Fault Tolerant Asynchronous Adder through Dynamic Self-reconfiguration

Song Peng and Rajit Manohar
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, USA
{speng,rajit}@csl.cornell.edu

Abstract

This paper presents a systematic method for the design of a reconfigurable self-healing asynchronous adder. We propose a graph-based model for the design of a fault-tolerant linear array with external inputs and outputs with a minimum number of spare resources. A K-fault-tolerant asynchronous adder design is presented based on this analysis, together with the necessary support logic for dynamic self-reconfiguration. Experimental evaluations show that our method incurs both low hardware cost and small performance overhead compared to traditional approaches to fault-tolerance.

1 Introduction

The continuous advances of microelectronics leads to substantial reduction in both transistor dimensions and power supply voltages, which makes circuit become faster and consume less active power. However, the shrinking transistor size makes the circuit more sensitive to defects in fabrication processes such as contaminants [12]. Smaller device size coupled with increased power densities also becomes threatening the nearly unlimited lifetime reliability standards that customers have come to expect, leading the onset of significant lifetime reliability problems [7]. Thus, hard-error tolerant designs, which improve both fabrication yield and lifetime reliability, are becoming an important issue in modern VLSI systems.

With higher clock frequency, decreased feature sizes and increased transistor counts, clock distribution and wire delays present a growing challenge to the designers of singly-clocked, globally synchronous systems. It is becoming more and more difficult and expensive to distribute a global clock singal with low skew throughout a processor die. On the other hand, asynchronous circuits do not suffer such problem since they do not have a global clock. This fact forces industrial researchers to eventually abandom singly-clocked globally synchronous systems in favor of asynchrony [6]. Therefore, asynchronous design is expected to be a popular topic in future microprocessors.

One important class of asynchronous circuits, which we consider in this paper, is quasi delay-insensitive (QDI) circuits. They are an interconnection of logic gates without any clock signal for sequencing, and operate correctly regardless of gate delays [17]. A QDI system is constructed as a collection of concurrent hardware modules (called *processes*) that communicate with each other through message-passing channels. These messages consist of atomic data items called *tokens*, which are usually multi-rail encoded [17]. Each process can send and receive tokens to and from other processes through one-to-one communication by means of handshake protocols [17]. Due to the lack of global clock and multi-rail encoded data communication, QDI circuits have potential to achieve self-checking and halt the circuit in presence of failures [3, 5].

Although there is wealth of research on fault tolerant clocked logic designs, little attention has been paid to asynchronous circuits. The absense of clock signals means that a faulty asynchronous circuit might exhibit problems that would not normally arise in a clocked logic, which makes competent fault tolerant techniques for synchronous systems ineffective or inefficient [3]. Thus, new methods have to be explored to achieve fault tolerance in asynchronous circuits.

To reduce design complexity, a systematic way to build a fault tolerant system, and in particular, a fault tolerant asynchronous system, is to make each components fault tolerant. The design of a fault tolerant datapath is an important step in constructing a fault

tolerant microprocessor. Many datapath blocks such as adders, shifters, etc, can be modeled as a linear array or a collection of linear arrays, given that either data or control propagates linearly through them. The design of a self-healing adder shows a typical example to achieve fault tolerance in such blocks, and provides the basis for building a fault tolerant datapath.

The goal of this paper is to exploit inherent fail-stop behavior in asynchronous circuits to construct a self-healing adder with both small hardware cost and performance overhead in a systematic way, and thus take an important step toward the design of a fault tolerant asynchronous processor.

Two metrics are commonly used to evaluate a fault tolerant system, *hardware overhead* and *performance overhead*. Since it is often difficult to reduce both overheads in reality, designers often trade off one metric with the other [11, 15]. The most widely used approach to achieving fault tolerance is hardwired duplication such as NMR [18]. However, power, area and yield pressures cause less scope for full duplications today [1]. Another approach is to introduce reconfiguration overhead but less hardware redundancy to maintain functionality in presence of failures. This approach can be conveniently described as a graph problem [14]. Suppose a graph G respresents the topology of a multiprocessor system, an interconnection network, or a VLSI circuit. We say another graph G' is a *K-fault-tolerant* (*K-FT*) graph of the *target graph* G if G is isomorphic to a subgraph of the graph derived by deleting any K nodes and all their incident edges from G'. Since every edge fault is a part of some node fault, G' can tolerate both node faults and edge faults. In the remaining of this paper, we mean *node fault tolerant* wherever we say *fault tolerant*.

In this paper, we propose a near-optimal graph model for K-fault tolerant linear array with external inputs and outputs (called *open linear array*), and examine the case with a minimum number of spare resources (Section 2). Based on this graph model, we present a systematic method to design a self-healing asynchronous adder: a K-fault tolerant open linear array with the support logic for dynamic self-reconfiguration (Section 3). By utilizing fail-stop potential of asynchronous logic, we show that this proposed design method achieves small hardware cost, high performance and low power consumption compared to hardwired duplication approaches such as NMR [18], TSTMR [15], QTR [11], etc (Section 4). Meanwhile, we analyze the relationship between reconfiguration complexity and spare resource cost, and show how to reduce total hardware overhead by choosing an appropriate linear array size. Section 5 reviews the related work, and Section 6 draws the conclusions.

2 Fault Tolerant Open Linear Arrays

A module that is part of a larger system has a set of internal components V which are connected to each other, as well as connections to a set of external components V_e. The graph of interest when analyzing the module contains *internal* edges from $V \times V$, and *external* edges from $(V \times V_e) \cup (V_e \times V)$. We say that a graph $G = (V \cup V_e, E)$ is *closed* if $E \subseteq V \times V$; otherwise, the graph is said to be *open*.

What makes the construction of a fault-tolerant model for a open graph challenging is the fact that external inputs to the graph are not interchangeable. This makes nodes in a open graph heterogeneous. There is a wealth of research on fault tolerant graph models for closed linear arrays [9, 13, 14, 19, 20]. However, a direct application of these results requires ensuring that every external vertex V_e has an edge to every internal node.

We proposed two solutions to this problem that minimize the amount of replication required for external edges in [2]. This section briefly describes one solution with a minimum number of spare nodes, as the self-healing asynchronous adder in this paper is built on this graph model. More details can be found in [2]. For the remainder of this section, we use the term linear array to mean a open directed linear array.

Let $u_1, u_2, \ldots, u_N$ be the internal nodes in the linear array, and let $e_1, \ldots, e_N$ be the external nodes. The edges are of the form (u_i, u_{i+1}) for $1 \leq i < N$ and (e_i, u_i) for $1 \leq i \leq N$. For the graph to be K-FT, there must be at least K spare nodes. We name these nodes $u_{N+1}, \ldots, u_{N+K}$. Also, If a node in the array u_i connects to an external node v, this connection must be replicated to at least K other distinct nodes $u_{i_1}, u_{i_2}, \ldots, u_{i_K}$ that differ from u_i; otherwise the graph will not be K-FT. Next we consider the internal edges in the linear array, assuming that we have a minimum number of spares as well as minimum external edge replication. For simplicity in what follows, we only focus on the internal edges, and we assume that $N \geq 2$.

We define a *tail candidate* to be any node that can be at the end of the linear array after K nodes are removed; a node that can never be at the end of the array is a *non-tail candidate*. Note that since a minimum number of external edges are introduced, there must be exactly $K + 1$ tail candidates.

Let $P_{N,K}$ be a K-fault tolerant open linear array with minimum spares and external edges. Let P_N is the target array.

Any non-tail candidate of $P_{N,K}$ can't be at the end

of remaining linear array after any K nodes are removed. In other words, there must be an outgoing edge from that node after removing any other K nodes. Thus, the out-degree of each non-tail candidate of $P_{N,K}$ is at least $K+1$. Moreover, at most one tail candidate can have zero out degree. Otherwise, no linear array can be found in the remaining graph after removing all the tail candidates with outgoing edges. Therefore, we can conclude that an optimal K-FT open linear array with minimum spares and external edges must have $(N-1)\times(K+1)+\Omega(K)$ internal edges.

We can construct a K-FT graph $P_{N,K}$ based on the linear array as follows. (i) K spare nodes u_{N+1}, ..., u_{N+K} are introduced; (ii) External edges are added. For $1 \le i \le N$, we introduce replicas (e_i, u_{i+1}), ..., (e_i, u_{i+K}) of edge (e_i, u_i); (iii) Internal edges are added. For each node u_i $(i < N)$, K replicas (u_i, u_{i+2}), ..., (u_i, u_{i+K+1}) of edge (u_i, u_{i+1}) are introduced. Finally for each node u_i $(N \le i < N+K)$, $N+k-i$ edges from u_i to u_{i+1}, ..., u_{N+K} are introduced. The proof that $P_{N,K}$ is K-fault tolerant can be found in [2].

$P_{N,K}$ has a minimum number of spares, as well as minimum external edges. The number of internal edges is $(N-1)\times(K+1)+K\times(K+1)/2$. Since $K \ll N$ for most real fault-tolerant designs, in practice we should have a number of internal edges that is very close to the loose lower bound. Moreover, the construction $P_{N,K}$ can be applied to general open linear arrays, where each node in $P_{N,K}$ represents a subgraph, and the external edges can correspond to possibly replicated external input and/or outputs.

As an example, Figure 1 shows the construction for $P_{4,2}$. In $P_{4,2}$, 2 spare nodes (shaded), 8 external edges (bold), and 9 internal edges (bold) are added in order to tolerate any 2 node faults.

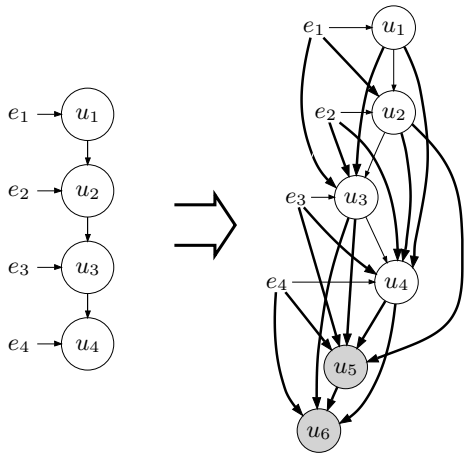


Figure 1. Fault tolerant array construction example.

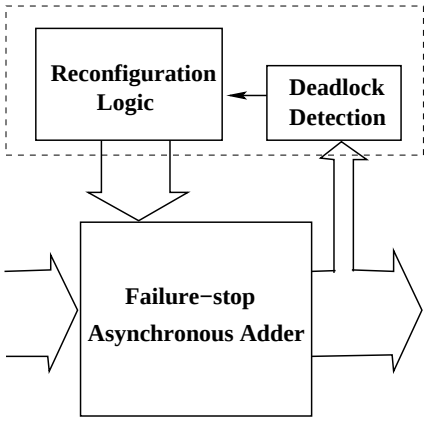


Figure 2. Block diagram of reconfigurable self-healing asynchronous adder

3 Self-healing Asynchronous Adder Design

In this section, we present the design of a self-healing asynchronous adder with respect to any K faults, using the construction outlined in Section 2. A block diagram of the self-healing adder is shown in Figure 3. It is composed of a fail-stop asynchronous adder, combined with deadlock detection and online reconfiguration logic.

The asynchronous adder is built based on the K-fault tolerant linear array in Section 2 and extra logic is added to achieve fail-stop. When the adder stops in presence of failure (due to fail-stop logic), the deadlock detection circuit captures the deadlock and activates online reconfiguration logic. Reconfiguration logic then exhaustively tries different reconfigurations until it finds a workable one. In order to achieve self-healing, the reconfiguration logic and deadlock detection circuits have to be fault free.

Fault Model. We investigate single stuck-at fault ($SSAF$) in a 1-bit adder cell and thus multiple stuck-at faults are allowed in an N-bit adder. According to stuck-at fault model, a circuit line is stucked-at one or zero if it is disconnected from any other circuits' wires and connected to the power supply or ground respectively. Although stuck-at fault model is simple and is not accurate to represent all possible fault cases, it can cover most real fabrication defects [1] and is effective for modeling many other permanent faults and unrecoverable failures [4]. Thus, a stuck-at-fault-tolerant system has the potential to achieve high reliablity.

Implementing Fail-Stop Behavior. Because asynchronous circuits use multi-rail encoding to represent data bits and use handshake prtocols to communicate data between different processes, those circuits have the potential capability of self-checking [3, 5].

For high performance and reasonable transistor count, a 1-bit QDI adder cell is usually implemented in terms of precharge half-buffer (PCHB) template [16] (shown as Figure 3). In this template, each variable X is dual-rail encoded (X^f, X^t) with an active-low acknowledge (X^e). Validity and neutrality of the inputs and the output are checked (NOR gates) and synchronized (C-element) in Figure 3(a), generating corresponding enable signal en and input acknowledge L^e. Data output is computed or reset in Figure 3(b), depending on the enable signal en and output acknowledge R_o^e. Details about the handshake sequence by PCHB circuit can be found in [16].

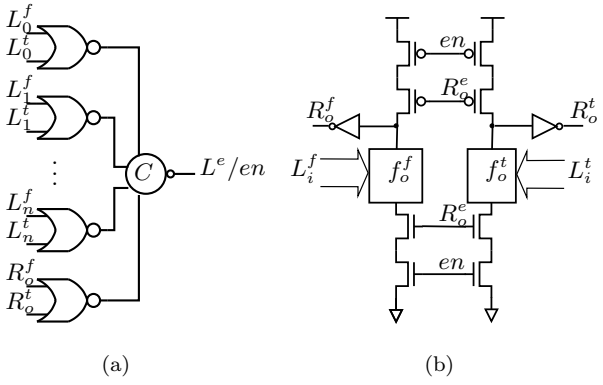


Figure 3. Template of precharge half buffer.

Because handshake completion relies on both up and down transitions on all signals in Figure 3(a) and data computation is decided by input data rails to pull-down stacks, it can be proved that any failure by single stuck-at fault in the PCHB circuit either deadlocks the circuit or throws an illegal encoded output ($R_o^f R_o^t$='11') [5]. By checking each output channel and blocking the output acknowledge in presence of any illegal data token, we can easily make the circuit also deadlock if any illegal data token is generated from current process. Therefore, an N-bit adder in terms of PCHB circuits can achieve fail-stop with respect to stuck-at fault(s) as long as single stuck-at fault in a 1-bit cell.

Deadlock Detection. System deadlock is recognized by a deadlock detector, which monitors channel activity. The deadlock detector works in the following way. At any time, if a transition occurs on the data channel, a timer is started (implemented as a delay line [10]). The deadlock detector waits for the next valid protocol state to occur. If it does not occur for a large amount of time, it assumes that the circuit has deadlocked. Note that there are some states where the circuit can wait for its environment. In these states, a timer is not set.

Reconfiguration. The reconfiguration logic is activated by the deadlock detector, and reconfigures the asynchronous adder to a different configuration so that faulty nodes won't be used any longer. The fail-stop asynchronous adder has to be changed in order to achieve self-healing with respect to any K faults through online reconfiguration. (i) K spare nodes are added to the adder and the whole N-bit adder topology is a fault tolerant linear array of Section 2, where each node is a C-bit adder cell ($C \leq N$). Each external edge represents the non-carry channels of the adder cell, and each internal edge represents carry propagation channels between adjacent adder cells. (ii) In order to achieve dynamic reconfiguration, there is a pass-gate on the wires for each edge, whose control input comes from external reconfiguration logic.

Generally speaking, there are two methods to achieve such reconfiguration. One is to locate faults and use a new configuration which excludes those faulty nodes directly. Although such a system is fast in terms of fault recovery time, fault location is complicated because non-faulty nodes will be also deadlocked by faulty nodes, resulting in complex location logic and large hardware overhead. Another possibility is to let the reconfiguration logic try all possible configurations until it finds a workable one. Although this will prolong fault recovery time, we save the fault location logic and the total hardware overhead can be actually reduced. Also, because fault rarely occurs, longer fault recovery time has little impact to system performance. For the purpose of this paper, we used this approach with a self-incrementer used as a state machine and with configuration outputs derived from the value of the incrementer. The block diagram of reconfiguration logic is shown in Figure 4.

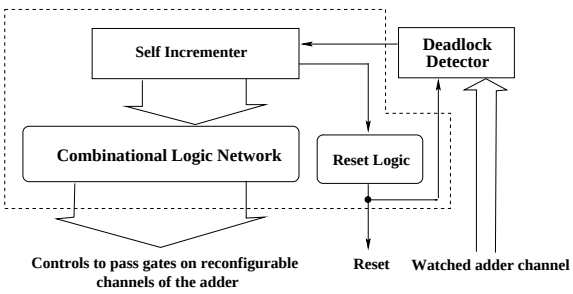


Figure 4. Reconfiguration logic diagram.

The system reconfiguration is achieved as follows. Whenever the adder deadlocks, the deadlock detector activates the self-incrementer by starting the handshake, and the latter increments itself and changes to the next state. The combintaional logic network is used to set the configuration outputs according to current incrementer output. Consequently, all pass-gate controls are updated accordingly, setting up new connections. Meanwhile, the incrementer also generates a local reset signal through reset logic (timing assumptions have to be made in reset stage for reset signal propagation), which re-initializes the adder to restart the handsking and resets the deadlock detector, making the system ready for re-issued computation. The above procedure repeats if the system deadlocks again, and another different configuration will be picked up.

There are $\binom{N+K}{K}$ possible configurations for a K-FT N-node array, corresponding to the possible fault locations. Thus, the incrementer output needs $M = \lceil log_2\binom{N+K}{K} \rceil$ bits (when $K \ll N$, $M \simeq Klog_2N$). Since each state of self-incrementer represents a choice of K nodes, the Boolean equation for each configuration output can be derived directly from the remaining graph after removing those K faulty nodes. A number of logic terms are shared between the Boolean equations of different configuration outputs so that the configuration logic can be further simplified by reusing common subexpressions.

4 Evaluation

We evaluate the self-healing asynchronous adder in terms of hardware cost, performance and power consumption, and compare this design method with traditional methods to achieve fault tolerance, like NMR, TSTMR [15], etc. We use M to denote the number of nodes in the original adder, K to denote the number of faults to be tolerated, C to denote the bit-width of the adder cell represented by each node, and the adder is N-bit wide, where $N = M \times C$.

4.1 Hardware overhead

The cost of a circuit in terms of the amount of hardware necessary is estimated by its transistor count. We define *hardware overhead* to be $100\% \times (C_{ft} - C_o)/C_o$, where C_o is the hardware cost of the baseline adder, C_{ft} is the hardware cost of the self-healing adder, which includes spare resources and reconfiguration logic.

The increased cost of a self-healing adder is due to the additional complexity of the implementation described in Section 3. While the complexity of the adder core can be estimated easily from the arguments in Section 2, it is harder to describe the transistor count required for the reconfiguration logic in analytical terms. We wrote a program to automate the design of combinational logic part of the reconfiguration logic. The program performs Boolean expression simplification for each configuration bit, attempting to share terms across all the outputs to reduce hardware cost. The program also ensures that no term in the logic expression requires more than four transistors in series for its implementation so that it can be easily implemented in CMOS without further gate decomposition.

We analyzed 8-bit, 16-bit, 32-bit and 64-bit adders that are 1-FT, 2-FT, 3-FT and 4-FT with nodes of 1-bit, 2-bit, ..., $(N/2)$-bits respectively. The corresponding hardware overheads are shown in Figure 5.

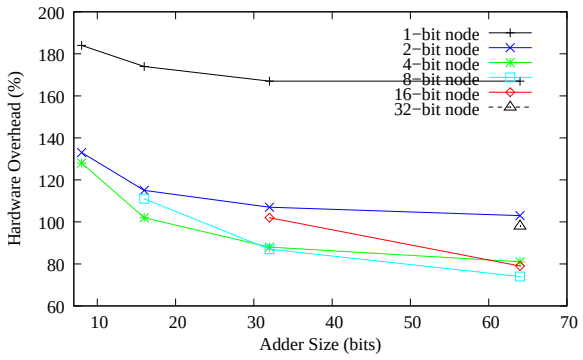
From Figure 5, we can draw several conclusions. First, hardware overhead can be reduced dramatically if the appropriate node size is chosen. Generally speaking, given an adder size N, larger node size C saves more hardware overhead when K increases, because the simplied graph due to less number of nodes, results in simpler configuration logic. Second, given the number of nodes M, larger adder size N results in less hardware overhead, because the hardware cost of reconfiguration logic becomes less compared with baseline adder. As the number of configuration bits grows (especially in the case of a 1-bit node), the complexity of the adder core itself is dwarfed by the reconfiguration logic.

The minimum hardware overheads of different node sizes from Figure 5 are summarized in Table 1. The overhead is reported as a percentage, with the associated node size reported in parentheses. 'x-FT' refers to self-healing adder with respect to x faults.

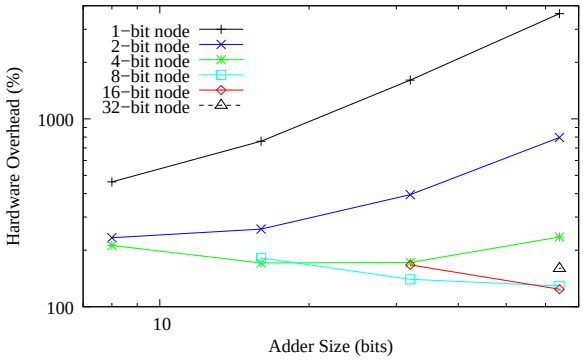
Size	1-FT	2-FT	3-FT	4-FT
8-bit	128% (4-bit)	212% (4-bit)	307% (4-bit)	387% (4-bit)
16-bit	102% (4-bit)	171% (4-bit)	258% (8-bit)	326% (8-bit)
32-bit	87% (8-bit)	140% (8-bit)	211% (8-bit)	296% (16-bit)
64-bit	74% (8-bit)	124% (16-bit)	178% (16-bit)	245% (16-bit)

Table 1. Minimum hardware overheads of self-healing adders.

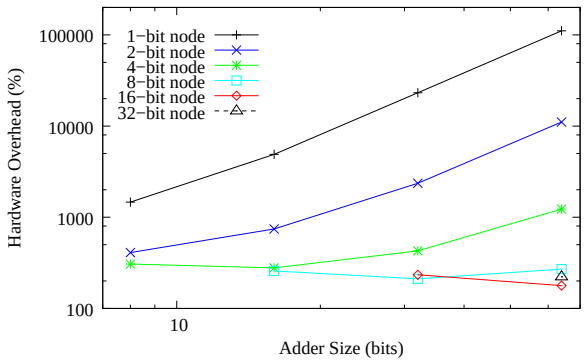
Let $S_{N,K}$ be the node size to achieve minimum hardware cost given the N and K. As mentioned before, the increased node size simplfies the fault tolerant linear array model, resulting in simplified reconfiguration



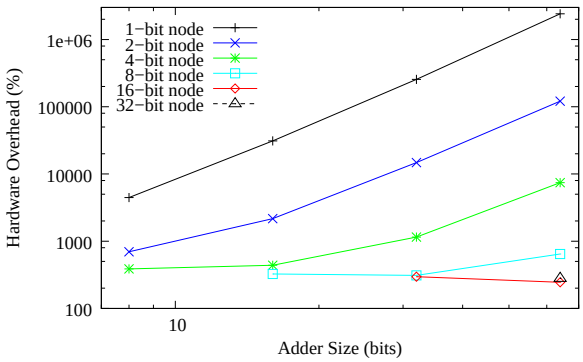
(a) Hardware overhead of 1-fault self-healing adder



(b) Hardware overhead of 2-fault self-healing adder



(c) Hardware overhead of 3-fault self-healing adder



(d) Hardware overhead of 4-fault self-healing adder

Figure 5. Fault tolerance hardware overhead analysis.

logic overhead. Thus, $S_{N,K}$ increases when N or K increases. On the other hand, increased node size increases the hardware cost for spares. Consequently, given K, the ratio of optimal node size to adder size, $S_{N,K}/N$ decreases with larger N so that the increase of hardware cost for spares can be covered by the reduction in reconfiguration logic.

Compared with the traditional NMR method where entities are simply replicated, whose hardware overhead is at least $2K \times 100\%$, our design has much lower hardware overhead even when the reconfiguration logic is accounted for. Table 2 shows the hardware overhead if the fault tolerant asynchronous adder is implemented using triple modular redundancy (TMR) [1].

The hardware overhead of this fault tolerant design is even comparable to the time redundancy method

Size	8-bit	16-bit	32-bit	64-bit
TMR	323%	315%	311%	309%
TSTMR	115%	99%	76%	72%

Table 2. Hardware overhead of TMR, TSTMR (synchronous).

such as TSTMR in synchronous system, whose hardware overhead is reported in [11] and shown in Table 2. Note that this assumes that the clock network is functional, and applying this technique to asynchronous circuits will require additional overhead that is not reported in Table 2.

4.2 Performance overhead

The reconfiguration logic and spares in our design are not on the critical path, the performance doesn't depend strongly on K. We used HSPICE to simu-

[1] To apply TMR to QDI circuits is non-trival: timers as well as non-negligible self-reconfiguration logic has to be added to voter. We omit those components and only investigate voter core here so that the reported hardware cost, performance and power consumption are optimistic.

late the fault tolerant adders with all configurations shown in Table 1, and compared the throughputs with corresponding baseline adder and TMR adder, which is shown in Figure 6. The HSPICE simulation uses TSMC 0.18um technology at 25°C.

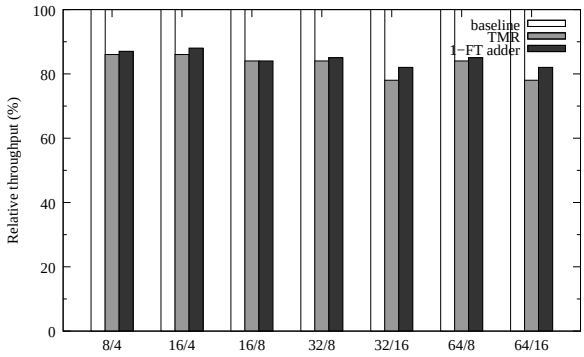


Figure 6. Comparison of throughputs

In Figure 6, X/Y refers to X-bit adder with Y-bit node. The performance degradation in this fault tolerant design is no more than 20%: the throughput is even a little higher than the TMR approach, and much better than TSTMR [15] and QTR [11] approaches, which reduces performance by a factor of 3 and 4 respectively. Note that the throughput of NMR will become much worse if K increases, because the majority voter logic becomes much more complicated and further limits the throughput. However, this situation does not arise in our reconfigurable fault tolerant design due to the fact that no extra logic other than a reconfigurable acknowledge circuit stays on the critical path. Thus, given a N, the performance overhead will not increase significantly when K changes.

4.3 Power overhead

Power consumption of a circuit can be divided into two parts: static power consumption and dynamic power consumption. A simple estimate of the static power can be obtained from the transistor count. From Table 1, the static power overhead of reconfigurable fault tolerant design is much less than that of the NMR approach and comparable to that of TSTMR.

Dynamic power consumption accounts for the majority of total power consumption today. Because there is no switching activity in both reconfiguration logic and spare hardware, the dynamic power overheads of this recofigurable adder only comes from the augmentation pass-gates. We used HSPICE (using TSMC 0.18um technology at 25°C) to simulate the fault tolerant adders with all configurations shown in Table 1 and compared the dynamic power consumption with the baseline adders and TMR adder. The results are shown in Figure 7.

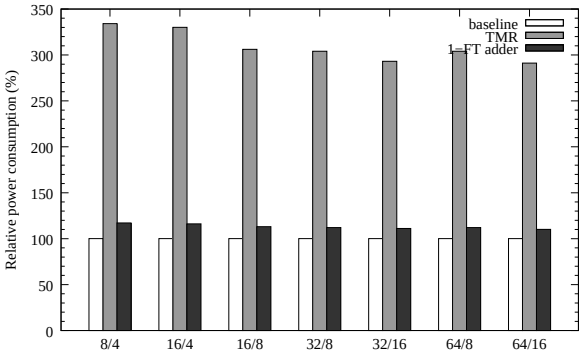


Figure 7. Comparison of power overheads

Because all replicas and voters are working all the time, the TMR approach incurs a large power overhead. However, the major power overhead of the proposed design comes from the pass-gates and the extra power consumption is small (less than 20%).

4.4 Fault recovery time

Fault recovery time is decided by the number of configurations the system has tried before it finds a workable one. It takes system τ_d before it decides that current configuration doesn't work and switches to another, where τ_d is the deadlock detection time. Therefore, the worst fault recovery time is that system has tried all possible configurations and finds the last one workable, which is $\binom{N+K}{K} \times \tau_d$. Normally, τ_d is several microseconds, resulting in a fault recovery time on the order of tens to hundreds of microseconds for low values of K. This time can be acceptable in practice as the time required by the system to reconfigure itself to handle failures.

5 Related Work

There is a wealth of research on graph-theoretical approach to fault tolerance through reconfiguration. Hayes [14] introduced the concept of K-FT graphs and proposed optimal K-FT graphs for linear array and circle. Alon et al. [9] constructed fault tolerant graphs for (undirected) linear arrays in a more general way. Ajtai et al. [8] presented a universal method based on probabilistic arguments for arbitrary fault tolerant graph. Haray et al. [13] discussed the design of optimal K-edge fault tolerant graphs of paths, circles and n-dimensional hypercube. Zhang [20] proposed a new

fault tolerant linear array to trade off maximum node degree with more spares, and a better construction was subsequently developed by Yamada et.al [19]. However, the graphs studied in the aforementioned work are closed. In other words, there is no external input or output with respect to the graph, because those graphs were used to model interconnection networks of parallel machines. Thus, the results cannot be applied to non-closed graphs without any change.

Substantial work has also been done on fault tolerant designs. A commonly-used method is N-modular redudancy (*NMR*), which was proposed by Von Neumann [18]. In order to tolerate K faults, $2K+1$ replicas are added and a majority voter is used to determine the final result. This approach becomes complicated and causes significant slowdown with increased K. In order to reduce the hardware overhead, The authors in [11, 15] used time redundancy to achieve 1-fault tolerance by trading off hardware cost for performance overhead. For example, QTR method in [11] reduces the extra hardware cost down to 32% while reducing performance by a factor of 4. However, none of these methods is able to achieve fault tolerance with both low hardware cost and small performance overhead.

6 Conclusion

In this paper, we proposed a graph model for fault tolerant open linear array, and presented a systematic method of reconfigurable self-healing asynchronous adder design based on that model, with respect to stuck-at faults. We analyzed the relationship between reconfiguration complexity and spare resource cost incurred by different linear array size, and showed how to reduce total hardware overhead by choosing a reasonable linear array size. Due to the inherent fail-stop property of asynchronous system, this reconfigurable fault tolerant design can achieve much lower hardware overhead compared with traditional NMR method and is even competitive with time redundancy method in synchronous logic. By keeping most of self-healing related logic away from the critical path, this reconfigurable design achieves small constant performance overhead, and consumes much less power compared with NMR. The key philosophy behind this design is to make self-healing system *cost-effective* by trading off prominent terms (hardware, etc) with less impactive terms (fault recovery time). Finally, this self-healing design method can also be applied to synchronous circuits, as long as failure detection logic is added.

References

[1] *International Technology Roadmap for Semiconductors*. Semiconductor Industry Association, 2004.

[2] S. Peng et al. Explicit constructions of fault-tolerant open linear arrays. Technical Report CSL-TR-2005-1044, Cornell University, 2005.

[3] C. LaFrieda et.al. Robust fault detection and tolerance in quasi delay-insensitive circuits. In *Proc. International Conference on Dependable Systems and Networks*, 2004.

[4] E. J. McCluskey et.al. Stuck-fault tests vs. actual defects. In *Proc. International Test Conference*, 2000.

[5] I. David et.al. Self-timed is self-checking. *J. of Electronic Testing: Theory and Applications*, 6(2), 1995.

[6] I. E. Sutherland et.al. Computers without clocks. *Scientific American*, 2002.

[7] J. Srinivasan et.al. The case for lifetime reliability-aware microprocessors. In *Proc. the 31st Annual International Symposium on Computer Architecture*, 2004.

[8] M. Ajtai et.al. Fault tolerant graphs, perfect hash functions and disjoint paths. In *Proc. IEEE Symposium on Foundations of Computer Science*, 1992.

[9] N. Alon et.al. Explicit construction of linear sized tolerant networks. *Discrete Math*, 72(1):15–19, 1988.

[10] N. R. Mahapatra et.al. Comparison and analysis of delay elements. In *Proc. the 45th Midwest Symposium on Circuits and Systems*, 2002.

[11] W. J. Townsend et.al. Quadruple time redundancy adders. In *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003.

[12] W. Kuo et.al. An overview of manufacturing yield and reliability modeling for semiconductor products. *Proceedings of the IEEE*, 87(8), 1999.

[13] F. Haray and J. P. Hayes. Edge fault tolerance in graphs. *Networks*, 23:135–142, 1993.

[14] J. P. Hayes. A graph model for fault-tolerant computing systems. *IEEE Trans. on Computers*, 25(9), 1976.

[15] Y. M. Hsu. *Concurrent Error Correcting Arithmetic Processors*. PhD thesis, The University of Texas, Austin, August 1995.

[16] A. M. Lines. Pipelined asynchronous circuits. Master's thesis, California Institute of Technology, 1995.

[17] A. J. Martin. Synthesis of asynchronous VLSI circuits. Technical Report CS-TR-93-28, California Institute of Technology, 1993.

[18] V. Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[19] T. Yamada and S. Ueno. Optimal fault-tolerant linear arrays. In *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[20] L. Zhang. Fault tolerant networks with small degree. In *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, 2000.