Heterogeneous BISR-approach using System Level Synthesis Flexibility [*]

Inki Hong, Miodrag Potkonjak† and Ramesh Karri‡
Synopsys Inc., 700 East Middlefield Road, Mountain View, CA 94043
†Computer Science Department, University of California, Los Angeles, California
‡Department of Electrical Engineering, Polytechnic University, Brooklyn, New York

Abstract

We propose a novel methodology for designing fault-tolerant real-time multi-processor systems-on-a-chip to achieve optimal productivity. The methodology employs the heterogeneous built-in-self-repair(BISR) based graceful degradation and yield enhancement technique as an embedded optimization engine. The technique exploits the flexibility provided in task-level scheduling and algorithm selection steps. We have developed a hardware fault model for modern superscalar processors and multi-processors which enables an efficient treatment of the synthesis and compilation goals. For the first time heterogeneous BISR is used at the task level. The key idea is to adapt scheduling and algorithm selection to the available non-faulty resources. If there is a fault in memory, the algorithms that use less memory are selected and the scheduler exploits the other abundant resource, namely the processors more vigorously to compensate for the loss of part of memory. Similarly, fault in a processor is backed up by memory. The synthesis approach minimizes the degradation in performance for single or multiple faults using simulated annealing-based algorithm selection, scheduling and assignment algorithms. On the large set of examples our adaptive algorithm selection and scheduling technique has achieved significant improvement of throughput compared to conventional nonadaptive schemes. The extensive experimental results also indicate that significant improvement in productivity can be achieved by utilizing the extra throughput gained from the technique.

[*] Revised Submission to IEEE Transactions on Reliability. A preliminary version of this paper was presented at the Asia and South Pacific Design Automation Conference, Yokohama, Japan, February 10-13, 1998.

1 Introduction

1.1 Motivation

Since the size of average application specific system and integrated systems has been doubling every year, the focus of behavioral and system level synthesis has been shifting from single task applications to multiple task applications. The increasing levels of integration and aggressive deep-submicron technologies imply a greater need for fault-tolerance. While classical fault-tolerance techniques such as duplication and triplication provide fault tolerance in a straightforward manner, the overhead is usually too high for many cost sensitive application domains such as consumer electronics and personal communication devices.

Recently, Guerra et al. [5] proposed low overhead heterogeneous BISR for single application ASICs. In this paper, we show that the application domain of heterogeneous BISR can be significantly enhanced, when inter-task relationships are explored. In some sense the technique is also a step forward in the level of heterogeneity with BISR. While the technique due to Guerra et al. was able to back up only different types of execution units using a given type of execution unit, the new approach uses synthesis flexibility to back up memory with execution units or vice versa.

To the best of our knowledge, this is the first heterogeneous system-level synthesis approach which explores task-level scheduling and algorithm selection flexibility for design of fault-tolerant systems. The experimental results clearly show the effectiveness of the new technique.

1.2 Motivational Example

In this sub-section we provide a motivational example for each of the two topics: (i) adaptive fault tolerant algorithm selection and scheduling problem and (ii) design of fault-tolerant system to optimize productivity [5] which is defined to be the ratio of the relative change in yield [20] over the relative change in area.

First, we introduce the adaptive fault tolerant algorithm selection and scheduling problem using the motivational example in Table 1. There are three homogeneous processors and 330 units of shared memory that are allocated. Given are five periodic tasks. We assume, for the sake of simplifying this example, that all the tasks have the same period. Each task has two different algorithmic choices. Each algorithm requires a certain amount of shared memory to execute so that contention due to shared memory may cause some tasks to wait if there are processors available for their execution. To switch from one task to the other, context switching time overhead occurs. The context switching time includes the overhead of loading task's code/data from permanent storage to shared memory, state update in the processor and other operating system overheads. We assume that as soon as the context switch for the new task begins, the peak memory requirement for the task should be

			Switching Time Between Tasks									
Task, Algorithm	Processing Time	Memory Requirement	A,1	A,2	B,1	B,2	C,1	C,2	D,1	D,2	E,1	E,2
A,1	25	90	-	-	11	20	20	6	20	20	21	26
A,2	50	30	-	-	15	25	15	5	18	22	8	20
B,1	50	60	25	20	-	-	20	13	10	29	18	19
B,2	70	40	28	20	-	-	15	8	15	27	22	20
C,1	20	60	8	23	20	30	-	-	25	15	9	26
C,2	15	90	10	21	17	27	-	-	23	14	9	29
D,1	100	80	15	23	23	30	6	30	-	-	19	23
D,2	70	100	10	23	18	27	9	25	-	-	18	22
E,1	25	150	22	17	10	16	25	18	22	16	-	-
E,2	35	80	3	15	12	18	27	16	20	15	-	-

Table 1. An instance of the adaptive fault tolerant algorithm selection and scheduling problem; number of allocated processors = 3; amount of allocated shared memory = 330.

reserved. If available memory is less than the memory requirement, then the context switch has to wait until the required memory is available. Therefore, a task can wait to save the memory usage if there is idle time available on a processor. Note that there is an idle period between the task E and A on a Processor 1 in Figure 1, due to this policy. We assume that a task to be scheduled next is assigned to the first available processor. If there are many processors available, then the processor with the minimum amount of context switching overhead will be selected. Typical real-time operating systems have context switching time of tens of microseconds, which is usually large compared to typical communication or execution times of one iteration of a periodic hard real-time task.

An optimal algorithm selection and schedule $S* = (A1, C2, D2, B1, E1)$ takes 95 units of time per period. It is illustrated in Figure 1 how the execution time for $S*$ is computed. When one of the three processors is down due to a permanent physical fault, S* takes 151 units of time while the other algorithm selection and schedule $S_p = (B1, D2, C2, A1, E1)$ takes 118. When 110 units of memory becomes unavailable, S* takes 153 time units while the other algorithm selection and schedule $S_m = (D2, B2, E2, A1, C1)$ takes 114 units. Instead of using the algorithm selection and schedule S*, consider using the following adaptive algorithm selection and schedule: (i) S* for fault-free situation, (ii) S_p when a processor is faulty and (iii) S_m when a part of the memory is faulty. Using this adaptive algorithm selection and schedule, the system throughput can be improved by 21.9% when a processor is faulty and by 25.5% when there is a fault in the memory subsystem. If the required schedule length is in the range [118..151] for a faulty processor situation and [114..153] for a faulty memory situation, the graceful degradation of throughput by adapting the algorithm selection and task schedule

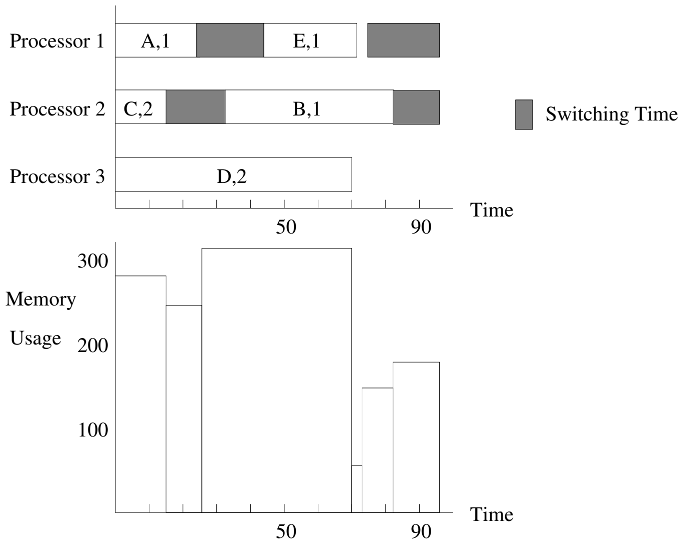

Figure 1. The execution time and memory usage of the algorithm selection and schedule $S*$

to the currently available resources renders the system operational. On the other hand, in order to tolerate a fault in processor or memory in a similar situation, the classical schemes require duplication of memory and processors.

# Tasks	Deadline	nonredundant configuration for throughput constraint			
		# Processor	# Memory	Yield	Schedule Length
10	106	5	15	0.122	104
		6	11	0.167	104
		7	9	0.185	105
		8	8	0.185	106
		12	7	0.135	104

Table 2. A motivational example for optimization of productivity.

Next, we consider a motivational example for the design of fault-tolerant system to optimize productivity. We assume that a nonredundant system with 8 processors and 8 memory units is the base system to compare with redundant systems. We assume, for the sake of simplicity, that the probability of a good processor and the probability of a good memory unit are both 0.9, and a processor and a memory unit take the same area in a chip. As an initial step, all nonredundant configurations that satisfy throughput constraint using the proposed BISR-approach as an optimization engine are found, which are provided in Table 2. # Memory in 4th column in the Table 2 is the number of memory units, where a single memory unit becomes unavailable by a single fault. Consider two redundant systems which employ 9 processors and 8 memory units and 8 processors and 9 memory units, respectively. The yield for the first redundant system becomes 0.334 and the yield for the second becomes 0.482 while both systems incur the same amount of area penalty. To show how the yield is calculated, we use the first redundant system as an example. The yield for the first redundant system is the probability that the system will work after manufacturing. There are two such cases that (i) 9 processors and 8 memory units work, or (ii) 8 processors and 8 memory units work. The probability of the first case is $0.9^9 \cdot 0.9^8$ while the probability of the second case is $9 \cdot 0.1 \cdot 0.9^8 \cdot 0.9^8$. Similarly, the yield for the second redundant system is the probability that (i) 7 or 8 processors and 9 memory units work, or (ii) 8 processors and 8 memory units work. The productivity of the second redundant system is greater by a factor of 1.44 than that of the first. The details of our methodology to find an optimal design for productivity are provided in Section 5.

1.3 Paper Organization

The rest of the paper is organized in the following way. First, in the next section, we review the related works on scheduling, algorithm selection, and behavioral and architectural level fault-tolerant techniques. In Section 3, we summarize the selected computation, hardware and fault models and provide a description of global synthesis approach for designing a fault-tolerant system to optimize productivity. Section 4 and 5 are technical core of the paper, where Section 4 formally defines the problem of the heterogeneous BISR using the adaptive algorithm selection and scheduling, presents a synthesis method for the problem and provides comprehensive experimental results while Section 5 defines the problem of designing fault-tolerant system with optimal productivity, provides a synthesis method for the problem, and present extensive experimental results. Finally, we conclude our contribution in Section 6.

2 Previous Work

Scheduling has been widely studied in many areas, such as behavioral synthesis [13], parallel processing [4], and hard real-time systems [12]. Recently, algorithm selection has been recognized as an important system-level synthesis topic, and several approaches have been proposed [15].

Behavioral synthesis provides the mechanism for design space exploration so that a variety of design goals can be optimized [13] [2]. Much of the behavioral synthesis research have targeted the optimization of area, speed (throughput), and more recently power and testability.

Relatively little work has been reported on behavioral level synthesis techniques for fault tolerant design. Raghavendra and Lursinsap [18] concentrated on designs with self-recovery from transient faults using micro roll-back and checkpoint insertion. Karri and Orailoglu [7] [8] [14] develop a spectrum of behavioral techniques which minimize the fault-tolerance overhead in application specific designs. Recently, Iyer et al. [6] develop a high-level synthesis approach to optimize manufacturability using redundant interconnects. Guerra et al. [5] develop a heterogeneous BISR behavioral synthesis system, which explores flexibility in scheduling a single task during high level synthesis so that the resulting design is operational when there are no more than k faulty units.

For a review of known fault tolerance techniques, we refer to [19] which provides comprehensive lists of relevant references.

3 Preliminaries

We outline the relevant preliminaries in this section. In particular, we describe the selected hardware, fault, and computation models and outline a global design flow of the fault-tolerant system.

3.1 Hardware and Fault Model

The hardware model being considered is shown in Figure 2. The processors, interconnect and memory shown in the figure are placed in a single-chip multi-processor platform. We assume that the area of a chip is the sum of the areas of processors and memory units, which accounts for the significant portion of the overall area of the chip [1]. insignificant. The ratio of the area between processor and memory is assumed to be provided. The increase in the area of a chip by the addition of a processor or memory is assumed to be just the area of the added component regardless of how many processors and memory units are in the current configuration.

It is virtually impossible to replace the faulty processors, memory, and interconnects in the single-chip system. Faults can occur in either a processor, a memory system, or an interconnect. We consider only the permanent physical faults. The fault in interconnect can be regarded as the fault in a processor. A faulty interconnect prevents its corresponding processor from receiving data from the memory system, as shown in P_2. A fault in a memory system causes a part or all of the memory unusable. A fault in a processor causes a single processor down, as shown in P_1. The identification of the faulty parts can be done by either an offline testing before packaging or an online testing. The controller of the system is reconfigured upon detection of a fault.

3.2 Computational Model

The syntax of a targeted computation is defined as a hierarchical data-control flow graph (CDFG) [17]. The CDFG represents the computation as a flow graph, with nodes, data edges, and control edges. The semantics underlying the syntax of the CDFG format is homogeneous synchronous data flow (HSDF) [11], which assumes a semi-infinite or a very long input stream of data arriving and being processed at periodic intervals, imposed by the nature of the specified applications. The HSDF model is well suited for specification of single task computations in numerous application domains such as digital signal processing, video and image processing,

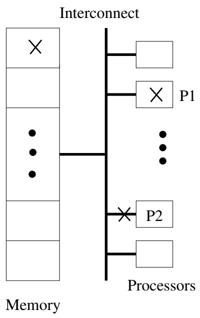

Figure 2. The hardware and fault model; x indicates a fault.

broadband and wireless communications, control, information and coding theory, and multimedia.

We assume that there are no data, control, or timing dependences among tasks. If there are dependences the tasks are merged in a new composite single task. We also assume that all tasks are periodic with the same period. This implies that all the tasks should be completed to start the next period. Again note that with no loss of generality using the least common multiple theorem [10] a set of tasks with arbitrary periods, can be transformed into the equivalent set of tasks with the same period in polynomial time. Since context switching overhead is usually high, we assume no task preemption. To execute a task, there must exist a unassigned processor and a required amount of unassigned operating memory for loading the task's code and data from the permanent secondary storage. Partly due to the loading of code and data, the context switch time is incurred before a new task starts its execution. The context switch times between tasks are described by a matrix between tasks.

4 Heterogeneous BISR Using Adaptive Algorithm Selection and Scheduling

In this section, we formulate the adaptive algorithm selection and scheduling problem and provide the computational complexity of the problem. Next, we propose synthesis method and present extensive experimental results.

4.1 Problem Formulation and Complexity

Our goal is to achieve the graceful degradation of throughput in the presence of faults in the system using the adaptive algorithm selection and scheduling technique. The adaptive algorithm selection and scheduling problem can be formally stated as follows:

$Problem$: Throughput Optimization Using Adaptive Algorithm Selection and Scheduling

$Instance$: Given are N tasks with A algorithmic choices that have processor and memory requirements, P homogeneous processors, M memory units, context switch time matrix CST and a period D.

$Question$: Are there a selection of algorithms and a schedule of tasks such that in the presence of any subset of size at most K1 faulty processors and any subset of size at most K2 faulty memory units the resulting schedule length is at most D?

$Proof$: We prove that the problem is NP-complete by using Karp's polynomial transformation technique from the traveling salesman problem which is well known to be NP-complete [3]. Let us consider a special case of the original problem formulation. Suppose there is a single processor with unlimited memory units. There are N tasks with zero execution times. It takes time only when a processor switches from one task to the other, which is described as the context switch time matrix CST. We assume that the CST matrix is symmetric. Let each task denote a city and the CST matrix denote a distance matrix between the cities. The reduced problem is the traveling salesman problem. Since a special case of the problem is NP-complete, the original problem is NP-complete.

4.2 Synthesis Approach

Since the computational complexity of the algorithm selection and scheduling problem forbids an exact or optimal solution, a general combinatorial optimization technique known as simulated annealing(SA) [9] has been used to obtain nearly optimal throughput given the faults in processors and memory units. A general combinatorial optimization technique known as simulated annealing(SA) has been employed as one of the heuristics. The technique can be applied to all the scenarios considered with some straightforward changes. The SA algorithm is described in Figure 3.

The actual implementation details are presented for each of the following areas; the cost function, the neighbor solution generation, the temperature update function, the equilibrium criterion and the termination criterion. Firstly, the actual schedule length for the given schedule has been used as the cost function. Secondly, the neighbor solution is generated by the random choice of two perturbations; (i) the interchange of two tasks in the schedule and (ii) the change of algorithms of a task in the schedule. Thirdly, the temperature is updated by the function $T_{new} = \alpha(T_{old}) * T_{old}$. α is a function of the temperature. For the temperature $T > 200.0$, α is chosen to be 0.1 so that in high temperature region where every new state has very high chance of acceptance, the temperature reduction occurs very rapidly. For $1.0 < T \leq 200.0$, α is set to 0.95 so that the optimization process explores this promising region more slowly. For $T \leq 1.0$, α is set to 0.8 so that T is relatively quickly reduced to firmly converge to a local minimum. The initial temperature is set to 4,000,000. Fourthly, the equilibrium criterion is specified by the number of iterations of the inner loop. The number of iterations of the inner loop is set to 20 times of the number of tasks. Lastly, the termination criterion is given by the temperature. If

Generate a starting solution S.
Set the best solution $S* = S$.
Determine a starting temperature T.
while not yet frozen,
  while not yet at equilibrium for the current temperature,
    Choose a random neighbor S' of the current solution.
    if $cost(S') < cost(S)$
      Set $S = S'$
      if $cost(S) < cost(S*)$
        $S* = S'$
    else
      Generate a random number r uniformly from [0, 1].
      $diff = cost(S') - cost(S)$;
      if $r <$ exp($- \, diff \, / \, T$)
        $S = S'$
  Update the temperature T.
Return the best solution $S*$.

Figure 3. The pseudo-code for simulated annealing algorithm.

the temperature falls below 0.1, the simulated annealing algorithm stops.

4.3 Experimental Results

We have generated a set of random examples by varying the number of tasks to show the effects of the problem size on the performance. We have tried 10, 20, 30, 40 and 50 tasks. The number of different algorithmic choices for each task is randomly chosen between 2 and 5. The number of processors and memories allocated are 5 and 1000 respectively. The processing times for tasks are chosen randomly between 10 and 100 while the memory requirements are randomly distributed between 10 and 350. The context switching times take values between 5 and 30. Each fault in processors is assumed to cause a processor unavailable. When a fault in memory occurs, we assume that 200 units of memory becomes unavailable. The Table 3 illustrates the effectiveness of the adaptive scheduling approach. The first number in each item represents the schedule length for the nonadaptive schedule which is obtained under the no fault assumption while the second number represents the schedule length for the adaptive schedule which is obtained under the appropriate fault situation. The third number provides the percentage improvement for the adaptive schedule from the nonadaptive one. The CPU times reported are the running times on SUN Sparc 4. The adaptive scheduling approach achieves an average improvement of 13.1% compared to the nonadaptive approach.

Number of Tasks	Synthesis Techniques	(The number of processor faults,The number of memory faults)									CPU time
		(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)	
10	initial	115	145	197	131	150	197	161	185	197	
	adaptive	115	132	148	123	123	156	142	152	169	0.66 min
	improvement (%)	0.0	9.0	24.9	6.1	18.0	20.8	11.8	17.8	14.2	
20	initial	217	279	340	258	293	340	333	340	363	
	adaptive	217	234	266	235	260	294	303	304	327	2.8 min
	improvement (%)	0.0	16.2	21.8	9.0	11.3	13.5	9.0	10.6	10.0	
30	initial	276	297	383	344	344	389	434	434	438	
	adaptive	276	278	319	321	328	341	392	392	412	7.5 min
	improvement (%)	0.0	6.4	17.7	6.7	4.7	12.4	9.7	9.7	6.0	
40	initial	395	475	673	507	516	673	642	642	715	
	adaptive	395	427	473	435	459	520	551	551	592	13.8 min
	improvement (%)	0.0	10.1	29.7	14.2	11.0	22.7	11.0	11.0	17.2	
50	initial	473	564	742	560	584	744	740	740	810	
	adaptive	473	478	547	541	550	593	676	692	702	18.6 min
	improvement (%)	0.0	15.3	26.3	3.4	5.8	20.3	8.7	6.5	13.3	

Table 3. Experimental results of adaptive algorithm selection and scheduling for random examples.

Tasks	Synthesis Techniques	(The number of processor faults,The number of memory faults)								
		(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)
Set 1	initial	1156	1769	2299	1276	1769	2299	1744	2009	2352
	adaptive	1156	1452	2096	1187	1472	2115	1494	1587	2136
	improvement (%)	0.0	18.0	8.2	7.0	16.8	8.0	14.3	21.0	9.2
Set 2	initial	1222	1873	2575	1392	1873	2575	1731	1885	2604
	adaptive	1222	1570	2242	1305	1567	2249	1631	1697	2269
	improvement (%)	0.0	16.2	13.0	6.2	17.3	12.7	5.8	10.0	12.9
Set 3	initial	693	693	858	830	830	868	973	973	973
	adaptive	693	693	776	754	754	808	955	956	956
	improvement (%)	0.0	0.0	9.6	9.2	9.2	6.9	1.8	1.7	1.7
Set 4	initial	1059	1739	2105	1107	1739	2115	1524	1789	2170
	adaptive	1059	1403	1885	1084	1403	1885	1321	1503	1900
	improvement (%)	0.0	19.3	10.5	2.1	19.3	10.9	13.3	16.0	12.4

Table 4. Experimental results of adaptive scheduling for real-life examples without algorithm selection.

We have constructed 4 different sets of 12 tasks from the 16 real-life tasks described in [16]. The tasks are the following: two GE controllers, two Honda controllers, a wavelet filter, four audio filters, a 8×8 discrete cosine transform, an NEC digital-to-analog converter, two components of modems, a LMS audio formatter, an echo canceler, and a linear controller for automotive motion control. For each task, we have used the number of operations and the area of implementation as the running time and the memory requirement, respectively. The context switching times are approximated such that they are proportional to memory requirement, the switching between similar tasks costs little overhead and the switching between different tasks requires a significant amount of overhead. The context switching costs are distributed between 10 and 173. The number of processors and memory units allocated are 4 and 160, respectively. Each fault in memory causes 40 units of memory unusable. Algorithm selection is not considered since we assume that only one algorithmic choice is available for each task. Table 4 presents the experimental results and clearly shows the effectiveness of our technique. The adaptive scheduling approach achieves an average improvement of 10.6% compared to the nonadaptive approach.

5 Design of Fault-Tolerant System for Optimizing Productivity

5.1 Global Design Flow

Our goal is to design a fault-tolerant system with optimal productivity using the heterogeneous BISR-approach described in the previous section. The first stage of the design flow is to design a nonredundant system that satisfies throughput requirement which is assumed to be given. The second stage is to find all nonredundant system configurations that can satisfy the throughput requirement using the heterogeneous BISR-approach which uses the task-level scheduling and algorithm selection flexibility. The third stage is to find an optimal redundant configuration that maximizes productivity by considering both yield enhancement and area penalty, which is described in next sub-section.

5.2 Synthesis Method

The yield and area for a given system configuration must be computed to get the productivity measure. Suppose the given system employs n_p processors and n_m memory units. The probability of a good processor is P_p and the probability of a good memory unit is P_m. All the n nonredundant system configurations are (p_1, m_1), (p_2, m_2), ..., (p_n, m_n), where p_i and m_i are the number of processors and memory units, respectively and $p_i \leq p_{i+1}$. We observe that if $p_i \leq p_{i+1}$, $m_i \geq m_{i+1}$. We also observe that if (p_i, m_i) is a nonredundant system configuration, then (p_j, m_j) is a redundant system configuration when $p_j \geq p_i$ and $m_j \geq m_i$. The yield $Y(n_p, n_m)$ is computed using the equation provided in Figure 4. The equation basically calculates the sum of the probabilities for all the possible functional configurations after manufacturing for a given number of processors and memory units. Each product term represents the probability that there are at least p_k functional processors and m_k memory units for $k = 1..n$. If n_p is smaller than p_k, the product term is 0. Area for a configuration is computed based on an area model in Section 3.

The algorithm to maximize productivity is based on enumeration, where, from every configuration, new configurations are generated by adding a processor or memory unit, and the further enumeration from the new configuration stops when the productivity of the new configuration is smaller than that of the previous configuration. The algorithm finds an optimal solution since the relative improvement in yield by addition of a

$$
\begin{aligned}
Y(n_p, n_m) = {} & \sum_{i=0}^{n_p-p_n} \binom{n_p}{i} P_p^{n_p-i}(1-P_p)^i \left(\sum_{j=0}^{n_m-m_n} \binom{n_m}{j} P_m^{n_m-j}(1-P_m)^j\right) + \\
& \sum_{i=max(n_p-p_n+1,0)}^{n_p-p_{n-1}} \binom{n_p}{i} P_p^{n_p-i}(1-P_p)^i \left(\sum_{j=0}^{n_m-m_{n-1}} \binom{n_m}{j} P_m^{n_m-j}(1-P_m)^j\right) + \\
& \ldots + \sum_{i=max(n_p-p_2+1,0)}^{n_p-p_1} \binom{n_p}{i} P_p^{n_p-i}(1-P_p)^i \left(\sum_{j=0}^{n_m-m_1} \binom{n_m}{j} P_m^{n_m-j}(1-P_m)^j\right)
\end{aligned}
$$

Figure 4. The yield equation for the system of n_p processors and n_m memory units; P_p - the probability of a good processor; P_m - the probability of a good memory unit; all the n nonredundant system configurations are (p_1, m_1), (p_2, m_2), $\cdots$, (p_n, m_n), where p_i and m_i are the number of processors and memory units, respectively and $p_i \leq p_{i+1}$.

processor or memory decreases as the number of processors or memory units increases, and the relative increase in area by addition of a processor or memory stays constant regardless of how many processors and memory units are used in the current configuration.

5.3 Experimental Results

We have used the same random examples used in the previous section with a change that 8 processors and 800 memory units are used, where 100 memory units are affected by a single fault in memory. The number of memory units reported in the paper is in terms of 100 memory units. Using the adaptive algorithm selection and scheduling, all the nonredundant system configurations have been identified and provided in Table 5. The design of fault-tolerant system with optimal productivity has been performed under various design parameters such as the probability of a good processor, the probability of a good memory unit, the area ratio of processor and memory, as suggested by Stapper et al. [20]. The results of redundant system with optimal productivity are provided in Tables 6 and 7 under the various range of values for the probabilities of a good processor and memory unit, where the first, second, third, fourth and fifth numbers in each entry represent the initial yield, the final yield, productivity improvement factor (PIF), the number of processors and memory units of the optimal design, respectively. The PIF is defined as the relative yield increase divided by the relative area increase. The relative yield increase is defined as the final yield minus the original yield divided by the original yield. The relative area increase is defined as the new area minus the old area divided by the old area. The running times for the entire design process ranged between several minutes and about an hour on Sun Sparc 4 depending on the number of tasks. The experimental results indicate significant improvement in productivity and yield by the

# Tasks	Deadline	# Processor	# Memory	Schedule Length
10	106	5	15	104
		6	11	104
		7	9	105
		8	8	106
		12	7	104
20	183	6	9	182
		7	9	182
		8	8	183
		9	6	173
		14	5	181
30	256	6	10	253
		7	9	247
		8	8	256
		9	7	242
		10	6	255
40	325	6	15	321
		7	9	316
		8	8	325
		9	7	315
50	379	7	13	378
		8	8	379
		10	7	378

Table 5. All nonredundant configurations that satisfy the throughput requirement using the proposed heterogeneous BISR-approach.

method.

6 Conclusion

For the first time we proposed a design methodology for the design of fault-tolerant real-time system to achieve optimal productivity by providing the novel approach for heterogeneous BISR based graceful degradation and yield enhancement using task-level algorithm selection and scheduling flexibility based on our hardware fault model for modern superscalar processors and multi-processors. The experimental results indicate that significant improvement in productivity can be achieved by the method. The adaptive algorithm selection and scheduling, as an optimization engine of the design methodology, allows the graceful degradation of throughput for single or multiple faults using simulated annealing-based algorithm. On the large set of examples our adaptive algorithm selection and scheduling technique has achieved significant improvement of throughput compared to conventional nonadaptive scheme.

# Tasks	P_m	P_p														
		0.75					0.85					0.95				
		IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m
10	0.75	0.010	0.628	164.5	10	12	0.027	0.690	77.8	9	12	0.066	0.603	43.1	8	11
	0.85	0.027	0.708	79.9	10	11	0.074	0.633	40.1	9	10	0.181	0.462	24.9	8	9
	0.95	0.066	0.468	48.4	9	9	0.181	0.496	27.9	8	9	0.440	0.792	12.8	8	9
20	0.75	0.010	0.739	193.8	11	11	0.027	0.397	108.4	10	8	0.066	0.458	94.2	9	8
	0.85	0.027	0.397	108.4	8	10	0.074	0.562	52.6	8	10	0.181	0.645	41.1	9	8
	0.95	0.066	0.458	94.2	8	9	0.181	0.645	41.1	8	9	0.440	0.825	14.0	8	9
30	0.75	0.010	0.697	182.8	11	11	0.027	0.632	88.7	11	9	0.066	0.529	55.8	10	8
	0.85	0.027	0.632	88.7	9	11	0.074	0.684	43.8	10	9	0.181	0.496	27.9	9	8
	0.95	0.066	0.529	55.8	8	10	0.181	0.496	27.9	8	9	0.440	0.792	12.8	8	9
40	0.75	0.010	0.665	174.4	11	11	0.027	0.599	83.8	10	10	0.066	0.473	49.0	9	9
	0.85	0.027	0.599	83.8	10	10	0.074	0.480	43.7	9	9	0.181	0.496	27.9	9	8
	0.95	0.066	0.473	49.0	9	9	0.181	0.496	27.9	8	9	0.440	0.792	12.8	8	9
50	0.75	0.010	0.668	150.1	12	11	0.027	0.748	70.5	11	11	0.066	0.669	36.3	10	10
	0.85	0.027	0.742	69.8	12	10	0.074	0.543	33.7	10	9	0.181	0.398	19.2	8	9
	0.95	0.066	0.528	37.1	11	8	0.181	0.398	19.2	9	8	0.440	0.863	7.7	9	9

(a) when the area ratio between processor and memory is 1

# Tasks	P_m	P_p														
		0.75					0.85					0.95				
		IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m
10	0.75	0.010	0.677	199.8	9	14	0.027	0.517	107.7	8	12	0.066	0.603	64.6	8	11
	0.85	0.027	0.478	99.2	8	12	0.074	0.433	58.0	8	10	0.181	0.462	37.4	8	9
	0.95	0.066	0.261	70.4	8	9	0.181	0.496	41.8	8	9	0.440	0.792	19.2	8	9
20	0.75	0.010	0.460	269.1	8	12	0.027	0.633	133.3	8	12	0.066	0.458	70.7	9	8
	0.85	0.027	0.397	162.6	8	10	0.074	0.562	78.9	8	10	0.181	0.474	39.0	8	9
	0.95	0.066	0.458	141.3	8	9	0.181	0.645	61.7	8	9	0.440	0.825	21.0	8	9
30	0.75	0.010	0.486	228.0	8	13	0.027	0.572	119.8	8	12	0.066	0.611	65.5	8	11
	0.85	0.027	0.454	125.2	8	11	0.074	0.480	65.5	8	10	0.181	0.462	37.4	8	9
	0.95	0.066	0.529	83.7	8	10	0.181	0.496	41.8	8	9	0.440	0.792	19.2	8	9
40	0.75	0.010	0.630	185.7	10	12	0.027	0.369	100.4	8	11	0.066	0.600	64.3	8	11
	0.85	0.027	0.227	88.0	8	10	0.074	0.433	58.0	8	10	0.181	0.462	37.4	8	9
	0.95	0.066	0.261	70.4	8	9	0.181	0.496	41.8	8	9	0.440	0.792	19.2	8	9
50	0.75	0.010	0.750	147.6	12	12	0.027	0.711	75.2	10	12	0.066	0.349	51.0	8	10
	0.85	0.027	0.611	64.2	11	10	0.074	0.492	33.7	9	10	0.181	0.398	28.8	8	9
	0.95	0.066	0.675	31.4	11	9	0.181	0.557	16.6	9	9	0.440	0.616	9.6	8	9

(b) when the area ratio between processor and memory is 2

Table 6. Redundant systems with optimal productivity when the probabilities of a good processor and memory unit is low; IY - initial yield; FY - final yield; PIF - improvement factor in productivity; n_p - # processors in the final design; n_m - # memory units in the final design; Initially, 8 processors and 8 memory units are used.

# Tasks	P_m	P_p														
		0.98					0.99					0.995				
		IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m
10	0.98	0.724	0.955	5.1	8	9	0.785	0.973	3.8	8	9	0.817	0.980	3.2	8	9
	0.99	0.785	0.975	3.9	8	9	0.851	0.988	2.6	8	9	0.886	0.993	1.9	8	9
	0.995	0.817	0.983	3.2	8	9	0.886	0.993	1.9	8	9	0.923	0.997	1.3	8	9
20	0.98	0.724	0.964	5.3	8	9	0.785	0.984	4.1	9	8	0.817	0.992	3.4	9	8
	0.99	0.785	0.984	4.1	8	9	0.851	0.990	2.6	8	9	0.886	0.996	2.0	9	8
	0.995	0.817	0.992	3.4	8	9	0.886	0.996	2.0	8	9	0.923	0.997	1.3	8	9
30	0.98	0.724	0.955	5.1	8	9	0.785	0.975	3.9	9	8	0.817	0.983	3.2	9	8
	0.99	0.785	0.975	3.9	8	9	0.851	0.988	2.6	8	9	0.886	0.993	1.9	9	8
	0.995	0.817	0.983	3.2	8	9	0.886	0.993	1.9	8	9	0.923	0.997	1.3	9	8
40	0.98	0.724	0.955	5.1	8	9	0.785	0.975	3.9	9	8	0.817	0.983	3.2	9	8
	0.99	0.785	0.975	3.9	8	9	0.851	0.988	2.6	8	9	0.886	0.993	1.9	9	8
	0.995	0.817	0.983	3.2	8	9	0.886	0.993	1.9	8	9	0.923	0.997	1.3	9	8
50	0.98	0.724	0.974	2.8	9	9	0.785	0.911	2.6	8	9	0.817	0.948	2.6	8	9
	0.99	0.785	0.911	2.6	9	8	0.851	0.993	1.3	9	9	0.886	0.957	1.3	8	9
	0.995	0.817	0.948	2.6	9	8	0.886	0.957	1.3	9	8	0.923	0.998	0.7	9	9

(a) when the area ratio between processor and memory is 1

# Tasks	P_m	P_p														
		0.98					0.99					0.995				
		IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m	IY	FY	PIF	n_p	n_m
10	0.98	0.724	0.955	7.7	8	9	0.785	0.973	5.7	8	9	0.817	0.980	4.8	8	9
	0.99	0.785	0.975	5.8	8	9	0.851	0.988	3.8	8	9	0.886	0.993	2.9	8	9
	0.995	0.817	0.983	4.9	8	9	0.886	0.993	2.9	8	9	0.923	0.997	1.9	8	9
20	0.98	0.724	0.964	5.3	8	9	0.785	0.984	4.1	9	8	0.817	0.992	3.4	9	8
	0.99	0.785	0.984	4.1	8	9	0.851	0.990	2.6	8	9	0.886	0.996	2.0	9	8
	0.995	0.817	0.992	3.4	8	9	0.886	0.996	2.0	8	9	0.923	0.997	1.3	8	9
30	0.98	0.724	0.955	7.7	8	9	0.785	0.973	5.7	8	9	0.817	0.980	4.8	8	9
	0.99	0.785	0.975	5.8	8	9	0.851	0.988	3.8	8	9	0.886	0.993	2.9	8	9
	0.995	0.817	0.983	4.9	8	9	0.886	0.993	2.9	8	9	0.923	0.997	1.9	8	9
40	0.98	0.724	0.955	7.7	8	9	0.785	0.973	5.7	8	9	0.817	0.980	4.8	8	9
	0.99	0.785	0.975	5.8	8	9	0.851	0.988	3.8	8	9	0.886	0.993	2.9	8	9
	0.995	0.817	0.983	4.9	8	9	0.886	0.993	2.9	8	9	0.923	0.997	1.9	8	9
50	0.98	0.724	0.840	3.8	8	9	0.785	0.911	3.8	8	9	0.817	0.948	3.8	8	9
	0.99	0.785	0.983	2.0	9	9	0.851	0.920	1.9	8	9	0.886	0.957	1.9	8	9
	0.995	0.817	0.948	1.9	9	8	0.886	0.996	1.0	9	9	0.923	0.960	1.0	8	9

(b) when the area ratio between processor and memory is 2

Table 7. Redundant systems with optimal productivity when the probabilities of a good processor and memory unit is high; IY - initial yield; FY - final yield; PIF - improvement factor in productivity; n_p - # processors in the final design; n_m - # memory units in the final design; Initially, 8 processors and 8 memory units are used.

References

[1] T.D. Burd and R.W. Brodersen,“Processor design for portable systems”, Journal of VLSI Signal Processing, Vol. 13, No 2-3, pp. 203-221, 1996.

[2] G. De Micheli, Synthesis and Optimization of Digital Circuits, McGraw Hill, New York, NY, 1994.

[3] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York, 1979.

[4] M. J. Gonzales,“Deterministic Processor Scheduling”, ACM Computing Surveys, Vol. 9, No. 3, pp. 173-204, 1977.

[5] L. Guerra, M. Potkonjak, J. Rabaey, “High Level Synthesis for Efficient Built-In Self Repair”, International Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 41-48, 1993.

[6] B. Iyer, R. Karri, I. Koren, “Phantom Redundancy: A High-Level Synthesis Approach for Manufacturability,” International Conference on Computer-Aided Design, pp. 658-661, 1995.

[7] R. Karri, A. Orailoglu, “Transformation-Based High-Level Synthesis of Fault-Tolerant ASICs,” Design Automation Conference, pp. 662-665, 1992.

[8] R. Karri, A. Orailoglu, “High-Level Synthesis of Fault-Secure Microarchitectures,” Design Automation Conference, pp. 429-433, 1993.

[9] S. Kirkpatrick, C. Gelatt, M. Vecchi, “Optimization by Simulated Annealing,” Science, Vol. 220, No. 4598, pp. 671-680, 1983.

[10] E.L. Lawler, C.U. Martel, “Scheduling periodically occurring tasks on multiple processors,” Information Processing Letters, Vol. 12, No. 1, pp. 9-12, 1982.

[11] E. A. Lee and D. G. Messerschmitt, “Synchronous dataflow” , Proceedings of the IEEE, Vol. 75, No. 9, pp. 1235-1245, 1987.

[12] C. L. Liu, J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment”, Journal of ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[13] 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.

[14] A. Orailoglu, R. Karri, “Automatic Synthesis of Self-Recovering VLSI Systems,” IEEE Trans. on Computers, Vol. 45, No. 2, pp. 131-142, 1996.

[15] M. Potkonjak, J. Rabaey, “Algorithm Selection: A Quantitative Computation-Intensive Optimization Approach”, International Conference on Computer-Aided Design, pp. 90-95, 1994.

[16] M. Potkonjak, W. Wolf: “Cost Optimization in ASIC Implementation of Periodic Hard-Real Time Systems using Behavioral Synthesis Techniques”, International Conference on Computer-Aided Design, pp. 446-451, 1995.

[17] J. Rabaey, et al., “Fast prototyping of data path intensive architecture”, IEEE Design and Test, Vol. 8, No. 2, pp. 40-51, 1991.

[18] V. Raghavendra, C. Lursinsap, “Automated Micro-Roll-Back Self Recovery Synthesis,” Design Automation Conference, pp. 385-390, 1991.

[19] D.P. Siewiorek, R.S. Swartz, Reliable Computer Systems: Design and Evaluation, 2nd edition, Digital Press, Burlington, MA, 1992.

[20] C.H. Stapper, J.A. Patrick, R.J. Rosner, “Yield Model for ASIC and Processor Chips”, International Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 136-143, 1993.