Korrektur transienter Fehler in eingebetteten Speicherelementen

Imhof, Michael E.; Wunderlich, Hans-Joachim

5. GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf (ZuE'11) Vol. 231, Hamburg-Harburg, Germany, 27-29 September 2011

url: https://www.vde-verlag.de/proceedings-de/453357015.html

Abstract: In der vorliegenden Arbeit wird ein Schema zur Korrektur von transienten Fehlern in eingebetteten, pegelgesteuerten Speicherelementen vorgestellt. Das Schema verwendet Struktur- und Informationsredundanz, um Single Event Upsets (SEUs) in Registern zu erkennen und zu korrigieren. Mit geringem Mehraufwand kann ein betroffenes Bit lokalisiert und mit einem hier vorgestellten Bit-Flipping-Latch (BFL) rückgesetzt werden, so dass die Zahl zusätzlicher Taktzyklen im Fehlerfall minimiert wird. Ein Vergleich mit anderen Erkennungs- und Korrekturschemata zeigt einen deutlich reduzierten Hardwaremehraufwand.

Preprint

Korrektur transienter Fehler in eingebetteten Speicherelementen

Michael E. Imhof, Hans-Joachim Wunderlich
Institut für Technische Informatik, Universität Stuttgart, D-70569 Stuttgart

Kurzfassung / Abstract

In der vorliegenden Arbeit wird ein Schema zur Korrektur von transienten Fehlern in eingebetteten, pegelgesteuerten Speicherelementen vorgestellt. Das Schema verwendet Struktur- und Informationsredundanz, um Single Event Upsets (SEUs) in Registern zu erkennen und zu korrigieren. Mit geringem Mehraufwand kann ein betroffenes Bit lokalisiert und mit einem hier vorgestellten Bit-Flipping-Latch (BFL) rückgesetzt werden, so dass die Zahl zusätzlicher Taktzyklen im Fehlerfall minimiert wird. Ein Vergleich mit anderen Erkennungs- und Korrekturschemata zeigt einen deutlich reduzierten Hardwaremehraufwand.

In this paper a soft error correction scheme for embedded level sensitive storage elements is presented. The scheme employs structural- and information-redundancy to detect and correct Single Event Upsets (SEUs) in registers. With low additional hardware overhead the affected bit can be localized and reset with the presented Bit-Flipping-Latch (BFL), thereby minimizing the amount of additional clock cycles in the faulty case. A comparison with other detection and correction schemes shows a significantly lower hardware overhead.

Schlüsselwörter / Keywords

Transiente Fehler, Soft Error, Single Event Upset (SEU), Erkennung, Lokalisierung, Korrektur, Latch, Register
Single Event Effect, Soft Error, Single Event Upset (SEU), Detection, Localization, Correction, Latch, Register

1 Einführung

Transiente Fehler, verursacht durch Partikeleinschläge, beeinträchtigen die Zuverlässigkeit digitaler Systeme. In sequentiellen Schaltungen können sowohl Speicherblöcke als auch freie Logik von Soft Errors betroffen sein [1], und die fortschreitende Technologieskalierung und sinkende Strukturgrößen erhöhen die absoluten Fehlerraten in der ungeschützten freien Logik. Bereits vor einigen Jahren waren die kombinatorische Logik und die sequentiellen Elemente eines typischen Entwurfs für rund 60% der gesamten Soft-Error-Rate (SER) verantwortlich [2].

Der Beitrag kombinatorischer Logik zur SER ist geringer als der Beitrag der sequentiellen Speicherelemente wie Latches, Flip-Flops oder Register [2, 3], da Single Event Transients (SETs) in kombinatorischer Logik durch unterschiedliche Mechanismen gefiltert werden.

In freier Logik eingebettete sequentielle Elemente haben das größte Potential, die Soft Error Rate zu erhöhen. Single Event Upsets (SEUs), die ein Latch direkt betreffen, können den gespeicherten Wert ändern, sind somit direkt am Ausgang des Latches sichtbar und beeinträchtigen oft große Teile einer Schaltung. Der Schutz sequentieller Elemente wie Latches vor SEUs ist unerlässlich, falls eine besonders hohe Robustheit einer Schaltung erforderlich ist.

Es wurde bereits eine Vielzahl an Schemata zum Schutz sequentieller Elemente in freier Logik vorgeschlagen. Strukturredundanz wurde zuerst als Dreifach Modulare Redundanz (triple modular redudancy, TMR [4, 5]) und Dreifach-Latches [6] eingesetzt und später um Zeitredundanz ergänzt. Die Korrektur wird entweder durch Neuberechnung (RAZORII [7]), Wiederherstellung des fehlerfreien Wertes aus einem Schattenelement (GRAAL [8], RAZOR [9], BISER [10]) oder durch Informationsredundanz [11, 12] realisiert. Andere Lösungen filtern explizit alle Eingangssignale (DF-DICE [13]).

Das in [12] vorgestellte Schema analysiert die Fähigkeit von Hammingcodes Register zu schützen, und die Autoren weisen auf einen nicht vernachlässigbaren Hardwaremehraufwand hin.

Das im Folgenden vorgestellte Verfahren basiert auf einem linearen Code und kann Single Event Upsets in Registern, die in freier Logik eingebettet sind, sowohl erkennen als auch korrigieren. Informationsredundanz wird zu allen Registern hinzugefügt und dazu verwendet, Single Event Upsets mit deutlich geringeren Hardwarekosten zu erkennen, als es in [12] berichtet wurde. Eine Korrektur kann dann mittels Neuberechnung durchgeführt werden.

Zusätzlich kann der eingesetzte Code verwendet werden, um das betroffene Bit innerhalb des Registers lokalisieren. Mit dem vorgestellten Bit-Flipping-Latch (BFL) können so Single Event Upsets innerhalb einer Taktperiode korrigiert werden. Treten keine SEUs auf, werden keine zusätzlichen Taktzyklen benötigt und die maximale Betriebsgeschwindigkeit der Schaltung wird

nicht reduziert, da keine Gatter in den Datenpfad eingefügt werden.

Das vorgeschlagene Schema besteht aus den folgenden Teilen:

1) Effiziente Berechnung eines fehlererkennenden und -korrigierenden Codes: Eine Modulo-2 Adress-Charakteristik wird verwendet, um eine log(n)-bit Prüfsumme des n-bit Registerinhalts zu bestimmen. Die Berechnung der Charakteristik wird mittels speziellen Standardzellen implementiert und benötigt nur geringen Hardwaremehraufwand.
2) Geschützte Speicherung der Fehlerbedingung: Die Charakteristik erlaubt die Erkennung eines ungewollten Zustandswechsels und die Lokalisation des betroffenen Bits. Zusätzlich wird die gespeicherte Charakteristik mit einem Paritätsbit geschützt.
3) Bit-Flipping-Latch (BFL): Der lokalisierte SEU kann mit Hilfe der entworfenen BFL-Standardzelle in einem Taktzyklus korrigiert werden.

Im Verlauf dieses Beitrags wird gezeigt, dass dieses Schema die folgenden Vorteile bietet:

i) Alle Single Event Upsets können mit geringem Hardwaremehraufwand erkannt und durch Neuberechnung korrigiert werden.
ii) Der Time Vulnerability Faktor (TVF), d.h. die ungeschützte Zeitspanne eines Registers, ist Null.
iii) Das gesamte Schema benötigt nur geringen Hardwaremehraufwand.

Der Rest dieses Beitrags gliedert sich wie folgt: Kapitel 2 beschreibt die verwandten Arbeiten zur Erkennung und Korrektur von Soft Errors, während Kapitel 3 die grundlegenden Konzepte der vorgestellten Schemata aufzeigt. Kapitel 4 schildert die flächeneffiziente Implementierung und Kapitel 5 diskutiert, wie die gespeicherte Fehlerbedingung gegen Single Event Upsets geschützt werden kann. Basierend auf dieser Information beschreibt Kapitel 6 den Entwurf der neuen BFL Standardzelle, die ihren internen Zustand invertieren kann, und wie diese zur Korrektur von Soft Errors verwendet werden kann. Kapitel 7 diskutiert kurz das Zeitverhalten und die zeitliche Vulnerabilität TVF. Kapitel 8 beschreibt die experimentellen Ergebnisse sowie einen Vergleich zu anderen Schemata.

2 Stand der Technik

Dreifach Modulare Redundanz (triple modular redundancy, TMR) [4, 5] beschreibt den Aufbau eines zuverlässigen Systems aus unzuverlässigen Komponenten. Die grundlegenden Bausteine werden verdreifacht und ihre Resultate mittels eines synchronen Mehrheitsentscheiders kombiniert. Triple Latches erweitern TMR um Zeitredundanz und einen asynchronen Mehrheitsentscheider um die Erkennungswahrscheinlichkeit zu verbessern [6]. Eine Verdreifachung erlaubt die Korrektur aller Upsets, die nur ein Bit betreffen. Der Hardwaremehraufwand von ungefähr +400% (zwei weitere Latches plus der Mehrheitsentscheider) kann durch verschiedene Techniken reduziert werden.

Die meisten Schemata verwenden lokale Redundanz für jedes Bit. Die Implementierung pro Bit limitiert aber die erreichbare Effizienz bezüglich des Hardwaremehraufwands. Die bekanntesten Schemata beinhalten BISER [2, 10], GRAAL [8], RAZOR [9], DF-DICE [13] und RAZOR2 [7].

Das BISER Schema [2, 10] kombiniert ein Latch und den Mehrheitsentscheider aus TMR in einem asynchronen Mehrheitsentscheider, dem C-Element. Falls das Register Teil eines Prüfpfads ist, kann die dafür hinzugefügte Logik zur Implementierung der beiden verbleibenden Latches verwendet werden. BISER erkennt Einzelbitfehler und verhindert deren Propagierung an den Ausgang des Mehrheitsentscheiders.

Das GRAAL Schema basiert auf Struktur- und Zeitredundanz [8] für pegelgesteuerte Entwürfe. Der Wert eines redundanten sequentiellen Elements wird mit dem Wert des funktionalen Latches verglichen. Im Fall eines Fehlers wird der korrekte Wert aus dem Schattenelement wiederhergestellt.

Der RAZOR Ansatz [9] ähnelt dem GRAAL Schema. Beide nutzen dieselben Grundgatter, aber RAZOR unterstützt einen flankengesteuerten Entwurfsstil. Im Fall eines erkannten Fehlers stellt das Schema den Flip-Flop-Inhalt aus den Latches für alle Bits eines Registers wieder her. Zur selben Zeit wird der Takt aller weiteren Register des Moduls unterdrückt und der fehlerbehaftete Inhalt des Flip-Flops damit an der Ausbreitung gehindert.

Das DF-DICE Speicherelement [13] basiert auf Pulsfilterung, mit der transiente Pulse an den Eingängen bis zu einer gegebenen Dauer gefiltert werden. Single Event Upsets im internen Speicherelement ändern den gespeicherten Wert und werden nicht erkannt.

Das von Das et. al in [7] vorgeschlagene RAZOR2 Schema erlaubt die Erkennung von transienten Fehlern in Registern, wobei unerwünschte Transitionen im zustandsspeichernden Latch als Fehler erkannt werden. Die Korrektur wird mittels Neuberechnung durchgeführt. Diese benötigt mehrere Taktzyklen um das korrekte Ergebnis zu erzeugen.

Die in [11] vorgeschlagene zeitredundante Parität nutzt einen Paritätsbaum um SEUs mittels Informationsredundanz zu erkennen. Dabei ist die Lokalisierung des betroffenen Bits nicht möglich und die Korrektur erfolgt durch Neuberechnung. [12] schlägt einen Schutz mittels Hamming-Codes vor. Das Schema kann SEUs erkennen und korrigieren, bringt dabei aber einen hohen Hardwaremehraufwand mit sich. Der Einsatz von Informationsredundanz für ein ganzes Register ist vielversprechend, muss aber sorgfältig entworfen und implementiert werden, um die eingeführten Nachteile auf ein wirtschaftliches Maß zu begrenzen.

Die meisten Schemata für in freier Logik eingebettete sequentielle Elemente führen Strukturredundanz für jedes Bit des Registers ein. Dies ermöglicht die Erkennung von Single Event Upsets durch den Vergleich des originalen Werts mit dem redundanten Wert eines Bits. Die Korrektur wird mittels Mehrheitsentscheid, der Wiederherstellung des korrekten Werts aus der redundanten Kopie oder mittels Neuberechnung durchgeführt.

Negative Eigenschaften der diskutierten Lösungen sind:

- Strukturredundanz kombiniert mit einem Mehrheitsentscheid resultiert in einem hohen Flächenmehraufwand.
- Die Wiederherstellung des korrekten Wertes aus einem Schattenelement erfordert ebenfalls nicht vernachlässigbaren Flächenmehraufwand.
- Wird der korrekte Wert durch Neuberechnung generiert, wird zwar der Flächenaufwand reduziert, aber die Laufzeit im Fehlerfall deutlich verlängert.
- Der bislang vorgeschlagene Einsatz von Informationsredundanz erlaubt die flächeneffiziente Erkennung von SEUs, falls ein Paritätsbit verwendet wird. Da eine Lokalisierung aber nicht möglich ist, muss der korrekte Wert neu berechnet werden.

3 Korrekturablauf

Bild 1-a zeigt ein ungeschütztes Register aus n Latches und Bild 1-b zeigt eine abstrakte Sicht der vorgeschlagenen Erkennung transienter Fehler. Das Register wird mit einer flächeneffizienten Berechnung einer Modulo-2 Adress-Charakteristik ergänzt. Die Referenzcharakteristik wird in $log(n)$ zusätzlichen Latches gespeichert und mit der aktuellen Charakteristik verglichen. Falls eine Differenz erkannt wird, wird das $fail$ Signal auf '1' gesetzt und stößt die Neuberechnung an.

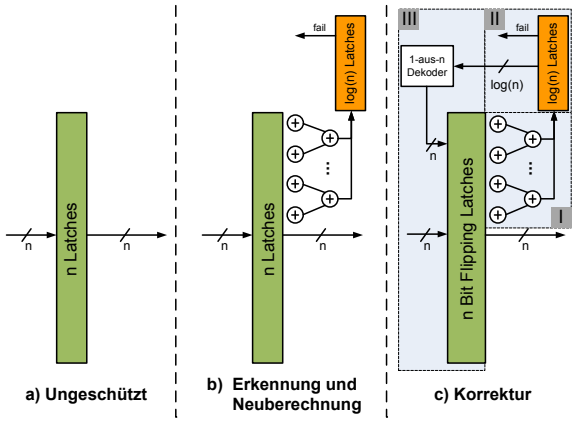

Bild 1 Vorgeschlagene Konfigurationen

Falls eine schnellere Korrektur als durch Neuberechnung implementiert werden soll, kann das Schema durch einen Korrekturmechanismus erweitert werden (Bild 1-c). Die n Latches werden durch n Bit-Flipping-Latches ersetzt, die ihren internen Zustand invertieren können. Aus der Differenz der aktuellen und der Referenzcharakteristik lässt sich die Adresse des betroffenen Bits ableiten. Die Adresse wird dekodiert und zur Steuerung der BF-Latches genutzt, die den korrekten Zustand in einem zusätzlichen Taktzyklus wiederherstellen. Für alle anderen sequentiellen Elemente wird während dieser Phase mittels des $fail$ Signals der Takt unterdrückt.

Bei den vorgestellten Konfigurationen werden keine zusätzlichen Elemente in den Datenpfad eingefügt, so dass im fehlerfreien Fall keine zusätzliche Verzögerungen hinzugefügt werden. Falls ein transienter Fehler auftritt, wird entweder die globale Neuberechnung angestoßen (Bild 1-b) oder es wird ein zusätzlicher Taktzyklus verwendet, um den Effekt des transienten Fehlers zu korrigieren (Bild 1-c).

Kapitel 4 stellt die Eigenschaften des verwendeten Codes und seine effiziente Implementierung vor (Block I in Bild 1-c). Kapitel 5 erklärt, wie die gespeicherte Fehlerbedingung geschützt werden kann (Block II). Kapitel 6 beschreibt das für die Korrektur entwickelte Bit-Flipping Latch (Block III).

4 Modulo-2 Address-Charakteristik

Das vorgeschlagene Schema nutzt eine auf räumlicher Redundanz basierende Modulo-2 Adress-Charakteristik, um den Registerinhalt zu schützen. Die Anwendung dieser Charakteristik für den transparenten Test regulärer Speicherfelder wurde in [14, 15] vorgeschlagen. Diese transparente Technik wurde in [16] für freie Logik angepasst und für die Fehlererkennung und -lokalisierung innerhalb von - in freier Logik eingebetteten - Registern verwendet.

Die Modulo-2 Adress-Charakteristik eines Registers wird durch das bitweise XOR der Registeradressen berechnet, die eine 1 enthalten (Bild 2). Ihre Erkennungseigenschaften sind mit denen von Hammingcodes identisch, die Einzelfehler korrigieren können (SEC). Die Charakteristik kann aber effizienter berechnet und einfacher zur Korrektur benutzt werden, da sich die Adresse des fehlerbehafteten Bits als bitweises XOR der Referenz- und der aktuellen Charakteristik ergibt.

Beispiel (Bild 2) Sei R ein Register mit n Bit ($|R| = n$) und den Werten r_{adr} ($1 \leq adr \leq n$), wobei adr die Adresse des entsprechenden Bits ist. Die Modulo-2 Charakteristik c wird dann als das bitweise XOR aller Adressen adr berechnet, die eine 1 enthalten ($r_{adr} = 1$). Das Bit r_0 wird nicht genutzt, da die Adresse 0 nicht zu c beiträgt. Daher gilt $|c(R)| = \lceil log_2(n+1) \rceil$. Somit ist $c_t(R)$ die Charakteristik von R zum Zeitpunkt t mit $|c_t(R)| = \lceil log_2(n+1) \rceil$.

Um einen Fehler zu erkennen, wird die Charakteristik des ursprünglichen Registerinhalts zum Zeitpunkt $t1$ bestimmt und in $|c_{t1}(R)|$ zusätzlichen sequentiellen Elementen gespeichert. Wir nennen $c_{t1}(R)$ die Referenz Charakteristik c_{ref}.

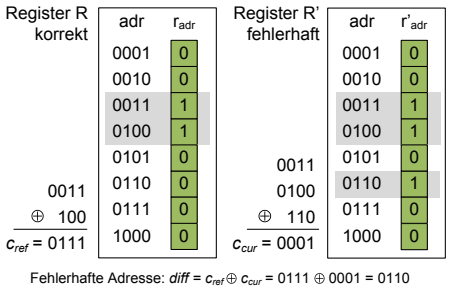

Bild 2 Modulo-2 Address-Charakteristik

Ändert ein transienter Fehler den Registerinhalt von R nach R', so unterscheidet sich die zum Zeitpunkt $t2$ ($t1 < t2$) berechnete aktuelle Charakteristik $c_{cur} := c_{t2}(R')$ von c_{ref}.

Das Schema muss unterscheiden, ob der transiente Fehler das Register R beeinflusst und somit c_{cur} geändert hat, oder ob er nur die Referenzcharakteristik c_{ref} verändert hat. Wir nehmen zunächst an, dass transiente Fehler nur in R vorkommen, und beschreiben später die Unterscheidung von Fehlern in c_{cur} oder c_{ref}.

Wird ein Fehler erkannt, so enthält die Differenz *diff*$= c_{ref} \oplus c_{cur}$ die Adresse des betroffenen Bits. Wie bei SECDED Codes üblich, ist das Schema nicht in der Lage, Mehrfachfehler zu korrigieren.

Flächeneffiziente Implementierung: Die Charakteristik-Berechnung kann mittels XOR2 Standardzellen effizient implementiert werden [16]. Der Mehraufwand für das Routing wird minimiert, wenn nur signifikante Bits zwischen den Ebenen ausgetauscht werden (Bild 3).

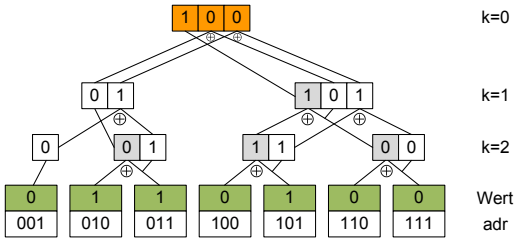

Bild 3 Block I) Flächeneffiziente Charakteristik Berechnung.

5 Geschützte Speicherung der Fehlerbedingung

Nehmen wir nun an, dass transiente Fehler nicht nur das Register R sondern auch das zusätzlich für die Speicherung von c_{ref} hinzugefügte Register verändern können (Bild 4). Ist eines der die Referenz-Charakteristik speichernden Latches von einem transienten Fehler betroffen, weist die Differenz zwischen c_{ref} und c_{cur} auf einen Fehler hin und löst die Korrektur aus, obwohl die Daten auf dem Datenpfad nicht betroffen und korrekt sind. Um dieses Verhalten zu verhindern, ist auch die Korrektheit der Referenz-Charakteristik c_{ref} zu prüfen, sobald eine Differenz beobachtet wird.

Dies wird mit einem Paritätsbit $p(c_{ref})$ in einem zusätzlichen Latch erreicht (Bild 4). Diese Lösung kann zwischen den für einen transienten Fehler möglichen Orten unterscheiden, während der Hardwaremehraufwand minimiert wird. Die Orte sind:

- Original Register (R): Der Charakteristik-Schutz zeigt eine Differenz ($c_{ref} \neq c_{cur}$), das Paritätsbit nicht ($p(c_{ref}) = p(c_{cur})$). Der Fehler hat die Daten auf dem Datenpfad geändert, eine Korrektur wird ausgelöst.
- Referenz-Charakteristik (c_{ref}): Referenz-Charakteristik und Paritätsbit zeigen eine Differenz ($c_{ref} \neq c_{cur}$ und $p(c_{ref}) \neq p(c_{cur})$). Der Fehler hat c_{ref} verändert, nicht die Daten des Datenpfads, eine Korrektur ist nicht nötig.
- Parität ($p(c_{ref})$): Die Referenz-Charakteristik zeigt keine Änderung, aber das Paritätsbit ($c_{ref} = c_{cur}$ und $p(c_{ref}) \neq p(c_{cur})$). Die Referenzparität ($p(c_{ref})$) wurde geändert, sonst nichts. Die Daten auf dem Datenpfad sind korrekt, es ist keine Korrektur nötig.

Das Korrektursignal $correct$ wird somit als $correct = (c_{ref} \neq c_{cur}) \wedge (p(c_{ref}) = p(c_{cur}))$ definiert. Das Hinzufügen des Paritätsschutzes von c_{ref} ermöglicht dem Schema, jeden transienten Fehler in den eingebetteten Speicherelementen zu erkennen und eine Korrektur durchzuführen, sobald die Daten auf dem Datenpfad verändert wurden.

Fehlererkennung: Alle Einzelfehler können erkannt, korrekt lokalisiert und korrigiert werden. Doppelfehler können erkannt werden aber nicht korrigiert werden. Die Erkennung weiterer Mehrfachfehler kann nicht garantiert werden. Generell kann der Hammingabstand des verwendeten Codes erhöht werden um eine Erkennung, Lokalisierung und somit Korrektur von Mehrfachfehlern zu ermöglichen.

6 Bit-Flipping-Latch

Das vorhergehende Kapitel hat gezeigt, wie ein transienter Fehler erkannt und lokalisiert werden kann. Dieses Kapitel erweitert das Schema um eine Korrektur auf Bitebene. Die während der Lokalisierung gewonnene Information wird verwendet, um das betroffene

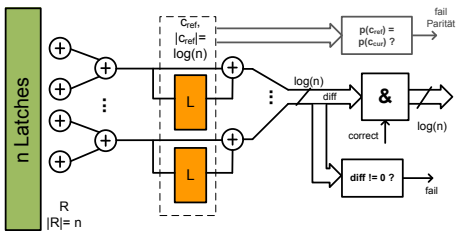

Bild 4 Block II) Bestimmung und Schutz der Fehlerbedingung

Bit zu invertieren, während der Zustand aller anderen Bits beibehalten wird.

Die Lokalisierung verfälschter Bits geschieht mit der Differenz *diff* aus c_{ref} und c_{cur}. Diese wird mit einem 1-aus-n Dekoder in n Korrektursignale dekodiert. Das Korrekturschema ist in Bild 4 verdeutlicht, es verwendet sogenannte Bit-Flipping-Latches (BFL). Diese haben gegenüber anderen Lösungen, den Latchinhalt zu invertieren, den Vorteil, dass keine zusätzlichen Gatter oder Multiplexer im Datenpfad liegen und das Zeitverhalten beeinträchtigen.

Überlicherweise besteht ein Latch aus zwei Invertern (INV) und zwei Transmissionsgattern (TG), die durch ein Paar Steuersignale $\{L, \overline{L}\}$ angesteuert werden. Dieses legt fest, ob ein neuer Wert aus D übernommen wird (das Latch ist transparent) oder ob der interne Zustand beibehalten wird (das Latch sperrt).

Das *Bit-Flipping-Latch (BFL)* ist eine Erweiterung, die es erlaubt, durch einen zusätzlichen Steuereingang $\{HI, \overline{HI}\}$ seinen Inhalt zu invertieren (Bild 5).

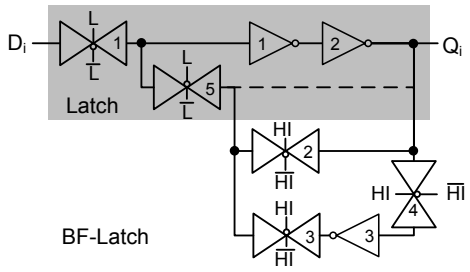

Bild 5 Block III) Bit-Flipping-Latch: Schematischer Aufbau

Für $\{HI, \overline{HI}\} = \{1, 0\}$ wird die obere Schleife durchlaufen, und das BFL arbeitet wie ein normales Latch ohne Beeinträchtigung des Zeitverhaltens. Ist jedoch $\{HI, \overline{HI}\} = \{0, 1\}$, sind sowohl die obere als auch die untere Schleife blockiert, aber es wird als Wert die Ausgabe des unteren Inverters übernommen. Die Blockade beider Schleifen verhindert eine Oszillation, während der Zustand des BFL invertiert wird.

7 Zeitverhalten

Bild 6-a veranschaulicht das Zeitverhalten eines ungeschützten Registers (vgl. Bild 1-a), während Bild 6-b das Zeitverhalten des vorgeschlagenen Korrekturschemas zeigt (Bild 1-c). Ein transienter Fehler trifft das Register A zum Zeitpunkt t_1 und ist bei t_2 an seinem Ausgang sichtbar. Im ungeschützten Fall übernimmt Register B die durch die Schaltung propagierten fehlerbehafteten Daten (Bild 6-a). Das vorgestellte Schema erkennt die Änderung der Daten bei t_3 und unterdrückt den Takt aller anderen Register (Bild 6-b). Der Fehlereffekt wird somit eingedämmt und eine Korrektur ist zum Zeitpunkt t_4 ohne den Verlust von Daten möglich. Das fallende $fail$ Signal zum Zeitpunkt t_5 zeigt die erfolgreiche Korrektur an.

Diese theoretischen Betrachtungen zeigen, dass es kein Zeitfenster gibt, in dem das Register ungeschützt ist

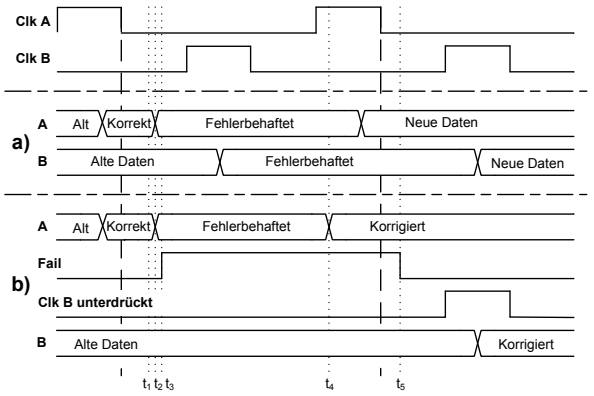

Bild 6 Zeitverhalten: a) Ungeschütztes Register (vgl. Bild 1-a), b) Geschütztes Register mit Korrektur (vgl. Bild 1-c)

und der „Time Vulnerability Factor“ (TVF) bei Null liegt. Die im nächsten Kapitel vorgestellten Simulationen bestätigen dies.

8 Experimentelle Ergebnisse

Das oben beschriebene Bit-Flipping-Latch ist im Full-Custom-Stil für das Predictive Technology Model (PTM [17]) in 45nm Technologie entworfen worden und als Standardzelle in die OpenCellLibrary (OCL [18]) eingebracht worden. Sowohl die weiteren Zellen zur Implementierung des vorgestellten Schemas als auch die Vergleichszellen zur Bestimmung des Hardwaremehraufwands und der Vergleichszeiten stammen aus dieser Bibliothek.

Alle im Folgenden dargestellten Flächenresultate beinhalten den kompletten Mehraufwand für die zusätzliche Verdrahtung, um die Charakteristik zu berechnen, zu speichern und zu schützen, sowie um den Korrekturvektor an die Bit-Flipping Latches zurückzuführen und diese zu steuern. Es wird nur ein einzelnes globales Signal benötigt, um die Neuberechnung anzustoßen oder den Takt für einen Zyklus zur Korrektur zu unterdrücken.

8.1 Zeitverhalten

Fehlerfreier Fall: Um eine mögliche Änderung des Zeitverhaltens durch Einsatz des Korrekturschemas im fehlerfreien Fall zu bestimmen, wird die Verzögerung des Bit-Flipping-Latches und des „low enable“ Latches aus der OCL mittels SPICE bestimmt. Beide Latches werden mit einem Inverter (INV_X1) als Last am Ausgang simuliert und es wird der Zeitpunkt bestimmt, an dem das Ausgangssignal 10% (logische Null) bzw. 90% der Nominalspannung (logische Eins) erreicht. Am Eingang werden jeweils eine steigende und eine fallende Flanke mit einer Flankensteilheit von $22\,V/ns$ angelegt, d.h. der Nominalwert von 1.1 bzw. 0 Volt wird nach $0,05$ Nanosekunden erreicht. Der Ausgang des OCL Latches erreicht bei einer fallenden Flanke nach $0,0837\,ns$ $0,11\,V$. Bei der steigenden Flanke werden nach $0,0707\,ns$

0.99 Volt erreicht. Das Bit-Flipping-Latch hat bei einer fallenden Flanke eine Verzögerung von $0,0723\,ns$ und bei einer steigenden Flanke eine Verzögerung von $0,0691\,ns$. Somit ist das BFL in der Simulation sogar noch schneller als das standardmäßige Latch aus der Bibliothek, was mit der sorgfältigen Skalierung der Transistoren erklärt werden kann. Das Zeitverhalten einer Schaltung wird somit durch den Einsatz der Bit-Flipping-Latches nicht negativ beeinträchtigt.

Fehlerbehafteter Fall: Um den „Time Vulnerability Factor“ (TVF) des vorgeschlagenen Schemas zu bestimmen werden Soft Errors in ein 8-Bit Register injiziert, das mit dem vorgeschlagenen Korrektur-Schema geschützt ist (Register A in Bild 6). Der verwendete Takt hatte eine Periode von 4 ns mit einer 25% High-Phase (Clk A, wie in Bild 6 dargestellt). Register B (mit Clk B) bezeichnet ein im Datenpfad folgendes Register und dient der Veranschaulichung der verwendeten Taktunterdrückung. Währen der High-Phase des Taktes sind die Latches transparent, Single Event Upsets können zu Glitches führen, aber nicht permanent den sequentiellen Zustand ändern. Während der Low-Phase sperren die Latches und sind für Single Event Upsets anfällig. Es wurde eine Reihe von Simulationsexperimenten durchgeführt, in der ein SEU in ein zufällig gewähltes Latch injiziert wird. Dazu wird der Ausgang des ersten Rückkopplungsinverters auf den invertierten Simulationswert gesetzt (Injektionszeitpunkt). Eine Testbench zeichnet dann folgende Zeitpunkte auf: Sichtbarkeit am Register-Ausgang, steigendes Fehlersignal das einen Upset anzeigt, Sichtbarkeit des korrigierten Werts am Ausgang und ein fallendes Fehlersignal. Die Experimente wurden für die komplette Low-Phase von 3 ns durchgeführt, wobei der Injektionszeitpunkt pro Experiment um $100\,ps$ nach hinten verschoben wurde. Die Resultate in Tabelle 1 zeigen, dass SEUs während der kompletten Low-Phase erkannt und korrigiert werden.

Beispiel: In Bild 6 b) entspricht die mit „200 (bit 6)“ beginnende Zeile dem Experiment, in dem ein Fehler zum Zeitpunkt $t_1 = 200\,ps$ injiziert wird, bei $t_2 = 300\,ps$ am Ausgang sichtbar wird und bei $t_3 = 400\,ps$ erkannt wird. Der Fehler wird bei $t_4 = 3200\,ps$ korrigiert und das fallende Fehlersignal zeigt bei $t_5 = 4300\,ps$ eine erfolgreiche Korrektur an.

Zeit (ps) Injektion	Sichtbar am Ausgang	Erkannt (fail steigt)	Korrigiert am Ausgang	Ende (fail fällt)
0 (bit 4)	100	300	3200	4300
100 (bit 6)	200	300	3200	4300
200 (bit 6)	300	400	3200	4300
...	...	...	...	...
2600 (bit 0)	2700	2800	3200	4300
2700 (bit 1)	2800	2900	3200	4300
2800 (bit 5)	2900	3000	3200	4300

Tabelle 1 Zeitverhalten: 8-Bit Register mit Erkennung & Korrektur

8.2 Erkennung auf Registerebene

Ein ungeschütztes Register wurde mittels der „high enable“ Latches aus der OCL implementiert. Das „high enable“ Latch DLH_X1 hat eine Zellhöhe von 1.4 μm, eine Breite von 2.09 μm und eine Fläche von 2.929 μm^2. Diese ungeschützte Fläche wird im Folgenden als Bezugsgröße verwendet. Der Flächenmehraufwand wurde für die mit dem vorgeschlagenen Erkennungsschema ausgerüsteten Register für unterschiedliche Registergrößen bestimmt. Die Erkennung erweitert den n-bit Register Block (bestehend auf DLH_X1 Zellen) um die Charakteristikberechnung, um das zusätzliche Register zur Speicherung von c_{ref} und um den Vergleicher. Ein ODER-Baum aggregiert die berechnete Differenz in ein einzelnes Signal, und ein zusätzlicher Paritätsschutz des Registers für c_{ref} unterscheidet, ob R oder c_{ref} betroffen sind. Tabelle 2 zeigt die Ergebnisse für unterschiedliche Registergrößen. Die Spalte 2 enthält die Registergröße der ungeschützten Register in μm^2. Die Spalten 3 und 4 zeigen die erforderliche sequentielle und kombinatorische Flächen, um die Erkennung zu implementieren, während die letzte Spalte den Hardwaremehraufwand gegenüber der ungeschützten Implementierung enthält.

Register Größe	Ungeschützt (Bild 1-a)	Fehlererkennend (Bild 1-b) Seq	Comb	Mehraufwand
3	8.78	18.09	13.83	+263.55%
7	20.48	31.92	26.87	+187.06%
15	43.89	57.46	53.2	+152.13%
31	90.71	106.4	105.34	+133.43%
63	184.34	202.16	207.48	+122.22%
127	371.6	391.55	412.57	+116.39%

Tabelle 2 Hardwaremehraufwand - Erkennung (μm^2)

Der beobachtete Flächenmehraufwand hängt von der Registergröße ab und fällt mit zunehmender Registergröße. Die Ausrüstung eines Registers mit der vorgeschlagenen Erkennung bringt einen Mehraufwand zwischen +188% für ein 7-Bit Register und +117% für ein 127-Bit Register mit sich. Bereits aus dieser Tabelle ist zu erkennen, dass der hier vorgestellte Ansatz ab einer Registergröße von 7 günstiger als TMR Lösungen ist, die einen Flächenmehraufwand von deutlich mehr als 200% erfordern.

Für die anderen in der Literatur vorgestellten Verfahren sind keine Flächenberechnungen in der hier verwendeten Technologie bekannt, so dass im folgenden lediglich Transistorzahlen verglichen werden können. Für die Fehlererkennung beschränken wir uns auf den Vergleich mit RAZOR2, das nach [7] aus drei Teilen besteht. Dem Latch, einer Transitionserkennung (transition-detector, TD) und einer Takterzeugung für die Erkennung (detection clock generator, DC), die für mehrere Latches verwendet werden kann. Das Schema benötigt 47 Transistoren, falls die Erkennung und Taktgenerierung für jedes Latch implementiert werden. Falls der Takt für mehrere Latches erzeugt wird, werden 39 Transistoren benötigt, plus 8 Transisto-

ren für die globale Takterzeugung. Ein zusätzliches ODER-Gatter wird für die Aggregation der Fehlerinformation auf Registerebene benötigt. Der Mehraufwand für die Implementierung des RAZOR2 Schemas beträgt somit im schlechtesten Fall +537.5% und +437.5% im besten Fall.

8.3 Korrektur auf Bitebene

Der Zusatzaufwand für die Korrektur auf Bitebene setzt sich aus dem Mehraufwand für das Bit-Flipping-Latch (BFL), aus der Dekodierlogik für die Fehlerlokalisierung und der zugehörigen Verdrahtung zusammen. Das BFL wurde im Full-Custom Stil als Standardzelle entworfen und enthält zusätzlich einen Inverter und drei Transmissionsgatter. Bild 7 zeigt das Layout der Bit-Flipping Latch Standardzelle. Die Reihenfolge der einzelnen Gatter von links nach rechts ist wie folgt: TG1, TG5, INV1, INV2, TG2, TG4, TG3, INV3.

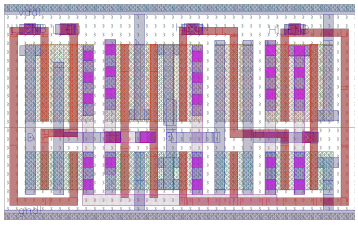

Bild 7 BFLATCH_X1: Latch mit invertierender Rückkopplung

Die Zelle wurde entsprechend der Entwurfsregeln des FreePDK Process Design Kits entworfen [19]. Die Zellhöhe beträgt - wie in der OpenCell Bibliothek - 1.4 μm. Das Bit Flipping Latch hat eine Breite von 2.28 μm und eine Gesamtfläche von 3.192 μm^2. Verglichen mit dem OCL DLH_X1 Latch beträgt der für die invertierende Rückkopplung nötige Flächenmehraufwand nur 9%.

Das gesamte Korrekturschema wurde mit der beschriebenen Standardzelle und weiteren Zellen aus der OCL für unterschiedliche Registergrößen synthetisiert. Die Ergebnisse finden sich in Tabelle 3.

Register Größe	Ungeschützt (Bild 1-a)	Fehlerkorrigierend (Bild 1-c)		
		Seq	Comb	Mehraufwand
3	8.78	18.89	20.75	+351.37%
7	20.48	33.78	44.16	+280.57%
15	43.89	61.45	87.25	+238.78%
31	90.71	114.65	169.18	+212.89%
63	184.34	218.92	326.65	+195.96%
127	371.6	425.33	623.5	+182.25%

Tabelle 3 Flächenmehraufwand - Erkennung & Korrektur (μm^2)

Der Flächenmehraufwand sinkt mit steigender Registergröße. Für ein 7-Bit Register beträgt der Mehraufwand +281%, während die Implementierung der Korrektur für ein 127-Bit Register im Vergleich zur ungeschützten Implementierung +183% zusätzliche Fläche benötigt.

Vergleich mit anderen Schemata: Tabelle 4 zeigt den Mehraufwand an Transistoren der Erkennungs- und Korrekturschemata aus Kapitel 2. Die Spalten 2 und 3 enthalten die verwendeten Basisgatter zusammen mit der für die Implementierung benötigten Transistoranzahl. Die restlichen Spalten stellen für TMR-basierte Korrekturverfahren sowie für GRAAL und RAZOR1 und das vorgeschlagene Verfahren BFL die benötigten Transistoren dar, die nach Gattertyp aufgeschlüsselt wurden. Die Zeile „Register“ enthält dabei den beim GRAAL- und RAZOR1-Schema für die Erzeugung der lokalen Fehlersignale benötigen ODER-Baum. TMR resultiert in einem Hardwaremehraufwand von 400%. Durch das GRAAL-Schema geschützte Latches benötigen jeweils 34 Transistoren. Die für jedes Bit generierten Error-Signale müssen pro Register durch einen ODER-Baum aggregiert werden, was 4 Transistoren pro Bit benötigt. Gegenüber den 8 Transistoren einer ungeschützten Implementierung ergibt sich einen Mehraufwand von 375%. Falls für RAZOR1 die effiziente Implementierung aus [9] verwendet wird, ergibt sich eine gesamte Transistoranzahl mit Metastabilitätserkennung von 52, während das Schema ohne diese Erweiterung 38 Transistoren benötigt (wie in Tabelle 4 dargestellt). Inklusive dem ODER-Baum pro Register ergeben sich 42 Transistoren und ein Hardware-Mehraufwand von 425%. Der Mehraufwand des vorgeschlagene Verfahrens BFL hängt von der Registergröße ab. Bezogen auf die Transistoranzahl beträgt er 348%. Durch die effiziente Implementierung der Korrektur mittels der vorgestellten Bit-Flipping-Latches ist der reale Flächenmehraufwand von 185% deutlich geringer.

9 Zusammenfassung

Es wurde ein flächeneffizientes Schema zur Korrektur von Soft Errors in eingebetteten Speicherelementen präsentiert. Es erkennt und lokalisiert SEUs in Registern mit einem von der Registergröße abhängigen Flächenmehraufwand zwischen +188% (7-bit) und +117% (127-bit). Die geschützte Speicherung der Prüfbits eliminiert falsch-positive Korrekturen, die durch die Prüfsumme betreffende SEUs hervorgerufen werden können. Das vorgestellte Bit-Flipping-Latch ermöglicht zusammen mit der auf Registerebene gewonnenen Lokationsinformation eine effiziente Korrektur von SEUs auf der Bitebene. Der Flächenmehraufwand für die Korrektur reicht von +281% für ein 7-bit Register bis +183% für ein 127-bit Register. Es wurde gezeigt, dass der Schutz eingebetteter Speicher mittels Prüfsummen durchführbar ist und in Kombination mit einer Korrektur auf Gatterebene in einem gegenüber anderen Lösungen deutlich reduzierten Mehraufwand resultiert.

10 Danksagung

Die vorliegende Arbeit wurde durch die Deutsche Forschungsgemeinschaft im Rahmen des Projekts „Realtest“ (Wu245/5-2) gefördert.

	Gatter	# Transistoren	TMR	GRAAL	RAZOR1	BFL - 32 Bit	BFL - 128 Bit
Pro Bit	LATCH	8	24	8	24	64	80
	BFL	16				512	2048
	FF	16		16			
	Voter	16	16				
	XOR/XNOR	6		6	6	352	1408
	OR/NOR	4				116	144
	MUX	4		4	4		
	INV	2			4	78	290
	AND/NAND	4				172	616
Total			40	34	38	1294	4586
	OR	4		4	4		
Register			40	38	42	40, 44	35, 831
+%			+400%	+375%	+425%	+406%	+348%

Tabelle 4 Transistormehraufwand der Korrekturschemata

Literatur

[1] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, May–June 2005.

[2] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *IEEE Computer*, vol. 38, no. 2, pp. 43–52, 2005.

[3] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *IEEE VLSI Test Symposium (VTS99)*. IEEE, 1999, pp. 86–94.

[4] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*, C. E. Shannon and J. McCarthy, Eds. Princeton University Press, 1956.

[5] R. E. Lyons and W. Vanderkulk, "The use of triple modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[6] J. Wang, W. Wong, S. Wolday, B. Cronquist, J. McCollum, R. Katz, and I. Kleyner, "Single event upset and hardening in 0.15 μm antifuse-based field programmable gate array," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2158–2166, 2003.

[7] S. Das, C. Tokunaga, S. Pant, W. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, 2009.

[8] M. Nicolaidis, "Graal: a new fault tolerant design paradigm for mitigating the flaws of deep nanometric technologies," *IEEE International Test Conference (ITC07)*, pp. 1–10, 21-26 Oct. 2007.

[9] D. Ernst, N. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, *et al.*, "Razor: a low-power pipeline based on circuit-level timing speculation," *Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 7–18, 2003.

[10] M. Zhang, S. Mitra, T. M. Mak, N. Seifert, N. J. Wang, Q. Shi, K. S. Kim, N. R. Shanbhag, and S. J. Patel, "Sequential element design with built-in soft error resilience," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 12, pp. 1368–1378, 2006.

[11] D. Palframan, K. N.S., and M. Lipasti, "Time redundant parity for low-cost transient error detection," in *Design, Automation and Test in Europe (DATE11)*, 2011, pp. 52–57.

[12] R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, and R. Reis, "Analyzing area and performance penalty of protecting different digital modules with Hamming code and triple modular redundancy," in *Symposium on Integrated Circuits and Systems Design*, 2002, pp. 95–100.

[13] R. Naseer and J. Draper, "The DF-dice storage element for immunity to soft errors," *Proceedings of the 48th IEEE International Midwest Symposium on Circuits and Systems*, 2005.

[14] S. Hellebrand, H.-J. Wunderlich, A. A. Ivaniuk, Y. V. Klimets, and V. N. Yarmolik, "Efficient online and offline testing of embedded drams," *IEEE Trans. Computers*, vol. 51, no. 7, pp. 801–809, 2002.

[15] S. Boutobza, M. Nicolaidis, K. M. Lamara, and A. Costa, "A transparent based programmable memory bist," in *11th European Test Symposium (ETS2006)*, 2006, pp. 89–96.

[16] M. Imhof, H. Wunderlich, and C. Zoellin, "Integrating scan design and soft error correction in low-power applications," in *IEEE International On-Line Testing Symposium (IOLTS08)*. IEEE Computer Society, 2008, pp. 59–64.

[17] "Predictive Technology Model (PTM)." [Online]. Available: http://www.eas.asu.edu/˜ptm

[18] "Nangate Open Cell Library v1.3 v2009/07." [Online]. Available: http://openeda.si2.org/projects/nangatelib

[19] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, *et al.*, "FreePDK: An Open-Source Variation-Aware Design Kit," in *IEEE International Conference on Microelectronic Systems Education*. IEEE Computer Society Washington, DC, USA, 2007, pp. 173–174.