Self-Healing Asynchronous Arrays

Song Peng and Rajit Manohar[*]
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, USA

Abstract

This paper presents a systematic method for designing of a self-healing asynchronous array in the presence of errors. By adding spare resources in one of three different ways and forcing the asynchronous circuit to stall in case of failure, the specific self-reconfiguration logic is activated by a deadlock detector and the array circuit can be reconfigured around the faulty components and recover from errors automatically. Experimental evaluations show that this method requires less hardware cost, smaller critical circuit size, lower performance overhead and is more scalable than traditional NMR-based techniques.

1 Introduction

The continuous advance of microelectronics has led to a substantial reduction in both transistor dimensions and power supply voltages, helping VLSI circuits operate faster and consume less active power. However, technology scaling causes circuits to be more sensitive to defects in fabrication [3] and threatens the nearly unlimited lifetime reliability standards that we have come to expect [18]. The reduced amount of charge stored on circuit nodes also makes circuits more susceptible to transient faults [3]. Thus, fault tolerant design, which improves both fabrication yield and chip reliability, is once again becoming an important issue.

While there is a wealth of literature that examines fault tolerance in clocked logic [8], less attention has been paid to asynchronous circuits. The absence of clock signals means that a faulty clockless circuit might exhibit problems that would not normally arise in a clocked system [9], making existing fault tolerance techniques for synchronous systems ineffective or inefficient. For instance, the most widely used approach to achieving fault tolerance in clocked VLSI systems is the hardwired duplication-and-comparison method such as N-modular Redundancy (NMR) [8]. However, it is non-trivial to apply such duplication-and-comparison techniques to asynchronous logic without significant gate timing assumptions [19]. Unlike clocked systems where the outputs from all replicas can be sampled at the same time and thus easily compared against each other, the local handshake in asynchronous circuits makes it unclear when the non-directly related outputs are expected to match. In addition, faults in asynchronous logic may prevent the result from appearing on the output, permanently blocking the comparison procedure.

Besides hardwired duplication-and-comparison, another possible fault tolerance approach, which can be conveniently formulated as a graph problem [5], is to utilize self-checking and reconfiguration to maintain functionality in the presence of failures. Although this approach incurs fault detection and reconfiguration overheads as well as fault recovery time, smaller hardware redundancy and less power consumption make it an attractive defect/fault tolerance method [3]. Moreover, the absence of a comparison procedure makes this approach better suited for asynchronous circuits.

To reduce design complexity, a systematic way to build a reconfigurable fault tolerant asynchronous system is to make each of its components fault tolerant. In a digital VLSI system, many computation modules such as adders, array multipliers, FIR filters, etc, can be modeled as a linear array or a collection of linear arrays with external inputs and outputs, given that communication propagates linearly through them. Thus, the construction of a self-healing asynchronous array provides the basis for reconfigurable fault tolerant asynchronous VLSI design at fine-grained level.

The class of asynchronous circuits considered in this paper, are quasi-delay-insensitive (QDI). QDI circuits are designed to operate correctly under the assumption that gates and wires have arbitrary finite delay, except for a small number of special wires known as isochronic forks [12]. A QDI system can be taken as a collection of concurrent hardware modules (called *processes*) that communicate atomic data items (called *tokens*) with each other through one-to-one message-passing channels. The message-passing channels usually consist of data and acknowledge rails. The notion of causality and event-ordering is implemented in terms of handshake protocols on those channels [12].

The following contributions are made in this paper. First,

[*]E-mail: {speng,rajit}@csl.cornell.edu

we propose a general framework of reconfigurable fault tolerant design for asynchronous circuits, as well as the 2- and 3-Dimensional implementation methods (Section 2). Second, we develop three fault tolerant array models for this framework (Section 3) and present the construction of corresponding self-reconfiguration logic (Section 4). Third, we evaluate all the self-healing designs of different array models, and show that they result in smaller hardware cost, higher performance and lower energy overheads than traditional NMR method, as well as better scalability (Section 5). Fourth, we analyze the relationship between reconfiguration complexity and spare resource cost, compare the self-healing designs of different array models, and assess the advantages of each scheme (Section 5).

2 General Framework of Self-Healing Asynchronous Circuit Design

In this section, we propose a general framework of self-healing asynchronous circuits with respect to an arbitrary number of hard and soft errors, which is shown as Figure 1.

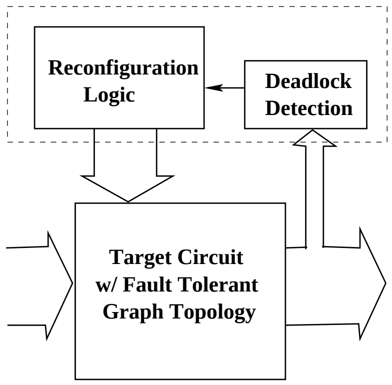

Figure 1. Block diagram of a reconfigurable self-healing asynchronous circuit.

The target asynchronous circuit is built on a K-fault tolerant graph model with spare resources. Pass gates, whose control inputs come from the reconfiguration logic, are added to the wires of graph edges to make the target circuit reconfigurable. Self-checking logic is added to the target circuit to achieve deadlock in the presence of failure (*fail-stop*). When the target circuit deadlocks, the deadlock detection logic recognizes this and activates the online reconfiguration logic, which reconfigures the target circuit around the faulty components. The computation restarts from the beginning or the last architectural checkpoint after the circuit has been reconfigured.

Since no extra circuitry other than self-checking logic and pass-gates is on the critical path, small performance overhead is expected in this reconfigurable fault tolerant design. Moreover, there is no switching activity in the reconfiguration logic when the target asynchronous circuit operates correctly, therefore low energy overhead is also anticipated. Unlike hardwired NMR method where the latency severely increases with large K due to the dramatically higher complexity of the voter, performance overhead of this fault tolerant design does not increase significantly with K because the number of gates being used for each configuration remains the same. Thus, this reconfigurable fault tolerant design is expected to scale well in terms of performance and energy overhead with respect to K.

In this framework, the reconfiguration logic and deadlock detection circuits must be fault-free to achieve fault tolerance. Thus, those circuits are critical (error-sensitive) and must be made highly reliable. With traditional 2D (2-Dimensional) integration technology, those circuits could be implemented using conservative layout design rules and large transistor sizing (even with thicker oxide). With recent 3D integration technology [2] where planar device layers are stacked in a three-dimensional structure and adjacent device planes can be connected by short and vertical wires, all the error-sensitive transistors can be placed onto a separate device layer which is fabricated with a robust/conservative (micron or submicron) technology, while the target circuits are placed onto another device layer with an aggressive (deep-submicron or nanometer) technology.

We choose a half-buffer based circuit template (called *precharge half buffers (PCHB)*) [10] as the target QDI circuit. A PCHB circuit can have multiple inputs and outputs, and it can be used to construct almost any pipelined QDI logic. For instance, the asynchronous MiniMIPS microprocessor [13] uses PCHBs for more than 90% of its circuits. Thus, implementing self-healing behavior in PCHB circuits takes an important step toward fault tolerance in general asynchronous logic. Similar to a precharge domino circuit in synchronous design, a PCHB circuit performs computations using pull-down (NMOS) networks, making it fast. In this circuit, each variable is usually dual-rail encoded with an explicit acknowledge. Validity and neutrality of the inputs and the output(s) are checked and synchronized (by C-elements), which generates the common acknowledge to all inputs and precharge/enable signal for data computation.

By adding separate data validity rail to each variable, replicating all explicit acknowledges and crosschecking between duplicated internal control signals, we developed a PCHB-based circuit template (called *FS-PCHB*) [15] which achieves fail-stop with respect to both hard and soft errors. Figure 2 shows the block diagram of a FS-PCHB circuit.

In Figure 2, the data computation is dual-rail encoded and implemented in terms of functions f^f and f^t. Output data validity R^v depends on input validity rails (L_i^v) and becomes valid only if all those rails are true. For each input/output variable X, the explicit validity rail X^v crosschecks the data rails X^t and X^f. There is a counterpart for any internal signal of FS-PCHB, and the circuit state

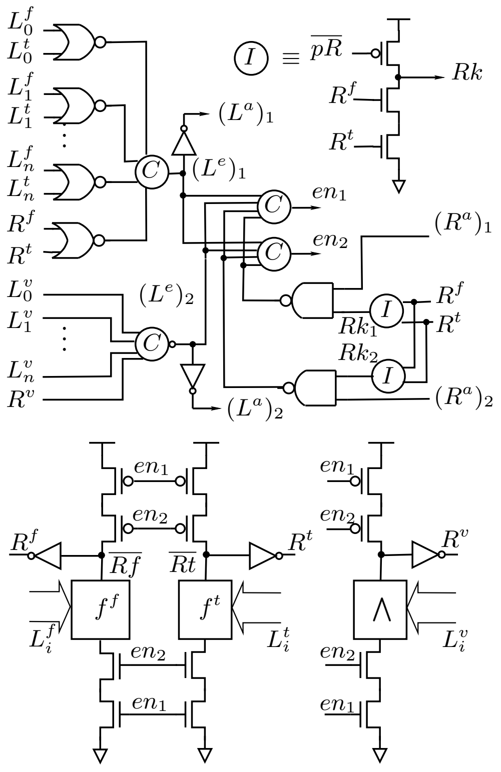

Figure 2. Fail-Stop Precharge Half Buffer (FS-PCHB).

will not change unless both signals match. Any illegally encoded ($'11'$) dual-rail output will reset Rk_1 and Rk_2, blocking output acknowledges permanently. It can be proved that any failure by single stuck-at fault or single event upset in a FS-PCHB circuit causes the circuit to deadlock. Further details of the construction can be found in [15].

3 Fault Tolerant Array Models

In the self-healing design of Figure 1, the target fail-stop QDI circuit is built on a K-fault tolerant (K-FT) array so that faulty components can be replaced by workable spare resources. This section develops three fault tolerant array models, each with different spare resource cost, maximum degrees (node fanouts) and reconfiguration overhead.

3.1 Preliminaries

Suppose a graph G represents the topology of a multiprocessor system, an interconnection network, or a VLSI circuit. We say another graph G' is a *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 this paper, we mean *node fault tolerant* wherever we say *fault tolerant*. To achieve reconfiguration, supporting logic is required for G' and its complexity strongly depends on node output degree and the total number of edges in G'. Thus, the *hardware overhead* of G' should include both spare resources and reconfiguration logic.

Generally speaking, a VLSI module with inputs/outputs can be modeled as 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 directed graph of interest when analyzing such 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 $G_o = (V \cup V_e, E)$ (E is the set of edges) is *closed* if $E \subseteq V \times V$. Otherwise, the graph is *open*. As to a node u ($u \in V(G_o)$), the number of its incoming/outgoing internal edges is *internal in/out degree*, and the number of incoming/outgoing external edges is *external in/out degree*. Any external edge in a target graph G must be replicated to at least K other distinct nodes in a K-FT graph G'.

What makes the construction of a fault-tolerant model for an open graph challenging is the fact that external inputs/outputs to the graph are not interchangeable, making nodes in a open graph 'heterogeneous'. Although much work has been devoted to constructing fault tolerant graph models for closed linear arrays [1, 4, 5, 20, 21], a direct application of these results requires ensuring that every external vertex of V_e has an edge to every internal node, which results in a large number of external edges and prohibitive reconfiguration cost.

In the following subsections, we provide efficient solutions to this problem that minimizes the amount of replication required for external edges. We focus on open linear arrays, which have N internal nodes connected in a linear fashion, together with N external nodes where each is connected to one internal node. With minimum external edge replication, a reconfigurable fault tolerant open linear array with full duplications, a minimum number of spare nodes or a small internal out degree, is constructed respectively.

The target open linear array P_N ($N \geq 2$) is defined as follows. Let u_1, u_2, …, u_N be the internal nodes in the linear array, and let e_1, e_2, …, e_N be the external nodes. Node u_1 is *head node* and u_N is *tail node*. The edges are of the form (u_i, u_{i+1}) and (e_i, u_i) for $1 \leq i \leq N$. Let $P_{N,K}$ be a K-FT linear array of P_N.

It should be noted that the construction $P_{N,K}$ in this paper can be used for general open linear arrays, where each node in $P_{N,K}$ represents a subgraph (a VLSI sub-module), and the external edges can correspond to possibly replicated external inputs and/or outputs. For the following subsections, we use the term *linear array* to mean an *open linear array*, and the term *out/in degree* to mean *internal out/in degree* (the external out/in degree is always $K + 1$).

3.2 K-FT linear arrays with full duplications

A straightforward way to construct a reconfigurable K-FT linear array is to use $K+1$ (instead of $2K+1$ in hard-wired NMR) full duplications. We call it *full-duplication model*. Although a large amount of spare resources are introduced, this model incurs both the minimum internal out degree (the degree of 1 for each node) and simple array-switching based reconfiguration logic.

3.3 K-FT linear array with minimum spares

For a given linear array P_N to be K-FT, there must be at least K spare nodes. We name these nodes u_{N+1}, $\ldots$, u_{N+K}. We construct a K-FT open linear array $P_{N,K}$ with minimum spares as follows. (i) K spare nodes u_{N+1}, $\ldots$, u_{N+K} are introduced; (ii) External edges are added. For $1 \leq i \leq N$, we introduce replicas (e_i, u_{i+1}), $\ldots$, (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}), $\ldots$, (u_i, u_{i+K+1}) of edge (u_i, u_{i+1}) are introduced. Finally for each node u_i $(N \leq i < N+K)$, $N+k-i$ edges from u_i to $u_{i+1}, \ldots, u_{N+K}$ are introduced.

The proof that $P_{N,K}$ is a K-FT N-node linear array, can be found in [16]. There are $(N-1)(K+1)+K(K+1)/2$ internal edges in $P_{N,K}$, and $N-1$ nodes have internal out(in)-degree of $K+1$. Since a K-FT linear array with minimum spares must have at least $(N-1)(K+1)+\Omega(K)$ internal edges [16], $P_{N,K}$ is a near optimal K-FT linear array with minimum spares and external edges when $K \ll N$ (which is generally true in practice). We call this $P_{N,K}$ *min-spare model*. A construction example can be found in [17].

3.4 K-FT linear array with small out degree

In this subsection, we develop a fault tolerant graph model with constant small node out degree and reasonable spare resource cost, through recursive construction.

$(2^M, P_N)$ **Graph.** A $(2^M, P_N)$ graph $(M \geq 0)$ is transformed from 2^M full replications of N-node linear arrays through coalescing nodes and adding extra edges. The goal is to develop a graph that tolerates (2^M-1) faults without 2^M full replications. Specifically, it is constructed recursively in the following way.

$M=0$. $(2^0, P_N)$ graph is simply a replica of P_N.

$M=1$. If $N=1$, $(2^1, P_1)$ graph is composed of two identical linear arrays of P_1. Otherwise, node coalescing is applied to those two identical linear arrays: nodes u_i of the first P_N and $u_{\lceil N/2\rceil+i}$ of the second P_N $(1 \leq i \leq \lfloor N/2\rfloor)$ are merged respectively. An extra edge from $u_{\lceil N/2\rceil}$ of the second P_N to $u_{\lceil N/2\rceil+1}$ of the first P_N is added. Figure 3 shows the construction of $(2^1, P_N)$ graph $(N \geq 2)$. The $\lfloor N/2\rfloor$ pairs of nodes with the same labels and the corresponding edges between them are coalesced. The bold edge represents the extra edge added in the node coalescing. After node coalescing, a $(2^1, P_N)$ graph has $N+\lceil N/2\rceil$ different nodes and $N+\lfloor N/2\rfloor$ distinct edges in total.

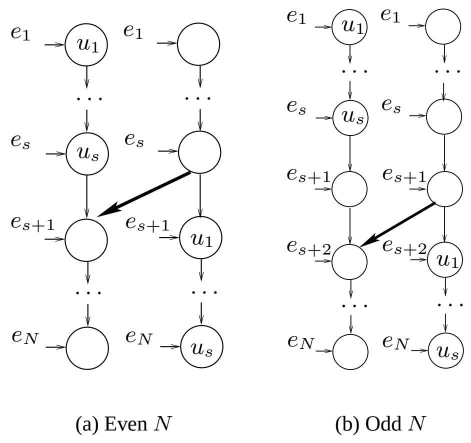

Figure 3. Construction of $(2, P_N)$ graph ($N \geq 2$, $s = \lfloor N/2\rfloor$).

$M>1$. If $N<4$, $(2^M, P_N)$ graph $(M>1)$ is simply 2^{M-1} replicas of $(2^1, P_N)$ graph. Otherwise, the $(2^M, P_N)$ graph is constructed from $(2^{M-1}, P_{\lfloor N/2\rfloor})$ and $(2^{M-1}, P_{\lceil N/2\rceil})$ graphs, as shown in Figure 4.

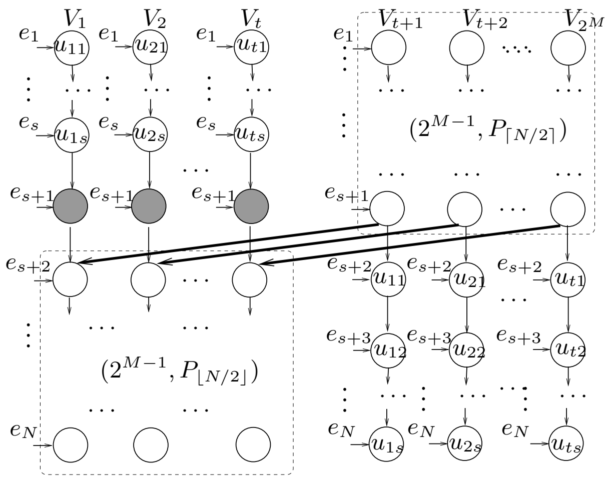

Figure 4. Construction of $(2^M, P_N)$ graph (Odd N, $N>4$, $s=\lfloor N/2\rfloor$, $t=2^{M-1}$).

In Figure 4, 2^{M-1} replicas of $\lceil N/2\rceil$-node and $\lfloor N/2\rfloor$-node linear arrays are added, shown as left-top and rightbottom respectively. The right-top is $(2^{M-1}, P_{\lceil N/2\rceil})$ graph and the left-bottom is $(2^{M-1}, P_{\lfloor N/2\rfloor})$ graph. $\lfloor N/2\rfloor \times 2^{M-1}$ pairs of nodes with the same label are coalesced respectively, so are the corresponding edges. In addition, 2^{M-1} extra edges (bold) are added. The $(2^M, P_N)$ graph with even N is almost the same as Figure 4 except that $(2^{M-1}, P_{\lceil N/2\rceil})$ subgraph is replaced by $(2^{M-1}, P_{\lfloor N/2\rfloor})$ and those 2^{M-1} grey-colored nodes are removed.

Claim 1 *A $(2^M, P_N)$ graph is $(2^M - 1)$-fault tolerant.*

Proof: The proof can be found in [16]. ∎

According to Figures 3 and 4, at most 1 (or 2) extra edges is added to each node during the recursive construction of $(2^M, P_N)$ graph. Consequently, the maximum internal in/out degree of $(2^M, P_N)$ graph is 2 (or 3).

Since there is at least one path staring with node V_i ($i = 1, 2, \cdots, 2^M$) in Figure 4, a $(2^M, P_N)$ graph has at least 2^M paths with 2^M distinct head nodes in total [16]. The overall number of paths $P(M,N)$ in a $(2^M, P_N)$ graph can be calculated recursively as follows. Let Y_i ($1 \le i \le 2^M$) be the number of paths with V_i as the head node, we have

$$P(M,N) = \sum_{i=1}^{2^M} Y_i \tag{1}$$

where,

$$\begin{aligned} Y_i &= \sum_{j=R}^{M} Z(i, 2^j) + 1 \\ R &= \max(1, M + 1 - \lfloor \log_2 N \rfloor) \\ Z(i, 2^j) &= \begin{cases} 0 & \text{if } 0 < (i \bmod 2^j) \le 2^{j-1} \\ Y_{i-2^{j-1}} & \text{otherwise} \end{cases} \end{aligned}$$

Construction. We show how to construct a K-FT linear array based on $(2^M, P_N)$ graphs. Say a $(2^i, P_N) \bigcup (2^j, P_N)$ graph is composed of a $(2^i, P_N)$ graph and a $(2^j, P_N)$ graph disjointly, without any node coalescing between them. It should be clear that a $(2^i, P_N) \bigcup (2^j, P_N)$ graph is $(2^i + 2^j - 1)$-fault tolerant. Let $K + 1 = \sum_{i=0}^{n} b_i \times 2^i$, where $n = \lfloor \log_2(K+1) \rfloor$ and b_i is either 0 or 1. Let $S_{\{b_i=1\}} = \{i | b_i = 1\}$. Therefore, the $\bigcup_{i \in S_{\{b_i=1\}}} (2^i, P_N)$ graph G' is K-fault tolerant. We name this K-FT linear array *small-degree model*. The total number of spare nodes in this model is $O(N \times K/2)$ (which is about half of that in full-duplication model), and the maximum internal in/out degree remains 2 or 3 [16]. The small internal degree reduces node output-degree and potentially simplifies reconfiguration logic. As an example, Figure 5 shows the construction of a 3-FT 4-node array using small-degree model.

Since $K = 3$ and $K + 1 = 2^2$, the 3-FT 4-node array is essentially a (2^2, 4) graph which is recursively constructed from (2^1, 2) graphs. In Figure 5, the nodes with the same

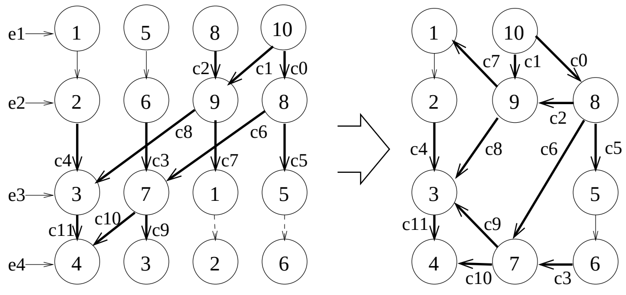

Figure 5. 3-FT array of $(2^2, 4)$ graph.

label are coalesced, and the dashed lines denote the corresponding merged edges. After node coalescing, there are 10 distinct internal nodes and 14 internal edges in total, as well as 4 external nodes ($e_1, \cdots, e_4$). Each external connection is replicated into four connections to the four different nodes in the same row. For instance, external node e_1 connects the internal nodes of 1, 5, 8 and 10.

4 Deadlock Detection and Reconfiguration

When the fail-stop QDI circuit stalls permanently in the presence of failure, the deadlock is recognized by deadlock monitor through watching handshake activity. At any time, if a transition occurs on the data channel, a timer is started. The deadlock detector waits for the next valid protocol state to occur. If it does not occur for a large amount of time (in terms of microseconds or milliseconds), it assumes that the circuit has deadlocked. The timer of deadlock detector is implemented as delay line [11], which is a current-starved inverter chain with an immediate reset (triggered by the following valid transition). By reducing charge/discharge current to enough small amount, the propagation delay of a 6 or 8 cascaded inverter chain, can be increased to the order of milliseconds. Note that the circuit can wait for its environment infinitely in the completion state of a handshake cycle. Thus, the delay line should remain reset for this state (i.e., the timer is disabled). Compared with target asynchronous circuit and reconfiguration logic, the hardware cost of deadlock detector is usually negligible.

Online reconfiguration logic, which is a key module of self-healing circuits, is used to change target circuit topology by replacing faulty components with spare workable components when any failure occurs. Generally speaking, there are two methods to achieve the online reconfiguration. (i) One is to locate faults and use a workable configuration directly. Although such a system is fast in terms of fault recovery time, fault location logic can largely increase hardware overhead, which not only increases design complexity but also hurts the overall reliability by exposing more transistors to unreliable environment. (ii) The other approach we use in this paper, 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, reducing hardware overhead. Also, a longer fault recovery time has little impact on system performance, as faults are not expected to occur frequently.

The core of the online reconfiguration logic is a cyclic state machine which searches all target graphs P_N embedded in the fault tolerant graph $P_{N,K}$ for a working one. All pass-gate control signals are derived from the output of that state machine. Specifically, the system is reconfigured in the following way. Whenever the target circuit deadlocks, the deadlock detector activates the state machine and the latter advances to the next state. All control signals to pass-gates are then updated according to this new state, setting up new connections which corresponds to another P_N embedded in $P_{N,K}$. A local reset signal, which is used to re-initialize the target circuit during reconfiguration, is generated by the deadlock detector. After the reconfiguration is completed, the deadlock detector will be reset, making the new circuit ready for the restarted computation. The above procedure repeats if the new configuration is still not fault-free (system deadlocks again in this case), and another different configuration will be chosen. During each reconfiguration, the propagation delay of local reset and pass-gate control signals is assumed to be bounded. In order to prevent any hardware resource permanently disabled by soft error, no malfunctional configuration is excluded by the state machine during the search. Therefore, an unworkable configuration due to soft error will become reusable in the future as long as the transient fault source disappears at that time.

Since the primary input to reconfiguration logic is the time-out signal from deadlock detector and its primary outputs are control signals to the pass gates in the target circuit, no handshake occurs between reconfiguration logic and its environment. Thus, it is efficient to implement the reconfiguration circuitry in synchronous logic, resulting in less hardware cost than the asynchronous implementation with fake handshake signals. Note that no global clock distribution is required for this synchronous implementation because the reconfiguration logic is triggered sporadically by the local time-out signal. Due to the very long interval of deadlock detections, conservative timing can be applied to reconfiguration logic to guarantee its functionality. Pass gates in the target circuit can be implemented with single-device (instead of complimentary fashion) without any threshold loss, as long as higher power supply voltage is used for reconfiguration logic. All these help reduce the hardware overhead of our self-healing design.

Since the amount of replication for external edges of all three graph models are minimized, the wires of all external edges must be augmented with pass-gates to achieve reconfiguration with respect to K faults. However, the amount of internal edges which have to be augmented with pass-gates for reconfiguration, is different for various graph models. In the next subsections, we present how to implement the reconfiguration logic with minimized pass-gate augmentation, for the three K-FT linear array models of Section 3.

4.1 Full-Duplication and Min-Spare Models

Full-duplication model is composed of $K+1$ replicas of target array P_N, and no internal edge has to be made reconfigurable with pass-gate. Reconfiguration is realized by simply switching to a different replica of P_N by setting its external connections. Specifically, reconfiguration logic can be implemented as a $K+1$-bit one-hot counter (a cyclic shift register with unique bit-'1'). At any time, the unique bit-'1' sets up the connections between external nodes and the corresponding replica while the bit-'0's disable the external connections of all other K replicas.

With the min-spare model, all the remaining nodes after K nodes removed from $P_{N,K}$ have to be used. Therefore, all internal edges have to be augmented with pass-gates to achieve reconfigurability, and the total number of configuration outputs is the sum of all internal and external edges in $P_{N,K}$. There are $\binom{N+K}{K}$ configurations for $P_{N,K}$ of min-spare model, corresponding to all possible fault locations. The state machine of reconfiguration logic can be implemented as a $\lceil \log_2 \binom{N+K}{K} \rceil$-bit counter (when $K \ll N$, $M \simeq K \log_2 N$). Each output of the counter represents a unique choice of K nodes. Thus, the Boolean equation for each configuration output can be derived directly from the remaining graph after removing those K faulty nodes, and static complementary combinational logic is used to implement those Boolean equations (with counter output bits as variables). With Karnaugh map generation for each configuration output, greedy grouping of min/max-terms and search for common subexpressions, the boolean equations with both minimized and simplified logic can be generated automatically. Further, gate decomposition can be achieved through expression tree disintegration so that the resulting boolean logic can be easily implemented in CMOS.

4.2 Small-Degree Model

We define a *basic block* to be a largest sequence of consecutive nodes where all the nodes are on a directed path with no possibility of branching except at the beginning and the end. For instance, nodes of $u_1, u_2, \cdots, u_s$ in Figure 3(a) form one basic block. The construction of small-degree model determines that all nodes of a basic block must be used if any one is used in a configuration. Thus, only internal edges between different basic blocks need to be augmented with pass-gates for reconfiguration. Those reconfigurable internal edges are exactly the ones which are from/to a node with out/in degree of 2 or 3 [16]. Since less internal edges have to be made configurable, the reconfiguration logic for small-degree model can be simpler.

All paths in $P_{N,K}$ of small-degree model have $K+1$ distinct head nodes in total, and there can be multiple paths with the same head node. Since there is no back edge in $P_{N,K}$, all the paths with the same head node u forms a tree structure with node u as the root, and the nodes with out-degree of 2 or 3 provide branches in this tree. Consequently, to search all configurations (embedded target paths) with a given head node u is equivalent to a tree walk with root u. In order to find a workable configuration, the state machine of reconfiguration logic implements the traversal of all $K+1$ trees with different roots. Figure 6 shows the top-level diagram of reconfiguration logic for small-degree model.

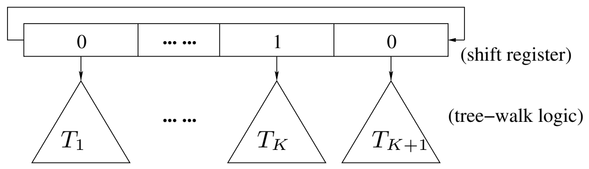

Figure 6. Reconfiguration logic diagram for small-degree model.

In Figure 6, there is a $(K+1)$-bit one-hot counter, which is triggered by the time-out signal from local deadlock detector. This counter is used to initiate a tree walk logic (to look for a workable configuration P_N with a given head node) at one time. After one tree walk is completed (i.e., all the configurations with a given head node have been searched but none is workable), the bit-'1' will be shifted to the neighbor cell, initiating another tree walk. This procedure repeats until a workable configuration is found. Note that gates may be shared between different tree walk logic in Figure 6, if those trees belong to the same $(2^M, N)$-graph.

The internal edges of small-degree model can be reconfigured as follows. For the basic block including the root of current tree walk, the one-hot counter output generates the configuration bit(s) (pass-gate control signal(s)) of out-edges of this basic block; for the basic block without current tree root, the out-edge configuration bit(s) is deduced from the in-edge configuration bit(s). Specifically, it could be one of the three cases of Figure 7 where all the grey-colored nodes form a basic block. The reconfigurable internal edges are bold-colored, and the nearby literal represents the configuration bit of that edge.

A basic block is a *out-branched block* if its tail node has more than one out-edge. A basic block is a *top out-branched block* if there is no other out-branched block on the path from the root of current tree structure to this basic block. A basic block *A* is a *direct out-branched child block of basic block* B if there is a path from block *B* to block *A* in current tree structure and block *A* is the first out-branched block on that path. Figure 7 shows how to generate the out-edge configuration bits according to the in-edge configuration bits of that basic block. (i) The basic block in Figure 7(a) has two in-edges and one out-edge. The configuration bit of the out-edge is a logic-OR of the configuration

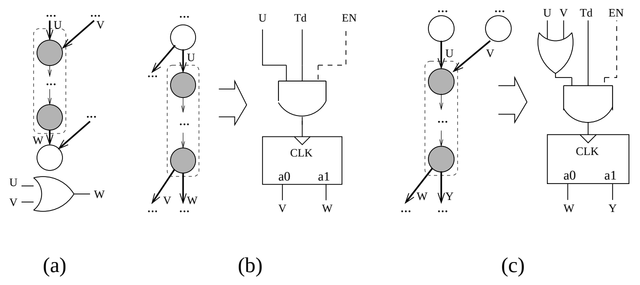

Figure 7. Configuration of internal edges.

bits of the in-edges. In other words, the out-edge is enabled only if one (and only one) in-edge is enabled. (ii) The out-branched block in Figure 7(b) has one in-edge and two out-edges. Neither out-edge is enabled unless the in-edge has been enabled. If the in-edge becomes enabled ($U = 0 \rightarrow 1$) during the reconfiguration (time-out signal $Td = 1$), it triggers the 2-bit edge-sensitive counter (with cyclic output sequence $VW = 00 \rightarrow 01 \rightarrow 10$) and enables one (and only one) out-edge *e*. If there is any direct out-branched child block of this basic block, the enabling of out-edge *e* will trigger another 2-bit cyclic counter. In this case, out-edge *e* has to keep enabled in order to achieve tree walk. Thus, another enable signal EN is added to freeze current configuration outputs (through blocking the clock input) until the cyclic counters for all the direct out-branched child blocks have completed their output cycles. (iii) The logic for configuration bits of basic block in Figure 7(c) is the same as that in Figure 7(b), except that a logic-OR output triggers the cyclic counter so that an out-edge becomes active when either in-edge is enabled. Finally, an extra enable signal should be added to freeze present one-hot counter output (by blocking the clock input) if current tree walk is not completed (i.e., at least one cyclic counter output for the top out-branched blocks is not $'00'$).

The configuration bits of external edges are the same if the internal endpoints are in the same basic block. These bits can be generated from one-hot counter output and internal edge configuration bits directly: they are derived as logic-OR of configuration bit(s) of all (internal) in-edges to this basic block in current tree structure. Note that those logic-OR gates can be re-used from the logic which generate internal edge configuration bits. In most cases, almost no extra logic is required in order to generate the configuration bits of external edges.

As an example, we show the construction of reconfiguration logic for 3-FT 4-node array of Figure 5. Bold lines in Figure 5 denote reconfigurable internal edges. The label of *Cx* beside each bold line denotes the reconfiguration bit of that edge. All external edges are reconfigurable, and we use $D_{i,j}$ represent the configuration bit of the external edge between external node e_i and internal node j. Figure 8 shows the corresponding reconfiguration logic diagram. Blocks $< 1 >$–$< 4 >$ are 2-bit cyclic counters for out-branched blocks. Deadlock detector generates the primary input, time-out signal *To*, which is delayed by τ (signal *Td*) to trigger the cyclic counters so that the one-hot counter outputs have been updated before initiating a new tree walk.

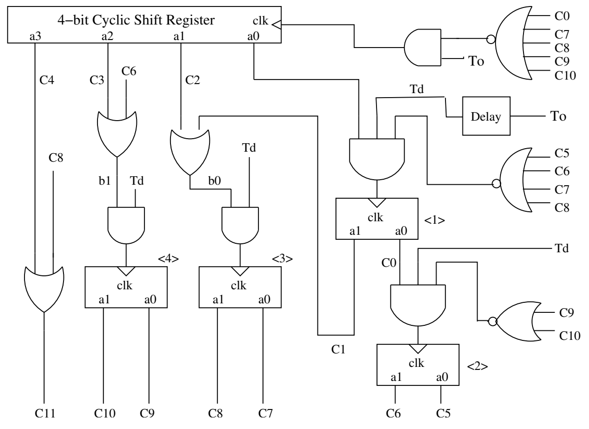

Figure 8. Reconfiguration for 3-FT array.

The reconfiguration logic works in the following way. Initially, all configuration bits of internal and external edges are reset to *0*. When the circuit deadlocks, the upward transition of *time-out* signal activates the one-hot counter, which moves the bit-$'1'$ to the neighbor cell (say a_0). The cyclic counter $< 1 >$ is triggered after τ, setting configuration bit *C0*. Once *C0* becomes high, the clock input of the one-hot counter is blocked, freezing its current output. Meanwhile, the upward transition on *C0* further triggers counter $< 2 >$, setting configuration bit *C5*. After *C5* becomes $'1'$, the clock input of counter $< 1 >$ is blocked. At this point, a new configuration with nodes 10, 8, 5 and 6 is set up. If this configuration is not workable, the circuit deadlocks again. The second upward transition of time-out signal *To* triggers Counter $< 2 >$ after τ, which resets *C5* and sets *C6*. The upward transition on *C6* further triggers the counter $< 4 >$ which sets configuration bit *C9*. At this point, another new configuration with nodes 10, 8, 7 and 3 is set up. Such procedure repeats until current tree walk is completed. Meanwhile, the one-hot counter output remains the same because its clock input is blocked all the time. For the configuration bits of external edges, they are derived directly from one-hot counter output and internal edge configuration bits. For example, $D_{1,1} = D_{2,2} = C4$, $D_{3,3} = C11$, $D_{4,4} = C11 \vee C10$.

5 Evaluation

We evaluate the design of self-healing asynchronous array in terms of hardware cost, performance, energy consumption and fault recovery time, and compare the results with traditional NMR-based method[1]. We say a circuit to be *K-SFT* if it achieves self-healing with respect to K errors. Since full adder is a common datapath operator and a widely-used array-sized VLSI module, we choose it as the target circuit for evaluation. For an M-bit full adder, the number of nodes in the linear array is $N = M/C$ if each node is a C-bit adder cell. Although the carry out is propagated linearly through different nodes, each C-bit adder cell (a node) itself doesn't have to be in ripple-carry fashion (if $C > 1$). In fact, each C-bit adder cell can be implemented using any structure (e.g., carry-look-ahead for high performance), without compromising the fault tolerance property. In the M-bit adder, each 1-bit adder element is implemented in terms of a FS-PCHB circuit. Thus, the M-bit adder can guarantee fail-stop with respect to M errors as long as they are in distinct FS-PCHB circuits (i.e., different 1-bit elements). Here we choose the adder size to be 64-bit.

5.1 Hardware overhead

The cost of a circuit in terms of the amount of hardware necessary is estimated by its transistor count. We define *normalized hardware cost* to be $H(A_{ft})/H(A_o)$, where $H(A_o)$ is the hardware cost of the baseline adder, and $H(A_{ft})$ is the hardware cost of fault tolerant adder. We investigate normalized hardware costs of the self-healing design based on three graph models of Section 3 respectively. In the remaining of this section, we use the term *hardware cost* to mean *normalized hardware cost*.

Since no extra spare resource is added to baseline adder, its hardware cost has nothing to do with C and K. Because full-duplication model is composed of $K + 1$ full replicas of the baseline adder, the hardware cost is only decided by M and K and independent of node size C. For min-spare and small-degree models, larger node size C results in more spare resource cost (given K) but simpler reconfiguration logic due to less number of nodes. Consequently, their hardware costs are affected by node size C.

We first investigate the impact of node size on the hardware costs of min-spare and small-degree models so that an appropriate node size can be used to reduce the total hardware cost, given the M and K. Specifically, total hardware costs of a K-SFT 64-bit adder with various node sizes of 1-, 2-, 4-, ..., 32-bit, are studied. Figure 9 shows the results, where *MIN* and *SML* represent the self-healing adder based on min-spare and small-degree model respectively.

Several conclusions can be drawn from Figure 9. First, the hardware cost of min-spare model varies dramatically with respect to different node sizes, while that of small-degree model changes much less. An appropriate node size does reduce the hardware costs of both models. Second,

[1]To apply NMR to QDI circuits is non-trivial: timers and non-negligible self-reconfiguration logic have to be added to the voter. We omit those components and only investigate the voter core here, so the reported hardware cost, performance and energy consumption are optimistic.

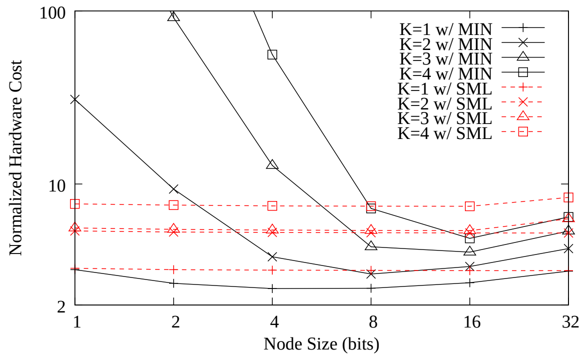

Figure 9. Hardware costs of K-SFT 64-bit adder with different node sizes

larger node size tends to reduce reconfiguration and pass-gate augmentation overheads of both models (although it incurs more spare resource cost), because the graph is simplified with fewer nodes. Third, the complexity of the adder core itself is quickly dwarfed by that of the reconfiguration logic for min-spare model when N and K increase. Consequently, the optimal node size which incurs the minimum hardware cost of min-spare model, becomes larger and closer to $M/2$ when M or K grows. However, the optimal node size for small-degree model gradually reduces with larger K because the reconfiguration cost varies much less, which makes spare resource cost become the deciding factor when the node size is large enough.

Table 1 shows the total hardware costs (shown in column *Total*) of the K-SFT 64-bit adder based on different graph models with the optimal node sizes (shown in column *Cell Size*), as well as the corresponding hardware cost breakdowns. Meanwhile, we compare those hardware overheads with traditional NMR-based design which uses $2K + 1$ replicas and a majority voter [8]. Because the configurations in Table 1 result in small node numbers in the fault tolerant arrays, the wiring overhead can be neglected compared with the node cost. Thus, the reported numbers in this table are good approximations to the real results.

In Table 1, *MIN*, *SML* and *DUP* denote min-spare, small-degree and full-duplication model respectively. The total hardware costs in the table are further decomposed into three categories: (i) The hardware cost of reconfiguration logic with deadlock detection (*Recfg*) ; (ii) The hardware cost of spare resources with fail-stop augmentation logic (*Spare*); (iii) Other hardware cost including the M-bit adder with fail-stop logic and all configuration pass-gates. For each category, higher hardware cost is generally expected with larger K. Regarding small-degree model, however, the case of $K = 2^M - 1$ (e.g., $K = 7$) usually results in smaller spare cost than the case of $K = 2^M - 2$ (e.g., $K = 6$) because more nodes are coalesced in the first situation, which may further result in less total hardware cost. As to min-spare model, higher K requires larger number of configuration bits while leads to more possible gate sharing of different reconfiguration bits, which may slightly reduce the overall reconfiguration overhead (for example, the cases of $K = 8$ and $K = 7$).

Generally speaking, min-spare model incurs the minimum hardware overhead, because an appropriate node size simplifies reconfiguration logic while keeping the minimum spare resource cost; Small-degree model results in low hardware cost, as it reduces the spare resources through node coalescing while maintaining simple reconfiguration; Full-duplication model incurs reasonable hardware overhead due to its $K+1$ full duplications; NMR-based method causes the highest hardware cost (except the case of K=1) due to both $2K + 1$ full replicas and non-negligible voter logic. Second, the size of error-sensitive circuit (*Recfg*) of full-duplication model is negligible, while that size remains small in small-degree model and becomes large (but still reasonable) in min-spare model. However, NMR incurs the largest error-sensitive circuit (*Voter*) and that cost becomes prohibitive when K increases (because voter has to compare every combination of $K + 1$ signals out of $(2K + 1)$ inputs). Thus, it is impractical to apply NMR design at fine granularity to tolerate a large number of faults. Third, NMR design is not scalable with K, because of the dramatic increase of voter complexity which not only dreadfully slows down the system but also consumes a lot more energy.

It is safe to conclude that our self-healing design is less costly and more scalable (with K) than NMR method. For the self-healing design of different models: Full-duplication model becomes the best choice if tiny protected circuit size is required; Min-spare model can be the appropriate candidate if the designers want to reduce the hardware cost as much as possible; Otherwise, small-degree model is a good solution due to its small protected circuit size and reasonable hardware overhead.

5.2 Performance overhead

We used HSPICE to simulate the self-healing 64-bit adder of three graph models, and compared the throughputs with the baseline adder and NMR adder. We define *normalized throughput* to be $Thr(A_{ft})/Thr(A_o)$, where $Thr(A_o)$ is the throughput of baseline adder, and $Thr(A_{ft})$ is the throughput of fault tolerant adder. Figure 10 shows the normalized throughputs, where *DUP*, *MIN* and *SML* represent the self-healing adder with full-duplication, min-spare and small-degree model, and *FS* denotes the fail-stop 64-bit adder without any spare resource. All the K-SFT adders use the optimal node sizes in Table 1. The HSPICE simulation uses TSMC 0.18um technology at 25°C.

Because the reconfiguration logic and spare resources are not on the critical path, the performance of our self-healing design does not strongly depend on K (it only changes within 10% for different Ks), exhibiting better per-

Table 1. Hardware costs of K-SFT 64-bit asynchronous adder

K	Cell Size		Recfg/Voter				Spare				Total			
	MIN	SML	MIN	SML	DUP	NMR	MIN	SML	DUP	NMR	MIN	SML	DUP	NMR
1	4	32	0.14	0.03	0.01	0.62	0.12	0.96	1.92	2.00	2.48	3.16	4.09	3.62
2	8	32	0.21	0.04	0.01	1.41	0.48	2.88	3.85	4.00	3.01	5.21	6.14	6.41
3	16	16	0.14	0.08	0.02	4.43	1.44	2.88	5.77	6.00	4.01	5.38	8.19	11.43
4	16	16	0.21	0.09	0.02	17.34	1.92	4.81	7.69	8.00	4.68	7.44	10.23	26.34
5	16	16	0.31	0.11	0.03	73.28	2.40	5.77	9.62	10.00	5.39	8.54	12.28	84.28
6	16	16	0.54	0.12	0.03	314.06	2.88	7.69	11.54	12.00	6.24	10.59	14.33	327.06
7	16	8	2.64	0.14	0.04	1342.03	3.37	6.73	13.46	14.00	8.95	9.81	16.37	1357.03
8	16	8	2.40	0.16	0.04	5699.22	3.85	8.65	15.38	16.00	9.31	11.87	18.42	5716.22

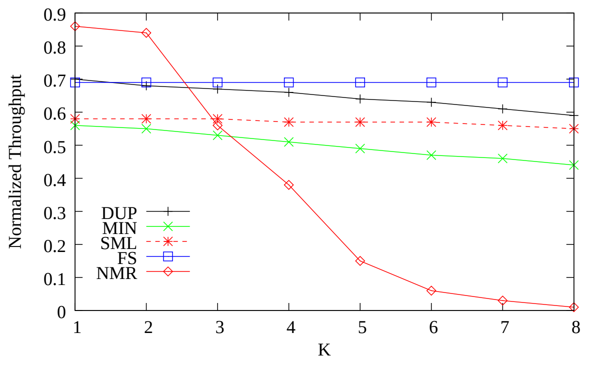

Figure 10. Normalized throughputs.

formance scalability than NMR. Note that the performance overhead of self-healing design primarily comes from the fail-stop augmentation logic (shown as *FS* in Figure 10) and the graph model themselves only incur small performance overheads. Given another fail-stop implementation method with higher throughput, our self-healing design can achieve even better performance.

Among those three graph models, full-duplication model always achieves the best performance, because there is no pass-gate overhead on the carry propagation. Since there are only $\log_2 \lfloor N/2 \rfloor$ (instead of $N-1$ in min-spare model) pass-gates added to the carry propagation in small-degree model, small-degree model achieves higher throughput than min-spare model. With larger K, the pass-gates on external edges become the majority of pass-gate augmentation overhead. Thus, the throughputs of full-duplication and small-degree models become growingly closer to each other.

5.3 Energy overhead

Generally speaking, energy consumption of a circuit can be divided into two parts: leakage energy consumption and dynamic energy consumption. A simple estimate of the leakage energy consumption can be obtained from the transistor count. It can be concluded from Table 1 that the leakage energy overheads of the self-healing designs are much less than NMR method.

The same HSPICE simulations are applied for dynamic energy consumptions. We define *normalized energy consumption* to be $E(A_{ft})/E(A_o)$, where $E(A_o)$ and $E(A_{ft})$ are the energy consumptions of baseline and fault tolerant adders respectively. Figure 11 shows the normalized results, where the labels are the same as those in Figure 10.

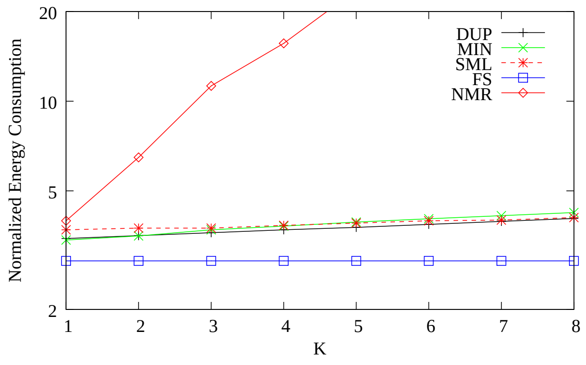

Figure 11. Normalized energy consumptions.

Because all $2K+1$ replicas and the voter are working all the time, the NMR adder incurs the largest energy overhead and such overhead increases dramatically when K grows. As to our self-healing design, the dynamic energy overhead is much less because there is no switching activity in both reconfiguration logic and spare hardware. Most of the energy overhead comes from fail-stop augmentation logic being used which remains constant (shown as *FS* in Figure 11). Hence the total energy overheads do not change significantly (within 22%) with different Ks.

5.4 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 depends on target circuit size and is usually in terms of micro/milli-seconds. The worst fault recovery time, which can be used to estimate the expected fault recovery time, is that all possible configurations have been searched and only the last one is found workable. In our self-healing design, the worst fault recovery time can be denoted as $|P(G)| \times \tau_d$, where $|P(G)|$ is the total number of paths embedded in the $P_{N,K}$. Since τ_d is constant for the given circuit, we can normalize the worst fault recovery time to be $|P(G)|$. Given the fault rate of once per hundreds hours and τ_d of milliseconds, $|P(G)|$ can be thousands or tens of thousands while with little impact to overall system performance.

As to full-duplication model, $|P(G)| = K+1$ because of $K+1$ full replicas; For min-spare model, $|P(G)| = \binom{N+K}{K}$ as any K out of $N+K$ nodes can be faulty; When it comes to small-degree model, $|P(G)|$ can be calculated recursively with equation (1) in Section 3.4. Figure 12 shows the normalized worst fault recovery times of K-SFT 64-bit adders with the optimal node sizes of Table 1.

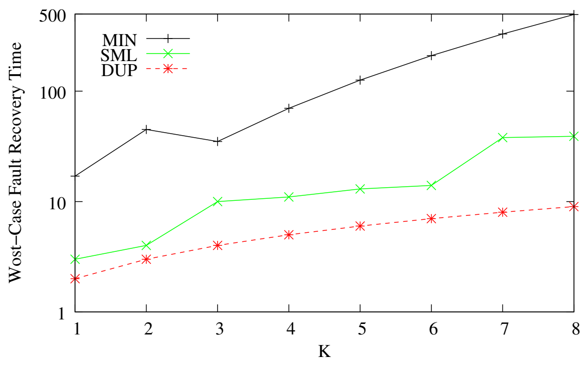

Figure 12. Normalized worst fault recovery time.

Because the optimal node size changes at some Ks for min-spare and small-degree models, there are significant variation of the normalized fault recovery times (the number of paths) at those points. Generally speaking, full-duplication model incurs the minimum fault recovery time, while min-spare model takes the longest fault recovery time. Besides, the fault recovery time of min-spare model dramatically increases when K grows, noticeably reducing its maintainability. On the other hand, the fault recovery time of small-degree model is always at the same order of that of full-duplication model, and thus remains acceptable in practice even with respect to a large number of faults.

6 Related Work

Although a lot of research has been conducted on fault tolerant synchronous design [8], only a little work has been done for asynchronous circuits. Jackson et al. [6] implemented a biologically embryonic asynchronous array based on clocked FPGA to achieve fault tolerance, while with large hardware overhead for redundant copies of configuration bits and complex re-placement-and-routing logic. With full duplication of circuit parts and synchronization of replicated results through C-elements, the authors in [9, 14] developed several fault detection methods and hardening techniques for QDI circuits. Although their approaches can improve the robustness of QDI circuits, significant timing assumptions are required in order to detect errors and those methods cannot guarantee fault detection and tolerance all the time. By using doubled-up production rules, Jang et al. [7] proposed a SEU-tolerant QDI circuit design without any requirement of significant timing assumptions. However, this approach cannot be applied to hard error tolerance, and usually results in large hardware cost and significant performance overhead (for example, the resulting circuit can be three times larger and twice slower [7]). Moreover, this approach is designed for single error tolerance and cannot be extended to an arbitrary number of faults. Compared with the aforementioned work, the method proposed in this paper can be applied to achieving fault tolerance with respect to any number of hard or soft errors, while with reasonable hardware cost and acceptable overheads.

There is a wealth of research on graph models of reconfigurable fault tolerant linear arrays. Hayes [5] introduced the concept of fault tolerant graphs and proposed optimal *K*-FT graphs for linear array and circle. Alon et al. [1] constructed fault tolerant graphs for (undirected) linear arrays in a more general way. Haray et al. [4] discussed the design of optimal *K*-edge fault tolerant graphs of paths, circles and *n*-dimensional hypercube. Zhang [21] 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 [20]. However, there is no external input or output with respect to the array, because they treat the whole topology of parallel systems or interconnection networks as only one graph.

7 Conclusion

For asynchronous circuits, the causality and event-ordering is realized by handshake and data are usually encoded with redundant rails. Thus, they have the potential to achieve self-checking with small hardware overhead [9]. However, it is non-trivial to apply conventional duplication-and-comparison method to asynchronous logic to achieve fault tolerance due to the lack of a global synchronization. To efficiently tolerate errors in asynchronous circuits, this paper proposed a general framework for constructing a self-healing array-sized QDI circuit which achieves K-fault tolerance without comparison procedure while exploiting the self-checking potential. Three fault tolerant array models as well as efficient implementations of the corresponding self-reconfiguration logic were presented for this framework. Furthermore, the relationship between reconfiguration complexity and spare resource cost was analyzed for all the graph models. By exploiting inherent self-checking potential of QDI logic, this reconfigurable fault tolerant design can achieve much lower hardware overhead than traditional NMR method (especially for large K). By keeping most of self-healing related logic off the critical path, this reconfigurable design also achieves small and nearly constant performance and energy overheads with respect to different Ks, exhibiting much better scalability than NMR.

Regarding the self-healing design based on different graph models, the energy overheads are close to each other because fail-stop logic and pass-gates on external edges contribute most of the extra energy consumption. The min-spare model requires the least hardware cost but the largest error-sensitive circuit size, performance overhead and expected fault recovery time; the full-duplication model results in the most hardware cost but the minimum error-sensitive circuit size, performance overhead and expected fault recovery time; the small-degree model is a compromise between min-spare and full-duplication model: it incurs modest hardware overhead, small critical circuit size, low performance overhead and short fault recovery time. Therefore, the min-spare model is the most effective for the fault tolerant designs with small K or for the minimum hardware cost, while the full-duplication model can be used for the cases which require tiny error-sensitive circuit size or very short fault recovery time. Otherwise, the small-degree model becomes the appropriate solution.

Finally, this self-healing method can be conveniently applied to synchronous design. The only significant change is the implementation of fail-stop behavior in target clocked circuit. One way to do this is duplicating the target circuit and comparing the results off the critical path on each clock cycle. If any mismatch is reported, the clock will be shut down and online reconfiguration will be activated.

References

[1] N. Alon and F. Chung. Explicit construction of linear sized tolerant networks. *Discrete Math*, 72:15–19, 1988.

[2] K. Banerjee, S. J. Souri, and P. Kapur et al. 3-D ICs: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. *Proc. IEEE*, 89(5), 2001.

[3] G. Bourianoff. The future of nanocomputing. *Computer*, 36(8), 2003.

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

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

[6] A. H. Jackson and A. M. Tyrrell. Implementing asynchronous embryonic circuits using AARDVArc. In *Proc. NASA/DoD Conference on Evolvable Hardware*, 2002.

[7] W. Jang and A. J. Martin. SEU-tolerant QDI circuits. In *Proc. International Symposium on Asynchronous Circuits and Systems*, 2005.

[8] B. W. Johnson. *Design and Analysis of Fault Tolerant Digital Systems*. Addison Wesley, 1989.

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

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

[11] N. R. Mahapatra, A. Tareen, and S. V. Garimella. Comparison and analysis of delay elements. In *Proc. the 45th Midwest Symposium on Circuits and Systems*, 2002.

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

[13] A.J. Martin, A. Lines, and R. Manohar et.al. The design of an asynchronous MIPS R3000. In *Proceedings of the Conference on Advanced Research in VLSI*, 1997.

[14] Y. Monnet, M. Renaudin, and R. Leveugle. Hardening techniques against transient faults for asynchronous circuits. In *Proc. IEEE International On-Line Testing Symposium*, 2005.

[15] S. Peng and R. Manohar. Efficient failure detection in pipelined asynchronous circuits. In *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2005.

[16] S. Peng and R. Manohar. Explicit constructions of fault-tolerant open linear arrays. Technical Report CSL-TR-2005-1044, Cornell University, 2005.

[17] S. Peng and R. Manohar. Fault tolerant asynchronous adder through dynamic self-reconfiguration. In *Proc. IEEE International Conference on Computer Design*, 2005.

[18] J. Srinivasan, S. V. Adve, and P. Bose et al. The impact of technology scaling on lifetime reliability. In *Proc. International Conference on Dependable Systems and Networks*, 2004.

[19] T. Verdel and Y. Makris. Duplication-based concurrent error detection in asynchronous circuits: Shortcomings and remedies. In *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2002.

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

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