not only on the problem structure but also on the way the solution is encoded as chromosomes. Crossover exchanges partial solutions from two chromosomes that have been probabilistically selected based on their fitness functions. Mutation is applied with a very low probability to introduce new search points. This process is repeated until either the best solution is found or the maximum number of generations is reached.

A genetic algorithm can be applied to a problem, once the solutions to the problem are encoded as chromosomes. An effective genetic algorithm representation and a simple and meaningful fitness function are key to the success deployment of a GA.

The gracefully degradable data path synthesis problem can be formulated as follows: Given a CDFG and a hardware model, synthesize a gracefully degradable data path such that:

1. Performance of the unimpaired system is not compromised.
2. Performance degradation in the event of any single functional unit failure is minimal.
3. Area overhead of reconfiguration is minimal.

We will now outline the fitness function, the problem specific coding of the solutions, the genetic operators and the fitness proportionate stochastic selection scheme of the genetic algorithm to solve this RT level synthesis problem.

3.3.1 The Fitness Function

The proposed algorithm simultaneously explores the time and space domains of the design space. A candidate solution α is evaluated by the following fitness function

$$C(\alpha) = w_2 \times \text{area}(F) + w_3 \times \text{area}(M) + w_4 \times \text{area}(X) + w_5 \times \text{area}(IC) \times (w_1 \times S)^{\gamma}$$

where, w_i are user defined weights, F is the set of functional units, M is the set of registers, and X is the set of multiplexers, IC is the interconnect complexity, S is the number of clock cycles and γ is a user defined parameter. Thus the fitness function is of the form area $\times$ time $^{\gamma}$.

The interconnect complexity IC is obtained as the weighted sum of the area required for the links associated with the inputs and outputs of each functional unit and the number of buses required to provide the requisite data transfers as given below.

$$IC(\alpha) = \sum_{f_i \in F} (\rho(i_1(f_i)) + \rho(i_2(f_i)) + \rho(o(f_i))) + wb \times \max_{s_i \in S} \left(distinct \quad src / \sin ks(s_i) \right)$$

where, $i_1(f_i)$, $i_2(f_i)$, and $o(f_i)$ denote the sets of variables assigned to the left input, the right input and the output, respectively of functional unit f_i. A $\rho(i_1(f_i))$ input multiplexer is provided at input f_i of the functional unit if the number is greater than one. The area cost of the multiplexers is obtained from a table containing the area of the multiplexer for different number of multiplexer inputs. $\rho(var)$ is the minimum number of registers required to store the set of variables *var*, given the lifetime table of the variables

in var. This is obtained using the left-edge algorithm [25]. The number of links is computed as the sum of the number of links used by each of the functional units in the architecture. The number of buses required is the maximum number of distinct sources and sinks over all time steps.

3.3.2 Problem specific genetic coding

Each valid solution is encoded using four **chromosomes (strings)** g_1, g_2, g_3 and g_4.

Module selector string g_1 encodes the space dimension of the solution. For each functional unit type there is one gene (element) in g_1 indicating the number of units of that type. Each element in the string g_1 can vary from 1 to n_{tmax}, where n_{tmax} is the maximum number of functional units of type t. The value of n_{tmax} is either assigned by the user or is computed from an as soon as possible schedule without resource constraints. For example, for the CDFG shown in Figure 7, two adders and two multipliers are required as shown in string g_1.

Module allocation string g_2 encodes the functional unit on which an operation is carried out. For each operation in the CDFG there are two adjacent entries in the string g_2. The first entry gives the index of the functional unit that will carry out the operation. The second entry is a binary value that indicates if the operator is commutative. Commuting the inputs is useful in optimizing the multiplexers at the inputs to the functional unit. For example, from the g_2 string in Figure 7 it can be seen that operation a2 is performed on adder A0 whose left and right inputs can be interchanged.

Time stamp string g_3 encodes the time dimension of the scheduling and binding problem. For each node in the CDFG there is one entry in string g_3. Data dependencies between operations in the CDFG complicate the coding scheme. Towards ensuring that

the crossover and mutation operators yield valid schedules in the presence of data dependencies we use two techniques. Firstly, the genes within a chromosome are placed based on a depth-first ordering of the corresponding nodes in the CDFG. Secondly, a **differential coding scheme** is used wherein each gene denotes the time by which the corresponding operation is deferred from the earliest schedulable clock cycle. From the g_3 string in Figure 7, it can be seen that operations a2, a3 and a7 are scheduled one clock cycle away from their earliest schedulable clock cycles.

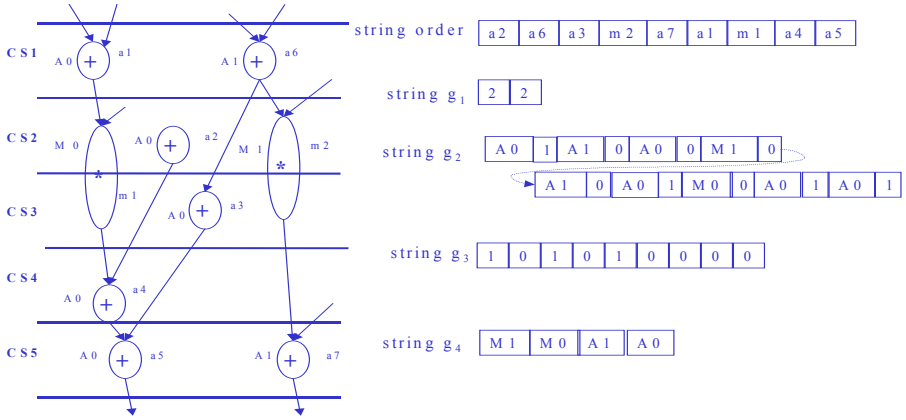

Figure 7: An Example CDFG and g_1, g_2, g_3 and g_4 strings.

Backup pair string g_4 identifies the backup pairs in the reconfigured architecture. A natural representation for g_4 is as a permutation of the indices of the functional units with the permutations corresponding to a single functional unit being grouped together in the string. The elements in the string g_4 are read pair wise in a cyclic order from left to right and each of these pairs form one backup pair. In the event of a failure of any single functional unit, the other unit in the backup pair takes over. The functional units should be partitioned in such a way that the worst-case performance degradation in the presence of any single functional unit failure is minimal.

3.3.3 Construction of a schedule and binding

Strings g_1 through g_4 are used to construct the schedule, binding and backup pairs of the functional units. First constructing a schedule and binding assuming that none of the functional units fail create a gracefully degradable data path. Then a schedule and binding is constructed by assuming single faulty functional units as follows. There are two types of operations: those that are bound to a faulty unit and those that are bound to a fault free unit.

An operation that is bound to a faulty functional unit is re-bound to its backup unit. If the backup unit is free, the operation is scheduled in that clock cycle. Otherwise, the operation is deferred to a later clock cycle when the backup unit becomes available.

An operation bound to a fault-free functional unit may have to be re-scheduled and re-bound when its predecessor operations in the CDFG are re-scheduled and re-bound. If the functional unit is still free, then the operation is scheduled into the clock cycle. Otherwise, if the backup unit is free then the operation is re-bound to the backup unit. In the worst case, the operation is deferred until one of the units in the backup pair becomes available.

Faults in different functional units impact the performance degradation of the data path to a different extent. Hence, the worst-case performance degradation δ of the gracefully degrading data path should be considered. The worst-case performance degradation δ is measured, as the additional clock cycles required executing the CDFG. The cost function is modified to account for δ as follows:

$$C(\alpha) = (w_2 \times \text{area}(F) + w_3 \times \text{area}(M) + w_4 \times \text{area}(X) + w_5 \times \text{area}(IC)) \times (w_1 \times S + w_6 \times \delta)^{\gamma}$$

Where, w_6 is the user-defined weight for the worst-case performance degradation δ for the gracefully degradable data path and S is the number of clock cycles for the basic data path.

3.3.4 Genetic Operators

The genetic encoding employed allows the use of simple crossover and mutation operators. We use the **single point crossover** for the strings g_1, g_2, and g_3. The minimal area overhead constraint requires that each functional unit have only one backup unit. This implies that the index of each functional unit must appear only once in string g_4. This cannot be enforced using the single point crossover. Hence we employ the **partially matched crossover (PMX)** [18] for string g_4. PMX ensures that all functional unit indices used in a design appear at least once and no more.

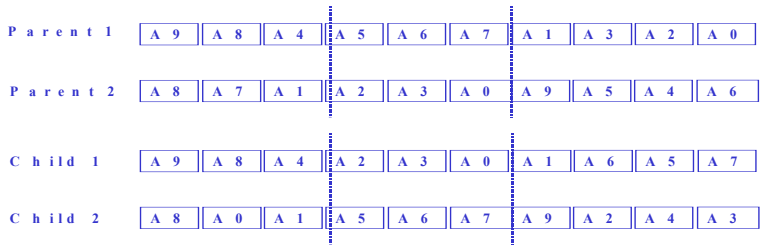

Figure 8: partial matched crossover (PMX)

In PMX, two parent strings are aligned and two crossover sites are selected at random along the strings. These two points define the matching section that is used to effect a crossover through position-by-position exchange operations. This is shown in Figure 8. PMX proceeds by position-by-position exchange between the two strings. First, the string $Parent_2$ is mapped to $Parent_1$, and the entries in the matching section are exchanged. In Figure 8, the sub string A2A3A0 $Parent_2$ is mapped on to the corresponding sub string in $Parent_1$. This results in duplicate A2, A3 and A0. To remove these matching elements A4,

A5 and A6 from $Parent_2$ are exchanged yielding the $Child_1$ string. Following this, the string $Parent_1$ is mapped on to $Parent_2$ and the matching entries are once again exchanged. A simple shuffle operator that randomly exchanges two genes of the same function unit type is used as the **mutation** operator.

3.3.5 Selection scheme

An individual is selected for crossover and reproduction with a probability that is proportional to its fitness [18]. This selection scheme increases the representation of above average fitness chromosomes in the population and has a marked effect on the performance of the genetic algorithm. The genetic operators exploit this increased concentration of quality chromosomes in constructing better solutions as the generations evolve.

4. Results

We summarize the trade-off studies conducted on a set of benchmark examples including the fifth order elliptic wave digital filter (EWF5), an AR filter (ARMA), a third order bilinear loss-less discrete integrator filter (LDI3), and the FIR filter (FIR16).

4.1 Phantom Redundancy using 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. Also, we assume that the adder is also capable of carrying out subtract operations. The system word length is 24 bits and the filter coefficients are 16 bits.

Ex. Name	Number of		Perf. (Clock Cycles)			# of registers		Chip area			% Overhead		
	Mult	Add	Orig	Degr	% Degr	Orig	Phant	Orig	Phant	BISR	Phant	BISR	% impr
Ewf5	3	3	17	19	11.76	6	8	1.82	1.84	2.40	1.47	31.78	30.31
Arma	2	2	19	34	78.95	6	15	1.24	1.30	1.82	4.59	46.58	41.99
Arma	4	3	11	18	63.64	6	12	3.15	3.21	3.92	1.91	24.50	22.59
Arma	4	2	11	18	63.64	6	13	3.12	3.18	3.89	2.06	4.75	22.69
Ldi3	2	2	8	11	37.50	3	5	1.22	1.25	1.80	1.73	47.17	45.44
Fir16	4	4	8	12	0.00	8	11	2.80	2.84	3.47	1.42	24.09	22.67
Fir16	8	8	6	8	33.33	8	9	7.84	7.89	8.81	0.67	12.31	11.64

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

Results of this experiment are summarized in Table 1. The second and third columns show the number of multipliers and adders used in the design. While the fourth column (titled basic) shows the number of clock cycles required for executing the CDFG when the system is unimpaired, the fifth column gives the worst case execution time (in clock cycles) to compute the CDFG in the presence of any single functional unit failure. The next column summarizes the percentage degradation in performance. Supporting phantom redundancy entails use of additional multiplexers at the inputs and outputs to the modules used in the data path. This increases the clock cycle duration by about 10% when compared to the clock cycle of the basic data path. The overall performance overhead when this is considered is 1.1x of that discussed in the rest of the paper. To compare the performance overhead of the proposed scheme viv-a-vis the BISR scheme, we assume that the performance of a BISR design is same as that of the basic design. Hence, the reported performance degradation for phantom redundancy is also its performance degradation when compared to BISR. Number of registers in the original design and number of registers in the gracefully degradable design appear in the next two

columns. The chip area estimates for the original IC, the gracefully degradable IC, and the BISR IC are given in the columns titled Chip Area Orig., Chip Area Phant. and Chip Area BISR. The reported chip area estimates were obtained using HYPER hardware database [11]. These area estimates include the controller area, are fairly accurate and are known to be, in the worst case, 15% off the actual layout areas [11]. The BISR area was computed by assuming one redundant functional unit of each type and does not consider any additional interconnect and multiplexing overheads. Thus the figures quoted for BISR consistently underestimate the area overhead of an actual implementation. The percentage area overheads for the phantom redundancy and BISR over that of the original chip area are indicated in the twelfth and thirteenth columns. The last column summarizes the percentage savings in area overhead of the phantom redundancy vis-a-vis BISR.

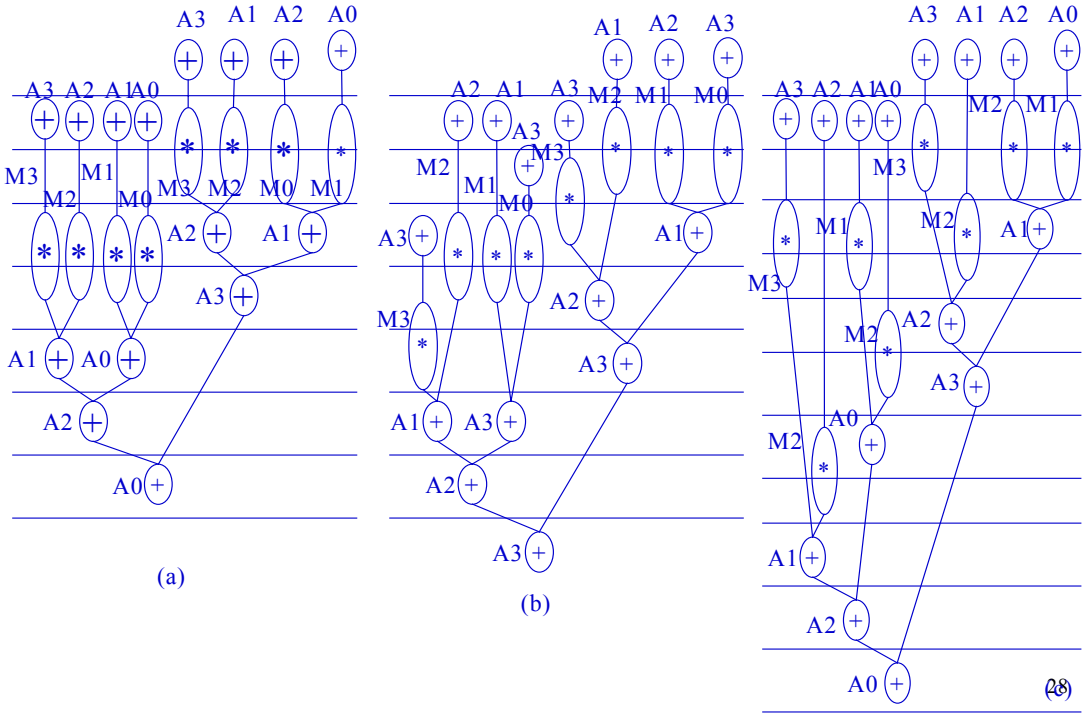

Figure 9: (M0, M2), (M1, M3), (A0, A3) and (A1, A2) are the back up pairs. (a) 8 clock cycle Schedule and binding (b) Schedule and binding of the reconfigured data path when

adder A0 fails (c) Schedule and binding of the reconfigured data path when multiplier M0 fails.

From Table Table 1, it can be seen that on an average, phantom redundancy entails 28.19% less area (with a standard deviation of 11.96%) when compared to BISR. The figures for LDI and EWF5 indicate that large savings in area can be obtained at the cost of a small performance loss - 2 clock cycles for the EWF5 and 1 clock cycle for LDI3. While area overhead of phantom redundancy is negligible when compared to that of the original design, additional area required for BISR corresponds to a significant proportion of the original chip area. If the basic design has a large number of functional units, then the phantom redundant design has to support one backup schedule for each possible failure scenario. This is the main source of controller overhead and contributes to the overall area overhead.

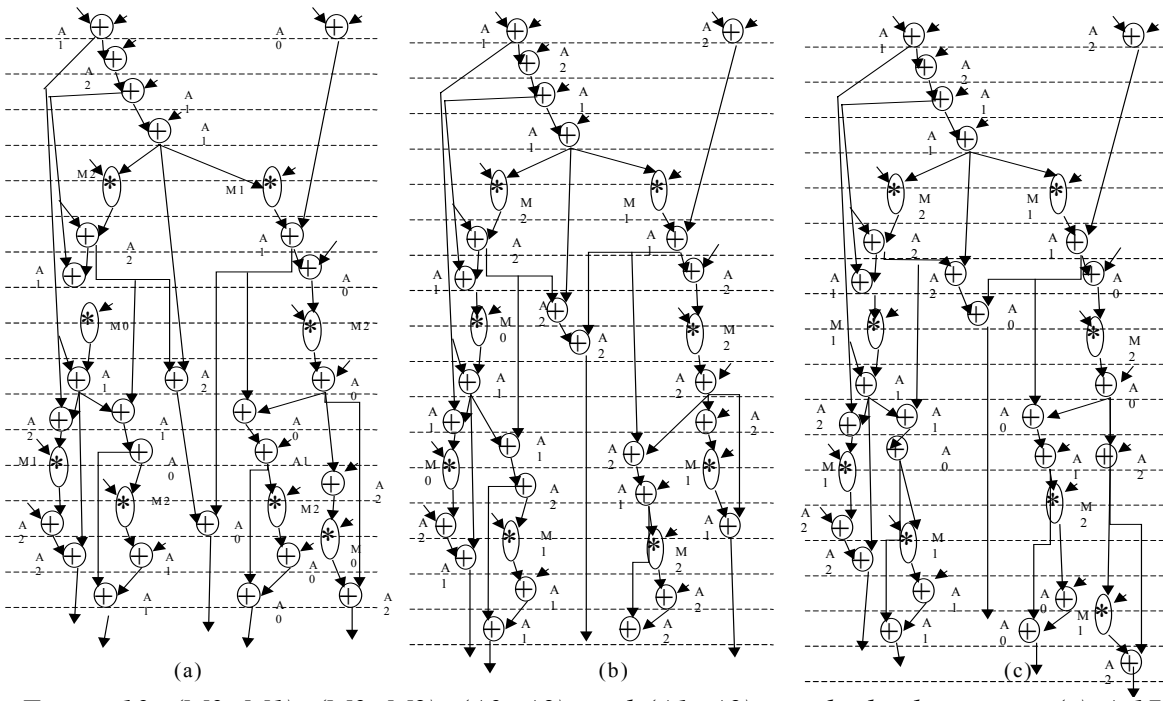

Figure 10: (M0, M1), (M0, M2), (A0, A2), and (A1, A2) are the backup pairs. (a) A 17-clock cycle schedule (b) Schedule and binding of the reconfigured data path when adder A0 fails (c) Schedule and binding of the reconfigured data path when multiplier M0 fails.

Performance degradation of these gracefully degradable data paths using non-pipelined functional units is quite significant. 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 factors. Firstly, the number of functional units of a given type is very small resulting in a small number of operational functional units in the presence of a failure. Secondly, the multiplication and the addition operations in the AR filter are clustered. This dramatically increases the critical path in the algorithm.

The schedule and binding corresponding to the seventh row Table 1 for the 16- tap FIR filter with 4 adders and 4 multipliers is shown in Figure 9 (a). The schedule and binding when adder A0 fails is shown in Figure 9 (b). The performance degradation is one clock cycle. The schedule and binding when multiplier M0 fails is shown in Figure 9 (c). The performance degradation is four clock cycles. Similarly, a 17-clock cycle schedule for EWF5 with 3 adders and 3 multipliers (corresponding to second row in Table 1) is shown in Figure 10 (a). The reconfigured 18-clock cycle schedule and binding when adder A0 fails is shown in Figure 10 (b). The reconfigured 19-clock cycle schedule and binding when multiplier M0 fails is shown in Figure 10 (c).

4.2 Impact of Pipelined Functional Units on Phantom Redundancy

We have seen that non-pipelined functional units entail large performance degradations. However, all these reconfigured ICs are obtained at no or minimal additional cost, thereby increasing the effective number of usable ICs (perfect ICs + partially good ICs). We will now assess the impact of pipelined functional units on performance degradation.

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**.**

	# of		Clock cycles		% Perf Degr	# registers		IC area		% Overhead		% Improvement	
	×	+	Orig	Degr		Orig	Phant	Orig	Phant	BISR	Phant	Phant	BISR
Ewf5	2	3	17	19	11.76	7	10	1.27	1.30	1.85	2.29	45.53	43.24
Arma	2	2	13	19	46.15	6	14	1.24	1.29	1.81	4.18	46.58	42.40
Arma	4	3	11	13	18.18	6	10	3.15	3.19	3.92	1.48	24.50	23.02
Arma	4	2	11	16	45.45	6	10	3.12	3.16	3.89	1.40	24.75	23.34
Ldi3	2	2	8	9	12.50	3	4	1.22	1.24	1.80	1.31	47.17	45.85
Fir16	4	4	7	9	28.57	6	9	2.79	2.83	3.46	1.43	24.19	22.76
Fir16	8	8	6	7	16.67	8	8	7.84	7.89	8.81	0.56	12.31	11.74

Table 2: Impact of pipelined functional units on phantom redundancy

The best results were obtained for the EWF5 example where the performance degradation is 11.76% while the area overhead incurred is only 2.29%. The BISR strategy leads to no performance degradation but the area overhead amounts to 45.53%. The worst performance degradation occurs for the ARMA example with 2 multipliers and 2 adders. The increase in degradation in performance 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. This leads to a 45.45% penalty in performance when one of the adders fails. On an average the savings in area over the spare-based BISR approach is 30.34% while the standard deviation is 13.28%. The average degradation in performance is 25.61% and the standard deviation is 14.85%.

There is a marked reduction in performance degradation when pipelined functional units are used. This is because pipelined units permit the initiation of operations at a much higher rate when a functional unit fails. Thus operations assigned to a failed unit are

scheduled in earlier steps as opposed to waiting for a multi-cycle operation to be completed before the next operation can be initiated. Pipelining is particularly effective when operations of a type are clustered in the CDFG. The reduction in performance degradation will be even more significant with deeply pipelined functional units (multiple pipeline stages).

4.3 Phantom Redundancy using multifunctional ALUs

In Table 3 we present the results when multifunctional ALUs are employed. The ALU is assumed to carry out multiplication, addition and subtraction in a single clock cycle. The period of each clock cycle is, however, considerably longer.

Ex	#	Clock cycles		% Perf	# of registers		Chip Area		% Overhead		% Impr	
Name	ALU	Unimp	Degr	Degr	Orig.	Phant	Orig	Phant	BISR	Phant	Phant	BISR
Arma	3	12	15	25.0	6	11	1.75	1.78	2.32	1.93	33.08	31.14
Arma	4	9	14	55.56	7	14	3.06	3.12	3.83	1.92	25.20	23.28
Arma	6	8	10	25.0	6	12	4.53	4.59	5.30	1.27	17.02	15.75
Fir16	4	9	13	44.44	9	13	2.70	2.73	3.37	1.29	25.03	23.74
Fir16	8	5	7	40.00	8	11	7.52	7.57	8.49	0.63	12.82	12.19

Table 3 : Results on benchmark examples using multifunctional ALUs

The results for the examples using ALU type functional units are summarized in The average saving in area over the BISR technique is 21.22% and the standard deviation is 7.42%. The average degradation in performance is 38% and the standard deviation is 13.15%. The degradation values are higher in this table as compared to Table 1 and Table 2 because the number of functional units used in the examples is much smaller.

4.4. Phantom redundancy based on an enhanced fault model

A closer look at the components of the area of an IC shows that interconnect uses about 50% of the total area. Hence, targeting faults in single functional units alone may not be

sufficient. Consequently, we adopted an enhanced functional fault model proposed in [3] that targets single faults in functional units, register files and interconnections. A fault in a register file is considered as a fault in the functional unit that the register file feeds, while a fault in an interconnect line is considered as a fault in the functional unit/register file from which it emanates. These constraints are then modeled as register and bus allocation constraints.

Table 4 summarizes the results of phantom redundancy based data path synthesis on four additional benchmark examples (the HAMMING code, 7^{th} order IIR filter, 8^{th} order IIR filter and WANG's discrete cosine transform) and using this enhanced fault model. The first major column shows the benchmark, the second major column identifies the type of design (basic and phantom redundancy based data path) and the third major column shows the number of clock cycles in the design.

		Clock cycles	# of schedules	Hardware allocation					Area (mm^2)			overhead
				+	-	*	Reg	Mux	Active	interconnect	Total	(%)
HAMMING	basic	8	1	6	0	1	83(15)	18	25.1	23.9	49.0	-
	phant	9-10	3	6	0	1	113(16)	28	29.8	29.4	59.3	21.0
IIR7	basic	10	1	2	2	2	39(20)	11	32.1	23.0	55.1	-
	phant	11-14	5	2	2	2	62(20)	16	35.2	30.4	65.6	19.1
IIR8D	basic	20	1	3	0	3	48(30)	9	42.1	36.3	78.4	-
	phant	21-22	6	3	0	3	89(39)	26	47.9	51.8	99.7	27.2
WANG	basic	10	1	3	3	4	54(44)	17	31.8	33.5	65.3	-
	phant	12	7	3	3	4	74(44)	59	37.6	40.6	78.2	19.7

Table 4: Results on benchmark examples using the enhanced fault model

The fourth major column shows the number of backup schedules (each with a different latency) required to tolerate all possible 1-unit faults. The five minor columns of the fifth major column show the number of adders, subtractors, multipliers, registers and multiplexes in the design. The sixth major column summarizes the area of the design. For each design, the area includes the active area (data path +controller) and the interconnect area. Finally, the last major column shows the area overhead of the phantom redundant

data path design. Corresponding to each benchmark, there are two rows in the table –the basic and phantom redundant. Except for HAMMING data path, all designs tolerate all 1-unit faults. For the HAMMING, the basic design uses a single multiplier and hence a fault in it cannot be tolerated without a spare module. Enforcing the constraints based on the enhanced fault model resulted in the larger area overhead (15% -30% range).

4.5 Discussion

Phantom redundancy is applicable only when there are at least two functional units of each type. Performance degradation is due to two sources. Additional multiplexers used at the inputs and outputs of functional units used to create the backup pairs are the first source. The second source is the additional clock cycles used to perform the computation in the presence of a faulty unit. Performance degradation is quite severe in small data paths that use a small number of functional units. There is a corresponding increase in the number of registers. Intermediate results in a gracefully degradable data path need to be stored longer before being consumed by the fault-free functional units. This increase in lifetime of the intermediate results in the computation translates into an increase in the number of registers.

Pipelined functional units entail smaller performance degradation when compared to multi-cycle non-pipelined functional units. This is particularly true when deeply pipelined functional units are used. This is because pipeline units permit the initiation of operations at a much higher rate. Thus operations assigned to a failed unit can be re-scheduled earlier clock cycles when compared to multi-cycle functional units. Benefit of deploying pipelined functional units is especially evident when operations of a type are clustered in the CDFG.

Phantom redundancy can be used for defect tolerance in mature fabrication processes where the process yield is sufficiently high to make the area overhead of BISR unreasonable. In mature fabrication process the phantom redundancy technique provides a low cost technique for improving the yield of ICs by yielding partially good chips. In application scenarios where the targeted performance quality is not stringent, phantom redundancy is once again a good alternative. BISR does not entail performance degradation and may be preferable in designs where tight performance constraints on the system should be met. BISR is good for new process lines where stringent performance standards must be satisfied.

The synthesis results showed that the interconnect area is about 50% of the total area. Phantom redundancy technique presented in this paper that is based on enhanced fault model can tolerate single functional unit failures, registers, multiplexers and interconnection lines. Graceful degradation in the presence of single functional unit failures is sufficient in almost all application scenarios. Further, these single-fault tolerant designs yield some amount of multiple-fault tolerance as well. Guaranteed reconfiguration around multiple functional unit failures can also be handled as an extension of the proposed technique.

The controller overhead associated with phantom redundancy based graceful degradation is due to extra control signals to support execution of multiple backup schedules in the presence of faulty functional units. Notwithstanding this important source of controller overhead, the reasons for small control area overhead associated with phantom redundancy are three-fold (less than 5% of the total area overhead):

1. In data dominated designs, the data path tends to occupy far more area than the controller. So the impact of control on the overall area of the chip is small.
2. In the hardware model that we use the control is distributed throughout the design and any increase in controller area is accommodated in the dead areas of the layout.
3. While the complexity of the control logic increases, it does not correspond to the controller area in a one-to-one fashion. The controller area is a non-linear function of the schedule, allocation and mapping. Moreover, state assignment for the new controller results in a different encoding of the states and hence is a factor in keeping the control overheads low.

If controller fault-tolerance is necessary, then existing techniques such as [12] or straightforward duplication can be used.

5.0 Conclusions

In this paper we presented a low-cost RT level technique for designing gracefully degradable data paths called phantom redundancy. Phantom redundancy adds extra interconnections to make the resulting data path degradable in the presence of any single functional unit failure. The proposed technique supports reconfiguration and should be used in conjunction with a concurrent error detection technique. We demonstrated the interdependence of the scheduling, allocation and reconfiguration and presented a genetic algorithm based global optimization approach for the synthesis of gracefully degradable data paths. Gracefully degradable data paths can be used for performance-based binning of designs at fabrication-time or graceful degradation in real-time. The effectiveness of phantom redundancy is demonstrated on several benchmark examples and compared it with the BISR approach.

References

1. B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989.
2. D.M. Blough, F.J. Kurdahi, and S.Y. Ohm, "Optimal Recovery Point Insertion For High-Level Synthesis of Recoverable Microarchitectures", *Proceedings IEEE Fault Tolerant Computing Symposium,* Jun 1995.
3. L. Guerra, M. Potkonjak, and J. Rabaey, "High-Level Synthesis Techniques for Efficient Built-In-Self-Repair", *IEEE Transactions on VLSI Systems,* Sep 1998.
4. R. Karri and A Orailoglu, "High-Level Synthesis of Fault-Secure VLSI Digital Signal Processors", *IEEE Transactions on Reliability*, Sep 1996.
5. R. Karri and A. Orailoglu. "Self Recovering Microarchitecture Synthesis", *IEEE Transactions on Computers,* Feb 1996.
6. Ramesh Karri and B. Iyer "Introspection: A Register Transfer Level Technique for Concurrent Error Detection and Diagnosis in Data Dominated Designs," *ACM Transactions on Design Automation of Electronic Systems,* vol.7, no.1, Jan 2002.
7. G. Lakshminarayana, Raghunathan, N. K. Jha, "Fault-tolerant data path synthesis based on aliasing constraints", *Proceedings of IEEE Fault Tolerant Computing Symposium,* Jun 1996.
8. D. D. Gajski, N. D. Dutt, A. Wu, and S. Lin, "High-Level Synthesis: Introduction to Chip and System Design", *Kluwer Academic Publishers*, 1992.
9. M. McFarland, A. Parker, and R. Camposano, "High-Level Synthesis of Digital Systems", *Proceedings of IEEE*, vol. 78, pp. 301-318, Feb. 1990.
10. D. Micheli, D. Ku, F. Mailhot, and T. Truong, "The Olympus Synthesis System", *IEEE Design & Test of Computers*, vol.7, pp.37-53, Oct 1990.
11. J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast Prototyping of Data Path Intensive Architectures", *IEEE Design & Test of Computers*, pp.40-51, 1991.
12. S. Hellebrand, H. J. Wunderlich and A. Hertwig, "Synthesizing Fast, On line Testable Control Units", *IEEE Design & Test of Computers*, pp.36-41, 1998.
13. M. R. Garey and D. S. Johnson, "Computers and Intractability: A guide to the theory of NP-Completeness", *W. H. Freeman and Co.*

14. R. Negrini, M. Sami, and R. Stefanelli, "Fault tolerance techniques for array structures used in supercomputing", *IEEE Computer*, vol. 19, No. 2, pp. 77-87, Feb. 1986.
15. S-Y. Kuo and W.K. Fuchs, "Efficient spare allocation in reconfigurable arrays", *IEEE Design & Test of Computers*, vol.4, No. 1, pp. 24-31, Feb. 1987.
16. W.R. Moore, "A review of fault-tolerant techniques for the enhancement of integrated circuit yield", *Proceedings of the IEEE*, vol. 74, pp. 684-698, May 1986.
17. D.P. Siewiorek and R.S. Swartz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, Burlington, MA, 1984.
18. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, 1989.
19. S. W. Director, "Optimization of parametric yield", *Proceedings of IEEE International Workshop on Defect & Fault Tolerance of VLSI Systems*, pp. 1-18, 1992.
20. V. Chiluvuri and I. Koren, "New routing and compaction strategies for yield enhancement", *Proceedings of IEEE International Workshop on Defect & Fault Tolerance of VLSI Systems*, pp. 325-334, 1992.
21. G. A. Allan, A. J. Walton, and R. J. Holwill, "A yield improvement technique for IC layout using local design rules", *IEEE Transactions on Computer Aided Design*, pp. 1355-1362, 1992.
22. A. Chandrakasan, M. Potkonjak, J. Rabaey and B. Broderson, "HYPER-LP: A System for Power Minimization Using Architectural Transformations*", Proceedings of IEEE International Conference on CAD,* pp. 300-303, 1992.
23. I.G. Harris and A. Orailoglu, "Microarchitectural synthesis of VLSI designs with high test concurrency", *Proceedings of Design Automation Conference*, 1994.
24. T. Lee, W. H. Wolf, and N. J. Jha, "Behavioral Synthesis for Easy Testability in Data Path Scheduling", *Proceedings of IEEE International Conference on CAD*, pp. 616-619, 1992.
25. F.J. Kurdahi and A.C. Parker, "REAL: A Program for REgister ALlocation", Proceedings of Design Automation Conference, pp. 210--215, Jun 1987.
26. I. Ghosh and N. Jha, High Level Test Synthesis: A Survey", Integration, the VLSI Journal, Nov. 1998.

27. J.Periaux and M.Galan, "Genetic Algorithms in Engineering and Computer Science", John Wiley, 1995.

28. L.D. Whitley & M. D.Vose, "Foundations of Genetic Algorithms", vol 3, Morgan Kaufman, 1995.