Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Multi-Error Mitigation Techniques on SoC FPGAs for Improved Fault Tolerance

Διπλωματικη Εργασια

του

Νικόλαου Μαστοράκη

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων
Αθήνα, Ιούνιος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Multi-Error Mitigation Techniques on SoC FPGAs for Improved Fault Tolerance

Διπλωματικη Εργασια

του

Νικόλαος Μαστοράκης

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Ιουνίου 2022.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Διονύσιος Πνευματικάτος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2022

(Υπογραφή)

.....

Νικολαος Μαστορακης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή μου, κ. Δημήτριο Σούντρη, για την εμπιστοσύνη που μου έδειξε και την ευκαιρία να εκπονήσω αυτή τη διπλωματική εργασία μέσω του εργαστηρίου του MicroLab. Ένα μεγάλο ευχαριστώ στον υποψήφιο διδάκτρορα κ. Βασίλη Λέων και στον μεταδιδακτορικό ερευνητή κ. Γεώργιο Λεντάρη, για την ακούραστη καθοδήγηση και την πολύτιμη βοήθεια που μου προσέφεραν. Θα ήθελα ακόμα να ευχαριστήσω τους φίλους μου, οι οποίοι ήταν συμπαραστάτες στις εύκολες και δύσκολες στιγμές όλα αυτά τα χρόνια. Τέλος θέλω να εκφράσω την αμέριστη ευγνωμοσύνη στην οικογένεια μου, στους γονείς μου Νικόλαο και Αικατερινή καθώς και στα αδέρφια μου Δημήτρη και Ανθούλα για την εμπιστοσύνη και την υποστηρίξη που μου έδωσαν όλα αυτά τα χρόνια.
Την Διλωματική εργασία την αφιερώνω στον πρόσφατα εκπλιπόντα Παππού μου Νικόλαο Χριστινίδη για τις μοναδικές αναμνήσεις και τα μαθήματα ζωης που μου πρόσφερε.

Περίληψη

Τα FPGA είναι ολοκληρωμένα κυκλώματα που περιέχουν έναν τεράστιο αριθμό προγραμματιζόμενου υλικου που μπορεί να διαμορφωθεί για να υλοποιεί τις επιθυμητές απαιτητιτκέσ υπολογιστικές λειτουργίες. Ένα από τα πιο σημαντικά χαρακτηριστικά ορισμένων τύπων FPGA, όπως FPGA που βασίζονται σε φλαση ή SRAM, είναι η δυνατότητα επαναδιαμόρφωσής τους. Αυτή η ξεχωριστή ικανότητα τα καθιστά κατάλληλα για την επίτευξη προσαρμόσιμων σχεδίων, και ταυτόχρονα ικανά να προσαρμόζονται εκ νέου στις αλλαγές ανάλογα με την εφαρμογή. Τα FPGA, ειδικά τα βασισμένα σε SRAM, κερδίζουν σσταδιακά δυναμική σε διάφορους τομείς, όπως η αυτοκινητοβιομηχανία, η ρομποτική και το διάστημα. Ωστόσο, σε περιβάλλοντα όπως αυτά του διαστήματος, τα FPGA είναι ευαίσθητα σε ιονίζουσα ακτινοβολία, η οποία μπορεί να προκαλέσει ανατροπές ενός συμβάντος (SEUs) στη μνήμη στατικής και δυναμικής διαμόρφωσής τους.Αυτές οι ανατροπές μπορεί να προκαλέσουν αποκλίσεις στα σχέδια που έχει προγραμματίσει ο χρήστης, επηρεάζοντας την κανονική του συμπεριφορά. Ως εκ τούτου, η σύγχρονη βιομηχανία θα πρέπει να λάβει σοβαρά υπόψη τα SEU, καθώς οι κρίσιμες αποστολές δεν μπορούν να αντέξουντη διαφθορά σημαντικών εφαρμογών επεξεργασίας δεδομένων. Λαμβάνοντας υπόψη ότι υπάρχει μια τάση χρήσης ντί για radiation-hardened (για λόγους απόδοσης και ευελιξίας), διεξάγεται έρευνα για τη βελτίωση της αξιοπιστίας τους και την παροχή λύσεων για την ανοχή σφαλμάτων.

Σε αυτό το πλαίσιο, η παρούσα Διπλωματική παρουσιάζει διάφορες αρχιτεκτονικές ανοχής σφαλμάτων, με στόχο την προστασία των FPGA SoC Zynq από τα SEU. Αυτές οι αρχιτεκτονικές προσπαθούν να ξεπεράσουν την επίδραση των SEU, χρησιμοποιώντας τις πιο αποτελεσματικές τεχνικές μετριασμού, όπως το Internal Configuration Scrubbing (μέσω του SEM IP του Xilinx), το Triple Modular redundancy (TMR), το Partial Reconfiguration (PR) και την επανεκκίνηση μέσω ενός Watchdog Timer. Για τον σκοπό της επαλήθευσης της ορθότητας των προτεινόμενων αρχιτεκτονικών, χρησιμοποιήσαμε τον ελεγκτή IP SEM και εφαρμόσαμε μια στρατηγική εμφυτεύσης, η οποία στοχεύει ευαίσθητα bits στη μνήμη SRAM διαμόρφωσης των εμπορικών πλακετών Zynq-7000 SoC της Xilinx.

Τα αποτελέσματα του πειράματος της εμφυτεύσης έχουν αξιολογηθεί με βάση την αξιοπιστία, τη διαθεσιμότητα, το ποσοστό σφάλματος και το μέσο σχετικό σφάλμα. Σύμφωνα με τα πειράματά μας στην πλακέτα Zybo, κάθε προτεινόμενη παραλλαγή των τεχνικών μετριασμού για SEU βελτιώνει την ευαισθησία του συστήματος ανοχής σε σφάλματα. Ταυτόχρονα, παρατηρούμε ότι η ευαισθησία βελτιώνεται, καθώς εμπλέκονται περισσότερες τεχνικές μετριασμού, δηλαδή στις υβριδικές μας αρχιτεκτονικές ανοχής σε σφάλματα. Όταν συγκρίνουμε

μεμονωμένα τις αρχιτεκτονικές, συμπεραίνουμε ότι το Ιντερναλ ὃνφιγυρατιον Σςρυββινγ ξεπερνά σημαντικά τις άλλες τεχνικές μετριασμού. Η πιο στιβαρή λύση είναι ο συνδυασμός όλων των τεχνικών μετριασμού, που σε σύγκριση με τον μη περιορισμένο σχεδιασμό, παρέχει 72% καλύτερη λειτουργικότητα στα αποτελέσματα της εφαρμογής που υλοποιήσαμε, 72% λιγότερο χρόνο διακοπής λειτουργίας και 98% δυνατότητα διόρθωσης στην εμφάνιση SEU, ενώ χρησιμοποιεί λιγότερο από 4K LUT, 5K FF, 6 RAMB στη διαμόρφωση μνήμης SRAM.

Λέξεις Κλειδιά

SRAM FPGA, Zynq, SEM, Scrubbing, TMR, SEU, ECC, CRC, Partial Reconfiguration, Error Injection, Fault Tolerance, Reliability

Abstract

FPGAs are integrated circuits that contain a huge number of programmable fabric that can be configured to implement desired compute-intensive functions. One of the most important features of certain FPGA types, such as flash- or SRAM-based, is their reconfiguration capability. This distinct capability makes them suitable for achieving customizable designs, and at the same time capable of re-adapting to changes depending on the application. FPGAs, especially the SRAM-based, are gaining momentum in various domains, such as automotive, robotics, space and avionics. Nevertheless, in environments such as those of space, FPGAs are susceptible to ionizing radiation, which can cause Single Event Upsets (SEUs) in their Static and Dynamic configuration memory. These upsets may cause deviations to the user programmed designs, affecting its normal behaviour. Therefore modern industry should take into serious consideration SEUs, as critical missions can not afford the corruption of important data processing applications. Considering that there is a trend of using Commercial Off-The-Shelf (COTS) FPGAs rather than radiation-hardened ones (for performance and flexibility reasons), research is conducted on improving their reliability and providing solutions for fault tolerance.

In this context, this thesis presents various fault tolerance architectures, aiming to protect the Zynq SoC FPGAs from SEUs. These architectures attempt to overcome the effect of SEUs, making use of the most effective mitigation techniques, such as Internal Configuration Scrubbing (via Xilinx's SEM IP), Triple Modular redundancy (TMR), Partial Reconfiguration (PR) and reboot through a Watchdog Timer. For the purpose of verifying the robustness of the proposed architectures, we utilized SEM IP controller and implemented an injection strategy, which targets sensitive bits in the configuration SRAM memory of the Xilinx's Zynq-7000 SoC commercial boards.

The results of the injection setup have been evaluated based on reliability, availability, error rate and mean relative error. According to our experiments on the Zybo board, each proposed variation of the SEU mitigation techniques improves the sensitivity of the fault-tolerant system. At the same time, we observe that sensitivity improves, as more mitigation techniques are involved, i.e., in our hybrid fault-tolerant architectures. When comparing individually the architectures, we conclude that Internal Configuration Scrubbing significantly outperforms the other mitigation techniques. The most robust solution is the combination of all the mitigation techniques, which compared to the unmitigated design, provides 72% more correct functionality, 72% less downtime and 98% correction

capability in the occurrence of SEUs, while using less than 4K LUTs, 5K FFs, 6 RAMBs in the configuration SRAM-memory.

Keywords

SRAM FPGA, Zynq, SEM, Scrubbing, TMR, SEU, ECC, CRC, Partial Reconfiguration, Error Injection, Fault Tolerance, Reliability

Contents

List of Figures

List of Tables

Εκτεταμένη Περίληψη

Εισαγωγή

Η συνεχής καινοτομία στην τεχνολογία για την ανάπτυξη και την κατασκευή FPGA επέτρεψε την αύξηση του επιπέδου ολοκλήρωσης. Αυτή η υψηλή ενσωμάτωση επιτρέπει λειτουργίες, όπως υψηλή χωρητικότητα, χαμηλή κατανάλωση ενέργειας και ταχύτερη λειτουργία. Ωστόσο, αυτή η ενοποίηση έχει ορισμένα μειονεκτήματα. Το ένα είναι ότι η ακτινοβολία παράγει περισσότερα σφάλματα. Ως εκ τούτου, αυτή η ευαισθησία τώρα είναι μια πραγματική ανησυχία ακόμη και από εφαρμογές σε επίπεδο εδάφους. Τα διαφορετικά αποτελέσματα αυτών των σφαλμάτων ονομάζονται Επιδράσεις Μεμονωμένων Συμβάντων (SEEs). Όταν εργάζεστε με FPGA που βασίζεται σεSRAM, το πιο επικίνδυνο αποτέλεσμα αυτής της ομάδας είναι οι ανατροπές ενός συμβάντος (SEU). Τα SEU είναι μη μόνιμα μαλακά σφάλματα που παράγονται από σωματίδια άλφα ή όταν τα πρωτόνια και οι κοσμικές ακτίνες από το διάστημα αλληλεπιδρούν με την ατμόσφαιρα και δημιουργούν υποατομικά σωματίδια που συγκρούονται με άτομα πυριτίου. Οι SEU μπορούν να επηρεάσουν ταbitαναστροφής των FPGA και στις δύο μνήμες χρήστη ή στη μνήμη διαμόρφωσης. Οι SEU που βρίσκονται στη μνήμη διαμόρφωσης είναι η πιο κρίσιμη περίπτωση επειδή μπορούν να αλλάξουν τον υλοποιημένο σχεδιασμό αλλάζοντας τη διαμόρφωση των κρίσιμων στοιχείων ή τις διασυνδέσεις τους. Το αντίκτυπο των SEU στις διαμορφώσεις βασίζεται στην εφαρμογή στην οποία υλοποιείται η συσκευή FPGA, συμπεριλαμβανομένου του αριθμού τωνbitδιαμόρφωσης.

Ως αποτέλεσμα μιας αναστάτωσης, ολόκληρο το σύστημα ή ένα κρίσιμο τμήμα του μπορεί να μην είναι διαθέσιμο ή να υπάρχει δυσλειτουργία. Για το λόγο αυτό, σε ορισμένες εφαρμογές, ειδικά σε τομείς που σχετίζονται με την ανθρώπινη ασφάλεια ή πολύ πολύτιμες τεχνολογίες, όπως σιδηροδρομικές, αεροηλεκτρονικές ή διαστημικές εφαρμογές ένα μόνο σφάλμα μπορεί να είναι καταστροφικό. Σε αυτές τις περιπτώσεις, απαιτούνται υψηλά επίπεδα αξιοπιστίας, κάτι που απαιτεί την εφαρμογή τεχνικών ανοχής σφαλμάτων για τη σκλήρυνση των σχεδίων. Το διάστημα είναι ένα από τα περιβάλλοντα όπου οι εποχούμενοι επεξεργαστές και οι επιταχυντές εκτίθενται άμεσα στην ακτινοβολία. Η διαστημική βιομηχανία χρησιμοποιεί διαστημικά FPGA σκληρυμένα με ακτινοβολία λόγω της αυξημένης αξιοπιστίας τους. Ωστόσο, στοχεύοντας στη βελτιωμένη απόδοση, υπάρχει μια τάση χρήσης εμπορικών συσκευών εκτός ραφιού(COTS) συμπεριλαμβανομένων των κλασικών FPGA καιSoC FPGA. Αυτά τα FPGA είναι μέρη αρχιτεκτονικών μικτής κρισιμότητας , που ενσωματώνουν συσκευές διαστημικής καιCOTS. Ωστόσο, οι συσκευέςCOTS είναι πιο ευαίσθητες στην ακτινοβολία και τις επιπτώσεις της, και ως εκ το-

ύτου, πρέπει να αυξήσουν την αξιοπιστία τους χρησιμοποιώντας τεχνικές μετριασμού ανοχής σφαλμάτων.

Σε αυτη την διπλωματική παρουσιάζονται διάφορες Αρχιτεκτονικές ανοχής βλαβών προκυμένου να δείξουμε οτι προτείνει τα εμπορικά zynq-7000 System-On-Chip FPGA που βασίζονται σεSRAM από τη Xilinx μπορούν πράγματι να χρησιμοποιηθούν σε αυτήν την περίπτωση. Για κατάλληλες εφαρμογές, τα FPGA που βασίζονται σεSRAM μπορούν να παρέχουν μια πολύ οικονομική εναλλακτική λύση. Τα FPGA που εξετάζονται σε αυτή τη διπλωματική περιορίζονται σε FPGA που βασίζονται σε zynq-7000 System-On-ChipSRAM. Επιπλέον, οι FPGA που εξετάζονται σε αυτήν την εργασία είναι αποκλειστικά XilinxFPGA.

Συμβολή διπλωματικής

Οι συνεισφορές αυτής της διπλωματικής εργασίας είναι οι ακόλουθες:

- Μια ολοκληρωμένη αναφορά τεχνικών μετριασμού, όπως Εσωτερικός καθαρισμός, Τριπλε Μοδυλαρ Ρεδυνδανςψ και μερική επαναδιαμόρφωση και το υλικό που είναι απαραίτητο για την εφαρμογή αυτών των τεχνικών μετριασμού στα FPGA της σειράς 7 τηςZynq.

- Εφαρμογή επτά αρχιτεκτονικών ανοχής σφαλμάτων, με στόχο την προστασία των FPGASoCZynq από τις SEU. Τρεις από αυτές ενσωματώνουν χωριστά παραδοσιακές τεχνικές μετριασμού (TMP, Ιντερναλ Σχρυββερ και Partial Reconfiguration) ενώ οι άλλες τις συνδυάζουν εν μέρει. Το τελευταίο Αρχιτεκτυρε ενσωματώνει όλες τις προαναφερθείσες τεχνικές μαζί με το Watchdog Timer που υλοποιείται σε έναν ελεγκτή Arduino.

- Αξιολόγηση με βάση την αξιοπιστία, τη διαθεσιμότητα, το ποσοστό σφάλματος και το μέσο σχετικό σφάλμα των υλοποιήσεων των Φαυλτ Τολερανςε Αρχιτεκτυρες για την οικογένεια Zynq-7000SoC μέσω της δοκιμής εμφυτευσεις. Τα αποτελέσματα αξιολογούνται με βάση τις μετρήσεις αξιολόγησης της αξιοπιστίας, της διαθεσιμότητας, του ποσοστού λάθους και του μέσου σχετικού σφάλματος.

Σύμφωνα με τα πειράματά μας στην πλακέτα ZYBO, κάθε προτεινόμενη παραλλαγή των τεχνικών μετριασμού της SEU βελτιώνει την ευαισθησία του συστήματος ανοχής σε σφάλματα. Ταυτόχρονα, παρατηρούμε ότι η ευαισθησία βελτιώνεται, καθώς εμπλέκονται περισσότερες τεχνικές μετριασμού, δηλαδή στις υβριδικές μας αρχιτεκτονικές ανοχής σε σφάλματα. Όταν συγκρίνουμε μεμονωμένα τις αρχιτεκτονικές, συμπεραίνουμε ότι το Internal Configuration Scrubbing ξεπερνά σημαντικά τις άλλες τεχνικές μετριασμού. Η πιο στιβαρή λύση είναι ο συνδυασμός όλων των τεχνικών μετριασμού, που σε σύγκριση με τον μη περιορισμένο σχεδιασμό, παρέχει 72% πιο σωστή λειτουργικότητα, 72% λιγότερο χρόνο διακοπής λειτουργίας και 98% δυνατότητα διόρθωσης στην εμφάνιση SEU, ενώ χρησιμοποιεί λιγότερο από 4K LUT, 5K FF και 6 RAMB.

Οι Αρχιτεκτονικές Ανοχής Βλαβών που παρουσιάζονται σε αυτή τη διπλωματική είναι ιδανικές για διαστημικές εφαρμογές καθώς και για πειράματα φυσικής υψηλής ενέργειας, τα οποία πρέπει να λειτουργούν αξιόπιστα σε περιβάλλοντα με υψηλά ποσοστά ανατροπής. Αυτές

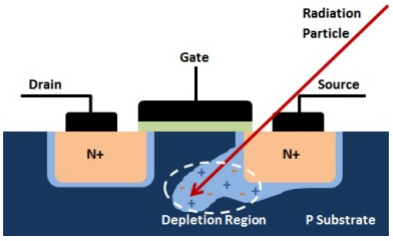
οι ισχυρές αρχιτεκτονικές μπορούν να προσαρμοστούν και να χρησιμοποιηθούν από χρήστες FPGA προκειμένου να καλύψουν τις ανάγκες των ψηφιακών τους σχεδίων.

Background

Η αναγνώριση της πραγματικής απειλής που αποτελεί η ακτινοβολία για τα FPGA σε περιβάλλοντα υψηλής ακτινοβολίας παρακινεί την ανάγκη για βελτιωμένες τεχνικές μετριασμού. Αρκετές βασικές τεχνικές μετριασμού ανατροπών επισημάνθηκαν σε αυτό το κεφάλαιο.

Ιονίζουσα ακτινοβολία

Η ιονίζουσα ακτινοβολία είναι κάθε τύπος σωματιδίου ή ηλεκτρομαγνητικού κύματος που μεταφέρει αρκετή ενέργεια για τον ιονισμό ή την αφαίρεση ηλεκτρονίων από ένα άτομο [2, 3, 4]. Για τρανζίστορ που αποτελούν τη μνήμη (όπως αυτά τηςSRAM), η ιονίζουσα ακτινοβολία μπορεί να διαταράξει την κατάσταση του τρανζίστορ προκαλώντας υπερτάσεις ρεύματος ή διακοπή της ροής ρευμάτων. Η ιονίζουσα ακτινοβολία προέρχεται από μια ποικιλία πηγών και έχει ένα ευρύ φάσμα επιδράσεων στα FPGA.



μ 1: The Effect of a Radiation Particle Strike on a Transistor.

Το φαινόμενο Σινγλε Εεντ Εφφεστ (SEEs)προκαλείται όταν τα σωματίδια ακτινοβολίας συγκρούονται με το πλέγμα ημιαγωγών των τρανζίστορ, ενέργεια μεταφέρεται στα ατομικά σωματίδια του τρανζίστορ. Αυτά τα χτυπήματα ακτινοβολίας μπορεί να προκαλέσουν κενές περιοχές στα τρανζίστορ αφήνοντας πίσω μια περίσσεια ηλεκτρονίων ή τρύπες στο πέρασμά τους [5]. Ανάλογα με τον προσανατολισμό των ελεύθερων ηλεκτρονίων ή των οπών που δημιουργούνται, ηλεκτρικό ρεύμα μπορεί να αρχίσει να ρέει ή να σταματήσει να ρέει, κάτι που με τη σειρά του θα μπορούσε να προκαλέσει το ψηφιακό κύκλωμα για να αλλάξει τη λογική του κατάσταση (βλ. Εικόνα 1). Όταν συμβαίνει ένα τέτοιο συμβάν σε ένα FPGA, είναι γνωστό ως Ανατροπή Μεμονωμένου Συμβάντος (SEU).

Όπως ξέρουμε η μνήμη διαμόρφωσης XilinxFPGA είναι οργανωμένη σε μονάδες που ονομάζονται frame που είναι οι μικρότερες διευθυνσιοδοτούμενες μονάδες σε ένα FPGA. Εάν μια SEU επηρεάζει μόνο ένα λογικό bit σε ένα frame, είναι γνωστό ως ανατροπή ενός ενβιτ (SBU). Τα SBU σε διαφορετικά πλαίσια ταξινομούνται το καθένα ως ξεχωριστά SBU για λόγους διόρθωσης. Όταν συμβαίνουν πολλαπλές ανατροπές στο ίδιο frame, το αποτέλεσμα είναι μια ανατροπή πολλαπλών μπιτ (MBU). Ένα μεμονωμένο συμβάν που προκαλεί την ανατροπή πολλώνbitστο ίδιο frame είναι ένα Multi-Cell Upset (MCU) [6]. Όπως σημειώθηκε στο [7], τα MBU είναι όλο και πιο πιθανά καθώς η τεχνολογία των τρανζίστορ συνεχίζεται να συρρικνώνεται. Τα ατομικά σωματίδια ίδιου μεγέθους βομβαρδίζουν όλο και μικρότερες γεωμετρίες συσκευών, προκαλώντας αυξανόμενος αριθμός αναστατωμένων γεγονότων.

(a) Routing and Logic Before Upset. **(b)** Routing and Logic After Upset.

Οι ανατροπές στα κελιά διαμόρφωσηςSRAM μπορούν πραγματικά να αλλάξουν τη λειτουργικότητα του κυκλώματος σχέδιο. Ένα παράδειγμα ψηφιακού κυκλώματος που υλοποιήθηκε σε ένα FPGA φαίνεται στο σχήμα 2α. Τα λογικά bits της μνήμης διαμόρφωσης εμφανίζονται καθορίζοντας τη δρομολόγηση και τη λειτουργία των πόρων του FPGA που χρησιμοποιούνται για την υλοποίηση μιας πύλης ΑΝΔ δύο εισόδων. Όταν συμβαίνει μία ανατροπή στη λογική δρομολόγησης, όπως φαίνεται στο σχήμα 2β, μια από τις εισόδους προς την πύλη είναι αποσυνδεδεμένη. Όταν συμβεί μια άλλη ανατροπή μέσα σε ένα από τα LUT, η πύλη AND αλλάζει σε πύλη XOR. Η ακριβής λειτουργία του αρχικού κυκλώματος έχει χαθεί.

Μεταβατικό μεμονωμένο συμβάν

Ένα μεταβατικό μονού συμβάντος (SET) είναι μια προσωρινή αιχμή τάσης ή ρεύματος σε έναν κόμβο ενός ολοκληρωμένο κύκλωμα, το οποίο επάγεται από ένα μόνο φορτισμένο σωματίδιο που ιονίζει έναν ημιαγωγό υλικό κατά τη διέλευση μέσω ή δίπλα σε μια ηλεκτρικά ευαίσθητη διασταύρωση. Αν το πλάτος και το πλάτος του παλμού είναι επαρκές, το σφάλμα μπορεί να διαδοθεί μέσω του κυκλώματος [8]. Μετατρέπεται σε Ανατροπή Μεμονωμένου Συμβάντος εάν κλειδωθεί σε ένα κελί αποθήκευσης κατά την άφιξη στην εισαγωγή δεδομένων κατά την άκρη του ρολογιού. Το SET είναι ένα μη καταστρεπτικό και ανακτήσιμο σφάλμα, το οποίο μπορεί να εξελιχθεί σε ανατροπή ενός συμβάντος ή λειτουργική διακοπή ενός συμβάντος.

Λειτουργική διακοπή μεμονωμένου συμβάντος

Μια λειτουργική διακοπή ενός συμβάντος (SEFI) είναι ένα μη καταστροφικό σφάλμα που έχει ως αποτέλεσμα την παρεμβολές κανονικής λειτουργίας μιας πολύπλοκης ψηφιακής συσκευής ή συστήματος.

Στην περίπτωση συσκευών FPGA, ένα SEFI συχνά σχετίζεται με μια ανατροπή σε έναbitελέγχου ή register [9]. Συνήθως σχετίζεται με αστοχία στα ενσωματωμένα κυκλώματα της μνήμης διαμόρφωση ή εκ νέου διαμόρφωση ή ενεργοποίηση επαναφοράς[10]. Για να επαναφέρετε τη σωστή λειτουργία μετά από ένα SEFI, απαιτείται πλήρης επαναδιαμόρφωση ή κύκλος ενεργοποίησης.

Γενικές τεχνικές μετριασμού ανατροπών FPGA

Πολλές βασικές τεχνικές μετριασμού σφαλμάτων έχουν αναπτυχθεί για τη βελτίωση του FPGA αξιοπιστία. Πολλές τρέχουσες λύσεις αξιοπιστίας FPGA συνδυάζουν αρκετά από αυτά τα μέτρα μετριασμού τεχνικές. Μια περίληψη των πιο κοινών τεχνικών δίνεται παρακάτω, ωστόσο, αυτή η περίληψη δεν προορίζεται να είναι εξαντλητική. Αυτές οι τεχνικές μετριασμού αναστατώσεων χρησιμοποιούνται σε πολλές συστήματα μετάδοσης δεδομένων υψηλής ταχύτητας και μνήμης σε όλο το φάσμα των ηλεκτρονικών. ο Οι περιλήψεις που παρουσιάζονται εδώ θα περιγράψουν τον τρόπο με τον οποίο εφαρμόζονται ορισμένες τεχνικές μετριασμού αναστατώσεων ειδικά σε σχέση με τα Xilinx FPGA, παρόλο που ορισμένες από αυτές δεν εφαρμόζονται στη διπλωματική μας. Μπορούν να διακριθούν οι ακόλουθες δύο κατηγορίες τεχνικών που βασίζονται στο σχεδιασμό.

- **Στατικές τεχνικές** βασίζονται στην έννοια της κάλυψης σφαλμάτων. Επιτυγχάνουν ανοχή σφαλμάτων μέσω παθητικού πλεονασμού και μηχανισμών σύγκρισης/ψηφοφορίας, χωρίς να απαιτείται καμία ενέργεια από την πλευρά του συστήματος.

- **Δυναμικές τεχνικές** επιτυγχάνουν ανοχή σφαλμάτων ανιχνεύοντας την ύπαρξη σφάλματα και την εκτέλεση ορισμένων ενεργειών για την αφαίρεση του ελαττωματικού υλικού (ή βλάβης) από το σύστημα. Αυτές οι τεχνικές βασίζονται στη χρήση ταυτόχρονης ανίχνευσης σφαλμάτων, θέσης σφάλματος και σφάλματος ανάκτηση.

Radiation Hardening

Η σκλήρυνση με ακτινοβολία συνήθως επιτυγχάνεται χρησιμοποιώντας μία από τις δύο μεθόδους. Radiation hardened by design (RHBD) είναι μια διαδικασία πλεονασμού όπου χρησιμοποιούνται περισσότερα τρανζίστορ για την κατασκευή ένα κελί SRAM. Τα τρανζίστορ τοποθετούνται σε ειδική διάταξη έτσι ώστε η πιθανότητα ότι Το ίδιο ιόν συγκρούεται με πολλαπλά τρανζίστορ της ίδιας κυψέλης SRAM είναι πολύ χαμηλό, καθιστώντας το είναι απίθανο να προκαλέσει αναστάτωση [11].

Η σκλήρυνση με ακτινοβολία μέσω διεργασίας RHBP) είναι όπου κατασκευάζεται το FPGA με κάποιο τρόπο ότι τα τρανζίστορ προστατεύονται από ιονίζουσα ακτινοβολία σε επίπεδο πυριτίου [12]. Περιφραγμένο Η σκλήρυνση αντιστάσεων είναι ένα παράδειγμα RHBP)

που χρησιμοποιεί μια μεταβλητή αντίσταση για να αυξήσει το οριακή τάση που απαιτείται για την αλλαγή της κατάστασης της κυψέλης μνήμης. Μια άλλη μέθοδος είναι να διατηρήστε την απόδοση του κυκλώματος μειώνοντας την οριακή τάση όταν η εγγραφή ενεργοποιείται στο Το κελί μνήμης είναι υψηλό [11].

Η χρήση εξαρτημάτων που έχουν σκληρυνθεί με ακτινοβολία είναι μια ασφαλής επιλογή, αλλά τα εξαρτήματα είναι συνήθως πολύ περισσότερα ακριβά από τα εμπορικά ανταλλακτικά εκτός ραφιού (COTS). Εάν απαιτείται σχεδόν τέλεια αξιοπιστία, τότε αυτή η τεχνική είναι ίσως η καλύτερη επιλογή. Ωστόσο, εάν οι απαιτήσεις αξιοπιστίας είναι πιο ευέλικτες, τότε οι άλλοι μηχανισμοί που παρουσιάζονται σε αυτή την εργασία είναι πολύ πιο αποδοτικές εναλλακτικές λύσεις.

Error Correction Code

Ένας κωδικός διόρθωσης σφάλματος (SECDED) είναι ένας μηχανισμός κωδικοποίησης πλεονασμού που είναι χρήσιμος για τη διόρθωση SBU. Τα ECC επιτυγχάνουν Διόρθωση ενός λάθους και Ανίχνευση διπλού σφαλμάτων (SECDED) σε βάση ανά frame. Κάθε frame διαμόρφωσης σε Xilinx FPGA περιέχει μια λέξη ECC για την παροχή βασικής διόρθωσης SBU. Εντοπίζονται ανατροπές διπλούbitόταν χρησιμοποιείτε ένα ECC αλλά δεν μπορούν να διορθωθούν.

Ένα ECC υλοποιείται σε υλικό με ένα αποκλειστικό κύκλωμα που αποτελείται από ΞΟΡ αλυσίδες που συνδυάζουν συγκεκριμένα bit σε ένα frame για τον υπολογισμό ενός αθροίσματος ελέγχου, γνωστό ως σύνδρομο[13].Κάθε θέση ενβιτ στο frame υπολογίζεται κάπου στο κύκλωμα [13]

Το ECC από μόνο του δεν κάνει ουσιαστικά κανένα μετριασμό της αναστάτωσης. Ένα κύκλωμα ελεγκτή απαιτείται για την αποκωδικοποίηση του ECC και την εκτέλεση των δεδομένων πλαισίου μέσω του κυκλώματος ECC. Το αποτέλεσμα υποδεικνύει πού βρίσκεται στο frame ένα SBU (εάν υπάρχει) ή εάν έχει προκύψει ανατροπή διπλού bit. Ο ελεγκτής μπορεί στη συνέχεια να χρησιμοποιήσει αυτές τις πληροφορίες για να διορθώσει την αναστάτωση. Ένα σύνδρομο όλων των μηδενικών δείχνει ότι το ECC ήταν ικανοποιημένο. Ένα «ικανοποιημένο» ECC συνήθως σημαίνει ότι δεν υπάρχουν Σφάλματα. Ωστόσο, είναι πολύ πιθανό αρκετά κομμάτια στα σωστά σημεία να ανατρέπονται στο frame ώστε να μην ανιχνεύεται από το ECC. Η λέξη ECC μέσα στο frame, γνωστή ως το Το ¨χρυσό ECC¨, δημιουργείται για να είναι η ακριβής τιμή που γεφυρώνει το χάσμα μεταξύ τωνbitστο το frame και ένα μηδενικό (ικανοποιημένο) σύνδρομο.

Τα ECC στα FPGA Xilinx εντοπίζουν όλες τις μονές ανατροπές [1]. Εντοπίζουν επίσης και τις δύο ανατροπές. Οι ζυγές ανατροπές μεγαλύτερες από δύο μπορεί να περάσουν απαρατήρητες επειδή επαρκούν Ο αριθμός των bits ανατρέπεται στα σωστά σημεία, έτσι ώστε να μην πιαστούν από το ECC, αλλά μια τέτοια περίπτωση είναι αρκετά σπάνια.

Cyclic Redundancy Check

Ο κυκλικός έλεγχος πλεονασμού ("P") είναι ένας άλλος μηχανισμός κωδικοποίησης που είναι χρήσιμο για τον εντοπισμό αναστατώσεων. Το κύριο πλεονέκτημά του σε σύγκριση με άλλους μηχανισμούς κωδικοποίησης όπως Τα ECC είναι η ικανότητά του να ανιχνεύει ανατροπές μέσα σε εκατομμύρια bit χρησιμοποιώντας μία μόνο τιμή CRC. Η αλλαγή έστω και ενός κομματιού της διαμόρφωσης θα έχει ως αποτέλεσμα μια διαφορετική τιμή CRC. Το κύριο μειονέκτημά του, ωστόσο, είναι ότι δεν μπορεί να εντοπίσει πού βρίσκονται συγκεκριμένα σφάλματα. Έτσι, χρησιμοποιείται κυρίως ως μηχανισμός ανίχνευσης σφαλμάτων. Ένα κύκλωμα που χρησιμοποιείται για τον υπολογισμό ενός CRC βασίζεται στο υπόλοιπο ενός προσεκτικά σχεδιασμένου πολυωνυμικό κύκλωμα διαίρεσης. Τα CRC προστίθενται στη μνήμη διαμόρφωσης των FPGA προς παρέχουν ένα μέσο ελέγχου της ακεραιότητας των δεδομένων. Υπολογίζεται μία τιμή 32bitγια ολόκληρη τη μνήμη διαμόρφωσης. Κάθε πλαίσιο της διαμόρφωσης ενός FPGA συμβάλλει σε αυτόν τον υπολογισμό.

Τα CRC είναι ιδιαίτερα χρήσιμα για την ανίχνευση MBU σε ένα FPGA. Ένα τέτοιο χαρακτηριστικό είναι είναι ανεκτίμητο όταν άλλοι κωδικοί όπως οι ECC μπορούν να ανιχνεύσουν μόνο μονές και συγκεκριμένες ζυγές ανατροπές. 32-βιτ CRC σε συσκευές Ξιλινξ εντοπίζει οποιοδήποτε σφάλμα 31 bit σε ένα μόνο πλαίσιο με 100 τοις εκατό ακρίβεια. Πέρα από 31 βιτ, εξακολουθεί να είναι πολύ ασφαλές, αλλά υπάρχουν μερικές περιπτώσεις όπου το MBU θα παρέμενε απαρατήρητο. Με τόσο υψηλή κάλυψη σφαλμάτων, η πιθανότητα να μην εντοπιστεί μια ανατροπή διαμόρφωσης είναι πολύ μικρή. Τα CRC παίζουν σημαντικό ρόλο σε πολλές αρχιτεκτονικές καθαρισμού, συμπεριλαμβανομένης της εσωτερικής IPτης SEM αρχιτεκτονική.

Triple Modular Redundancy

Το Triple Modular Redundancy (TMR) είναι μια τεχνική μετριασμού αναστατώσεων που βελτιώνει την αξιοπιστία του FPGA. Σε αντίθεση με τις τεχνικές που παρουσιάστηκαν προηγουμένως που κατασκευάστηκαν στο υλικό του FPGA, το TMR είναι μια τεχνική στην οποία οι σχεδιαστές πρέπει να ενσωματώσουν το κύκλωμά τους. Σε ένα σχέδιο με TMR, το κύκλωμα σχεδίασης τριπλασιάζεται σε τρία πανομοιότυπα αντίγραφα και συνήθως τοποθετούνται σε τρεις διαφορετικές τοποθεσίες στο FPGA. Οι έξοδοι (και μερικές φορές οι είσοδοι) όλων αυτών των κυκλωμάτων συνδέονται με ένα κοινό κύκλωμα ψηφοφόρου το οποίο συγκρίνει τις επιμέρους εξόδους του κυκλώματος και επιλέγει την πλειοψηφική τιμή. Για να αποφύγετε ένα μοναδικό σημείο αποτυχίας στο κύκλωμα ψηφοφόρων, οι ψηφοφόροι συχνά τριπλασιάζονται επίσης. Το TMR παρατείνει την αξιοπιστία του κυκλώματος που δίνει στον μηχανισμό επισκευής χρόνο να διορθώσει το πρώτο σφάλμα πριν μια άλλη μονάδα είναι αναστατωμένη και το συνολικό κύκλωμα αποτυγχάνει.

Memory Scrubbing

Σε αντίθεση με τη μνήμη στατικής διαμόρφωσης ενός FPGA, οι δυναμικές μνήμες αλλάζουν συνεχώς με την ανάγνωση και την εγγραφή από μια εφαρμογή. Οι δυναμικές αναμνήσεις προκαλούν επίσης αναστατώσεις πρέπει να μετριαστούν, αλλά οι μέθοδοι που χρησι-

μοποιούνται είναι κάπως διαφορετικές από αυτές που χρησιμοποιούνται για τη στατική αναμνήσεις.

Ο Memory Scrubbing μπορεί να πραγματοποιηθεί σε μνήμες όπως οι μπλοκ PAM (BPAM) ενός FPGA. θα χρησιμοποιούν συχνά παραλλαγές ECC ή/και TMR [14]. Για παράδειγμα, κάθε μπλοκ μνήμης μπορεί να περιέχει μια λέξη ECC και μια ειδική τράπεζα μνήμης χρησιμοποιείται για την αποθήκευση των χρυσών τιμών ECC. Κάθε φορά που το μπλοκ της μνήμης τροποποιείται, ένα νέο Το ECC υπολογίζεται και αποθηκεύεται στην τράπεζα μνήμης. Ένας άλλος μηχανισμός μπορεί στη συνέχεια περιοδικά συγκρίνετε τις τιμές ECC σε κάθε μνήμη με την αντίστοιχη τιμή στην τράπεζα και διορθώστε SBU εάν προκύψουν αναντιστοιχίες.

Το TMR θα μπορούσε να χρησιμοποιηθεί με τριπλασιασμό της μνήμης και ανάγνωση από κάθε μνήμη περιοδικά για να διαπιστωθεί εάν υπάρχει διαφορά στις μνήμες. Το TMR μπορεί στη συνέχεια να χρησιμοποιήσει την τιμή που συμφωνήθηκε με την πλειοψηφία για να διορθωθεί στη συνέχεια η ανατροπή στην τοποθεσία που διαφέρει [15].

Ο πραγματικός καθαρισμός της μνήμης εκτελείται είτε πιθανολογικά είτε ντετερμινιστικά [16]. Πιθανοτικός καθαρισμός είναι όταν τα μπλοκ της μνήμης καθαρίζονται μόνο κάθε φορά που γίνεται πραγματική πρόσβαση σε αυτή τη μνήμη (μέσω ανάγνωσης ή εγγραφής). Ωστόσο, εάν μερίδες από Η μνήμη δεν είναι ποτέ προσβάσιμη, τότε θα μπορούσαν να συσσωρευτούν ανατροπές σε αυτές τις αναμνήσεις. Ντετερμινιστικά το Memory Scrubbing είναι μια συνεχής διαδοχική σάρωση που ελέγχει πάντα όλες τις μνήμες. Το ντετερμινιστικό Memory Scrubbing είναι φυσικά πιο ενεργοβόρο και επιβαρύνει την απόδοση, αλλά επιτυγχάνει καλύτερη αξιοπιστία σε σύγκριση με πιθανοτικό καθαρισμό.

Configuration Scrubbing

Η μνήμη διαμόρφωσης αναμένεται να παραμείνει στατική καθ΄ όλη τη διάρκεια ζωής του ψηφιακού σχεδιασμού. Ο καθαρισμός διαμόρφωσης περιλαμβάνει περιοδικό έλεγχο αυτής της μνήμης, ανίχνευσης ανατροπές και, στη συνέχεια, χρησιμοποιήστε δυναμική μερική αναδιαμόρφωση για να αντικαταστήσετε τις ανατροπές [17]. Η αρχική διαμόρφωση πρέπει πρώτα να έχει ολοκληρωθεί προτού να συμβεί οποιοδήποτε τρίψιμο. Διαμόρφωση Configuration Scrubbing πραγματοποιείται συνήθως σε περιοδικά διαστήματα ενώ το FPGA βρίσκεται σε λειτουργία (επομένως ΄δυναμική' αναδιαμόρφωση). Αυτή η μερική αναδιαμόρφωση δεν προορίζεται να διακόψει την κανονική λειτουργία του FPGA.

Ο καθαρισμός διαμόρφωσης έχει δύο κύριες αρμοδιότητες: ανίχνευση σφαλμάτων και διόρθωση. Ένα σκρυββερ είναι ένας ελεγκτής που καθορίζει πότε και πού θα πραγματοποιηθεί μια μερική αναδιαμόρφωση για διόρθωση. Οι Configuration Scrubbers ελέγχουν επίσης περιοδικά τη μνήμη για να ανιχνεύσουν ενοχλήσεις. Οι Configuration Scrubbers έρχονται με μια ποικιλία αντισταθμίσεων, όπως μέγεθος, κατανάλωση ενέργειας, απόδοση, στιβαρότητα, κ.λπ.

Watchdog Timer

Ο χρονοδιακόπτης παρακολούθησης (WDT) είναι ένας χρονοδιακόπτης υλικού ή λογισμικού που συνήθως μπορεί να προκαλέσει επαναφορά του συστήματος εάν δεν ¨πατηθεί' ή ¨τροφοδοτηθεί' περιοδικά. Γενικά, στα ενσωματωμένα συστήματα ο κύριος προγραμματισμός είναι σε κάποιο τύπο βρόχου. Κάθε επανάληψη αυτού του βρόχου ενεργοποιείται το WDT για να διασφαλιστεί ότι το σύστημα δεν επαναφέρεται Ένα WDT παρέχει τρόπο για να επαναφέρετε γρήγορα το σύστημα για να μετριαστούν προβλήματα που μπορεί να προκαλέσουν το σύστημα να σταματήσει. Είναι επίσης σύνηθες ότι ενα αντίγραφο ασφαλείας αποθηκεύεται σε ένα μια εξωτερική φλαση. Εάν ένα WDT ενεργοποιηθεί πολλές φορές διαδοχικά, το αντίγραφο θα φορτωθεί στη θέση της τρέχουσας εικόνας. Αυτό ελπίζουμε ότι επαναφέρει το σύστημα σε μια γνωστή κατάσταση όσο το δυνατόν γρηγορότερα.

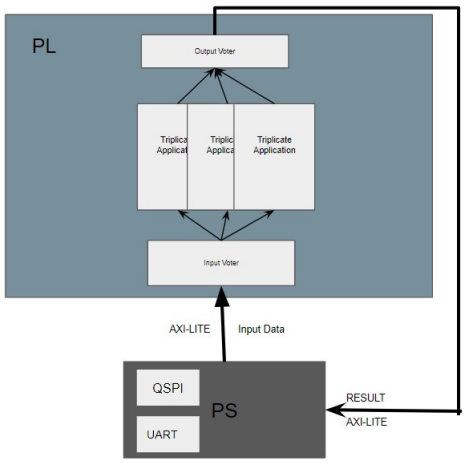
Προτεινόμενες αρχιτεκτονικές

Αυτό το κεφάλαιο παρουσιάζει μεθόδους και τεχνικές που μπορούν να χρησιμοποιηθούν για τον μετριασμό των ανατροπών συμβάντων, και κατά συνέπεια επιτρέπουν τη χρήση FPGA βάσεωνSRAM σε συστήματα που λειτουργούν σε περιβάλλοντα ακτινοβολίας. Σε αυτό το Κεφάλαιο επίσης οι μέθοδοι και οι τεχνικές που αναλύονται ενσωματώνονται στον σχεδιασμό μας για την υλοποίηση διαφόρων Αρχιτεκτονικών Ανοχής Βλαβών.

Προτεινόμενη αρχιτεκτονική βασισμένη σε τεχνικές μετριασμού TMR

Σε αυτήν την ενότητα η προτεινόμενη αρχιτεκτονική για την επίλυση τυχόν SEU που εμφανίζονται, είναι να αναπαράγουμε περιττα σχέδια μιας ολόκληρης ενότητας και να ψηφίζουμε τις τελικές εξόδους των μονάδων (Block-TMR). Το TMR εφαρμόζεται σε οποιοδήποτε σχέδιο μπορεί να τριπλασιαστεί. Επομένως στο σχεδιασμό μας λόγω περιορισμού σε πόρους δεν μπορούμε τριπλασιάσουμε τους επεξεργαστές ARM (PS). Στην περίπτωσή μας μια ενότητα αντιπροσωπεύει η εφαρμογή φίλτρου (FIR) που θα πραγματοποιηθεί στο PL. Αυτό είναι ένα πολύ αποτελεσματικό μέσο μετριασμού σε SEU που είναι εύκολο εφαρμοστεί και μπορεί να εκτελεστεί εξ ολοκλήρου μέσα σε μία μόνο συσκευή εφόσον η μονάδα δεν χρησιμοποιεί περισσότερο από το ένα τρίτο της συνολικής συσκευής. Για να ενσωματώσουμε το TMR στο σχεδιασμό μας έχουμε εφαρμόσει δύο βασικά συστατικά. Η εισροή και η εκροή πλειοψηφία Ψηφοφόροι. Όπως φαίνεται 3 το πρώτο τμήμα PS θα στείλει 3 σετ δεδομένων στο τμήμα PL.

Το buffer εισόδου λαμβάνει τα δεδομένα εισόδου και το Input IP Voter διανέμουν διαφορετικά δεδομένα σε κάθε FIR. Αντίθετα, ένα σύνολο των ίδιων δεδομένων χωρίζεται σε τρία διαφορετικά FIR για υπολογισμούς και κάθε κύκλος θα υπολογίστε 3 αποτελέσματα. Ο output voter ip θα πραγματοποιήσει μια σύγκριση μεταξύ των τριών αποτελεσμάτων εξόδου και θα στείλει στο PS την ελαττωματική μονάδα μαζί με τα τρία αποτελέσματα εξόδου. Ο ψηφοφόρος που βρίσκεται στην πλευρά PS θα συγκρίνει ξανά αυτά τα τρία σύνολα δεδομένων



μ 3: Architecture of TMR.

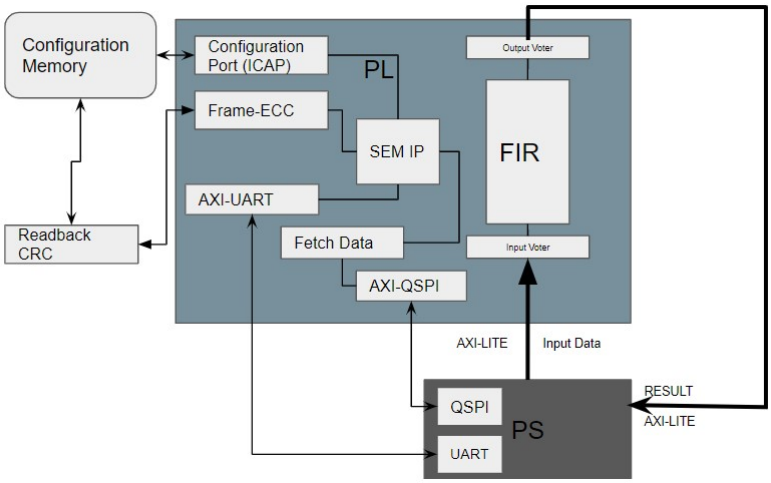
και θα επιλέξει το αποτέλεσμα υπολογίστηκε πανομοιότυπα τουλάχιστον δύο φορές. Οποιαδήποτε διαφορά μεταξύ του ψηφοφόρου PL και PS θα αποτελεί ένδειξη ότι ο ψηφοφόρος εξόδου έχει παραβιαστεί. Παράλληλα, εάν ένα SEU συνέβη στη διαδικασία υπολογισμού, θα καταγράψει και θα καταγράψει επίσης το FIR που προκάλεσε το σφάλμα. Το PS θα στείλει τα νέα δεδομένα στο buffer δεδομένων και τα τρία FIR θα εκτελούν την ίδια λειτουργία σε κάθε κύκλο. Μετά την ολοκλήρωση των υπολογισμών ολόκληρης της εφαρμογής, η πλευρά PS θα ψηφίσει για την επιλογή του τελικού αποτελέσματος.

Προτεινόμενη αρχιτεκτονική βασισμένη στον SEM IP

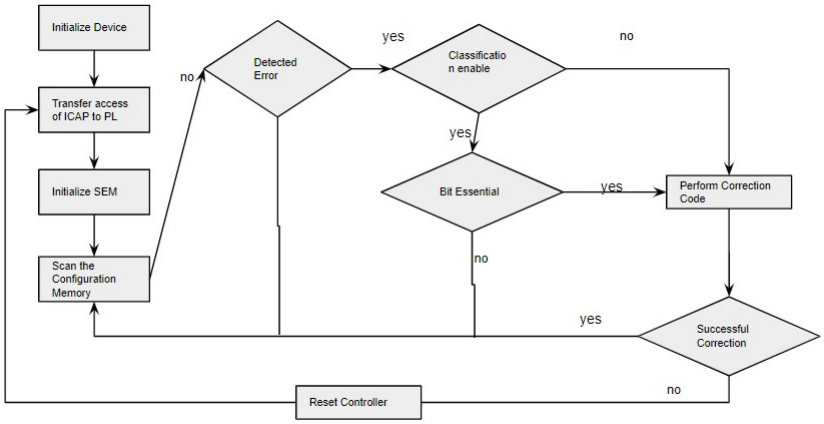
Η Xilinx δημιούργησε το SEM IP ως μηχανισμό μετριασμού αναστατώσεων για επίγειες εφαρμογές. Ο SEM IP χρησιμοποιεί το εσωτερικό υλικό Readback CRCγια να εκτελέστε ανίχνευση σφαλμάτων χρησιμοποιώντας τους ενσωματωμένους κωδικούς ECC και CRC [1]. Η SEM IP, Ωστόσο, εκτελεί τη δική του διόρθωση (σε αντίθεση με το να αφήνει το Readback CRC πραγματοποιήσει τη διόρθωση) μέσω του ICAP. Αυτή η διόρθωση περιλαμβάνει δυνατότητες διόρθωσης MBU από δεδομένα.

Το SEM IP ὸντρολλερ προσφέρει επίσης διαφορετικούς τρόπους διόρθωσης σφαλμάτων. Η λειτουργία επισκευής είναι ουσιαστικά το ισοδύναμο του Readback CRC που χειρίζεται όλα τα SBU διόρθωση, αλλά δεν μπορεί να διορθώσει τα MBU. Η λειτουργία βελτιωμένης επιδιόρθωσης προσθέτει μεμονωμένα frame ECC για υποστήριξη SBU και διόρθωση σφαλμάτων διπλών bit, αλλά δεν μπορεί να διορθώσει άλλα MBU. Η λειτουργία αντικατάστασης είναι ουσιαστικά η ίδια με τον blind Scrubber. Η λειτουργία αντικατάστασης μπορεί να διορθώσει όλα τα μεγαλύτερα MBU.

Η ανίχνευση για το SEM IP επιτυγχάνεται μέσω του Readback CRC. Η ταχύτητα



μ 4: Architecture of SEM IP as Internal Scrubber.

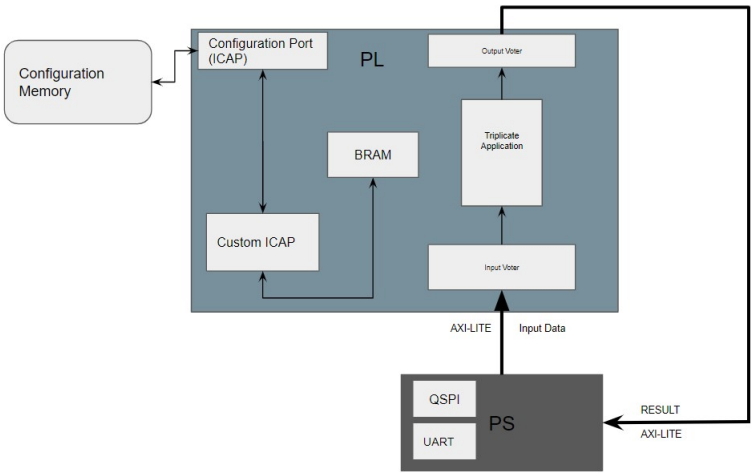


μ 5: Architecture flow of SEM IP as Internal Scrubber.

ανίχνευσης εξαρτάται από τον αριθμό των frames του FPGA και την συχνότητα ρολογιού του CRC Readback. Προκειμένου να ενσωματώσουμε με επιτυχία το SEM IP στο σχεδιασμό μας, τα παρακάτω interface πρέπει να υλοποιηθούν ή να εισαχθούν στην αρχιτεκτονική όπως φαίνεται στην Εικόνα 4. Όταν αυτά υλοποιηθούν, ο ελεγκτής SEM θα λειτουργεί όπως υποτίθεται ότι αναφέρεται στο [1].

Αφού παρουσιαστεί ένα σφάλμα, ο SEM θα ακολουθήσει την παρακάτω διαδικασία 5 για να το διορθώσει. Έτσι, ανάλογα με την προσαρμογή που έχει γίνει σε σχέση με τη λειτουργία διόρθωσης, ο SEM IP θα προσπαθήσει να διαμορφώσει εκ νέου τη ρύθμιση παραμέτρων σε περίπτωση εμφάνισης ουσιώδους σφάλματος μνήμης γράφοντας το σωστό Frame of Frames μέ ICAP.

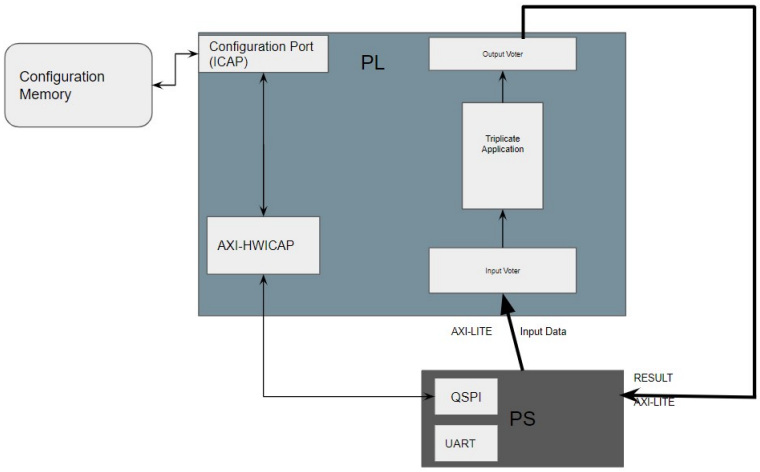
Προτεινόμενη Αρχιτεκτονική που βασίζεται σε Μερική Αναδιαμόρφωση



μ 6: Architecture of Dynamic Partial reconfiguration.

Σε αυτή τη διπλωματική έχουμε εφαρμόσει δύο αρχιτεκτονικές μερικής αναδιαμόρφωσης. Η πρώτη είναι ενος custom interface ICAP στην οποία τα partial bitstream αποθηκεύεται στην εσωτερική μνήμη BRAM του FPGA 6. Η διαφορά μεταξύ των δύο αρχιτεκτονικών είναι ότι στην περίπτωση του custom ICAP, το σύνολο διαδικασία ανάκτησης του μερικού bitstream που είναι αποθηκευμένο σε Block RAM δεν πραγματοποιείται από μια εφαρμογή λογισμικού, αλλά από μια μονάδα υλικού που έχει αναπτυχθεί από την αρχή στο RTL

Στη δεύτερη αρχιτεκτονική, τα partial bitstream παραδίδονται στο interface ICAP της συσκευής FPGA, μέσω του πυρήνα IP του AXI HWICAP [18] του Xilinx. Για να το δείξουμε χρησιμοποιήσαμε τα πρωτόκολλα ARM-Processor και AXI lite. Πιο συγκεκριμένα, μια εφαρμογή λογισμικού που εκτελείται σε ARM-Processor μεταφέρει τα partial bitstream από τη μνήμη flash QSPI του τσιπ και τα παραδίδει στο interface ICAP μέσω του πυρήνα AXI HWICAP όπως φαίνεται στο 7. Τα αρχεία αποθηκεύονται σε QSPI-flash και στη συνέχεια μεταφέρονται σε DDR επειδή η μνήμη flash είναι ευρέως διαδεδομένη χρησιμοποιείται σε όλες τις διαστημικές εφαρμογές και προς το παρόν δεν υπάρχει άμεσα διαθέσιμοεναλλακτική λύση με ανοσία στην ακτινοβολία.



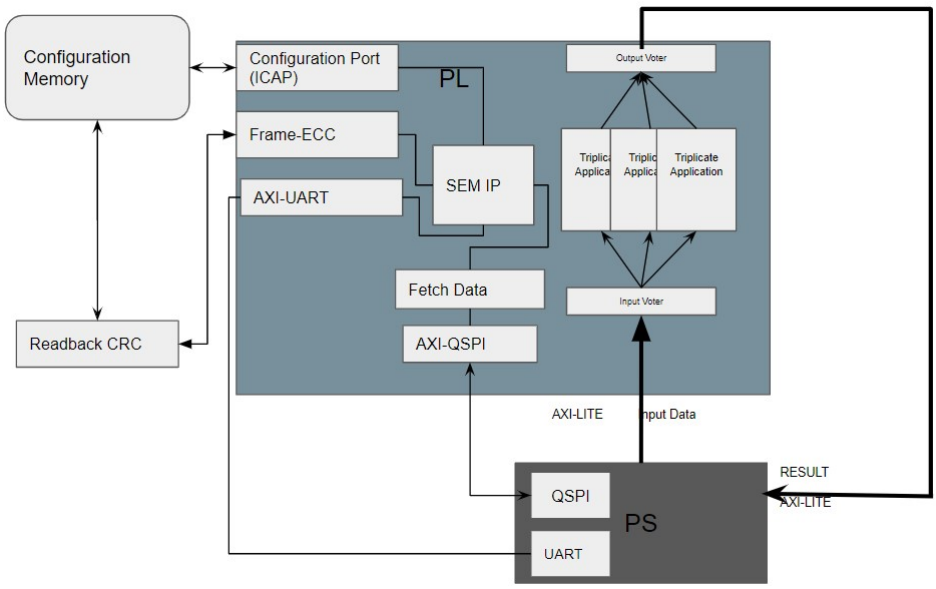
μ 7: Architecture of Dynamic Partial reconfiguration.

Προτεινόμενη αρχιτεκτονική βασισμένη σε Εσωτερικό Scubber και TMR

Για να εφαρμόσουμε αυτόν τον συνδυασμό Αρχιτεκτονικής, έχουμε εφαρμόσει μια Αρχιτεκτονική που βασίζεται στο TMR και το SEM IP ως Εσωτερικό Σχρυββινγ. Προκειμένου να εφαρμοστεί το TMR όπως περιγράφεται παραπάνω στην ενότητα TMR η εφαρμογή επιθυμίας (FIR) που θα πραγματοποιηθεί στη διαμόρφωση PL τριπλασιάζεται. Επίσης, όπως συζητήθηκε παραπάνω έχουμε εφαρμόσει δύο βασικά συστατικά. Η εισροή και η εκροή πλειοψηφία Ψηφοφόροι. Όπως έδειξε 9 το πρώτο τμήμα PS θα στείλει 3 σετ δεδομένων στο τμήμα PL.

Το buffer εισόδου λαμβάνει τα δεδομένα εισόδου και το Input IP Voter θα διανέμουν τα ίδια δεδομένα σε κάθε FIR. Έτσι, ένα σύνολο ίδιων δεδομένων χωρίζεται σε τρία διαφορετικά FIR για υπολογισμούς και κάθε κύκλος θα υπολογίστε 3 αποτελέσματα που θα σταλούν πίσω στο τμήμα PS. Ο ψηφοφόρος που βρίσκεται στην πλευρά PS θα συγκρίνει αυτά τα τρία σύνολα δεδομένων και θα επιλέξει το αποτέλεσμα που υπολογίστηκε πανομοιότυπα τουλάχιστον δύο φορές. Παράλληλα, εάν ένα SEU συνέβη στη διαδικασία υπολογισμού, θα καταγράψει και θα καταγράψει επίσης το FIR που προκάλεσε το σφάλμα. Μετά από αυτό, το PS θα στείλει τα νέα δεδομένα στο buffer δεδομένων και τα τρία FIR θα εκτελέσουν την ίδια λειτουργία με τη σειρά τους. Μετά την ολοκλήρωση των υπολογισμών ολόκληρης της εφαρμογής, η πλευρά PS θα ψηφίσει για την επιλογή του τελικού αποτελέσματος.

Ταυτόχρονα έχουμε το SEM Controller που χρησιμοποιεί το εσωτερικό υλικό Readback CRC για την ανίχνευση σφαλμάτων χρησιμοποιώντας τους ενσωματωμένους κωδικούς ECC και CRC [1]. Ο SEM IP, πραγματοποιεί τη δική του διόρθωση μέσω του ICAP χρησιμοποιώντας QSPI-flash για την αποθήκευση του χρυσού αντιγράφου καθώς και μέσω του AXI-UART έχουμε την αμφίδρομη εποικοινωνία με τον controller. Τα σχήματα 8 και 9 δε-

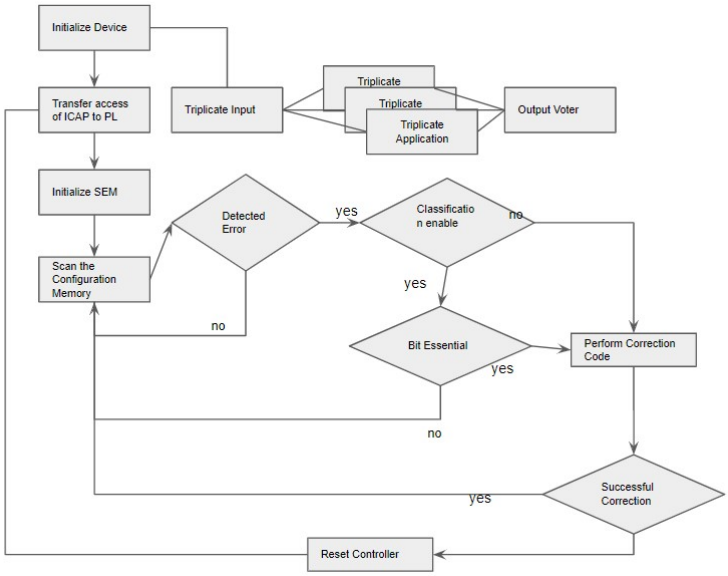


μ 8: Architecture of Internal Scrubber and TMR.

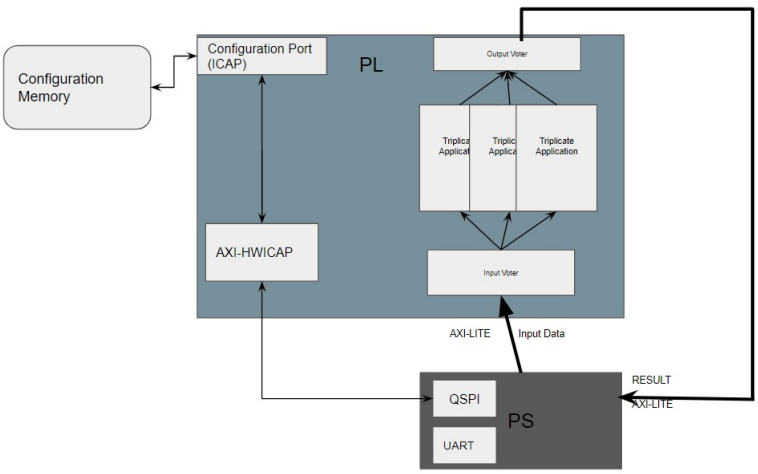
ίχνουν την αρχιτεκτονική που περιγράφεται και τη συμπεριφορά του συστήματος σε περίπτωση εμφάνισης Σφάλματος.

Προτεινόμενη αρχιτεκτονική βασισμένη σε συνδυασμό TMR με μερική αναδιαμόρφωση

Το TMR είναι μια πιθανή λύση για τον μετριασμό αυτών ανατροπές στο FPGA. Όπως περιγράφεται παραπάνω στο TMR ενότητα (M) τριπλασιάζεται και ένα κύκλωμα ψηφοφόρων της πλειοψηφίας παράγει το σωστό αποτέλεσμα. Αυτό η αρχιτεκτονική όπως περιγράφεται έχει ένα κύριο μειονέκτημα και αυτό είναι η απουσία έναν μηχανισμό επισκευής όταν έχει εντοπιστεί μόνιμο σφάλμα σε ένα από τα ενότητες. Το TMR παράγει εσφαλμένο αποτέλεσμα εάν ο ψηφοφόρος της πλειοψηφίας το κύκλωμα αποτυγχάνει να επιλέξει αποτέλεσμα. Ως εκ τούτου, το TMR μπορεί να εντοπίσει και να διορθώσει μόνο μία αποτυχία μονάδας κάθε φορά. Το πρόβλημα με το TMR είναι η εμφάνιση μη παρακείμενων ανατροπών διπλού γεγονότος (DEUs) που επηρεάζουν δύο μονάδες, γεγονός που προκαλεί αστοχία συστήματος. Το TMR λύνει Προβλήματα DEU εάν προκύψουν στην ίδια λειτουργική μονάδα αλλά παράγει ένα λάθος έξοδο όταν επηρεάζονται ταυτόχρονα δύο κελιά μνήμης δύο μονάδων. Έτσι, για να ενσωματώσουμε με επιτυχία το TMR στο σχεδιασμό μας, πρέπει να εισαχθεί ένας μηχανισμός επισκευής των ελαττωματικών μονάδων. Δυναμική μερική αναδιαμόρφωση (DPR) μπορεί να χρησιμοποιηθεί για την ανάκτηση της μονάδας που απέτυχε.



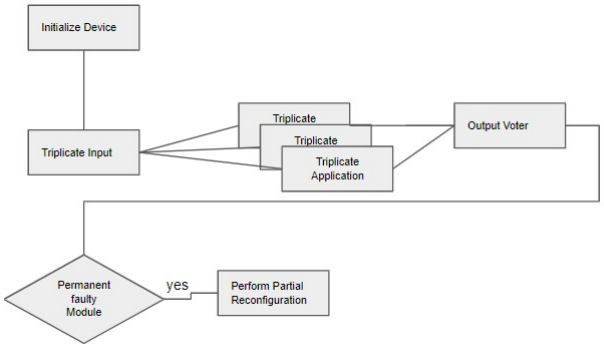
μ 9: Architecture flow of Internal Scrubber and TMR.



μ 10: Architecture of TMR and Dynamic Partial Reconfiguration.

Η προτεινόμενη αρχιτεκτονική σε αυτήν την ενότητα, όπως φα10 αποτελείται από TMR και Dynamic Partial Reconfiguration. Το TMR εφαρμόζεται στις προδιαγραφές του συστήματος, προκειμένου να εντοπιστεί και να μετριαστεί η εμφάνιση ανατροπής bit. Ο ψηφοφόρος δέχεται

τα τρία αναπαράγει αποτελέσματα και υπολογίζει τα σωστά δεδομένα. Επίσης στην περίπτωση της αποτυχία μιας μονάδας, σηματοδοτεί ποια από τις αναπαραγόμενες μονάδες υπολογίζει μια εσφαλμένη τιμή. Αυτές οι πληροφορίες θα χρησιμοποιηθούν για τον έλεγχο της διαδικασίας εκ νέου διαμόρφωσης σε περίπτωση που το σφάλμα αναγνωριστεί ότι επηρέασε την ελαττωματική μονάδα πολλές φορές, και αποκαλύπτοντας έτσι μια «μόνιμη» λανθασμένη συμπεριφορά.

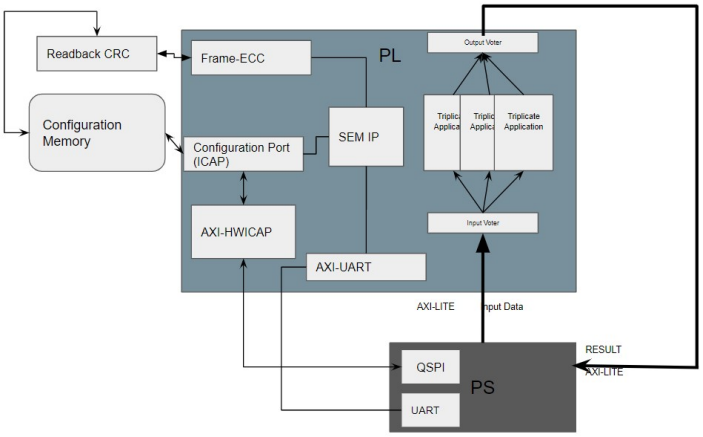


μ 11: Architecture flow of TMR and Dynamic Partial Reconfiguration

Πιο συγκεκριμένα, ο ελεγκτής που λαμβάνει τα αποτελέσματα των ψηφοφόρων και διαχειρίζεται την επαναδιαμόρφωση, καθορίζει εάν το σφάλμα παρουσιάστηκε σε προσωρινό καταχωρητή που χρησιμοποιείται για υπολογισμούς ή στη μνήμη διαμόρφωσης μέσω παρακολούθησης η έξοδος των ψηφοφόρων σε διαδοχικούς κύκλους, και στην τελευταία περίπτωση πυροδοτεί μια εκ νέου διαμόρφωση του τμήματος της μονάδας που επηρεάζεται από το σφάλμα. Η μερική αναδιαμόρφωση εκτελείται με τον ίδιο τρόπο, που χρησιμοποιήσαμε στην ενότητα. Όπου υπάρχουν partial bitsreams που παραδίδονται στο interface ICAP της συσκευής FPGA, μέσω του Xilinx Πυρήνας IP AXI HWICAP. Έτσι χρησιμοποιούμε τον επεξεργαστή ARM και τον AXI-lite πρωτόκολλα για την ανάγνωση δεδομένα διαμόρφωσης από τη μνήμη flash QSPI του τσιπ και παραδίδοντάς τα στη διεπαφή ICAP μέσω του πυρήνα AXI HWICAP όπως φαίνεται στο σχήμα 10. Όπως μπορούμε να δούμε στο 11 στην ανίχνευση ενος SEU, αυτή η αρχιτεκτονική εκτελεί δυναμική μερική αναδιαμόρφωση στην ελαττωματική μονάδα.

Προτεινόμενη αρχιτεκτονική βασισμένη σε συνδυασμό εσωτερικού Σκρυββερ με μερική αναδιαμόρφωση και TMR

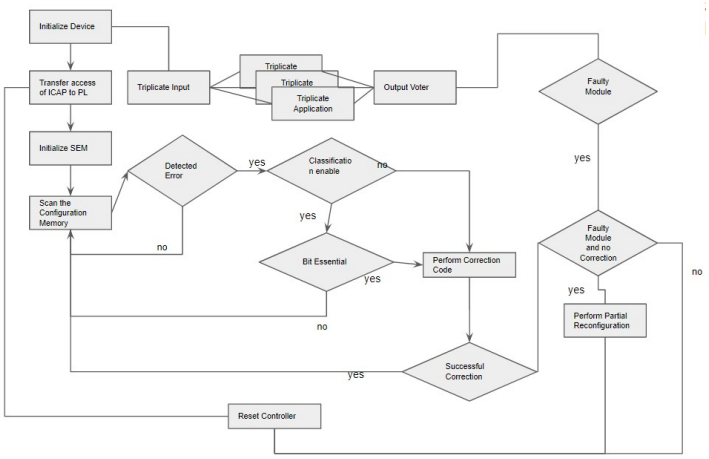
Σε αυτη την αρχιτεκτονική, το TMR μπορεί να εφαρμοστεί με επιτυχία χωρίς να δημιουργήσουμε πρόβλημα στο σχεδιασμό μας, εφόσον η ενότητα δεν υπερβαίνει το 33% της υπόλοιπης περιοχής PL. Τα μερικά αρχεία που είναι αποθηκευμένα στην QSPI-flash είναι αφιερωμένα την περιοχή των τριπλών μονάδων (FIR). Σε αυτήν την περίπτωση, όπου εκτελείται μερική αναδιαμόρφωση, η ελαττωματική μονάδα αντικαθίσταται από μια σωστή διαμόρφωση. Λαμβάνοντας υπόψη την μερική παραχώρηση του interface ICAP έχουμε εφαρμόσει ενα στιβαρό σύστημα που φαίνεται στο 12. Εκεί έχουμε συνδυάσει επιτυχώς SEM IP με την μερική αναδιαμόρφωση και TMR για την εφαρμογή ενός πιο στιβαρού συστήματος.



μ 12: Architecture of Internal Scrubber, TMR and Dynamic Partial Reconfiguration.

Η επαναδιαμόρφωση μπορεί να εφαρμοστεί διατηρώντας παράλληλα την ικανότητα αξιοπιστίας. Ο χρόνος της μερικής αναδιαμόρφωσης είναι 10 φορές μικρότερος από χρόνο αρχικοποίησης. Αυτό σημαίνει ότι η SEM IP είναι το σημείο συμφόρησης για το πόσο συχνά μπορεί να εφαρμοστεί μερική αναδιαμόρφωση. Αξίζει, για να πραγματοποιησουμε τη Μερική αναδιαμόρφωση,να γίνονται πολλές περιοχές ταυτόχρονα, ώστε να μπορούν να καλυφθούν με μία εκκίνηση SEM IP. Έτσι για να λύσουμε Αυτό το πρόβλημα εκτελούμε τη μερική αναδιαμόρφωση όταν μια δεύτερη ενότητα καθίσταται ελαττωματικό αυτό οδηγεί στην απώλεια ορισμένων σωστών αποτελεσμάτων εξόδου, αλλά μας δίνει την ευκαιρία να κερδίσουμε σε λιγότερο χρόνο διακοπής λειτουργίας. Έτσι, το λογισμικό του ARM-Processor είναι υπεύθυνο για την εκτέλεση της αναδιάρθρωσης μέσω του AXI HWICAP IP της Xilinx core [18] χρησιμοποιώντας τα επιμέρους αρχεία αποθηκεύονται στη μνήμη QSPI-Flash κάθε φορά που δεν λειτουργούν περισσότερες από μία μονάδες.

Σε σχέση με τον ελεγκτή SEM χρησιμοποιεί το εσωτερικό υλικό Readback CRCγια να εκτελεστεί ανίχνευση σφαλμάτων χρησιμοποιώντας τους ενσωματωμένους κωδικούς ECC και



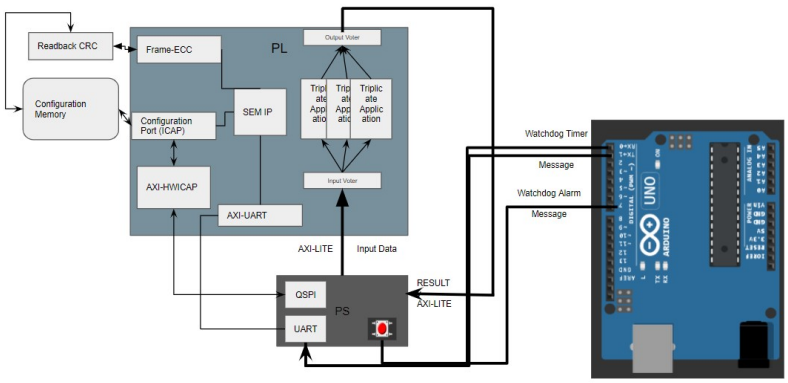
μ 13: Architecture flow of TMR and Dynamic Partial Reconfiguration and Internal Scrubber

CRC[1]. Ενώ ο SEM IP, πραγματοποιεί όμως τη δική του διόρθωση μέσω του ICAP. Έχουμε εφαρμόσει το SEM IP σε λειτουργία Βελτιωμένης επισκευής. Επιλέγουμε τη λειτουργία βελτιωμένης επισκευής καθώς χρειάζεται λιγότερο χρόνο από τη λειτουργία αντικατάστασης για τη διαμόρφωση ενός πλαισίου, επίσης, έχουμε δημιουργήσει ένα AXI-UART για λόγους επικοινωνίας. Τα σχήματα 12 13 παρακάτω δείχνουν την αρχιτεκτονική που περιγράφεται και τη συμπεριφορά του συστήματος σε περίπτωση εμφάνισης Σφάλματος. Όπως θα δούμε στην ενότητα Αποτελέσματα, είναι μια πολύ αποτελεσματική Αρχιτεκτονική που παρέχει υψηλή αξιοπιστία στο σύστημά μας. Το μόνο πρόβλημα με αυτήν την αρχιτεκτονική είναι ότι υπάρχουν ορισμένες καταστάσεις στις οποίες το σύστημα πέφτει, και ο μόνος τρόπος για να λυθεί αυτό σημαίνει να επαναφέρετε ολόκληρη τη διαμόρφωση σε ένα σημείο στο οποίο το σύστημα λειτουργούσε όπως θα έπρεπε.

Προτεινόμενη αρχιτεκτονική βασισμένη σε συνδυασμό Ωατσηδογ και Internal Scrubber με Μερική αναδιαμόρφωση και TMR

Παρατηρήθηκαν σφάλματα που δεν ήταν ανιχνεύσιμα από όλες τις παραπάνω αρχιτεκτονικές οπότε θα πρέπει να εισαχθούν νέες τεχνικές μετριασμού προκειμένου να επιτευχθεί α ολοκληρωμένη λύση για κάθε Upset. Για να τα διορθώσουμε λοιπόν λάθη που έχουν εισαχθεί στο σύστημα. Ενας Watchdog Timer θα πρέπει να είναι σε θέση να ανιχνεύει όλους τους μη φυσιολογικούς τρόπους λειτουργίας λογισμικού και επαναφέρετε το σύστημα σε μια γνωστή κατάσταση. Θα πρέπει να έχει το δικό του ρολόι και θα πρέπει να μπορεί να παρέχει επαναφορά υλικού κατά το χρονικό όριο σε όλα τα περιφερειακά συστήματα. Το προτεινόμενο Watchdog Timer λειτουργεί ανεξάρτητα από το επεξεργαστή και το FPGA. Η αρχιτεκτονική ακολουθεί έναν Watchdog Timer με παράλληλη εκτέλεση. Μια σημαία αποτυχίας υψώνεται όταν λήξει το χρονόμετρο παρακολούθησης και μετά από ένα καθορισμένο χρονικό διάστημα

από την ανύψωση της σημαίας, ενεργοποιείται μια επαναφορά.



μ 14: Architecture design of TMR, Dynamic Partial Reconfiguration, Internal Scrubber and Watchdog Timer.

Στη διπλωματική μας, αυτή η εργασία παραδίδεται από έναν ελεγκτή Arduino. Ο ελεγκτής λαμβάνει περιοδικά ένα μήνυμα από το FPGA μας. Στην απουσία αυτού του μηνύματος, το Arduino είναι υπεύθυνο για την επαναφορά του συστήματός μας. Λόγω προβλημάτων στον πίνακα ZYBO αυτή η επαναφορά πραγματοποιείται πατώντας ένα κουμπί στο FPGA μετά από την εμφάνιση μηνύματος από το Αρδυινο. Αυτή είναι μια δαπανηρή Αρχιτεκτονική γιατί χρησιμοποιεί πολλούς πόρους και κατανάλωση ενέργειας, αλλά εάν το FPGA μας δεν είναι εύκολα προσβάσιμο, τότε είναι υποχρεωτικό να συμπεριλάβουμε αυτή την τεχνική στο σχεδιασμό μας.

Η προτεινόμενη αρχιτεκτονική αυτού του στιβαρού συστήματος για την αποφυγή της καταστροφής παρουσία του SEFI και των μη διορθωμένων MBU ή SBU υλοποιείται και φαίνεται παρακάτω. Έχει αναπτυχθεί ένα σύστημα ανοχής σε σφάλματα το οποίο βασίζεται σε πολλαπλά ιεραρχικά επίπεδα συμπληρωματικών σχημάτων που χρησιμοποιούνται για την ανίχνευση και την διόρθωση του σφάλματος όπως αναπαρίσταται καλύτερα στο 14.

Συγκρίσεις

Σε αυτήν την ενότητα, θα συζητηθούν όλες οι αρχιτεκτονικές μαζί μέσα απο τους διάφορεους μετρητές αξιολόγησης. Αρχικά, εξετάζουμε τη λειτουργικότητα του FPGA κατά τη διάρκεια το τεστ μας. Τα αποτελέσματα του πίνακα 0.2 δείχνουν το τμήμα του χρόνου που το FPGA δεν είναι προσβάσιμο (Downtime) και έχει λανθασμένη/σωστή λειτουργικότητα. Όπως φαίνεται, ο χρόνος διακοπής λειτουργίας του συστήματος είναι στο 92% της εφαρμογής ενώ κάθε άλλη ανοχή σφαλμάτων μόνο βελτιώνει το χρόνο διακοπής λειτουργίας του συστήματος σε 73% (internal Scrubber, SEM IP), 79% (Partial Reconfiguration) και 80%

(TMR). Επίσης το Internal Scrubber (SEM IP) παρουσιάζει τη σωστή λειτουργία για περισσότερο χρόνο (22%) ενώ το TMR και η μερική αναδιαμόρφωση περιορίζονται στο 1–8%. Το πλεονέκτημα του Internal Scrubber (SEM IP) βρίσκεται στον εντοπισμό σφαλμάτων στη δημιουργία τους καθώς ελέγχει τη μνήμη διαμόρφωσης και δεν εξαρτάται από την έξοδο της εφαρμογής. Επί πλέον,το TMR παρέχει σχεδόν μηδενική σωστή λειτουργικότητα. Αυτό συμβαίνει λόγω της φύσης του πειράματός μας. Εάν οι εμφυτεύσεις δεν είχαν καθοριστεί στο εξάρτημά μας, το TMR θα είχε καλύτερα αποτελέσματα.

Architecture	**Test duration** (ms)	**Downtime** (ms)	**Non-Functional Time** (ms)	**Functional Time** (ms)
–	40000	36800	3180	20
TMR	40000	31878	7904	218
SEM	55000	40200	2800	12000
PR1	40956	32596	4592	3408
TMR & PR2	40642	14496	16642	9504
TMR & SEM	58830	18450	14000	26380
TMR & PR2 & SEM	54826	14826	12336	27664
TMR & PR2 & SEM &Watchdog	49896	9896	4028	35972

0.1: Test time duration of all Architectures.

FT Architecture	Downtime/ No Functionality	Erroneous Functionality	Correct Functionality
–	92%	8%	0%
TMR	80%	19%	1%
SEM	73%	5%	22%
PR1	79%	13%	8%
TMR & PR2	36%	41%	23%
TMR & SEM	31%	24%	45%
TMR & PR2 & SEM	27%	22%	51%
TMR & PR2 & SEM &Watchdog	20%	8%	72%

0.2: Evaluation test time comparisons of all Architectures.

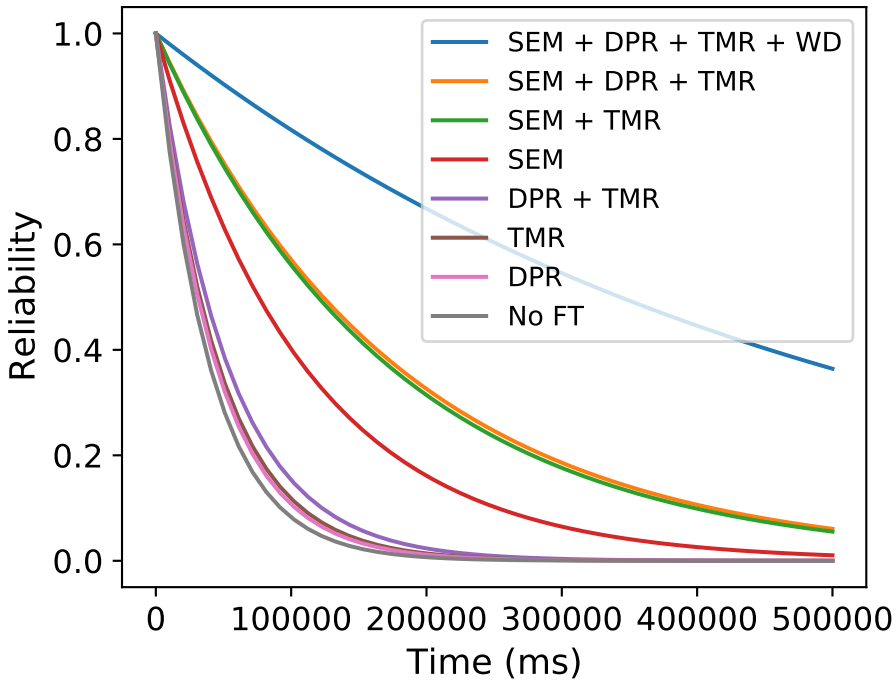
Ωστόσο, όταν συνδυάζεται το TMR είτε με Internal scrubber (SEM IP) είτε με PR, η λειτουργικότητα του FPGA βελτιώνεται ριζικά. Αυτή η βελτίωση οφείλεται στο γεγονός ότι τα σφάλματα κατανέμονται και σε μεγαλύτερη σχεδιαστική λογική 0.3 αλλά και στην δυνατότητα της Μερικής Αναδιαμόρφωσης να συνδυαστεί με ένα σύστημα ανίχνευσης και του Εσωτερικού Scrubber να καλύπτει την έξοδο σφάλματος μέσω του τριπλασιασμού της εξόδου. Όπως μπορούμε να παρατηρήσουμε ο συνδυασμός και των τριών τεχνικών αυξάνουν ακόμη περισσότερο την αξιοπιστία του συστήματος, καθώς το FPGA λειτουργεί σωστά κατά το ήμισυ του πειράματος. Τέλος, το πλεονέκτημα του συνδυασμού που περιλαμβάνει ένα Watchdog Timer είναι ότι, εκτός από τον γρήγορο εντοπισμό σφαλμάτων, καταφέρνει να επαναφέρει τον Internal Scrubber (SEM IP) σε περίπτωση μόνιμου σφάλματος. Ως αποτέλεσμα, αυξάνει

σημαντικά την ευρωστία του συστήματος, παρέχοντας 72% σωστή λειτουργία.

Architecture	Injections(max)	Mem. Confi + BRAMs Utilization	Mem Utiliz. Incr.	# Errors Detected	# Errors Corrected
–	800	3.8%+3.1%	x1	0	0
TMR	1500	8,3%+ 12.5%	x2.5	300	218
SEM	7000	6,7% +4.6%	x1.75	5013	5012
PR1	2000	23%+95%	x6	0	852
TMR & PR2	10000	6,1%+12.5%	x2.7	4000	2376
TMR & SEM	10000	10,6%+17.2%	x3.3	7883	6595
TMR & PR2 & SEM	10000	12,8% +17.2%	x4.1	9900	9100
TMR & PR2 & SEM &Watchdog	10000	17,5% +17.2%	x4.5	9956	9884

0.3: Comparison Architecture in terms of Memory Utilization, Errors Detected and Corrected.

Με βάση τα ίδια αποτελέσματα, το σχήμα 15 δείχνει την αξιοπιστία των Αρχιτεκτονικών ανοχής σφαλμάτων με την πάροδο του χρόνου. Υποθέτουμε εκθετικά κατανεμημένα τυχαία σφάλματαμε τιμή λ και ότι η αξιοπιστία του συστήματος ορίζεται ως $R(t) = e^{-\lambda * t}$ [19]. Το μοτίβο παραμένει το ίδιο, όπως και εδώ οι πιο αξιόπιστες αρχιτεκτονικές είναι ο συνδυασμός. Ωστόσο, για αυτήν τη μέτρηση, η Μερική αναδιαμόρφωση παρέχει λιγότερη αξιοπιστία, επειδή στερείται μηχανισμού ανίχνευσης, με αποτέλεσμα μια σχεδόν πάντα εσφαλμένη κατάσταση.



μ 15: Reliability R(t) of all Architectures for 10K injections in the experiment operational time.

Ο πίνακας 0.4 δείχνει τη χρήση των διαθέσιμων πόρων FPGA για κάθε Αρχιτεκτονική μετριασμού. Όσον αφορά τους πόρους, όλες οι αρχιτεκτονικές ανοχής σφαλμάτων παρουσιάζουν μικρή αύξηση στην χρήση των πόρων του FPGA, δηλαδή λιγότερο από 4K LUT, 5K FF, 6 RAMB. Η κύρια διαφορά μεταξύ των Αρχιτεκτονικών μπορούν να παρατηρηθούν στο

ποσοστό χρήσης BRAM και IO του Partial Reconfiguration και γι΄ αυτό έχουμε εφαρμόσει το δικό μας ICAP FSM που χρησιμοποιεί το BRAM της PL για να αποθηκεύσετε το μερικό αρχείο που είναι υπεύθυνο για την περιοχή του φίλτρου FIR. Διαφορετικά βλέπουμε κανονική αύξηση των πόρων του FPGA. Κάτι που είναι κατανοητό καθώς αυτή η αύξηση είναι παράλληλη με το συνδυασμό πολλαπλών τεχνικών μετριασμού σε ένα σχέδιο.

Architecture	**LUTs**	**FFs**	**IOs**	**BRAMs**
Non-FT FIR (app)	609	1461	4	1
TMR	1316	2327	11	4
SEM	2021	2761	1	1.5
PR1	1112	1756	59	57
PR2	1315	2501	4	0
TMR & PR2	2022	3367	11	4
TMR & SEM	2727	3801	12	5.5
TMR & PR2 & SEM	3433	4841	15	5.5
TMR & PR2 & SEM &Watchdog	3756	5072	17	5.5

0.4: Comparison of all Architecture.

Ως εκ τούτου, υπάρχει χώρος για την υλοποίηση άλλων στοιχείων απαιτείται στην αρχιτεκτονική συνεπεξεργασίας, π.χ. περισσότερα DSP, κωδικοποιητές δεδομένων και συμπιεστές δεδομένων. Τελος το SEM IP καταλαμβάνει το μεγαλύτερο χώρο στην επιφάνεια του FPGA (x2,5 περισσότερα από άλλες τεχνικές ανοχής σφαλμάτων). Η αύξηση πόρων στην αρχιτεκτονική FT με το Watchdog Timer (x4.5) οφείλεται στην επικοινωνία UART μεταξύ του Arduino (μικροελεγκτής). Τέλος, σύμφωνα με τις μετρήσεις μας, το SEM διορθώνει ένα frame μνήμης 404 bytes σε 18 ms ενώ η απόδοση της μερικής αναδιαμόρφωσης μέσω HWICAP είναι 67 MB/s.

Τα αποτελέσματα που παρουσιάζονται στον Πίνακα 0.5 υποδηλώνουν ότι COTS FPGA που βασίζονται σεSRAM από τη Xilinx μπορούν πράγματι χρησιμοποιηθούν σε ορισμένες διαστημικές εφαρμογές. Για κατάλληλες εφαρμογές, όπως τα μη κρίσιμα όργανα ή οι επικοινωνίες, τα FPGA που βασίζονται σε SRAM μπορούν να παρέχουν μια πολύ αποδοτική εναλλακτική από πλευράς κόστους προς περιοχή. Το TMR πρέπει να συνδυαστεί με το ιντερναλ σκρυββινγ για να είναι αποτελεσματικό, διαφορετικά θα υποφέρει τελικά από συσσώρευση σφαλμάτων. Τα αποτελέσματα που παρουσιάζονται στο 0.5 ισχύουν για την επιλεγμένη εφαρμογή, δηλαδή το μπλοκ FIR. Εφαρμογές με φυσικά παράθυρα διακοπής λειτουργίας μπορούν επωφεληθείτε από το απλό περιοδικό scrubbing. Μερική αναδιαμόρφωση σε συνδυασμό με το TMR αποτελεί μια αποτελεσματική εναλλακτική ανοχή σφαλμάτων , αλλά υπάρχει σημαντική επιβάρυνση πόρων και η επίβλεψη περιορίζεται στις περιοχές όπου εκτελείται μερική αναδιαμόρφωση. Το εσωτερικό Scrubber που βασίζεται στον ελεγκτή SEM Xilinx αντιπροσωπεύει μια αποδοτική από πλευράς πόρων επιλογή που μπορεί να συνδυαστεί με πλήρη επαναδιαμόρφωση παρέχει μια επαρκή λύση για ένα ισχυρό σύστημα ανοχής σφαλμάτων.

Architecture	Number of Injections			
	100	1000	5000	10000
–	99%	-	-	-
TMR	3%	78%	-	-
SEM	11%	47%	50%	-%
PR1	20%	49%	87%	91%
TMR & PR2	29%	66%	75%	76%
TMR & SEM	7%	8%	12%	34%
TMR & PR2 & SEM	2%	7%	8%	31%
TMR & PR2 & SEM &Watchdog	3%	1%	3%	10%

Architecture	Number of Injections			
	100	1000	5000	10000
–	101%	-	-	-
TMR	33 %	832%	-	-
SEM	0 %	8.6%	15%	-
PR1	7.5%	53%	150%	188%
TMR & PR2	2.4%	25.3%	54.3%	67%
TMR & SEM	5.9%	8.9 %	12 %	40%
TMR & PR2 & SEM	1.1%	2.4%	3.7%	21%
TMR & PR2 & SEM &Watchdog	4%	5.7%	7.2%	7.8%

0.5: Comparison of all Architectures in terms of Error Rate (upper table) and Mean Relative Error (lower table)

Συμπεράσματα

Τα πειραματικά σενάρια αξιολόγησης που συζητούνται σε αυτή την διπλωματική δείχνουν ότι τα COTS FPGAs που βασίζονται σε SRAM μπορούν να χρησιμοποιηθούν σε διαστημικές εφαρμογές, αλλά τόσο τα δεδομένα χρήστη όσο και η μνήμη διαμόρφωσης πρέπει να προστατεύονται. Το proof-of-concept μας έχει σχεδιαστεί σε FPGA-SoC με πυρήνες ARM Cortex-A9 που χρησιμοποιούνται ευρέως, ενώ η αναπαραγώγιμη παραλλαγή σχεδίασης που περιγράφεται σε αυτήν τη συνεισφορά χρησιμοποιεί προγραμματιζόμενη λογική για την εφαρμογή τεχνικών μετριασμού. Μπορεί να αναπαραχθεί μόνο με τυπικά εργαλεία σχεδίασης και IP βιβλιοθήκης, τα οποία είναι διαθέσιμα σε χαμηλό κόστος σε πολλούς σχεδιαστές σε ακαδημαϊκούς και ερευνητικούς οργανισμούς.

Αυτή η Διπλωματική εκμεταλλεύεται τέσσερις τεχνικές Μετριασμού Internal configuration scrubbing, TMR, Dynamic Partial Reconfiguration και Watchdog Timer. Το πείραμα εξέτασε όλες τις τεχνικές μετριασμού για SEU από τον απουσία τεχνικών μετριασμού έως τον σχεδιασμό όλων των τεχνικών. Η δοκιμή αυτών των συνδυασμών βοήθησε να απομονωθούν τα οφέλη της κάθε SEU τεχνική μετριασμού και εξήγησε τις συμπληρωματικές σχέσεις μεταξύ τους. Η βελτίωση μετριέται ως προς τη βελτίωση της αξιοπιστίας και επίσης τη μείω-

ση του ποσοστού σφάλματος και του μέσου σχετικού σφάλματος στο ίδιο πείραμα έγχυσης σφάλματος. Τα αποτελέσματα από την έγχυση σφαλμάτων δείχνουν ότι κάθε παραλλαγή των τεχνικών μετριασμού για SEUs βελτιώστε την ευαισθησία SEUs του συστήματος στην ανοχή σφαλμάτων και αυτή η βελτίωση αυξάνεται καθώς συνδυάζονται περισσότερες τεχνικές μετριασμού. Η βελτίωση που επιτυγχάνεται από τον καθαρισμόεν Internal configuration scrubbing είναι μεγαλύτερη από τις άλλες τεχνικές μετριασμού. Όταν και οι τέσσερεις τεχνικές μετριασμού συνδυάζονται υπάρχει βελτίωση x72. Κάτι που δεν παρατηρήσαμε στη διπλωματική μας ήταν την αυξημένη ευαισθησία λόγω της λογικής που προστέθηκε σε κάθε αρχιτετκονική. Αυτό ήταν αναμενόμενο λόγω της φύσης του πειράματός μας και του γεγονότος ότι εμείς στοχεύσαμε σε ευαίσθητα bits και όχι σε όλη τη μνήμη του FPGA.

Η αρχιτεκτονική που αναλύεται σε αυτή τη διπλωματική συνδυάζει μερικές αποτελεσματικές τεχνικές μετριασμού ανοχής σφαλμάτων στην προσπάθεια επιτυχίας το καλύτερο δυνατό αποτέλεσμα. Παρακάτω στο 0.6 υπάρχουν οι τεχνικές μετριασμού που αναλύονται σε αυτή τη διπλωματική καθώς και οι δυνατότητες κάθε Αρχιτεκτονικής.

Architecture	Detection	Correction	Error Types	Device Reboot
No Mitigation Technique	No	No	-	No
TMR	Yes	Yes	Transient	No
SEM	Yes	Yes	Transient & Permanent	No
PR	No	Yes	Transient & Permanent	No
TMR & PR	Yes	Yes	Transient & Permanent	No
TMR & SEM	Yes	Yes	Transient & Permanent	No
TMR & PR &SEM	Yes	Yes	Transient & Permanent& SEFI	YEs
TMR &PR &SEM & Watchdog Timer	Yes	Yes	Transient & Permanent& SEFI	YEs

0.6: Comparison of all SEU mitigation Architectures

Chapter 1

Introduction

The continued innovation in the technology for developing and manufacturing FPGAs has allowed to increase the integration level. This high integration enables features, such as high capacity, low-power consumption and faster operation. However this integration has some drawbacks. One is that the space radiation produces more faults. Hence this sensitivity that initial was a problem in spacial application now has been an actual concern even form a ground-level applications. The different effects of of these faults is called Single Event Effects (SEEs). When working with SRAM based FPGA ,the most dangerous effect of this group are Single Event Upsets(SEU). SEUs are non-permanent soft errors produced by contaminant alpha particles in electronics or when protons and cosmic rays from outer space interact with the atmosphere and generate subatomic particles that collide with silicon atoms. SEUs can affect FPGAs flipping bits in both, user memories or the configuration memory. SEUs located in the configuration memory are the most critical case because they can alter the implemented design by changing the configuration of crucial elements or their interconnections. The repercussion of the SEUs on the configurations relies on the application in which the FPGA device is implemented, including the number of configuration bits.

As a result of an upset the entire system or a critical part of it can be unavailable or present malfunction. For this reason, in certain applications, especially in fields related to human safety or very valuable technologies, such as railway, avionic or spacial applications the result of a single error can be catastrophic. In those cases, high levels of reliability are required, which demands to apply fault tolerance techniques to harden the designs. Space is among the environments where the on-board processors and accelerators are directly exposed to radiation. The space industry employs radiation-hardened space-grade FPGAs [20, 21, 22, 23] due to their increased reliability. However, targeting improved performance, there is a trend to use Commercial Off-The-Shelf (COTS) [24] devices, including classic FPGAs and SoC FPGAs [25, 26, 27]. These FPGAs are parts of mixed-criticality architectures [28, 29, 30], which integrate both space-grade and COTS devices. Nevertheless, the COTS devices are more susceptible to radiation and its effects, and thus, they need to increase their reliability using fault-tolerance mitigation techniques.

This thesis presents various Fault Tolerance Architectures to suggest that commercial zynq-7000 System-On-Chip SRAM-based FPGAs from Xilinx can indeed be used in those case. For suitable applications SRAM based FPGAs can provide a very cost-to-area-efficient alternative. The FPGAs considered in this thesis are limited to zynq-7000 System-On-Chip SRAM-based FPGAs as opposed to anti-fuse, flash-based, or other types of FPGAs. The programming cells of anti-fuse FPGAs are not sensitive to radiation, but they also do not support the ability to be reconfigured in the field. Flash-based FPGAs are generally smaller and less flexible than SRAM-based FPGAs [31]. Furthermore, the FPGAs considered in this work are exclusively Xilinx FPGAs.

1.1 Thesis Contribution

The contributions of this thesis are the followings:

- A comprehensive reference of mitigation techniques such as Internal scrubbing, Triple Modular Redundancy and Partial reconfiguration operations and the hardware necessary for implementing those mitigation techniques in Zynq 7-Series FPGAs.

- Implementation of seven fault tolerance architectures, aiming to protect the Zynq SoC FPGAs from SEUs. Three of them incorporate separately traditional mitigation techniques (TMR, Internal Scrubber and Partial Reconfiguration) while the other combine them partially. The last Architecture integrates all the abovementioned techniques alongside Watchdog Timer implemented in an Arduino Controller.

- Evaluation based on Reliability, Availability, Error Rate and Mean Relative Error of the implementations of the Fault Tolerance Architectures for the Zynq-7000 SoC family through the injection test setup.

. According to our experiments on the Zybo board, each proposed variation of the SEU mitigation techniques improves the sensitivity of the fault-tolerant system. At the same time, we observe that sensitivity improves, as more mitigation techniques are involved, i.e., in our hybrid fault-tolerant architectures. When comparing individually the architectures, we conclude that Internal Configuration Scrubbing significantly outperforms the other mitigation techniques. The most robust solution is the combination of all the mitigation techniques, which compared to the unmitigated design, provides 72% more correct functionality, 72% less downtime and 98% correction capability in the occurrence of SEUs, while using less than 4K LUTs, 5K FFs, and 6 RAMBs.

The Fault Tolerance Architectures presented in this Thesis are ideal for space applications as well as high-energy physics experiments, which must function reliably in environments with high upset rates. These robust architectures can be customized and used from FPGA users in order to meet the needs of their digital designs.

1.2 Thesis Organization

Chapter 2 provides a general background of the basic layout of FPGAs, bitstreams, essential bit as well as Single Events upset in the Configuration Memory. This background discusses where ionizing radiation comes from, presents the effects of ionizing radiation on FPGAs, and explains how each FPGA's Configuration Memory is susceptible to ionizing radiation . Chapter 2 will also introduce common upset mitigation mechanisms that have been implemented previously to handle radiation upsets. Variations of these mechanisms serve as building blocks for the architectures presented in this thesis.

Chapter 3 explains the configuration details necessary for implementing Triple Module Redundancy, Internal scrubbers and Partial Reconfiguration on Xilinx FPGAs as well as reviews existing mitigation solutions developed primarily for older FPGA families. Also in Chapter 3 we described the proposed Architecture for each Mitigation Technique. Chapter 3 presents various Hybrid Architectures developed by this thesis with complete methodologies and explanations of the components needed to implement the architectures.

Chapter 4 also gives the results of injections tests performed on Xilinx's Zynq-7000 SoC commercial board namely Zybo running all the architectures proposed in Chapter 3 . The performance metrics as well the test setup and the injection strategy of all architectures is also described in chapter 4. This work will conclude in Chapter 5.

Chapter 2

Background

SRAM-based FPGAs are sensitive to ionizing radiation. As FPGA transistor size decreases, radiation particle strikes have a greater impact on the SRAM memory cells of FPGAs. When used in space or other radiation environments, SRAM-based FPGAs require a certain level of reliability and protection from radiation to function properly. This chapter will address how upsets occur from ionizing radiation, where the radiation comes from, and how an FPGA can be detrimentally affected.

This chapter will provide important configuration details include a summary of Xilinx FPGA architecture and an explanation of configuration bitstreams. Most of the information presented in this chapter is explicitly given in the 7-Series Configuration Guide. In addition, this chapter will give a sufficient background of mitigation techniques employed to remedy these upsets. Understanding the effects of radiation upsets as well as the foundational mitigation strategies employed to handle them is important to comprehending the configuration scrubbing techniques discussed later in this thesis.

2.1 FPGA Configuration

FPGAs are integrated circuits made up of a number of smaller configurable building blocks including, but not limited to, the digital logic circuitry (Complex Logic Block), memory elements (Block RAM), input/output circuitry (IO Block), signal processing blocks (Digital Signal Processing Block), clocking circuitry (Clock Management Tile), interconnect switches etc. The functionality of the FPGA is defined by the configuration and interconnection of these blocks. The configuration of these blocks and interconnects is defined by a memory region referred to as the configuration memory. The standard configurable resource in a Xilinx FPGA is known as a Configuration Logic Block (CLB). CLBs are the most prevalent of the physical building blocks that are connected together to make up a user's design in hardware. CLBs are tiled across an FPGA in rows and columns with channels of routing wires lying in between them (see Figure 2.1). CLBs contain various amounts of logic resources. For instance, a 7-Series CLB contains eight 6-input Look-up-Tables (LUTs), 16 Flip-Flops (FFs), arithmetic carry logic, and wide-function

multiplexers [32]. Some CLBs contain elements that support the ability to implement shift registers and distributed RAM. Fig. 2.1 gives a breakdown of CLB resources.

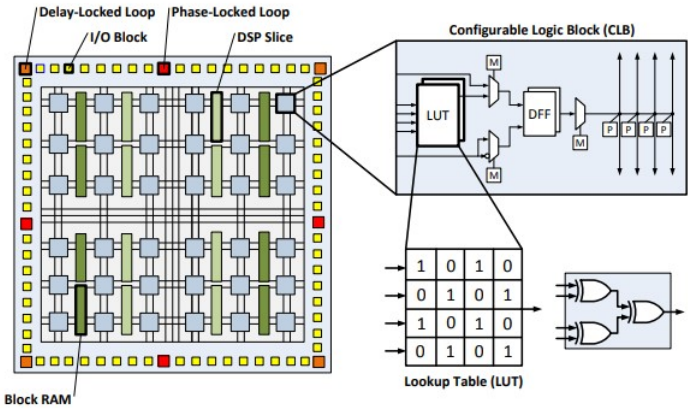


Figure 2.1: FPGA Architecture Overview.

Fig. 2.2 illustrates a LUT as a 4:16 decoder, and shows an illustration of the underlying configuration memory with each LUT bit stored in an SRAM cell.

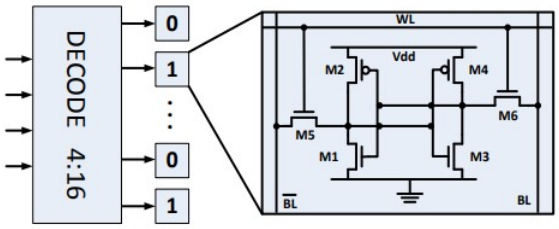


Figure 2.2: LUT Configuration.

The routing matrix and CLB-internal routing structures are made up of switchboxes, multiplexers, buffers and programmable interconnect points (PIP). All of these routing resources are configured by the corresponding bits in the configuration memory. Fig. 2.3 shows a section of the FPGA with four CLBs and their local interconnect matrix. The top fold-out shows an example of a connection box, with the crosses representing active, "on", PIPs. Each PIP is configured to be active or inactive by a single bit in the configuration memory. The bottom fold-out shows a switchbox, with fully configurable connections between all vertical and horizontal connection lines.

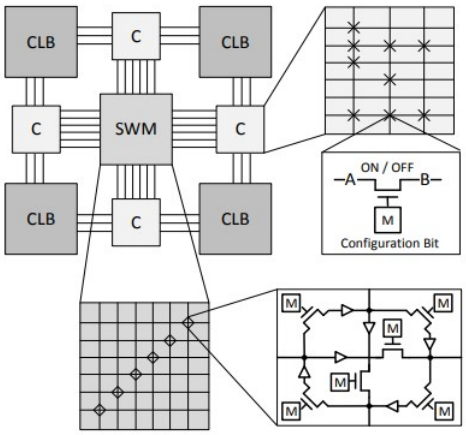


Figure 2.3: Interconnect Matrix.

The contents of the configuration memory are generated by the design tool software suite in the form of a configuration bit-stream. The configuration bit-stream is loaded into the memory through the programming interface(s) which makes the device perform the required functionality [33].The configuration memory defines the function and operation of all the described resources as well as the routing and connections on the FPGA, and can be seen as an underlying device definition layer. Fig. 2.1 gives an overview of a generic FPGA architecture. The fold-out illustrates a simplified LUT-DFF pair inside a slide, inside a CLB. In this particular example the LUT implements an XOR function.

Bitstreams are divided into groups of 32-bit words known as frames. Frames are the smallest addressable unit of the configuration memory space [34]. All configuration operations must act on one or more frames, as opposed to individual words or bits. The mapping between the bits in the bitstream and the specific FPGA resource that the bit controls is not made public by Xilinx. Fortunately, this information is not needed to implement configuration scrubbing.

Theoretically, any change in the configuration memory can result in a disruption of the FPGA functionality. But this is not always the case, as all the FPGA resources are rarely used. Hence the configuration memory bits that correspond to used FPGA resources are a candidate to affect the functionality and are referred to as the essential bits.

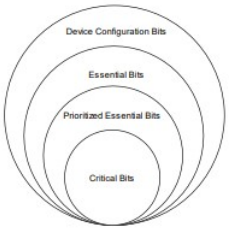


Figure 2.4: Critical Bits.

Xilinx essential bits technology [35] is an algorithm that identifies the essential bits in the configuration bit-stream. Although essential bits are the bits that define the circuit in the FPGA, any change in an essential bit may or may not cause a functional fault as the fault may be masked in space (part of the defined circuit that is not executed) or in time (the bit flip might create a fault only on specific combination of signals or specific clock cycle). The bits that cause a functional fault are known as critical bits. Essential bits that correspond to a certain hierarchical region of the FPGA design might be useful in some cases and are referred to as prioritizes essential bits [36]. Figure 2.4 shows the relation between configuration bitstream and its different sub-sets.

2.2 Ionizing Radiation Overview

Ionizing radiation is any type of particle or electromagnetic wave that carries enough energy to ionize or remove electrons from an atom [2, 3, 4]. For transistors that make up memory cells (such as SRAM cells), ionizing radiation can disrupt the transistor state by causing surges of current to flow or by stopping currents from flowing. Ionizing radiation comes from a variety of sources and has a wide range of effects on FPGAs.

2.2.1 Radiation Sources

Due to the advantages that modern FPGA devices offer, they are widely used in many different applications as explain in [37, 38]. Various environments in which those devices are incorporated also include radiation environments, in particular:

- Particle accelerators and high-energy physics experiments.
- Space environment: Low Earth Orbit, Geostationary Orbit, and deep space.
- High-altitude environment.
- Nuclear plants and nuclear research facilities.
- Nuclear medicine and imaging apparatuses.
- Weapons and military applications.

Single Event Effects

Single Event Effect is caused when radiation particles collide with the semiconductor lattice of transistors, their energy is transferred to the atomic particles of the transistor. These radiation strikes may cause depletion regions to form in the transistors by leaving behind an excess of electrons or holes in their wake [5]. Depending on the orientation of the free electrons or holes generated, electrical current may begin to flow, or cease flowing, which in turn could cause the digital circuit to change its logical state (see Figure 2.5). When such an event occurs in an FPGA, it is known as a Single Event Upset (SEU).

As see in 2.1 Xilinx FPGA configuration memory is organized into units called frames which are the smallest addressable units on an FPGA. If an SEU only affects one logical bit in a frame, it is known as a Single-Bit Upset (SBU). SBUs in different frames are each classified as separate SBUs for correction purposes. When multiple upsets occur in the same frame, the result is a Multi-Bit Upset (MBU). A single event that causes multiple bits in the same frame to upset is a Multi-Cell Upset (MCU) [6]. As noted in [7], MBUs are increasingly more likely as the transistor process technology continues to shrink. The same-size atomic particles bombard ever smaller device geometries, causing increasing numbers of upset events. Upsets to SRAM configuration cells can actually change the functionality of the circuit design. An example of a digital circuit implemented on an

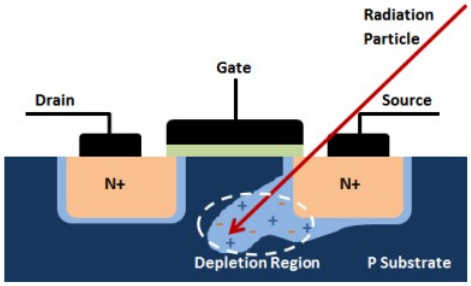
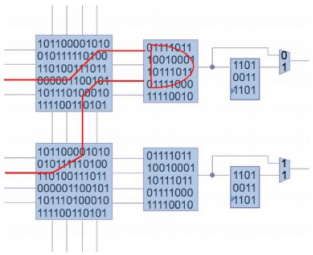
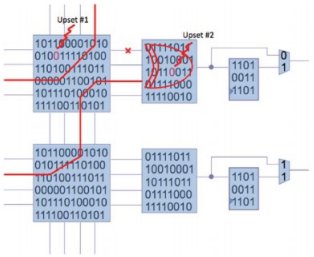


Figure 2.5: The Effect of a Radiation Particle Strike on a Transistor.



(a) Routing and Logic Before Upset.



(b) Routing and Logic After Upset.

FPGA is shown in Figure 2.6a. The logical bits of the configuration memory are shown defining the routing and function of FPGA resources used to implement a two-input AND gate.When one upset occurs in the routing logic, as shown in Figure 2.6b, one of the inputs to the gate is disconnected. When another upset occurs inside one of the LUTs, the AND gate is changed to an XOR gate. The exact function of the original circuit has been lost.

As we see in 2.1 is important to note that not all upsets to configuration bits cause problems. Many digital circuit designs do not fully utilize all of the resources on an FPGA. Upsets to these areas will not necessarily cause critical failures to the user design. Furthermore, radiation upsets are typically not permanent in the sense that no lasting damage to FPGA transistors occurs.

Single Event Transient

A Single Event Transient (SET) is a temporary voltage or current spike at a node of an integrated circuit, which is induced by a single charged particle ionizing a semiconductor material while traversing through or next to an electrically sensitive junction. If the width and amplitude of the pulse is adequate, the glitch can propagate through the circuit [8]. It becomes a Single Event Upset if latched into a storage cell when arriving at data input during clock edge. An SET is a non-destructive and recoverable error, which may develop into a Single Event Upset or Single Event Functional Interrupt.

Single Event Functional Interrupt

A Single Event Functional Interrupt (SEFI) is a non-destructive error that results in the interference of normal operation of a complex digital device or system.

In the case of FPGA devices, a SEFI is often associated with an upset in a control bit or register[9]. It is usually related to a failure in the embedded circuits of memory configuration or re-configuration, or power-on-reset[10]. To restore correct operation after a SEFI, a full reconfiguration or power-up cycle is required.

2.3 General FPGA Upset Mitigation Techniques

Several basic error mitigation techniques have been developed to improve FPGA reliability. Many current FPGA reliability solutions combine several of these mitigation techniques. A summary of the most common techniques is given below however, this summary is not meant to be exhaustive. These upset mitigation techniques are used in several high-speed data transmission and memory systems all across the electronics spectrum. The summaries presented here will describe how some upset mitigation techniques are implemented specifically with regard to Xilinx FPGAs even though some of these are not implemented in our Thesis. The following two classes of design based techniques can be distinguished.

- **Static techniques** rely on the concept of fault masking. They achieve fault tolerance by means of passive redundancy and comparator/voting mechanisms, without requiring any action on the part of the system.
- **Dynamic techniques** achieve fault-tolerance by detecting the existence of faults and performing some action to remove the faulty hardware (or fault) from the system. These techniques rely on using concurrent fault detection, fault location, and fault recovery.

Radiation Hardening

Radiation hardening is commonly achieved by using one of two methods. Radiation-hardened by design (RHBD) is a redundancy process where more transistors are used to

build one SRAM cell. The transistors are placed in a special layout such that the probability that the same ion collides with multiple transistors of the same SRAM cell is very low, making it improbable to cause an upset [11]. The Virtex-5QV (released July 2010) is the most recent Xilinx RHBD FPGA despite the release of three subsequent Xilinx families [31].

Radiation-hardened by process (RHBP) is where the FPGA is fabricated in a way that the transistors are protected from ionizing radiation at the silicon level [12]. Gated resistor hardening is one example of RHBP that uses a variable resistor to increase the threshold voltage required to change the state of the memory cell. Another method is to preserve circuit performance by lowering the threshold voltage when the write enable to the memory cell goes high [11].

Using radiation-hardened parts is a safe option, but the parts are typically much more expensive than commercial off-the-shelf (COTS) parts. If near-perfect reliability is required, then this technique is probably the best option. However, if reliability requirements are more flexible, then the other mechanisms presented in this work are much more cost-effective alternatives.

Error Correction Code

An Error Correction Code (ECC) is a redundancy coding mechanism that is useful for correcting SBUs. ECCs achieve Single-Error Correction and Double-Error Detection (SECDED) on a per-frame basis. Each configuration frame in Xilinx FPGAs contains an ECC word to provide basic SBU correction. Double-bit upsets are detected when using an ECC but are not able to be corrected.

An ECC is implemented in hardware with a dedicated circuit consisting of XOR chains that combine specific bits in a frame to compute a checksum, known as a syndrome[13]. .Every bit location in the frame is accounted for somewhere in the circuit [13]

The ECC by itself does not actually do any upset mitigation. A controller circuit is needed to decode the ECC and run the frame data through the ECC circuit. The result indicates where in the frame an SBU (if any) is located, or if a double-bit upset has occurred. The controller can then use that information to correct the upset(s). A syndrome of all zeros indicates that the ECC was satisfied. A "satisfied" ECC usually means that there are no errors. However, it is quite possible that enough bits in the right places are upset in the frame so as to not be detected by the ECC. The ECC word inside the frame, known as the "golden ECC," is generated to be the exact value that bridges the gap between the bits in the frame and a zero (satisfied) syndrome.

ECCs in Xilinx FPGAs detect all odd-numbered upsets [1]. They also detect all two bit upsets. Even-numbered upsets higher than two may go undetected because a sufficient number of bits are upset in just the right places so as to not be caught by the ECC, but such an instance is quite rare. Section 3.2.1 we analyze how we will use ECCs on Xilinx FPGAs.

Cyclic Redundancy Check

A Cyclic Redundancy Check (CRC) is another redundant coding mechanism that is useful for upset detection. Its primary advantage compared to other coding mechanisms like ECCs is its ability to detect upsets within millions of bits using a single CRC value. Because of the heavy redundancy, changing even one bit of the configuration will result in a vastly different CRC value. Its main disadvantage, however, is that it cannot identify where the errors are specifically located. Thus, it is used primarily as an error detection mechanism. A circuit used to compute a CRC is based off of the remainder of a carefully designed polynomial division circuit. CRCs are added to the configuration memory of FPGAs to provide a means of checking data integrity. One 32-bit value is calculated for the entire configuration memory. Each frame of an FPGA's configuration contributes to this calculation, yet the CRC does not have to expand in bit-length with each subsequent frame (hence the "redundancy").

CRCs are particularly useful in detecting MBUs across an FPGA. Such a feature is invaluable when other codes like ECCs can only detect odd-numbered and specific even numbered upsets . 32-bit CRC on Xilinx devices detects any 31 bit error in a single frame with 100 percent accuracy. Beyond 31 bits, coverage is still very good, but a few cases exist where the MBU would go undetected. With such high error coverage, the probability that a configuration upset goes undetected is very low. CRCs play a significant role in several scrubbing architectures, including the SEM IP internal scrubbing architecture presented by this thesis. Chapter 3.2.1 gives more details about the usage of CRCs in Xilinx FPGAs and in SEM IP Controller.

Triple Modular Redundancy

Triple Modular Redundancy (TMR) is an invaluable upset mitigation technique for improving FPGA reliability. Unlike the techniques presented previously which were built into the hardware of the FPGA, TMR is a technique that designers must incorporate into their circuit. In a design with TMR, the design circuitry is triplicated into three identical copies and typically placed at three different locations in the FPGA. The outputs (and sometimes the inputs) of all of these circuits are connected to a common voter circuit which compares the individual circuit outputs and chooses the majority value. To avoid a single point of failure in the voter circuit, the voters are often triplicated as well.TMR prolongs the reliability of the circuit granting the repair mechanism time to fix the first error before another module is upset and the overall circuit fails.

In Chapter 3.1.1 we introduce further Triple Modular Redundancy as well the implementation of it in our Architecture.

Memory Scrubbing

Unlike the static configuration memory of an FPGA, dynamic memories change constantly with reads and writes by an application. Dynamic memories also incur upsets that

need to be mitigated, but the methods used are somewhat different than those used for static memories.

Memory scrubbing can be performed on memories such as the Block RAMs (BRAMs) of an FPGA. These scrubbers will often use variations of ECCs and/or TMR [14]. For instance, each block of memory may contain an ECC word, and a dedicated memory bank is used to store the golden ECC values. Whenever the block of memory is modified, a new ECC is computed and stored in the memory bank. Another mechanism can then periodically compare the ECC values in each memory to the corresponding value in the bank and correct SBUs if mismatches occur.

TMR could be used by triplicating the memory and reading from each memory periodically to see if a difference in the memories exists. TMR can then use the value agreed upon by the majority vote to then correct the upset in the location that differs [15].

The actual scrubbing of the memory is performed either probabilistically or deterministically [16]. Probabilistic scrubbing is when the blocks of memory are only scrubbed whenever that memory is actually accessed (via a read or write). However, if portions of memory are never accessed, then upsets could build up in those memories. Deterministic scrubbing is a continuous sequential scan that always checks all memories. Deterministic scrubbing is naturally more power-consuming and performance-taxing, yet achieves a better reliability compared to probabilistic scrubbing.

Configuration Scrubbing

Configuration memory is expected to remain static throughout the life of the digital design. Configuration scrubbing involves periodically checking this memory, detecting upsets, then using dynamic partial reconfiguration to overwrite the upsets [17]. The initial configuration must first be complete before any scrubbing can occur. Configuration scrubbing is usually performed at periodic intervals while the FPGA is in operation (hence "dynamic" reconfiguration). This partial reconfiguration is not intended to interrupt normal FPGA operation.

Configuration scrubbing has two primary responsibilities: error detection and correction. A scrubber is a controller that determines when and where to perform a partial reconfiguration for correction. Scrubbers also periodically check the memory to detect upsets. Scrubbers come with a variety of trade-offs, including size, power consumption, performance, robustness, etc. Chapter 3.2 presents several examples of configuration scrubbers.

Recognizing the real threat that radiation poses to FPGAs in high-radiation environments motivates the need for improved upset mitigation techniques. Several basic upset mitigation techniques were highlighted in this chapter that are used in the scrubbing architectures presented in this thesis. The focus of this work is configuration scrubbing upset mitigation mechanisms. To understand how configuration scrubbing is performed, one must first understand the configuration mechanisms of an FPGA analyzed in section 2.1.

Watchdog Timer

A watchdog timer (WDT) is a hardware or software timer that will typically cause a system reset if it is not periodically "kicked" or "fed". Generally, on embedded systems the main programming is in some type of a loop. Each iteration of this loop the WDT is kicked to ensure that the system is not reset. Since it is common for embedded systems to be unmonitored or unmanaged and potentially control critical systems, the developers want to quickly resolve issues with the system. A WDT provides a way to quickly reset the system to try and mitigate issues that may cause the system to hang. It is also common that a backup firmware image or known good state is stored on a flash chip. If a WDT is kicked too many times consecutively, the known good image will be loaded in place of the currently running image. This hopefully resets the system back to a known good state as quickly as possible. It is important to keep this in mind when researching embedded systems while debugging since it is possible to unknowingly switch running firmware.

Chapter 3

Proposed Fault-Tolerant Architectures

As already discussed, commercial SRAM-based FPGA devices offer an abundance of logic resources and functionally predefined components, which make them suitable candidates for processing units in multi-channel data readout systems in high-energy physics experiments. However, as these systems must often be located close to particle detectors, they operate in radiation environments. As described in Chapter 2, the operation of SRAM-based FPGAs, which do not contain radiation-hard by design circuits, can be compromised be radiation-induced effects. This Chapter presents methods and techniques that can be employed to mitigate Event Upsets, and in consequence allow for the utilization of SRAM-bases FPGAs in systems operating radiation environments. In this Chapter also the methods and techniques that are analyzed are incorporate in our design to implement various Fault Tolerance Architectures.

3.1 Mitigation Methods

An FPGA design without radiation mitigation methods employed, when implemented in an SRAM-based FPGA operating in radiation environment, will have a substantial soft error rate disrupting its correct work and may not satisfy the requirements of some critical application. However, if radiation mitigation methods based on logical masking and correction circuits are incorporated, then the functional error rate resulting from operation in radiation environment can be decreases and a commercial SRAM-based FPGA may be candidate for system use. Various mitigation methods have both advantages and penalties, so the best way to protect digital circuits implemented in SRAM-based FPGAs is to use a combination of complementary mitigation techniques.In this Experimental results showing benefits of different radiation methods are presented and discussed in detail in Chapter 4.

3.1.1 Spatial Redundancy

Spatial redundancy is a mitigation method based on replicating sensitive functional design modules and comparing their outputs together in order to detect discrepancies. This mitigation method is not intended to correct soft errors, such as SEUs in the configuration memory of the SRAM-based FPGAs. Instead it is used to:

- detect occurrencies of soft errors modifying the expected operation of the circuit
- mask soft errors and their propagation through the circuit.

Spatial redundancy incorporates additional design module instances withe other auxiliary components like comparators or voters. As the mismatch between results outputted by different module replicas is detected by either a comparator or voter, they become a critical part of these architectures.

Dual Modular Redundancy

The Dual Modular Redundancy (DMR) scheme, presented in 3.1, can be applied both to combinational and sequential logic. In the DMR two identical design blocks operate in parallel and the comparator detects faults but does not correct them, the DMR is often called a fail-stop architecture. When both results are identical, then the comparator does not report any error. Conversely, if the outputted results are different, then the comparator asserts an error signal and it is up to the designer to foresee a circuit response to such an error. The potential solutions are either to mark the result as faulty and continue operation, or stop an repeat.

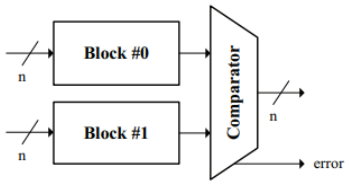


Figure 3.1: Dual Modularar Redundancy(DMR) architecture

Triple Modular Redundancy

Triple Modular Redundancy is a widely used hardware fault tolerance methodology to protect against radiation-induced Signle Event Upset. In the TMR, a base design block is replicated three times and a majority voters votes for the correct output. In commercial-grade FPGAs a user can implement TMR using proper HDL instantions. Similarly as in the DMR, the TMR does not correct soft errors. Instead it masks them. The various types of TMR techniques are differentiated by the portions of circuity that are replicated and where the voters are placed. There are several TMR schemes:

- NO-TMR, shown in 3.2, a basic block is implemented without TMR. No additional logic is added to the design pertaining to SEU mititgation.

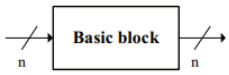


Figure 3.2: Basic block implemented without TMR

- Local-TMR (LTMR) shown in 3.3, is a scheme where only flip-flops are triplicated and data-paths stay singular. Voters are placed after flip-flops Combinational logic paths, clock reset lines are shared, and they are single sources of failure. With this mitigation scheme only effects of flip-flop SEUs are reduced, as they are masked. SETs in data-paths of combinational logic can be captured by endpoint flip-flops.

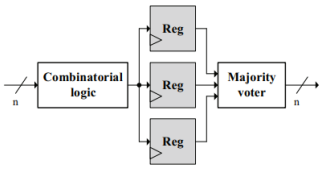


Figure 3.3: Local TMR architecture

- Block-TMR show in 3.4, is a scheme where complex functions containing combinational logic or flip-flops are triplicated, and majority voters are placed at the output of the triplets. In the BTMR it is not required that the inputs come from a common source However, if the I/Os are not fanned out to the replicated blocks from common source, then voting will be unreliable due to synchronization issues. The outputs of the replicated block are voted and the voter can have an additional warning output that informs the surrounding circuitry that not all the blocks output identical data. The BTMR only provides masking capability and does not correct soft errors. Thus this techniques is only practical for design modules that can be regularly reset. In this case soft errors (SETs in data-paths and SEUS in flip-flops) are regularly flushed, and the design blocks can be forced to reach a deterministic state.

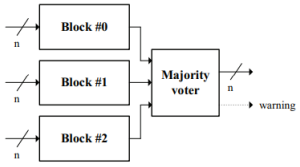


Figure 3.4: Block Triple Modular Redundancy architecture.

- Distributed-TMR(DTMR), shown in 3.5 and 3.6 is a scheme where all the functional logic is triplicated except for global routes:clock,resets and high-fanout enables. This scheme reduces data paths upsets, but since the global routes are not mitigated,

SETs on global routes can still disrupt the system. In the DTMR voters are placed after the flip-flops.

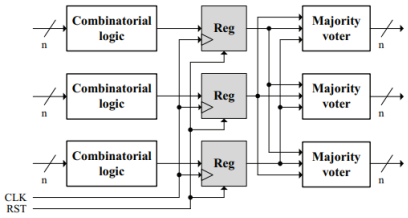


Figure 3.5: Distributed Triple Modular Redundancy architecture.

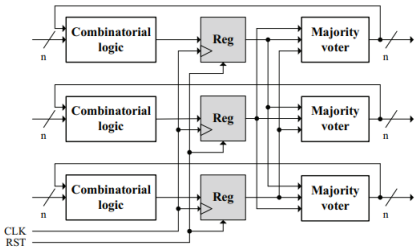


Figure 3.6: Architecture of Distributed Triple Modular Redundancy with feedback paths.

- Global-TMR(GTMR), in 3.7, is a scheme where the entire design is triplicated including all global routes:clock,resets and high-fanout enables. The GTMR reduces data path upsets. However,modern FPGA devices employ additional logic outside of the data-paths that cannot be mitigated. Incorporating the GTMR reduces the upset rate,but still has some points of failure.

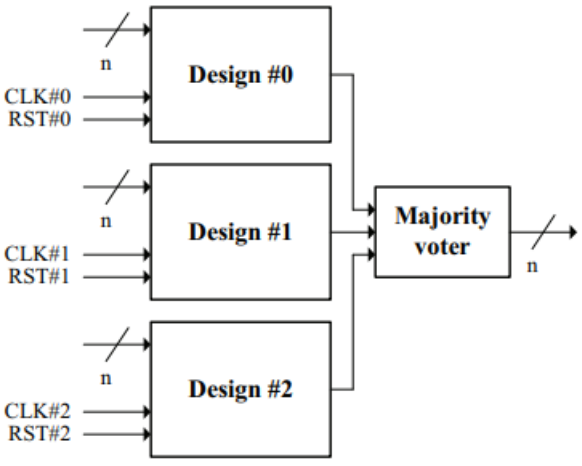


Figure 3.7: Global Triple Modular Redundancy architecture.

As the TMR only masks soft errors and does not correct them, the affected circuit replicas may remain faulty or unsynchronized with each other. For some application,base TMR

implementation does not provide enough mitigation. Thus, to further increase the TMR performance, it can be implemented with various synchronization mechanisms and configuration reconfiguration. The TMR implementation is not straightforward task, and it is improper insertion can jeopardize the system radiation performance Placement of the voters can directly affect the radiation susceptibility of the TMR design.

Proposed architecture based on TMR mitigation techniques

In this section the proposed architecture to resolve any SEU that appear in our design is to replicate redundant instances of an entire module and vote the final outputs of the modules(Block-TMR). TMR is implemented in any design can be triplicated. Therefore in our design due to limitation in resources we can't triplicate the ARM-processors (PS). In our case a module represent the desire application (FIR) that will take place in the PL configuration PL block will be triplicated. This is a very effective means of SEU mitigation that is easy to implement and can be performed entirely within a single device as long as the module does not utilize more than a third of the total device. In order to incorporate TMR in our design we have implemented two key components. The input and the output majority Voters. As it shown 3.8 the first PS part will send 3 sets of data to PL part.

The input buffer receives the input data, and the Input IP Voter will not distribute different data to each FIR. Instead, one set of the same data is divided into three different FIR for calculations and every cycle will calculate 3 results. The output voter will conduct a comparison between the three output result and send to PS the faulty module alongside the three output results. The voter located on the PS side will again compare these three data sets and select the result that was calculated identically at least two times. Any difference between the PL and PS voter will be an indication that output voter has been compromised. At the same time, if a SEU happened in the calculation process, it will also capture and record the FIR that caused the error. PS will send the new data into the data buffer, and the three FIR will perform the same operation in every cycle. After completing the calculations of the entire application, the PS side will vote to select the final result.

Also by phase-shifting the three clock lines, one can achieve better immunity against SET, because the data are latched at different times, so that a short spike would possibly affect only one of the three modules. In order to archive that the Output voter will not calculate the results in every cycle but in the simultaneous assertion of the Valid_out signal. In spatial TMR, by using three different FIR to perform the same operation for the same data, the FIR is multiplexed from the perspective of space and time, thus achieving an effect similar to TMR.

The disadvantage of module level mitigation techniques is that they do not provide a simple and robust recovery mechanism after an error has been detected in one of the modules. In random logic with sequential elements, it is not ensured that the error will be detected until it manifests itself on the output of the module where it is compared

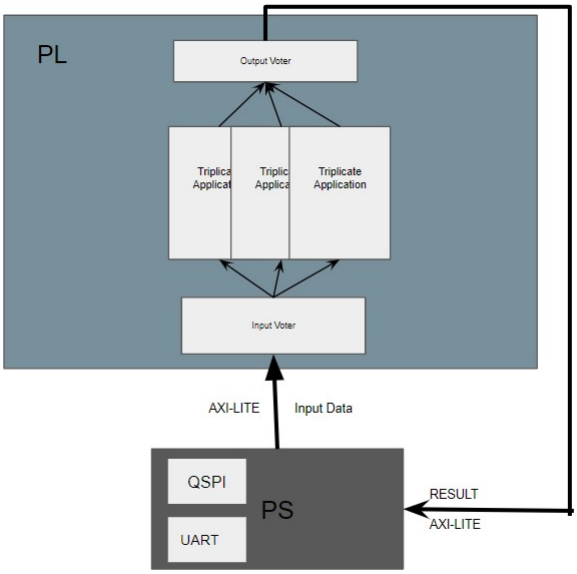


Figure 3.8: Architecture of TMR.

with the outputs of the redundant modules. The internal state of the erroneous module can at that stage be very much different from the state of the redundant modules. Any further execution will be meaningless since the erroneous state will not be automatically recovered from. The probable consequence is that the application has to be reset or some other means of action has to be taken to resynchronize the modules. This will lead to loss of data and operational down time.

3.2 Configuration memory scrubbing

Spatial redundancy methods, described in Section 3.1.1, only mask soft errors and do not correct SUEs in the configuration memory of the SRAM-based FPGAs. In order to keep spatial redundancy methods working reliably, they must often be complemented with an additional mitigation method called scrubbing which avoids fault accumulation and restores the correct configuration memory content. Scrubbing is a mitigation method that relies on periodic cycles of simultaneous writing to the configuration memory with the intent of correction bit errors , while the device's functional logic area is still operating. Scrubbing prevents the accumulation of configuration upsets and significantly reduces the probability that two accumulation of configuration upsets and significantly reduces the probability that two SEUs may overcome TMR. It is possible because modern SRAM-based FPGAs offer a functionality called Partial reconfiguration, where part of the design

can be independently reconfigured. An example of the configuration Interfaces and circuitry inside the Xilinx 7 Series FPGAs is presented in Fig. 3.9. He configuration module manages the configuration memory consisting of frames, which are the smallest addressable elements. The configuration memory could be accesses and modified via PCAP, JTAG or ICAP interfaces. As explained in detail in [15], apart from the configuration memory scrubbing, the content of the BRAM can be also scrubbed. Scrubbers' classification is

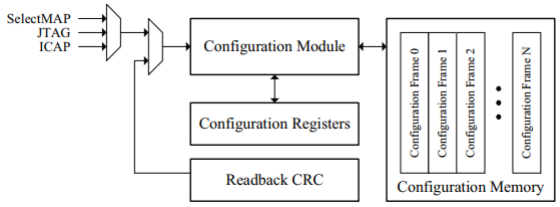


Figure 3.9: Configuration interfaces and circuitry.

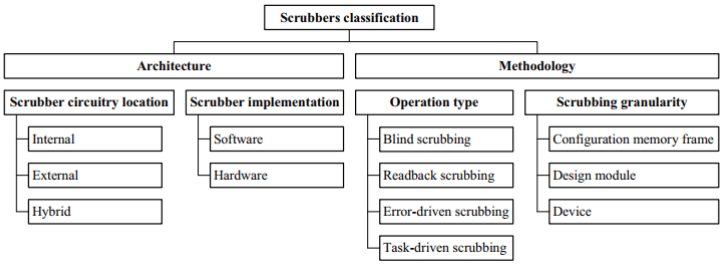


Figure 3.10: Configuration interfaces and circuitry.

presented in Fig3.10 and it is explained in detail in [39, 17, 40]. They can be categorized depending on the scrubbing system architecture or adopted scrubbing methodology. Depending on the selected scrubbing architecture, there are internal, external, and hybrid configuration memory scrubbers. They can be implemented either in software and run on a processor, or be instantiated in a programmable logic and operate in an FPGA. However, depending on the operation type that the scrubber executes, there are blind, readback, error and task driven scrubbers. The blind scrubber reads golden configuration data from an external non-volatile memory and writes it back to the configuration memory of the SRAm-based FPGA. In this method it is impossible to gather information about how many SEUs were induced to the configuration memory. On the other hand, the Redback scrubber first reads back the configuration frame data from both SRAM-based FPGA and the external non-volatile memory. Next, the scrubber controller compares the data using either a direct comparison or Error Correction Codes (ECCs). If a discrepancy is found, the faulty configuration frame of the SRAM-based FPGA is a rewritten using golden data stores in the external memory.

While using the Redback scrubber, it is possible to count the number of induced SEUS and get valuable information about the radiation environment where the target application operates. Also, it allows for verifying simulated radiation environment parameters with

the real ones. Depending on the the scrubbing granularity, scrubbers operate on a different sizes of memory blocks. In a single run, they can process either only one configuration frame, one design module, or the full device.

3.2.1 Internal Scrubber

The scrubbing controller in the internal configuration memory scrubber is implemented either as an on-chip hard core or is created out of user fabric blocks located inside the target SRAM-based FPGA. The internal scrubber architecture is presented in Fig3.11. The scrubbing controller is connected to the internal configuration memory of the SRAM-based FPGA via a dedicated configuration interface. It can also optionally interface with an external non-volatile memory holding the golden copy of the configuration data. Usually, internal scrubbers use Single Error Correct Double Error Detect(SEC-DED) technique together with the ECC codes embedded in the configuration frames to detect and correct SEUs, and the detect MBUs. The configuration containing MBUs are rewritten by fetching correct data from the external memory holding the golden bitstream.

An example of the internal scrubber that can be instantiated in modern Xilinx SRAM-based FPGAs is the Soft Error Mitigation Controller which delivers functionality of correcting both SEUS and MBUs [1]. There also exist other implementations of internal scrubbers. One example is the internal scrubber based on a coarse grain TMR FPGA design. Although each TMR branch is implemented in a different FPGA region it uses the same configuration data. Thus any configuration frame of the TMR branch can be repaired using the data from the other identical branches. Also, as demonstrated in [41], the internal scrubber can be implemented in a system that requires Partial Reconfiguration Functionality.

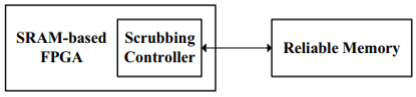


Figure 3.11: Architecture of Internal Scrubber.

Proposed Architecture based on SEM IP

Xilinx created the SEM IP as an upset mitigation mechanism for terrestrial applications. The SEM IP utilizes the internal Readback CRC hardware to perform error detection using the built-in ECC and CRC codes [1]. The SEM IP, however, performs its own correction (as opposed to letting the Readback CRC perform the correction) by means of the ICAP. This correction includes capabilities for MBU correction from a golden data. SEM IP controller has different levels of customization depending on the needs of the user design. It includes the ability to turn off or on many features to conserve potentially

limited resources like power, area utilization, etc. One of these features is a classification component which can determine whether an upset bit was an essential bit or not. It uses the generated essential bits file to accomplish this task. The classification of errors allows the SEM IP to avoid resorting to potentially disruptive recovery procedures for bits that will not affect proper design function.

SEM IP Controller also offers different modes of error correction. Repair mode is essentially the equivalent of the Readback CRC which handles all SBU correction, but cannot correct MBUs. Enhanced Repair mode adds individual frame EECs to support SBU and double-bit adjacent error correction, but cannot correct other MBUs. Replace mode is essentially the same as blind scrubbing in that several frames are written from the contents of golden data stored in a large memory. Replace mode can correct all larger MBUs.

Detection for the SEM IP is accomplished by means of the Readback CRC. The detection speed depends on the number of frames of the FPGA and the clock frequency of the Readback CRC. For instance, Time Detection for the XZC07010 FPGA with a Readback CRC frequency of 100 MHz is 4.02 ms. The SEM IP's correction speeds are given in [1] on page 20. Correction Time for SBUs is 610 µs using Repair or Enhanced Repair correction modes in frequency of 100 MHz. Two-bit adjacent errors have a Time correction of 18.79 ms when using the Enhanced Repair mode. The Replace correction mode corrects all errors from golden data, thus all error types have a correction Time of 890 μs. While the SEM IP circuit is fairly complex (see Figure 3.12), it is also relatively small in its resource utilization (see Table 4.14 for a breakdown of SEM IP resource utilization on a Zynq FPGA). All of these internal components are, however, just as susceptible to the same radiation effects as the rest of the FPGA.

In order to successfully incorporate SEM IP in our design, the below interfaces must implemented or imported into the architecture as shown in Figure 3.14.

- Uart offers communication between the SEM IP Controller and the processor. Uart uses MON shim component of SEM IP (see [1]) which translate RS-232 to UART and then AXI-streaming to communicate with PS ARM-processors. A translation from RS-232 to AXI-streaming seem more sensible than RS-232 to UART and then from UART to AXI-streaming. This interface offers a transmitter and receiver channel. These channels operate independently from each other. The transmit channel is used to send reports about what the SEM IP is doing. These reports include the information of the Status Interface except the heartbeat and more. And since it is in ASCII format and therefore human readable it seems interesting to log this information. The receive channel can be used for the commands. It requires ASCII characters as commands. If an invalid character or character sequence reaches this interface, that character or sequence will simply be ignored by the SEM IP and it will wait for the next command attempt

- Fetch Interface is a mechanism that allows the SEM IP to fetch data from an external

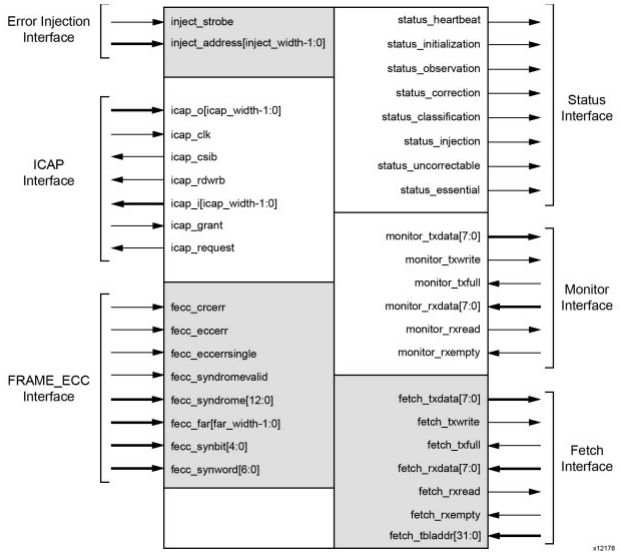


Figure 3.12: SEM IP Controller Ports [1].

memory. This data being the Reference Data or the Essential Bit Mask. So this interface is only used when using the Essential Bit Mask and/or the Replace method to repair. This interface uses the RS-232 protocol for transport. It communicates in binary data. This interface can also be split into two independent channels, one to send requests and one to receive data. The first channel will send a request for certain data. This request contains a 4 byte address and a 2 byte data length. After this request has been sent the SEM IP will wait until the exact amount of data that was requested is received at the receiving side of the interface. The data that can be requested can be split into two types of data. The first type is Reference Data contain the data that should be in the configuration memory. Meaning fetching this data and writing it back can repair any amount of bit errors in a configuration frame. The second type is an Essential Bit Mask indicating what bits in a design

are essential for correct operation and which are not. Meaning that the SEM IP can differentiate between errors that will influence the functional design or not. And therefore it is possible to know if action is required.

- ICAP is the physical connection to to configuration memory content. The SEM IP uses this to read back configuration memory data to check on errors. After this it will write this data back with correction if needed and possible. This interface will need to be shared if a hybrid architecture is implemented. In the Xilinx application [42] it is explained how the ICAP should be shared between the SEM IP and another ICAP unit that performs Partial Reconfiguration

- FRAME ECC is the physical connection to the memory in which the configuration memory checksum is stored. This data is read back at the same time a configuration memory data frame is read. FPGA Frame ECC logic detects single or double bit errors in configuration frame data. It uses a 13-bit Hamming code parity value that is calculated based on the frame data generated by BitGen. After locating the error, it provides ICAP with the correct frame in Enhanced Repair and Repair mode, while in Replace mode it updates the Fetch interface to start retrieving the gold data.

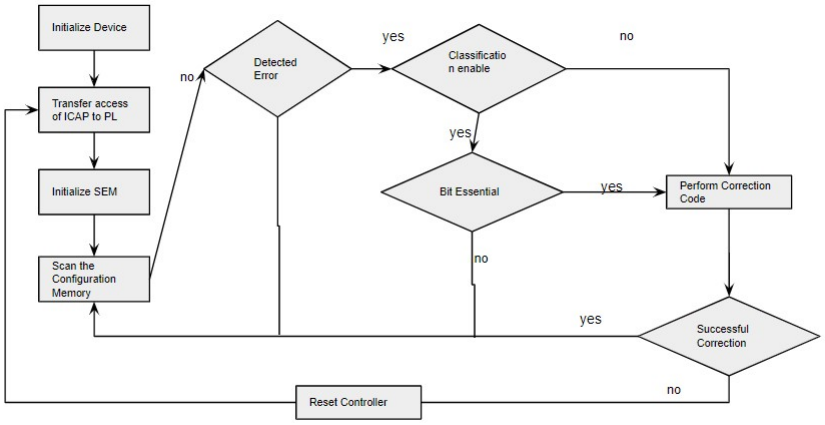


Figure 3.13: Architecture flow of SEM IP as Internal Scrubber.

When those are implemented SEM controller will function as its supposed as refers to [1]. After an error has occur SEM will follow the bellow procedure 3.13 to fix it .So

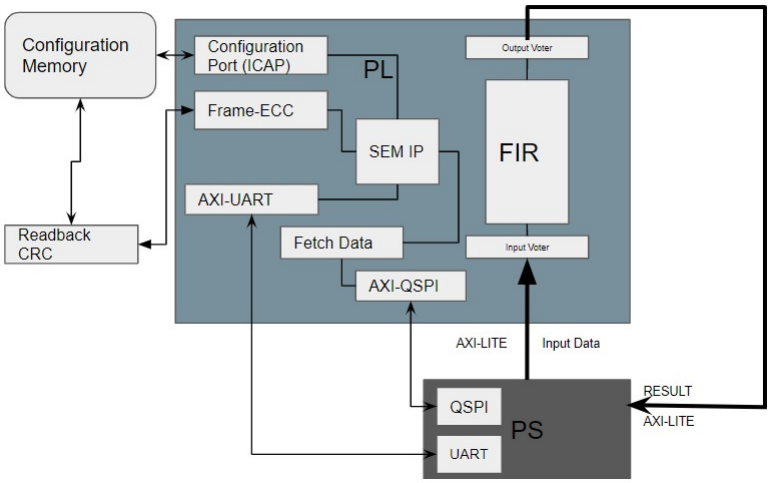


Figure 3.14: Architecture of SEM IP as Internal Scrubber.

depending on the customization that has been made in respect to the correction mode SEM IP in the occurrence of an essential error will try to reconfigure the Configuration Memory by writing the correct Frame of Frames through ICAP.

3.2.2 External Scrubber

The scrubbing controller in the external scrubber is located outside the target SRAM-based FPGA. External scrubber architecture is presented in FIG 3.15. External scrubbers are created out of user fabric blocks located inside the anti fuse or flash-based FPGAs [43], or are implemented as standalone radiation-hardened ASICs [44]. As demonstrated, the external scrubbers have better performance than the Internal ones [45]. External scrubbers usually operate in either blind or readback mode. An example of blind scrubber implementation in a high-energy physics experiment is the CBM Readout Controller board. It is equipped with the XIlinx Virtec-4-SRAM-based Fpga [46] which operates as the core data processing device. It is continuously scrubbed by a flash based MicroSEMi ProASIC3 A3P125 FPGA, which reads golden configuration data from an on-board,non-volatile, flash memory. Another example of blind scrubber implementation is the Readout Board of ALICE Inner Tracking System detector. Its architecture is presented in 3.16. The Xilinx Ultrascale XCKU060 SRAM-based Fpga is continuously scrubber by the flash-based MicroSEMi ProASIC3 A3PE600L[47].

Although the architecture of the readback scrubbers is more complicated than the design of the blind scrubbers, they are also implemented because they provide useful statistics on the number of induced SEUs. An example of the rradback scrubber implementation is the Readout Control Unit [48] utilized within the TPC detector at the ALICE experiment. The configuration memory of the SRAM-based Xilinx Virtex-II is monitored refreshed by MicroSEMi PROASIC APA075.

External flash memories utilized within scrubbing architectures are susceptible to radi-

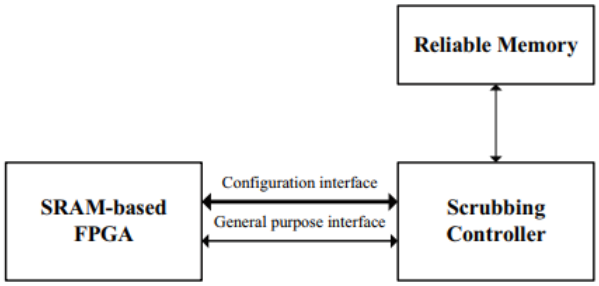


Figure 3.15: Architecture of External scrubber.

ation and they increase the complexity of the system. Because of this, external scrubbers that do not need access to the golden bitstream are usually implemented, the first example is the external scrubber for the system with triplicated SRAM-based FPGAs. The external scrubber continuously compares the configuration memories of the three FPGAs, and if a discrepancy is found, then the faulty configuration frames of the affected device are reconfigured using the data retrieved from other FPGAs. Another example of the external scrubber is a solution based on the Error Detection and Correction mechanism, which does not require an access to the golden configuration data. Yet another example is the JTAG Configuration Manager (JCM), which is the external scrubber that can operate as both blind and readback scrubber[49].

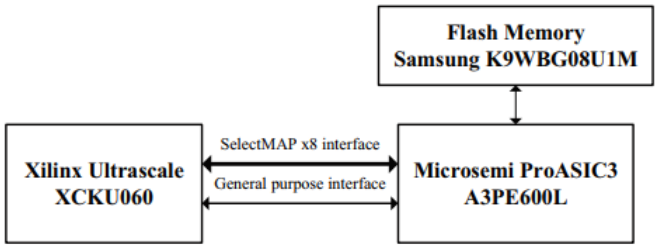


Figure 3.16: Scrubber architecture implementation in Readout Board of Its Readout System.

3.2.3 Partial Reconfiguration

Some times what is needed in a a small section of a design needs to be modified, a full reconfiguration is actually unnecessary. In fact, a full reconfiguration is quite disruptive because it resets all of the logic state of memory resources and restarts the device completely. Partial reconfiguration is the ability to reprogram selected configuration frames while permitting the remaining resources in other frames to continue to operate without interruption. Partial reconfiguration strictly targets a few frames in the configuration and generates a small partial bitstream to reprogram only those frames. The past several generations of Xilinx FPGAs support partial reconfiguration. Partial bitstreams serve as the primary model for constructing scrubbing command sequences. Scrubbing operations are

performed using many of the same commands as full and partial bitstreams. In addition, the ability of a scrubber to support partial reconfiguration (i.e., to not detect the changes of partial reconfiguration as errors) is a desirable feature.

The Partial Reconfiguration technology allows us to reconfigure a single module of our application without affecting the rest of the system. To illustrate why this is so important, let us Dynamic Partial Reconfiguration features open a variety of designing systems exploiting this method, if for example run-time systems change their behavior on-demand initiated by processing input of data from external source. New approach using this feature makes it possible to use FPGAs with smaller configuration memory and consequently smaller chip size by storing configuration data on external memory. Thus it is possible to save cost and reduce power consumption because it does not actually use new modules of a system and do not allocate configuration memory and corresponding power consuming hardware. Nevertheless power dissipation during reconfiguration has to be considered. The next sections of the paper mainly concentrate on the Partial Reconfiguration of the FPGA and how to use it as a correction mitigation technique and not how it is usually used to.

Proposed Architecture based on Partial Reconfiguration

Partial in contrast with Full Reconfiguration is re-configuring only the part of FPGA. Also the full requires more time to get reconfigured. In the partial reconfiguration FPGA is divided in two regions one is static part which will never change in reconfiguration and the second is dynamic part which will change as and when require. This dynamic part can be reconfigured by two ways:

- Static Partial Reconfiguration
- Dynamic Partial Reconfiguration

In Static Partial Reconfiguration the system will stop working at the time of reconfiguration and the part which is required to be reconfigured will only be programmed again. While in the Dynamic Partial Reconfiguration the system will keep on running when the dynamic part is getting reconfigured. In Dynamic Partial Reconfiguration, while run-time, different configurations were sent via the configuration access port to the configuration memory. Both logic elements and routing resources can be influenced and adapted to a new functionality and routing. It becomes clear that changes influence the behaviour of functions in the neighbourhood if signal lines cross the area where partial reconfiguration occurs. Systems implementing partial self-reconfiguration are still uncommon in most FPGA applications due to the relatively large overhead required for the initial analysis phase to find a working tool flow. Xilinx has added more documentation to their website and has produced a more user-friendly floor planning tool used for partial reconfiguration, but much of this information is currently limited to an Early Access area that requires approval for access.

In this section dynamic partial reconfiguring system is implemented by making use of the Internal Configuration Access Port (ICAP). A partial bitstream is written to the ICAP, which then reconfigures the specified portions of the FPGA with the new logic. Communication with the ICAP can be implemented through soft core processor or through a custom VHDL logic design. Using the custom logic or by writing the custom routine in the soft core processor we can frequently reconfigure the module of the design which is very critical and we can mitigate or remove the accumulation of single bit-flip error which we called as SEU.

In this thesis a partial reconfiguration is performed to restore an area or module to a safe state 3.17. Where safe state is the proper behavior of a design specifications. This mitigation technique is not very effective in SEUs or MEUs because it does not have an error detection mechanism when it occurs. Nevertheless this mitigation technique when it combined with a detection mechanism is very effective. Below are analysed how to perform dynamic partial reconfiguration in the module or the application.

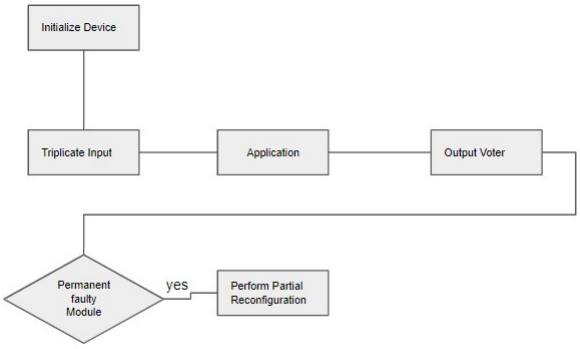


Figure 3.17: Architecture flow of Dynamic Partial Reconfiguration.

- **JTAG**: The Joint-Test Action Group (JTAG) interface is a common, easy-to-use, serial interface that supports the IEEE 1149.1 standard for Test Access Port and boundary-scan architectures. Configuration commands through the JTAG interface must be wrapped in special JTAG command headers and footers to sequence through the JTAG state machine protocol, including access to the various JTAG registers.

One of the main advantages of this interface is that the JTAG hardware ports are already built into almost all Xilinx FPGA boards, so using JTAG only requires the use of those four pins [34]. JTAG is also the highest priority configuration interface [50], and will always be able to access the configuration module regardless of the configuration mode pins. This interface can program multiple devices, but must do so in a serial chain, (not in parallel like SelectMAP). This interface is typically accessed from an external source, with clock rates up to 66 MHz [51].

- **ICAP**: Another option for performing configuration operations is to program the device from within the FPGA itself. The Internal Configuration Access Port (ICAP) can only be accessed from a user design via primitive instantiation (see 3.18). The ICAP allows a user design to reconfigure the device (either partially or fully), after the initial configuration has been completed. It is not able to perform the initial configuration, however. The ICAP is primarily used for partial reconfiguration. One of its primary advantages is its configuration speed, an operating frequency of 100 MHz [34].

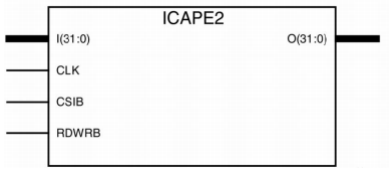


Figure 3.18: ICAPE2 Configuration Interface.

- **PCAP** : The Processor Configuration Access Port (PCAP) is a unique interface that enables access from a hard processor to the configuration module. The PCAP is only found on the Zynq-7000 family. It is the bridge between the Zynq's dual-core ARM processing system and the FPGA configuration memory. One of its most important features is that it allows software programs running on the processors to access the configuration module at runtime via configuration commands written in software. The PCAP clock can run at frequencies as high as 500 MHz, though it usually runs at no higher than 100 MHz for most applications[52].

In this thesis we have implement two architectures of partial reconfiguration. The first one is a customize ICAP interface in which the partial bitstream are stored into internal FPGA's BRAM memory 3.19. The difference between these two architectures is that in the case of custom ICAP, the whole process of retrieving the partial bitstream stored in Block RAM and delivering it to the ICAP interface, is not carried out by a software application, but from a hardware module developed from scratch in RTL to implement a custom state machine in the FPGA fabric. ICAPE2 primitive is automatically instantiated by this custom ICAP controller, which is able to connect directly (without any processor bus) the output port of the BRAM memory to the input port of the ICAP primitive.

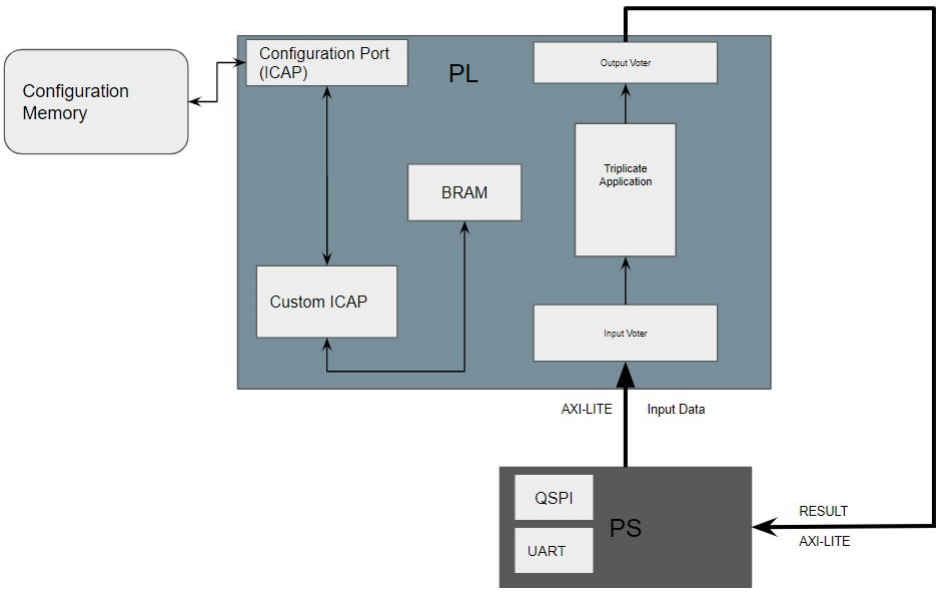


Figure 3.19: Architecture of Dynamic Partial reconfiguration.

In the second architecture partial bitstreams are delivered to the ICAP interface of the FPGA device, through the Xilinx's AXI HWICAP IP core [18]. To do show we have use the ARM-Processor and the AXI lite protocols. AXI-HWICAP is connected to an embedded soft-core ARM-processor which is also implemented in the FPGA fabric to control the operation of all IP cores of our system. More specifically, a software application running on ARM-Processor is responsible for reading bitstream configuration data from the the QSPI flash memory of the chip and delivering them to the ICAP interface through the AXI HWICAP core as it is shown in 3.20. The files are stored in QSPI-flash and then transfer to DDR because flash memory is widely used in all space applications, and currently there is no readily available radiation-immune alternative.

3.2.4 Hybrid Scrubber

In this thesis we had not implemented a hybrid scrubber as a technique but it was necessary to point out for future work. The hybrid scrubber, whose architecture is presented in Fig. 3.21,has got two controllers. The internal controller is usually implemented as an on-chip hard core inside the target SRAM-based FPGA. It provided the SEC-DED functionality, and using the ECC codes embedded in each configuration frame, it corrects SEUs and detect MBUs. When the internal controller finds a configuration frame containing an MBUa, its address is sent to the external controller. It fetched the correct

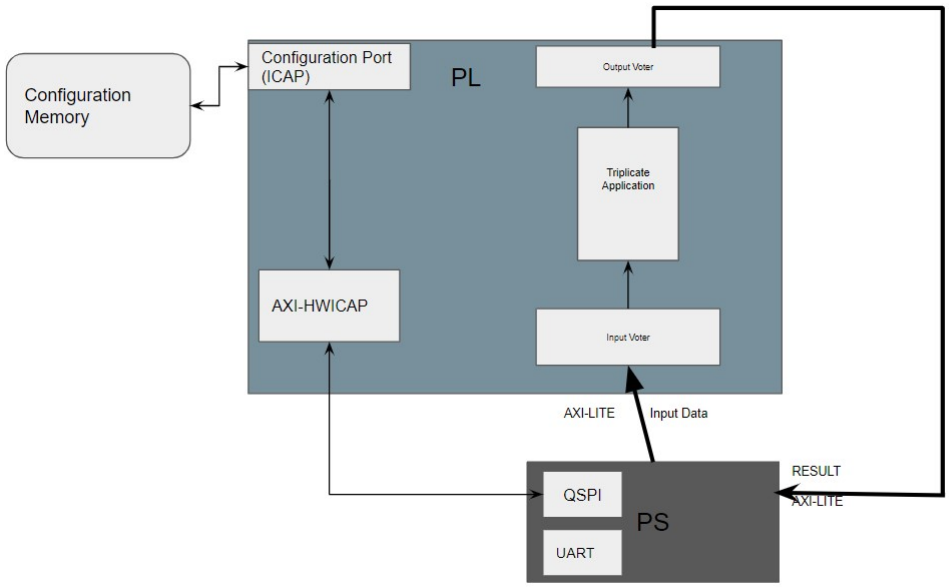


Figure 3.20: Architecture of Dynamic Partial reconfiguration.

configuration frame form an external memory and rewrites the faulty data frame using either JTAG, SelectMap, or PCAP interfaces[49]. The external controller of the Hybrid scrubber is usually:

- implemented with fabric blocks located inside the antifuse or flash-based FPGAs.
- is implemented in the processor of a System-on-a-chip[53] or in the external System-on-a-chip.
- is realized as a dedicated software running on a PC[54].

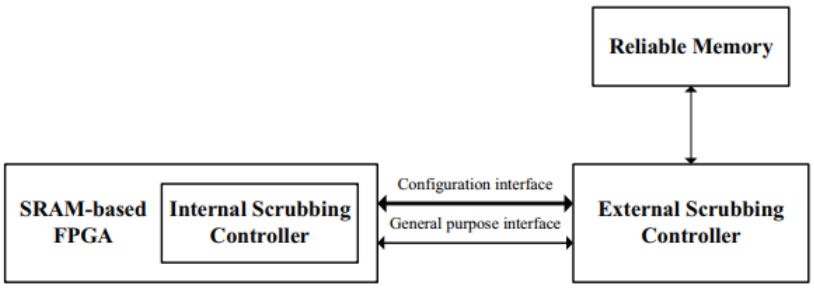


Figure 3.21: Architecture of Hybrid scrubber.

An example of the hybrid scrubber implementation in a high-energy physics experiment

is the readout board of the TOF detector at CBM experiment[55]. The readout board is equipped with the SRAM-based XIlinx Artix-7 Fpga. The Fpga employs the Xilinx SEM [1], which operates as the internal controller of the hybrid scrubber, the controller corrects SEUs and detects MBUs in the configuration memory. While the MBUs are repaired by the dedicated software, which interfaces with the JTAg port of the SRAM-based Fpga via the radiation-tolerant GBT-SCA adapter[56]. The external controller of the hybrid scrubber is implemented in software, which runs in a PC located in the radiation free area.

3.3 Hybrid mitigation techniques

This section covers the different architectures we have implemented in order to develop a more robust system than the ordinary fault tolerance techniques. In this attempts we encounter many problems and errors like synchronization of all modules, simultaneous communication through modules and user as well as some resources limitations(as the existence of one ICAP, a small amount of BRAMS and a 100Hz limitation in SEM controller). Despite those problems we were able to integrated all the mitigation techniques in different architectures.

3.3.1 Internal Scubber and TMR

In the previous chapter TMR was chosen because it can handle faults in the configuration memory, block memory as well as SETs. It is widely used for these reasons and if it is combined with scrubbing it is possible to correct errors without the need for pausing the design. The advantage with this method is that it increases the reliability during a finite time-period. Also we have seen that Scrubbing is a mitigation method that relies on periodic cycles of simultaneous writing to the configuration memory(CRAM) with the intent of correction bit errors, while the device's functional logic area is still operating. Scrubbing prevents the accumulation of configuration upsets and significantly reduces the probability that two SEUs may overcome TMR.

As indicated above the advantages of TMR and Internal scrubbing techniques we implement an architecture that combine both and provide better results which leads to a faster correction and detection system as well as a better Reliability in our design.

Thus to implement this combine Architecture we have implemented an Architecture which is base on TMR and SEM IP as Internal Scrubbing. In order to implement TMR as described above in TMR 3.1.1 the desire application (FIR) that will take place in the PL configuration PL block is triplicated. Also as discussed above we have implemented two key components. The input and the output majority Voters. As it shown 3.22 the first PS part will send 3 sets of data to PL part.

The input buffer receives the input data, and the Input IP Voter will not distribute different data to each FIR. Instead, one set of the same data divided into three different

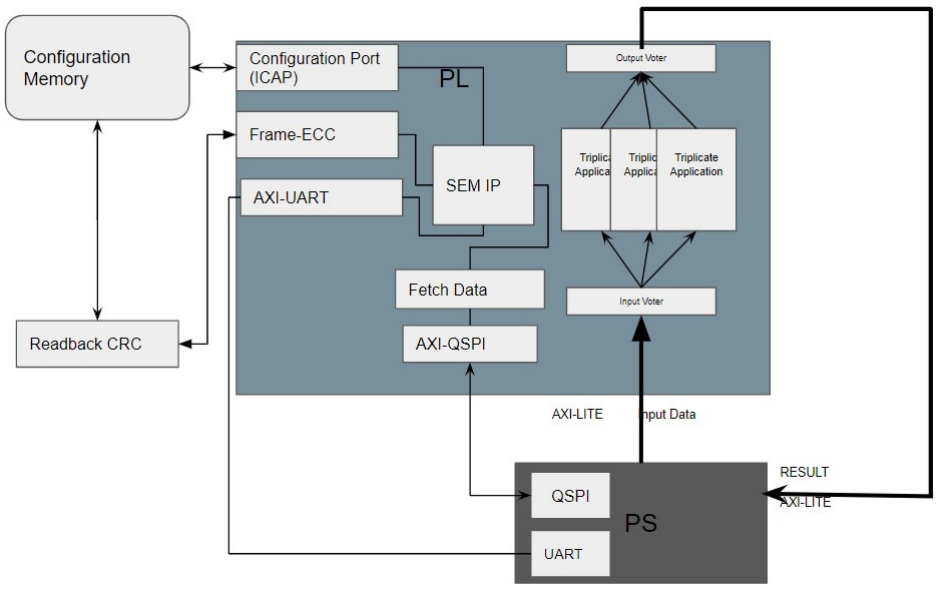


Figure 3.22: Architecture of Internal Scrubber and TMR.

FIR for calculations and every cycle will calculate 3 results that will be send back to PS part. The voter located on the PS side will compare these three data sets and select the result that was calculated identically at least two times. At the same time, if a SEU happened in the calculation process, it will also capture and record the FIR that caused the error. After that, the PS will send the new data into the data buffer, and the three FIR will perform the same operation in turn. After completing the calculations of the entire application, the PS side will vote to select the final result.

Simultaneously we have SEM Controller utilizes the internal Readback CRC hardware to perform error detection using the built-in ECC and CRC codes [1]. While The SEM IP, however, performs its own correction by means of the ICAP. We have implemented SEM ip in replace mode as described 3.2.1 using QSPI-flash to store the golden copy and also AXI-UART for communication purposes. The Figures 3.22 and 3.23 below shows the architecture described and the behaviour of the system in the occurrence of an Error.

In conclusion Scrubbing and TMR should be considered as complementary methods for the mitigation of radiation effects. While TMR makes the device immune to single upsets of control and data processing block, scrubbing heals flipped bits in the configuration memory. It thus avoids accumulation of upsets which can lead to malfunction even if the design is TMR. In simple words, TMR protects against a few SEUs in the user or configuration memory, while scrubbing keeps the number of upset as low as possible.

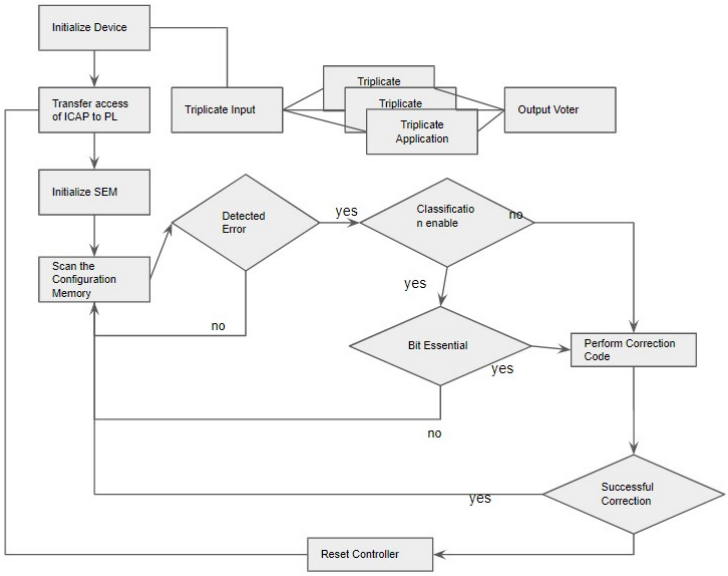


Figure 3.23: Architecture flow of Internal Scrubber and TMR.

3.3.2 TMR Combine with Partial Reconfiguration

Scrubbing is the most commonly technique used for mitigating upsets in FPGA. The major problem with scrubbing is the detection time. There is a certain amount of time between the occurrence of an error in the configuration memory and discovering it then repairing it. This time interval can be in the order of milliseconds before the correct state of the configuration memory (CRAM) bits is restored. The system may produce an erroneous result, during this time interval, and this is not acceptable in safety critical modules in automotive systems therefore, scrubbing cannot be used to recover from SEUs or MEUs.

Triple modular redundancy (TMR) is a potential solution to mitigate those upsets in FPGA. As described above in TMR targeted module (M) is triplicated, and a majority voter circuit produces the correct output. This Architecture as described has a main disadvantage and that is the absence of a repair mechanism when a permanent error has been detected into one of the modules. TMR produces an incorrect output if the majority voter circuit fails, i.e., the detection time interval is not a problem with TMR. Hence, TMR can only detect and correct a single module failure at a time. The problem with TMR is the occurrence of non-adjacent double event upsets (DEUs) affecting two modules, which produces a system failure. TMR solves DEU problems if they occur in the same module but produces an erroneous output when two memory cells of two modules are simultaneously

affected. So in order to successfully incorporate the TMR in our design a repair mechanism of the faulty modules must be introduced. Dynamic partial reconfiguration (DPR) can be used to recover the failed module

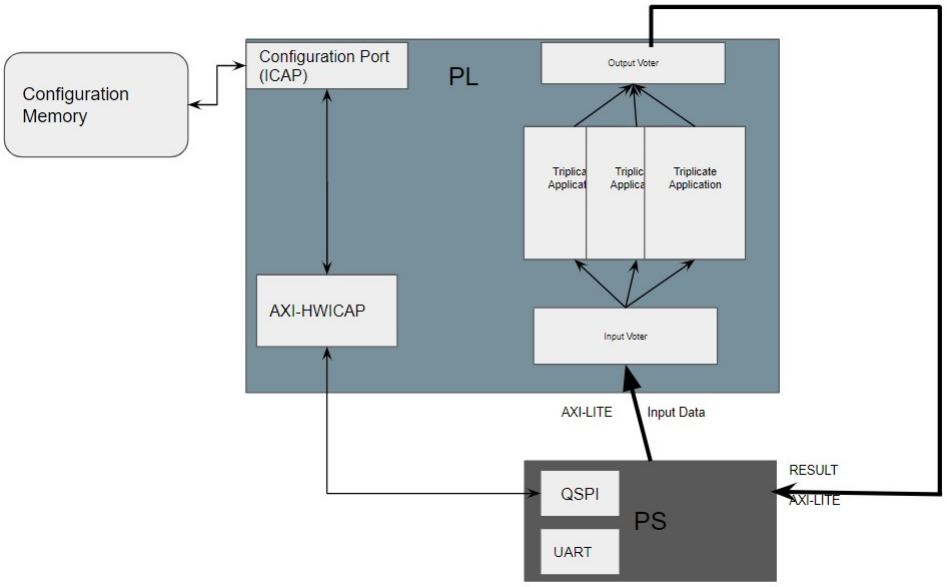


Figure 3.24: Architecture of TMR and Dynamic Partial Reconfiguration.

The proposed architecture in this section as it is shown in 3.24 consists of TMR and Dynamic Partial reconfiguration. TMR is applied on the system specification, in order to identify and mitigate the occurrence of a bit-flip. The voter takes in input the three replicated results and computes the correct data. Also in the case of a single module failure, signal which one of the replicated modules computes an erroneous value. This information will be used to control the re-configuration process in case the fault is identified as affected the faulty module multiple times, and thus exposing a "permanent" erroneous behavior. More precisely, the controller receiving the outputs of the voters and managing the reconfiguration, determines whether the fault occurred in a temporary register used for computation or in the configuration memory by monitoring the voter outputs in consecutive cycles, and in the latter case it triggers a re-configuration of the portion of the module affected by the fault.

In order to obtain this kind of system, the initial specification is partitioned into modules which to apply TMR to. A module is possible to recover from a fault in the configuration memory by applying partial reconfiguration using the partial bitstream bi in the area. As a consequence, when the system is being designed, the complete bitstream

is generated, together with all the partial bitstreams b1, b2, b3 used to re-configure only the single module of the FPGA.

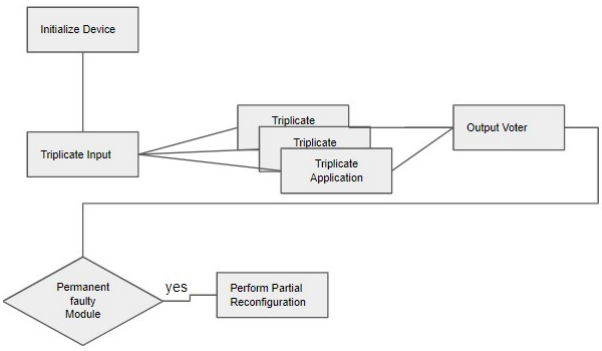


Figure 3.25: Architecture flow of TMR and Dynamic Partial Reconfiguration

In contemplation of the execution of partial reconfiguration, we have use the same method described in section 3.2.3. Where partial bitstreams are delivered to the ICAP interface of the FPGA device, through the Xilinx's AXI HWICAP IP core. To do show we have use the ARM-Processor and the AXI lite protocols. AXI-HWICAP is connected to an embedded soft-core ARM-processor which is also implemented in the FPGA fabric to control the operation of all IP cores of our system. More specifically, a software application running on ARM-Processor is responsible for reading bitstream configuration data from the the QSPI flash memory of the chip and delivering them to the ICAP interface through the AXI HWICAP core as it is shown in 3.24.As we can see in 3.25 in the detection of an SEU this architecture performs a Dynamic Partial Reconfiguration into the faulty module.

3.3.3 Internal Scrubber combine with Partial Reconfiguration and TMR

In order to apply all the previous architecture in one design first we should know the limitation of combing the above mentioned architecture. The first limitation is the use of the same component ICAP. In order to avoid this inconvenient we transfer the access of ICAP to PL when SEM need to use it and then take it back for HWICAP to perform the partial reconfiguration of the desired area whenever is it critical to replace

the fault component . Also an other thing to consider is that combining the SEM IP with Partial Reconfiguration does have an effect on the rate at which Partial Reconfiguration can be performed. The Partial Reconfiguration rate will be a trade off versus reliability. During Partial Reconfiguration period and reset of the SEM IP, the SEM IP is not able to perform the configuration memory scrubbing. Therefore the configuration memory is not protected against errors during this period. In the time Partial Reconfiguration is performed the SEM IP is idle. And afterwards the SEM IP needs to reset to consider the new functionality correct. So any error happened in the preceding idle period will not be detected. Even worse since the errors are present during reset they will be considered as a desirable change. So an error will be counted as correct behaviour. So in order to keep reliability as high as possible the scrubbing time needs to be as high as possible. The current solution is to maximize the ICAP clock reducing the time to Partial Reconfigure and the time to initiate. But this means that Partial Reconfiguration rate should be limited to keep the reliability above an acceptable level.

So if an error happens during the time of partial Reconfiguration or during the time initialization in a frame the SEM IP did not reach yet to compute ECC values for. The error will be considered as a correct value and incorporated into the ECC check for the frame. This means the SEM IP will not be able to observe if there is an error, therefore never trigger a repair and thus always fail to repair this error. To give an numerical feeling for the the time of partial Reconfiguration consider the next example. The ICAP as a 32 bit data interface and a clock at 100 MHz. Write speed of a partial file is 112MB/s or 67MB/s while the typical partial binary files used in this project were 350KB. So time for partial Reconfiguration were 1.05 ms.

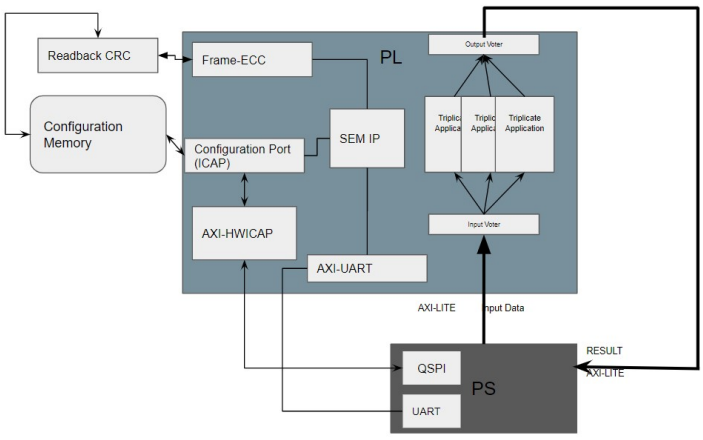


Figure 3.26: Architecture of Internal Scrubber, TMR and Dynamic Partial Reconfiguration.

As we can recall, the initialization of SEM IP was 12.5 ms for the Repair and Replace as correction methods. And 1.5s for Enhanced Repair as correction method. Since the time

Enhance Repair takes to initialize is considerably larger this method is very impractical to use in combination with Partial Reconfiguration since this would put serious constrains on how often Partial Reconfiguration can be applied while maintaining reliability ability. Time of the partial reconfiguration is 10 times smaller than time of initialization. This means the SEM IP is the bottleneck to how often Partial Reconfiguration can be applied and not the actual Partial Reconfiguration itself. Take this into account in our case there are multiple Partial Reconfigurable Regions in your design. It worth to schedule the Partial Reconfiguration such that multiple regions are done at the same time so they can be covered with one SEM IP initiate. So to solve this problem we perform the partial reconfiguration when a second module becomes faulty this lead to the lost of some correct output results but gives us the opportunity to lust longer as an application which leads to less downtime. So the software of ARM-Processor is responsible for performing the reconfiguration through Xilinx's AXI HWICAP IP core [18] using the partial files stores in the QSPI-Flash memory whenever more than one modules is down.

As far as reference is concerned TMR can be successfully implemented in our design without making a problem to our design as long as the area of the module does not exceed the 33% of the remaining PL area. The partial files that are stored in the QSPI-FLASH are dedicated to the area of the triplicated modules. In that occasion wherever partial reconfiguration is performed the faulty module is replaced by a correct configuration. Taking into consideration all the following problems we have implemented a robust system which is shown in 3.26. There we have successfully combined SEM IP, partial reconfiguration and TMR to implement a more robust system.

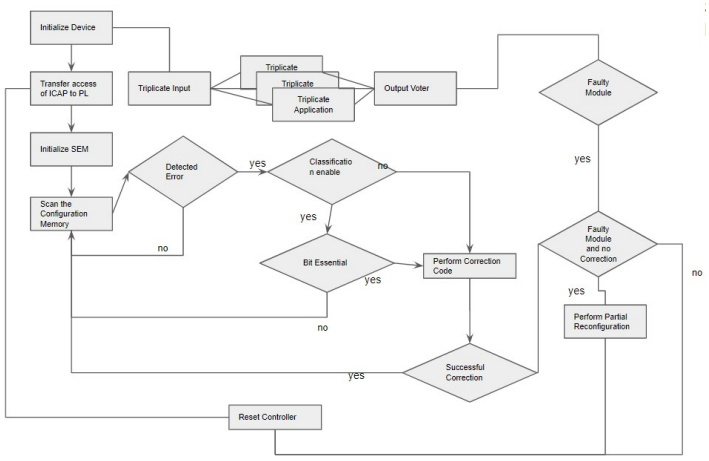


Figure 3.27: Architecture flow of TMR and Dynamic Partial Reconfiguration and Internal Scrubber

In respect of SEM Controller utilizes the internal Readback CRC hardware to perform error detection using the built-in ECC and CRC codes [1]. While The SEM IP, however,

performs its own correction by means of the ICAP. We have implemented SEM IP in Enhanced Repair mode as described 3.2.1. We choose Enhanced repair mode as it needs less time than replace mode to configure a frame also we have established an AXI-UART for communication purposes. The Figures 3.26 and 3.27 below shows the architecture described and the behaviour of the system in the occurrence of an Error. As we will see in the Result section, it is a very effective Architecture that provides high reliability to our system. The only problem with this architecture is that there are certain situations in which the system falls, and the only way to solve this is to restore the entire configuration to a point at which the system was working as it should be.

3.3.4 Watchdog and Internal Scrubber combined with Partial Reconfiguration and TMR

There was seen errors that was untraceable from all the above architectures so a new mitigation techniques should imported in order to achieve a complete solution for every Upset. So in order to correct those errors a watchdog has been introduce into the system. An effective watchdog should be able to detect all abnormal software modes and bring the system back to a known state. It should have its own clock and should be capable of providing a hardware reset on timeout to all the peripherals. The watchdog timer proposed operates independently of the processor and the FPGA. The architecture follows a windowed watchdog implementation. A fail flag is raised when the watchdog timer expires and after a fixed amount of time from raising the flag, a reset is triggered.

In our thesis this job is delivered by an Arduino controller. The controller is receiving a consistent message from our FPGA. In the absence of this message, Arduino is responsible for resetting our System. Due to problems in the Zybo Board this reset is performed by pushing a button in the Fpga in the occurrence of a message by Arduino. This a costly Architecture because its uses a lot of resources and power consumption but if our FPGA is not easily accessible then it is mandatory to include this technique in our design.

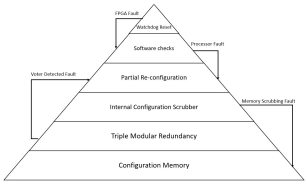


Figure 3.28: Hierarchical Healing

The proposed architecture of this robust system to avoid the destruction in the presence of SEFI and uncorrected MBUs or SBUs is implemented and described below. A fault-tolerant system has been developed which is based on multiple hierarchical layers of complementary schemes that are used to detect the faults and correct them as it is best represented in 3.28.

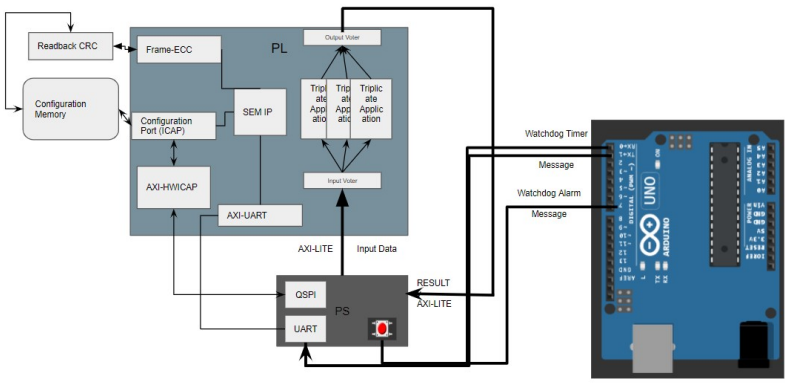


Figure 3.29: Architecture design of TMR, Dynamic Partial Reconfiguration, Internal Scrubber and Watchdog Timer.

The basic components of the critical system are implemented using TMR. But since these controller circuits are defined by the configuration memory, which itself is sensitive to radiation errors, configuration memory scrubbing (SEM IP) is also implemented to protect this design. If any of the TMR processing module are permanent disable then the ICAP primitive through the Xilinx's AXI HWICAP IP core [18] should perform the reconfiguration of the faulty module as it is described in 3.3.3. In our proof-of-concept, we utilized DDR4-DRAM as main memory, to store operating system code and data while we use the QSPI-Flash to store FPGA's configurations. DRAM and flash memories are commonly used in all FPGA-based systems and their use currently cannot be avoided. However, relatively low-cost radiation-robust variants of these components are available commercially. At the time of writing, flash memory is widely used in all space applications, and currently there is no readily available radiation-immune alternative. If also the ICAP or any basic component is hit by a permanent fault that compromise their functionality, then the software executed needs to re-configure the SoC-Fpga device via PCAP. If the watchdog timer expires without receiving the desire message then a reset of the FPGA must occur.The Figures 3.29 and 3.30 below shows the architecture described and the behaviour of the system in the occurrence of an Error.

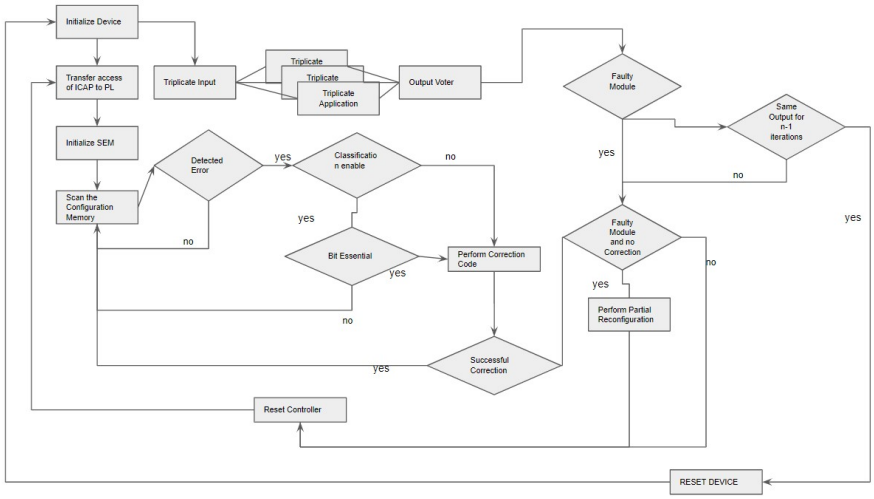


Figure 3.30: Architecture flow of TMR, Dynamic Partial Reconfiguration, Internal Scrubber and Watchdog

Chapter 4

Evaluation

The above mentioned fault detection and repair approach is evaluated by designing a system core with stand alone module. In Chapter 3 the design and implementation of a fault tolerant Architectures were discussed and introduced. This system should be able to correct the effects of any single fault and detect any double fault occurring in the pipeline, a data word, or an address word. It must be verified that the implemented solutions provide the SoC-Fpga with these capabilities. For this purpose, On board testing has be executed in contrast with testing in simulation. Furthermore, simulation based testing might not catch all problems in the design, as it is an approximation of real life. On board testing will provide better verification of the design, as this gives a much better approximation of real life. A test Architecture was build to control the verification design and perform automated verification. In this chapter the test of Architecture used will be explained and the results of this verification process will be presented in each section as well as the evaluation metrics in Section 4.1. The resource utilization of the Xilinx ZYBO SOC-Fpga will be presented in Section 4.1.1. The timing analysis performed for the proof of concept will be presented in each Section for every Architecture separately. The resource utilization as well as performance results are then compared in Section 4.2.

4.1 Experimental Evaluation

As we mention in Chapter 1 Fault is the cause of an error. It could be an event, a bug, a faulty circuit connection etc. An Error is what is directly affected by the fault such as an erroneous output, a non-functioning module or a faulty state. A Failure is when a service is not delivered or when it does not comply with the specification. A program which fails to deliver a value or a circuit which fails to write an output are both examples of failures. There are a number of different failure modes which are commonly used:

- Value failure – An incorrect value is delivered by the service
- Signalled failure – A failure signal is provided by the service
- Timing failure – A result is delivered too early or too late by the service

- Silent failure – No result is delivered by the service.

When looking at a system as a whole it is often useful to be able to model and simulate the reliability and behaviour. This is important when planning and designing a system to be able to implement the most efficient fault tolerant method. With known component failure distributions, it is possible to calculate metrics such as mean time to failure (MTTF) and the probability of the system working at a given time.

The lifetime of a component can often be modelled by the use of known distribution functions. More specifically, a probability density function describes the relative probability of an event. The probability density function is often denoted f(x). A distribution function, also referred to as accumulative distribution function, is defined according to 4.1. The density function describes the probability that a stochastic variable X will have a value equal or smaller than a variable x, as shown in 4.1.

$$F(x) = \int_{-\infty}^{\infty} f(x)\,dx = P(X \leq x) \tag{4.1}$$

An application where the mentioned functions are used is when modelling lifetimes of components. Assume the probability density function describes the probability of a failure as a function of time with a known distribution. The distribution function can then be used to calculate the probability of a failure having occurred at a given time. The reliability function, or survival function, is defined according to 4.2. The reliability function is the probability that a stochastic variable X has a value greater than the variable x. As shown by 4.2, the reliability function can simply be resolved by calculating the remaining probability of a distribution function for the same variable. Using the earlier component lifetime application, the reliability function would return the probability of a failure not having occurred at a given time.

$$R(x) = 1 - F(x) = P(X > x) \tag{4.2}$$

In the case where the failure rate is described by a distribution, a probability density function can be used to resolve information as well. In this case, MTTF, is nothing but the expected value E[x] of the same probability density function, which is defined according to 4.3. MTTF is frequently used as a metric when describing fault-tolerant systems.

$$MTTF = E[x] = \int_{0}^{\infty} tR_{\lambda}(t)\,dt \tag{4.3}$$

The failure rate, denoted $\lambda(x)$, is the frequency of failures per unit time. It is defined according to 4.4 as the ratio between the probability density function and reliability function. It is commonly expressed as failures per hours, which is another frequently occurring metric. Mean time between failures (MTBF) can be resolved as the inverted failure rate according to 4.4, but also as the sum of MTTF and MTTR.

$$\lambda(x) = \frac{f(x)}{R(x)} = \frac{1}{MTBF} \tag{4.4}$$

In order to estimate the failure rate of the Reliability model we use the experimental failure rate as described in [19]. For a given sample size m, there will be n failures after t hours Operating hours. If 'm' operated for 't' hours before the failure-count 'n' was noted, then Operating Hours = m*t

$$\lambda_a vg = \frac{n}{m * T}$$

Thus System reliability is the probability that a system is operating without faults after a specified time period. Assuming exponentially-distributed random faults at rate λ, the system reliability is traditionally defined as shown in Eq. 4.5.Also Reliability of a system can be experimentally determined by the ratio of time the system is operating correctly to the total experiment run-time as we can see in 4.6.

$$R(t) = e^{-\lambda \cdot t}, \quad \text{where} \quad \lambda = \frac{n}{m \cdot t} \ \ (n\text{: failures after } t,\ m\text{: samples at } t) \tag{4.5}$$

$$\text{Reliability} = \frac{\text{Functional execution time}}{\text{Total execution time}} \tag{4.6}$$

The ratio between the downtime and running time of the system is the definition of Availability. It can also be calculated using MTTF and MTTR as shown in 4.8. A common availability standard for critical systems is the "five-nines" standard , indicating an availability of 99.999% [57]. Also Availability can be experimentally determined by the ratio of time the system is operating to the total experiment run-time as we can see in 4.7.

$$\text{Availability} = \frac{\text{Functional execution time + non-Functional time}}{\text{Total execution time}} \tag{4.7}$$

$$A = \frac{MTTF}{MTTF + MTTR} \tag{4.8}$$

In conclusion Reliability is the likelihood that a system will remain operational (potentially despite failures) for the duration of a mission. For instance, the requirement might be stated as a 0.999999 availability for a 10-hour mission. Very high reliability is most important in critical applications such as the space shuttle or industrial control, in which failure could mean loss of life. Availability expresses the fraction of time a system is operational. A 0.999999 availability means the system is not operational at most one hour in a million hours. It is important to note that a system with high availability may in fact fail. However, its recovery time and failure frequency must be small enough to achieve the desired availability. High availability is important in many applications, including airline reservations and telephone switching, in which every minute of downtime translates into significant revenue loss.

Also in order to evaluate the application which is running we should introduce two other metrics beside Availability and Reliability with whom which we will preform the evaluation. Those two metrics are the Error Rate and the Mean Relative Error of the

Architecture which describes the measurement of the effectiveness of our application in terms of accuracy and quality metrics [58].

Error Rate is the ratio of the number of erroneous units of data to the total number of units of data received as shown in 4.9.

$$ER = \frac{1}{n}\sum_{i=1}^{n}(X_i) \tag{4.9}$$

where X_i is 0 or 1 if the output is the expected or not and n: number of data received.

Mean Relative Error denotes the ratio of the 1 norm of the error vector $Y_i - \hat{Y}_i$ to the number of samples and is defined as shown in 4.10. The MRE is used to measure the closeness of the prediction to the eventual outcomes between predicted values ($\hat{y}$) and observed values (y) where n is the number of observations.

$$MRE = \frac{1}{n}\sum_{i=1}^{n}(\frac{Y_i - \hat{Y}_i}{\hat{Y}_i}) \tag{4.10}$$

4.1.1 Experimental Setup

We selected Xilinx's Zynq-7000 SoC (System-on-Chip) commercial board namely Zybo as our implementation platform. The reason for this selection is that Zybo contains a XC7Z010 device from Zynq-7000 AP SoC family of Xilinx, which includes a dual core ARM Cortex™-A9 processor and 28nm Artix-7 based SRAM-based programmable logic (FPGA) allowing us to easily program and control the behavior of the reconfigurable logic via the Arm cores. The test method proposed in this paper is not restricted to the selected FPGA platform. It is rather a general method that can be employed for all types of SRAM-based FPGAs. The architectural overview of this board is given in Fig. 4.1. As can be seen in this figure, the board has two main components connected via AXI buses: the processing system (PS) and the programmable logic (PL). Zybo XC7Z010 also has two ports that allow us to access and change the FPGA configuration bits: the processor configuration access port (PCAP) and the internal configuration accesport (ICAP). While PCAP is used to access the FPGA configuration bits from the PS, ICAP is utilized to access the FPGA configuration bits from the PL part.

The software utilized to develop and implement all the mention Architectures in Chapter 3 was Vivado Design Suit(2016.2). It provide the tools needed to develop complex designs,optimizing the implementation for Xilinx devices, providing IP sub-systems reuse and accelerating the design process by enabling to work at a high level of abstraction.Also to generate the desire bistreams (.bit). Additional software Xilinx SDk was used to implement the C code that will run as the bare metal project in the ARM processors. Also an Arduino UNO was use to implemented the watchdog timer as an external inspector.

In order to obtain reliable and comparable result in the different validation tests the methodology described below has been meanly followed. Once the VHDL code of the

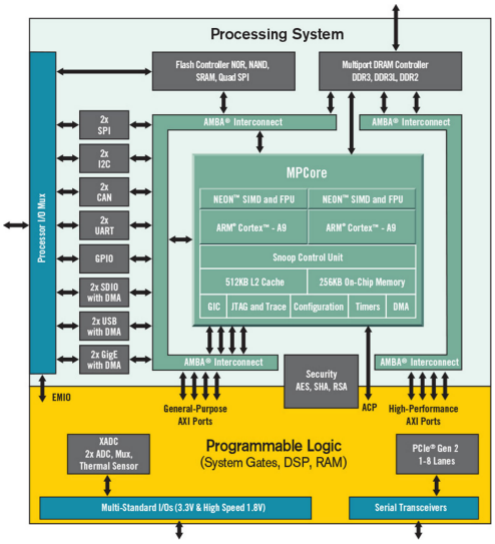


Figure 4.1: Architectural overview of the Zynq-7000 SoC.

design has been properly written and checked, the first validation step has been to synthesize and simulate it. This step has been done by using Vivado 16.2 and its simulator. After debugging and improving the designs, the next step has been to implement and generate the bitstream (partial and other) as also the necessary ebd,ebc files to be download into the FPGA. Also Vivado 16.2 suite was employed to carry out these processes. In a further phase, the designs have been validated by running real case applications on the soft-core processors under test. This have provided the insight into correctness of the results obtained in a real world scenario. Despite that in some cases specific applications have utilized, the mainly used one has been a RS232 serial transmission based application. This application sends 8 bits data word as input in a FIR filter and receives 16-bits word as an output of the FIR.

4.1.2 Overview of the Test Setup

In order to inject in the configuration memory, a first option is to follow the procedure explained in [59], which basically consists in performing an exhaustive campaign in all the configuration bits of the FPGA. After each injection, the behavior of the design is compared to the golden,counting how many times a wrong output has been produced. The number of injections that did actually produce an error divided by the total number of configurations bits in the board (that is equal to the total number of performed injections) would represent the percentage of Errors induced in the design. This can be used as a figure of metric of the sensitivity of the FPGA with the tested design implemented in it. However, this is not exactly true, because since the injections are not targeted, many of them would have affected configuration bits on the board that are not part of the design. In this case, an absence of error at the output does not have to mean that the injected bit is not sensitive, just that it is not part of the design. Therefore, a scale factor should be

applied to account for this fact and obtain the normalized sensitivity. This scale factor is simple, just the quotient of the total number of slices in the board divided by the number of slices in the design. This represent how large the design is respect to the total capacity of the board (all details can be found in [59]).

As an example, let us assume that the obtained sensitivity (injections that have produced an error divided by the total number of injections) is 8%. Let us also assume that the design under test occupies just 10% of the board capacity. Then the scale factor would be 10, and therefore the normalized sensitivity would be ten times 8%, which is 80%. This method is quite simple, but it has a huge drawback: it is based on exhaustive injections, so all the bits in the configuration memory have to be tested. This is not always possible due to the high number of configuration bits in most FPGAs. For example, our Board has 12,924,768 configuration memory bits and a larger FPGA such as the Kintex UltraScale KCU105 has 69,338,912 bits. This large number of injections is likely unfeasible in most of the cases. Another option, especially for complex designs, is to perform statistical injections. This means to inject randomly in a limited number of the configuration bits, enough to obtain a representative behavior of the injections. In this case, when statistical t^2 injections are performed, the concept of confidence level and error margin have to be taken into account. This comes defined in equation 1, proposed in [60].

$$n = \frac{N}{1 + e^2 \cdot \frac{N-1}{t^2 \cdot p \cdot (1-p)}} \tag{1}$$

where N is the total number of configuration bits in the design, e is the error margin of the results, and it is a factor associated to the confidence level, in other words, the probability that the obtained result is within the specified error margin. For example an error margin of 1% (e=0.01) and a confidence level of 95% (t=1.96) means that the obtained result is 95% likely to have an error of 1% or less (the relation of a given confidence level and the related t can be found in [60]). p is a statistical parameter in the [0, 1] range, with a worst-case value of 0.5. Finally, n is the number of injections that are required on N in order to achieve the specified error margin and confidence level.

As an example, let us consider our FIR application implemented in an FPGA. A typical value of N would be around $6x10^5$ essential bits. In order to achieve an error margin of 1% and a confidence level of 95%,we would need a number of injections of n=9,904. Therefore, just with that number of injections, we would get the results in the expected error margin. The problem with this approach is that we need to have the list of the configuration bits essential to the design (N), in order to select the random subset in which to perform the injections (n). If that is the case, then the method can be directly used. But if the target FPGA and the associated design framework does not provide this possibility, then the process becomes quite inconvenient. Let us assume a board, with for example $12x10^6$ configurations bits, in which a our design with $6x10^5$ essential bit(N) is implemented. If the latter list of bits is not known, then a random but blind injection campaign has to be performed in the board. Since in this case N is 10 timer smaller than the total configuration

bits, then around $\frac{1}{10}$ of the injections would actually affect bits in N. In other words, if 9,604 injections are performed, statistically only 900 of them would actually be effective, what would lead to a worsening of the error margin. If we want to keep it at the expected level, then we would have to perform 10 times the number of injections, i.e. 99,04. This means that not having a clear idea of which of the bits in the configuration memory are essential to the design, implies a large time overhead in the injection process, having to perform more injections that the minimum required.

There are some ways to obtain the list of configuration bits associated to a design. For example, Xilinx provides some methods to generate this list [61]. But even if we are in the situation in which we actually know the list of essential bits in a design, we still may have problems in some situations. Let us assume that we do have the list of $6x10^5$ essential bits of the previous design. That would allow an optima injection campaign based on the previously described statistical method. But let us imagine that we want to test the behavior of a new protection technique devised for a specific component of the design, i.e. the FIR filter. Of course we want to exercise the FIR integrated with the whole design, in order to have a functional design and be able to run a benchmark. In this situation, even if we have the list of essential bits of the processor, what we also need is the subset of the bits that are essential to the FIR. Let us assume that these are just 30,000 of the total $6x10^5$.

Now, our N would be these 30,000, and to keep the error margin and confidence level selected above, we would need a number of injections of n=9,307, according to (1). But if we do not have the list of these 30,000 bits, we would need to perform the injection campaign in the overall $6x10^5$ essential bits. And to guarantee that statistically 9,307 of them affect the FIR, we would need to perform 20 times this number of injections, i.e. 186,140 injections (since the whole design is 20 times larger than the FIR). This number of injections would be unfeasible in most of the cases. And even if it is feasible, it would take a huge time to complete the injection campaign, leading to an inefficiency of the method.

Therefore, As Xilinx provide the fundamental tool that can provide the list of essential bits associated to a design. We use ACME tool [62]which provide us the subset of the essential bits associated to a sub module of the design. In Fig. 4.2 we present the command that enables an algorithm, already implemented in the Vivado Design Suite, which automatically creates the EBD file together with the bitstream of the design. Once the EBD file is generated and the FIR coordinates noted down, ACME can be executed to obtain the injection addresses of the FIR module that can be sent to the SEM IP to perform the fault injection campaign.

```
set_property BITSTREAM.SEU.ESSENTIALBITS yes [current_design]
```

Figure 4.2: Necessary command in xdc file in order to generate the EBD file

So after we have obtain the linear addresses of essential bits we have implement a test where where 10000 injections are made in specific components of the design. In this

test the injection are made in different component of the design in order to see how our Architecture respond to different kind of Errors in configuration memory. To do so the PS is connected to the SEM IP via the Uart. The test system verifies the frame information by injecting error in any Configuration Frame. Error injection Via Xilinx SEM IP is injecting Errors in the Configuration memory in the area of Pl.So in order to successfully perform an experiment that can quantifies the Errors and their correction,we set a bidirectional communication between the PS and PL of the SOC - FPGA with the help of Uart Protocol. The instruction parsing interface of the SEM IP is called the monitoring interface. The PS generates the instructions and sends them to the SEM IP via the UART pins. A dedicated PS-UART is used for communication.we have incorporate into our system the injection interface of the SEM IP core.

To inject an error the controller must be in idle state. Once in the idle, the error can be injected by issuing the following command followed by a carriage return:

$$N\{10 - DigitHexValue\} < ret >$$

The 10-digit hex value specifies the bit location where the error is to be injected. The location can be specified using either linear frame address or physical frame address, the word offset the bit offset. In ZYNQ-7000 family only the linear frame injection is applied. The format of the 10-digit hex value is given by Figure 4.3. Successful injection of the error is indicated by the assertion and de-assertion of the injection status signal as well as on the monitor interface be showing a state changing report that indicating that the controller has successfully transitioned from the injection state back to the idle state.

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	0	S	S	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	W	W	W	W	W	W	B	B	B	B	B	B

Figure 4.3: Error injection Command word Linear Address

SS = Hardware SLR set to 00 For Non-SSI devices
LLLLLLLLLLLLLLL = Linear frame address (17-bit)
WWWWWWW = Word Address(7-bit)
BBBBB= Bit Address (5-bit)

4.1.3 Error Resiliency of our Design

In order to test the resiliency of our design we implemented the test described in 4.1.1. In this section we observe how the injection are affecting a simple design. It is critical to mention that only 4% of the configuration bits are essential bits in our design implemented. However we see in 4.1 that our injection strategy is working and the injections addresses are affecting the functionality of the design.

In 4.1 below we see that the resilience of the system without a detection or correction mechanisms is very low and the Error Rate is approaching a dead system. Another take

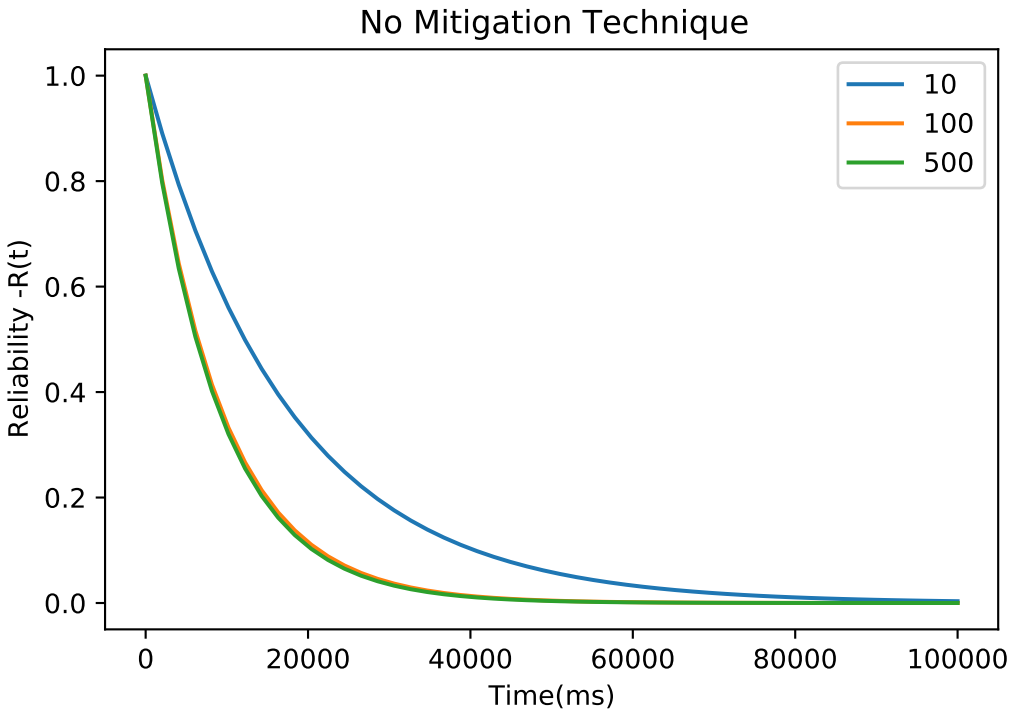


Figure 4.4: Reliability of no mitigation technique: each function corresponds to # injections.

on our experiment is that almost the 0.5% of the essential bits are critical bits. This mean that those bits drive the system into dead states which means its doesn't respond to any commands. That is the reason why even though we made 10K of injections the systems stop to respond after 900 of them. Also in 4.4 we can observe that the more injections are introduced into the system the more the failure rate is increasing thus the reliability of the system decreases. We see in action and in reality how crucial it is to implement fault tolerance techniques in our design. The results from this first test were used as reference for the comparison of the different mitigation techniques.

Number Of Injection	Error Rate	Mean Relative Error
10	95%	17.3%
30	97.5%	37%
60	98.57%	65%
100	99.09%	101%
160	99.41%	153%
250	99.62%	227%
500	99.8%	456%
800	99.8%	739%

Table 4.1: Error Rate and Mean Relative Error for No Mitigation Technique.

4.1.4 Evaluation of Triple Modular Redundancy

TMR is a recognized technique for improving the reliability of FPGAs in a radiation environment. Several projects have demonstrated unique ways of implementing TMR within FPGAs to improve reliability. The effectiveness of TMR circuits produced is cleared to our design as in fact TMR gives the ability to protect the system against the injections in specific essential bits.

In this section we have deployed the TMR as fault tolerance technique. TMR comes at great cost. Full TMR of a design requires, at a minimum, three times the hardware to implement three identical copies of a given circuit. Additional hardware is also required to perform the majority voting on the three circuit modules. Some studies have shown that TMR can require up to six times the area of the original circuit [59]. Furthermore, TMR can negatively affect time, as incoming voters are a combination logic circuit and therefore increase path lengths as well the extra resources ultimately also require more power which can lead to memory leaks.

	Without TMR		With TMR	
Number Of Injections	Error Rate	Mean Relative Error	Error Rate	Mean Relative Error
10	95%	17.3%	30%	33%
30	97.5%	37%	10%	33%
60	98.57%	65%	5%	33%
100	99.09%	101%	3%	33%
160	99.41%	153%	2.5%	34%
250	99.62%	228%	14.8%	476%
500	99.8%	456%	57%	622 %
800	99.8%	739%	73%	732 %
1000	-	-	78.5%	832%
1500	-	-	85.5%	854%
1800	-	-	87.9%	905%

Table 4.2: Comparison of TMR Architecture with in No Mitigation Technique in terms of Error Rate and Mean Relative Error.

In the 4.2 it is very clear that we have a 13% decrements in the Error Rate while we have gain the ability to detect the 70% of the Errors occur. Also it is very crucial that TMR can limit the Mean Relative Error below 35% in the first 250 injections. Which give us the ability to use TMR in an environment with minimal injections. An other observation from the results is that the reliability 4.5 of the system is much better that the None Fault tolerant Architecture and justifies the use of TMR in many ground terrestrial applications but as it is shown in 4.2 this is very cleared and inevitable as the injections are increasing and this cause the system Reliability to descending.

Something that we did not observe in our experiment was the increased sensitivity due

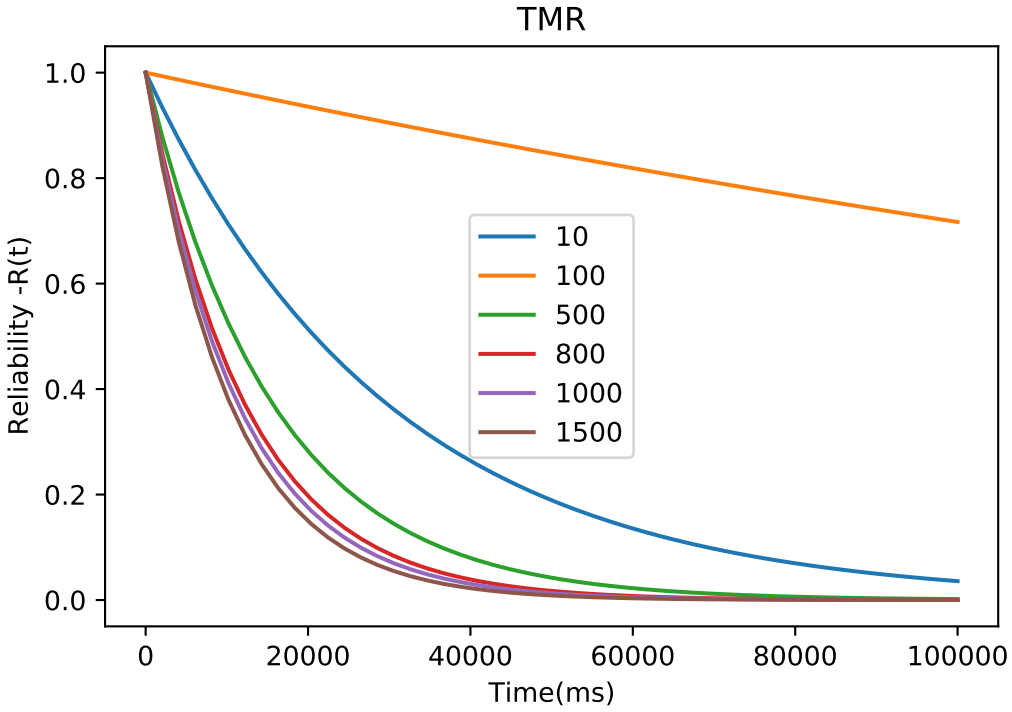


Figure 4.5: Reliability of TMR: each function corresponds to # injections.

to the mitigation logic added to the design. This was expected due to the nature of our experiment and the fact that we already targeted sensitive bits and not all the FPGA's Memory. While TMR with repair does improve the radiation resiliency of the FPGA circuit, it is not as effective as it could be. There are single bits within the configuration memory that when upset can cause instantaneous TMR circuit failure. In essence, these bits represent a failure mechanism that is not repairable by the TMR. These failures cannot be avoided by using TMR alone, but can only be avoided by providing specific mitigation for them.

4.1.5 Evaluation of SEM Scrubber

In this section we will analyze the impact of SEM IP as scrubber. We will see how the different capabilities in error correction can have different impact in the resilience of our design. The test described in 4.1.1 that took place to estimate the effects of SEM IP in Error Correction, detection and the deviation of Mean Relative Error. Fig 4.6 shows that the SEM Controller gives an exiting edge into the reliability of the system even though after 7000 injections a failure occurs and we lose communication with the our system. In order to evaluate the SEM Scrubber as a fault tolerance Architecture we first need to evaluate the three option of mitigation techniques and describe.

In the test that have been perform Soft Error mitigation Controller highlighted some problems. The First one was that Replace mechanism even-thought is the most promising

	Without SEM		With SEM	
Number Of Injection	Error Rate	Mean Relative Error	Error Rate	Mean Relative Error
10	95%	17.3%	10%	0%
30	97.5%	37%	3.3%	0%
60	98.57%	65%	1.6%	0%
100	99.09%	101%	11%	0%
160	99.41%	153%	26.87%	2.7%
250	99.62%	228%	35.2%	3.1%
500	99.8%	456%	43.2%	5.4 %
800	99.8%	739%	46.9%	7.8 %
1000	-	-	47.5%	8.6 %
1500	-	-	48.7%	10.3%
1800	-	-	49.17%	11.2 %
2500	-	-	49.7%	12.5 %
3600	-	-	49.9%	14.5%
5000	-	-	50.1%	15 %
7000	-	-	50.4%	18%

Table 4.3: Comparison of SEM IP Architecture with in No Mitigation Technique in terms of Error Rate and Mean Relative Error.

mode in respect of the correction of the errors in our configuration needs an external flash memory to store the golden copy and need a lot of time to extract this. We use AXI-QSPI to extract the file from flash memory but this lead to the actual a time of$T_{corr} = 448ms$. An alternation of this design would be to connect via SPI protocol and use external pins to connect with an external memory. SEM IP when connected via SPI protocol with external memory the correction time will significant reduced.The replace via a golden copy will be the best option as the total correction will eventually reduced to 890 μs plus time to extract the file from external memory. This will be very small compared to the AXI protocol, as the correction time for errors is 448ms, it leads to the fact that the system can fix up to two errors per second. A major portion of this time is consumed by the slow serial link, this time can be reduced by increasing the serial link data rate. From the results, approximately 6.5ms are required on top of the communication time for fixing the error. So, if a higher correction frequency is required, this communication time should be reduced accordingly. For instance, if the required correction frequency is 50 errors per second then the total correction time should be less than or equal to 20ms. Subtracting the time components other than the communication time leaves 13.5ms. Total transmission for fixing one frame is 420 bytes (one frame has 101 word of 32 bits), and to get this much information transmitted in 13.5ms, the baud rate should be set to approximately 250Kbps. In other FPGA this will be feasible but not in ZYBO. So it will be best to use the Enhanced repair mode in ZYBO Board as it consumes less resources and can correct

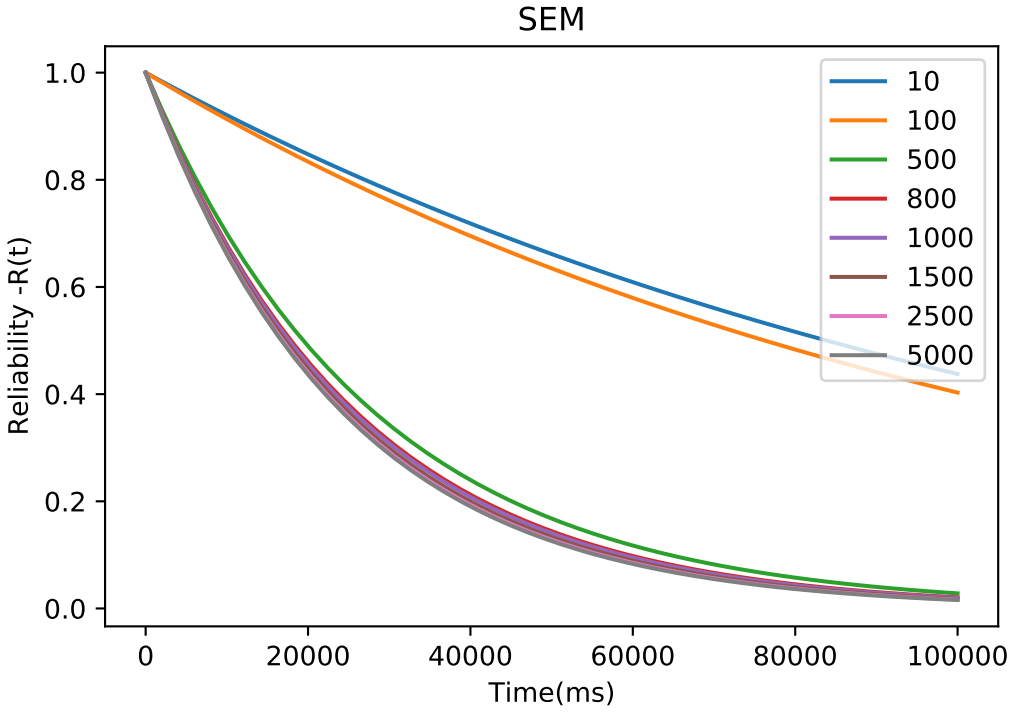


Figure 4.6: Reliability of SEM: each function corresponds to # injections

up to 2 bits in a frame under 18 ms.

The SEM IP Reliability is 4 times higher than the FPGA with none mitigation technique (based on 4.6 measurements). SEM IP show that almost 50% of events in essential bits are correctable in Xilinx ZYBO Board.

Correction Mode	Errors in Frame	Correction Time
Repair	1-bit (Correctable)	610 µs
Enhanced Repair	1-bit (Correctable)	610 µs
	2-bit (Correctable)	18 ms
Replace	Any (Correctable)	448 ms

Table 4.4: Max Error Correction Latency at ICAP_Fmax and No Throttling on Monitor Interface

The experimental evaluation of the SEM Architecture indicate the problems of this Architecture. The First one is that if a configuration bits that are responsible for the SEM implementation are effected by the injection then system has unpredictable behaviour as we can easily spotted in 4.6. This is something that needs to be fixed and will eventually be solved by combining other methods of mitigation techniques. Another problem is that SEM IP repairs errors before it reports there is an error. This means that if an error is detected it needs to be corrected by the SEM IP but there are some cases the SEM IP is unable to repair. It will report this but the SEM IP cannot take further action. It is advised to restart SEM IP at this point of time since correctness of the configuration

memory can no longer be guaranteed. At this point after the restart, we hope that the errors that will occur will not influenced by the previous uncorrected error. This is something that is happening as we can see in 4.3, 1958 uncorrected errors do not lead to a large increase in the mean Mean Relative Error metric while in fact it is limited in 20% but as we can see it affects the functionality of the design as after the 7k injections the system is unreachable. Another problem that occurs with the functionality resets is that is not possible to force the SEM IP to skip a part of the configuration memory. It will always scan the whole configuration memory. Since it cannot skip the Partial Regions and SEM IP corrects before it reports, an error reported in these regions needs to be fixed by the SEM IP. So if there was an uncorrectable error the SEM IP will still tries to correct it even if it cannot so a functional reset of SEM IP is needed to start scanning again for new errors. So a part functional reset with be ideal for this purpose.

In order to create new and improved scrubbing Architectures, it is useful to examine what has already been developed. This section presented an overview of the results from the evaluation of SEM IP controller. As we can see any investigation in Fault Tolerant Architecture must involving SEM IP or a similar Internal Scrubber because of its capabilities.

4.1.6 Evaluation of Partial Reconfiguration

In this section we will analyze the impact of Partial Reconfiguration as a correction method in our system. We have executed the experiment that has been evaluated in 4.1.1. Following the test we have see that Partial Reconfiguration increases system performance. Although a portion of the design is being reconfigured, the rest of the system can continue to operate.

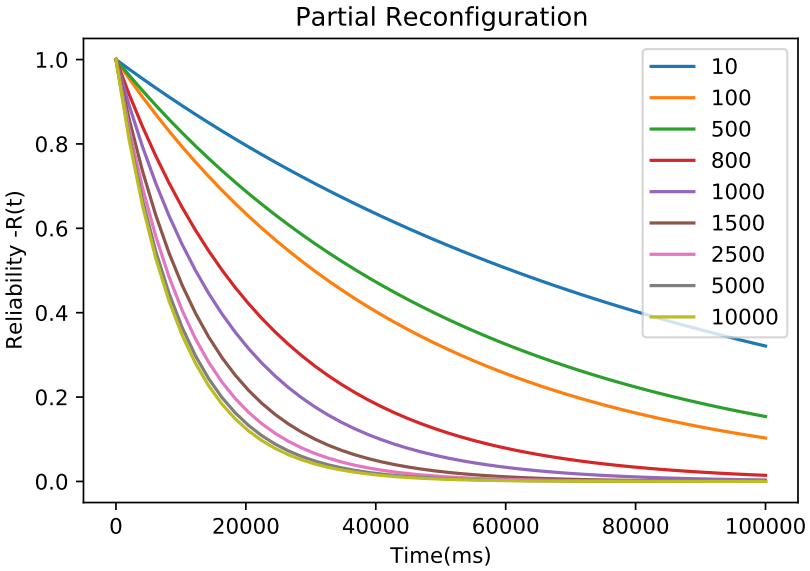


Figure 4.7: Reliability of PR: each function corresponds to # injections.

There is no loss of performance or functionality with unaffected portions of a design (no down time). It also allows for multiple applications on a single FPGA. This can be observed in 4.7 as we can see even-though the reliability of the system is low due the late reconfiguration process, the availability of the system is very high and that is happening because Partial Reconfiguration Manages to keep the system in working order. Xilinx FPGAs can be updated at any time, locally or remotely.

Also other benefits is that reduces device count, power consumption, smaller boards, and overall lower costs. Last and most significant advantage of Partial Reconfiguration is the shorter reconfiguration times. Configuration time is directly proportional to the size of the configuration bitstream. Partial reconfiguration allows you to make small modifications without having to reconfigure the entire device. By changing only portions of the bitstream as opposed to reconfiguring the entire device the total reconfiguration time is shorter. As we seen in 4.14 Partial Reconfiguration absorb great amount of resources

	Without PR1		With PR1	
Number Of Injection	Error Rate	Mean Relative Error	Error Rate	Mean Relative Error
10	95%	17.3%	10%	1.5 %
30	97.5%	37%	15%	5.5 %
60	98.57%	65%	17%	2.34 %
100	99.09%	101%	20%	7.5%
160	99.41%	153%	18.8%	7.7%
250	99.62%	228%	17.31%	8.1 %
500	99.8%	456%	16.4%	10%
800	99.8%	739%	37.25%	34%
1000	-	-	49.8%	53 %
1500	-	-	66%	81 %
2500	-	-	77%	110%
3600	-	-	83%	131%
5000	-	-	87%	150%
10000	-	-	91.5%	188 %

Table 4.5: Comparison of Partial Reconfiguration Architecture with in No Mitigation Technique in terms of Error Rate and Mean Relative Error.

such as IO and BRAMs. This is because we have implement our own custom ICAP which stores the partial files into BRAMs. This even though it is very effective in the reconfiguration is very expensive in resources and vulnerable to single bits Errors because BRAMs are the first to be affected form radiation as it is shown in[63]. So in a Real-Time application this method will not be effective.

As far Error Rate will be close to the none Fault tolerance Architecture as it shown in our experiment results 4.5 and that is because of the absence of detection mechanism. This absence is responsible for the high values of Error Rate and Mean Relative Error. Partial

Reconfiguration has the worsts deviation from the estimated values. So the delay of the perform of the reconfiguration drives the system to the worst results of any Architecture introduced in this Thesis. This Architecture is overlooked in continuity due to the volume of resources it achieves and from the absence of a detection mechanism.

Partial Reconfiguration Type	Throughput (MBs/sec)	Configuration Memory Resources			
		LUTs	**FFs**	**IOs**	**BRAMs**
Custom Icap (PR1)	1199	758	1436	59	57
Xilix HWICAP (PR2)	67	706	1040	0	0
Pcap	128	0	0	0	0

Table 4.6: Partial Reconfiguration comparison.

In the 4.6 we see the differences in various Reconfiguration techniques. After investigate the results in this chapter we will select Xilinx Hwicap as our prefer method to partial reconfiguration. The reason we pick this method to perform partial reconfiguration, despite the fact that Custom Icap (PR1) is much faster ,is that in condition likes the ones in experiment [63] will affect BRAMs and will eventually creates actually more problems. Also the HWICAP has the ability to be controlled by the ARM-Cortex and uses the external memory to allocated the desired partial bitstreams. Which give an edge to implement into PL logic whatever is needed without minding the resources minimization by the custom ICAP and BRAMs to store the bitsreams.

4.1.7 Internal Scrubber combine with TMR

Internal Scrubber(SEM) and TMR should be considered as complementary methods for the mitigation of radiation effects. While TMR makes the device immune to single upsets of control and data processing block, scrubbing heals flipped bits in the configuration memory. It thus avoids accumulation of upsets which can lead to malfunction even if the design is TMR. In simple words, TMR protects against a few SEUs in the user or configuration memory, while scrubbing keeps the number of upset as low as possible. This is a very interesting combination of fault tolerance techniques because it solve the problem of TMR which is the absence of a repair mechanism and a reactive protection when its face permanent Errors and also saves time from SEM detection and correction when the Errors are transients. But it lucks the ability to correct critical Errors in the TMR area which leads to the termination of the application.

The combination of TMR and Scrubbing (SEM), is an effective scheme, but providing a protection against SEUs is not sufficient to validate our system against a radiation environment. The architecture has limitations, both in the TMR and in the Scrubber, and these limitation entail an imperfect reliability, which must be quantified as we can see in 4.8 as the injections are increasing. The first limitation of this combined Architecture is that TMR will break if errors are present in more than one redundancy domain within

Number Of Injection	Without TMR & SEM		With TMR & SEM	
	Error Rate	Mean Relative Error	Error Rate	Mean Relative Error
10	95%	17.3%	20%	0 %
30	97.5%	37%	13.3%	5.8%
60	98.57%	65%	8%	5.8 %
100	99.09%	101%	7%	5.9%
160	99.41%	153%	7.5%	6.6%
250	99.62%	228%	7.6%	7 %
500	99.8%	456%	7.6%	8.4%
800	99.8%	739%	6.6%	8.7%
1000	-	-	8.4%	8.9%
1500	-	-	10.1%	9.7%
2500	-	-	11.2%	10%
5000	-	-	12.2%	12%
10000	-	-	34.05%	40%

Table 4.7: Comparison of TMR & SEM Architecture with in No Mitigation Technique in terms of Error Rate and Mean Relative Error.

the same module. There are two situation in which this can be observed. First is the accumulation of soft errors in the device, assuming each particle only causes errors in at most one domain. The second occurs when this assumption does not hold, when a single particle event (SEU) causes errors in more than one redundancy domain, this is called a Domain Crossing Error (DCE). The last one can easily observed in 4.7 after the 6K injections more than two module has been affected and the Error Rate is starting to grow. However we can identify from 4.7 that Mean Relative Error is gradually increasing until 7k injections while after that it is growing exponentially. This is happening because SEM IP after 7k injections encounters multiple un-correctable errors which affects its functionality and it is not able to correct a DCE and thus it increases the chance of system failure.

An important fact to note is that what ever type of error is the uncorrected error, the Xilinx SEM IP Core will stop all detection/correction operations until reconfiguration and needs a restart to continue operate. Therefore behave like it never happen and continue to scan for errors.

The problem here is that because SEM can only repair 50% of essential bits in a configuration memory another repair mechanism must be introduced. This is happening because SEM controller is design to mask the Errors that can not be fixed by reset his functionality . This mean that every time an essential error is corrected the SEM resets and re-initialized and calculates again the CRC account in order to be able to repair the new Errors. This is very critical in a competitive environment where injections are many and frequent because some of those Errors can modify the functionality of the design and drives into a false functionality of the design. If the applications is into a competitive

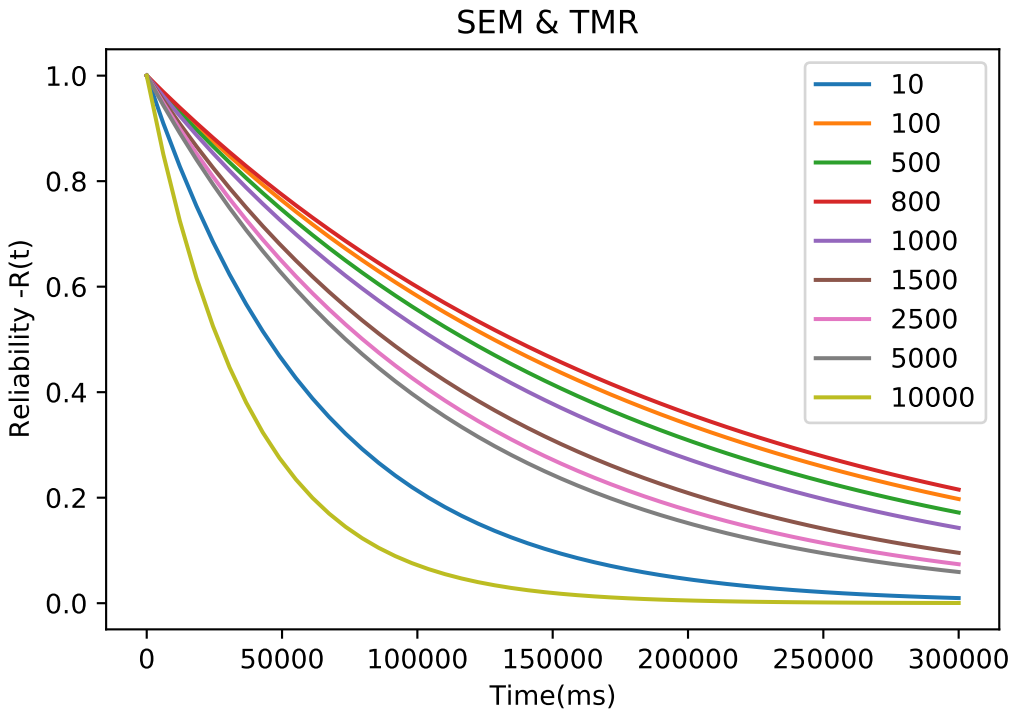


Figure 4.8: Reliability of SEM and TMR: each function corresponds to # injections.

environment in respect of the frequency of the error a Fully or Partial reconfiguration must be introduced or a return to a checkpoint where the functionality of the application is still is intact.

4.1.8 TMR Combine with Partial Reconfiguration

As we see in TMR section in order to fully unlock the potentials of TMR we need a repair mechanism. In this subsection we will observe how the Partial Reconfiguration of the faulty partition of TMR affects our system. TMR mechanism implies implementing the same module/function three times. The redundant modules run in parallel, performing the same computation. The results from the three modules are analyzed by another module, responsible for voting the correct result. This way, even if a module produces wrong results, the other two will guarantee the correct operation. The weak point in this strategy is the voter, as if it fails. The major advantage of TMR is its error detection and correction feature as we saw in 4.1.4. An important drawback is the need to employ three times more hardware than other mitigation Techniques nnd also the fact that is the absence of Repair mechanism.

In this section we use Partial Reconfiguration as a repair mechanism of TMR and it can easily observed in 4.8 and 4.2 this addition of repair mechanism it provide a great reduction of Error Rate and Mean Relative Error. Following the test we have see that Partial Reconfiguration increase system performance. Although a portion of the design is being

reconfigured, the rest of the system continues to operate and with the combination of TMR this does not affect the output results. There is a serious amount of loss in the functionality of the design but the unaffected portions of the design give us the opportunity to have less down time. It can be easily observes that TMR combine with Partail Reconfiguration gains a great amount of Availability time of the System unlike TMR alone. As wem can see this Architecture is still in operation even after the 10k injections while in the TMR this is not possible. In 4.9 we can see that the reliability of the system is low due to the delayed start of the Reconfiguration process. This should be a key factor in the design of the next Architecture.

	Without TMR & PR2		With TMR & PR2	
Number Of Injection	Error Rate	Mean Relative Error	Error Rate	Mean Relative Error
10	95%	17.3%	10%	0 %
30	97.5%	37%	3.3%	0 %
60	98.57%	65%	18.8%	1.44%
100	99.09%	101%	29%	2.43 %
160	99.41%	153%	37.5%	3.2%
250	99.62%	228%	42%	4.5 %
500	99.8%	456%	56.2%	12.7%
800	99.8%	739%	64.5%	22.2 %
1000	-	-	66.5%	25.3 %
1500	-	-	69.9%	32.46 %
2500	-	-	72.9%	41.8%
5000	-	-	75%	54.3 %
10000	-	-	76%	67%

Table 4.8: Comparison of TMR & Partial Reconfiguration Architecture with in No Mitigation Technique in terms of Error Rate and Mean Relative Error.

Finally, it should be mentioned that a TMR ,with Partial Reconfiguration as repair mechanisms ,implemented in an SRAM-based FPGA is capable to protect from SEUs and DCEs as we can see in our experimental process in 4.8. In addition, unlike the simple TMR, reconfiguration is performed to remove the error module while application is still running since it is masked and detected by the voter. But any error that it is not detected by the voter ,as configuration bits related to the input routing that can be affected by bit flips, can impact all modules while they produce the same output (and therefore the voter do not detect a mismatch) but the outputs are wrong since the input values have been modified by the routing error. Also in the occurrence of multiple errors in the external voter ,which is responsible for the reconfiguration of the faulty area, the sequential reconfiguration of every module will start and as a result Error Rate,Mean Relative Error will begin to increase while the system reliability will begin to degrade as we see in 4.8.That is why is very critical to introduce a internal scrubber to repair silent

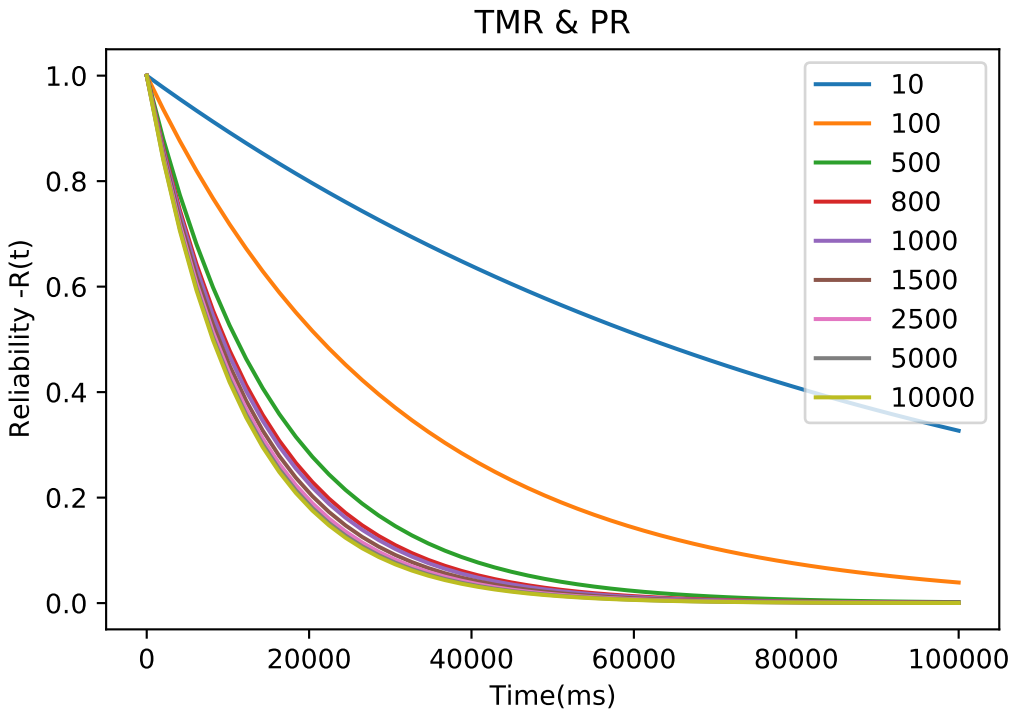


Figure 4.9: Reliability of PR and TMR: each function corresponds to # injections.

Errors which as shown to be very effective in the destruction of the design redundancy. This may cause tremendous problem in a image processing applications. Therefore, the proper operation of the TMR technique has to be preserved by adding an error removal technique (e.g., scrubbing) to prevent the accumulation of these Errors.

4.1.9 Internal Scrubber combine with Partial Reconfiguration and TMR

In 3.3.3 section we analysed the combination of Partial Reconfiguration, TMR and Internal scrubbing. Internal Scrubber and TMR should be considered as complementary methods for the mitigation of radiation effects. While TMR makes the device immune to single upsets of control and data processing block, scrubbing heals flipped bits in the configuration memory. It thus avoids accumulation of upsets which can lead to malfunction even if the design is triplicated(TMR). However the problem with those two Architecture is that SEM can only repair 50% of essential bits in a configuration memory another repair mechanism must be introduced. This is very critical in a competitive environment where injections are many and frequent because some of those Errors can modify the functionality of the design and drives into a false functionality of the design. If the applications is into a competitive environment (high injection ratio), a Fully or Partial reconfiguration must be introduced or a return to a checkpoint where the functionality of the application is still is intact.

As explained in Section 3.3.3 a design was created that implements this technique by

Number Of Injection	Without TMR & PR2 & SEM		With TMR & PR2 & SEM	
	Error Rate	Mean Relative Error	Error Rate	Mean Relative Error
10	95%	17.3%	10%	0%
30	97.5%	37%	3.3%	0%
60	98.57%	65%	1.6%	0%
100	99.09%	101%	2%	1.1%
160	99.41%	153%	1.8%	1.6%
250	99.62%	228%	1.2%	1.7%
500	99.8%	456%	1%	2%
800	99.8%	739%	7.5%	2.2%
1000	-	-	7%	2.4%
1500	-	-	6.7%	2.7%
2500	-	-	8%	3%
3600	-	-	7.5%	3.4%
5000	-	-	8.6%	3.7%
10000	-	-	30%	21%

Table 4.9: Comparison of SEM & PR & TMR Architecture with in No Mitigation Technique in terms of Error Rate and Mean Relative Error.

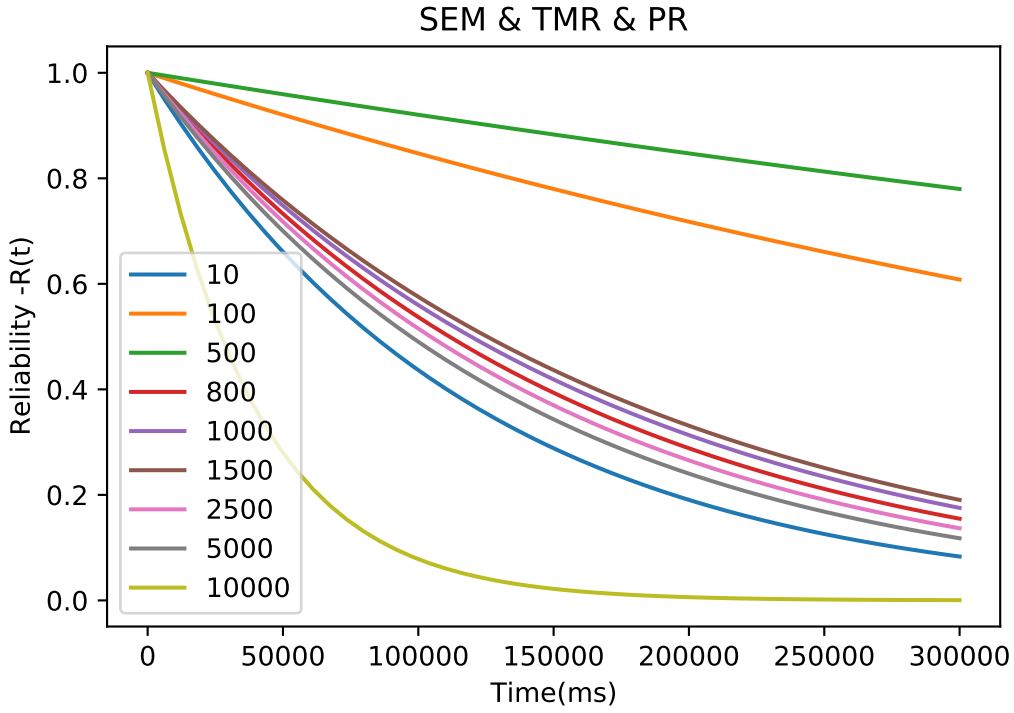


Figure 4.10: Reliability of SEM & PR & TMR: each function corresponds to # injections.

performing Partial Reconfiguration for three identical FIR circuits. This work demonstrates that it is possible to implement partial reconfiguration in systems that require scrubbing. SEM combine with TMR Architecture achieve a value of Error Rate at 36% while this Architecture is providing a max Error Rate of 30%. The difference of those Architectures can be spotted in Mean Relative Error as the quality of the output is much better than the previous one ,giving as the ability to achieve a more robust system. This Architecture achieves higher Reliability with the Partial reconfiguration ability. In 4.10 we see how the combination of these three Architecture affect our system concerning the Reliability of our design as well as the downtime of the system (see 4.12).

In conclusion the combination of TMR and Partial Reconfiguration with the internal configuration scrubber (SEM controller) is very effective in terms of reliability, availability and Error Rate (see in 4.10). This Architecture can repair silent Errors which as shown to be very effective in the destruction of the design redundancy as well as the complete area of a faulty module when it is necessary. Those capabilities are responsible for a robust Architecture that can be implemented in different environments with demanding injection ration. The problem with this Architecture is that there are certain situations in which the system fails, and the only way to solve this is to restore the entire configuration to a point at which the system was working as it should be.

4.1.10 Watchdog and Internal Scrubber combined with Partial Reconfiguration and TMR

Number of	Without TMR & PR2 & SEM &Watchdog		With TMR & PR2 & SEM &Watchdog	
Injection	Error Rate	Mean Relative Error	Error Rate	Mean Relative Error
10	95%	17.3%	10%	0 %
30	97.5%	37%	3.3%	0%
60	98.57%	65%	3.3%	0%
100	99.09%	101%	3%	4 %
160	99.41%	153%	2.5%	4.2 %
250	99.62%	228%	2%	4.2 %
500	99.8%	456%	1.6%	4.9 %
800	99.8%	739%	1.5%	5.4 %
1000	-	-	1.5%	5.7 %
1500	-	-	1.47%	6 %
2500	-	-	1.92%	6.5 %
3600	-	-	1.75%	6.9 %
5000	-	-	3%	7.2 %
10000	-	-	10%	7.8 %

Table 4.10: Comparison of SEM & TMR & PR & Watchdog Architecture with in No Mitigation Technique in terms of Error Rate and Mean Relative Error.

In 3.3.4 we described a fault-tolerant Architecture system which is based on multiple hierarchical layers of complementary schemes that are used to detect the faults and correct them. The basic components of the critical system are implemented using TMR. But since these controller circuits are defined by the configuration memory, which itself is sensitive to radiation errors, configuration memory scrubbing (SEM IP) is also implemented to protect this design. If any of the TMR processing module are permanent disable then the ICAP primitive through the Xilinx's AXI HWICAP IP core [18] should perform the reconfiguration of the faulty module. If also the ICAP or any basic component is hit by a permanent fault that compromise their functionality, then the software executed needs to re-configure the SoC-Fpga device via PCAP. If the watchdog timer expires without receiving the desire message then a reset of the FPGA must occur.

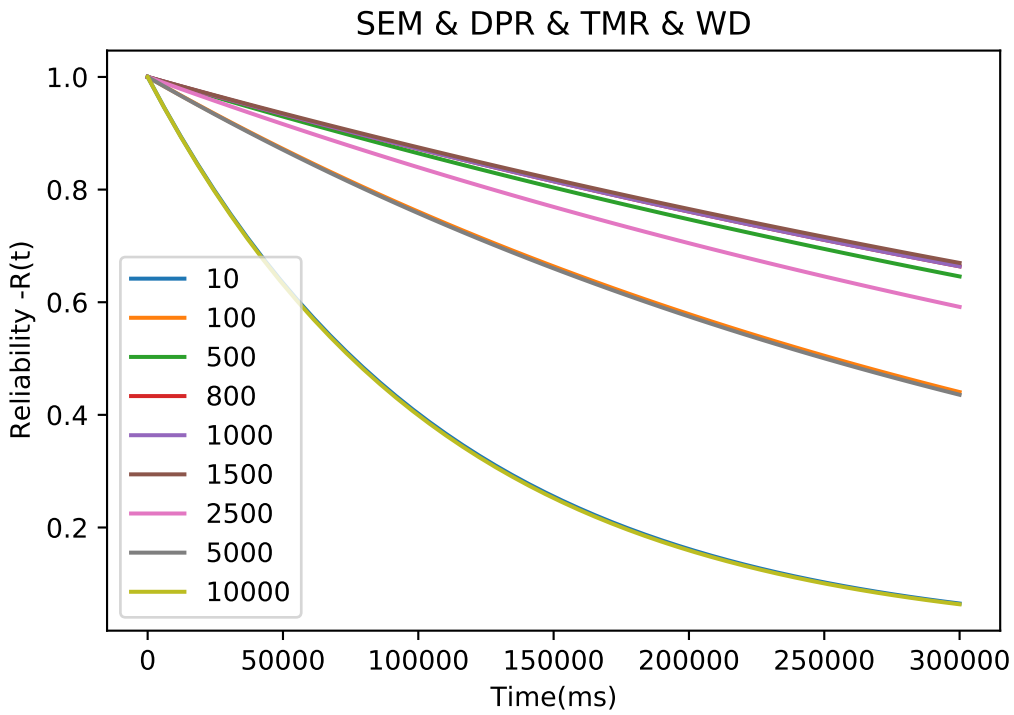


Figure 4.11: Reliability of SEM & PR & TMR & Watchdog : each function corresponds to # injections.

This Architecture is capable to correct the liability of TMR while using Partial reconfiguration as well as the halt of Internal Scrubber (SEM IP) while resetting it and also restoring if the Internal Scrubber is incapable to correct Critical Errors by Fully reconfigure all the device with the help of the Watchdog Timer. So we can see that this system solves the problems of every Architecture thus providing a robust system that can be established in a competitive environment (High injection ratio).

The Architecture analysed in this chapter as we see combine some of the most effective and efficient fault tolerance mitigation techniques in attempt to succeed the best result

possible. We observe in Figure in 4.11 this Architecture is the best possible in respect of Reliability. Also in the Table 4.10 we observe that that it reduces the Error Rate under 10% after 10k injections as well as provide a robust system that protect the quality of the output as we see in Mean Relative error (only 7.7%).

4.2 Comparisons

In this final section ,it will be discussed all the architecture together in respect to every evaluation metric. At first, we examine the functionality of the FPGA during our test. The results of Table 4.12 show the portion of time that the FPGA is unreachable (Downtime) and has erroneous/correct functionality. As shown, the standalone application provides downtime of the system in 92% of the application while every other Fault Tolerance alone improves the downtime of the system to 73% (Internal Scrubber,SEM IP), 79% (Partial Reconfiguration) and 80% (TMR). Also Internal Scrubber (SEM IP) exhibiting the correct functionality for more time (22%) while TMR and Partial Reconfiguration are limited to 1–8%. The advantage of Internal Srubber (SEM IP) is found in detecting errors in their creation as it checks the configuration memory and is not dependent on the output of the application. Furthermore, TMR provides near-zero correct functionality. This is happening because of the nature of our experiment. If the injections weren't specified into our component the TMR would have better results.

Architecture	**Test duration** (ms)	**Downtime** (ms)	**Non-Functional Time** (ms)	**Functional Time** (ms)
–	40000	36800	3180	20
TMR	40000	31878	7904	218
SEM	55000	40200	2800	12000
PR1	40956	32596	4592	3408
TMR & PR2	40642	14496	16642	9504
TMR & SEM	58830	18450	14000	26380
TMR & PR2 & SEM	54826	14826	12336	27664
TMR & PR2 & SEM &Watchdog	49896	9896	4028	35972

Table 4.11: Test time duration of all Architectures.

Nevertheless, when combining TMR with either Interanl scrubber(SEM IP) or PR, the functionality of the FPGA is radically improved. This improvement is due to the fact that errors are distributed across larger design 4.13 as well as the ability of Partial Reconfiguration to be combined with a detection system and of Internal Scrubber to mask the fault output through the triplication of the output. As we can observe The combination of all three Fault Tolerance techniques increases the reliability of the system even more, as the FPGA operates correctly for half of the test duration. Finally, the benefit of the combination including a Watchdog Timer is that, besides quickly detecting errors by both TMR and Internal Scrubber, it also captures the case that the Internal Scrubber (SEM IP) design has a permanent fault. As a result, it significantly increases the robustness of

FT Architecture	Downtime/ No Functionality	Erroneous Functionality	Correct Functionality
–	92%	8%	0%
TMR	80%	19%	1%
SEM	73%	5%	22%
PR1	79%	13%	8%
TMR & PR2	36%	41%	23%
TMR & SEM	31%	24%	45%
TMR & PR2 & SEM	27%	22%	51%
TMR & PR2 & SEM &Watchdog	20%	8%	72%

Table 4.12: Evaluation test time comparisons of all Architectures.

the system, providing 72% correct execution.

Architecture	Injections(max)	Mem. Confi + BRAMs Utilization	Mem Utiliz. Incr.	# Errors Detected	# Errors Corrected
–	800	3.8%+3.1%	x1	0	0
TMR	1500	8,3%+ 12.5%	x2.5	300	218
SEM	7000	6,7% +4.6%	x1.75	5013	5012
PR1	2000	23%+95%	x6	0	852
TMR & PR2	10000	6,1%+12.5%	x2.7	4000	2376
TMR & SEM	10000	10,6%+17.2%	x3.3	7883	6595
TMR & PR2 & SEM	10000	12,8% +17.2%	x4.1	9900	9100
TMR & PR2 & SEM &Watchdog	10000	17,5% +17.2%	x4.5	9956	9884

Table 4.13: Comparison Architecture in terms of Memory Utilization, Errors Detected and Corrected.

Based on the same results, Figure 4.12 demonstrates the reliability of the Fault Tolerance architectures over time. We assume exponentially-distributed random faults at rate λ and that the system reliability is defined as $R(t) = e^{-\lambda * t}$ [19]. The pattern remains the same, as the most reliable architectures are the combined one. However, for this metric, Partial Reconfiguration provides less reliability, because it lacks a detection mechanism, resulting in an almost always erroneous state.

Table 4.14 shows the utilization of the FPGA resources available for every mitigation Architecture. Regarding resources, all the Fault Tolerance architectures exhibit small utilization, i.e., less than 4K LUTs, 5K FFs, 6 RAMBs. The major difference between the Architectures designs can be observed in the utilization percentage of BRAM and IO of the Partial Reconfiguration and that's why we have implement our own Icap FSM which uses PL's BRAM to store the partial file responsible for the area of the FIR Filter. Otherwise we see a normal rise of FPGA's resources. Which is understandable as that increase is in tandem with the combination of multiple mitigation techniques in one design.

Therefore, there is room for the implementation of other components required in the co-processing architecture, e.g., more DSP functions, data transcoders and data compressors. In more detail, the SEM IP of the Internal Scrubber technique imposes the

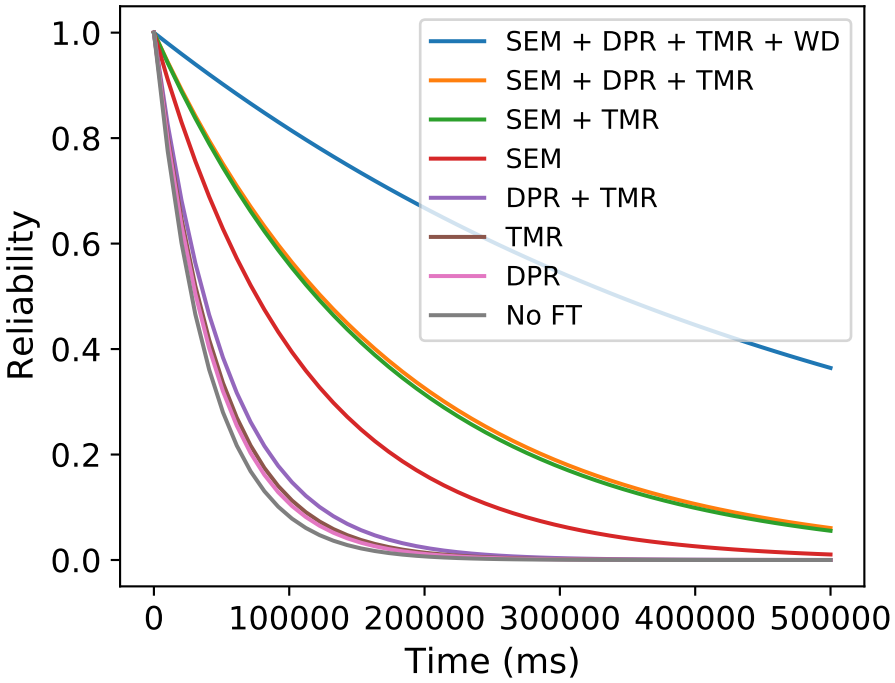


Figure 4.12: Reliability R(t) of all Architectures for 10K injections in the experiment operational time.

Architecture	LUTs	FFs	IOs	BRAMs
Non-FT FIR (app)	609	1461	4	1
TMR	1316	2327	11	4
SEM	2021	2761	1	1.5
PR1	1112	1756	59	57
PR2	1315	2501	4	0
TMR & PR2	2022	3367	11	4
TMR & SEM	2727	3801	12	5.5
TMR & PR2 & SEM	3433	4841	15	5.5
TMR & PR2 & SEM &Watchdog	3756	5072	17	5.5

Table 4.14: Comparison of all Architecture.

largest resource overhead (x2.5 more than other Fault Tolerance Techniques). The resource increment in FT architecture with the Watchdog Timer(x4.5) is due to the UART communication between the watchdog (microcontroller) and the FPGA, and the change configuration of SEM to "enhanced repair" mode instead of "replace". Finally, according to our measurements, SEM corrects a memory frame of 404 bytes in 18ms in enhanced repair and all the frames in 448 ms in replace mode at baud rate 9600, while the throughput of partial reconfiguration via HWICAP is 67 MB/s.

Architecture	Number of Injections			
	100	1000	5000	10000
–	99%	-	-	-
TMR	3%	78%	-	-
SEM	11%	47%	50%	-%
PR1	20%	49%	87%	91%
TMR & PR2	29%	66%	75%	76%
TMR & SEM	7%	8%	12%	34%
TMR & PR2 & SEM	2%	7%	8%	31%
TMR & PR2 & SEM &Watchdog	3%	1%	3%	10%

Architecture	Number of Injections			
	100	1000	5000	10000
–	101%	-	-	-
TMR	33 %	832%	-	-
SEM	0 %	8.6%	15%	-
PR1	7.5%	53%	150%	188%
TMR & PR2	2.4%	25.3%	54.3%	67%
TMR & SEM	5.9%	8.9 %	12 %	40%
TMR & PR2 & SEM	1.1%	2.4%	3.7%	21%
TMR & PR2 & SEM &Watchdog	4%	5.7%	7.2%	7.8%

Table 4.15: Comparison of all Architectures in terms of Error Rate (upper table) and Mean Relative Error (lower table)

The results presented in Table 4.15 suggest that standard, commercial SRAM-based FPGAs from Xilinx can indeed be used in some space applications. For suitable applications, such as non-critical instruments or communications, SRAM based FPGAs can provide a very cost-to-area-efficient alternative. TMR needs to be combined with scrubbing in order to be efficient, or it will eventually suffer from error buildup. The results presented in 4.15 apply to the selected test application, i.e., the FIR block. Applications with natural downtime windows can benefit from simple periodic scrubbing. Partial-reconfiguration in conjunction with TMR constitutes a very efficient fault tolerance alternative, but there is significant resource overhead and supervision limited to the areas where partial reconfiguraiton is performed. The internal Scrubber based on the Xilinx SEM Controller represents a resource-efficient choice that can be combined with externally-triggered full reconfiguration and provides a sufficient solution for a robust Fault Tolerance System.

Chapter 5

Conclusion and Future Work

Conclusion

The experimental evaluation scenarios discussed in this paper show that commercial SRAM-based FPGAs can be used in space applications, but both user data and configuration memory have to be protected. Our proof-of-concept is designed in an FPGA-SoCs with ARM Cortex-A9 cores widely used in, while the reproducible design variant described in this contribution uses Programmable Logic to apply mitigation techniques. It can be replicated with just standard design tools and library IP, which are available in low cost to many designers in academic and research organizations. Therefore, our Architecture scales with technology, instead of struggling against it. It benefits from performance and energy efficiency improvements that can be achieved through improved technology nodes, and scales for designs with more, and powerful processor cores.

This Thesis exploit four Mitigation techniques Internal configuration Scrubber, TMR, Dynamic Partial Reconfiguration and Watchdog Timer . The experiment tested all of SEU mitigation techniques from unmitigated design to the fully mitigated design. Seven SEU mitigation variations were tested: without SEU mitigation, TMR alone,Dynamic Partial Reconfiguration,Internal configuration scrubbing(via SEM IP) TMR with Dynamic Partial Reconfiguration, TMR with Internal configuration scrubber, and TMR with both Partial Reconfiguration and Internal configuration scrubber as well all of this with a Watchdog Timer. Testing these combinations helped isolate the benefits of each SEU mitigation technique and explain the complementary relationships between them. Fault injection were conducted. Improvement is measured in terms of Reliability improvement and also Error Rate and Mean Relative Error reduction in the same fault injection experiment. The results from fault injection demonstrate that each variation of SEU mitigation techniques improve the SEU sensitivity of the Fault Tolerance system, and that improvement increases as more mitigation techniques are combined. Improvement gained from Internal configuration scrubbing is greater than the other mitigation techniques. When all four SEU mitigation techniques are combined, x72 improvement is observed by the test methods. Something that we did not observe in our thesis was the increased sensitivity due to the mitigation logic added to the design. This was expected due to the nature of our

experiment and the fact that we already targeted sensitive bits and not all the FPGA's Memory.

The architecture analysed in this Thesis combine some of the most effective and efficient fault tolerance mitigation techniques in attempt to succeed the best result possible. Below in 5.1 there are the mitigation techniques that are analyzed in this Thesis as well as the capabilities of every Architecture.

Architecture	Detection	Correction	Error Types	Device Reboot
No Mitigation Technique	No	No	-	No
TMR	Yes	Yes	Transient	No
SEM	Yes	Yes	Transient & Permanent	No
PR	No	Yes	Transient & Permanent	No
TMR & PR	Yes	Yes	Transient & Permanent	No
TMR & SEM	Yes	Yes	Transient & Permanent	No
TMR & PR &SEM	Yes	Yes	Transient & Permanent& SEFI	YEs
TMR &PR &SEM & Watchdog Timer	Yes	Yes	Transient & Permanent& SEFI	YEs

Table 5.1: Comparison of all SEU mitigation Architectures

Future Work

There are several things that can be taken up as an extension of the work presented in the thesis.

- Implement SEUs mitigation techniques such as ECC, CRC, TMR and Memory Scrubbing in BRAMs. For instance, the on-chip BRAMs and the off-chip DRAMs can be protected using a TMR memory controller, that writes the same data to three different locations and votes on them while reading to see if any of them has been corrupted. The corrupted data is re-written with the majority voted data.

- In respect to the External Memory, different memories type can be tested to speed up the correction time including a direct SPI connection, SD card, local DDR memory, or hardware shim to directly read the replacement data .

- A worthy investigation would be to use an external Scrubber for a different device that would have access to the configuration memory or using the Second Arm-Cortex Processor to be the external Scrubber with the help of PCAP. The second Arm-Cortex Processor can also be used to apply the reconfiguration on the Programmable Logic while the other is responsible for controlling the Internal configuration Scrubber(SEM IP).

- Regarding evaluation processes it would be ideal to use wide spectrum neutron beam for testing of integrated circuits. Although more expensive and difficult to conduct, radiation testing provides a more accurate estimate of the sensitivity of the design to single-event upsets since radiation testing upsets all of the internal memory of the FPGA as well as single event transients (SET).

Bibliography

[1] S. E. M. C. L. IP, "Product guide (pg036)," *Xilinx, San Jose, CA, USA*, 2015.

[2] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and materials reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[3] F. Wang and V. D. Agrawal, "Single event upset: An embedded tutorial," in *21st International Conference on VLSI Design (VLSID 2008)*. IEEE, 2008, pp. 429–434.

[4] P. Adell, G. Allen, G. Swift, and S. McClure, "Assessing and mitigating radiation effects in xilinx sram fpgas," in *2008 European Conference on Radiation and Its Effects on Components and Systems*. IEEE, 2008, pp. 418–424.

[5] R. Katz, "A scientific study of the problems of digital engineering for space flight systems, with a view to their practical solution," *NASA Office of Logic Design, Tech. Rep*, 2005.

[6] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527–1538, 2010.

[7] X. Iturbe, M. Azkarate, I. Martinez, J. Perez, and A. Astarloa, "A novel seu, mbu and she handling strategy for xilinx virtex-4 fpgas," in *2009 International Conference on Field Programmable Logic and Applications*. IEEE, 2009, pp. 569–573.

[8] M. Berg, H. Kim, M. Friendlich, C. Perez, C. Seidleck, K. LaBel, and R. Ladbury, "Seu analysis of complex circuits implemented in actel rtax-s fpga devices," *IEEE Transactions on Nuclear Science*, vol. 58, no. 3, pp. 1015–1022, 2011.

[9] J. Jedec, "Measurement and reporting of alpha particles and terrestrial cosmic ray-induced soft errors in semiconductor devices: Jesd89a," *JEDEC STANDARD, JEDEC Sold State Technology Association, No. 89*, pp. 1–85, 2006.

[10] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-event upset mitigation selection guide," *Xilinx Application Note, XAPP987 (v1. 0)*, p. 69, 2008.

[11] M. R. Gardiner, *An evaluation of soft processors as a reliable computing platform.* Brigham Young University, 2015.

[12] S. Lee, R. Kjar, J. Peel, and G. Kinoshita, "Radiation-hardened silicon-gate cmos/-sos," *IEEE Transactions on Nuclear Science*, vol. 24, no. 6, pp. 2205–2208, 1977.

[13] Xilinx, "7 series frame ecce2 port descriptions and functionality." 2016. [Online]. Available: http://www.xilinx.com/support/answers/54350.html

[14] N. Rollins, M. Fuller, and M. J. Wirthlin, "A comparison of fault-tolerant memories in sram-based fpgas," in *2010 IEEE Aerospace Conference.* IEEE, 2010, pp. 1–12.

[15] G. Miller, C. Carmichael, and G. Swift, "Single-event upset mitigation for xilinx fpga block memories," *XILINX Application Note*, 2007.

[16] A. M. Saleh, J. J. Serrano, and J. H. Patel, "Reliability of scrubbing recovery-techniques for memory systems," *IEEE transactions on reliability*, vol. 39, no. 1, pp. 114–122, 1990.

[17] I. Herrera-Alzu and M. Lopez-Vallejo, "Design techniques for xilinx virtex fpga configuration memory scrubbers," *IEEE transactions on Nuclear Science*, vol. 60, no. 1, pp. 376–385, 2013.

[18] A. H. IP, "Product guide (pg134)," *Xilinx, San Jose, CA, USA*, 2016.

[19] F. Afsharnia, "Failure rate analysis," in *Failure Analysis and Prevention.* IntechOpen, 2017.

[20] C. Urbina-Ortega, G. Furano, G. Magistrati, K. Marinis, A. Menicucci, and D. Merodio-Codinachs, "Flash-based fpgas in space, design guidelines and trade-off for critical applications," in *14th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2013, pp. 1–8.

[21] K. Maragos, V. Leon, G. Lentaris, D. Soudris, D. Gonzalez-Arjona, R. Domingo, A. Pastor, D. M. Codinachs, and I. Conway, "Evaluation Methodology and Reconfiguration Tests on the New European NG-MEDIUM FPGA," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 127–134.

[22] V. Leon, I. Stamoulias, G. Lentaris, D. Soudris, R. Domingo, M. Verdugo, D. Gonzalez-Arjona, D. M. Codinachs, and I. Conway, "Systematic Evaluation of the European NG-LARGE FPGA & EDA Tools for On-Board Processing," in *European Workshop on On-Board Data Processing (OBDP)*, 2021, pp. 1–8.

[23] V. Leon, I. Stamoulias, G. Lentaris, D. Soudris, D. Gonzalez-Arjona, R. Domingo, D. M. Codinachs, and I. Conway, "Development and Testing on the European Space-Grade BRAVE FPGAs: Evaluation of NG-Large Using High-Performance DSP Benchmarks," *IEEE Access*, vol. 9, pp. 131 877–131 892, 2021.

[24] A. D. George and C. M. Wilson, "Onboard Processing with Hybrid and Reconfigurable Computing on Small Satellites," *Proceedings of the IEEE*, vol. 106, no. 3, pp. 458–470, 2018.

[25] C. Wilson and A. George, "CSP Hybrid Space Computing," *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 215–227, 2018.

[26] A. Pérez, A. Rodríguez, A. Otero, D. González-Arjona, A. Jiménez-Peralo, M. A. Verdugo, and E. De La Torre, "Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation," *IEEE Access*, vol. 8, pp. 59 891–59 905, 2020.

[27] V. Leon, G. Lentaris, D. Soudris, S. Vellas, and M. Bernou, "Towards Employing FPGA and ASIP Acceleration to Enable Onboard AI/ML in Space Applications," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–4.

[28] F. C. Bruhn, N. Tsog, F. Kunkel, O. Flordal, and I. Troxel, "Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space," *CEAS Space Journal*, vol. 12, pp. 551–564, 2020.

[29] V. Leon, C. Bezaitis, G. Lentaris, D. Soudris, D. Reisis, E.-A. Papatheofanous, A. Kyriakos, A. Dunne, A. Samuelsson, and D. Steenari, "FPGA & VPU Co-Processing in Space Applications: Development and Testing with DSP/AI Benchmarks," in *28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–5.

[30] V. Leon, G. Lentaris, E. Petrongonas, D. Soudris, G. Furano, A. Tavoularis, and D. Moloney, "Improving Performance-Power-Programmability in Space Avionics with Edge Devices: VBN on Myriad2 SoC," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 3, pp. 1–23, 2021.

[31] F. Brosser, E. Milh, V. Geijer, and P. Larsson-Edefors, "Assessing scrubbing techniques for xilinx sram-based fpgas in space applications," in *2014 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2014, pp. 296–299.

[32] U. Guide, "Series fpgas configurable logic block," *Xilinx, San Jose, CA*, vol. 1, no. 7, 7.

[33] D. P. Schultz, L. C. Hung, and F. E. Goetting, "Fpga configuration circuit including bus-based crc register," Feb. 20 2001, uS Patent 6,191,614.

[34] X. U. Guide, "Series fpgas configuration," *UG470, v1*, vol. 11, pp. 1–176, 7.

[35] C. D. Patterson, P. Sundararajan, B. J. Blodget, and S. P. McMillan, "Method and system for identifying essential configuration bits," Jul. 29 2008, uS Patent 7,406,673.

[36] R. Le, "Soft error mitigation using prioritized essential bits," *Xilinx XAPP538 (v1. 0)*, p. 72, 2012.

[37] G. Santin, P. Truscott, R. Gaillard, and R. G. Alía, "Radiation environments: space, avionics ground and below," *RADECS 2017 Short Course Notebook*, 2017.

[38] F. Faccio, "Radiation issues in the new generation of high energy physics experiments," in *Radiation Effects and Soft Errors in Integrated Circuits and Electronic Devices*. World Scientific, 2004, pp. 95–115.

[39] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of radiation effects in sram-based fpgas for space applications," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, pp. 1–34, 2015.

[40] J. L. T. Seclen, "Frame-level redundancy scrubbing technique for sram-based fpgas," 2015.

[41] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "Fpga partial reconfiguration via configuration scrubbing," in *2009 International Conference on Field Programmable Logic and Applications*. IEEE, 2009, pp. 99–104.

[42] M. Welter, "Demonstration of soft error mitigation ip and partial reconfiguration capability on monolithic devices," *Jun-2015*, 2015.

[43] M. Berg and K. LaBel, "New developments in error detection and correction strategies for critical applications," in *Single Event Effects (SEE) Symposium and Military and Aerospace Programmable Logic Devices (MAPLD) Workshop*, no. GSFC-E-DAA-TN42793, 2017.

[44] D. R. Czajkowski, P. K. Samudrala, and M. P. Pagey, "Seu mitigation for reconfigurable fpgas," in *2006 IEEE Aerospace Conference*. IEEE, 2006, pp. 7–pp.

[45] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of internal versus external seu scrubbing mitigation strategies in a xilinx fpga: Design, test, and analysis," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2259–2266, 2008.

[46] P. Pirsch, N. Demassieux, and W. Gehrke, "Virtex-4 family overview," *Product Specification, Xilinx, DS112 (v3. 1) August*, vol. 30, 2010.

[47] M. R. Ersdal, J. Almea, M. Bonorac, P. Giubilatod, M. Lupic, S. V. Nesbøa, A. U. Rehmana, D. Röhricha, G. Aglieri, J. S. Rinellac *et al.*, "External scrubber implementation for the alice its readout unit," in *Topical Workshop on Electronics for Particle Physics TWEPP2019*, vol. 2, 2019, p. 6.

[48] K. Røed, J. Alme, D. Fehlker, M. Richter, K. Ullaland, D. Röhrich, and H. Helstrup, "Radiation tolerance of an sram based fpga used in a large tracking detector," *Large Scale Applications and Radiation Hardness of Semiconductor Detectors*, p. 43, 2009.

[49] A. Gruwell, P. Zabriskie, and M. Wirthlin, "High-speed fpga configuration and testing through jtag," in *2016 IEEE AUTOTESTCON*. IEEE, 2016, pp. 1–8.

[50] H. Michel, A. Belger, T. Lange, B. Fiethe, and H. Michalik, "Read back scrubbing for sram fpgas in a data processing unit for space instruments," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2015, pp. 1–8.

[51] K. U. F. D. Sheet, "Dc and ac switching characteristics," *Xilinx Inc. DS893 (v1. 7.1) April*, vol. 4, 2015.

[52] C. Kohn, "Partial reconfiguration of a hardware accelerator with vivado design suite for zynq-7000 ap soc processor," *XILINX Application Notes*, 2015.

[53] A. Stoddard, A. Gruwell, P. Zabriskie, and M. J. Wirthlin, "A hybrid approach to fpga configuration scrubbing," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 497–503, 2016.

[54] M. Wirthlin and A. Harding, "Hybrid configuration scrubbing for xilinx 7-series fpgas," in *FPGAs and Parallel Architectures for Aerospace Applications*. Springer, 2016, pp. 91–101.

[55] A.-D. Oancea, C. Stuellein, J. Gebelein, and U. Kebschull, "A resilient, flash-free soft error mitigation concept for the cbm-tof read-out chain via gbt-sca," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 1–4.

[56] A. Caratelli, S. Bonacini, K. Kloukinas, A. Marchioro, P. Moreira, R. De Oliveira, and C. Paillard, "The gbt-sca, a radiation tolerant asic for detector control and monitoring applications in hep experiments," *Journal of Instrumentation*, vol. 10, no. 03, p. C03034, 2015.

[57] N. Storey, *Safety-critical computer systems*, 1996.

[58] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through ceu (code emulating upsets) injection," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2405–2411, 2000.

[59] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin, "Estimating soft processor soft error sensitivity through fault injection," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 143–150.

[60] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.

[61] R. Le, "Soft error mitigation using prioritized essential bits, april 2012," *Application Note, XAPP538*.

[62] L. A. Aranda, A. Sánchez-Macián, and J. A. Maestro, "Acme: A tool to improve configuration memory fault injection in sram-based fpgas," *IEEE Access*, vol. 7, pp. 128 153–128 161, 2019.

[63] D. S. Lee, M. P. King, W. L. Evans, M. Cannon, A. Perez-Celis, J. Anderson, M. Wirthlin, and W. C. Rice, "Single-event characterization of 16 nm finfet xilinx ultrascale+ devices with heavy ion and neutron irradiation." 7 2018. [Online]. Available: https://www.osti.gov/biblio/1570816