A Rollback Interval for Networks with an Imperfect Self-Checking Property

by

John J. Shedletsky

December 1975

Technical Report No. 96

This work was supported by National Science Foundation Grant GJ-40286,
A ROLLBACK INTERVAL FOR NETWORKS WITH AN IMPERFECT SELF-CHECKING PROPERTY

John J. Shedletsky

Technical Report No. 96

December 1975

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California

*This work was supported by National Science Foundation Grant GJ-40286
A ROLLBACK INTERVAL FOR NETWORKS WITH AN
IMPERFECT SELF-CHECKING PROPERTY

John J. Shedletsky

Technical Report No. 96

December, 1975

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California

ABSTRACT

Dynamic self-checking is a technique used in computers to detect a fault quickly before extensive data contamination caused by the fault can occur. When the self-checking properties of the computer circuits are not perfect, as in the case with self-testing only and partially self-checking circuits, the recovery procedure may be required to roll back program execution to a point prior to the first undetected data error caused by the detected fault.

This paper presents a method by which the rollback distance required to achieve a given probability of successful data restoration may be calculated. To facilitate this method, operational interpretations are given to familiar network properties such as the self-testing, secureness, and self-checking properties.

An arithmetic and logic unit with imperfect self-checking capability is analyzed to determine the minimum required rollback distance for the recovery procedure.

INDEX TERMS: checkpoint, rollback distance, rollback interval, self-checking, self-testing.
INTRODUCTION

Computer systems can be fault tolerant only if recovery from a fault detection can be successfully accomplished \cite{1}. Recovery is defined \cite{2} as the continuation of system operation with data integrity after a fault is detected. Data integrity must be restored by overcoming the data contamination that may have occurred prior to fault detection. The error-control provided by the self-testing and self-checking properties of logic networks can be an important asset to the recovery procedure \cite{3,4}.

A network is \textbf{self-testing} \cite{3,5} for a fault if at least one normally applied input vector results in a detection of the fault. Coded outputs and code checkers are the usual means of detection. A network is \textbf{secure} for a fault if every incorrect output caused by the fault is detected. A network that is both self-testing and secure is \textbf{self-checking}.

A fault in a self-testing network is eventually detected, but undetected data errors due to the fault may be propagated prior to fault detection. The additional property of secureness in a \textbf{self-checking} network guarantees that no such propagation occurs, thereby simplifying the recovery process. The recovery algorithm in a \textbf{self-checking} standby computer system for example, need not validate the integrity of previously generated data. Program execution may continue immediately after initialization of the spare.

In some cases, a self-checking capability may be infeasible, or cost prohibitive in terms of hardware, software, or time redundancy.
The alternative is an imperfect-self-checking capability provided by networks that are self-testing for a fault, but not completely secure. Self-testing only and partially self-checking [6] networks provide this capability,

Program rollback may be necessary to restore data integrity when the self-checking capability of the system is imperfect. A successful data restoration can only occur when the program rollback is to a point prior to the first undetected data error.

The required rollback distance for successful data restoration is critical in real-time control applications where commands, once issued, cannot be rescinded. In this case, the required rollback distance translates to a required output buffer delay. Application dependent requirements may place a bound on this delay, thereby constraining the attainable reliability.

This paper presents a model to quantitatively describe the rollback distance required for successful data restoration in imperfect-self-checking networks. The model is probabilistic so the rollback distance is a function of the desired probability of successful data restoration.

In an operational sense, the required rollback distance is the only property of interest for an imperfect-self-checking network. Classifying a property of a network with respect to a certain fault set reveals little about the operational behavior of the network if the fault set does not include every fault that is reasonably likely to occur. The familiar dynamic checking properties are re-defined to facilitate an operational approach to the analysis. The error latency model
[7,8] is basic to the analysis.

An example arithmetic and logic unit (ALU) with imperfect-self-checking capability [9] is analyzed in section 4. This ALU exemplifies the design techniques used in the Stanford SDC-16 computer. The required rollback distance is plotted as a function of instruction mix.

In the following a fault is a physical defect that changes the function realized by a network. An error occurs when data assumes an incorrect value due to a fault.

THE ROLLBACK INTERVAL OF A FAULT IN AN IMPERFECT-SELF-CHECKING NETWORK

In a system environment, an imperfect-self-checking network with a fault can respond to the application of an input vector in three mutually exclusive ways, denoted by C, D, and E:

- **C (correct response)**: Correct data is generated and propagated by the network.
- **D (detection)**: A fault is detected, initiating system recovery. No data is propagated.
- **E (error)**: Incorrect data is generated or propagated undetected by the network.

A fault is detected by a checker that signals the appearance of a **noncode** output vector [10]. The detected fault may be internal to the network or it may be a fault in a predecessor network that causes the application of an improper input vector. In the latter case, the detected fault is viewed as a multiple fault sticking each input line of the network to the corresponding bit values of the improper vector.
The input vectors applied to a network in a computer are generated by a process too complex to describe in the general case. A reasonable approach is a probabilistic description. We assume successive input vectors are applied to a network randomly and independently. The probability of applying a given input vector is specified by an input-vector probability distribution, which is assumed known.

**Definition.** The conditional response probabilities $c_i$, $d_i$, and $e_i$ of a fault $F_i$ in a network are the conditional probabilities of the responses $C$, $D$, and $E$ respectively, given that the fault $F_i$ is active in the network:

$$c_i = \Pr[C|F_i], \quad d_i = \Pr[D|F_i], \quad e_i = \Pr[E|F_i].$$

Every input vector with a nonzero probability of application belongs to the set of normal input vectors $\mathcal{N}$. A fault $F_i$ partitions the set $\mathcal{N}$ into three subsets $\mathcal{N}_i^C$, $\mathcal{N}_i^D$, and $\mathcal{N}_i^E$. An input vector $X$ is a member of the set $\mathcal{N}_i^C$, $\mathcal{N}_i^D$, or $\mathcal{N}_i^E$ if the application of $X$ to the faulty network elicits the responses $C$, $D$, or $E$ respectively. The conditional response probability $c_i$, $d_i$, or $e_i$ is equal to the sum of the probabilities of those vectors belonging to $\mathcal{N}_i^C$, $\mathcal{N}_i^D$, or $\mathcal{N}_i^E$.

$$c_i = \sum_{x_j \in \mathcal{N}_i^C} \Pr[X_j] \quad d_i = \sum_{x_j \in \mathcal{N}_i^D} \Pr[X_j] \quad e_i = \sum_{x_j \in \mathcal{N}_i^E} \Pr[X_j] \quad (2.1)$$

The conditional response probabilities depend on the input vector. The conditional response probabilities sum to 1.

**Example 1:** In the one-out-of-eight decoder [11] in Fig. 1, a fault detection occurs when $z_1 z_2 = \langle 0 0 \rangle$ or $\langle 1 1 \rangle$. Assuming equally-likely input vectors, Table 1 lists the single stuck-at faults and their respective conditional response probabilities.
Fig. 1. - A one-out-of-eight decoder.

Table 1. - Conditional response probabilities of the single stuck-at faults in the network of Fig. 1 (equally-likely inputs).

<table>
<thead>
<tr>
<th>Lines stuck-at 1</th>
<th>$c_1$</th>
<th>$d_1$</th>
<th>$e_1$</th>
<th>Lines stuck-at 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>.500</td>
<td>0</td>
<td>.500</td>
<td>1-3</td>
</tr>
<tr>
<td>52-59</td>
<td>.125</td>
<td>0</td>
<td>.875</td>
<td>52-59</td>
</tr>
<tr>
<td>4-9,42-51</td>
<td>.500</td>
<td>.500</td>
<td>0</td>
<td>4-9,50-51</td>
</tr>
<tr>
<td>10-33</td>
<td>.875</td>
<td>.125</td>
<td>0</td>
<td>10-49</td>
</tr>
<tr>
<td>34-41</td>
<td>.125</td>
<td>.500</td>
<td>375</td>
<td></td>
</tr>
</tbody>
</table>
Table 2 defines the dynamic-checking properties of a network with respect to a given fault $F_i$ in terms of the values assumed by the conditional response probabilities of $F_i$.

<table>
<thead>
<tr>
<th>untested</th>
<th>$d_i=0$</th>
<th>$d_i\neq 0$</th>
<th>$e_i=0$</th>
<th>$e_i\neq 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>self-testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>secure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insecure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>self-checking</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. - The dynamic checking properties of a network.

The definitions in Table 2 are applicable to fault sets as well as individual faults. A network is self-testing for a fault set $F$ for example, if it is self-testing for every member of $F$. The network in Fig. 1 is self-testing for the set of single faults affecting lines 4-51, and self-checking for the set of stuck-at 0 faults affecting lines 4-51.

A fault in an imperfect-self-checking network need not cause an immediate detection or error response. The first error may precede or follow the first detection (Fig. 2).

Fig. 2. - Latency in a self-testing network.
Definition. The error latency $EL_i$ (detection latency $DL_i$) of a fault $F_i$ is the number of input vectors applied to a network while $F_i$ is active until the first error (detection) response due to $F_i$ is observed.

The mean error (and detection) latency can be extremely large in some cases [7,8].

Definition. The latency difference $LD_i$ of a fault $F_i$ is the difference between the detection latency $DL_i$ and the error latency $EL_i$ ($LD_i = DL_i - EL_i$).

The latency difference is positive (negative) when the first error occurs before (after) the first detection. The latency difference is never zero. The minimum rollback distance required for successful data restoration after a first time fault-detection is equal to $\max(DL_i, 0)$.

The latency difference is equal to positive $n$ when the first error occurs, then $n-1$ non-detections, and then the first detection. Any number of correct responses may precede the first error. The probability that the latency difference is equal to $n$ is

$$\Pr[LD_i = n] = \frac{(1+c_1+c_1^2+c_1^3+\ldots)(e_1(1-d_1)^{n-1}d_1)}{(d_1e_1/(e_1+d_1))(1-d_1)^{n-1}; \; n>0} \quad (2.2)$$

$$\Pr[LD_i = -n] = \frac{(1+c_1+c_1^2+c_1^3+\ldots)(d_1(1-e)^{n-1}e_1)}{(d_1e_1/(e_1+d_1))(1-e_1)^{n-1}; \; n>0.} \quad (2.3)$$

The probability that the latency difference is greater than $n$ is

$$\Pr[LD_i > n] = \sum_{j=n+1}^{\infty} \frac{d_1e_1}{d_1+e_1} (1-d_1)^{j-1} = \frac{e_1}{d_1+e_1}(1-d_1)^n; \; n\geq0. \quad (2.4)$$
Finally, the probability distribution function for the latency difference is,

$$\Pr[LD_i < n] = 1 - \Pr[LD_i > n] = 1 - \frac{e_i}{e_i + d_1} (1 - d_1)^n ; \ n > 0. \ (2.5)$$

When the latency difference is less than or equal to n, the first error occurs after the first detection or at most n inputs before the first detection. Assuming that a program rollback restores data integrity, a successful data restoration occurs when the rollback is at least to the point of the first error. Equation (2.5) describes the probability of successful data restoration when the rollback distance from the point of detection of $F_i$ is n inputs.

When n is zero, equations (2.4, 2.5) have the following degenerate forms,

$$\Pr[LD_i > 0] = \Pr[\text{first E before first D}] = \frac{e_i}{e_i + d_1} \quad (2.6)$$

$$\Pr[LD_i < 0] = \Pr[\text{first D before first E}] = \frac{d_1}{e_i + d_1}. \quad (2.7)$$

Equation (2.6) is implicitly used for the specific analysis of a checked decoder in [11].

**Definition.** The rollback interval $rb(x)_i$ of a fault $F_i$ in a self-testing network is the minimum rollback distance (from the point of detection of $F_i$) necessary to have probability of successful data restoration at least equal to $x$.

If $x$ is less than or equal to $\Pr[\text{1st D before 1st E}]$ then the rollback interval is zero. If $x$ is greater than $\Pr[\text{1st D before 1st E}]$ then the rollback interval is a positive integer. A positive rollback interval is obtained by setting the left side of equation (2.5) to $x$ and solving for $n$. 
\[ rb(x)_1 = \max \left( \left[ \frac{\log(1-x) + \log(1+(d_1/e_1))}{\log(1-d_1)} \right] , 0 \right) \] (2.8)

Note that the rollback interval of a fault \( F_1 \) is only defined when the network is self-testing (\( d_1 \neq 0 \)) for \( F_1 \). If the network is also secure (\( e_1 = 0 \)) for \( F_1 \), the rollback interval is always zero.

**Example 2.** For the fault \( F_1 = 34 \) stuck-at 1 in Fig. 1, \( c_1 = .125 \), \( d_1 = .500 \), and \( e_1 = .375 \). The probability distribution function of the latency difference is plotted in Fig. 3. The rollback interval \( rb(.99)_1 \) is 6 inputs; the probability of successful data restoration given the detection of \( F_1 \), is at least .99 when the rollback distance is 6 inputs or more.

---

**Fig. 3.** The probability distribution function of the latency difference of a fault \( F_1 \).
Theorem 1. If $p_{\min}$ is the probability of the least-likely input vector in the normal set $\mathcal{N}$ of input vectors, then the upper-bound on the rollback interval for any fault in an imperfect-self-checking network is denoted by $rb(x)_{\text{ub}}$ and is,

$$
rb(x)_{\text{ub}} = \max\left(\left[\frac{\log(1-x)}{\log(1-p_{\min})} - 1\right], 0\right).
$$

Proof. Equation (2.8) indicates that the rollback interval has maximum value when $d_i$ has minimum value and $e_i$ has maximum value. The minimum possible value of $d_i$ is $p_{\min}$ while the maximum possible value of $e_i$ is $1-p_{\min}$. This condition occurs when the least-likely input vector causes a detection while the remaining input vectors cause an error.

When each input vector in the normal set $\mathcal{N}$ is equally-likely then $p_{\min} = 1/N$ where $N$ is the cardinality of $\mathcal{N}$. Table 3 lists the upper-bound rollback interval for this case.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$rb(.95)_{\text{ub}}$</th>
<th>$rb(\frac{1}{2})_{\text{ub}}$</th>
<th>$N$</th>
<th>$rb(.95)_{\text{ub}}$</th>
<th>$rb(\frac{1}{2})_{\text{ub}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>16</td>
<td>25</td>
<td>16</td>
<td>46</td>
<td>71</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>29</td>
<td>32</td>
<td>94</td>
<td>145</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>34</td>
<td>64</td>
<td>190</td>
<td>292</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>39</td>
<td>128</td>
<td>381</td>
<td>587</td>
</tr>
<tr>
<td>10</td>
<td>28</td>
<td>43</td>
<td>256</td>
<td>765</td>
<td>1176</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>512</td>
<td>1532</td>
<td>2355</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>1024</td>
<td>3066</td>
<td>4713</td>
</tr>
</tbody>
</table>
One recovery philosophy is to set the rollback distance equal to
the upper-bound rollback interval \( rb(x)_{ub} \). Then the probability of
successful data restoration is at least equal to \( x \) for any fault that
may occur. This "upper-bound" philosophy uses a simple analysis that
leads to a rough approximation (i.e. the upper-bound) of the rollback
distance actually required.

Two other recovery philosophies use progressively finer
approximations to the required rollback distance, but at a progressively
greater cost of analysis. The increased accuracy generally allows
the recovery philosophy to set a shorter rollback distance while main-
taining a given probability of successful data restoration. A "worst-
case" philosophy is discussed below and a "fault-set" philosophy is
discussed in the following section.

In the set of reasonably likely faults in a network, the fault with
the greatest rollback interval is the worst-case fault. The "worst-case"
recovery philosophy sets the rollback distance equal to the rollback
interval \( rb(x)_{wc} \) of the worst-case fault. The probability of successful
data restoration is at least equal to \( x \) for any fault that occurs.

The worst-case fault is discovered in two steps. The first step
is to form a list \( L_1 \) of reasonably likely faults with their corresponding
conditional response probabilities. If the set of reasonably likely
faults includes all single stuck-at faults, then only one member of each
fault equivalence class \([12,13]\) need be listed. If all multiple stuck-at
faults are to be considered, then the enormous number of faults to be
listed may be significantly reduced by a method discussed in the appendix
to \([8]\).
In step 2, the worst-case fault is chosen from the list $L_1$. Unfortunately, the choice of the worst-case fault depends on the probability $x$ of successful recovery desired. Example 3 illustrates the problem.

Example 3. Let $L_1$ be the list of faults $F_1, F_2,$ and $F_3$ below.

- $F_1$: $c_1 = .930 \quad d_1 = .020 \quad e_1 = .050$
- $F_2$: $c_2 = .985 \quad d_2 = .010 \quad e_2 = .005$
- $F_3$: $c_3 = .994 \quad d_3 = .005 \quad e_3 = .001$

The probability distribution function of the latency difference is plotted in Fig. 4 for $F_1, F_2,$ and $F_3$. If $x < 1.286$ then the rollback interval is 0 for all three faults and there is no worst-case fault. If $1.286 < x \leq .843$ then the worst-case fault is $F_1$. If $.843 < x \leq .916$ then the worst-case fault is $F_2$, and if $x > .916$ the worst-case fault is $F_3$.

![Fig. 4. The probability distribution function of three faults, each of which is a worst-case fault in turn.](image-url)
The following algorithm forms a greatly reduced list $L_2$ of faults from which the worst-case fault is chosen.

**Algorithm 1.**

1. Initially all faults in list $L_1$ are unmarked and list $L_2$ is empty. Mark those faults in $L_1$ whose $d_j=0$ or whose $e_i=0$.
2. Set $k$ equal to the minimum of the values $d_j/e_i$ corresponding to the unmarked faults in list $L_1$.
3. From the faults with $d_j/e_i$ equal to $k$, choose the fault $F_j$ with minimum valued $d_j$. Set $g=d_j$ and add $F_j$ to list $L_2$.
4. In list $L_1$, mark all the faults with $d_j/e_i$ equal to $k$ and all faults with $d_i \geq g$. If an unmarked fault remains in $L_1$, go to step 2. Otherwise exit.

**Theorem 2.** If a worst-case fault for a given probability $x$ of successful data restoration exists, then it is in the list $L_2$ generated by algorithm 1.

**Proof.** Step 1 eliminates faults for which the network is untested or secure. The rollback interval is undefined in the former case and zero in the latter case. A fault is discarded when it is not added to list $L_2$ in step 3 and then marked in step 4. The probability distribution function $Pr[L_{D_1} \geq n]$ of a discarded fault is always greater than or equal to the probability distribution function of the fault chosen for list $L_2$; it is initially greater or equal at $n=0$ and rises toward 1 at a greater or equal rate as $n$ increases. Hence a discarded fault can never have a rollback interval greater than the fault chosen for list $L_2$. 
Algorithm 1 is surprisingly effective in reducing the search for the worst-case fault. In section 4 a list \( L_1 \) of 156 faults is reduced to a list \( L_2 \) of 2 faults. The worst-case fault is chosen from list \( L_2 \) by finding the fault with the greatest rollback interval for a given probability \( \pi \) of successful data restoration.
THE ROLLBACK INTERVAL OF A FAULT SET IN AN IMPERFECT-SELF-CHECKING NETWORK

The worst-case recovery philosophy fails to assess the beneficial effects of the faults that are not worst-case but may be just as likely to occur. The following fault-set latency measure computes the average effect of all the faults in a fault set.

Definition. The error latency $EL_\mathcal{F}$ (detection latency $DL_\mathcal{F}$) of a fault set $\mathcal{F}$ is the number of input vectors applied to a network while one fault in $\mathcal{F}$ is constantly active, until the first error (detection) due to the fault is observed. Inherent in the definition and use of the fault-set error (detection) latency is an assumption that the one particular fault that becomes active is chosen from $\mathcal{F}$ according to some conditional probability distribution.

Each fault in $\mathcal{F}$ has a conditional probability of being the one fault that occurs, given that some fault in $\mathcal{F}$ must occur. The conditional probabilities must sum to 1.

$$\sum_{i:F_i \in \mathcal{F}} \Pr(F_i | \mathcal{F}) = 1 \quad (3.1)$$

The error latency of a fault set is discussed in detail in 18,141.

Definition. The latency difference $LD_\mathcal{F}$ of a fault set $\mathcal{F}$ is the difference between $DL_\mathcal{F}$ and $EL_\mathcal{F}$, where the network is self-testing for every fault in $\mathcal{F}$ ($LD_\mathcal{F} = DL_\mathcal{F} - EL_\mathcal{F}$).

The probability that the fault-set latency difference is equal to $n$ is obtained by summing the probabilities of the disjoint union of subevents indicated below,
The probability distribution function of the fault-set latency difference $LD_g$ is an average of the probability distribution functions of the latency difference $LD_i$ for each fault $F_i$ in $\mathcal{F}$,

$$Pr[LD_g=n] = \sum_{i:F_i \in \mathcal{F}} Pr[LD_i=n, F_i] = \sum_{i:F_i \in \mathcal{F}} Pr[LD_i=n]Pr(F_i|\mathcal{F}) . \quad (3.2)$$

The rollback interval $rb(x)$ of a fault set $\mathcal{F}$ in a self-testing network is the minimum rollback distance necessary to have probability at least equal to $x$ of successful data restoration from an undiagnosed but detected fault $F_i$ belonging to $\mathcal{F}$.

The fault-set rollback interval $rb(x)_{\mathcal{F}}$ is the minimum value of $n$ that makes the left side of equation (3.3) at least equal to $x$.

When $\mathcal{F}$ is the set of all reasonably likely faults, the probability $x$ specified for $rb(x)_{\mathcal{F}}$ is the overall probability of successful data restoration. The coverage factor $[15,16]$ of a system is the probability of system recovery given a fault. Since successful data restoration is a prerequisite of successful recovery, the probability $x$ is an important
factor of the coverage factor. To say the least, the coverage factor cannot be greater than $x$. Because the mean time to system failure is extremely sensitive to deviations in the value of the coverage factor from $1 \[15,16\]$, it is critical that the rollback distance be great enough to insure a value of $x$ close to 1.

Example 4. The network in Fig. 1 is self-testing for the single faults involving lines 4-51. Table 4 lists the rollback interval for this fault set when each fault in $\mathcal{F}$ is equally-likely.

<table>
<thead>
<tr>
<th>Fault-Set $\mathcal{F}$</th>
<th>$rb(.90)$</th>
<th>$rb(.99)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper-bound</td>
<td>18</td>
<td>34</td>
</tr>
<tr>
<td>worst-case</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>fault-set $\mathcal{F}$</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

$\mathcal{F}$ is the set of single faults on lines 4-51.

Table 4. - Rollback intervals for the network in Fig. 1 (equally-likely inputs).

THE ROLLBACK INTERVAL OF AN IMPERFECT-SELF-CHECKING ARITHMETIC AND LOGIC UNIT

In this section we determine the rollback interval of a moderately complex example, the arithmetic and logic unit (ALU) contained in the imperfect-self-checking network in Fig. 5. The network in Fig. 5 exemplifies the imperfect-self-checking design techniques used in the Stanford SDC-16 computer $[9,17]$. We choose to analyze the ALU since it is the most "insecure" component of the otherwise self-checking SDC-16.
Fig. 5. - A simplified diagram of an imperfect-self-checking ALU network.
The network in Fig. 5 contains a bit-sliced (Fig. 6), ripple-carry ALU composed of separate check symbol and data part sections, and a checker/re-encoder. Operands are encoded in an error-detecting code preserved by arithmetic operations, but not by logic operations.*

The checker is enabled \((y_1 y_2 = \langle 0 1 \rangle)\) during code preserving operations, so that a detection is signaled \((z_1 z_2 = \langle 0 0 \rangle \text{ or } \langle 1 1 \rangle)\) if the preserved check symbol does not agree with the check symbol generated from the output of the ALU data section. The checker is disabled \((y_1 y_2 = \langle 1 0 \rangle)\) during non-preserving operations, allowing errors produced by the ALU data section to escape detection and be re-encoded. In this case an error occurs.

The code is a low-cost residue code [19] with group length \(a=2\). A codeword has a 16-bit data part and a 2-bit check symbol. The check symbol is equal to the data part modulo \(2^a-1=3\), with check symbol \(\langle 1 1 \rangle\) recognized equivalent to \(\langle 0 0 \rangle\). Both the check symbol and data part sections of the ALU have end-around-carry so that arithmetic operations are modulo \(2^2-1\) and modulo \(2^{-16}-1\) respectively (1's complement).

The effect of an arithmetic error on a correct ALU output \(V\) is to change it to \(V + R\) where in coding terminology, \(R\) is the error value. This code detects all error values of the forms \(\pm 2^j\) and \(\pm 2^{j+1} \pm 2^j[17, 20]\). An error value of the form \(\pm 2^{j+1} \pm 2^j\) is not detected.

For example, line 28 stuck-at-1 in a data section bit-slice affects

*There is no error-detecting code short of duplication for logic operations [18]. The preserving arithmetic operations are: PLUS, MINUS, LR, B, B, A, A, all 1's, and all 0's. The logic operations are: AB, A+B, A@B.
Fig. 6. – An ALU bit-slice and operation frequencies.
only the output $F_j$, so it can only cause an error value of $\pm 2^j$. This fault can cause a correct response or a detection during an add. Line 23 stuck-at 1 in a data section bit-slice affects both the output $F_j$ and the carry-out $C_{j+1}$. It causes the error values $-2^j$, $-2^{j+1}+2^j$, and $-2^{j+1}-2^j$, depending on the values of the input lines $A_j$, $B_j$, and $C_j$. This fault may cause a correct response, a detection, or an error during an add.*

The ALU is self-testing for the set of single faults [17]**, insecure for logic operations, and except for lines 23 and 5 stuck-at-1, secure for arithmetic operations.

Lemma 1 simplifies the analysis of the ALU,

**Lemma 1.** Let $c_i(Y_k)$, $d_i(Y_k)$, and $e_i(Y_k)$ be the conditional response probability components of a fault $F_i$ in a network where input subvector $Y$ is fixed at $Y_k$. If input subvector $Y$ is independent, then

$$c_i = \sum_k c_i(Y_k) \Pr[Y_k].$$

The same is true for the response probabilities $d_i$ and $e_i$.

Assuming the control input vector is independent of the operand vectors, lemma 1 permits the calculation of the conditional response probabilities by determining the conditional response probability components for each operation, then taking an average according to the probability of each operation.

*Line 5 stuck-at-1 is the only other single fault in the ALU for which the network of Fig. 5 is not secure during all arithmetic operations.

**In the analysis to follow, the conditional response probabilities of the lines 32 and 33 stuck-at 1 (Fig. 6) are $c_i=1.0$, $d_i=0$, $e_i=0$, so they are excluded from the set of single faults considered.
The analysis strategy is as follows. Simulate each single stuck-at-fault in the bit slice of Fig. 6. Associate the input lines S0-S3, M with a control subvector \( Y \), and the operand lines \( A_j \), \( B_j \) and carry-in line \( C_j \) with an operand subvector. For each fault \( F_i \), and for each operation specified by a control subvector \( Y_k \), determine the error value in the output of the ALU caused by the application of each of the possible operand subvectors. Analyze the error values so that each operand subvector can be classified as eliciting either an error, a detection, or a correct response. The probabilities of these operand subvectors are summed to obtain the conditional response probability components \( c_i(Y_k), d_i(Y_k), \) and \( e_i(Y_k) \). The conditional response probabilities \( c_i, d_i, \) and \( e_i \) are calculated using lemma 1.

An end-around-carry adder as shown in Fig. 5 has a somewhat sequential nature when the input lines \( A_j \) and \( B_j \) to every bit-slice are carry-propagating inputs \( (A_j, B_j = <0, 1> \) or \( <1, 0>) \) [21]. This situation occurs only when a number and its 1's complement are added. The sum may be plus or minus zero (all 0's or all 1's) depending if the carry lines are all 1 or all 0. Both states of the carry lines are stable, and the final state reached depends on the previous condition of the carry lines and the relative propagation delays of the bit-slices, since both \( \pm 0 \) are recognized by the 1's complement number system employed here, this sequential nature is no theoretical problem. The duration of the race before a stable state is reached however, can cause timing problems, so some method to drive the carry lines to a stable state is desirable.

The probabilities of the operand subvectors applied to a bit-slice
Table 5. The probability of applying the operand subvector \( A_j B_j C_j \) to the \( j \)th ALU bit-slice during the indicated operation.

<table>
<thead>
<tr>
<th>( A, B, C )</th>
<th>( \text{AND} )</th>
<th>( \text{OR} )</th>
<th>( \text{NL} )</th>
<th>( \text{LR} )</th>
<th>Plus</th>
<th>Minus</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(-1.25 \times 10^8)</td>
</tr>
<tr>
<td>001</td>
<td>((1-p_a)(1-p_b)(1-p_c))</td>
<td>((1-p_a)(1-p_b))</td>
<td>((1-p_a)(1-p_b))</td>
<td>((1-p_a)(1-p_b))</td>
<td>(-1.25 \times 10^8)</td>
<td>(-1.25 \times 10^8)</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(-1.25 \times 10^8)</td>
</tr>
<tr>
<td>011</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>(-1.25 \times 10^8)</td>
<td>(-1.25 \times 10^8)</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(-1.25 \times 10^8)</td>
</tr>
<tr>
<td>101</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>(-1.25 \times 10^8)</td>
<td>(-1.25 \times 10^8)</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(-1.25 \times 10^8)</td>
</tr>
<tr>
<td>111</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>((1-p_a)p_b)</td>
<td>(-1.25 \times 10^8)</td>
<td>(-1.25 \times 10^8)</td>
</tr>
</tbody>
</table>

For all single faults in the ALU check section:
- \( p_a = 1.0 \)
- \( p_b = 1.0 \)
- \( p_c = 1.0 \)

(because checker is disabled)

Comments:
- \( p_a = \text{Pr} [A_j = 1] \) (data section) = 0.5
- \( p_b = \text{Pr} [B_j = 1] \) (data section) = 0.5
- \( p_c = \text{Pr} [C_j = 1] \) (data section) = 0.5
- \( p_d = \text{Pr} [\text{operand check symbol} = 000] = (1/6)(1 + 1/2^3) = 0.166672 \)
- \( p_{12} = \text{Pr} [\text{operand check symbol} = 001] = (1/6)(1 + 1/2^3) = 0.333333 \)
- \( p_j = \text{Pr} [\text{operand check symbol} = 1,1] = (1/6)(1 + 1/2^3) = 0.333333 \)

* Multiplies \( C_j \)

**Assume bus line \( A_3 \) on \( B_j \) is 1 when an operand is absent.
are listed in Table 5. Table 5 assumes that the carry lines are initially driven to 0 for arithmetic operations [21], that the operand lines \( A_j \) and \( B_j \) are 1 in the absence of an operand, that the signal probabilities [22] of lines \( A_j \) and \( B_j \) in the data section are independent and equal to .5 for an operand, and that the proportions of the check symbols \(<0 0>\) and \(<1 1>\) (both are recognized as 0 modulo 3 [17]) are equal.

Since the operand subvector probabilities are the same for each bit-slice in the check section and for each bit-slice in the data section of the ALU, the conditional response probabilities of the same faults in different bit-slices of those sections are the same. We need only examine one bit slice from each section.

When only faults restricted to a bit-slice are considered, the upper-bound rollback interval for the ALU in Fig. 5 is equal to the upper-bound rollback interval of the bit-slice in Fig. 6. The minimum input-vector probability \( p_{\text{min}} \) for a bit-slice is the minimum of the products of the control and operand subvector probabilities. Using Table 5 and the example operation mix listed in Fig. 6, \( p_{\text{min}} = (.01)(.16667) = .001667 \) and theorem 1 gives the upper-bound rollback interval.

A worst-case single fault for the ALU is among the 156 single faults examined in the check and data section bit-slices (78 faults each). Using algorithm 1, the list \( L_1 \) of 156 faults is reduced to a list \( L_2 \) of 2 faults. The worst-case fault for both entries in Table 6 is 16 stuck-at 1 in the data section bit-slice. Table 6 lists the upper-bound and worst-case rollback intervals for the ALU. Table 6 also lists the fault-set rollback interval for the set of \((78)(18)=1404\) single faults in the ALU.
Table 6 - Rollback intervals for the ALU In Fig. 5.

<table>
<thead>
<tr>
<th></th>
<th>rb(.90)</th>
<th>rb(.99)</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper-bound</td>
<td>1380</td>
<td>2760</td>
</tr>
<tr>
<td>worst-case</td>
<td>42</td>
<td>133</td>
</tr>
<tr>
<td>single-fault set</td>
<td>0</td>
<td>J-6</td>
</tr>
</tbody>
</table>

Pr[1st D before 1st E] = .936

Table 6 clearly demonstrates the smaller values obtained for the approximation to the required rollback distance by the increasingly accurate and costly analysis methods.

In the example operation mix in Fig. 6, the probabilities of the logic operations (AND, OR, and XOR) sum to .10. Increasing the relative proportion of these insecure logic operations should increase the required rollback distance. Fig. 7 plots the rollback interval of the ALU against the probability of a logic operation. Relative operation probabilities within the groups of logic and arithmetic operations remain the same as in Fig. 6. The rollback interval is roughly linear with the probability of a logic operation.
The effectiveness of a self-testing network is best measured by the rollback distance required when a fault is detected. The upper-bound, worst-case, and fault-set rollback intervals are three increasingly accurate measures of this distance.

The analysis directly relates the rollback distance to the probability of successful data restoration. Since the probability of successful data restoration is an upper-bound on the coverage factor, it follows that the overall system reliability strongly depends on the rollback distance. The analysis offers a way to choose the appropriate rollback distance.

An inexpensive, imperfect-self-checking ALU was shown to require a rollback distance of only 16 inputs to have probability .99 of successful data restoration when a single fault is detected.
REFERENCES


