CONFIGURABLE SPARE PROCESSORS: A NEW APPROACH TO SYSTEM LEVEL FAULT-TOLERANCE [1]

Kyosun Kim, Ramesh Karri
{kkim,karri}@ecs.umass.edu
Department of ECE
University of Massachussetts
Amherst, MA 01003

Miodrag Potkonjak
miodrag@cs.ucla.edu
Department of Computer Science
University of California,
Los Angeles, CA 90095

Abstract

In this paper, we have developed a methodology for behavioral synthesis of an important class of reconfigurable data path designs called **configurable spare processors.** Traditionally, a processor failure has been tolerated by dedicating a spare for the processor. However, this has a significant area overhead. In contrast, we present a new technique wherein several processors share one or more configurable spare processors. A configurable spare efficiently implements any of k applications and can be configured to substitute for a faulty processor implementing one of these k applications.
In this paper, we address three important techniques targeting configurable spare processor synthesis.

- Firstly, we address **application bundling** wherein n application control-data flow graphs (CDFGs) are bundled into at most m groups such that the sum of the areas of the corresponding implementations is minimized. All throughput and fault-tolerance constraints for all applications are satisfied.
- The area overhead of each of the application bundles is further optimized by **retiming the applications within a bundle** by considering its effects on the remaining applications in the bundle.
- Finally, each application bundle is **synthesized into a configurable spare processor.**

The effectiveness of all approaches, algorithms, and software implementations is demonstrated on a number of real-life examples. The validation of all presented examples is complete in a sense that we conducted functional simulation to complete layout implementations.

1 Introduction

Configurable data paths are especially useful in designing low-cost fault-tolerant systems by providing multiple functionalities and flexibility. Towards illustrating the benefits of configurable data paths in the context of low-cost fault-tolerant system design consider a system consisting of dedicated data paths for three signal processing applications shown in figure 1.
The area overheads of the physical layouts of each of the data paths are shown in figure 2. A straightforward approach to tolerate any single processor failure within such a system involves adding a backup processor of each type. This entails 100% area overhead. The area overhead becomes even more prohibitive as the number of faulty processors that can be tolerated increases.

[1]This research was partially supported by SAMSUNG Electronics Co., Ltd.

1063-6722/96 $5.00 © 1996 IEEE

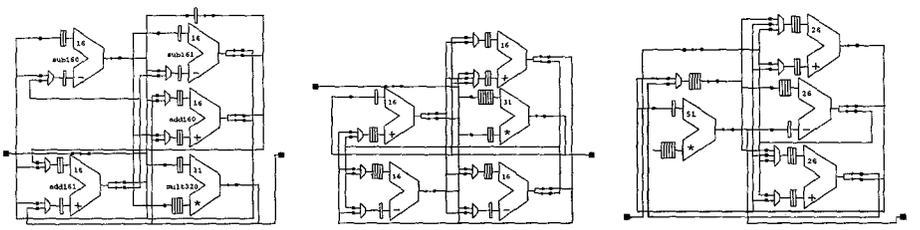

Figure 1: Dedicated data paths for (a) Low pass LDI filter (LDILP) (b) fifth order wave digital filter (WDF5) and (c) ninth order wave digital filter (WDF9)

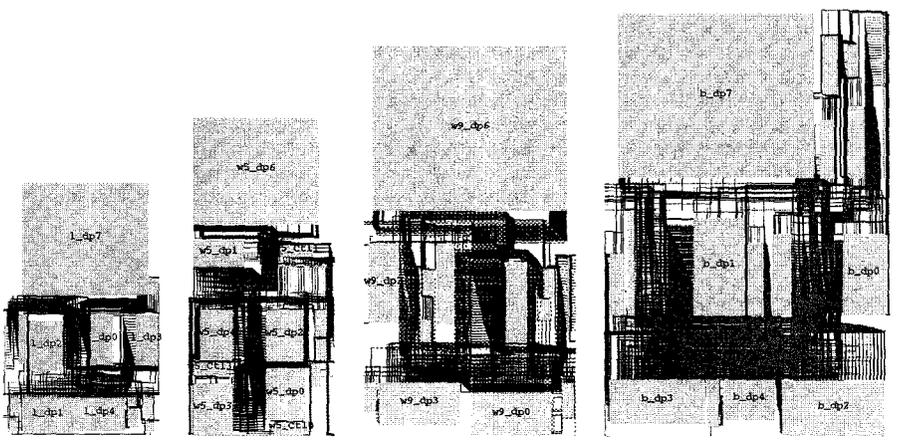

Figure 2: Layouts of (a) LDILP (21.3 mm^2) (b) WDF5 (25.4 mm^2) (c) WDF9 (45.1 mm^2) and (d) Configurable Spare Processor (66.5 mm^2)

Alternately, consider a configurable spare data path that subsumes the functionality of the three applications as shown in figure 3 [2]

The flexibility provided by such a configurable spare data path can be used to tolerate the failure of any single processor in the system. For example, when the LDILP processor fails, the spare data path can be configured to implement the LDILP application. Similarly, when the WDF5 or WDF9 processors fail, the spare data path can be programmed to implement the corresponding functionality. Furthermore, the area of the layout of such a configurable spare is only 66.5 mm^2. The overhead when compared with the dedicated spare approach is only 72.4%. These savings become even more significant as the number of functionalities increases. However, the overhead due to the disparity of the requirements in terms of hardware, interconnection, and word length prevents excessive number of functionalities from being implemented on a single spare processor.

[2]The schematics and layouts of figures 1, 2, and 3 are generated automatically by the system.

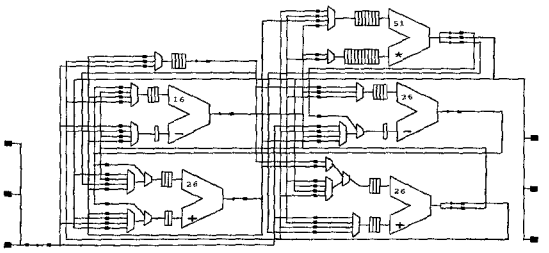

Figure 3: Configurable spare data path supporting LDILP, WDF5 and WDF9 applications

In this paper, we present a behavioral synthesis approach for designing such configurable spare data paths and discuss the relationship between configurable data paths, degree of fault-tolerance and area overhead.

2 Related Research

Behavioral synthesis has been an active area of research for more than two decades [4, 2, 10], and numerous outstanding systems have been built targeting both data path oriented and control oriented applications. Behavioral synthesis traditionally has been addressing synthesis and optimization of a single CDFG for sampling rate, area, and more recently power and test hardware overhead minimization [10, 2]. Recently, a few efforts have been reported on behavioral synthesis techniques for fault tolerant designs. Karri and Orailoglu [13] presented scheduling, assignment and transformation-based methods for fault-tolerance against transient faults. Guerra et al. [5] presented the first work which concentrates on permanent faults. They showed how fault tolerance achieved using a set of spare units can be used for yield and productivity enhancement. Recently Iyer et al [6] introduced a method which explores trade-offs between performance and yield.

Automatic synthesis of self-recovering microarchitectures has been previously addressed. An algorithm that intertwines checkpoint insertion and scheduling (of operations in the input algorithm to clock cycles) to synthesize self-recovering microarchitectures for supporting fault-recovery in hardware was first presented in [12, 7]. Guerra *et. al* have developed synthesis for built-in self-repair using redundant modules[5]. More recently, Blough, Kurdahi and Ohm [3] presented an algorithm for recovery point insertion in recoverable microarchitectures that minimizes the number of rollback points given constraints on the number of registers and maximum number of time steps between any two rollback points (the stride). These RT-level techniques for transient and permanent fault-tolerance have been successful in certain design scenarios. The main target for built-in-self-repair (BISR) techniques for yield enhancement are traditionally systems that are bit-, byte-, or digit- sliced, and in particular memories [11, 15] and PLAs [8, 15].

3 Preliminaries

Computational and Timing Models: Majority of multimedia, DSP, video, communication, control, and graphics application are specified as periodic computations on a stream of data. Therefore, a natural and proper computational model for those important application domains is homogeneous synchronous data flow [9]. Each application is defined

by a control data flow graph (CDFG) and the set of timing constraints, most commonly throughput requirements[14].

Hardware Model: Modern data path designs, both general purpose and application specific, invariably group registers in register files in order to better enable sharing of control logic and to facilitate area-efficient layouts. We assume the dedicated register file model [14] where each register is connected to a single input of an execution unit, while each unit can send data to an arbitrary number of registers. This model is also exceptionally well suited for implementing fault tolerance for yield enhancement. The control is synthesized by combining different controllers into one using logic synthesis tools resulting in small hardware overhead.

System Level Fault Model and Fault Diagnosis: We assume that a processor is either faulty or fault-free. The proposed approach requires fault detection and diagnosis as a preprocessing step. Any off-line testing and diagnosis scheme such as full-scan, combinational ATPG and BIST can be used.

Although in our implementation we followed the presented models, the approach can be easily retargeted to models of other design platforms.

4 Algorithm

In this section we describe the synthesis and optimization algorithms used in programs which realize the proposed configurable spare approach to system level fault-tolerance. The overall flow of our synthesis system for design of fault-tolerant application specific systems using configurable spare processors is given by the following pseudo-code.

```
Synthesis_Algorithm()
{
    ASIC_Synthesis();
    Application_Bundling();
    Bundle_Retiming();
    Configurable_Spare_Synthesis();
}
```

The ASIC_Synthesis() is the standard behavioral synthesis step where each CDFG is realized. The information on scheduling vis-a-vis the hardware and interconnection requirements is created in this step and fed into following steps.

4.1 Application Bundling

Towards synthesizing area-efficient spare processors, we propose an approach within which applications with similar topology, hardware types, etc., are identified and bundled into a chip. This is because designing a reconfigurable data path for an arbitrary set of applications will be often time counter-productive. Whereas putting topologically incompatible applications can entail significant interconnect overhead, applications with incompatible hardware types can entail significant functional unit overhead. Following application bundling, each bundle is synthesized into a separate, area-efficient configurable processor. Formally, application bundling can be defined as follows:

Given an underlying hardware model and N applications, each with its own execution time bound and hardware requirement when implemented as a dedicated processor, partition the applications into bundles minimizing the overall area of the configurable processors programmed for the N applications.

A probabilistic rejectionless framework has been used for application bundling. First, applications are bundled randomly. Based on the incompatibility between the applications and

the bundles, the algorithm proceeds in a way similar to probabilistic iterative techniques. A source bundle is randomly chosen, probabilistically favoring bundles with incompatible applications. From such a bundle, an incompatible application is probabilistically selected and moved to another bundle where applications are compatible with the selected application. The hardware area of all bundles is then computed and the current bundling configuration is saved if it is the best one so far. This continues until no more improvement is obtained for a given number of iterations.

In most data path intensive systems, the area of an IC is approximated as the sum of the active region and the interconnection region. Also, the increase in area of the controller and the dead spaces are regarded as proportional to the total area, and therefore excluded from the estimation. If $h_t(g)$ is the number of units of type t and $Area_t$ is the area of a unit of type t, the area of the active region is:

$$Area_{active} = \sum_{t \in T} \max_{g \in B_i} h_t(g) \cdot Area_t$$

Let b_i be the number of bits of bus i and p be the wire pitch. Since the average wire length of buses is $(Area_{active} + Area_{ic})^{1/2}$, and the total number of bits is $b = \sum_{i \in I} b_i$, the area of the interconnection region is:

$$Area_{ic} = \frac{b^2p^2 + (b^4p^4 + 4b^2p^2 Area_{active})^{1/2}}{2}$$

For increased accuracy, the number of fanouts of bus i, f_i is taken into account, and the effective total number of bits becomes $b = \sum_{i \in I} b_i \cdot (1 + 0.25(f_i - 1))$. The fanouts for all the buses are determined as follows. Firstly, the interconnection between the hardware units is expressed as a matrix with rows of execution units and columns of register files. If an entry is 1, a connection exists between the execution unit corresponding to its row and the register file corresponding to its column. For each application, this table is created by ASIC_Synthesis(). Since the hardware and interconnection are shared among the applications in a bundle, the hardware units and register files of each application have to be mapped into those of the bundle, and then the interconnection matrix of the application also has to be projected on to that of the bundle. Now, for each bus, the number of fanouts can be obtained by counting 1's in the row which corresponds to the fanin hardware unit of the bus.

k−processor fault-tolerance is incorporated by ensuring that that each application is included in at least k distinct bundles.

4.2 Bundle Retiming

The area overhead of each configurable spare processor can be further reduced based on the following two observations:

- The peak usage of a given hardware type for an application can be minimized by retiming the CDFG corresponding to the application[14].
- In a configurable spare processor, different applications may contribute to the hardware requirements of the different hardware types. The peak usage of one type of hardware may be reduced by exploiting the flexibility afforded by the the peak hardware usage of a different type of hardware. For example, let applications A and B be implemented as a configurable spare processor. If application A uses three adders and two multipliers, and application B uses two adders and three multipliers, the

configurable spare processor uses three adders and three multipliers. Application B can be retimed to reduce the number of multipliers by one even if this results in it requiring three adders.

We have developed an integrated retiming technique that retimes all of the application CDFGs concurrently so as to mix-and-match the hardware requirements of some applications with the peak hardware requirements of some other applications.

4.3 Configurable Spare Synthesis

To take into account the unique characteristics of configurable spare synthesis, we have developed area-efficient allocation, upgrading, assignment and scheduling schemes for mapping heterogeneous applications onto a configurable processor. The configurable spare synthesis can be defined within this framework:

Given an underlying hardware model and N applications, each with its own execution time bound, synthesize a minimum area design so that any one of these N applications can be executed at any given time.

```
    G = { g | applications in the bundle }
    A = { a | hardware allocation }
    T = { t | temporary hardware allocation }
    C[type] = hardware criticality

    ConfigurableSpareSynthesis(G)
    {
1:      A ← InitialAllocation(G, φ)

        /* Find a Feasible Solution */
2:      while ( (g ← ApplicationOrdering(G, ∀types, A) ) ≠ φ)
3:          T ← InitialAllocation({g}, A)
4:          while (AssignAndSchedule(g, T, &C) = FAIL)
5:              type ← MostCriticalHWTypeToUpgrade(C)
6:              T ← T ∪ { a ← Upgrade(g, A, type) }
7:          RemoveRedundancy(g, T, C)
8:          A ← A ∪ T

        /* Remove Global Redundancy */
9:      while ( (type ← LeastCriticalHWTypeToDowngrade(C) ) ≠ φ)
10:         while ( (g ← ApplicationOrdering(G, type, A) ) ≠ φ)
11:             T ← Downgrade({g}, A, type)
12:             if (AssignAndSchedule(g, T, &C) = FAIL) break
13:         if (g = φ) A ← Downgrade(G, A, type)
    }
```

Figure 4: Configurable Spare Synthesis Algorithm

The configurable spare synthesis algorithm is outlined in figure 4. An initial allocation, A for the bundle, G is derived in step 1. Beginning with the most critical application, a feasible solution for the entire bundle is obtained in steps 2-8. From the total hardware allocated

to the bundle, the hardware allocation T for the candidate application is obtained in step 3. Steps 4-6 constitute the synthesis loop. Assignment and scheduling of the candidate application are carried out using this allocation. If the allocated hardware is not sufficient, either the wordlength of one of the existing hardware units is increased or a new hardware instance is added. This is called **upgrading**. The upgraded hardware in T is reflected in the overall allocation in step 8. In step 7, any subset of the current allocation is checked for feasibility. Since the criticality of the applications changes dynamically with the changes in allocation, application ordering is included in the loop. In the global redundancy removal phase, the applications are also ordered dynamically, and the **downgrading** of a hardware type aborts as soon as an application fails to tolerate the removal.

5 Experimental Results

applications	# of nodes	# of edges	word len'	crit' path	avail time	allocation				area (mm^2)
						+	-	*	reg	
ADAPT	24	24	16	6	9	2	0	3	28(0)	33.93
ARAI	51	63	22	8	10	7	5	2	44(13)	83.87
DEC	67	79	16	16	16	2	4	2	58(12)	39.49
FFT8	30	31	16	5	5	3	3	2	20(2)	35.76
FIR20	32	42	16	3	7	10	0	3	52(10)	45.04
GM1M	20	27	20	14	14	2	1	2	22(7)	31.96
IIR8	39	57	11	9	12	2	0	2	42(18)	14.11
LDILP	14	18	16	6	6	2	2	1	19(4)	13.96
NC	66	73	16	12	19	2	1	3	71(11)	52.33
VOLTERRA	30	39	24	12	12	1	0	2	28(10)	43.50
WAVELET	53	65	16	14	14	2	1	2	44(13)	28.76
WDF5	23	29	16	12	12	2	2	1	25(6)	15.23
WDF7	31	37	22	12	12	2	4	2	35(7)	48.46
WDF9	23	28	26	9	9	2	1	1	27(5)	32.33

Table 1: Example Applications

Configurable spare data path synthesis techniques proposed in this paper were validated on the set of DSP, video, control and communication applications summarized in table 1. The selected applications span a wide range of complexities in computational structures and include Arai's fast DCT algorithm (ARAI), decimate-by-four wave digital fileter(DEC), S. Winograd's small-N DFT for N = 8 (FFT8), noise canceler (NC), and ninth degree bi-reciprocal WDF with Butterworth response (WDF9). For each application, columns 2-5 show the number of nodes, the number of edges, the word length, and the critical path, respectively. The input latency for each application is shown in column 6. The next four columns give the hardware allocation. The column titled "reg" shows the number of registers used in the implementation. The numbers in parentheses are the register counts for constants. The last column reports the estimated area in mm^2 of the dedicated implementation. This is used to evaluate the area overheads of the configurable spare data paths.

# of faulty processors	Bundles	Area	
		dedicated	config'ble
1	{ADAPT, GM1M, IIR8, NC, WAVELET}, {WDF9}, {ARAI, DEC, VOLTERRA, WDF7}, {FFT8, LDILP}, {FIR20, WDF5}	518.7 (100%)	368.0 (71%)
2	{ADAPT, GM1M, VOLTERRA}, {IIR8}, {WDF5}, {WAVELET, WDF9}, {ADAPT, LDILP, NC}, {GM1M} {ARAI, VOLTERRA, WDF7, WDF9}, {DEC, FFT8,}, {ARI, FFT8, WDF7}, {DEC, LDILP, WDF5}, {FIR20}, {FIR20, IIR8, NC, WAVELET},	1037.4 (200%)	736.6 (142%)

Table 2: Application Bundling: Minimization of Overall Area

5.1 Application Bundling: An Evaluation

Application bundling was invoked on this set of applications, targeting single and double processor failures. Table 2 summarizes the results. Column 2 shows the applications in each bundle which are enclosed in braces. The total area when implemented as dedicated chips and that of configurable spare processors are shown in columns 3 and 4. The overheads of configurable spare processors (with respect to a non-fault-tolerant implementation) are 71% and 142% for single and double processor failures, respectively. In row 2 of table 2, four of the bundles have a single application. This is because the applications are incompatible in terms of hardware and interconnection requirements with the remaining applications.
For a system with fourteen such dedicated application specific processors, five configurable spare processors and one dedicated processor as in the first row of table 2 can tolerate any single processor failure with. Similarly, the twelve configurable spare processors and four dedicated processors which implement the bundles in row 2 of table 2 can tolerate any two-processor failures. This is because each application is a member of at least two bundles.

5.2 Experiments using Configurable Spare Data path Synthesis

Each of the application bundles from table 2 is retimed to minimize the area of the configurable spare processor, and synthesized by the system. The results of twelve configurable spare data paths are summarized in table 3.
The applications in each configurable spare processor are summarized in column 1. Failures in a dedicated processor can be tolerated by programming the spare data path to implement the application. The hardware allocation is summarized in the next four columns. The area of the synthesized configurable spare data path is given in the 6th column. Finally, the total area as a percent of the dedicated spare data path approach is summarized in the last column. For the spare data path corresponding to the third row in table 3, this is calculated as $\frac{59.28}{33.93+13.96+52.33}$.

6 Concluding Remarks

We have presented a system level fault-tolerance technique using configurable spare data paths by exploiting the flexibility afforded by implementing multiple applications. The proposed techniques have been implemented and the resulting system has been used to

Configurable Spare Data path	allocation				area (mm^2)	area as a % of dedicated spare approach
	+	-	*	reg		
{FIR20, IIR8, NC, WAVELET}	4	1	2	110(49)	59.43	42.38
{ADAPT, GM1M, IIR8, NC, WAVELET}	2	1	3	98(47)	94.39	58.60
{ADAPT, LDILP, NC}	3	2	2	70(14)	59.28	59.15
{ADAPT, GM1M, VOLTERRA}	2	1	2	53(17)	72.80	66.55
{ARAI, VOLTERRA, WDF7, WDF9}	4	3	3	80(36)	146.14	66.99
{DEC, LDILP, WDF5}	2	4	2	71(22)	49.34	71.84
{ARAI, DEC, VOLTERRA, WDF7}	4	4	3	105(43)	155.75	72.23
{WAVELET, WDF9}	2	1	2	58(18)	55.98	73.82
{FIR20, WDF5}	5	2	2	69(16)	44.61	74.02
{DEC, FFT8}	3	4	2	69(14)	60.71	80.68
{ARAI, FFT8, WDF7}	4	3	3	56(22)	126.58	81.13
{FFT8, LDILP}	3	3	2	26(26)	40.95	82.36

Table 3: Configurable Spare Data path Synthesis

synthesize numerous industrial strength configurable spare data paths that can tolerate single-processor and two-processor faults.

References

[1] M. Abramovici, M.A. Breuer, A.D. Friedman, Digital Systems Testing and Testable designs, Computer Science Press, New York, NY, 1990.

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

[3] D. M. Blough, F. J. Kurdahi, and S. Y. Ohm, "Optimal Recovery Point Insertion For High Level Synthesis of Recoverable Microarchitectures," *Proc. 25th FTCS*, 1995.

[4] D. D. Gajski, N. D. Dutt, A. Wu, and S. Lin. *High-Level Synthesis: Introduction to chip and system design.* Kluwer, 1992.

[5] L.M. Guerra, M. Potkonjak, J. Rabaey, "High Level Synthesis Techniques for Efficient Built-in Self Repair," IEEE Workshop on DFT in VLSI systems, pp. 41-48, 1993.

[6] B. Iyer, R. Karri, I. Koren, "Phantom Redundancy: A High-Level Synthesis Approach for Manufacturability," ICCAD 95, pp. 658-661.

[7] R. Karri and A. Orailoğlu "Synthesis of Optimal Self-Recovering Microarchitectures," *Proc. FTCS*, June 1993.

[8] I. Koren, D.K. Pradhan, "Introducing Redundancy into VLSI Designs for Yield and Performance Enhancement", *Proc. FTCS*, pp. 330-335, 1985.

[9] E. A. Lee and D. G. Messerschmitt: "Static Scheduling of Synchronous Data flow Programs for Digital Signal Processing," IEEE Trans. on Computers, Vol. 36, No. 1, pp. 24-36, 1987.

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

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

[12] A. Orailoğlu and R. Karri. "Coactive Scheduling and Checkpoint Determination during the High Level Synthesis of Self Recovering Microarchitectures," *IEEE Trans on VLSI Systems*, 2(3):304-311, 1994.

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

[14] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Data Path Intensive Architectures," IEEE Design & Test, Vol. 8, No. 1, pp. 40-51, 1991.

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