

# Flexible Arithmetic Components for Area-Efficient Fault Tolerance

Vinu Vijay Kumar and John Lach  
University of Virginia

In this paper, we present a design technique for providing low area overhead fault tolerance in datapath dominant arithmetic circuits. Our method utilizes efficient flexible components capable of executing multiple arithmetic operations of the kind that occur frequently in digital signal processing functions. The system is synthesized for minimum area and delay with a hybrid library containing both fixed and flexible components. When a component fails, the operations are rescheduled to be implemented on the still-functional components, often at the cost of increased latency (i.e. number of control steps). However, the combined use of fixed and flexible components provides significant area benefits (or reliability enhancement for a given area), saving between 30% and 45% for a variety of DSP filters compared to traditional redundancy-based fault tolerance methods.

In addition, the scheduling flexibility provided by the runtime re-programmable flexible component ensures that the modified schedule is reasonably efficient (in terms of latency) compared to the original schedule. We introduce novel scheduling and allocation strategies that explore the design space at the behavioral-level in order to ensure that a successful adapted schedule with minimum latency is found. The delay penalty is equal to or less than that of existing techniques.

For such a technique to be effective, the benefits provided by the existence of flexible components cannot be outweighed by the penalties introduced by the component, such as the area, delay, and power penalties commonly associated with general-purpose reconfigurable arrays. The flexible components we have developed are primarily fixed logic but contain small amounts of application-specific programmable logic and interconnect (e.g. SRAM-based lookup tables (LUTs), SRAM-gated pass transistors, multiplexors (MUXes), etc.). Finely integrating this small amount of programmability at the gate level provides the necessary flexibility for multi-operation arithmetic components without the penalties of general-purpose reconfigurable arrays.

## Background

System reliability is rapidly becoming a dominant design issue, as increasingly complex hardware systems are used for mission critical applications in ever more demanding environments. System reliability is nowadays as important a metric as traditional metrics such as area, power, and speed for evaluating system efficiency. Built-In-Self-Test (BIST) techniques combined with some form of redundancy are the most popular for increasing reliability [Jha93]. These techniques are especially useful in military and aerospace systems that require zero downtime or are inaccessible for repairs [Avi71].

The large area overhead associated with having a redundant unit for every component has led to an exploration of other techniques. One approach is to use just one redundant unit per operation type, rather than providing a redundant adder, for example, for every adder in the system [Guerra98]. Another technique is to use programmable microcode to reschedule systems with faulty components [Kar93]. Dataflow graph are initially scheduled under timing and resource constraints. By relaxing the timing constraints and using graph transformations, a stricter resource constraint (imposed, for example, by the loss of a component due to a fault) can be met. These techniques are limited by the inherent inflexibility of the computational elements in the system. The modified schedules are often very inefficient, and even a single redundant unit per operation type imposes a large area overhead. The technique presented in this paper shows better results than these methods due to the increased system flexibility provided by the use of flexible arithmetic components.

## Flexible Arithmetic Components

The flexible units serve as more than just redundant units in the system; they are an active part of circuit. Flexible units are built inserting small amounts of programmable logic and interconnect into otherwise fixed-logic arithmetic components. This enables components to perform more than one operation with significantly less area overhead than a general-purpose reconfigurable array or ALU. Given that only small amounts of programmable logic and interconnect are used, the operations to be combined should have inherent similarities and sub-structures, such as adders and multipliers.

The flexible unit detailed in this paper can be configured to implement addition, subtraction, and multiplication on fixed-point numbers as well as bit-wise comparison. The detailed description will be provided in the full-length paper. From synthesis results, the area ratios for a 4-bit implementation were found to be:

Adder/Subtractor: 1.44X                      Multiplier: 4.5X  
Limited flexible unit (Adder/Subtractor + Multiplier): 4.81X  
Full flexible unit (Comparator + Adder/Subtractor + Multiplier): 5.31X  
where X is the area of a Comparator, the smallest component in the system.

For comparison, the multiplier implemented on a typical FPGA would be approximately 126.15X in size. In terms of delay, the full flexible unit is about 26% slower than the corresponding fixed unit, but the overall system latency can often be lower due to the schedule flexibility the unit provides.

Consider the simple dataflow graph shown below.

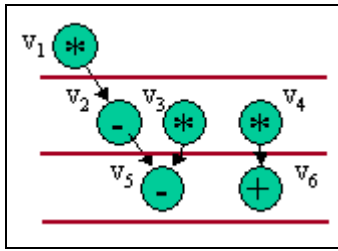


Figure 1: Original dataflow graph

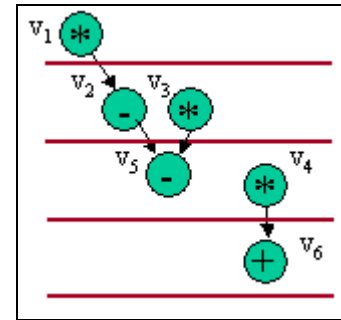


Figure 2: Dataflow after adder fails

Figure 1 shows the original dataflow graph, which is split up into 3 control steps (the minimum due to the data dependencies). An implementation using only fixed logic components would require 2 adders and 2 multipliers, for an area cost of 11.88X. With a flexible unit to execute nodes 4 and 6, an implementation in the same 3 steps would require just 1 adder, 1 multiplier, and 1 flexible unit for an area cost of 10.75X, a savings of 9.5% from the other implementation.

Now consider the situation when an adder fails. Current techniques would modify the schedule so that only a single adder is required, as in Figure 2. Notice the 1 step penalty in latency. Both the fixed logic implementation and the flexible component implementation can realize this augmented schedule. Similar results occur when a multiplier fails.

A high-level scheduling and allocation methodology has been developed for implementing this approach to fault tolerance. The well-known force directed list scheduling algorithm [Par01, Pau89, Ver95] is used for scheduling the base graph, with a modified force calculation altered to account for flexible components. A novel allocation algorithm follows scheduling to insert the flexible components into the system. Graphs corresponding to various component failure scenarios are then generated to have all recovery schedules at hand should a component fail. Simple constrained list scheduling can be used to schedule these modified graphs under the module constraints in each scenario. The presence of flexible components ensures that a successful minimum latency schedule is always found. The algorithms will be described in greater detail in the full paper.

## Results

These flexible component design, scheduling, and allocation techniques have been applied to DSP examples from the high-level synthesis literature. The circuits tested contain up to 241 operations and 121 time steps. Preliminary results are presented for circuit implementations with the minimum number of steps (as defined by the critical path) with the smallest area. Implementation areas are compared for redundancy based on a single redundant unit for each operation type. The reliability calculations are made under the assumption of a gate failure probability of .001. The results illustrate the large area and reliability benefits obtained by this approach.

Table 1: Area and Reliability Results

CIRCUIT	schedule without flexible unit				schedule with flexible unit					area savings	reliability increase
	# add	# mult	# comp	area	# add	# mult	# comp	# flex	area		
ELLIP	4	3	0	19.26	1	0	0	2	10.62	44.86%	58.10%
EDGE	2	3	0	16.38	0	1	0	1	9.31	43.16%	49.40%
ARFILT	3	5	0	26.82	0	1	0	3	18.93	29.42%	54.90%
FIRFILT	3	3	0	17.82	1	1	0	1	10.75	39.68%	49.40%
DIFFEQ	3	3	2	19.82	1	1	0	1	11.25	43.24%	67%

## References

- [Avi71] Avizienis, A., et al., "The STAR (Self-Testing And Repairing) computer: An investigation on the theory and practice of fault-tolerant computer design," *IEEE Transactions on Computers*, Vol. 20, No. 11, pp. 1312-1321, June 1971.
- [Jha93] Jha, N.K., and Wang, S.J., "Design and synthesis of self-checking VLSI circuits," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 879-887, June 1993.
- [Kar93] Karri, R., and Orailoglu, A., "High-level synthesis of fault-secure microarchitectures," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 429-433, June 1993.
- [Par01] Park, S., and Choi, K., "Performance-driven high-level synthesis with bit-level chaining and clock selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 199-212, Feb. 2001.
- [Pau89] Paulin, P.G., and Knight, J.P., "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661-679, June 1989.
- [Ver95] Verhaegh, W.F.J., et al., "Improved force-directed scheduling in high-throughput digital signal processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 945-960, Aug. 1995.