

Rapid Prototyping of Fault Tolerant VLSI Systems*

Ramesh Karri

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

Karin Hogstedt, Alex Orailoğlu

Department of Computer Science and Engineering
University of California, San Diego
La Jolla CA 92093

Abstract: In this paper, we relate fault-tolerance constraints to chip area and present a methodology for rapidly compiling an algorithmic description into area-efficient fault-tolerant VLSI ICs. Whereas detection and recovery from environment induced transient faults is accomplished by checkpointing and rollback, uninterrupted operation for the lifetime of a mission is ensured by injecting redundancy.

Towards validating this methodology, we synthesized fault tolerant implementations of a 16-point FIR filter starting from an algorithmic description. These fault-tolerant designs were then critically appraised.

1 Introduction

Until recently, fault-tolerance techniques such as concurrent error detection for on-board diagnostic systems, as well as checkpointing in self-recovering systems have been mainly embedded into software [14, 1]. For improving system reliability, most designs resort to expensive, system-level replication [11]. Whereas the time overhead associated with embedding of fault-tolerance in software is unacceptable in time-critical applications, the space overhead of straightforward replication-based fault-tolerance is overly expensive for consumer-oriented applications. Recent advances in VLSI Technology are making it feasible to design area-efficient, hardware-oriented approaches to fault-tolerance; general purpose processors with checkpointing capability [12], application-specific ICs for comprehensive concurrent error detection [15] and ICs with built-in reliability [5] are beginning to emerge. However all of these hardware oriented approaches entail significant effort on the part of the designer; the onus of either partitioning a system and inserting checkpoints, or meticulously handcrafting the fault-detection mechanism, rests entirely on the designer.

Earliest implementations of on-chip concurrent error detection and fault-tolerant general purpose processors have been mostly paper designs [3, 4]. Actual implementations have been relatively infrequent. Tsao *et al.* [13] designed one of the earliest single chip implementations of a fault-tolerant microprocessor. More recently, Yen *et al.* [15] have not only designed and

physically implemented a microprogram control with concurrent error detection capabilities but also provided the first critical appraisal of the overhead of fault-tolerance based on mask level information. Tamir and Tremblay [12] on the other hand, have implemented a fault-tolerant microprocessor that supports rollback based recovery capability. However, all of these fault-tolerant VLSI implementations have been designed manually.

In contrast to all of the earlier approaches, we will outline an automatic methodology for rapidly designing area-efficient fault-tolerant VLSI systems. The paper is organized as follows. In section 2 the general methodology for synthesizing fault-tolerant VLSICs will be presented followed by a detailed description of self-recovering and reliable VLSIC synthesis in sections 3 and 4. In section 5, the fault-tolerance overhead of these fault-tolerant designs will be evaluated vis-a-vis a non-redundant implementation.

2 CAD of Fault-Tolerant VLSI

In order to rapidly realize high-quality, area-efficient ICs without encumbering the designer, fault-tolerance constraints should be explicitly ingrained into a computer aided design methodology. *Microarchitectural* (or *high level*) synthesis and *physical design* are the two important steps in a top-down VLSI design methodology.

Microarchitectural synthesis transforms an algorithmic description of a system into a microarchitecture subject to designer specified constraints such as time and resource. Initially, the algorithmic description (which is described in a hardware description language) is translated into an intermediate representation called the data flow graph (DFG). Then each operation in the DFG is scheduled into a clock cycle. Subsequently, the operators are bound to the available functional units, variables are allocated to registers and data transfers are assigned to buses. This phase is called *data path synthesis*. Next, by modeling each clock cycle as a state and transitions between clock cycles as directed arcs between states a state diagram is synthesized. This state diagram is then automatically translated into a controller that generates control signals that ensure the desired system operation.

During physical design a microarchitecture is transformed into a physical layout. Initially, register trans-

*This work is supported by NSF I/UC Research Center on ICAS at University of California, San Diego and a University of California COR grant.

fer level modules in the microarchitecture are *placed* on a chip so as to minimize the estimated interconnect overhead. Such a module placement step partitions the routing space on the chip into multiple routing regions called the *channels*. The *global routing* step then apportions these routing regions among the various inter-module connections (or *nets*). Finally, all nets in a channel are routed without violating the inter-net constraints.

2.1 Assumptions

Our assumptions regarding the input formats and the models and metrics for performance and hardware at the microarchitectural and the physical design level are as follows:

Microarchitecture Level : Our optimization methods work on a language independent intermediate representation called the disjoint control and data-flow graph (CDFG) [9]. The disjoint CDFG decouples the maintenance and handling of the data flow constructs (for example, assignments and evaluations) from the control flow constructs (for example, conditionals and loops).

Physical Design Level : Consistent with the back-end physical design tool suite [2] the synthesized microarchitecture is described as a *netlist* of cells (instances of physical implementations of functional modules) together with the nets that interconnect them. The area is measured in λ^2 where λ is a technology independent unit of length. The building blocks that formed the basis for all the designs were created as standard cells. The area-

Module	Width	Area (λ^2)	Delay (ns)
Adder	8	828240.00	70.5
Shift-&-Add Mult	8	2036800.00	107.5
Register	8	186680.00	5.6
Voter	8	89096.00	5.7
Multiplexer	8	171648.00	18.6
Majority Voter	8	288696.00	14.0

Table 1: The Module Set

delay characteristics of these building blocks are summarized in table 1. The delay through a 1:2 Voter is close to that of a register. This is because a 1:2 Voter is implemented as a register with a line that signals the validity of its contents. Similarly, the delay through the Majority Voter (a 2-out-of-3 voter in this paper as the maximum amount of redundancy is only three) is approximately two times that of a register as the critical path of the implementation consists of two registers. That there is no propagation delay and area correlation between the adder and the add-and-shift multiplier is largely due to the fact that they were designed independently starting from different descriptions.

Finite Impulse Response(FIR) filtering is used extensively in automotive applications (such as active suspension, adaptive ride control, antiskid braking, and global positioning systems)[8] and space applications. Consequently, a 16-point FIR filter (using 8-bit fixed point arithmetic) was selected as the vehicle for our experiments.

3 Synthesis of Self-Recovering VLSICs

Systems that can detect and recover from (transient) faults are termed self-recovering. In our model for self-recovery(see figure 1), faults are detected by duplicating a computation on disjoint hardware and voting; both computations take the same input and a fault is detected if their outputs disagree. On the

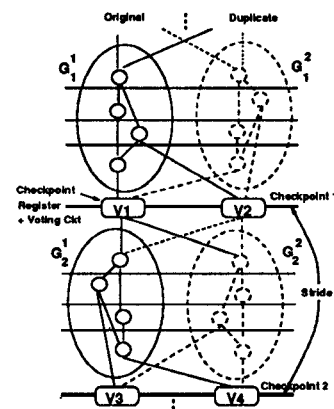


Figure 1: The model for Self-Recovery

other hand, recovery from faults is based on checkpointing and rollback. Whereas *rollback* is the process of backing up a system several cycles in response to a fault, a *checkpoint* is the state of the system to which rollback occurs. The maximum number of clock cycles between successive checkpoints is the *retry period*. The checkpoint overhead is the maximum of the state information, and by implication the voting overhead, over all checkpoints. Since voters are the single points of failure in an otherwise fault-tolerant design, they are meticulously handcrafted for fault-tolerance and are hence expensive.

3.1 Data Path Synthesis

A self-recovery scheduler[6] in addition to assigning operations to clock cycles and scheduling data transfers (between operations) to clock cycle boundaries, determines checkpoints, subject to constraints on system throughput, and allowable recovery time overhead.

Following the scheduling step, a self-recovering data path is realized by assigning the scheduled operations to specific operators subject to the constraint that the hardware used in the original and the duplicate

computations be disjoint. Towards validating the self-recovery synthesis system described earlier, we have designed a self-recovering 16-point FIR filter chip. Initially, an algorithmic representation of the 16-point FIR filter is mapped into a self-recovering schedule subject to recovery time constraints (specified as a retry period of three clock cycles) and performance constraint of nine clock cycles. For simplicity, we assumed that back-to-back chaining of operations in a clock cycle is not permissible and that detection is based on straightforward duplication. Whereas the schedule of the original computation entails two adders and two multipliers, straightforward duplication requires twice as many pieces of each type of hardware. The self-recovering data path (satisfying the hardware disjointness constraint) corresponding to this schedule is shown in figure 2. The shaded squares and rectangles denote muxes and checkpoint registers respectively, while the triangles denote voters. The duplicate data path is stacked behind the original.

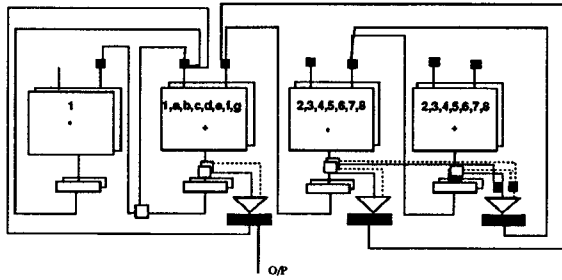


Figure 2: Data path of the Self Recovering 16-point FIR filter

3.2 Control Path Synthesis

A simple controller for the self-recovering data path is derived by modeling (i) every clock cycle and (ii) all checkpoints (observe that during scheduling a checkpoint was only a clock cycle boundary) as states in a state diagram. The resulting state diagram corresponding to the self-recovering 16-point FIR filter schedule is shown in figure 3. An immediate consequence of transforming checkpoints into clock cycles is the increase in the number of clock cycles. Clearly, at a checkpoint (shown as a dark oval in figure 3) detection of a fault initiates rollback (shown as dotted directed lines in figure 3). During rollback, the correct state information saved at the previous checkpoint is restored. For simplicity, the control signals are not shown in the state diagram. A standard cell implementation of the control path is derived using the state machine and standard cell synthesis tools in *OCTTOOLS*. The design characteristics are summarized in figure 2.

3.3 Physical Design

The net list of the resulting self-recovering microarchitecture (in the *bdnet*[2] format) is created and then

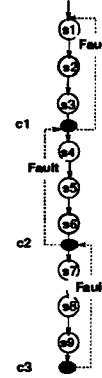


Figure 3: State machine for the self recovering 16-point FIR filter

	Controller	Complete Design
Area	1765800.00 λ^2	177435448.00 λ^2
Dimensions	1080 $\lambda \times 1635 \lambda$	13304 $\lambda \times 13337 \lambda$
Instance Area	36.11%	14.72 %
Remaining Area	63.89%	85.28 %
# of Instances	229	105
# of Vias	1189	4369
# of Connectors	1817	6649
# of Nets	247	572
Total Net Length	214325.00 λ	3609858.00 λ

Table 2: Controller and complete design statistics for the self-recovering design

transformed into an equivalent physical design using the macrocell place-and-route tools. The layout of the self-recovering 16-point filter is shown in figure 4. The area related characteristics of the self-recovering chip obtained using chipstat [2] are summarized in column 3 of table 2. Interestingly, the chip is dominated by interconnect (which constitutes about 85% of the total area).

Since voting is performed in a separate clock cycle, the total number of clock cycles is the sum of the number of functional clock cycles (C_{num}) and the number of checkpoint clock cycles (C_{check}). The throughput characteristics of the design are obtained as the total number of clock cycles ($C_{num} + C_{check}$) times the duration of a clock cycle (C_{dur}). In turn, C_{dur} of the self-recovering design is estimated as $Delay_{mult} + Delay_{reg} + Delay_{mux}$. Consequently, one pass through the filter takes $12 \times 131.7 = 1580.4$ ns.

4 Synthesis of Reliable VLSICs

Section 3 has focused on incorporating mechanisms for tolerating faults that arise mainly from phenomena *extrinsic* to the IC (for example, electromagnetic



Figure 4: Self-Recovering 16-point FIR Filter layout

radiation and vibrations[11]). In this section we devise synthesis techniques for reliability that are tailored towards faults *intrinsic* to the IC (for example, hot electron effect and electromigration). We will show how reliability constraints are ingrained into the module set selection phase of microarchitecture synthesis[10] and follow it up by deriving a reliable implementation of the 16-point FIR filter.

In order to synthesize reliable VLSICs, it is first necessary to characterize the individual module reliabilities. Towards this end we derived the failure rates using the empirical relationship MIL-HDBK-217E[11] given in equation 1:

$$\lambda = \pi_l \pi_q (C_1 \pi_t \pi_v + C_2 \pi_e) \quad (1)$$

where, π_l (range: 1-10) is the learning factor reflecting the maturity of the fabrication process, π_q (range: 0.25-20) is the quality factor depending on the burn-in procedures applied, π_t (range: 0.1-1000) is the temperature factor based on the ambient operating temperature, π_v (range: 1-10) is the voltage derating factor for CMOS devices, π_e is the application environment factor and C_1 ($\approx 0.00085\sqrt{\text{gates}}$), and C_2 are complexity failure rates dependent on the number of gates and number of pins[7]. We assumed that the system has a mission time of 10^6 hours. Typical values for module reliabilities are 0.995 for the adder and 0.98 for the multiplier. system reliability and let $\mathcal{R}_i^1, \mathcal{R}_i^2, \dots, \mathcal{R}_i^{n_i}$ be the reliabilities of the n_i modules of the i^{th} type. These can be computed using the formula for reliability. Let C be the overall cost constraint and C_i be the cost of one unit of the i^{th} functional unit type. Let $M_i^1, M_i^2, \dots, M_i^{n_i}$ be the n_i modules of the i^{th} type in the module-set and let there be k functionally distinct module types. Since each of the modules are redundant, let the number of functional

units in the module M_i^j be $|M_i^j|$. The derived module-set should satisfy maximum area and the minimum reliability constraints given by equations 2 and 3.

$$\sum_{i=1}^k (C_i * \sum_{j=1}^{n_i} |M_i^j|) \leq C \quad (2)$$

$$\prod_{i=1}^k \prod_{j=1}^{n_i} \mathcal{R}_i^j \geq \mathcal{R}_{sys} \quad (3)$$

Since the reliable module set selection as stated is a difficult optimization problem, we use a greedy heuristic for *redundancy injection*.

For a desired system reliability of 0.96 and time constraint of nine clock cycles, we invoked the Time-and-Reliability Constrained scheduler. The system has derived a module set comprising of {two 1MR adders, one 1MR multiplier, one 3MR multiplier} with an effective reliability of 0.969, and synthesized a schedule using it. The corresponding reliable datapath is shown in figure 5.

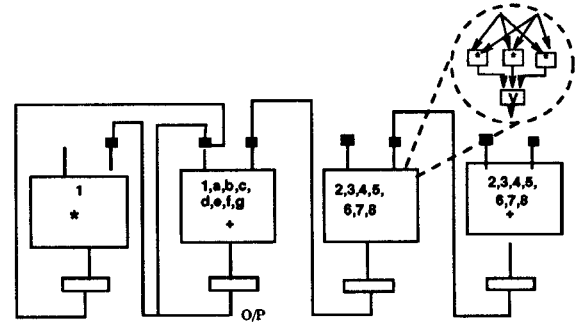


Figure 5: Reliable Data path of the 16-point FIR filter

4.1 Control Path Synthesis

The controller for the reliable data path is derived in a straightforward manner by modeling each clock cycle as a state in a state diagram. One of the multipliers is triple modular redundant (three multipliers feeding into a majority voter) and determines the duration of a clock cycle. Although the state machine is isomorphic to that of a non-redundant implementation of the FIR 16-point filter, it generates additional signals to control the 3MR multiplier. The area characteristics of the controller are summarized in table 3.

4.2 Physical Design

Next, the net list of the reliable microarchitecture is created using *bdnet*. Subsequently, the macrocell place-and-route subsystem is invoked to generate the physical layout shown in figure 6. From the layout it can be seen that there are very few empty spaces, indicative of a good dense layout. The chip area statis-

	Controller	Complete
Area	1253560.00 λ^2	66062952.00 λ^2
Dimensions	968 λ x 1295 λ	8262 λ x 7996 λ
Instance Area	38.75%	30.41 %
Remaining Area	61.25%	69.59 %
# of Instances	176	88
# of Vias	876	2748
# of Connectors	1349	3914
# of Nets	191	426
Total Net Length	143364.00 λ	2059329.00

Table 3: Controller and complete design statistics for the reliable design

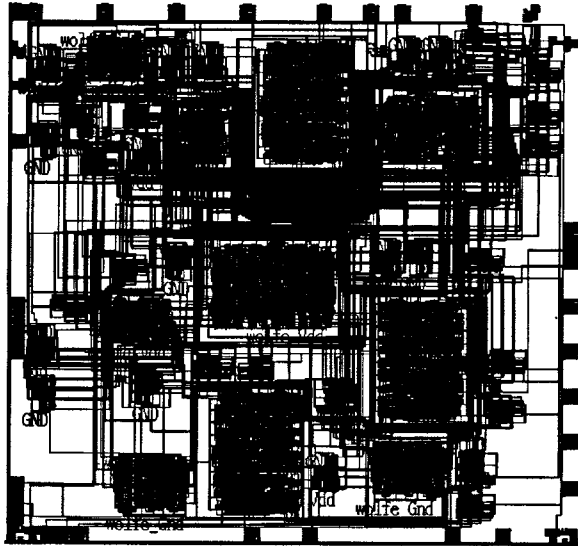


Figure 6: Chip Layout of the reliable 16-point FIR filter

tics of the reliable 16-point FIR filter IC are summarized in the last column of table 3. Contrast the 69.59% interconnect overhead of this implementation with that of the self-recovering implementation. This is because of the lesser amount of on-chip redundancy.

The clock duration (C_{dur}) is obtained as $Delay_{mult} + Delay_{majvoter} + Delay_{mux} + Delay_{reg}$. The first two terms represent the delay of the 3MR multiplier. Although the C_{dur} of the reliable implementation (145.7 ns) is larger than that of the equivalent self-recovering implementation (due to the larger propagation delay of the 3MR multiplier), the overall iteration time of the reliable implementation (1311.3 ns) is smaller than that of the self-recovering implementation. This can be attributed to the lesser number of clock cycles required to complete one iteration.

5 Evaluation of Fault-Tolerant Designs

In the previous two sections, we have systematically derived VLSI implementations of two fault-tolerant mechanisms and summarized their time and area overheads. These results convincingly demonstrate the feasibility of automating the design of fault tolerant VLSI systems. However, in order to provide a meaningful evaluation of these designs (in terms of area and performance overheads), we have designed a non-redundant version (or the basic design) of the FIR filter. The area statistics for the controller and the overall design are summarized in table 4. The clock duration (C_{dur}) as

	Controller	Complete Design
Area	1118512.00 λ^2	64822604.00 λ^2
Dimensions	848 λ x 1319 λ	7942 λ x 8162 λ
Instance Area	38.42 %	24.13 %
Remaining Area	61.58 %	75.87 %
# of Instances	153	85
# of Vias	763	2606
# of Connectors	1197	3445
# of Nets	167	400
Total Net Length	123334.50 λ	1545237.00 λ

Table 4: Controller and complete design statistics of the non-redundant 16-point FIR filter IC

well as the overall iteration time is smaller than that of the fault-tolerant designs. This is because neither C_{dur} ($= Delay_{mult} + Delay_{mux} + Delay_{reg}$) nor the number of clock cycles ($= 9$) is impacted by any overheads. The duration of a clock cycle and the time for one iteration are 131.7 ns and 1185.3 ns respectively.

5.1 Area Overhead of Fault-Tolerance

Initially, the designs are evaluated using a coarse grain measure of the hardware overhead, namely overall chip area. While the area overhead of enhancing the system reliability from 0.95 to 0.969 is negligible, the overhead for implementing on-chip recovery is significant. In fact, the area of the self-recovering design is about 2.73 times the corresponding non-redundant design. A cursory glance at the chip layout reveals that a straightforward manual rip up and rerouting (notice the islands of unused chip area in figure 4), can further reduce the overhead. From this coarse grain evaluation, we can conclude that a *fully automatic synthesis approach to self-recovering designs has yielded hardware overheads (173.8%) comparable to that of manually designed duplication based designs (138%)* [15]. Towards gaining insight into the specific apportionment of this hardware overhead, we conducted a detailed evaluation that focused on the overhead due to each of its several sources: the redundant hardware and voting circuitry in the data path, the additional control, and the extra interconnect.

The overhead due to redundant hardware (in the case of fault-recovery it is equal to the additional functional units and checkpoint registers together with the voting circuitry, while in the case of reliability enhancement it is equal to the additional functional units and voting circuitry) constitutes less than ten percent of the total chip area for both fault-tolerant implementations. Furthermore, the voting overhead is less than one percent of the total chip area justifying the non-redundant nature of their implementations.

Now we will evaluate the increase in complexity of the controller for each of the fault-tolerant designs against vis-a-vis the complexity of the controller for the non-redundant design. It is important to note that the controllers themselves are not fault-tolerant. Although the control overhead with respect to the controller for the non-redundant design is significant (12% for the reliable design and 58% for the self-recovering design), the control area as a percentage of the total chip area is by itself very negligible (less than 2%).

The interconnect overhead validates a perceived trend in VLSI: interconnect dominates the chip area. On an average, about 70% of the chip area is occupied by interconnect. The situation is even more acute for the self-recovering design, wherein 85% of the chip area is interconnect. Two useful metrics for evaluating the increase in interconnect due to fault-tolerance are *total net length* and *average net length*. The interconnect of the reliable design is about 1.33 times that of the basic design, while the interconnect of the self-recovering design is about 2.33 times. There is also a corresponding increase in the average net length.

Finally, the increase in pin count (from 26 in the basic design to 28 in the fault-tolerant designs) is negligible and hence is not a bottleneck for implementing on-chip fault-tolerance. The two additional I/O pins are used for the additional power and ground lines.

5.2 The Performance Penalty

The voting circuitry in the fault-tolerant design contributes to the performance penalty. The increase in clock period for the reliable VLSIC is 10.6% while there is no increase in clock period for the self-recovering design. This was because, instead of increasing the duration of the clock cycle, the number of clock cycles were increased in the latter case. The performance of the system is measured as the time for completing one computation. Consequently, there is a performance penalty of 10.6 % associated with reliability injection and 33.33% associated with self-recovery incorporation.

6 Conclusion

While the overhead of redundancy based reliability enhancement was very insignificant, duplication-based detection is competitive with previously reported manual fault-tolerant designs [15]. Following a critical appraisal of the fault-tolerant designs, the extra overheads can in part be attributed to: (i) the overpower-

ing contribution of interconnect towards the chip area (in the self-recovering design it constituted 85% of the total chip area while in the reliable design it constituted 62%). (ii) a combination of the large power requirements and conservative power estimation tools in the OCT tool suite resulting in wide power nets. (iii) the inadequacy of interaction between tools at various levels of the design abstraction.

References

- [1] P. Ames, P. Miles, E. Milham, C. Pilch, W. Rodda, L. Tedesco, and B. V. Tine. Automotive Electronics: A Design and Test Roundtable. *IEEE D&T of Computers*, 35(8):84-93, 1992.
- [2] A. Casotto. *OCTTOOLS 5.1 : The User Guide*. ERL, UC Berkeley, 1991.
- [3] Y. Crouzet and C. Landrault. Design of self-checking MOS-LSI circuits: Application to a four bit microprocessor. *IEEE Trans on Computers*, 29(6):532-537, 1980.
- [4] M. P. Halbert and S. M. Bose. Design Approach for a VLSI self-checking MIL-STD-1750A microprocessor. In *Proc of 12th FTCS*, pages 254-259, 1982.
- [5] C. Hu. The Berkeley Reliability Simulator BERT: an IC Reliability Simulator. *Microelectronics Journal*, 23(2):97-102, 1992.
- [6] R. Karri and A. Orailoglu. Scheduling with Roll-back Constraints in High-Level Synthesis of Self-Recovering ASICs. In *Proc of 22nd FTCS*, pages 519-526, July 1992.
- [7] J. Klion. *Practical Electronic Reliability Engineering*. Van Nostrand Publishers, 1992.
- [8] K. S. Lin. Trends of Digital Signal Processing in Automotive Electronics. In *Proc of International Congress on Transportation Electronics*, pages 65-76, 1988.
- [9] A. Orailoglu and D.D. Gajski. Flow Graph Representation. In *Proc of the 23rd DAC*, pages 503-506, 1986.
- [10] A. Orailoglu and R. Karri. A Design Methodology for the High-Level Synthesis of Fault-Tolerant ASICs. *VLSI Signal Processing*, 5:417-426, October 1992.
- [11] D. P. Siewiorek and R. S. Swarz. *The Theory and Practice of Reliable System Design*. Digital Press, 1989.
- [12] Y. Tamir and M. Tremblay. High-Performance fault-tolerant VLSI systems using Micro Rollback. *IEEE Trans on Computers*, 39(4):548-554, 1990.
- [13] M. M. Tsao and D. P. Siewiorek. The Design of C-fast: A Single Chip Fault-Tolerant Microprocessor. In *Proc of 12th FTCS*, pages 63-69, 1982.
- [14] J. S. Upadhyaya and K. K. Saluja. Rollback and recovery strategies for computer programs. *IEEE Trans on Software Engg.*, 37:546-556, 1986.
- [15] M. M. Yen, W. K. Fuchs, and J. A. Abraham. Designing for Concurrent Error Detection in VLSI: Application to a Microprogram Control Unit. *IEEE Journal of Solid State Ckts*, 22(4):595-605, 1987.