Heterogeneous BISR-approach using System Level Synthesis Flexibility

Inki Hong, Miodrag Potkonjak

Computer Science Department
University of California
Los Angeles, CA 90095-1596 USA

Ramesh Karri

Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003 USA

Abstract— We propose a novel methodology for designing fault-tolerant real-time system to achieve optimal productivity on a single-chip multi-processor platform using the heterogeneous built-in-self-repair(BISR) based graceful degradation and yield enhancement technique as an embedded optimization engine which exploits task-level scheduling and algorithm selection flexibility. We also developed a hardware fault model for modern superscalar processors and multi-processors which enables an efficient treatment of the synthesis and compilation goals.

I. INTRODUCTION

Since the size of average application specific system has been doubling every year and the size of integrated systems has been growing at just slightly low rate, 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 emphasized need for fault-tolerance. While classical fault-tolerance techniques 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. [2] proposed low overhead heterogeneous BISR for single task ASICs. In this paper, we show that the application domain of heterogeneous BISR can be significantly enhanced, when intertask relationships are explored. In some sense the technique is also a step forward in the level of heterogeneity with BISR. While the Guerra's technique 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.

We consider a motivational example for the following two topics: (i) the adaptive fault tolerant algorithm selection and scheduling problem and (ii) the design of fault-tolerant system to optimize productivity [2] which is defined to be the ratio of the relative change in yield [12] over the relative change in area.

First, we introduce the adaptive fault tolerant algorithm selection and scheduling problem. There are 3 homogeneous processors and 330 units of allocated shared memory. Given are 5 periodic tasks. Each task has two different algorithmic choices. Each algorithm requires a certain amount of shared memory to execute so that contentions to shared memory may cause some tasks to wait if there are processors available for their execution. To switch from a 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. Typical real-time operating systems have context switching time of several to 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 3 processors is down due to a permanent physical fault, S^* takes 151 units of time while the other algorithm selection and schedule $S_p = (B1, C2, D2, A1, E1)$ takes 126. When 110 units of memory becomes unavailable, S^* takes 153 time units while the other algorithm selection and schedule $S_m = (D2, B2, E2, A1, C2)$ takes 117 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 16.6% when a processor is faulty and by 23.5% when there is a fault in the memory subsystem. If the required schedule length is in the range [126-151] for a faulty processor situation and [117-153] for a faulty memory situation, the graceful degradation of throughput by adapting the algorithm selection and task schedule 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.

Next, we consider a motivational example for the design

TABLE I

AN EXAMPLE FOR THE ADAPTIVE FAULT TOLERANT ALGORITHM SELECTION AND SCHEDULING PROBLEM; 3 PROCESSORS AND 330 UNITS OF ALLOCATED SHARED MEMORY.

T.A	P_T	M_R	Switching Time Matrix Between Tasks									
			A1	A2	B1	B2	C1	C2	D1	D2	E1	E2
A1	25	90	-	-	11	20	20	6	20	20	6	26
A2	50	30	-	-	15	25	15	5	18	22	8	20
B1	50	60	25	20	-	-	20	9	10	29	18	19
B2	70	40	28	20	-	-	15	8	15	27	22	20
C1	20	60	8	23	20	30	-	-	25	15	9	26
C2	15	90	10	21	17	27	-	-	23	14	9	29
D1	100	80	15	23	23	30	6	30	-	-	19	23
D2	70	100	10	23	18	27	9	25	-	-	18	22
E1	25	150	5	17	10	16	25	18	22	16	-	-
E2	35	80	3	15	12	18	27	16	20	15	-	-

Fig. 1. The execution time and memory usage of the algorithm selection and schedule S^*

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 the same as 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 II. The 4th column in the Table II 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. 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 V.

The rest of the paper is organized in the following way. First, in the next section, we review the related works. In Section III, we summarize the selected computation, hardware and fault models. Section IV and V are techni-

TABLE II

A MOTIVATIONAL EXAMPLE FOR OPTIMIZATION OF PRODUCTIVITY.

Tasks	Deadline	nonredundant configuration for throughput constraint			
		Processors	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

cal core of the paper, where Section IV 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 V 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 VI.

II. PREVIOUS WORK

Recently, algorithm selection has been recognized as an important system-level synthesis topic, and several approaches have been proposed [9]. Karri and Orailoglu [4] [8] develop a spectrum of behavioral techniques which minimize the fault-tolerance overhead in application specific designs. Recently, Iyer et al. [3] develop a high-level synthesis approach to optimize manufacturability using redundant interconnects. Guerra et al. [2] 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.

III. PRELIMINARIES

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 because the area for interconnect is 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

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

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 hard or 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 is done by the testing before packaging. The controller of the system is reconfigured upon detection of a fault.

The syntax of a targeted computation is defined as a hierarchical data-control flow graph (CDFG) [11]. The semantics underlying the syntax of the CDFG format is homogeneous synchronous data flow (HSDF) [7].

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. Again note that with no loss of generality using the least common multiple theorem [6] 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 required before a new task starts its execution. The context switch times between tasks are described by a matrix between tasks.

IV. HETEROGENEOUS BISR USING ADAPTIVE ALGORITHM SELECTION AND SCHEDULING

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 $K1$ faulty processors and $K2$ faulty memory units the resulting schedule length is at most D ?

We proved 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 [1].

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) [5] with standard geometric cooling has been used to optimize the throughput given the faults in processors and memory units.

We 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 III illustrates the effectiveness of the adaptive scheduling approach. The first number in each item represents the schedule length for the nonadaptive schedule which is gotten under the no fault assumption while the second number represents the schedule length for the adaptive schedule which is gotten 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 has achieved the average 13.1% improvement from the non-adaptive approach.

We have constructed 4 different sets of 12 tasks from the 16 real-life tasks described in [10]. 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 canceller, 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

TABLE III
EXPERIMENTAL RESULTS OF ADAPTIVE ALGORITHM SELECTION AND SCHEDULING FOR RANDOM EXAMPLES.

n		(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)
10	initial	115	145	197	131	150	197	161	185	197
	adaptive	115	132	148	123	123	156	142	152	169
	imp (%)	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
	imp (%)	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
	imp (%)	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
	imp (%)	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
	imp (%)	0.0	15.3	26.3	3.4	5.8	20.3	8.7	6.5	13.3

TABLE IV
EXPERIMENTAL RESULTS OF ADAPTIVE SCHEDULING FOR REAL-LIFE EXAMPLES WITHOUT ALGORITHM SELECTION.

n		(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
	imp (%)	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
	imp (%)	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
	imp (%)	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
	imp (%)	0.0	19.3	10.5	2.1	19.3	10.9	13.3	16.0	12.4

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 IV presents the experimental results and clearly shows the effectiveness of our technique. The adaptive scheduling approach has achieved the average 10.6% improvement from the nonadaptive approach.

V. DESIGN OF FAULT-TOLERANT SYSTEM FOR OPTIMIZING PRODUCTIVITY

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 config-

uration that maximizes productivity by considering both yield enhancement and area penalty.

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 3. The area is computed based on the area model in Section III.

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 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

TABLE VI

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.

n	P_m	P_p									
		0.75					0.85				
		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
	0.85	0.027	0.708	79.9	10	11	0.074	0.633	40.1	9	10
	0.95	0.066	0.468	48.4	9	9	0.181	0.496	27.9	8	9
20	0.75	0.010	0.739	193.8	11	11	0.027	0.397	108.4	10	8
	0.85	0.027	0.397	108.4	8	10	0.074	0.562	52.6	8	10
	0.95	0.066	0.458	94.2	8	9	0.181	0.645	41.1	8	9
30	0.75	0.010	0.697	182.8	11	11	0.027	0.632	88.7	11	9
	0.85	0.027	0.632	88.7	9	11	0.074	0.684	43.8	10	9
	0.95	0.066	0.529	55.8	8	10	0.181	0.496	27.9	8	9
40	0.75	0.010	0.665	174.4	11	11	0.027	0.599	83.8	10	10
	0.85	0.027	0.599	83.8	10	10	0.074	0.480	43.7	9	9
	0.95	0.066	0.473	49.0	9	9	0.181	0.496	27.9	8	9
50	0.75	0.010	0.668	150.1	12	11	0.027	0.748	70.5	11	11
	0.85	0.027	0.742	69.8	12	10	0.074	0.543	33.7	10	9
	0.95	0.066	0.528	37.1	11	8	0.181	0.398	19.2	9	8

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

# Tasks	P_m	P_p									
		0.75					0.85				
		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
	0.85	0.027	0.478	99.2	8	12	0.074	0.433	58.0	8	10
	0.95	0.066	0.261	70.4	8	9	0.181	0.496	41.8	8	9
20	0.75	0.010	0.460	269.1	8	12	0.027	0.633	133.3	8	12
	0.85	0.027	0.397	162.6	8	10	0.074	0.562	78.9	8	10
	0.95	0.066	0.458	141.3	8	9	0.181	0.645	61.7	8	9
30	0.75	0.010	0.486	228.0	8	13	0.027	0.572	119.8	8	12
	0.85	0.027	0.454	125.2	8	11	0.074	0.480	65.5	8	10
	0.95	0.066	0.529	83.7	8	10	0.181	0.496	41.8	8	9
40	0.75	0.010	0.630	185.7	10	12	0.027	0.369	100.4	8	11
	0.85	0.027	0.227	88.0	8	10	0.074	0.433	58.0	8	10
	0.95	0.066	0.261	70.4	8	9	0.181	0.496	41.8	8	9
50	0.75	0.010	0.750	147.6	12	12	0.027	0.711	75.2	10	12
	0.85	0.027	0.611	64.2	11	10	0.074	0.492	33.7	9	10
	0.95	0.066	0.675	31.4	11	9	0.181	0.557	16.6	9	9

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

Fig. 3. 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 $(p_1, m_1), (p_2, m_2), \dots, (p_n, m_n)$, where p_i and m_i are the number of processors and memory units, respectively and $p_i \leq p_{i+1}$.

$$Y(n_p, n_m) = \sum_{i=0}^{n_p - p_n} \binom{n_p}{i} P_p^{n_p - i} (1 - P_p)^i \left(\sum_{j=0}^{n_m - m_n} \binom{n_m}{j} P_m^{n_m - j} (1 - P_m)^j \right) + \sum_{i=\max(n_p - p_n + 1, 0)}^{n_p - p_n - 1} \binom{n_p}{i} P_p^{n_p - i} (1 - P_p)^i \left(\sum_{j=0}^{n_m - m_n - 1} \binom{n_m}{j} P_m^{n_m - j} (1 - P_m)^j \right) + \dots + \sum_{i=\max(n_p - p_2 + 1, 0)}^{n_p - p_1} \binom{n_p}{i} P_p^{n_p - i} (1 - P_p)^i \left(\sum_{j=0}^{n_m - m_1} \binom{n_m}{j} P_m^{n_m - j} (1 - P_m)^j \right)$$

configuration.

We used the same random examples used in the previous Section assuming 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 V. The design of fault-tolerant system with optimal productivity has been performed under various design parameters such as the probability of

TABLE V

ALL NONREDUNDANT CONFIGURATIONS THAT SATISFY THE THROUGHPUT REQUIREMENT USING THE PROPOSED HETEROGENEOUS BISIR-APPROACH.

# 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

a good processor, the probability of a good memory unit, the area ratio of processor and memory, as suggested by Stapper et al. [12]. The results of redundant system with optimal productivity are shown in Tables VI and VII 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

TABLE VII

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.

# 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

the initial yield, the final yield, productivity, the number of processors and memory units of the optimal design, respectively. 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 results indicate several orders of magnitude improvement in productivity and yield by the method.

VI. CONCLUSION

For the first time we proposed a design methodology for 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. On a large set of examples, the approach resulted in significant improvement in productivity and throughput.

REFERENCES

- [1] 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.
- [2] 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.
- [3] 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.
- [4] R. Karri, A. Orailoglu, "High-Level Synthesis of Fault-Secure Microarchitectures," Design Automation Conference, pp. 429-433, 1993.
- [5] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by Simulated Annealing," Science, Vol. 220, No. 4598, pp. 671-680, 1983.
- [6] E.L. Lawler, C.U. Martel, "Scheduling periodically occurring tasks on multiple processors," Information Processing Letters, Vol. 12, No. 1, pp. 9-12, 1982.
- [7] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow", Proceedings of the IEEE, Vol. 75, No. 9, pp. 1235-1245, 1987.
- [8] A. Orailoglu, R. Karri, "Automatic Synthesis of Self-Recovering VLSI Systems," IEEE Trans. on Computers, Vol. 45, No. 2, pp. 131-142, 1996.
- [9] M. Potkonjak, J. Rabaey, "Algorithm Selection: A Quantitative Computation-Intensive Optimization Approach", International Conference on Computer-Aided Design, pp. 90-95, 1994.
- [10] 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.
- [11] J. Rabaey, et al., "Fast prototyping of data path intensive architecture", IEEE Design and Test, Vol. 8, No. 2, pp. 40-51, 1991.
- [12] 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.