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

Error Mitigation Techniques on VDHL circuits and SoC FPGAs

Διπλωματική Εργασία

του

ΠΑΓΩΝΗ ΓΕΩΡΓΙΟΥ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα,Μάρτιου 2023

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

Error Mitigation Techniques on VDHL circuits and SoC FPGAs

Διπλωματική Εργασία

του

ΠΑΓΩΝΗ ΓΕΩΡΓΙΟΥ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21 Μάρτιου 2023.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Σούντρης	Διονύσιος Πνευματικάτος	Διονύσιος Ρέϊσης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Κ.Π.Α.

Αθήνα,Μάρτιου 2023

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

(Υπογραφή)

.....

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

Περίληψη

Η διαστημική βιομηχανία τα τελευταία χρόνια έχει υποστεί μετασχηματισμό, μεταβαίνοντας από τα παραδοσιακά space-grade συστήματα, σε mixed architectures, που συνδυάζουν radiation-hardened components και Commercial-off-the-Shelf (COTS) συσκευές, μεταξύ των οποίων τα FPGAs. Τα πλεονεκτήματα αυτών των components είναι το SWaP-C (size, weight, power and cost), η απόδοση στην επεξεργασία και η ευελιξία ανάπτυξης των εφαρμογών, σε σύγκριση με τις κλασικές αρχιτεκτονικές που αδυνατούν να καλύψουν αποτελεσματικά τις αυξημένες απαιτήσεις. Ωστόσο, οι εμπορικές συσκευές είναι επιρρεπείς σε fault λόγω ιονίζουσας ακτινοβολίας. Τα Single Event Upsets (SEU), που αλλάζουν την λειτουργικότητα, μετριάζουμε σε αυτή την διπλωματική. Όλες οι fault-mitigation τεχνικές εφαρμόζονται στην MPSoC Ultrascale+ αρχιτεκτονική, που χρησιμοποιείται σε διαστημικές αποστολές. Αναπτύξαμε μια custom hardware architecture ενός Fast Fourier Transform αλγορίθμου, που θα χρησιμοποιηθεί ως το βασικό μας DSP component. Οι fault-mitigation αρχιτεκτονικές που εφαρμόσαμε είναι τόσο ανεξάρτητες της εφαρμογής, όσο και συγκεκριμένα για τον FFT, ενώ οι εκστρατείες fault injection και evaluation, χρησιμοποιούνται για την κατηγοριοποίηση και την ταξινόμηση των τεχνικών μας με βάση την μείωση του downtime της συσκευής. Εκτός αυτού, διερευνήσαμε τον αντίκτυπο της χρήσης διαφορετικών computational FPGA block, δηλαδή DSP, LUT στην αξιοπιστία την συσκευής. Οι τεχνικές error-mitigation ήταν Spatial (DMR, TMR), Temporal, Hybrid (DMR Temporal), καθώς επίσης και fine-grained redundancy(TMR σε Stage, Butterfly). Για την διόρθωση αυτών των component, συμπεριλάβαμε στην σχεδίαση μας την πλήρη και μερική επαναδιαμόρφωση (Dynamic Partial Reconfiguration) και τον internal memory scrubber που προσφέρετε από το SEM IP από την Xilinx. Η προσομοίωση γίνεται με την προσθήκη bit-flips στην configuration μνήμη. Για τον αντίκτυπο της χρήσης των διαφορετικών υπολογιστικών FPGA block, με την υλοποίηση με LUT να είχε αυξημένο downtime 2.2×, 3.68×, 2.7× όταν συγκινήθηκε με την υλοποίησε με DSP, για 8,16,32-point FFT. Οι καλύτερες αρχιτεκτονικές για την μείωση του downtime, είναι η temporal, λόγω του μικρού μεγέθους και την ανίχνευση σφαλμάτων που προσφέρει, και το TMR λόγω του Error Detection & Correction. Η καλύτερη βελτίωση στην μείωση του downtime είναι 95% και 65% στο 8,32-point FFT, καθώς και οι δύο αρχιτεκτονικές έχουν εκμεταλλευτεί όλες τις μεθοδολογίες διόρθωσης (FR, PR, CMS).

Λέξεις Κλειδιά

COTS FPGA, MPSoC UltraScale+, SEU Error Injection, SEM IP, Spatial | Temporal | Hybrid | Fine-Grained Redunancy, Full|Partial Reconfiguration, CMS

Abstract

The space industry has undergone a transformation in recent years, shifting from classic space-grade systems to mixed architecture that combines radiation-hardened components and Commercial-off-the-Shelf (COTS) devices, including FPGAs. The advantages of the COTS components are the SWaP-C (size, weight, power, and cost), the processing performance, and the development flexibility when compared with the classical-computational architectures, which stress meeting the increased data and computational demands effectively. However, commercial devices are susceptible to ionizing radiation failures. The Single Event Upsets (SEU), that alter the functionality are those, we try to mitigate in this thesis. All the fault-tolerance techniques are implemented in the MPSoC Ultascale+ Architecture, which is commonly used in space missions. We developed a custom hardware architecture for a Fast Fourier Transform Algorithm to be used as our digital signal processing (DSP) component. The fault-mitigation architectures are both application-independent and application-specific, while the fault injection and evaluation campaign we developed, is used to categorize and sort our techniques based on the reduction of the downtime of our device. Besides that, we explored the impact of utilizing different FPGA blocks, i.e. DSP and LUT on the reliability of the device. The fault mitigation techniques implemented in the Hardware accelerator are spatial(i.e. DMR, TMR), temporal, hybrid between the spatial and temporal (i.e. DMR Temporal), and in addition and fine-grain spatial redundancy (i.e TMR in Stage and Butterfly). For the module correction, we included in our design the full and partial reconfiguration (i.e. Dynamic Partial Reconfiguration), and the internal scrubber offered by Xilinx's SEM IP. The injection is performed also using the SEM IP, by adducing bit-flip in the configuration memory. The injection campaign involved 200,000 configuration memory addresses. As far as the impact of different FPGA computational blocks is considered, the LUT implementation suffered from increased downtime $2.2\times$, $3.68\times$, $2.7\times$, when comparing to the DSP implementation for the 8,16,32-point FFT respectively. The best reduction architectures are the temporal, due to its small resource utilization and Error Detection, as well as the TMR which offered Error Detection & Correction. The best improvement in downtime was the reduction of 95% and 65% in 8 and 32-point FFT, as both architectures have taken advantage of all the correction methodologies (FR, PR, CMS).

Keywords

COTS FPGA, MPSoC UltraScale+, SEU Error Injection, SEM IP, Spatial | Temporal | Hybrid | Fine-Grained Redunancy, Full|Partial Reconfiguration, CMS

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή κύριο Δημήτριο Σούντρη για την ευκαιρία και την εμπιστοσύνη που μου έδωσε να εκπονήσω την διπλωματική μου εργασία στο εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων. Επίσης ευχαριστώ ιδιαίτερα τον διδάκτορα Βασίλη Λέων και τον καθηγητή Γιώργο Λεντάρη για την καθοδήγησή τους, την ευκαιρία να αναπτύξω τις δικές μου ιδέες, την εξαιρετική και άψογη συνεργασία που είχαμε. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου, Ευθαλία Κοσμά και Παναγιώτη Παγώνη, για την καθοδήγηση και την ηθική συμπαράσταση που μου προσέφεραν όλα αυτά τα χρόνια. Ένα μεγάλο ευχαριστώ στους φίλους μου και σε όσους με βοήθησαν κατά την διάρκεια των φοιτητικών μου χρόνων.

Μάρτιος 2023

Παγώνης Γεώργιος

Contents

List of Figures

List of Tables

Εκτεταμένη Περίληψη

Introduction & Background

Το διάστημα αποτελεί σημαντική απειλή για τις ηλεκτρονικές συσκευές λόγω της παρουσίας ιονίζουσας ακτινοβολίας. Η εν λόγω ακτινοβολία μπορεί να αυξήσει την αγωγιμότητα των υλικών, οδηγώντας σε βλαβερά επίπεδα ρεύματος και ενδεχομένως να προκαλέσει μόνιμες ζημιές ή σφάλματα λειτουργίας στον ηλεκτρονικό εξοπλισμό. Ως αποτέλεσμα, οι ημιαγωγοί είναι ιδιαίτερα ευαίσθητοι σε αυτά τα φαινόμενα, τα οποία μπορούν να προκαλέσουν μοναδικά συμβάντα διαταραχής (Single Event Upsets - SEUs) στα ψηφιακά κυκλώματα, οδηγώντας σε θανατηφόρα σφάλματα στις αεροδιαστημικές εφαρμογές [10]. Για την προστασία των κυκλωμάτων σε αυτές τις εφαρμογές, συνήθως χρησιμοποιούνται τεχνικές αντοχής και ανθεκτικότητας στην ακτινοβολία. Ωστόσο, αυτές οι τεχνικές είναι κοστοβόρες και ενδέχεται να μην είναι εφικτές για χρήση σε μικρότερους δορυφόρους, όπως οι CubeSats.

Τα FPGAs που είναι εμπορικά διαθέσιμα (COTS) έχουν μια μοναδική θέση στην αντιμετώπιση των δυσκολιών που σχετίζονται με τον σχεδιασμό και τη λειτουργία των εφαρμογών διαστήματος. Αυτό οφείλεται στην ικανότητά τους να αναδιαρθρώνονται εν κινήσει και να προσαρμόζονται στο αυστηρό διάστημα, συνεχίζοντας τη λειτουργία τους αντιμετωπίζοντας βλάβες που προκαλούνται από ακτινοβολία. Η ακτινοβολία μπορεί να προκαλέσει πολλαπλές βλάβες σε ηλεκτρονικές συσκευές που χρησιμοποιούνται στο διάστημα, όπως παρατεταμένες επιπτώσεις και κορεσμό των ηλεκτρονικών κυκλωμάτων. Ως αποτέλεσμα, οι εταιρείες έχουν αναπτύξει FPGAs που είναι ανθεκτικές στην ακτινοβολία, παρέχοντας στους σχεδιαστές ανθεκτικές και πιστοποιημένες συσκευές που πληρούν τις απαιτητικές απαιτήσεις απόδοσης, αξιοπιστίας και διάρκειας ζωής των εφαρμογών διαστήματος.

Τα εμπορικά διαθέσιμα (COTS) FPGAs, λόγω της ικανότητάς τους να αναδιαρθρώνονται κατά τη διάρκεια της λειτουργίας, έχουν μια μοναδική θέση για να αντιμετωπίσουν αυτές τις δυσκολίες. Η ικανότητά τους να αναδιαρθρώνονται εν κινήσει τους επιτρέπει να προσαρμόζονται στο αυστηρό διάστημα και να συνεχίζουν τη λειτουργία τους αντιμετωπίζοντας βλάβες που προκαλούνται από ακτινοβολία. Ως αποτέλεσμα, οι εταιρείες έχουν αναπτύξει FPGAs ανθεκτικά στην ακτινοβολία, παρέχοντας στους σχεδιαστές ανθεκτικές και πιστοποιημένες συσκευές που πληρούν τις απαιτήσεις απόδοσης, αξιοπιστίας και διάρκειας ζωής των εφαρμογών διαστήματος. Επιπλέον, η συνεχής εξέλιξη των τεχνολογιών σχεδίασης FPGA συνεπάγεται ότι πρέπει να αναπτύσσεται συνεχώς νέες μεθόδους και προσεγγίσεις σχεδίασης για να διατηρούμε την ανθεκτικότητά τους. Αυτό μπορεί να περιλαμβάνει την εκμετάλλευση νέων υλικών και τεχνολογιών, όπως η χρήση προηγμένων τεχνικών σχεδίασης για τη μείωση των επιδράσεων της ακτινοβολίας, καθώς και την ανάπτυξη νέων αλγορίθμων για την ανίχνευση και την αντιμετώπιση σφαλμάτων.

Τα FPGAs που είναι εμπορικά διαθέσιμα (COTS) έχουν μια μοναδική θέση στην αντιμετώπιση των δυσκολιών που σχετίζονται με τον σχεδιασμό και τη λειτουργία των εφαρμογών διαστήματος. Αυτό οφείλεται στην ικανότητά τους να αναδιαρθρώνονται εν κινήσει και να προσαρμόζονται στο αυστηρό διάστημα, συνεχίζοντας τη λειτουργία τους αντιμετωπίζοντας βλάβες που προκαλούνται από ακτινοβολία.Η ακτινοβολία μπορεί να προκαλέσει πολλαπλές βλάβες σε ηλεκτρονικές συσκευές που χρησιμοποιούνται στο διάστημα, όπως παρατεταμένες επιπτώσεις και κορεσμό των ηλεκτρονικών κυκλωμάτων. Ως αποτέλεσμα, οι εταιρείες έχουν αναπτύξει FPGAs που είναι ανθεκτικές στην ακτινοβολία, παρέχοντας στους σχεδιαστές ανθεκτικές και πιστοποιημένες συσκευές που πληρούν τις απαιτητικές απαιτήσεις απόδοσης, αξιοπιστίας και διάρκειας ζωής των εφαρμογών διαστήματος.
Τα εμπορικά διαθέσιμα (COTS) FPGAs, λόγω της ικανότητάς τους να αναδιαρθρώνονται κατά τη διάρκεια της λειτουργίας, έχουν μια μοναδική θέση για να αντιμετωπίσουν αυτές τις δυσκολίες. Η ικανότητά τους να αναδιαρθρώνονται εν κινήσει τους επιτρέπει να προσαρμόζονται στο αυστηρό διάστημα και να συνεχίζουν τη λειτουργία τους αντιμετωπίζοντας βλάβες που προκαλούνται από ακτινοβολία. Ως αποτέλεσμα, οι εταιρείες έχουν αναπτύξει FPGAs ανθεκτικά στην ακτινοβολία, παρέχοντας στους σχεδιαστές ανθεκτικές και πιστοποιημένες συσκευές που πληρούν τις απαιτήσεις απόδοσης, αξιοπιστίας και διάρκειας ζωής των εφαρμογών διαστήματος. Επιπλέον, η συνεχής εξέλιξη των τεχνολογιών σχεδίασης FPGA συνεπάγεται ότι πρέπει να αναπτύσσεται συνεχώς νέες μεθόδους και προσεγγίσεις σχεδίασης για να διατηρούμε την ανθεκτικότητά τους. Αυτό μπορεί να περιλαμβάνει την εκμετάλλευση νέων υλικών και τεχνολογιών, όπως η χρήση προηγμένων τεχνικών σχεδίασης για τη μείωση των επιδράσεων της ακτινοβολίας, καθώς και την ανάπτυξη νέων αλγορίθμων για την ανίχνευση και την αντιμετώπιση σφαλμάτων.

Συμβολές διπλωματικής

Η παρούσα διπλωματική εργασία παρέχει σημαντικές συνεισφορές:

- Στοχεύουμε στα MPSoC UltraScale+ FPGA τα οποία είναι όλο και πιο διαδεδομένα σε συστήματα υψηλής αξιοπιστίας για το διάστημα, όπως τα Q8S και Leopard, ενώ η πλειονότητα των ερευνητικών εργασιών επικεντρώνεται στα Zynq FPGAs.

- Πραγματοποιήθηκαν δοκιμές με διαφορετικές μεθοδολογίες αντιμετώπισης, ανεξάρτητα από τον πυρήνα επεξεργασίας FFT, προκειμένου να αξιολογηθεί η αποτελεσματικότητά τους.

- Πραγματοποιήθηκαν δοκιμές με διαφορετικές μεθοδολογίες αντιμετώπισης, ανεξάρτητα από τον πυρήνα επεξεργασίας FFT, προκειμένου να αξιολογηθεί η αποτελεσματικότητά τους.

- Επεκτείνοντας και αναπτύσσοντας μια μεθοδολογία εγχύσεων,

- Η αξιοπιστία του αλγορίθμου FFT για διαφορετικά μήκη εισόδου (8/16/32).

- Διερευνούμε τον αντίκτυπο της χρήσης διαφορετικών μπλοκ FPGA για υπολογισμούς, όπως DSPs και LUTs, στην αξιοπιστία της συσκευής.

Υλοποίηση Fast Fourier Transform αλγορίθμου στο hardware

Στο πλαίσιο της παρούσας διπλωματικής εργασίας, έχουμε αναλύσει και υλοποιήσει έναν αλγόριθμο FFT (Fast Fourier Transform) βασισμένο στον αλγόριθμο του Cooley-Tukey. Η υλοποίηση αυτή βασίζεται στην τεχνική divide and conquer, η οποία επιτρέπει τον διαχωρισμό ενός μεγάλου προβλήματος σε μικρότερα, πιο απλά υποπροβλήματα. Αποτέλεσμα αυτού είναι η παραγωγή ενός πλήρως παραλληλοποιημένου Radix-2 FFT, ο οποίος υλοποιείται στο επίπεδο υλικού και ακολουθεί την κατανομή από κάτω προς τα πάνω των δεδομένων και των πράξεων στα στάδια του.

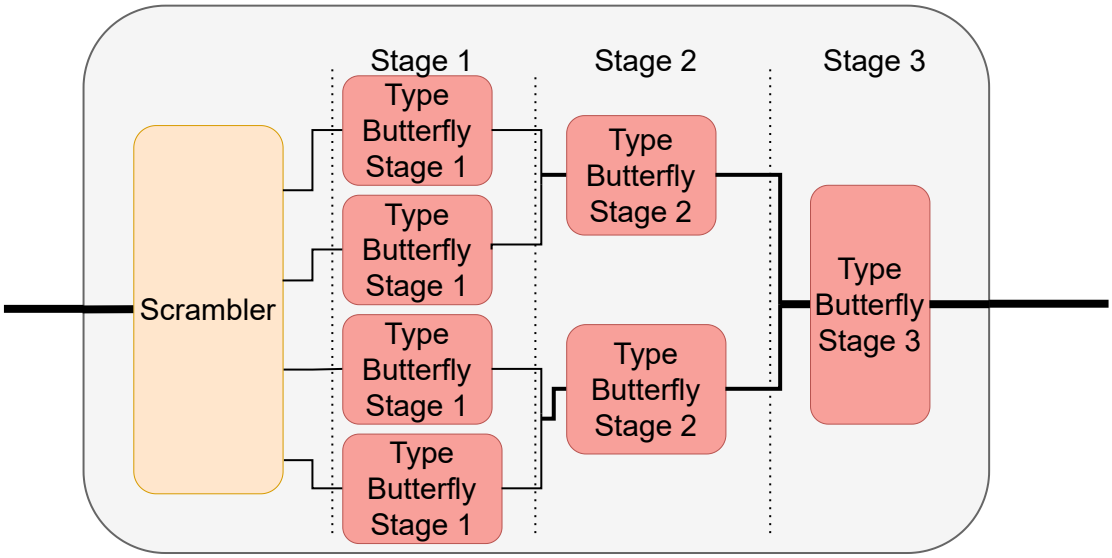


Figure 1: *FFT Kernel Architecture*

Ο πυρήνας αυτός αποτελείται από ξεχωριστά modules, όπως:

- Scrambler: χωρίζει και τοποθετεί σωστά τα δεδομένα εισόδου στον FFT, ώστε να εκτελεστούν οι πράξεις με τη σωστή σειρά.
- Butterfly: που εκτελεί τις πράξεις :

$$y0 = x0 + x1 * twiddle$$

$$y1 = x0 - x1 * twiddle$$

- Adder|Subber: εκτελούν complex πολλαπλασιασμούς, προσθέσεις και αφαιρέσεις που απαιτούνται από τις λειτουργίες των δομών ”butterfly”. Η αριθμητική αναπαράσταση που χρησιμοποιήθηκε είναι η fixed point Q1.15, η οποία περιορίζει τους αριθμούς στο εύρος [-1,1), χρησιμοποιώντας ένα signed bit και τα υπόλοιπα δεκαδικά. Για να διατηρηθούν οι περιορισμοί αυτοί, το αποτέλεσμα από κάθε πράξη περικόπτεται, καθιστώντας την πράξη στην πραγματικότητα μια διαίρεση δια 2.

Τεχνικές Αντιμετώπισης Σφαλμάτων

Για τις τεχνικές αντιμετώπισης σφαλμάτων, βασιστήκαμε σε :

Χωρικός Πλεονασμός (Spatial Redundancy)

Στα πλαίσια της έρευνάς μας, δοκιμάσαμε δύο διαφορετικές τεχνικές αντιμετώπισης των σφαλμάτων, συγκεκριμένα τον διπλασιασμό και τον τριπλασιασμό του πυρήνα FFT. Επιπλέον, χρησιμοποιήσαμε έναν voter, ο οποίος επιλέγει το σωστό αποτέλεσμα μεταξύ των αποτελεσμάτων που παράγονται από τους πολλαπλούς πυρήνες FFT, εξασφαλίζοντας την ακρίβεια των αποτελεσμάτων.

Διπλασιασμός Πυρήνα

Η μέθοδος του διπλασιασμού του κεντρικού module αποτελεί αποτελεσματικό τρόπο ανίχνευσης σφαλμάτων, αλλά δεν επαρκεί για τη διόρθωσή τους. Συνεπώς, σε περίπτωση σφάλματος, απαιτείται η επιδιόρθωση και των δύο module, με την προσθήκη χρόνου αδράνειας στη λειτουργία της εφαρμογής.

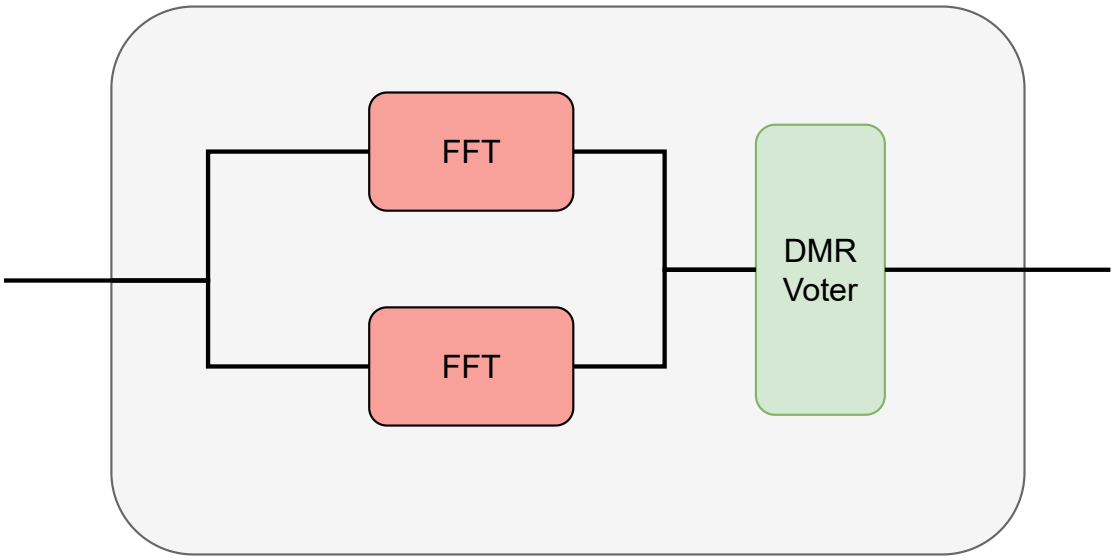


Figure 2: *DMR Architecture*

Τριπλασιασμός Πυρήνα

Με την εφαρμογή της τεχνικής του τριπλασιασμού, δίνεται η δυνατότητα να ανιχνεύσουμε τα σφάλματα που προκύπτουν στη λειτουργία του συστήματος και να τα διορθώσουμε ακόμη και σε περίπτωση που ένα από τα module δεχτεί αναστροφή bit (bit flip), αξιοποιώντας τις λειτουργικές δυνατότητες των δύο υπολοίπων. Πρόκειται για την πιο διαδεδομένη τεχνική αξιοποίησης στην αεροδιαστημική βιομηχανία, στην πυρηνική βιομηχανία και στον κλάδο της ιατρικής τεχνολογίας.

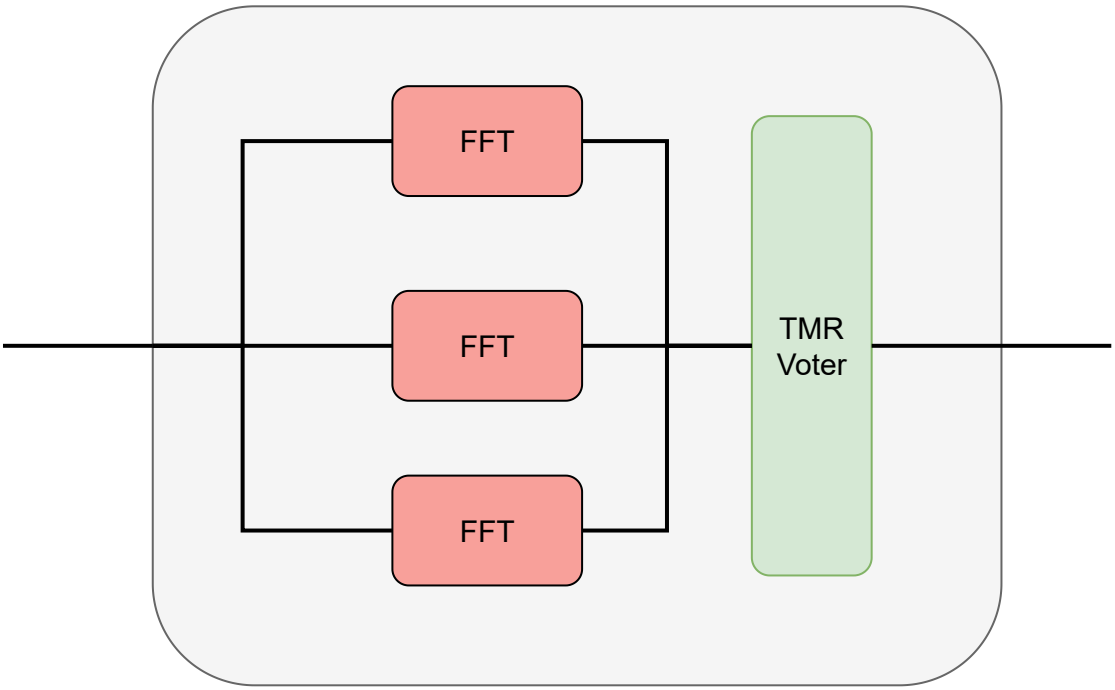


Figure 3: *TMR Architecture*

Χρονικός Πλεονασμός (Temporal Redundancy)

Χρησιμοποιώντας την τεχνική του χρονικού πλεονασμού, είναι δυνατόν να επαναλάβουμε τον υπολογισμό πάνω στα ίδια δεδομένα, προκειμένου να αντιληφθούμε τη χρονική στιγμή σε περίπτωση σφάλματος. Με αυτόν τον τρόπο, μπορούμε να αντιληφθούμε αν χρειάζεται να διορθώσουμε το σφάλμα στον FFT πυρήνα ή αν χρειάζεται να να διαμορφώσουμε όλη τη συσκευή. Είναι σημαντικό να σημειωθεί ότι αυτή η τεχνική μπορεί να αποδειχθεί πολύ χρήσιμη σε περιπτώσεις όπου το σφάλμα συμβαίνει κατά τη διάρκεια της επανάληψης των ίδιων δεδομένων. Ωστόσο, αν το σφάλμα συμβεί ανάμεσα στις επαναλήψεις, δεν θα είμαστε σε θέση να το αντιληφθούμε με αυτόν τον τρόπο.

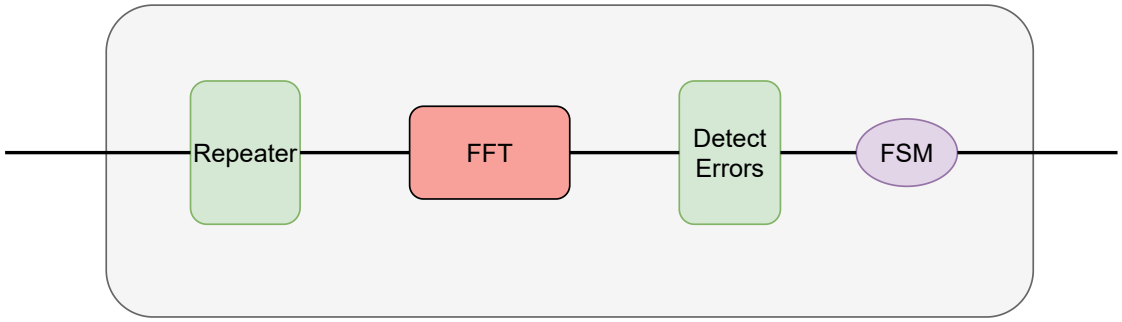


Figure 4: *Temporal Architecture*

Συνδιασμός Χρονικού και Χωρικού Πλεονασμού

Μια εναλλακτική προσέγγιση είναι να συνδυάσουμε τις δύο διαφορετικές τεχνικές, πολλαπλασιάζοντας τα δεδομένα εισόδου με τον αριθμό των πυρήνων και επαναλαμβάνοντας

τον υπολογισμό για κάθε επανάληψη. Ωστόσο, στην παρούσα εργασία, προτιμήσαμε τον διπλασιασμό και την επανάληψη των δεδομένων N (FFT kernel input length) φορές.

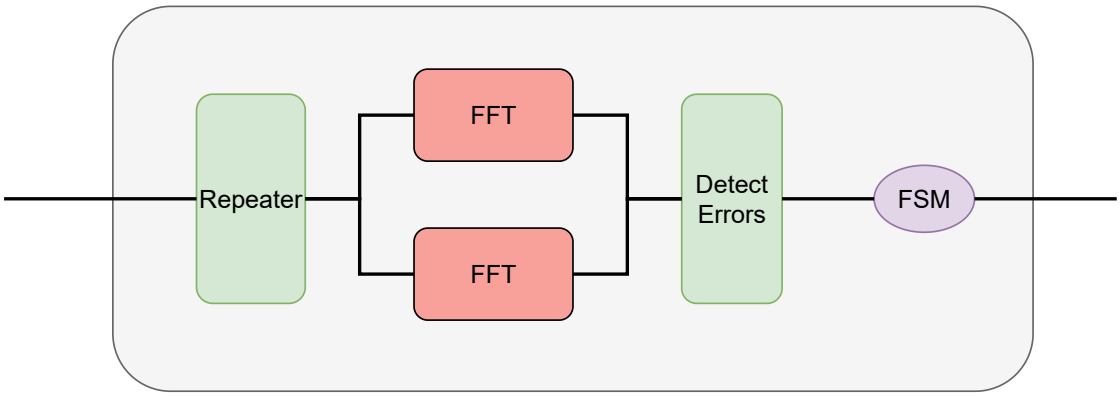


Figure 5: *DMR Temporal Architecture*

Χωρικός Πλεονασμός σε μικρότερα τμήματα (Fine-grain Spatial Redundancy) Μια εναλλακτική προσέγγιση θα ήταν η διαίρεση του πυρήνα σε μικρότερα τμήματα και η προστασία τους με την τεχνική του χωρικού περιορισμού, αντί να πραγματοποιήσουμε triplication σε ολόκληρη τη συσκευή. Μετά την υλοποίηση του πυρήνα FFT στο κάθε στάδιο, εφαρμόζεται τη μέθοδο στα μικρότερα τμήματα του πυρήνα καθώς και στα butterflies. Ωστόσο, για τη χρήση εσωτερικού χωρικού περιορισμού απαιτείται η χρήση επιπλέον voters.

FFT Stage TMR

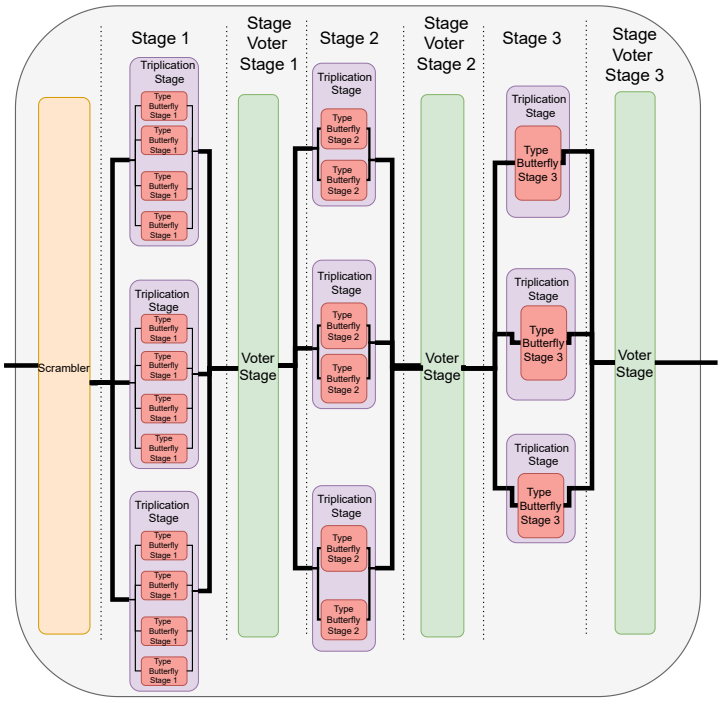


Figure 6: *TMR Stage Architecture*

FFT Butterfly TMR

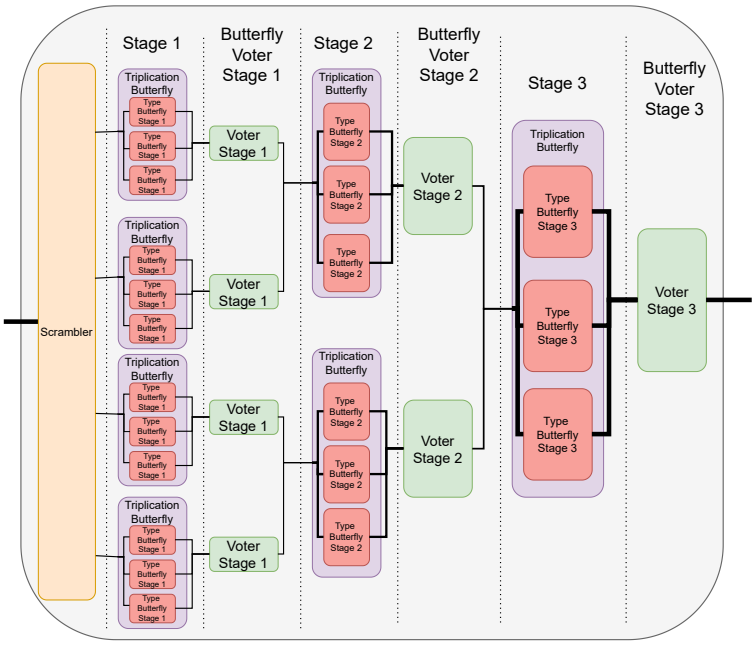


Figure 7: *TMR Butterfly Architecture*

Δυναμική Μερική Επαναδιαμόρφωση (Dynamic Partial Reconfiguration)
Στο πλαίσιο των FPGAs, είναι δυνατό να επαναδιαμορφώσουμε μόνο τμήματα της μνήμης ρύθμισης (configuration memory), αντί να επαναφορτώσουμε ολόκληρη τη μνήμη. Αυτό παρέχει τη δυνατότητα όχι μόνο να διορθώσουμε λάθη στο σχεδιασμό, αλλά και να διατηρήσουμε τη λειτουργία των ενοτήτων που δεν έχουν παρουσιάσει σφάλματα. Η εταιρεία Xilinx παρέχει ένα API με την ονομασία Xilfpga, το οποίο δίνει τη δυνατότητα εκτέλεσης είτε ενιαίας επαναδιαμόρφωσης (full reconfiguration) είτε μερικής επαναδιαμόρφωσης (partial reconfiguration) στη συσκευή. Τα bitstreams που απαιτούνται για την επαναδιαμόρφωση έχουν αποθηκευτεί σε μια κάρτα SD, από όπου μπορούμε να διαβάσουμε τα δεδομένα από το PL και να τα μεταφέρουμε στο FPGA fabric, προκειμένου να εκτελεστεί η επαναδιαμόρφωση.

Επικοινωνία PL-PS
Η επικοινωνία μεταξύ του επεξεργαστή (PS) και του PL στο σύστημά μας υλοποιήθηκε μέσω του πρωτοκόλλου AXI-Lite. Το πρωτόκολλο αυτό προσφέρει μια γρήγορη, memory mapped διεπαφή πάνω στην αρχιτεκτονική AMBA (Advanced Micorcontoler Bus Architecture). Η διεπαφή αυτή χρησιμοποιείται για τη μεταφορά δεδομένων και εντολών ανάμεσα στον επεξεργαστή και την λογική πύλη, με σκοπό τον έλεγχο και τη διαχείριση της λειτουργίας της PL.
Το πρωτόκολλο AXI-Lite χρησιμοποιεί μια απλή σειριακή δομή με τρεις κατηγορίες σημάτων: read, write και control. Τα σήματα read και write χρησιμοποιούνται για τη μεταφορά δεδομένων, ενώ το σήμα control χρησιμοποιείται για τον έλεγχο της λειτουργίας της PL. Η χρήση αυτής της διεπαφής προσφέρει ταχύτητα και ευελιξία στην επικοινωνία μεταξύ του επεξεργαστή και της λογικής πύλης, ενισχύοντας την απόδοση και την αξιοπιστία του συστήματος μας.

Υπολογισμός με την χρήση διαφορετικών FPGA Block e.g. DSP, LUT
Το FPGA προσφέρει δύο διαφορετικούς τρόπους για την εκτέλεση των υπολογισμών, τους οποίους μπορούμε να χρησιμοποιήσουμε ανάλογα με τις απαιτήσεις της εφαρμογής μας. Συγκεκριμένα, μπορούμε να χρησιμοποιήσουμε τους επεξεργαστές ψηφιακού σήματος (DSP) ή τους πίνακες αναζήτησης (LUT). Η βασική διαφορά ανάμεσά τους είναι το μέγεθος της επιφάνειας στο FPGA fabric που καταναλώνουν. Οι επεξεργαστές ψηφιακού σήματος καταναλώνουν λιγότερο χώρο σε σχέση με τους πίνακες αναζήτησης.

Διαδικασία Injection Σφαλμάτων
Προκειμένου να πραγματοποιήσουμε την αξιολόγηση των διαφορετικών τεχνικών που εφαρμόσαμε, χρειαζόμαστε έναν τρόπο να ενσωματώσουμε ένα σφάλμα εισαγωγής στη διαδικασία αξιολόγησης. Η SEM IP core από την Xilinx παρέχει αυτή τη δυνατότητα. Η διεπαφή που χρησιμοποιείται για να επικοινωνήσουμε με την SEM IP είναι ένας ελεγκτής uartlite που παρέχεται μαζί με την εφαρμογή. Για την πρόσβαση της SEM IP στη μνήμη διαμόρφωσης του FPGA χρησιμοποιείται η διεπαφή ICAP.

Whole Diagram

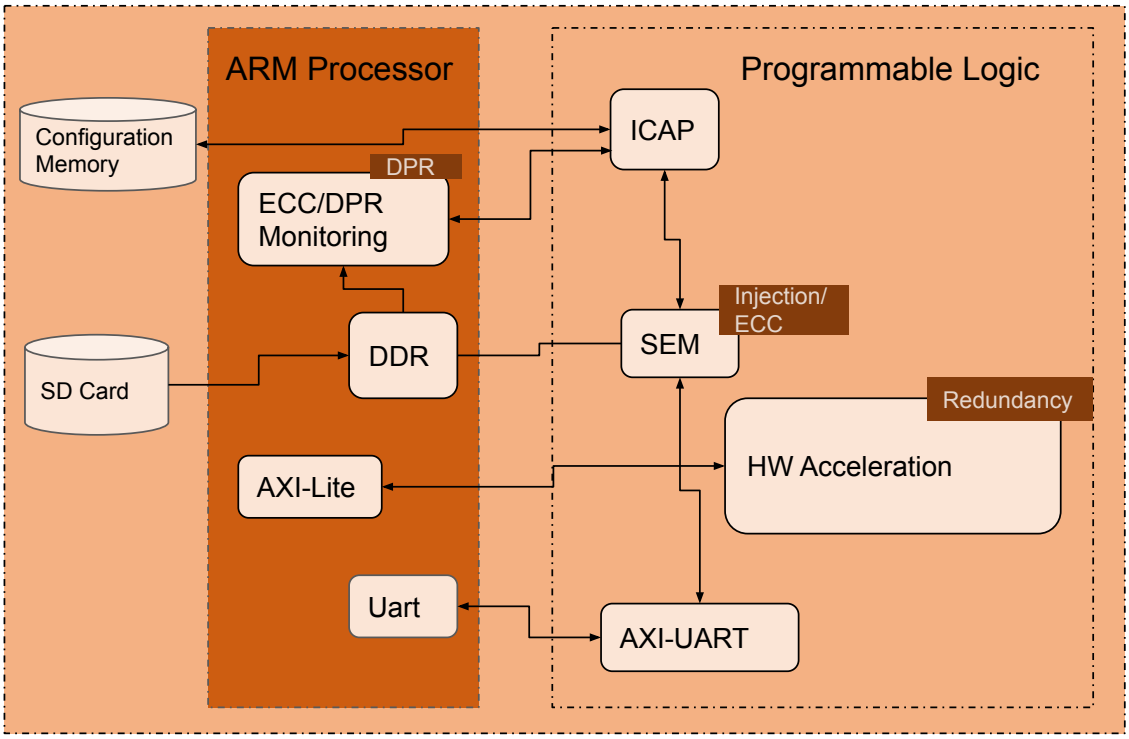


Figure 8: *Whole Diagram*

Evaluation

Στο πλαίσιο αυτής της διπλωματικής, αποφασίσαμε να χρησιμοποιήσουμε το MPSoC UltaScale+ ZCU106 board για την λήψη των μετρήσεων. Η επιλογή αυτή έγινε λόγω της αυξημένης παρουσίασης των Ultrascale+ based FPGAs σε εφαρμογές διαστήματος και λόγω του χαμηλού κόστους και της ευκολίας προγραμματισμού του.

Στο πλαίσιο αυτής της εφαρμογής, αποστέλλουμε 32-bit διανύσματα με σύνθετους αριθμούς στο FFT Kernel, χρησιμοποιώντας το πρωτόκολλο AXI-Lite. Για την αποστολή των δεδομένων, απαιτούνται οι διευθύνσεις των κρίσιμων bit από το στάδιο της υλοποίησης. Η λειτουργικότητα αυτή παρέχεται από το λογισμικό Vivado μέσω της εντολής:

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

Τα αρχεία που παράγονται είναι τα .edc και .ebd. Αυτά περιλαμβάνουν όλο το configuration memory content του FPGA με το .EBD, να είναι μια μάσκα του .EBC, όπου 1 δηλώνει ότι αυτό το bit είναι essential. Ωστόσο ήταν απαραίτητο να αντιληφθούμε τη συσχέτηση του αρχείου με τα clock regions του ZCU106 FPGA. Αυτό το καταφέραμε με το να τοποθετούμε ένα kernel σε κάθε σημείο και να αντιλαμβανόμαστε τα αποτελέσματα που έχει αυτό στην μορφολογία του .EBD. Η τοπολογία που καταλήξαμε φαίνεται στο κάτωθι σχήμα, και ακολουθεί μια zig-zag γραμμή στο FPGA fabric.

Χρησιμοποιώντας αυτή την λογική καταλήξαμε σε 200.000 διευθύνσεις στο FPGA στις οποίες κάναμε error injection. Τις κρατάμε σταθερές σε όλη την διάρκεια των μετρήσεων

μας, με σκοπό την αμεροληψία και αντικειμενικότητα των μετρήσεων.

Size

Οι κάτωθεν πίνανες εμφανίζουν τα resources που καταναλώνουν τόσο τα βασικά kernel, όσο και οι mitigation τεχνικές που εφαρμόσαμε.

Method	LUTs	DSPs	FFs	Es. Bits	LUTs	FFs	Es. Bits
FFT	2.77%	2.71%	1.24%	2.12%	2.04%	1.30%	2.74%
DMR	3.44%	5.38%	1.40%	2.57%	2.97%	1.52%	3.94%
TMR	4.26%	8.04%	1.51%	3.46%	3.96%	1.69%	5.19%
Temp	3.45%	2.71%	1.52%	2.39%	2.71%	1.58%	3.00%
DMR T	4.39%	5.38%	1.69%	3.12%	3.62%	1.80%	4.27%
TMR S	4.75%	9.08%	1.74%	4.39%	3.96%	1.84%	6.06%
TMR B	5.48%	9.08%	1.93%	4.34%	4.30%	2.02%	6.37%

(1) *Computation using DSP* **(2)** *Computation using LUT*

Table 1: *Resource utilization of FPGA with 8-input FFT*

Method	LUTs	DSPs	FFs	Es. Bits	LUTs	FFs	Es. Bits
FFT	3.82%	9.89%	1.28%	3.16%	3.65%	1.52%	4.75%
DMR	5.55%	19.73% %	1.61%	4.13%	4.21 %	2.08%	7.88%
TMR	7.58%	29.57%	1.82%	6.07%	6.07%	2.53%	11.36%
Temp	4.55%	9.89%	1.85%	**3.75%**	**3.75%**	2.08%	5.28%
DMR T	6.96%	19.73%	2.17%	5.36%	5.36%	2.64%	8.80%
TMR S	9.20%	20.89%	2.65%	**7.64%**	**7.64%**	2.91%	13.25%

(1) *Computation using DSP* **(2)** *Computation using LUT*

Table 2: *Resource utilization of FPGA with 16-input FFT*

Method	LUTs	DSPs	FFs	Es. Bits	LUTs	FFs	Es. Bits
FFT	6.09%	28.99%	1.44%	6.10%	6.37%	2.08%	11.05%
DMR	10.35%	57.92%	2.14%	10.65%	10.95%	3.42%	20.83%
TMR	15.27%	86.86%	2.63%	15.86%	15.44%	4.56%	30.35%
Temp	6.87%	28.99%	2.55%	7.14%	7.14%	3.19%	12.00%
DMR T	12.69%	57.92%	3.26%	12.21%	12.30%	4.54%	22.35%

(1) *Computation using DSP* **(2)** *Computation using LUT*

Table 3: *Resource utilization of FPGA with 32-input FFT*

Ενώ εδώ μπορούμε να παρατηρήσουμε την αλλαγή στα διαφορετικά resources με βάση την τεχνική και το μέγεθος του FFT Kernel.

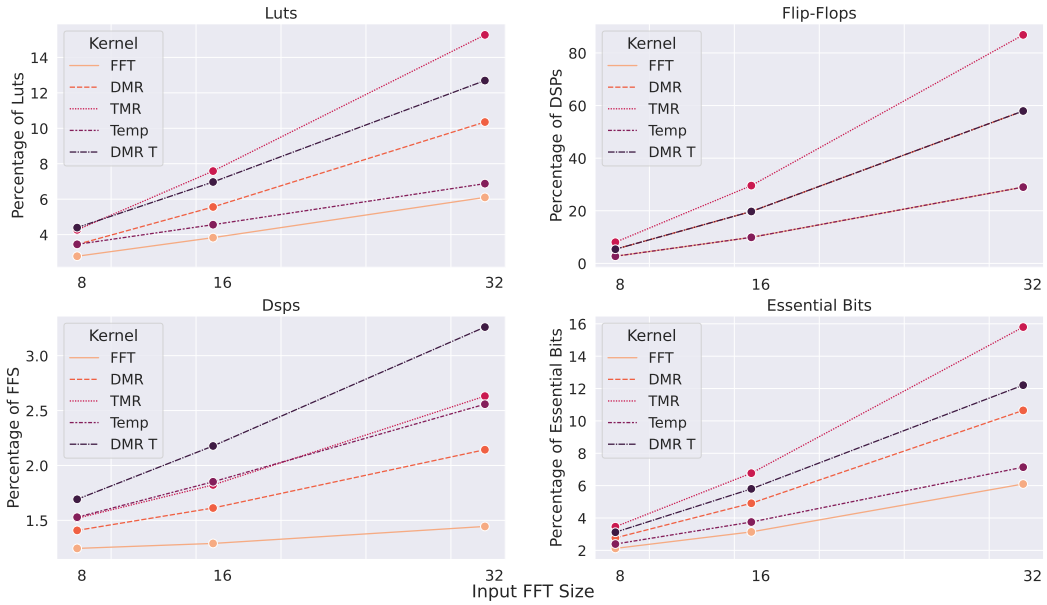


Figure 9: *Increase of usage based on input size using DSps*

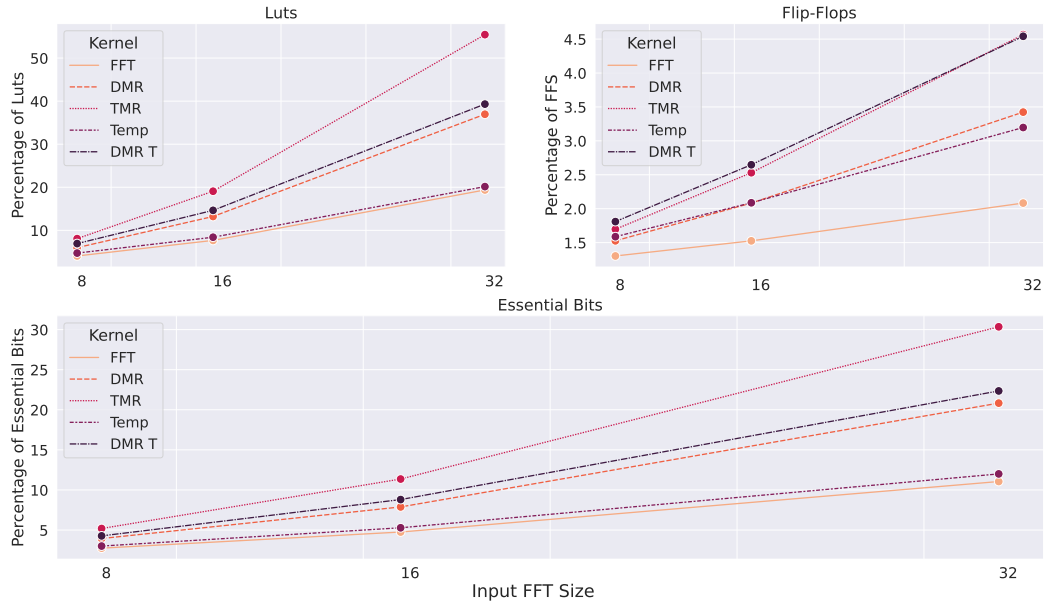


Figure 10: *Increase of usage based on input size using Luts*

Partial Reconfiguration

Για την μερική επαναδιαμόρφωση είναι απαραίτητη η παρατήρηση του χρόνου για κάθε ένα από τα reconfigurable modules που έχουμε εφαρμόσει αυτή την τεχνική. Είναι σημαντικό να σημειώσουμε ότι ενώ υπάρχει μια σύνδεση μεταξύ του μεγέθους του reconfigurable area, το startup time μπορεί να είναι σημαντικό στα modules με μικρότερα μεγέθη και δημιουργεί καθυστερήσεις. Επίσης, χωρίζοντας το design σε όλο και μικρότερα reconfigurable regions, στερούμε την δυνατότητα από το design να εφαρμόσει optimization,

μειώνοντας την χρήση των resources. Επομένως, ανεβαίνει η χρήση του area, που συνεπάγονται αύξηση του χρόνο reconfiguration.

-	Full Reconfiguration	19.3 MB	28.3
Kernel Size	Partial Module	Size of Module (KB)	Time (msec)
8	FFT Kernel	661	1
	Average Stage Module	330	0.6
	Average Butterfly Module	174	0.363
16	FFT Kernel	2354	3.59
	Average Stage Module	338	0.9
32	FFT Kernel	7492	11.46

Table 4: *Relation between the size and the time of Partial Reconfiguration*

Mitigation Techniques Evaluation

Τα παρακάτω διαγράμματα παρουσιάζουν την σύνδεση που υπάρχει μεταξύ του αριθμού λαθών που υπάρχει σε κάθε τεχνική και του downtime της εφαρμογής μας.

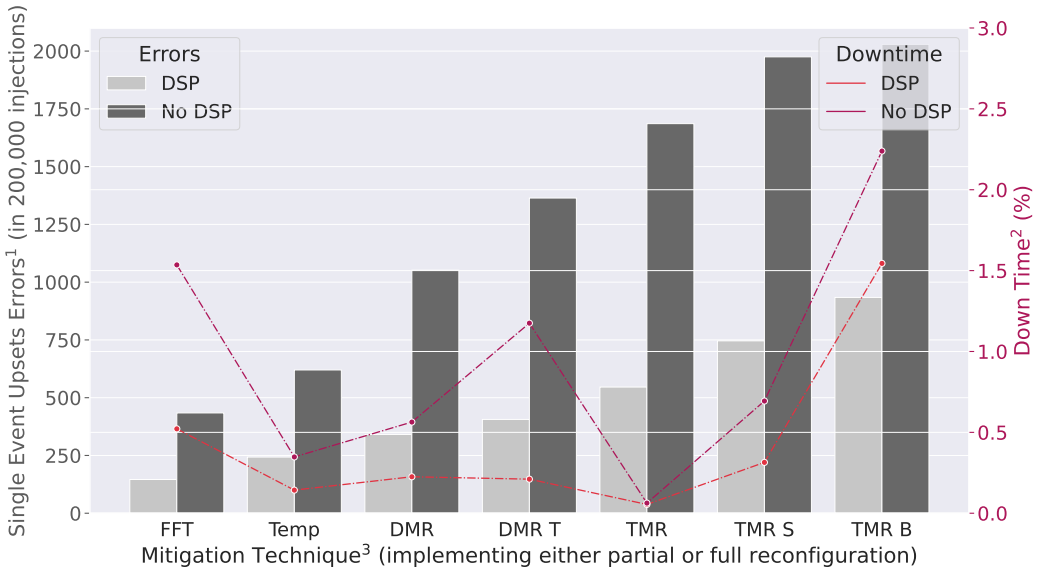


Figure 11: *Correlating SEU Errors and Downtime in a 8-Point FFT Kernel*

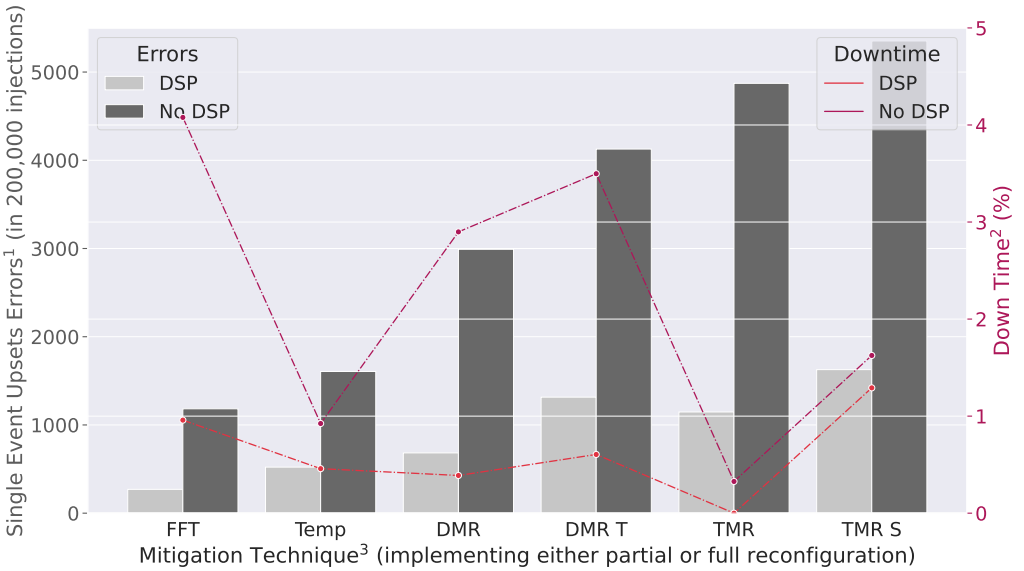


Figure 12: *Correlating SEU Errors and Downtime in a 16-Point FFT Kernel*

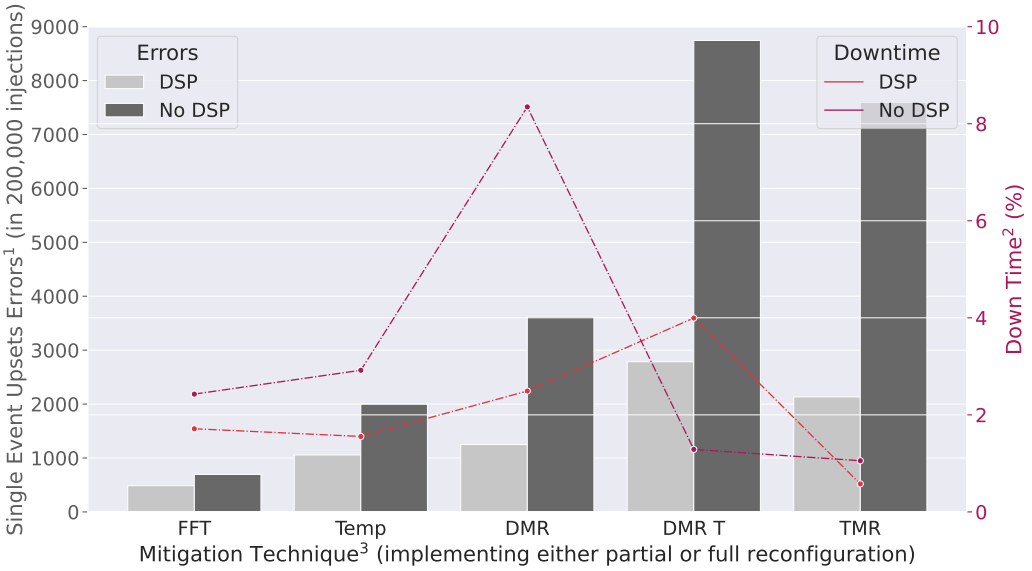


Figure 13: *Correlating SEU Errors and Downtime in a 32-Point FFT Kernel*

1. Single Event Upsets Errors: Η είσοδος λαθών στην εφαρμογή δεν οδηγεί πάντα στην παραγωγή SEU errors, και την αλλαγή στη λειτουργικότητα της εφαρμογής. Αυτά τα λάθη τα αναγνωρίζουμε είτε διαβάζοντας το λανθασμένο output είτε παρακολουθώντας τα flags, της εφαρμογής.

2. Downtime: Το downtime αναφέρεται σε μια περίοδο χρόνου, όπου η εφαρμογή δεν μπορεί να δεχθεί δεδομένα και είναι non-functional. Ωστόσο, σε περιπτώσεις που μεμονωμένα στοιχεία του design, βρίσκονται ανενεργά και εκτελείται σε αυτά μερική επαναδια-

μόρφωση, εφόσον τα άλλα components του design μπορούν να εκτελέσουν το compute, (TMR, DMR Temporal), τα input δεν χάνονται και ο χρόνος αυτός δεν προστίθεται στο downtime.
3. Mitigation Technique: Σε όλες τις τεχνικές, με το συνεχόμενο error injection, θα εμφανιστούν λάθη. Αυτά είτε τα αντιμετωπίζουμε με partial ή full reconfiguration. Αυτό επιτρέπει στην εφαρμογή μας να συνεχίσει και να ολοκληρώσει το σύνολο των error injection. Στις μετρήσεις μας, δεν συμπεριλάβαμε τεχνική που δεν περιλαμβάνει partial ή full reconfiguration.

Παρατηρούμε, αρχικά ότι χρησιμοποιώντας διαφορετικά FPGA blocks για την πραγματοποίηση των υπολογισμών, αυξάνουμε και το αριθμό των λαθών και το downtime, που είναι απόρροια του της μεγαλύτερης χρήσης των resources του FPGA. Οι καλύτερες τεχνικές που εφαρμόσαμε είναι η χρήση χρονικού πλεονασμού (temporal redundancy) στον FFT, που προφέρει την δυνατότητα να αντιληφθούμε τα λάθη στο design, και να οδηγηθούμε σε partial reconfiguration αντί για όλο το design, ενώ η δεύτερη είναι η χρήση του τριπλού χωρικού πλεονασμού. Στην τελευταία τεχνική υπάρχει η δυνατότητα για να ελαχιστοποιήσουμε το downtime, από την στιγμή που ενώ ένα από τα τρία FFT Kernels, έχει χτυπηθεί και υποστεί σφάλμα, τα άλλα δυο λειτουργούν, μη συνεισφέροντας έτσι στην μείωση της λειτουργικότητας του design μας. Παρατηρούμε, ωστόσο ότι το fine grain redundancy με τη χρήση σε κάθε πυρήνα δεν προφέρει καλά αποτελέσματα. Αυτό μπορεί να αποδοθεί πρώτον, στη αυξημένη χρήση των resources από την στιγμή που απαγορεύονται τα optimization στα resources μεταξύ των reconfigurable regions, και δεύτερον, η extra χρήση voters ανάμεσα στα στάδια αυξάνει την πιθανότητα ένας εκ των voter να αντιμετωπίσει σφάλμα, οδηγώντας την εφαρμογή σε full reconfiguration, που είναι $\times$28, πιο χρονοβόρο από το partial reconfiguration του FFT kernel 8 εισόδων.

Internal Scrubber Evaluation

Για τις καλύτερες μετρικές μας, αποφασίσαμε να προσθέσουμε και τη λειτουργικότητα του internal memory scrubbing που μας προσφέρει το SEM IP της Xilinx. Από όλα αυτά που δοκιμάσαμε επιλέξαμε το temporal και το tmr, με την χρήση των υπολογισμών με LUTS, για μέγεθος kernel 8 και 32. Η βελτίωση με την χρήση του interanl scrubber, ήταν της τάξης του 5% με 10% εξαίρεση του temporal για 32 input size, όπου η χρονική εξάρτηση των υπολογισμών οδήγησε σε χειρότερα αποτελέσματα.

Kernel Size	Mitigation Technique	DownTime (sec)	DownTime with CMS (sec)	Improvements of CMS
8	Simple FFT	12.47	11.781	5.5%
	Temporal	1.93	1.85	4.14%
	TMR	0.57	0.54	5.06%
32	Simple FFT	19.90	18.94	4.8%
	Temporal	23.53	23.34	-0.8%
	TMR	7.67	8.32	7%

Table 5: *Mitigation Techniques with and without Configuration Memory Scrubbing*

Comparisons

Σε αυτό το τελευταίο κεφάλαιο, παρουσιαζουμε μια συνολική εικόνα των καλύτερων απο-

τελεσμάτων με τις ανάλογες τεχνικές. Αυτό που μπορούμε να συνοψίσουμε είναι ότι οι καλύτερες τεχνικές είναι οι Temporal και TMR. Η πρώτη ότι λόγω του μικρού μεγέθους της κρατάει τον αριθμό των meaningful errors στο ελάχιστο, ενώ η χρήση του TMR προσφέρει την δυνατότητα να διατηρηθεί η λειτουργικότητα της εφαρμογής, ακόμα και αν κάποιο από το modules τεθεί εκτός λειτουργίας.
Ο τελικός πίνακας με τις καλύτερες μετρήσεις, μας παρουσιάζει τη βελτίωση στην μείωση του downtime σε σχέση με την baseline περίπτωση που δεν υπάρχει καμία από τις mitigation τεχνικές, όπως χωρικός ή χρονικός πλεονασμός, ή internal scrubbing.

Kernel Size	Comp FPGA Block	Mitigation Technique	Downtime Reduction
8	DSP	Temporal	72.7%
		TMR	89.5%
	LUT	Temporal	84.5%
		TMR	95.4%
		CMS	5.5%
		Temporal & CMS	85.09%
		TMR & CMS	95.64%
32	DSP	TMR	66.7%
	LUT	TMR	58.1%
		CMS	4.81%
		TMR & CMS	61.4%

Table 6: *The effectiveness of the proposed fault tolerant techniques in reducing downtime was compared against the baseline Fast Fourier Transform (FFT) Kernel. This was evaluated for each technique across various basecases, taking into consideration the number of points used for the FFT and the FPGA block utilized for computation.*

Chapter 1

Introduction

1.1 FPGAs

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that rely on a matrix of configurable logic blocks (CLBs) interconnected via programmable interconnects. A significant feature of FPGAs is their ability to be reprogrammed according to specific application or functionality requirements post-manufacturing. This ability distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are solely manufactured for dedicated design tasks. With the aid of FPGAs and associated design platforms, a higher level of flexibility, faster time-to-market, and a lower non-recurring engineering cost (NRE) are achievable for a broad spectrum of applications. Notably, FPGAs have been extensively applied in the fields of Aerospace & Defense, Automotive, Video & Image Processing, as well as Space missions and explorations.

Particularly in Space Applications, such as navigation, communication, the emerging era of commercialization of space, and observation of Earth and deep space, electronic systems require an extended lifespan due to the high costs and significant difficulties in maintenance, which may be impossible. Therefore, there is a pressing need to enhance the lifetime of electronics in space. Field Programmable Gate Array (FPGA) devices have been used in space for more than a decade with a mixed level of success. Until now, few reprogrammable devices have been used on European spacecraft due to their sensitivity to involuntary reconfiguration due to Single Event Upsets (SEU) induced by radiation. But with the advent of reprogrammable devices featuring a million system gates or more, it is no longer feasible to disregard these technologies. The FPGA vendors have already begun to develop SEU mitigation techniques to make their devices usable in space applications

The capacity and performance of FPGAs suitable for space flight have been increasing steadily for more than a decade. For reprogrammable devices, the increase has been from tens of thousands to millions of system gates. The application of FPGAs has moved from simple glue logic to complete subsystem platforms that combine several real-time systems functions on a single chip, even including microprocessors and memories [1] and [2]. The potential for FPGA use in space is steadily increasing, continuously opening up new application areas. FPGAs are more commonly used in critical applications and are replacing ASICs regularly.

The potential of reprogrammable FPGAs has been presented in [3] and [4] and is repeated hereafter. Many experiments [5] have been conducted to test the feasibility of reconfigurable devices in space applications. Reconfigurable computing technology is still a relatively new field of study for space applications. The space environment is different from terrestrial systems in that incident radiation can cause bit flips in memory elements and ionization failure in semiconductors. This kind of hardware fault cannot be debugged and repaired, requiring high-reliability manufacture, assembly, and operating techniques.

1.2 Landscape of COTS FPGAs Devices in Space

In the realm of space applications, meeting challenging requirements often requires the use of commercial off-the-shelf (COTS) devices [6]. While traditional military/space electrical, electronic, and electromechanical (EEE) parts have proven to be suitable for use in these applications, budget constraints and declining availability have made it necessary to find alternative solutions. As a result, more and more COTS devices are finding their way into space missions due to their higher performance and lower costs. Although radiation-hardened devices are often used in space, they are prohibitively expensive for smaller missions involving CubeSat solutions. Furthermore, extensive testing durations can delay deployment and overlook the advancements in computing and implementation methodologies that are available today.

The strict constraints for real-time processing and low power have forced the space community to examine alternative solutions for onboard processing. In more detail, general-purpose processors, such as the conventional Central Processors Units (CPUs) and microprocessors, cannot provide sufficient acceleration in compute-intensive tasks of space applications, e.g., for Vision-Based Navigation (VBN) and Earth Observation (EO). As a result, FPGAs [7, 8, 9, 10, 11, 12] and other novel accelerators (e.g., GPUs [13, 14], TPUs [15, 16], and VPUs [17, 18]) are employed to deliver enhanced performance and meet the demands of modern space applications. When even more improvement is targeted in performance and Space, Weight & Power (SWaP), there is a tendency to use mixed-criticality processing architectures [19, 20], which consist of both space-grade and COTS components. To put things into perspective, COTS processors are faster, less expensive, and more flexible in general than their radiation-hardened counterparts, and thus, they can facilitate on-board computing in several aspects.

Regarding the FPGAs, which are the focus of our work, they are typically used as main processors for demanding tasks (i.e., for DSP/AI acceleration) or as framing processors (i.e., for sensor data handling and data transcoding). Many space agencies and industries worldwide employing it in both consumer and research satellites and spacecraft. Companies like AMD Xilinx have responded to this demand by developing space-grade FPGAs such as the Virtex-5QV [21], the industry's first high-performance, rad-hard reconfigurable FPGA designed for processing-intensive space systems. Other notable FPGA platforms include the Q8S [22] developed by Xiphos Systems Corporation and the Leopard DPU [23] developed by KP Labs, which is compliant with CubeSat standards and enables the

application of Artificial Intelligence solutions in space. These platforms offer high flexibility and significant performance gains and are equipped with Zynq UltraScale+ MPSoC processing cores.

Considering the significant costs associated with space electronics, this thesis contribution focuses on the development and validation of FPGA-based mitigation solutions on the Xilinx MPSoC UltraScale+ platform, specifically, the ZCU106 [24]. By adopting this approach, the development process can be readily adapted to space-grade platforms with minimal effort and changes, while still taking advantage of the benefits offered by COTS devices.

1.3 Problem Statement

The space environment poses a significant threat to electronic devices due to the presence of ionizing radiation, which can increase the conductivity of materials, leading to damaging current levels and potentially causing permanent damage or operation errors in electronic equipment. Semiconductor microelectronics are particularly vulnerable to these phenomena, which can cause single-event upsets (SEUs) in digital circuits, leading to fatal errors in aerospace applications [25]. As a result, radiation hardening and tolerance techniques are typically employed to protect circuits in these applications. However, these techniques are expensive and may not be feasible for use in smaller satellites, such as CubeSats.

Commercial-of-the-shelf (COTS) Field-programmable gate arrays (FPGAs), with their reconfigurable nature, are uniquely positioned to overcome these difficulties. Their ability to be reconfigured on-the-fly enables them to adapt to the harsh space environment and continue operating in the face of radiation-induced faults. As a result, companies have developed radiation-hardened and radiation-tolerant FPGAs that provide designers with robust and qualified devices that meet the demanding performance, reliability, and lifecycle requirements of space applications.

Despite the availability of these specialized devices, there remains a lack of exploration in the area of different design technologies. This is partly due to the time-consuming nature of evaluating the radiation hardness of different design approaches, as well as the kernel-specific nature of the problem. However, it is crucial to develop a better understanding of the effects of ionizing radiation on different FPGA designs in order to improve their resilience and ensure their continued operation in space environments.

The contribution of this thesis is summarized as follows:

- We apply fault tolerance mitigation techniques in the MPSoC UltraScale+ FPGA, which has attracted significant interest from the space industry (e.g., it is integrated in Q8S [22] and Leopard [23]), while the majority of the previous works focused on the Zynq FPGAs [26].
- We propose various mitigation techniques that can be applied along with every accelerator (application), as well as application-specific mitigation techniques (i.e., for the FFT kernel).

- We develop our own fault injection campaign, which can be used to evaluate fault tolerant systems in the MPSoC UltraScale+ FPGA.
- We examine the error resilience and the reliability of the FFT algorithm for various input lengths (8/16/32).
- We explore the impact of utilizing different FPGA blocks (e.g., DSPs, LUTs) on the reliability of the device.

1.4 Thesis Structure

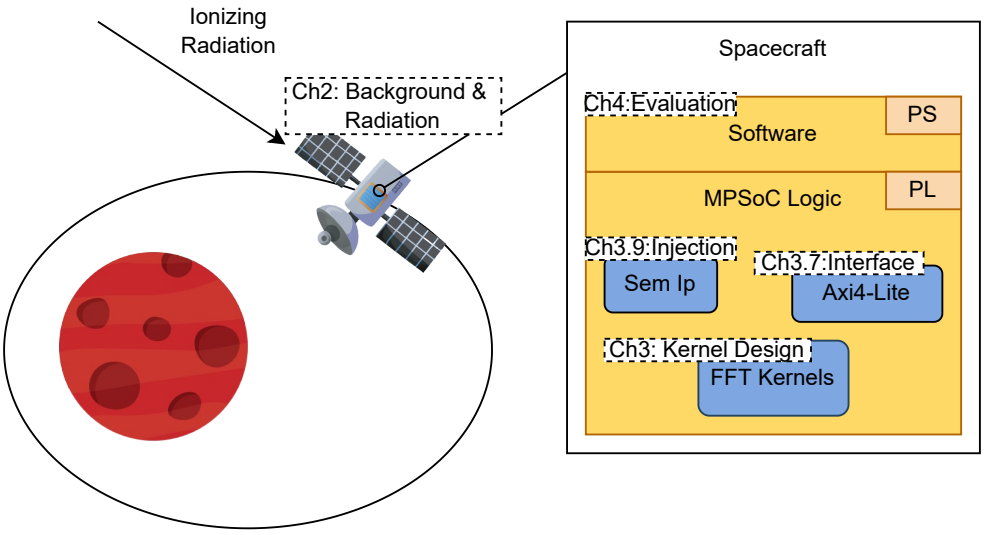


Figure 1.1: *Visual Chapter Guide*

A brief outline of the subsequent chapters follows, with a visual chapter guide depicted in Figure 1.1.

Chapter 2: A general background on Space Environment and Fault-Tolerance techinques

In Chapter 2, a general background is presented on the basic layout of Field-Programmable Gate Arrays (FPGAs) and the tools required for their programming. The effects of ionizing radiation on FPGAs are then discussed, along with an analysis of the harmful behavior that they may cause in satellites operating in a space environment. This chapter also introduces several common mitigation mechanisms that have been previously implemented in a variety of fields, encompassing both electronic and mechanical components. Many of these mechanisms serve as building blocks for the architecture presented in this thesis.

Chapter 3: Proposed Techniques and Methodologies for Error Mitigation

In Chapter 3, the implementation of a fast Fourier algorithm as a hardware module is outlined, serving as the fundamental building block for all the kernels designed and evaluated in this study. A detailed architecture and approach for designing these kernels are presented, along with thorough explanations of the implementation of the architectures. The communication protocol AXI4-Lite, used for receiving and communicating with the design, is also described. In addition, dynamic partial reconfiguration is introduced as a technique that enables recovery from the faulty configuration of the Programmable Logic (PL). The computational operation is transitioned from Digital Signal Processing components (DSPs) to Look Up Tables (LUTs), and the injection method utilizing Xilinx Core SEM IP is discussed in detail.

Chapter 4: Validation and Evaluation of Fault-Tolerance Mitigation Techniques

In Chapter 4, the results of injection tests performed on Xilinx's MPSoC UltraScale+ ZCU106 board are presented, which are covering all of the architectures proposed in Chapter 3. The chapter explains how errors were introduced during the tests, and it includes a comparison of the performance metrics and resource utilization of each kernel across input sizes of 8, 16, and 32. In addition, the injection campaign is described in detail, including the methodology that was followed during the execution of the tests.

Chapter 5: Conclusion and Future work

In Chapter 5, this thesis concludes with a comprehensive evaluation of the proposed mitigation techniques. Furthermore, a brief discourse on potential future avenues for research is provided, building on the techniques presented in this thesis.

Chapter 2

A general background on Space Environment and Fault-Tolerance techniques

In Chapter 2, a general background is presented on the basic layout of Field-Programmable Gate Arrays (FPGAs) and the tools required for their programming. The effects of ionizing radiation on FPGAs are then discussed, along with an analysis of the harmful behavior that they may cause in satellites operating in a space environment. This chapter also introduces several common mitigation mechanisms that have been previously implemented in a variety of fields, encompassing both electronic and mechanical components. Many of these mechanisms serve as building blocks for the architecture presented in this thesis.

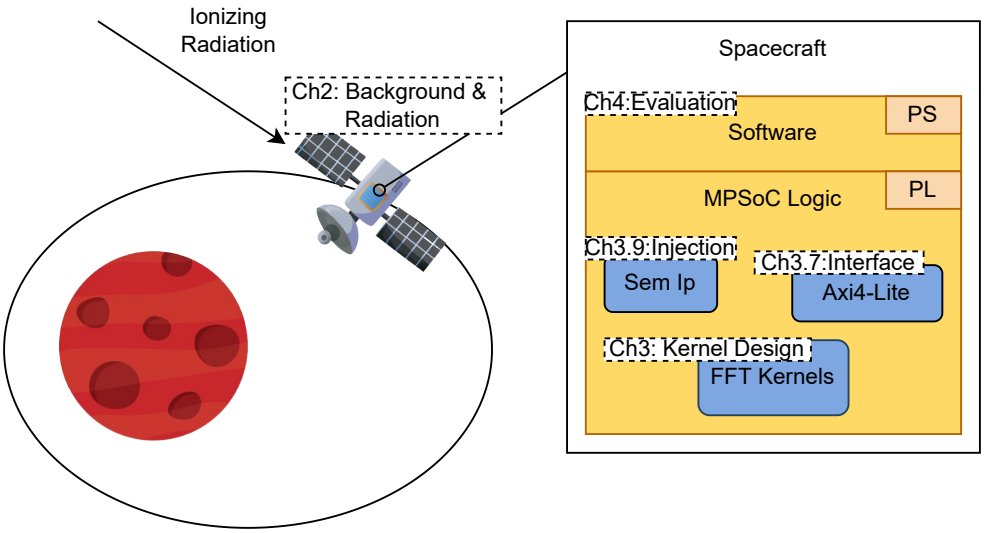


Figure 2.1: *Visual Chapter Guide*

2.1 MPSoC Tools and Architecture

The development board used as part of our thesis is the [24] Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit developed by AMD Xilinx. The included ZU7EV device is equipped with a quad-core Arm Cortex-A53 applications processor, dual-core Cortex-R5 real-time processor, and 16nm FinFET+ programmable logic. The device includes :

System Logic Cells (K)	504
Memory	38Mb
DSP Slices	1,728
Video Codec Unit	1
Maximum I/O Pins	464
LUT	230400
FF	460800
BUFG	544

Table 2.1: *Specification of Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC*

In general, FPGAs are integrated circuits that consist of smaller elements. Some of these elements are the CLBs (configurable logic blocks), memory elements, DSPs (digital signal processing blocks), IO blocks, etc. The functionality of the kernels is implemented in the FPGA fabric using both the functionality of the routing matrix and the CLB's internal routing and structures. The routing matrix consists of switch boxes and multiplexers, responsible for routing and transferring the information. The CLBs are tiled across the fabric in rows and columns with channels of routing wires lying in between them. The CLBs contain smaller logic elements and their configuration is determined by the manufacturer. In the MPSoC UltraScale+ architecture, Xilinx has created the following [27] CLB architecture.

Every CLB contains one slice with eight 6-input LUTs and sixteen storage elements. The LUTs are organized as a column with an 8-bit carry chain per CLB, called CARRY8. Wide-function multiplexers combine LUTs to create any function of 7, 8, or 9 inputs, or some functions of up to 55 inputs. SLICEL is the name used to describe CLB slices that support these functions, where the L is for logic. The LUT in a SLICEM, where the M is for memory, can be configured as a look-up table, 64-bit distributed RAM or a 32-bit shift register. The CLB for a SLICEL is referred to as a CLEL tile, and the CLB for the SLICEM is referred to as a CLE_M tile. The table summarizes the resources in one CLB.

CLB Slice	LUTs	Flip-Flops	Arithmetic and Carry Chains	Wide Multiplexers	Distributed RAM	Shift Registers
SLICEL	8	16	1	F7, F8, F9	N/A	N/A
SLICEM	8	16	1	F7, F8, F9	512 bits	256 bits

Table 2.2: *Logic Resources in One CLB Slice*

The function generators are implemented as six-input look-up tables (LUTs). There are six independent inputs (1 to 6) and two independent outputs (O5 and O6) for each of the eight function generators in a CLB slice. The function generators can implement:

- Any arbitrarily defined six-input Boolean function.
- Two arbitrarily defined five-input Boolean functions, as long as these two functions share common inputs.
- Two arbitrarily defined Boolean functions of three and two inputs or less.

Storage Elements There are 16 storage elements per CLB slice (two per LUT), which can all be configured as either edge-triggered D-type flip-flops or level-sensitive latches. Control Signals There are two clock inputs (CLK) and two set/reset inputs (SR) to every CLB for the storage elements.

Distributed RAM (SLICEM Only) The function generators (LUTs) in SLICEM can be implemented as a synchronous RAM resource also described as distributed RAM. Multiple LUTs in a SLICEM can be combined in various ways to store larger amounts of data up to 512 bits per SLICEM. Multiple SLICEMs can be combined to create larger memories.

Shift Registers (SLICEM Only) A SLICEM function generator can also be configured as a 32-bit shift register without using the flip-flops. When used in this manner, each LUT can delay serial data from one to 32 clock cycles. The shifting D (DI1 LUT pin) and shift out Q31 (MC31 LUT pin) lines cascade LUTs to form larger shift registers. The eight LUTs in a SLICEM are cascaded to produce delays of up to 256 clock cycles. It is also possible to combine shift registers across more than one SLICEM.

The contents of the configuration memory are generated by the design tool software suite in the form of a configuration bitstream. In the case of Xilinx, the Vivado-Vitis design suite is developed and enables a wide range of functionality in both the hardware and software design of Xilinx's FPGAs. 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. The Vivado tools allow the user to extract besides the bitstream file used for the programming of the FPGA, two more files, the [28] .ebc and .ebd labeled files. The EBC file is a reference file and contains the memory cell content, while the .EBD file is used to mask the EBC file, 1 in the EBD file corresponds to an essential bit in the EBC file.

If a soft error occurs, one or more memory bits are corrupted. The memory bits affected can be in the device configuration memory (which determines the behavior of the design), or might be in design memory elements (which determines the state of the design). The following four memory categories represent a majority of the memory in a device:

- Configuration Memory – Storage elements used to configure the function of the design loaded into the device. This includes function block behavior and function block connectivity. This memory is physically distributed across the entire device and represents the largest number of bits. Only a fraction of the bits is essential to the proper operation of any specific design loaded into the device.

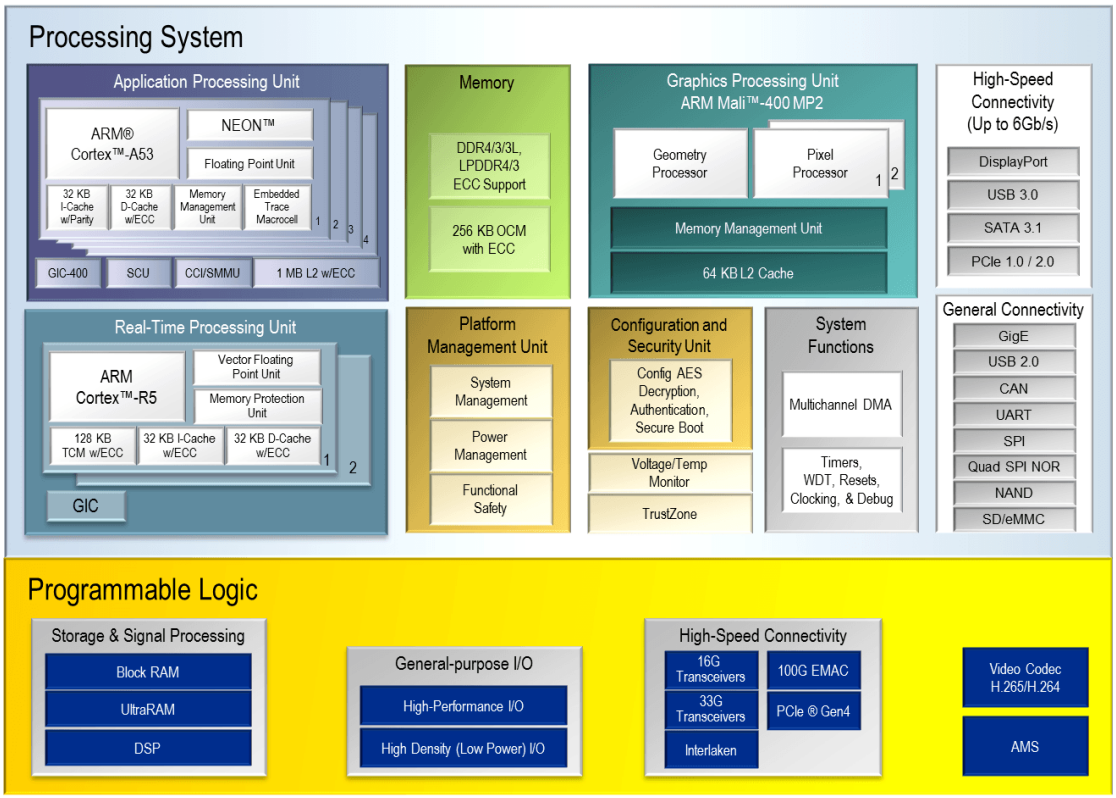


Figure 2.2: *MPSoC Architecture*

- Block Memory – High capacity storage elements used to store design state. As the name implies, the bits are clustered into a physical block, with several blocks distributed across the entire device. Block Memory represents the second largest number of bits.

- Distributed Memory – Medium capacity storage elements used to store design state. This type of memory is present in certain configurable logic blocks (CLBs) and is distributed across the entire device. Distributed Memory represents the third largest number of bits.

- Flip-Flops – Low capacity storage elements used to store design state. This type of memory is present in all configurable logic blocks (CLBs) and is distributed across the entire device. Flip-Flops represent the fourth largest number of bits. An extremely small number of additional memory bits exist as internal device control registers and state elements. Soft errors occurring in these areas can result in regional or device-wide interference that is referred to as a single-event functional interrupt (SEFI). Due to the small number of these memory bits, the frequency of SEFI events is considered negligible in this discussion, and these infrequent events are not addressed by the SEM controller.

An EBD file [29] is an ASCII text file that has an informational header, followed by a number of lines, where each line has 32 characters that are either zero (0) or one (1). Each line represents the classification of 32-bits or one word of CRAM. Zero means non-

essential and one means essential. The Least Significant Bit (LSB) of a word, which is sequentially considered it's the first bit, is on the far right. In the 16nm UltraScale+ family, each CRAM frame is 93[30] words. As each line in the EBD represents a word, You can visualize it by counting off groups of 93 lines in the EBD file. Each group of 93 lines of data in the EBD file holds the essential bits of data for a configuration frame. It is ordered by incrementing Linear Frame Address, or LA. The initial group of 118 lines consists of 25 dummy words and one dummy frame. The next group of 93 lines is for LA = 0, followed by the next group of 93 lines for LA = 1, and so on.

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.

2.2 Ionising radiation

Ionizing radiation is any type of particle or electromagnetic wave that carries enough energy to ionize or remove electrons from an atom. For transistors that are made up of 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.

Radiation effects in micro-electronics [31] are a serious concern for the performance and the survival of devices operating in radiation environments, from outer space to aircraft avionics to accelerators and nuclear power plants and even safety-critical equipment operation at ground level. Many experiments have been conducted to measure the significance of the f

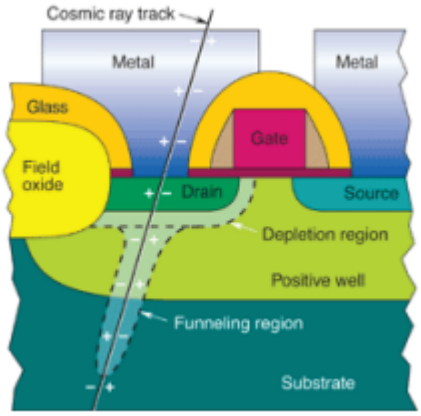


Figure 2.3: *Ionizing Injection*

Radiation effects in microelectronics are broadly grouped into several categories. The first distinction is whether the effect is the result of cumulative damage from the passage of many energetic particles, or whether it results from the passage of a single particle. Examples of cumulative damage include the total ionizing dose (TID) and displacement

damage dose (DDD). Failures resulting from a single particle can be destructive or non-destructive to the device and may include memory upsets, latch, gate rupture, burnout, and other phenomena, broadly described as single event effects (SEE).

Single Event Effects can be further divided into destructive and non-destructive. The destructive as the name suggests can result in catastrophic failure of the device. The most common types of catastrophic single-event effects include single-event latch-up (SEL), single-event burnout(SEB), and single-event gate rupture or dielectric rapture (SEGR/ SEDR). In FPGA the are rare, while non-destructive single-event effects on the programmable logic are more common, which effects do not result in device failure but still affect normal operation. The most commonly known of these effects is the single event upset (SEU) or ”bit-flip” in a memory cell. The effects of single event upsets have been thoroughly evaluated, especially in their impact on the configuration memory [32] of the FPGA.

2.3 Classic FT Techniques theoretical

Classical fault-tolerant techniques are a set of techniques used to ensure that a system or device continues to operate correctly in the presence of one or more faults or failures. These techniques are commonly used in safety-critical systems, such as aerospace, automotive, and medical devices, as well as in high-performance computing systems and other applications where system reliability is critical. Some of the classical fault-tolerant techniques are:

1. Redundancy: This technique involves duplicating the critical components or subsystems of a system so that if one component or subsystem fails, the redundant component or subsystem can take over. There are different types of redundancy techniques, such as hardware redundancy, software redundancy, and information redundancy.

2. Error detection and correction: This technique involves detecting and correcting errors that occur in a system or device. This can be done through techniques such as checksums, parity bits, and cyclic redundancy checks.

3. Fail-safe design: This technique involves designing a system or device in such a way that if a failure or fault occurs, the system or device will fail safely and predictably. This technique is commonly used in safety-critical systems, such as aircraft systems and medical devices.

4. Fault avoidance: This technique involves designing a system or device in such a way that faults or failures are less likely to occur. This can be done through techniques such as robust design, testing, and quality control.

5. Fault isolation: This technique involves designing a system or device in such a way that if a fault or failure occurs, it can be isolated to a specific component or subsystem, allowing the rest of the system or device to continue to operate correctly.

6. Recovery: This technique involves designing a system or device in such a way that if a fault or failure occurs, the system or device can be restored to its normal operation as quickly as possible. This can be done through techniques such as checkpointing, replication, and restart.

Indeed, the continuous development of fault-tolerant techniques and systems is crucial for improving the reliability and safety of a wide range of electronic and mechanical systems. As technology evolves and becomes more complex, the need for robust and fault-tolerant designs becomes increasingly important. This is particularly true in safety-critical systems such as aircraft, medical devices, and nuclear power plants, where a failure can have catastrophic consequences.

Advancements in fault-tolerant techniques and systems have resulted in the development of more capable and reliable platforms, which can withstand a wide range of failure modes and continue to operate safely and reliably. For example, modern aircraft use sophisticated fault-tolerant avionics systems that are designed to continue functioning even in the event of multiple component failures. Similarly, medical devices such as pacemakers and implantable defibrillators use advanced fault-tolerant designs to ensure that they continue to function correctly even in the presence of faults or failures. Such technics are starting to be deployed more and more in space applications, such as CubeSat or deep space missions.

In addition to improving the safety and reliability of electronic and mechanical systems, fault-tolerant techniques can also help to reduce the cost of development, deployment, and operation. By designing systems that can continue functioning in the presence of faults or failures, it may be possible to avoid costly shutdowns, repairs, replacements, or cancelations of the whole operation.

Overall, the development of fault-tolerant techniques and systems is an important area of research and development, which has the potential to improve the reliability and safety of a wide range of electronic and mechanical systems, while also reducing the cost of development and operation.

Chapter 3

Proposed Techniques and Methodologies for Error Mitigation

In Chapter 3, the implementation of a fast Fourier algorithm as a hardware module is outlined, serving as the fundamental building block for all the kernels designed and evaluated in this study. A detailed architecture and approach for designing these kernels are presented, along with thorough explanations of the implementation of the architectures. The communication protocol AXI4-Lite, used for receiving and communicating with the design, is also described. In addition, dynamic partial reconfiguration is introduced as a technique that enables recovery from the faulty configuration of the Programmable Logic (PL). The computational operation is transitioned from Digital Signal Processing components (DSPs) to Look Up Tables (LUTs), and the injection method utilizing Xilinx Core SEM IP is discussed in detail.

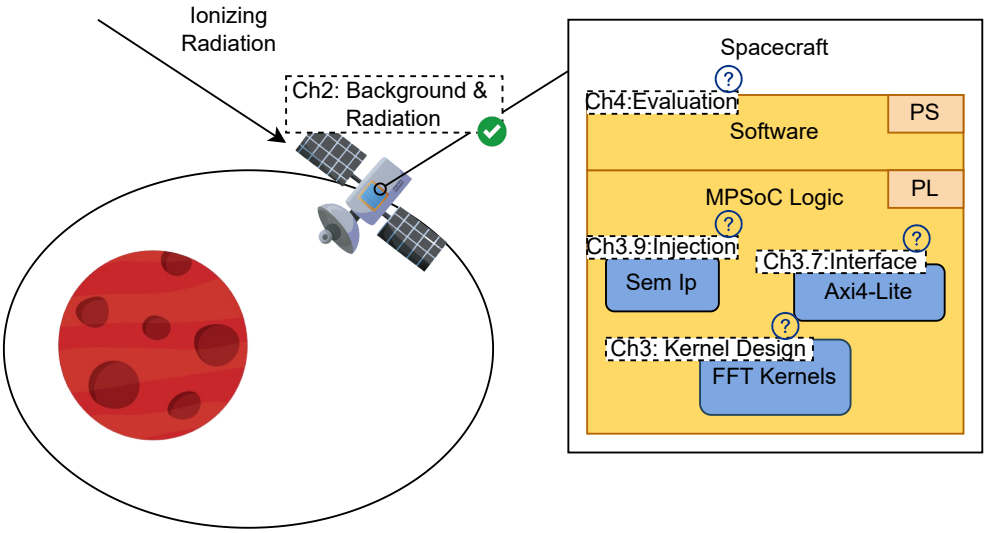


Figure 3.1: *Visual Chapter Guide*

3.1 Design of FFT Hardware Kernel

The first step towards implementing some fault mitigation techniques is to develop a custom design that allows us to understand the design deeply, in every part of it. The algorithm that, we decided to implement is the fast Fourier transform.

Fast Fourier transform is a crucial algorithm and a fundamental part of most of our modern digital signal processing [33]. A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors.[2] As a result, it manages to reduce the complexity of computing the DFT from $O(N^2)$ to $O(N \log N)$. The importance is enormous, especially considering the increasingly input size necessary, and the amount of data the transrfation needed to compute.

Fast Fourier transforms are widely used for applications in engineering, music, science, and mathematics. The basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805.[1] In 1994, Gilbert Strang described the FFT as ”the most important numerical algorithm of our lifetime”,[3][4] and it was included in the Top 10 Algorithms of the 20th Century by the IEEE magazine Computing in Science & Engineering.[5]

Both the discrete Fourier transform and the FFT transforms a sequence of N complex numbers.

$$\xi \sum_{i=1}^{10} t_i\{c\} a_{1,1}$$

$$X_k = \sum_{i=0}^{N-1} x_n e^{-i2\pi kn/N}$$

where $\{x_n\} = x_0, x_1, \ldots, x_{N-1}$ and $\{X_k\} = X_0, X_1, \ldots, X_{N-1}$.

The FFT is used in digital recording, sampling, additive synthesis, and pitch correction software.

The FFT's importance derives from the fact that it has made working in the frequency domain equally computationally feasible as working in the temporal or spatial domain. Some of the important applications of the FFT include:

Communications

The most important equation for communication and signal processing is the convolution between the input x(t) and the impulse of that system h(t) which results in the output Y(t). However, the convolution algorithm is time, thus transforming the domain from time to frequency, allowing us to use multiplication $Y(f) = x(f) * h(f)$, which is mathematically significantly easier than convolutions.

Optics, diffraction, and tomography

The discrete Fourier transform is widely used with spatial frequencies in modeling the way that light, electrons, and other probes travel through optical systems and scatter from objects in two and three dimensions. The dual (direct/reciprocal) vector space of three-dimensional objects further makes available a three-dimensional reciprocal lattice, whose construction from translucent object shadows (via the Fourier slice theorem) allows tomographic reconstruction of three-dimensional objects with a wide range of applications e.g. in modern medicine.

Data compression

The field of digital signal processing relies heavily on operations in the frequency domain (i.e. on the Fourier transform). For example, several lossy images and sound compression methods employ the discrete Fourier transform: the signal is cut into short segments, each is transformed, and then the Fourier coefficients of high frequencies, which are assumed to be unnoticeable, are discarded. The decompressor computes the inverse transform based on this reduced number of Fourier coefficients. (Compression applications often use a specialized form of the DFT, the discrete cosine transform, or sometimes the modified discrete cosine transform.)

3.1.1 Design and implementation of FFT

For this thesis, we implemented the most commonly used FFT, the Cooley–Tukey algorithm. This is a divide-and-conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1 \times N_2$ into many smaller DFTs of sizes N_1 and N_2 along with O(N) multiplications by complex roots of unity traditionally called twiddle factors[18]).

Below is an image of an 8-input FFT. With the recursive use of smaller and smaller components.

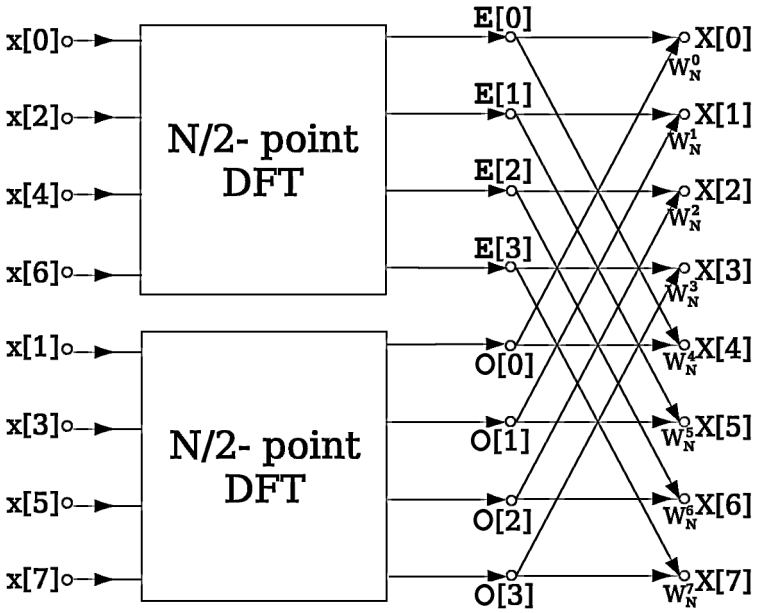


Figure 3.2: *Divide FFT Stages*

The more close representation of the design is :

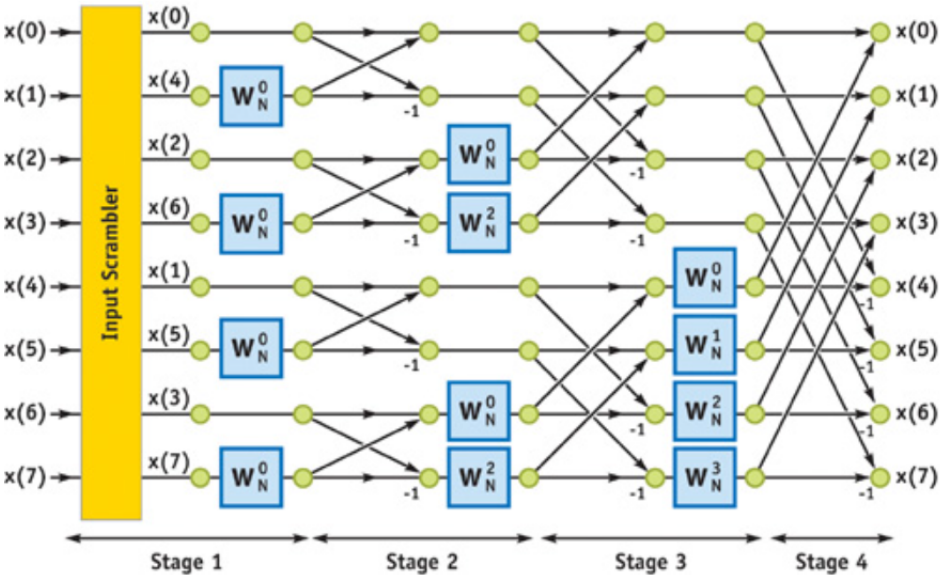


Figure 3.3: *Radix 2 FFT architecture*

The FFT component, we have implemented is a Radix-2 Decimation-in-Time (DIT), Pipeliend FFT. Some considerations of the implementation: - Radix-2 : Every level uses decomposition into half-sizes(mod 2) FFTs - Pipelined: Every stage/level store the output into a register. This is done to lessen the critical path which is now only an addition + multiplication

The circle that this FFT implementation needs to produce an output is log(N).

The implementation consists of 2 main components the scraber and the butterflies can be shown in the figure 3.4:

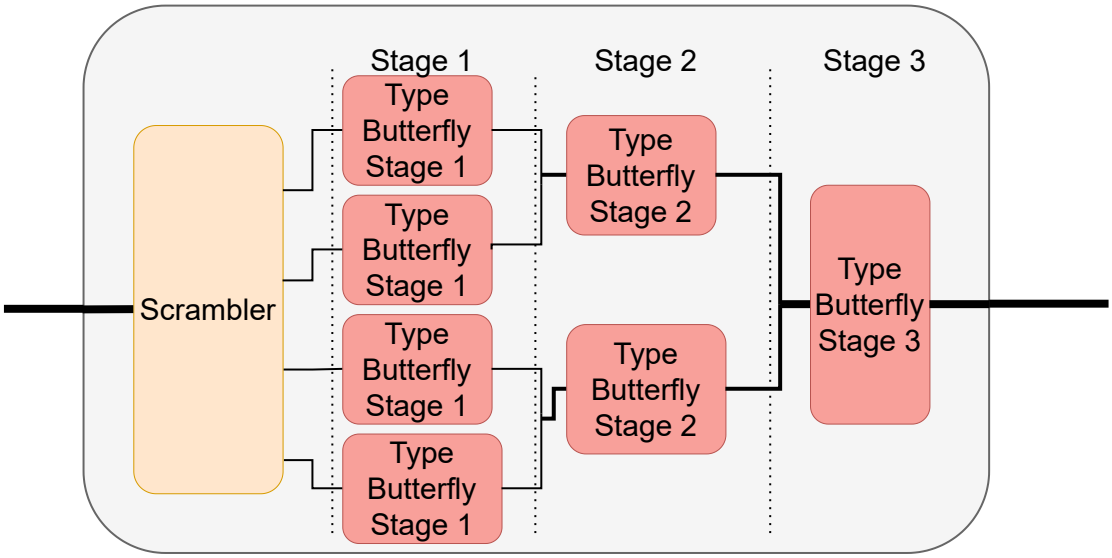


Figure 3.4: *FFT Architecture*

Twiddle Factors (Root of Unity)

The value of the Root of Unity is used in every stage multiplication. We have precalculated and placed those in a ROM inside the FPGA.

Scrambler

According to the 3.2 figure, in FFT Radix-2 we have to gather and process inputs into specific groups. This is more easily explained by following the 3.4 figure. Considering the inputs of a stage, the previous one is divided into two groups, the first one performs the even number calculations, and the second one the odd calculations. There is an easy trick to perform this operation by reversing the order of bits that represent the position of the input, and this scrambles the inputs to their correct orientation. We perform this with a component called Scrambler. Below is a representation of that operation on a 16-input Scrambler.

$$
\begin{aligned}
0000 &:= 0 \Rightarrow 0000 := 0 \\
0001 &:= 1 \Rightarrow 1000 := 8 \\
0010 &:= 2 \Rightarrow 0100 := 4 \\
0011 &:= 3 \Rightarrow 1100 := 12 \\
0100 &:= 4 \Rightarrow 0010 := 2 \\
0101 &:= 5 \Rightarrow 1010 := 10 \\
0110 &:= 6 \Rightarrow 0110 := 6 \\
0111 &:= 7 \Rightarrow 1110 := 14 \\
1000 &:= 8 \Rightarrow 0001 := 1 \\
1001 &:= 9 \Rightarrow 1001 := 9 \\
1010 &:= 10 \Rightarrow 0101 := 5 \\
1011 &:= 11 \Rightarrow 1101 := 13
\end{aligned}
$$

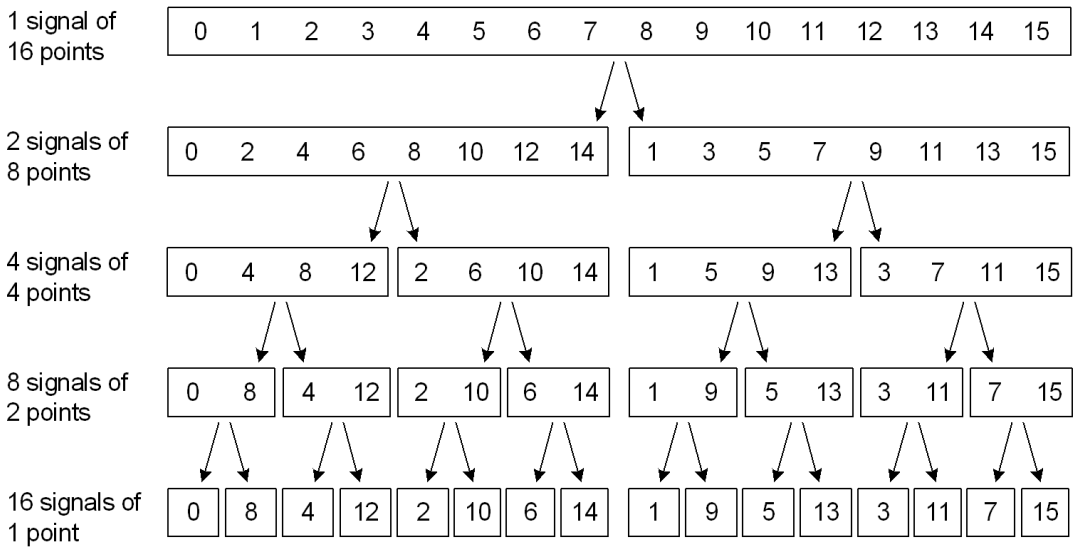


Figure 3.5: *Scrambler Architecture*

$1100 := 12 => 0011 := 3$
$1101 := 13 => 1011 := 11$
$1110 := 14 => 0111 := 7$
$1111 := 15 => 1111 := 15$

Type Butterfly Stage

The butterflies perform an operation :

$$y0 = x0 + x1 * twiddle$$

$$y1 = x0 - x1 * twiddle$$

We opted to alter the butterfly implementation, creating the type of butterfly by making it dependent on the stage. The inputs of each butterfly are $2 * *Stage$, and we perform the correct routing inside the butterfly component. The routing is matching the correct inputs to perform the operation. That is achieved by every input(i) combined with input(i+N) where N =2**(stage-1)/2 the number of pairs to produce an output. The output of each operation is performed by two more modules the adder and the subber. The adder produces the first output(i) and the subber the second one, output(i+N). The following logic can be represented in the following table for the 4-input or a 2-stage type butterfly:

I(0) + I(2) => O(0)
I(1) + I(3) => O(1)
I(0) - I(2) => O(2)
I(1) - I(3) => O(3)

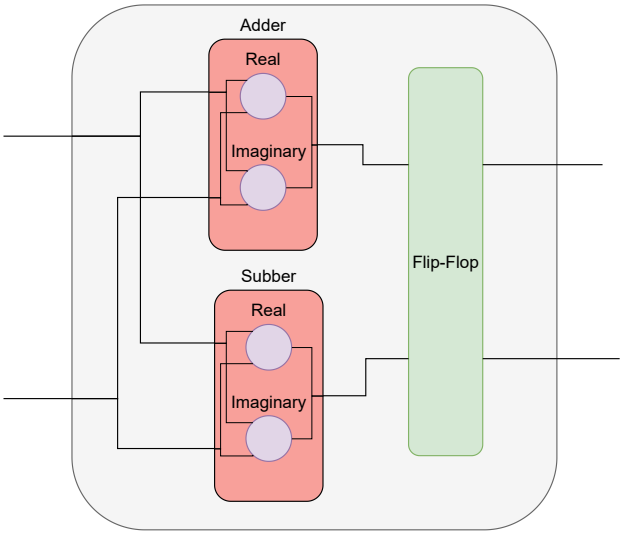


Figure 3.6: *Butterfly Set Stage 1*

The architecture of the type butterfly is depicted in figures 3.5 and 3.6 for a 1-stage and 2-stage respectively:

To achieve pipelining our design, we added a storage unit at the end of each butterfly, to minimize the critical path and allow for continuous operation of the components of the FFT. Another important distinction is to forward the twiddle factors into the lower module of the design. We observed a correlation the twiddle factors have with the stage, the number of inputs, and the current number of pairs using this formula :

$$twiddle((inputs/2) * i/(2 * *stage)/2)$$

where i = current number of pair

Adder | Subber

Both the adder and the subber take as input two 32 vectors of bits, with the first 16 representing the real part and the other the imaginary part, and output a single 32 vector of bits. The same format is followed by the twiddle factors. We opted for a fixed point arithmetic Q1.15 with 1 signed bit and 15 bits representing the fractional part of the number. That allows the application to run in a range of $[-1, 1)$. To perform each operation we divided the inputs into real and imaginary parts. The adder performs:

$$(i1.re + i1.im) + (i2.re + i2.im) * (t3.re + t3.im) = o.re + o.im$$

The operation can be further analyzed to :

$$o.re = 1.re + 2.re * 3.re - 2.im * 3.im$$

$$o.im = 1.im + 2.re * 3.im + 2.im * 3.re$$

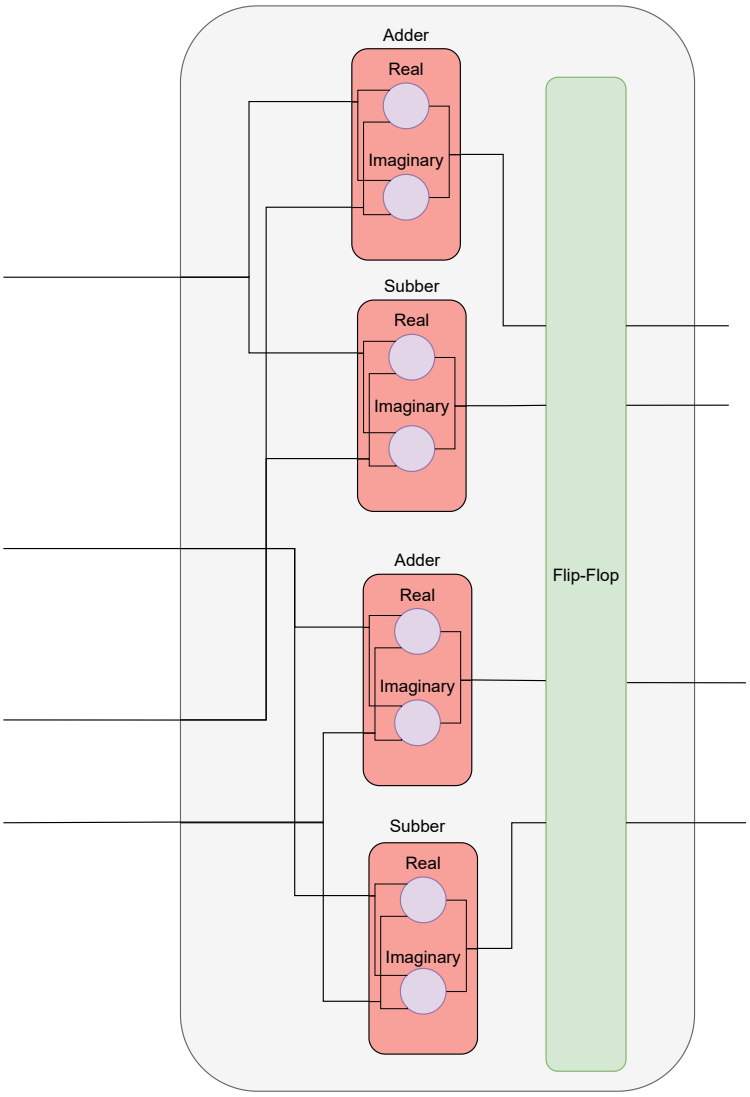


Figure 3.7: *Butterfly Set Stage 2*

Exactly the same as the subber:

$$(1.re + 1.im) - (2.re + 2.im) * (3.re + 3.im) = o.re + o.im$$

$$o.re = 1.re - 2.re * 3.re + 2.im * 3.im$$

$$o.im = 1.im - 2.re * 3.im - 2.im * 3.re$$

However, due to the arithmetic operations needing the bit vector length to increase, and because we wanted to keep it to 32 size, we truncated the results to fit that size. That meant that passing through every adder and subber the number was divided by two.

The latency of our implementation is log(N), incase of parallel inputs, while incase of serial arrival, the latency of the first result in N*log(N).

3.2 Spacial Redundancy

Spatial redundancy is the intentional multiplication of critical components or functions of a system with the goal of increasing the reliability of the system.

Besides the usage of spatial redundancy in digital design operations and kernels, it is also used in physical parts of many safety-critical systems, such as fly-by-wire and hydraulic systems in aircraft, An error in one component may then be out-voted by the other ones. All components of a system must fail before the system fails. Since the percentage of error in many parts is often small and the subcomponents are expected to fail independently, the probability of all three failings is calculated to be extraordinarily small, and it is ofter outweighed by other risk factors. Redundancy may also be known by the terms ”majority voting systems”[2] or ”voting logic”.[3]

However, specifically in ionizing radiation environments where the percentage of fault is closely correlatable with the area a component requires, increasing the number of components is not often able to produce the required results. For that reason, space exploration is needed for which different techniques fit a diverse range of applications in a diverse range of environments.

3.2.1 DMR

Dual Modular Redundancy (DMR) is a form of redundancy that uses two or more identical modules to independently process the same data. These modules are monitored to detect any discrepancies between them. If a discrepancy is found, the data is discarded and the system is reset.

The system is able to detect an error but not correct them. Although DMR is most commonly used in an application with a lower catastrophic error percentage, it offers smaller overhead and time to reconfigure than other methods we will explore and explain further. It is better than some other alternatives, where the error detection must happen in the software part of the design by comparing to some golden result. The hardware detection

allows to perform reconfiguration of the FFT in real-time and corrects a possible error in the design.

Our proposed architecture is the following:

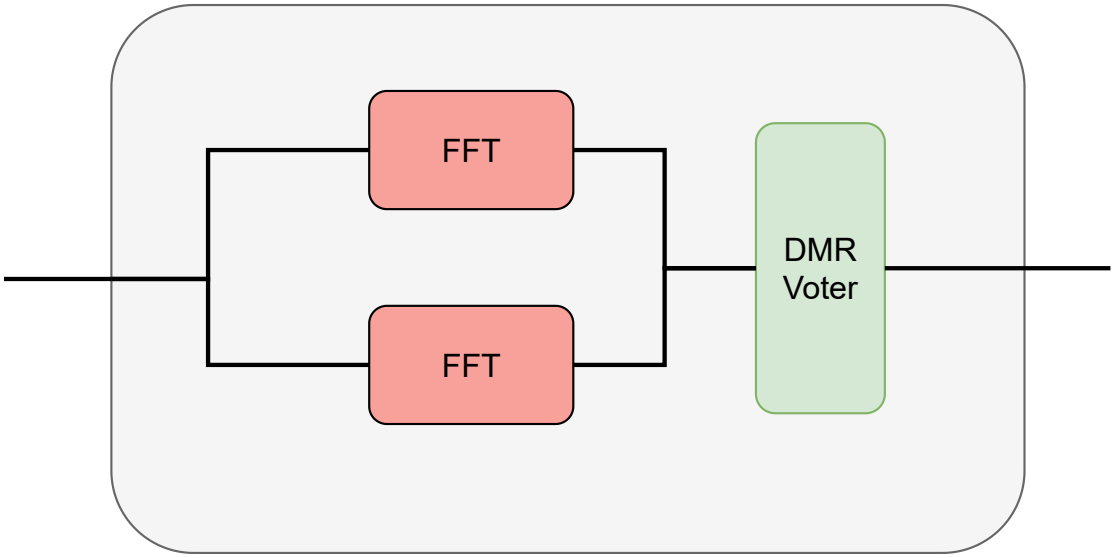


Figure 3.8: *DMR Architecture*

The voter compares the output of the two modules, and when an error is detected it outputs a flag that allows the ps to understand the error.

Voter output	Flags
same results	00
different results	11

Table 3.1: *DMR Voter to DMR Flags*

3.2.2 TMR

Triple modular redundancy (TMR) is a fault-tolerant form of redundancy used in safety-critical systems. It involves the use of three separate processing modules that independently perform the same task. The output from the three modules is then compared to verify that all three results are identical. If the results are not identical, then the system will enter a fault state and initiate an appropriate response. This type of redundancy is used to ensure that the system is highly reliable and that any errors can be detected and addressed quickly. TMR provides increased reliability by ensuring that any single error or fault in one of the three circuits will be detected and corrected by the other two. TMR is commonly used in applications such as aerospace systems, nuclear power plants, and medical equipment.

In this thesis, we opted for triplicating the FFT kernel and adding a voter connected to their outputs. The system is able to both detect and correct possible errors that may occur. The voter is able to compare the result of the FFT kernels. Even though many academic explorations [34], [35] have been conducted in the search for the most suitable voter for

fault tolerance, we used a single voter in the output. In case of one kernel differentiating from the other two then the voter outputs the correct result and raises a flag for the faulty kernel to be corrected. If all three kernels disagree in their results the voter selects the first FFT and raises that all the kernels needed to be reconfigured.

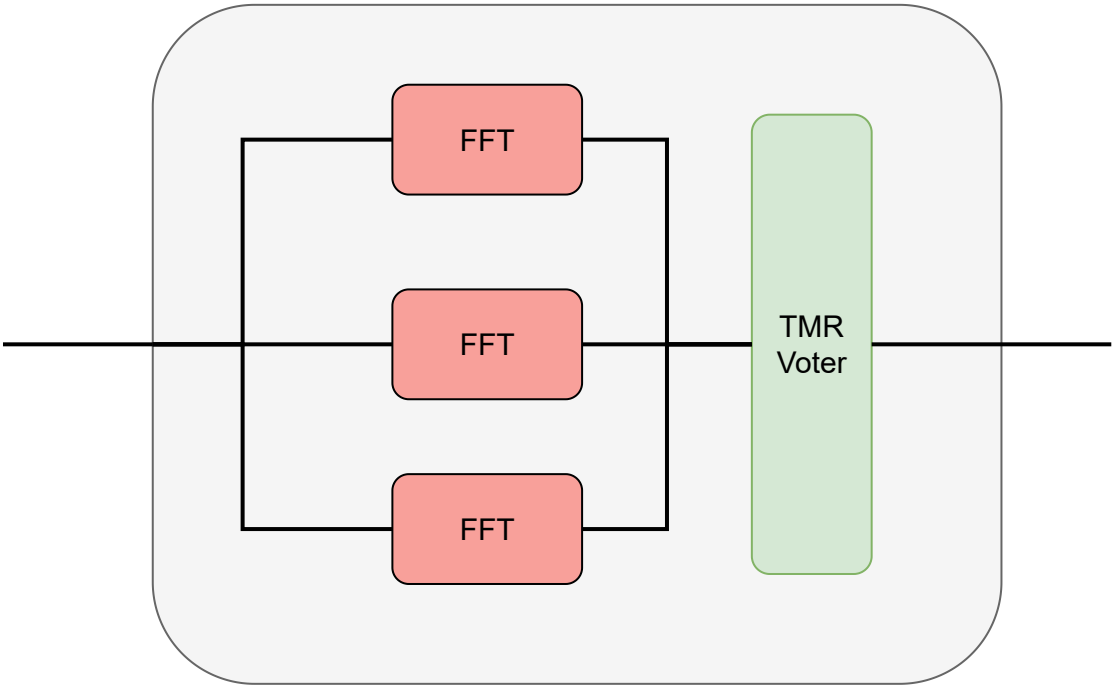


Figure 3.9: *TMR Architecture*

Voter output	Flags
all components same	000
first different, second & third same	001
second different, first & third same	010
third different, first & second same	100
all different	111

Table 3.2: *TMR Voter to TMR Flags*

3.3 Temporal Reduncancy

3.3.1 Simple Temporal Redundancy

Besides the concept of spatial redundancy where the duplication or triplication of critical kernels in the design is implemented, there is another form of redundancy spreading over time. Temporal redundancy is a type of data redundancy that occurs when the same operation is executed multiple times and can be used either as a complementary or main tolerance technique as a fault-tolerance architecture [36]. It's optimal for applications favoring checkpoint implementations where the faulted results can be discarded and the kernels can be repaired. The use of multiple operations tends to lead to higher energy consumption from the application where the modules can be set to idle waiting for inputs

to arrive, and can often lead to conflicts with requirements for real-time applications. Due to that, temporal redundancy is often used in modules with specific structures, resulting in simpler outputs, while programs with complex internal states and structures may result in higher performance overheads. However, with the time dependence of this approach, temporal redundancy is vulnerable to errors occurring between the start of multiplied operations.

In our thesis, we decided to execute the same input N times as many as the number of inputs of the FFT. This approach could detect errors happening during the re-execution phase of the operation. Imagining a continuous arrival of input data that approach could detect and correct errors during 1 - 1/N cycles. For that reason, further exploration is necessary regarding the favorable Pareto point featuring energy consumption and the percentage of errors between different bursts of executions.

Our approach includes : - an input repeater that multiplies the input N times - the FFT kernel - a mechanism observing the output of the FFT kernel emitting the correct result and raising a flag in case of a faulty operation. - an FSM that ensures that the same errors won't influence the error detection or even transient imbalances in the output could be detected.

The diagram of the FSM is the following:

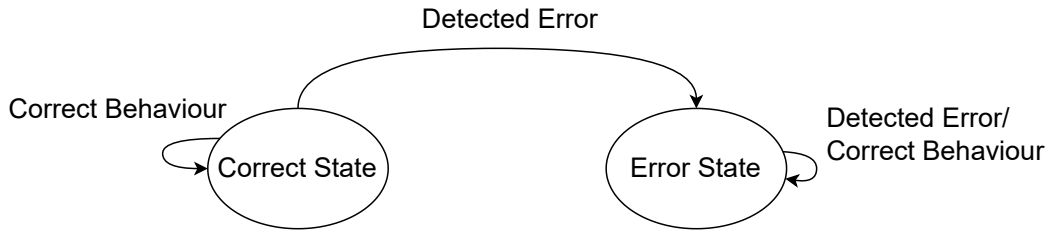


Figure 3.10: *Temporal FSM*

The flag table is :

State	Flags
Correct State	0
Error State	1

Table 3.3: *Temporal FSM Flags*

A holistic view of the whole kernel:

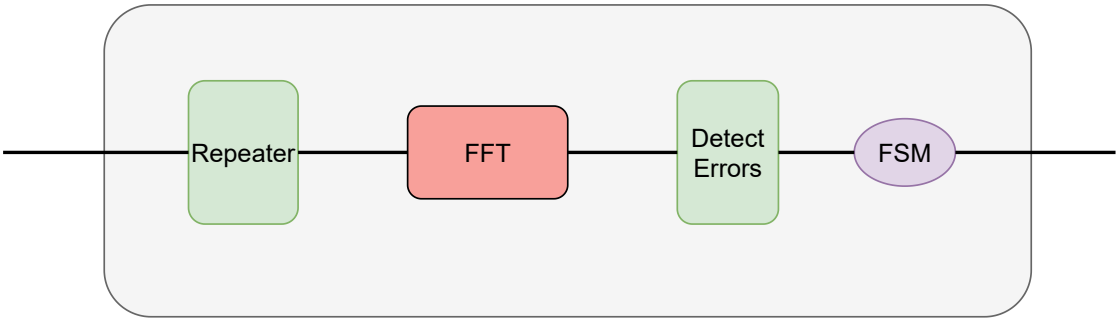


Figure 3.11: *Temporal Architecture*

3.4 Hybrid between Spatial and Temporal Redundancy

3.4.1 DMR Temporal

Another approach is to combine spatial redundancy with temporal one in a single kernel taking advantage of both techniques. We opted for a dual spatial redundancy and keeping the time repeating the input data N times.

The mechanism responsible for detecting errors in this approach is configured to also compare and detect the different outputs of the two FFT kernels.

The FSM has been altered to fit the extra kernel :

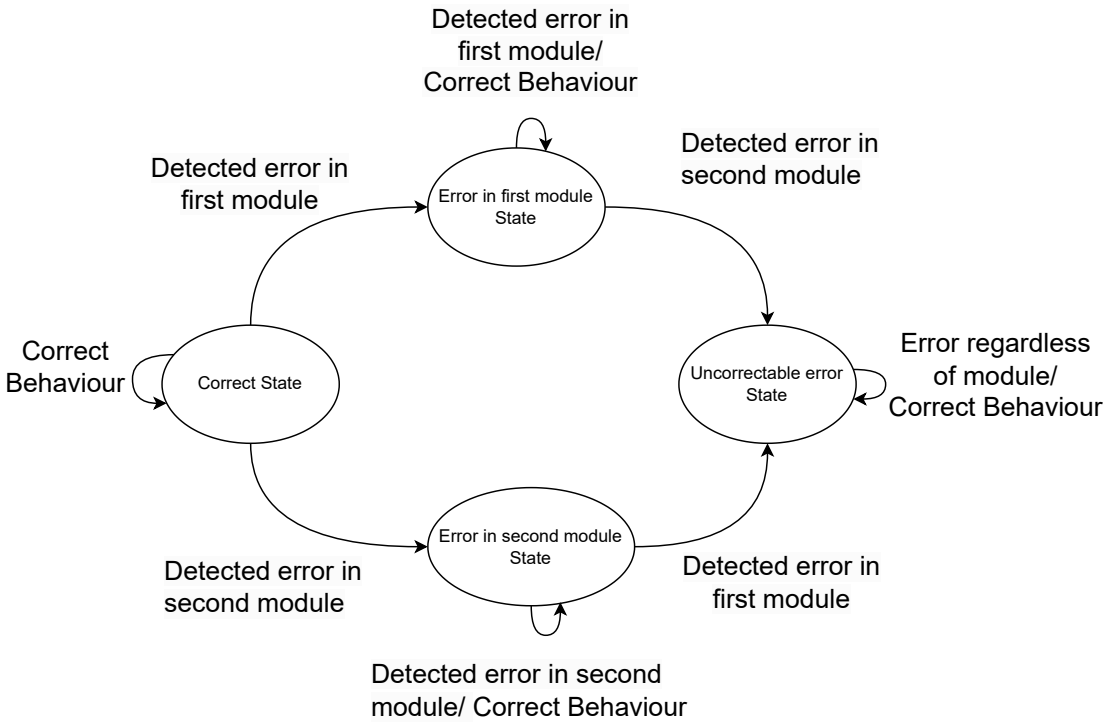


Figure 3.12: *DMR Temporal FSM*

The flag table is :

State	Flags
Correct State	000
Error in first	01
Error in second	10
Uncorrectable error	11

Table 3.4: *TMR Voter to TMR Flags*

A holistic view of the whole kernel:

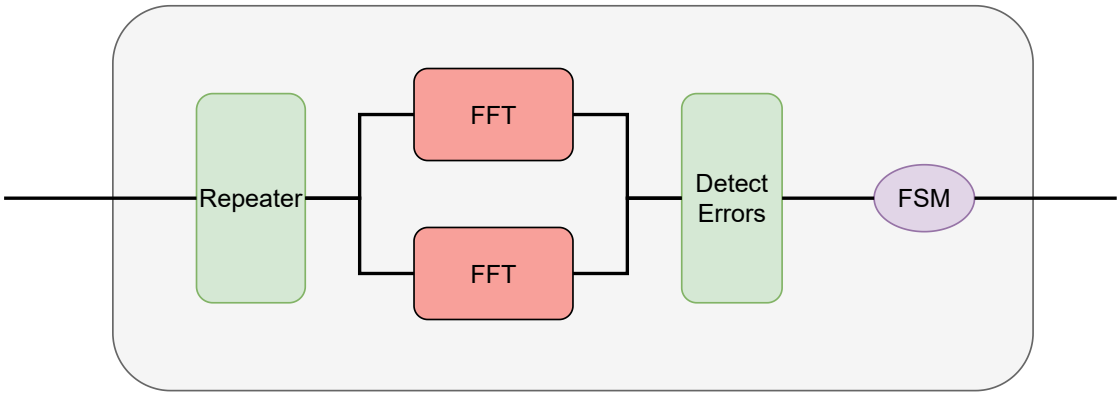


Figure 3.13: *DMR Temporal Architecture*

3.5 Fine Grain Spacial redundancy

Previous research has suggested that triplication may not always be the optimal approach for mitigating the effects of ionizing radiation in the space environment [citation needed]. As previously explained, there is a direct correlation between the size of a design and the percentage of single event upset in the device. This often necessitates decisions regarding which critical parts to protect and which to neglect. This methodology can also be applied to the inner components of each kernel, providing more control for device reconfiguration. Using smaller components requires less reconfigurable time and reduces overhead. In this thesis, we explore two kernels, one where triplication is performed at the stage level, and the other at the butterfly level of each FFT kernel.

3.5.1 FFT Stage TMR

The TMR Stage kernel operates by triplicating each stage and passing the resulting outputs through a voter that compares them to detect and correct possible errors. The voter module is identical to that of the TMR kernel but is now present at the end of every stage. In the event of an error, the targeted module can be reconfigured, minimizing device downtime as compared to reconfiguring the entire device.

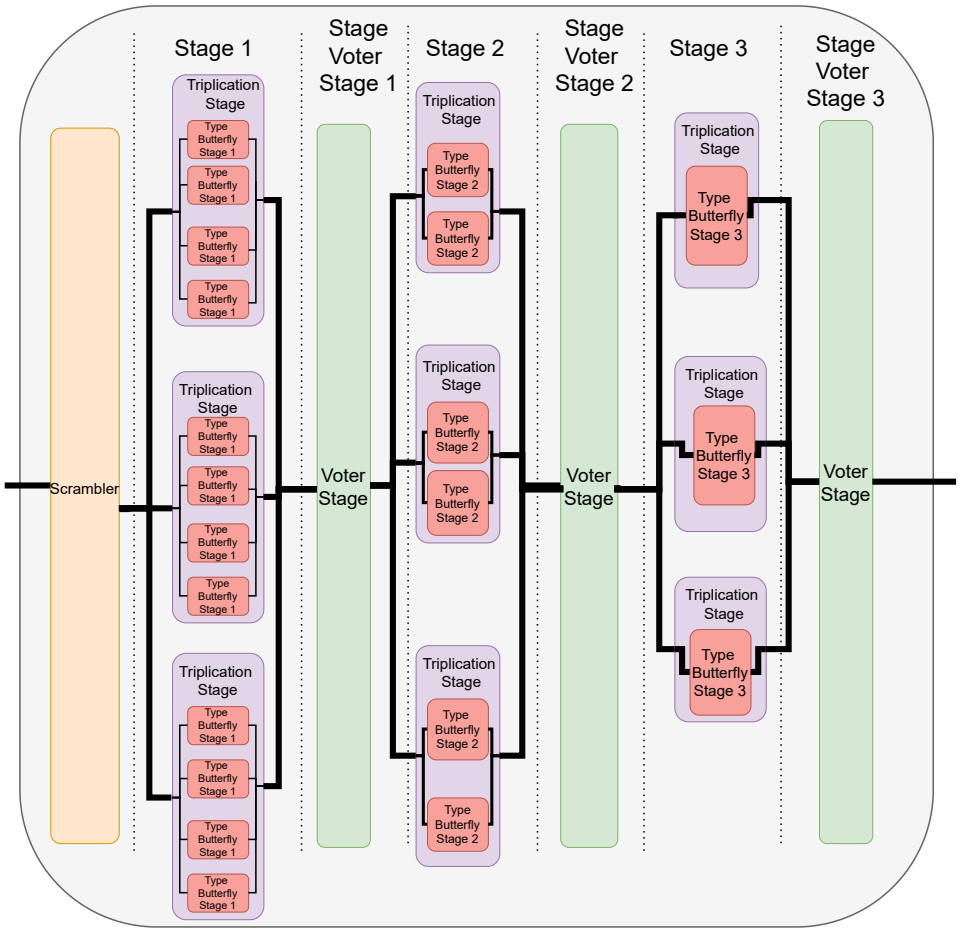


Figure 3.14: *TMR Stage Architecture*

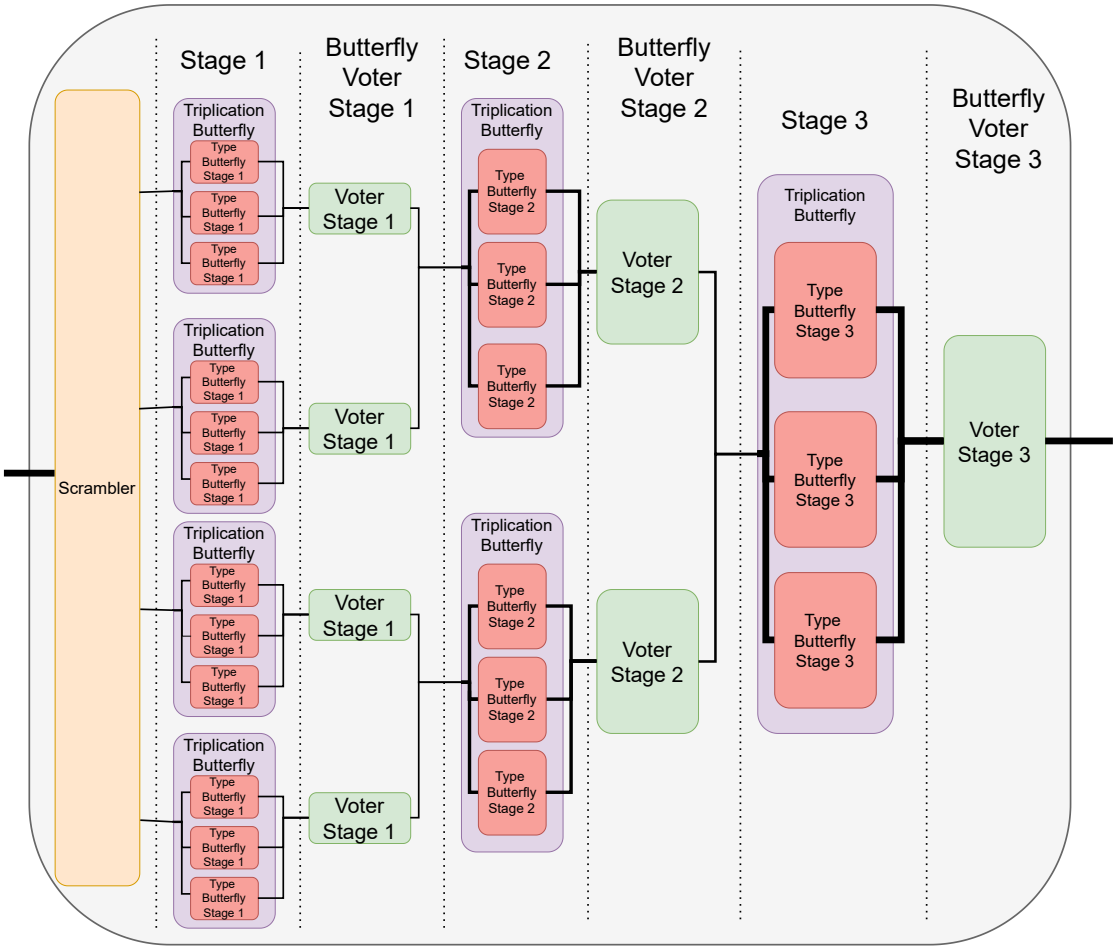


Figure 3.15: *TMR Butterfly Architecture*

3.5.2 FFT Butterfly TMR

Similar to the TMR Stage kernel, the TMR Butterfly kernel triplicates components, but on a butterfly level. This approach provides greater flexibility, but the creation of partitions and startup time overheads for partial reconfiguration is more significant in this design. The voter is slightly modified to consider the difference in input size at each level, but the error vector for incorrect outputs remains identical.

3.6 Dynamic Partial Reconfiguration

The FPGAs are logicaly dividid into two layers, the hardware layer, which include the logic resources, the routing resources, the DSP block, the RAM, the IO and others. Most FPGAs are able to be reconfigurable during their deployment. However many times, the situation does not require a full reboot of the device, and rather smaller parts need to be reconfigurable. This process is called partial reconfiguration. Partial reconfiguration in FPGAs is a feature that allows sections of an FPGA to be reconfigured while the FPGA is still running. This allows the FPGA to be used to its fullest potential, as different sections of the FPGA can be used for different tasks at any given time. This technology enables

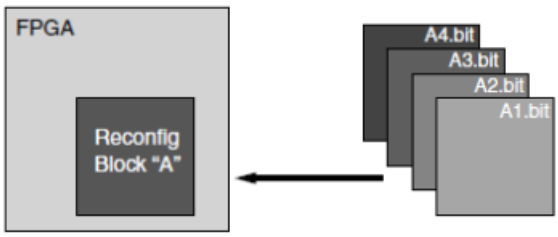


Figure 3.16: *Dynamic Partial Reconfiguration 1*

dynamic reconfiguration of the FPGA, allowing the FPGA to be used as a multi-function device. This technology has been explored in the correction of the single event upsets (SEU), through various forms either as reconfiguring manually areas of the FPGA fabric or as part of the Configuration Memory Scrubbing [37].

It also allows for the FPGA to be more energy-efficient as it only uses the resources that are necessary for the current task. Nevertheless, that functionality minimizes the area need in the FPGA fabric, as more and more modules can be stored in the external memory. That leads to lower deployment costs and even development time as more and more components can be developed independently ignoring the functionality of the whole system. Besides the dual functionality that, would have tremendous benefits on the field, dynamic partial reconfiguration allows us to combat problems with regard to the fault of a component in the design. This could be for example one out of the triplication modules used in spatial redundancy that malfunctions. Instead of disabling the whole device and reprograming it from the start, we have the opportunity to keep running the other two modules, while the faulty one would be dynamically reconfigured, in a smaller time than the time needed for the whole design. With this process, we are able to eliminate the downtime of the device, while dealing with the faulty component.

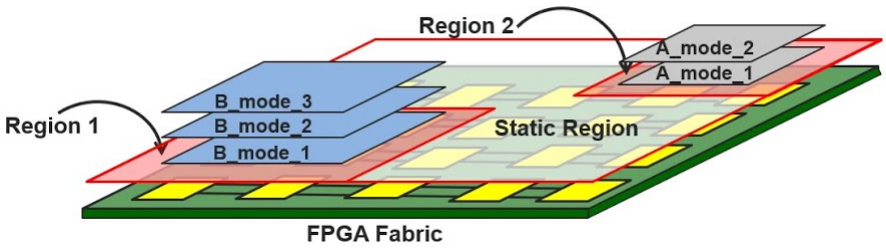


Figure 3.17: *Dynamic Partial Reconfiguration 2*

Another technic is the static partial reconfiguration of the design, however, the most significant problem of that design needs to stop working at the time of reconfiguration. That limitation is the reason most implementations of partial reconfiguration prefer to use the dynamic approach.

Xilinx FPGAs in their tool flow offer partial reconfiguration floorplanning and bitstream generation. In addition, many interfaces are used to be able to reconfigure the device.

ICAP

ICAP or Internal Configuration Access Port allows the user to program the device from within the FPGA itself. The ICAP should be explicitly enabled by the user design. The ICAP is unable to perform the initial reconfiguration, but when it is finished, allows the user decideto reconfigure the device partially or fully. One significant advantage the ICAP reconfiguration method has, is its configuration speed and high throughput, with an operating frequency of 100 MHz and throughput of up to 67 MBps. To access this interface of the FPGAs device, Xilinx provided the IP CORE AXI HWICAP. The software application running on the ARM-Processor is responsible for reading the configuration bitstream and delivering them to the ICAP interface.

PCAP

PCAP or Processor Configuration Access Port is a unique interface that enables access from a hard processor to the configuration module. It consists of a quickly found concept and is only present in the newer Xilinx Devices including the UltraScale+ MPSoC line of boards. Due to the importance of its functionality Xilinx develop a new API, to help with the accesibility of the reconfiguration. XilFPGA library [38] provides an interface for the users to configure the programmable logic (PL) from PS. The library is designed to run on top of Xilinx standalone BSPs. It acts as a bridge between the user application and the PL device. It provides the required functionality to the user application for configuring the PL device with the required bitstream. Some of the features supported in the Zynq UltraScale+ MPSoC platform are among others Full bitstream loading and Partial bitstream loading. 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.

In all applications, the configuration bitstream is transferred to DDR from an external memory (SD card) or using an SPI memory bus (QSPI). This is due to the fact that flash memory is widely used in all space applications, and currently, there is no readily available radiation-tolerant alternative.

In this thesis, we chose the PCAP approach as it is newer, better supported, and recommended by Xilinx. We implemented partial reconfiguration to crucial computational modules of the kernel, such as modules inside the FFT, such as adders and subtractors. We excluded the voters, the FSM, the repeater, and the error-detecting modules in the previously described architectures. The reason for this decision is that without these core components, the application does not function according to the specification since no redundancy or error mitigation technique is implemented for them. Additionally, their small size compared to the FFT main module does not offer significant time improvements over already necessary overheads needed for the partial reconfiguration to start. This makes the reconfiguration of these individual components inefficient both in detecting errors in these modules and the fact that the application is inoperable during the reconfiguration. For the initial stage of our application, we have made the decision to utilize an SD card to store all of the necessary bitstreams. These bitstreams will be subsequently transferred

to the DDR memory. This approach will enable us to transfer the stored data to fulfill the requirements of the partial reconfiguration process as and when needed.

In contrast, we can reconfigure an instance of an FFT kernel in which, for example, the application detects an error, while the other kernels continue to function. This is what partial reconfiguration offers our design.

3.7 AXI-Lite

Advanced eXtensible Interface 4 (AXI4) [39]is a family of buses defined as part of the fourth generation of the ARM Advanced Microcontroller Bus Architecture (AMBA) standard. The AMBA specification defines 3 AXI4 protocols:

- AXI4: A high-performance memory-mapped data and address interface. Capable of Burst access to memory-mapped devices. - AXI4-Lite: A subset of AXI, lacking burst access capability. Has a simpler interface than the full AXI4 interface. - AXI4-Stream: A fast unidirectional protocol for transferring data from master to slave.

The Vivado design suite offers the ability to create custom IP with AXI4 interfaces. These can be directly connected to ARM Processing System. The AXI4-Lite interface consists of five channels: Read Address, Read Data, Write Address, Write Data, and Write Response.

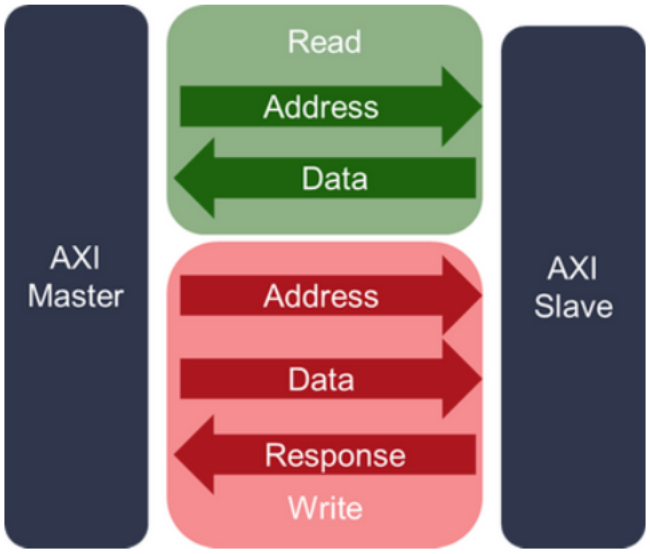


Figure 3.18: *AXI4-Lite*

The signals included in an AXI4-Lite Interface are :

```
AXI4-Lite
├── Global Signals
│   ├── ACLK : Clock source
│   └── ARESETN : Global reset source, active low
├── Read Address Channel
│   ├── ARADDR: Read address
│   ├── ARCACHE: Memory type
│   ├── ARPROT: Protection type
│   └── ARVALID: Read address valid
├── Read Data Channel
│   ├── RDATA: Read Data
│   ├── RRESP: Read response
│   ├── RVALID: Read valid
│   └── RREADY: Read ready
├── Write Address Channel
│   ├── AWADDR: Write address
│   ├── AWCACHE: Memory type
│   ├── AWPROT: Protection type
│   ├── AWVALID: Write address valid
│   └── AWREADY: Write address ready
├── Write Data Channel
│   ├── WDATA: Write data
│   └── WSTRB: Write strobes
└── Write Response Channel
    ├── BRESP: Write response
    ├── BVALID: Write response valid
    └── BREADY: Response ready
```

In order to perform transactions in the protocol, a handshake process is required. All five transaction channels employ the same VALID/READY process to transfer addresses, data, and control information. This two-way flow control mechanism enables both the master and slave to regulate the rate at which the information is exchanged between them. The source of the information generates the VALID signal to indicate when the address, data, or control information is available, while the destination generates the READY signal to indicate that it can receive the information. The handshake process is completed if both the VALID and READY signals in a channel are asserted during a rising clock edge.

In this thesis, an AXI-Lite interface is employed to facilitate communication from the processing system to the programmable logic, in order to transmit data, manual resets to the kernels, and receive the output as well as the signal indicating errors in the design. The function and testing of our own application do not require a more complex method, such as DMA transfers. The memory map allows us to directly influence the input and read the output necessary for testing for single-event upsets that may affect the functionality of the kernels.

3.8 DSPs vs LUTs

The field-programmable gate array (FPGA) consists of look-up tables (LUTs), which serve as the fundamental building blocks of the device. A LUT is capable of implementing any logic function of N Boolean variables. Essentially, it functions as a truth table, where different combinations of inputs yield various functions, ultimately leading to output values. Contrary, Digital signal processors (DSPs) are specialized multiply-accumulate hardware accelerators that significantly speed up the execution of signal processing functions.

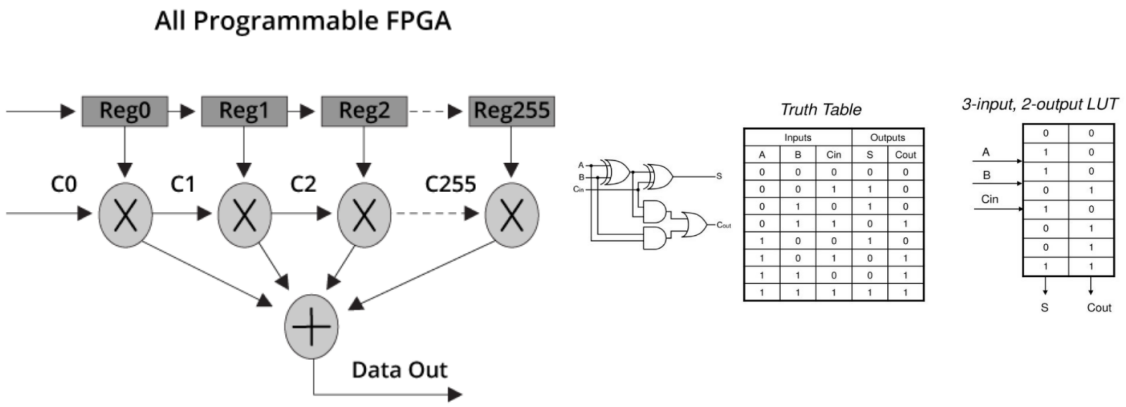


Figure 3.19: *DSP vs Lut*

Modern FPGAs can map specific functions directly to the device, and conversely, every DSP functionality can be emulated using LUTs. There exists a trade-off between the availability of DSP slices in the FPGA fabric and the speedup in the application and lower configuration memory. In addition, when attempting to mitigate the effects of ionizing radiation in space, the area in the FPGA fabric in which the kernels require execution is crucial. A direct correlation between the area and the percentage of signal even upset in the design is present, requiring an exploration to determine the optimal configuration and balance the use of LUTs and DSP.

This thesis explores our designs by testing them with the use of the maximum available and necessary DSPs, as well as excluding them from the design. The purpose of this approach is to test the kernels in extreme scenarios and understand how such decisions affect the implemented design. The use of LUTs is expected to increase the percentage of errors in our design, as the configuration memory is larger than the DSP implementation.

3.9 How to Perform Fault Injection

Ionizing radiation has the potential to induce undesired effects in most silicon devices, which are broadly referred to as single-event effects (SEEs). In most cases, these events do not cause permanent damage to the silicon device and are thus called soft errors. However, soft errors can reduce the reliability of a device. While soft errors are considered unavoidable within commercial and practical constraints, Xilinx has integrated soft error detection and correction capabilities into many of its device families. Moreover, Xilinx

offers an IP core with fault injection capabilities, named SEM IP, which helps in validating the design and measuring the device's tolerance. Specifically, for the Ultrascale+ boards, the UltraScale Soft Error Mitigation Controller v3.1 is available [28].

The use of the SEM IP as a means of simulating the effects of ionizing radiation on semiconductor devices has been widely recognized in the literature. This powerful tool allows for the injection of errors into the configuration memory of the programmable logic during runtime, effectively replicating single event upsets (SEUs) that occur in real-world environments. Despite the fact that the SEM IP includes mitigation techniques such as error detection, correction, and classification, it should be noted that a faulty SEM IP component could potentially interfere with the configuration memory, ultimately resulting in catastrophic failure of the entire device.

Several studies have been conducted in the literature regarding the efficacy of scrubbing techniques [40]. Interestingly, the results of these studies have revealed that the internal scrubber methodology employed by the SEM IP may be prone to certain limitations that could impact its overall integrity. The Xilinx internal scrubber has 4 general methods of inoperability:

1. SEU causing the SECDED to either: not be able to correct and therefore potential fault accumulation can occur (within a frame) or improperly correct a frame thus creating massive interconnect errors

2. Internal scrubber circuitry gets hit and causes improper function or cease of function

3. Utilized BRAM (pico-blaze of internal scrubber uses BRAM) gets hit and causes malfunction.

4. ICAP interface becomes inoperable

Therefore, it is crucial to introduce a radiation-tolerant SEM IP to ensure the reliability of semiconductor devices operating in harsh environments.

One potential solution is to implement a watchdog mechanism that checks essential bits of the SEM IP every time it communicates with the configuration memory. If any of these essential bits are found to be incorrect, the watchdog can trigger a recovery mechanism to restore the SEM IP to its original state. Another approach is to use triplication of the SEM IP, where three identical instances of the IP are used, and their outputs are compared. This technique can improve the overall reliability of the system by using a voting mechanism to select the correct output in the event of radiation-induced errors affecting one or two instances of the SEM IP. Partial reconfiguration of faulty components is another potential solution, where the IP can be partitioned into smaller subcomponents, and only the faulty subcomponents can be reconfigured while the rest of the IP remains operational. This approach can minimize downtime and reduce the impact of radiation-induced errors on the system's overall performance. By implementing these techniques, a radiation-tolerant SEM IP can significantly improve the reliability of semiconductor devices operating in harsh environments.

However, it is important to note that the aforementioned considerations extend beyond the scope of this thesis. Therefore, the focus of this work is solely on the deployment of the FPGA injection mechanism to simulate real-time environments.

The ports of the SEM IP v3.1 are depicted in the following figure:

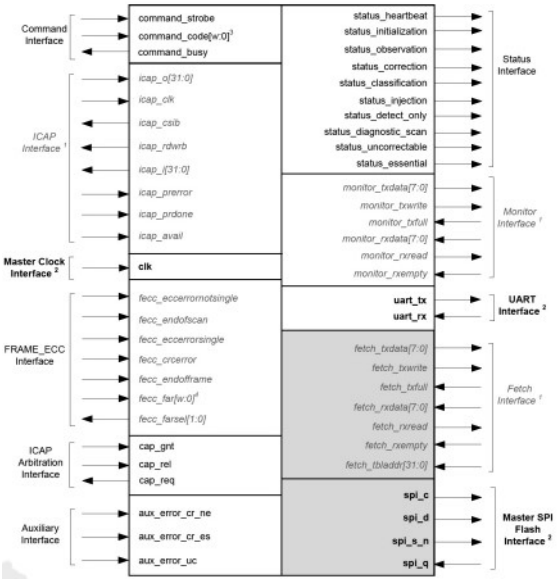


Figure 3.20: *Sem IP v3.1*

In order to communicate with the monitor interface of the SEM IP, we are using AXI UartLite IP clocked at 100 MHz. This allows us to send commands and receive reports. This is the recommended method from Xilinx as it is more verbose than others. The commands [28] are the following

Reset

The ”R” command is used to perform a software reset. The binary format of the command is:

```
R {2-digit hex value}
```

The controller transmits an initialization report if the command is successful.

Idle

The ”I” command is used to place the SEM core into the Idle state. This state is critical as only from it, the injection command can be performed. The binary format of the command is:

```
I
```

The controller transmits an initialization report if the command is successful.

Error Injection

The “N” command is used to perform an error injection by LFA (linear frame address). The controller only accepts this command when in the Idle state. The format of the command is:

```
N {11-digit hex value}
```

The binary format of the command is :

```
1100 0000 0000 ssLL LLLL LLLL LLLL LLLL wwww wwwb bbbb
```

, where

- ss are the hardware slr number
- L represents the 18-bits of the linear frame address
- w the word address ranging from 0 to 91
- b the bit address ranging from 0 to 31

Configuration Memory Scrubbing

The ”O” command is used to command the scanning electron microscope (SEM) IP during the observation state. During this state, the core is unable to inject errors; however, it can identify and rectify them by employing the error correction code (ECC) and cyclic redundancy check (CRC) algorithms. The command format for this operation is as follows:

```
O
```

[This paragraph is for the purpose of this draft empty, It will be finished until the final presentation]

To add the functionality of the SEM IP in our design, we had to consider some of the limitations and adduce more functionality to our design. The most specific limitation of our design is the fact that the SEM IP reads the configuration memory through the ICAP. However, in the booting of the device, the control of the configuration memory is not enabled. For that reason, we stalled the initialization of the SEM IP until the ICAP is enabled. The ICAP register is in the 0xFFCA3008 address for UltraScale+ devices, and we have to clear the 0 bit for the ICAP to be enabled. Then using some GPIO pins that allow communication between the processor and the programmable logic, we start the SEM IP. The example design provided by Xilinx includes the monitor UART interface which we include in our design.

The whole design can be depicted in the following diagram :

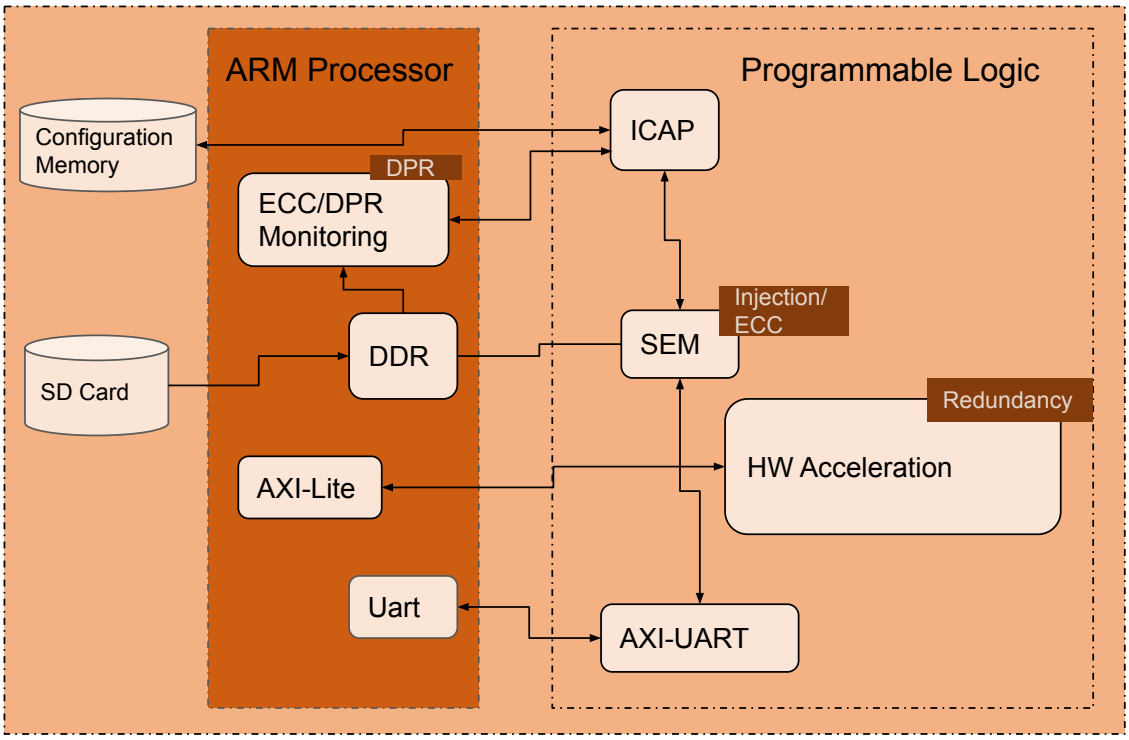
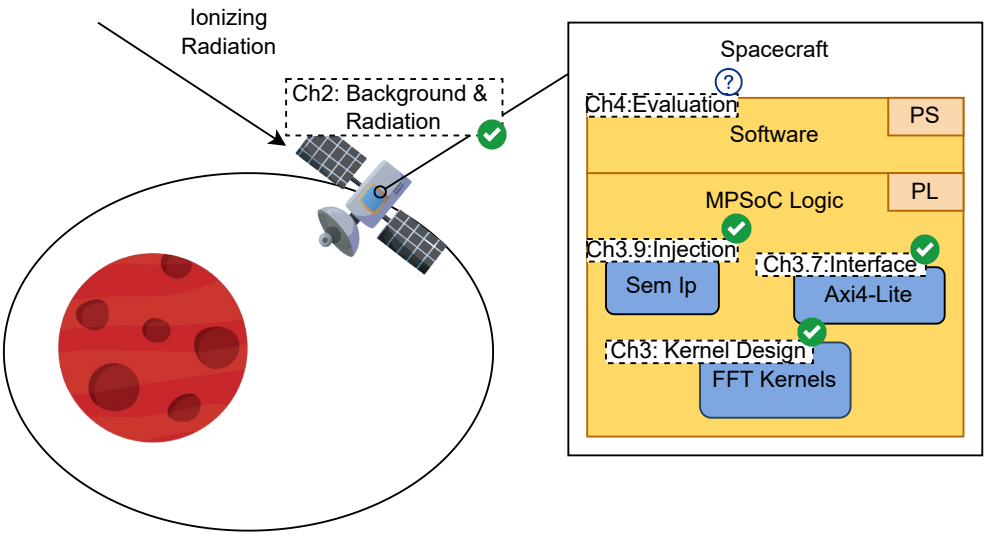


Figure 3.21: *Whole Diagram*

Chapter 4

Validation and Evaluation of Fault-Tolerance Mitigation Techniques

In Chapter 4, the results of injection tests performed on Xilinx's MPSoC UltraScale+ ZCU106 board are presented, which are covering all of the architectures proposed in Chapter 3. The chapter provides an explanation of how errors were introduced during the tests, and it includes a comparison of the performance metrics and resource utilization of each kernel across input sizes of 8, 16, and 32. In addition, the injection campaign is described in detail, including the methodology that was followed during the execution of the tests.



4.1 Experimental Setup

We selected Xilinx's MPSoC ZCU106 SoC (System-on-Chip) board as our implementation platform. The reason for this selection is the usage of this platform on space-grade applications and missions. The relatively lower cost and programmability of this board make it ideal for this thesis. The board includes among others a quad-core Arm Cortex-A53 applications processor and 16nm FinFET+ SRAM-based programmable logic. All those methods explored in this thesis, however, are not board dependent. It is a general method that can be employed for all types of SRAM-based FPGAs. The MPSoC architecture allows the user to access and change the FPGA configuration bits.

The software that allowed us to develop and implement the aforementioned architectures is the Vivado Design Suit (2020.2). It provided us with the right tools needed to develop our design, placing and route our architecture on the FPGA fabric, and prove the necessary IP developed by Xilinx. Besides that, for the software applications we used the Vitis software development kit to implement the software code necessary fro running our bare metal project in the ARM processor. The Vivado Design Suit also has the ability to generate the bitstreams necessary for both the full and the partial reconfiguration of the device, and the necessary .ebd file [29]. Using the following command in the .xdc file in the synthesis and implementation runs of the Vivado, the tool generates the .ebc and .ebd files.

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

- The EBC file is a reference file and contains the memory cell content. This is the same content read back by the SEM controller during the SEU readback of the FPGA.
- The EBD file shows the bits of the SEU Readback that are marked as essential. The EBD file is used to mask the EBC file meaning that a 1 in the EBD file corresponds to an essential bit in the EBC file.

However, it is important to note that this file is not the same as the bitstream used to program the board.

4.1.1 Main application and testing ground

Our main application sends 32-bit vectors, with the first 16 being the real and the other being the imaginary part, and expects the same format of data after being processed by the Fast Fourier Transform algorithm, implemented in hardware. We opted to send all the inputs in parallel in memory mapped registers. That allowed us to take advantage of our inherently parallel design, by continuously feeding data through the hardware.

As we mention in Chapter 2 the main cause of an error, is the upsets that are present in the design. The ratio that shows us the availability of the design follows in common critical systems the ”five-neds” standard, indicating an availability of 99.999%. The ratio

of the time the system is operating correctly to the total running time of the application is the definition of Reliability. Reliability is the likelihood that a system will remain operational for the duration of a mission. Higher levels of reliability are important in critical missions such as space satellites or in industrial control, environments where failure could mean loss of life or failure of costly missions. Availability expresses the fraction of time a system is operational. High availability is critical in many applications in which every minute of downtime translates to potential catastrophic errors of significant revenue loss. Both of these metrics can be extracted by calculating the error rate of the application. The error rate is the ratio of the number of erroneous data to the total number of data received.

4.1.2 Overview of the test setup

In order to inject errors in the configuration memory of the design, the first option follows the procedure explained in [41], 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 precalculated output, counting how many times a wrong output has been produced. This can be used as a figure of the metric of the sensitivity of the various techniques. However, this does not take into account that the injected error may not be a part of the design. Therefore, a more direct approach is necessary to be able to test and inject errors only in the essential bits part of the design.

4.1.3 Injection Campain

There are some ways to obtain the list of the essential configuration bits [42] associated with a design. Xilinx provides some methods to generate this list. However, in order to extract from that list the important subset for our kernel, we have to understand the topology and the correlation of the .ebd file lines with the area in the FPGA fabric. We use as guidance the ACME tool [41] which can provide a subset of the essential bits associated with a sub-module embedded in the FPGA fabric. However, the ACME tool is board dependent and difficult to implement on other boards. However, basic technics implemented in the ACME paper, which describes the development of that tool, could be used.

One of the basic understandings we need to have is the relationships between the frames of the ZCU106 board and its EBD file. As explained in the figure below reading the file from top to bottom is similar to moving the design following the error in a zigzag pattern.

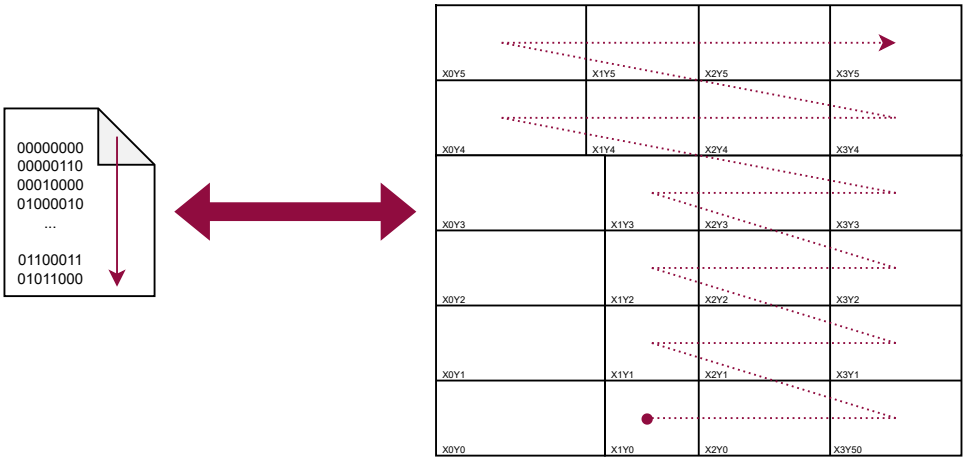


Figure 4.1: *Topological relationship of Clock Region of ZCU106 and its EBD file. Reading the file from top to bottom is similar to moving the design following the arrow on a zigzag basis.*

In this Xilinx forum question [30], it is described that the EBD file is an ASCII text file with a header followed by multiple lines of 32 characters, which represent the classification of 32-bit words in the CRAM. Each line of the EBD file represents one word, and the least significant bit of the word is on the far right. In the 16nm UltraScale+ family of FPGAs, each CRAM frame consists of 93 words, and the EBD file is organized into groups of 93 lines that hold the essential bit data for a configuration frame. The linear frame address (LA) is used to order the data, and the allowed range for LA is from zero to the maximum frame of the device. The word offset in a frame can range from 0 to 92, and the bit offset in a word can range from 0 to 31. The initial group of 118 lines in the EBD file consists of 25 dummy words and one dummy frame, followed by groups of 93 lines for each LA value, starting from LA = 0.

By implementing the above strategy, we can transform the lines of the EBD file into addresses that can be utilized by the SEM IP to inject errors. The subsequent crucial step is to comprehend the ranges of the clock regions. However, not all tiles existing in the clock region can be reconfigured, and thus, they are absent from the EBD file. In the MPSoC ZCU106 UltraScale+ architecture, a direct correlation cannot be performed. Nevertheless, an estimation of the position of a tile in the EBD file can be approximated by exploring an FFT kernel across the tiles. Only a signal FFT module is placed in the FPGA fabric, and the amount of congestion of the essential bits in the design is reported. The table 4.1 illustrates the relation between clock regions and their starting and ending lines in the EBD file. Although this approach is not entirely accurate, as the design may not precisely cover the entire clock region, we can extract an estimation for the position of our modules in the FPGA fabric.

Clock Region	Start	Finish
X1Y0	334.986	361.767
X2Y0	451.329	478.171
X3Y0	645.328	679.543
X1Y1	1.020.303	1.047.147
X2Y1	1.136.652	1.164.160
X3Y1	1.330.644	1.355.619
X1Y2	1.705.620	1.732.424
X2Y2	1.851.352	1.878.858
X3Y2	2.015.962	2.050.180
X1Y3	2.390.937	2.418.103
X2Y3	2.536.669	2.560.916
X3Y3	2.701.279	2.735.532

Clock Region	Start	Finish
X0Y4	2.826.558	2.847.818
X1Y4	3.076.257	3.101.216
X2Y4	3.221.991	3.246.225
X3Y4	3.386.598	3.420.888
X0Y5	3.570.553	3.631.386
X1Y5	3.761.574	3.788.386
X2Y5	3.877.914	3.905.436
X3Y5	4.071.915	4.096.831

Table 4.1: *Lines of EBD in relationship with the Clock Region of ZCU106*

By following this procedure, we can inject errors into our design and validate the functionality of our techniques using linear addresses of essential bits. To enable communication between the Processing System (PS) and the SEM IP, we utilize the monitor interface. The instruction parsing interface of the SEM IP establishes bidirectional communication between the PS and Programmable Logic (PL) and facilitates the injection of errors in the configuration memory within the PL. To inject an error in the kernel, the controller must be in an idle state. Once in the idle state, an error can be injected by executing the following command:

```
N {11-digit hex value}
```

The 11-digit hexadecimal value specifies the bit location where the error is to be injected. This linear frame address is obtained using the procedure described above. The format of the 11-digit hex value is depicted in the following figure. The successful injection of the error is indicated by the assertion and de-assertion of the injection status signal, as well as the state-changing report on the monitor interface, which indicates that the controller has transitioned from the injection state back to the idle state.

4.2 Methodologies Evaluation

All the methodologies discussed in Chapter 3 have been implemented, and we conducted two synthesis configurations for each of them: one utilizing DSPs and the other replacing the DSP functionality with LUTs. For every different methodology, the code parameterization enabled us to analyze the effects of different Fast Fourier Transform (FFT) sizes, specifically 8, 16, and 32. However, the metrics excluded the stage and butterfly triplication methods as the resources needed for the device made it unable to floorplan and place in the FPGA fabric.

4.2.1 Size

This thesis initiates with an exploration of the sizes of the methodologies mentioned above, and a comparison of their footprints with the error rates in our design. Previous chapters have explicated that the expansion of the different modules' areas in the FPGA

fabric correlates with the percentage of ionized particles that can potentially harm the device's functionality. This is a significant factor as an increase in logic within the components elevates the probability of single-event upsets. Therefore, it is crucial to comprehend and carefully consider what aspects we aim to protect from ionizing radiation to avoid undesired outcomes.

The subsequent figures depict the resource utilization of various methodologies and input sizes within the programmable logic.

Method	LUTs	DSPs	FFs	Es. Bits
FFT	2.77%	2.71%	1.24%	2.12%
Temp	3.45%	2.71%	1.52%	2.39%
DMR	3.44%	5.38%	1.40%	2.77%
DMR T	4.39%	5.38%	1.69%	3.12%
TMR	4.26%	8.04%	1.51%	3.46%
TMR S	4.75%	9.08%	1.74%	4.39%
TMR B	5.48%	9.08%	1.93%	4.34%

(1) *Computation using DSP*

LUTs	FFs	Es. Bits
4.04%	1.30%	2.74%
4.71%	1.58%	3.00%
5.97%	1.52%	3.94%
6.92%	1.80%	4.27%
8.06%	1.69%	5.19%
8.56%	1.84%	6.06%
9.30%	2.02%	6.37%

(2) *Computation using LUT*

Table 4.2: *Resource utilization of FPGA with 8-input FFT*

Method	LUTs	DSPs	FFs	Es. Bits
FFT	3.82%	9.89%	1.28%	3.14%
Temp	4.55%	9.89%	1.85%	3.75%
DMR	5.55%	19.73%	1.61%	4.91%
DMR T	6.96%	19.73%	2.17%	5.80%
TMR	7.58%	29.57%	1.82%	6.77%
TMR S	9.20%	20.89%	2.65%	8.04%

(1) *Computation using DSP*

LUTs	FFs	Es. Bits
7.65%	1.52%	4.75%
8.39%	2.08%	5.28%
13.21%	2.08%	7.88%
14.63%	2.64%	8.80%
19.07%	2.53%	11.36%
19.66%	2.91%	13.25%

(2) *Computation using LUT*

Table 4.3: *Resource utilization of FPGA with 16-input FFT*

Method	LUTs	DSPs	FFs	Es. Bits
FFT	6.09%	28.99%	1.44%	6.10%
Temp	6.87%	28.99%	2.55%	7.14%
DMR	10.35%	57.92%	2.14%	10.65%
DMR T	12.69%	57.92%	3.26%	12.21%
TMR	15.27%	86.86%	2.63%	15.80%

(1) *Computation using DSP*

LUTs	FFs	Es. Bits
19.37%	2.08%	11.05%
20.14%	3.19%	12.00%
36.95%	3.42%	20.83%
39.30%	4.54%	22.35%
55.44%	4.56%	30.35%

(2) *Computation using LUT*

Table 4.4: *Resource utilization of FPGA with 32-input FFT*

Tables 4.2, 4.3, and 4.4 present the resource utilization of FPGA resources with different input sizes and various mitigation techniques. As expected, the larger the input size of the FFT kernel, the more resources are required for its implementation. However, it is interesting to note the relationship between the size of the essential bits of the fine-grain mitigation techniques, specifically the stage and butterfly triplication methods. Contrary to our expectations, the number of essential bits in these triplication methods is increased, which can be attributed to the internal optimization performed by the tool that groups the functionality of some components together. When dividing them into PBlocks, these optimizations are unable to occur between them, thereby increasing the required resources.

The size metric is a crucial characteristic of the programmable logic implementation and has a direct correlation with the percentage of single-event upsets (SEUs) that might

occur. Therefore, it is significant to consider the area occupied by the features before multiplying them. Failure to do so may lead to unnoticeable results or even negative outcomes for the device.

The following figures illustrate the correlation the input size has on that resource. It is important to note as stated earlier, that the stage and butterfly triplication methods are not included in these figures, as their resource utilization was too high to allow for successful floorplan and placement in the FPGA fabric. Therefore, their impact on resource utilization could not be analyzed.

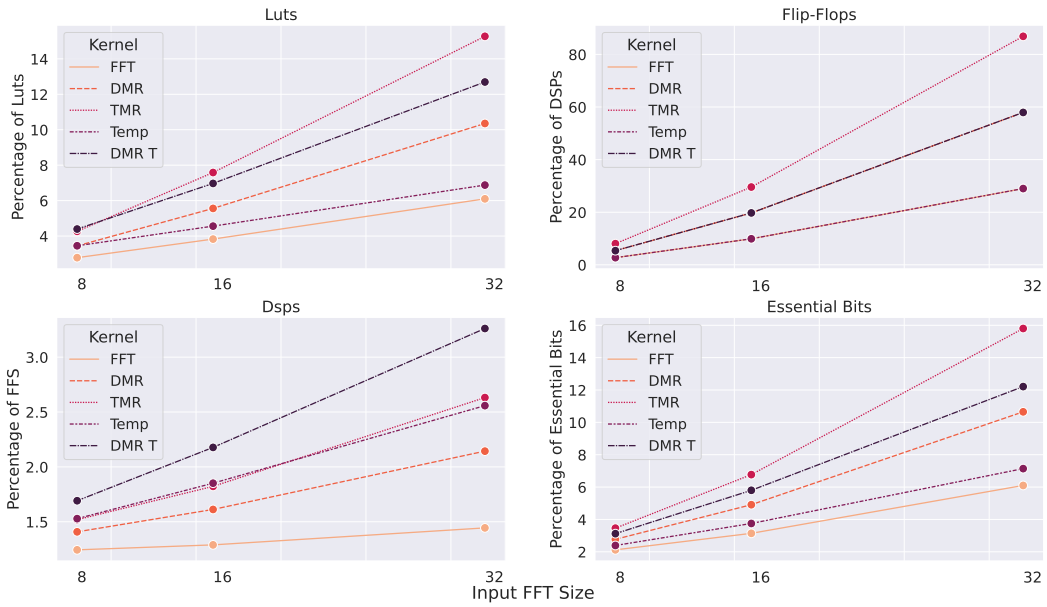


Figure 4.2: *Increase of usage based on input size using DSps*

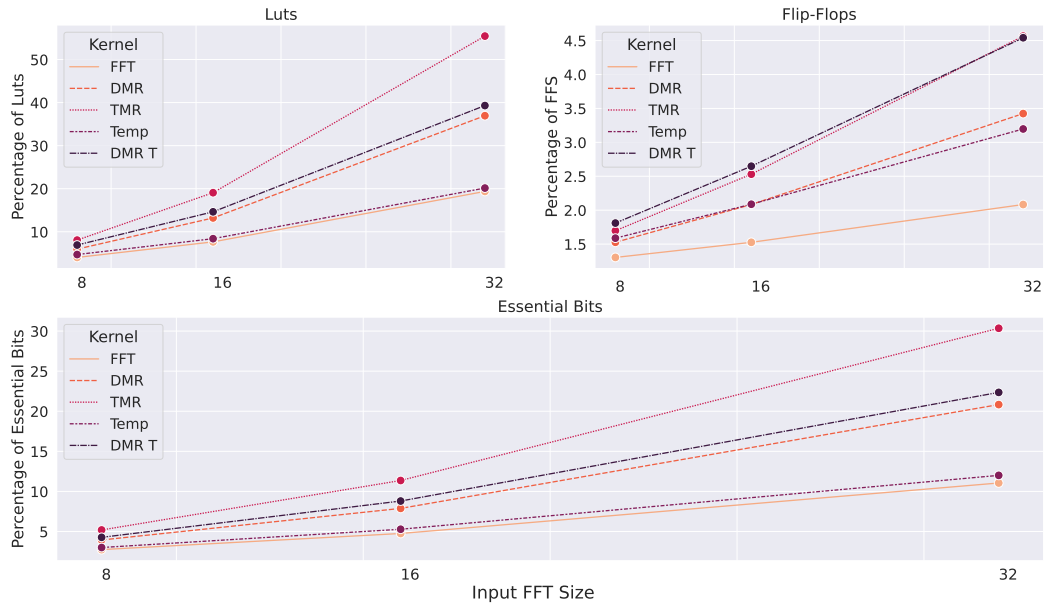


Figure 4.3: *Increase of usage based on input size using Luts*

The design must also account for variations in component size. Specifically, we align our differing sizes within the Fast Fourier Transform (FFT) framework without any mitigation techniques. Additionally, we incorporate temporal redundancy over a single FFT kernel, as well as Triple Modular Redundancy (TMR), by means of Lookup Tables (LUTs) rather than Digital Signal Processing (DSP) computations.

Kernel Size	Mitigation Technique	Module	LUTS	Flip-Flop
-	-	sem_ultra	288	403
-	-	sem_ultra_monitor_uart	61	54
8	No Fault Tolerance	FFT	4346	768
	Temporal	FSM	2	258
		Input Repeater	267	522
		Output Repeater	585	525
	TMR	TMR Voter	514	259
32	No Fault Tolerance	FFT	39956	5120
	Temporal	FSM	2	1026
		Input Repeater	1037	2060
		Output Repeater	1747	2061
	TMR	TMR Voter	2074	1027

Table 4.5: *Resource Utilization of Different Modules*

It is worth noting that these techniques require additional supportive modules to ensure proper functioning and testing, thereby introducing their own resource utilization. Although this may be of negligible significance when considering larger modules, it becomes crucial for small resource utilization, such as the 8-length input kernel. This aspect may or may not be a critical consideration when designing the final product.

4.3 Time of Reconfiguration of different regions

A crucial distinction in our study is the time required for partial reconfiguration of different size areas of the FPGA. As explained in Chapter 3, we chose to utilize the XilFPGA API provided by XILINX for partial reconfiguration. The XilFPGA library offers an interface for users to configure the programmable logic (PL) from the processing system (PS), specifically, Full bitstream loading and Partial bitstream loading, which are supported in the Zynq UltraScale+ MPSoC platform.

Table 4.5 depicts the various times required for different reconfiguration areas, ranging from the full FPGA bitstream to the smallest components on which we performed partial reconfiguration.

-	Full Reconfiguration	19.3 MB	28.3
Kernel Size	Partial Module	Size of Module (KB)	Time (msec)
8	FFT Kernel	661	1
	Average Stage Module	330	0.6
	Average Butterfly Module	174	0.363
16	FFT Kernel	2354	3.59
	Average Stage Module	338	0.9
32	FFT Kernel	7492	11.46

Table 4.6: *Relation between the size and the time of Partial Reconfiguration*

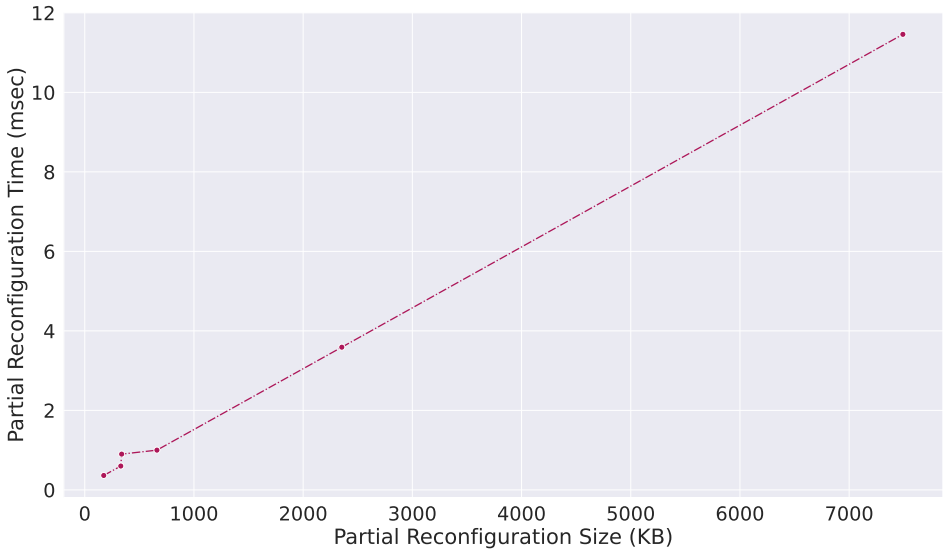


Figure 4.4: *Partial Size and Time*

Equally as important is the time needed for the reconfiguration implementing the internal scrubber. We can observe than the time need for the detection and correction of an error using the internal scrubber, is quite similar to the time needed for the reconfiguration of the entiry memory of the FPGA. The full reconfiguration is only 1.08× larger than the Xilinx approach. However, it should also be mentioned that most of the time is needed in the detection stage of the scrubbing, while the correction is relative small, in the 20-40 µs range

UltraScale+ XCZU7	Detection Time	26 ms
	Repair Correctable	44 µs
	Repair Uncorrectable	22 µs
	Any CRC-only (Uncorrectable)	9 µs

Table 4.7: *Latency of Internal Memory Scrubbing*

In our study utilizing XilFPGA for measurement purposes, we have extracted some notable findings. Firstly, we have determined that the time required for configuration is proportionate to the size of the area being reconfigured. For instance, in the comparison of the 32 and 16-length input FFT, the time required for reconfiguration is 3.19× longer, which is similar to the 3.18× increase in size. This pattern is also evident in the difference between the 8 and 16-length inputs, which require 3.56× and 3.59× longer reconfiguration times, respectively. However, it is important to note that the start-up time of the configuration significantly impacts the time required for reconfiguration, particularly for smaller sizes.

Regarding the various sizes of the 8-length input kernel, we observed that the time required for reconfiguration is 2.75× longer for the entire kernel size and 1.65× longer for the stage size than for the butterfly triplication modules. This implies that the reconfiguration time becomes more significant as the sizes become smaller, resulting in smaller

gains through fine-grain methods, which we will discuss in the next section.

Lastly, we encountered an issue with the size optimization of different implementations by the tool. During the Vivado synthesis and implementation algorithm, the design is floor planned and placed, with specific optimization performed for the utilization of some components. However, when the FFT kernel, which requires a substantial number of resources, is divided into smaller components, creating different reconfigurable PBLOCKs or regions, it leads to an increase in time. This is evident in the size of the various partial modules in the 8-length kernel size, where the stage was expected to be $1/3$ of the entire FFT kernel but was only $1/2$ on average, and the average butterfly was only 0.26 times smaller than the entire FFT kernel when optimized by the Vivado Design Suite.

4.4 Evaluation Campaign

In this thesis, an error injection campaign was developed to evaluate and categorize different methodologies and techniques, and to draw conclusions about their safety. The campaign assumed a continuous input of N-length complex numbers from the processing system (PS) to the programmable logic (PL). In spatial redundancy schemes, the inputs were continuous, whereas, in temporal redundancy schemes, the data was sent every N clock cycles. This distinction was crucial for the design to replicate the same output multiple times without losing incoming results. For each of the temporal redundancy methodologies, the results were replicated N times, which corresponded to the number of inputs of the Fast Fourier Transform (FFT) Kernel. This application was based on the notion that in the case of serial transmission of inputs to the FPGA fabric for calculation, the results could be grouped and sent in parallel with each other. This allowed the idle time that the kernel would wait for the transmission of the input data to be used for the proposed mitigation methodology to prevent possible errors. While this approach may lead to greater power and energy consumption, it was not within the scope of this thesis to measure its impact, as we only evaluated the resiliency and reliability of our design in single-event upsets. However, it is important to note that this may be critical to consider in real-time applications and deployed products.

To validate the outputs of our system, we generated a golden output from a known golden input and repeatedly submitted this input to our system. We used the Fast Fourier Transform (FFT) Kernel with the implementation of mitigation techniques to calculate the output. During each run, we monitored the resulting output and the flags raised by the system to gain insight into any potential errors in the computation. While this approach cannot be replicated in the context of real-world deployment, it is essential for evaluating the efficacy of our system's architecture and mitigation techniques. Our approach enables statistical evaluation of the system's performance, with the specific choice of evaluation criteria and techniques being left to the discretion of the system designer. The designer may consider submitting the golden input with the precalculated output on occasion to act as a watchdog for the system. However, this action should be undertaken sparingly to minimize the downtime of running applications.

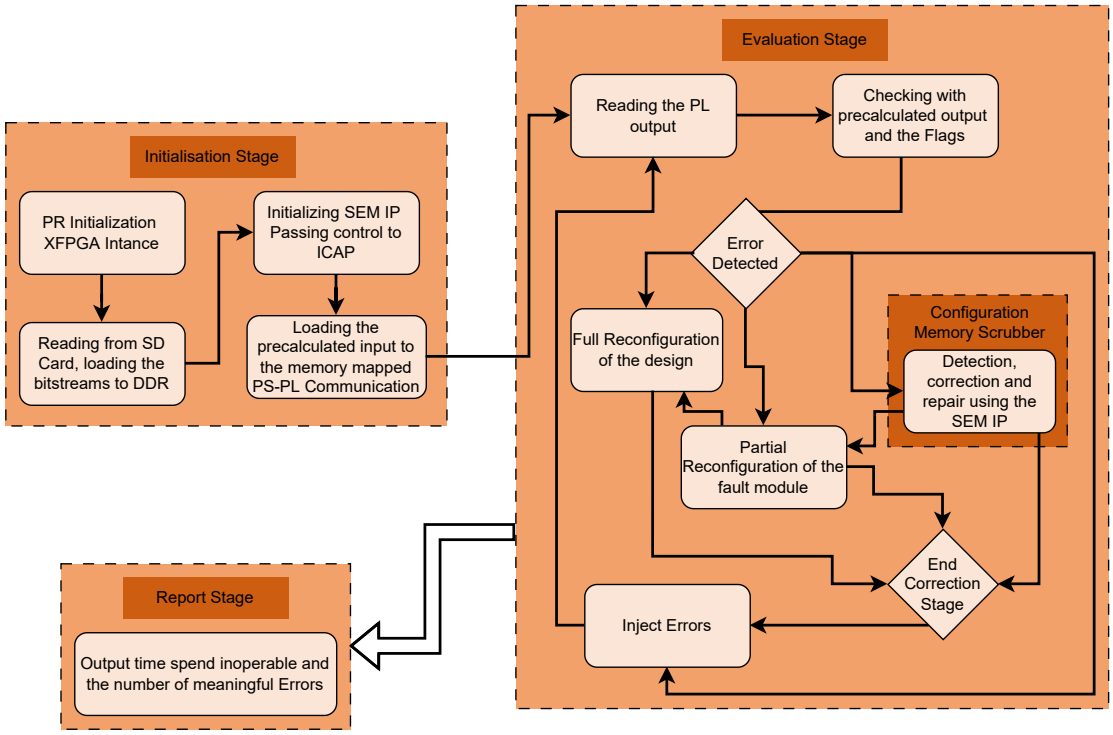


Figure 4.5: *Evaluation Champaign*

Each of the mitigation techniques employed in this study was approached in a distinct manner while maintaining a consistent underlying philosophy. In the event of an error, we undertook a thorough examination of the output and accompanying flags to locate the source of the error within the kernel. For cases where multiple errors were detected, we employed a strategy that involved reconfiguring each component of the partial designs to optimize performance. This approach was found to be more efficient than full reconfiguration, as the time required for multiple partial reconfigurations was less than that for a full reconfiguration. Following the reconfiguration process, we performed another check on the output to ensure the proper functioning of the system.

In situations where the flags were unable to reveal the error location, we initiated a full reconfiguration of the device. This approach enabled us to efficiently address errors that were not isolated to a specific component of the system.

To test the various methodologies, errors were injected into the addresses as per the injection campaign described in this implementation.

4.5 Evaluation

In this section, we evaluate the results of the different techniques, using the aforementioned two campaigns. These would give us an understanding of the effect those might have on reliability and the overall downtime of the device and application.

4.5.1 Mitigation Techniques Evaluation

The figures 4.5-4.7 depict the relation between the Number of Errors in the design and the downtime of each different technique, each mode in multiple input lengths of FFT Kernel.

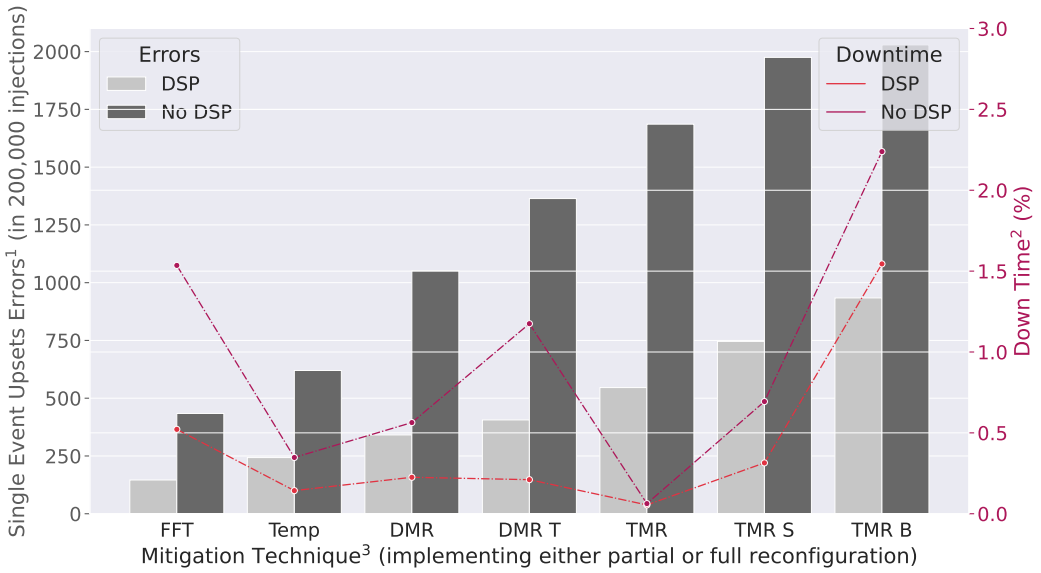


Figure 4.6: *Correlating SEU Errors and Downtime in a 8-Point FFT Kernel*

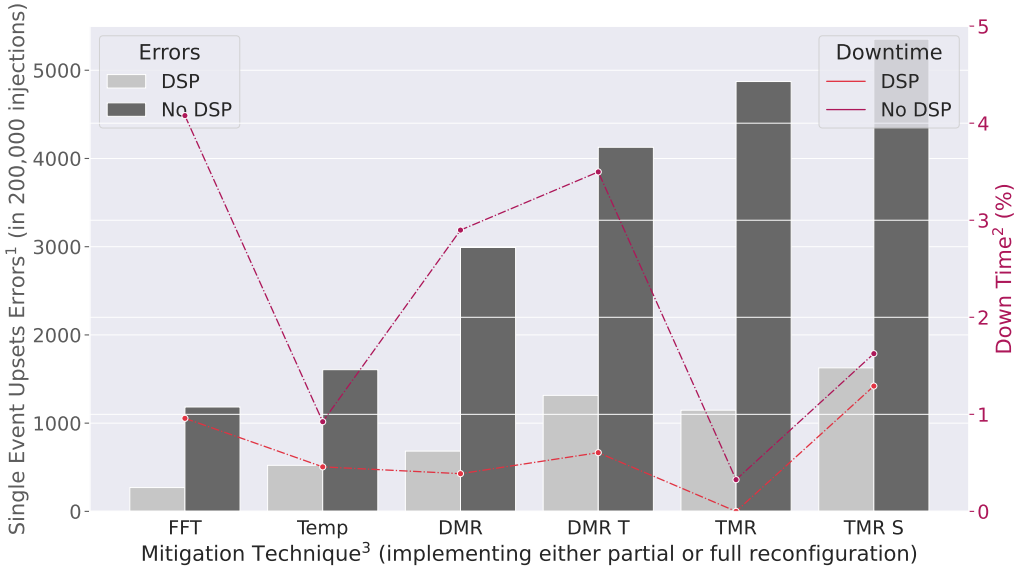


Figure 4.7: *Correlating SEU Errors and Downtime in a 16-Point FFT Kernel*

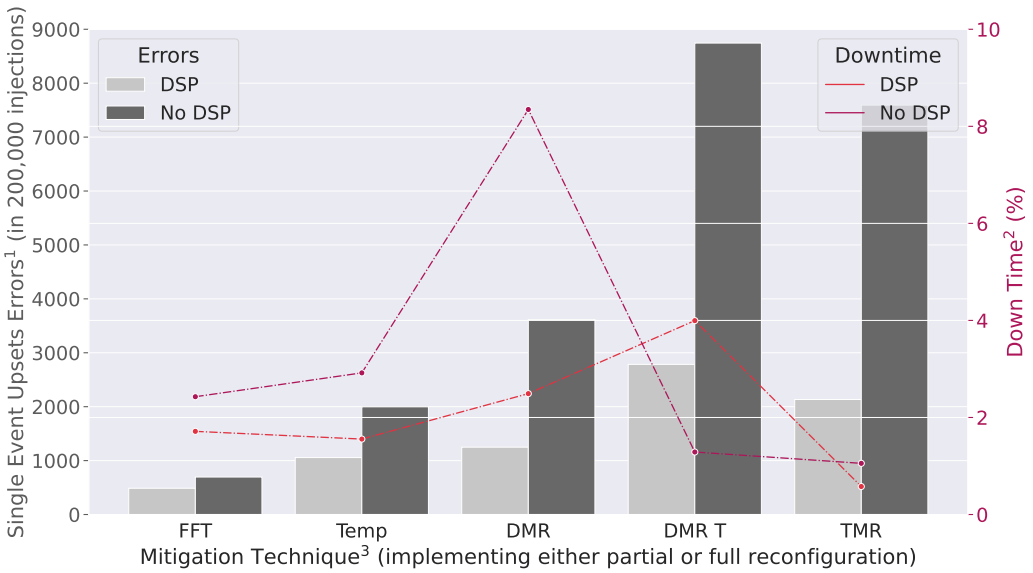


Figure 4.8: *Correlating SEU Errors and Downtime in a 32-Point FFT Kernel*

1. Single Event Effect Errors: The introduction of errors into a design does not always lead to the production of single Event Effect errors, and a change in device functionality. Specifically, it is necessary to determine the number of errors that cause a change in functionality, as these may result in incorrect outputs or trigger error flags. The error flags do not result in the wrong output, especially in mitigation techniques such as triple modular redundancy (TMR) or multiple operation time techniques like temporal redundancy methods such as the Temporal and DMR Temporal methods. These techniques can correct the output of the device.

2. Downtime: In all mitigation techniques, an increase in the number of injected errors can ultimately result in erroneous functionality, which can be detected through observation of the output or examination of error flags. Such flags indicate the presence of a faulty component, even if it is capable of correction. Downtime refers to the period during which the device cannot accept new data as it waits for reconfiguration to complete and restore correct functionality. During this time, inputs to the computational unit are lost as it is non-operational. It should be noted that downtime does not include the time during which a single component of the design is unavailable due to an error and undergoing partial reconfiguration. However, if other components are functioning normally and the application is able to compute without missing input data, this time is not added to the downtime.

3. Mitigation Technique: Each of the mitigation techniques and the unmitigated simple FFT kernel are susceptible to error and can ultimately result in a system error state. To restore the functionality of the system, partial or full reconfiguration is performed, depending on the specific situation. This ensures that all applications continue to function as necessary to complete the testing campaign. In our testing, we do not evaluate a method without employing the necessary partial or full reconfiguration.

The results obtained from the figures above indicate a significant relationship between the errors and the sizes calculated by the implementation of the Vivado Design Suite. It

is evident that the addition of extra logic for mitigation purposes increases the number of errors. Furthermore, the temporal redundancy also leads to an increase in the number of errors, as the additional logic required for the replication of the data input occupies a substantial portion of the design. This is particularly evident when considering the different sizes of the Kernel, and thus the replication time of the tools. For instance, in the input length 32 FFT Kernel, the amount of extra logic increases to a point where the number of errors in the DMR temporal technique exceeds that of the triplication modules.

Additionally, the use of different FPGA blocks for computation, such as DSPs or Luts, consistently results in higher errors and lower functional time of the design. Although both serve the same purpose, using Luts for computation consumes more space, as DSPs are more condensed.

The above diagrams indicate that the most effective techniques for minimizing the downtime of our design are the temporal redundancy and triplication module redundancy of the entire FFT kernel. While the replication of the calculation in the dual modular redundancy (DMR) technique allowed for partial reconfiguration of each distinct module, it did not manage to overcome the increase in size, and consequently, the increase in the number of errors. Although the results are compared using only spatial redundancy DMR, increasing the size occupied in the FPGA fabric leads to higher downtime.

It is critical to mention that the triplication of smaller components inside the FFT created more problems than it solved. The increased size and the higher number of voters lead to less functional time, as either more errors created problems or more hits in the voter parts of the design led to more time under full reconfiguration. As mentioned in the previous section, full reconfiguration is about 28$\times$ more time-consuming than a simple reconfiguration of the entire FFT kernel. Comparing the average stage and butterfly reconfiguration time of 1.6$\times$ and 2.75$\times$, respectively, it is evident that fine-grain triplication inside the FFT kernel and its components is not worth the extra functionality or the extra errors it introduces.

4.5.2 Internal Scrubbing Evaluation

We explained in the previous section, that the various mitigation techniques that impressed with their results were the temporal redundancy on a single FFT Kernel and the Triple Modular Redundancy again over the whole FFT. The first can be explained that we are able to understand errors happening during the computation and partially reconfigure the kernel, without needing to undergo the reconfiguration of the whole device. The triple modular redundancy has the ability to produce results even when a single one of the modules has been corrupted, as the other two would function properly. For that reason, the downtime is minimized and combat the effects of the increased size due to more modules.

In addition, we decided to consider the effects of the internal scrubber function, which we labeled Configuration Memory Scrubbing (CMS), build-in with SEM IP provided by Xilinx. Even though, we discussed that the internal scrubber is not always the proper choice [40], as it can also be affected by ionizing radiation. However, it is a good comparison, to understand how our results are fair incase of the operation of the internal scrubber

alongside our components.

We opted to integrate the internal scrubber in our metrics, by regarding only the ones using LUTs, and considering the extreme scenarios, like the 8 and 32 input kernel.

From the Chapter 3, it is clear that both the internal scrubber and the injection methodologies can not be used simultaniously from the SEM IP. For that reason, in this thesis implementation we have to choose one of the two approaches that are presented in the following table.

	Advantages	Disadvantages
After Every Injection	Higher Correction Rate	Higher power consumption, when sparce SEU
When Error is Detected	Operational only when needed	Lower correction Rate as MBU minimize the effect of ECC & CRC

Table 4.8: *Different approaches in Scrubbing*

We opted to implement the second route, opening the internall memory scrubber offered by the SEM IP, when an error is detected. This would lead to lower correction rates, as the accumulation of error in the memory frames would exceed the capabilities of the ECC algorith implemented inside the SEM IP, which has the ability to correct upto 4-bit errors in a single frame and upto 4 adjacent frames. This approach is closer to a real-time missions as the memory errors could be space in the time duration, maybe upto several in a month. So, the power consumption of the internal merory scrabber would have been greater for small improvements.

Kernel Size	Mitigation Technique	DownTime (sec)	DownTime with CMS (sec)	Improvements of CMS
8	Simple FFT	12.47	11.781	5.5%
	Temporal	1.93	1.85	4.14%
	TMR	0.57	0.54	5.06%
32	Simple FFT	19.90	18.94	4.8%
	Temporal	23.53	23.34	-0.8%
	TMR	8.32	7.67	7%

Table 4.9: *Mitigation Techniques with and without Configuration Memory Scrubbing*

The effectiveness of internal scrubbing in improving system performance is evident, although it is uncertain whether the benefits outweigh the drawbacks. Across various techniques, a notable improvement of between 5% and 10% was observed. The 32 temporal poses a distinct challenge, largely due to its algorithm's time dependency in our design. It is important to note that while the time of the error can potentially be identified during computation, any errors occurring between iterations would necessitate a complete reconfiguration of the design. This could result in varied timing of error injection during the evaluation campaign.

It is important to note that while the internal scrubber may not be the ideal choice, it offers a comparable representation to an external scrubber that implements the same ECC and CRC algorithms for error correction.

4.5.3 Comparisons- Summary

In this final section, we present a comprehensive overview of the best metrics in our design. As discussed before, the temporal and TMR techniques were found to be the best approaches for the mitigation of our design. Both provided excellent results, as the temporal approach could detect errors in the FFT kernel and partially reconfigure only the affected areas while keeping the occupied FPGA fabric minimal. However, the best overall approach was the triple modular redundancy, as it allowed the reconfiguration of one module of the design while the others kept running and calculating the Fourier transform algorithm. Even though triplicating the area needed for the implementation, the ability to exclude specific modules is critical to minimizing downtime and increasing the reliability of the device as a whole. We also included memory scrubbing in the best approach using different FPGA blocks, such as DSP and LUT, in the two kernel sizes 8 and 32. These gave a small but significant improvement in the overall reliability of the design.

The table 4.8 depicts the increase in reliability over the base case for each of the implementations. The increase is measured against the base case using no mitigation technique, ensuring that the measurement is as realistic and close to real-time applications as possible.

Kernel Size	Comp FPGA Block	Mitigation Technique	Downtime Reduction
8	DSP	Temporal	72.7%
		TMR	89.5%
	LUT	Temporal	84.5%
		TMR	95.4%
		Temporal & CMS	85.09%
		TMR & CMS	95.64%
32	DSP	TMR	66.7%
	LUT	TMR	58.1%
		TMR & CMS	61.4%

Table 4.10: *The effectiveness of the proposed fault-tolerant techniques in reducing downtime was compared against the baseline Fast Fourier Transform (FFT) Kernel. This was evaluated for each technique across various base cases, taking into consideration the number of points used for the FFT and the FPGA block utilized for computation.*

Chapter 5

Conclusion & Future Work

In Chapter 5, this thesis concludes with a comprehensive evaluation of the proposed mitigation techniques. Furthermore, a brief discourse on potential future avenues for research is provided, building on the techniques presented in this thesis.

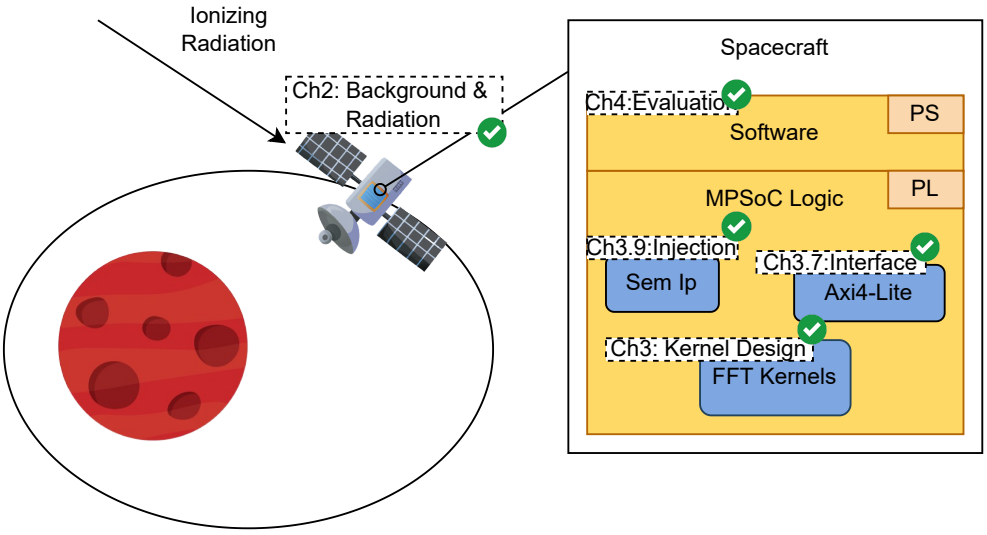


Figure 5.1: *Visual Chapter Guide*

Conclusion

This thesis presents experimental evaluation scenarios revealing that commercial-off-the-shelf FPGAs can be effectively utilized in space applications through the implementation of specific mitigation techniques to protect the configuration memory. Our proof-of-concept designs were implemented on an MPSoC Ultrascale+ ZU106 with quad-core Arm Cortex-A53 applications processor, while the kernel and redundancy mitigation techniques were implemented in 16nm FinFET+ programmable logic. All techniques presented in this thesis can be easily replicated on any board using these components with standard design tools and library IPs provided by Xilinx, which are widely available and relatively low cost compared to alternatives, making our methodology scalable with technology. This approach benefits from performance and energy efficiency improvements that can be achieved through improved technology nodes and can be scaled for designs with increasingly powerful processor cores while existing approaches are often custom-made and limited by restrictions of not being commercial products.

In this thesis, we developed a custom FFT kernel to serve as a building block for our mitigation approaches. The mitigation techniques implemented include spatial and temporal redundancy schemes in various forms, dynamic partial reconfiguration, and configuration memory scrubbing. The validation campaign implemented both injection and evaluation methodologies to test and categorize the proposed methods based on reliability and minimization of device downtime by comparing them against the unmitigated design. Specifically, we explored DMR and TMR spatial redundancy, temporal redundancy, a combination of the two using dual modules and repeating the operation, and fine-grain spatial redundancy on the smaller component of the FFT kernel. Additionally, every critical module of the application was able to be corrected by implementing dynamic partial reconfiguration. For each approach, different kernel sizes were implemented, such as 8, 16, and 32 input lengths of the FFT, and the different computational FPGA blocks like the DSPs and LUTs. Finally, we used the most resilient techniques, coupled with configuration memory scrubbing (CMS), to test their effects on the overall design. In each variation of SEU mitigation techniques, the same injection methodology was implemented to fairly compare results and measure improvement increases.

The combination of methods resulted in minimized device downtime, with improvements of up to 89.5% and 95.4% of the unprotected 8-input FFT kernel using Digital Signal Processors (DSPs) and Look-Up Tables (LUTs), respectively. Similar results were observed in the 32-input kernel, with 66.7% and 61.4% with the equivalent FPGA blocks.

Future Work

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

- One possible approach to mitigate single event upsets (SEUs) in on-chip BRAMs and off-chip DRAMs is to implement various techniques, such as error-correcting codes (ECC), cyclic redundancy check (CRC), triple modular redundancy (TMR), and memory scrubbing. For example, a TMR memory controller can be used to write data to three different locations and vote on them while reading to detect any corruption. If corruption is detected, the data can be rewritten with the majority-voted data.
- Introducing a radiation-tolerant SEM IP is a crucial step in ensuring the reliability of semiconductor devices operating in harsh environments. Some of the potential solutions are a watchdog mechanism that checks essential bits of the SEM IP every time it communicates with the configuration memory, to use of triplication of the SEM IP, where three identical instances of the IP are used and their outputs are compared paired with Partial reconfiguration of a faulty component.
- Another potential avenue is to use external memory scrubbing with other programmable cores in the system. One such example is having one core execute an application that uses the Fast Fourier Transform algorithm, while the second core acts as a watchdog responsible for periodically reading the device's configuration memory and correcting errors when detected. This method can also be further explored using ECC algorithms and other techniques.
- Regarding the evaluation process to measure the effectiveness of the proposed approach, extensive tests can be conducted by injecting errors across the entire configuration bitstream. Although time-consuming, this method can reveal the full spectrum of errors in the device. The next step in the verification process is to perform tests in a hazardous environment under ionizing radiation to assess the results in real-time scenarios. This can be done at a facility on the ground or an experimental satellite.

References

[1] T. Vladimirova and A. d. Curiel, “A system-on-a-chip for small satellite data processing and control (” chipsat”),” 2004.

[2] A. Farrington, A. Gray, B. Bell, V. Stanton, Y. Chong, K. Peters, C. Lee, and J. Srinivasan, “Software-reconfigurable processors for spacecraft,” 2005.

[3] D. Zheng, T. Vladimirova, H. Tiggeler, and M. Sweeting, “Reconfigurable single-chip on-board computer for a small satellite,” in *52nd International Astronautical Congress, Toulouse, France*, 2001.

[4] D. Zheng, T. Vladimirova, and M. Sweeting, “A ccsds-based communication system for a single chip on-board computer,” in *Proceedings of the 5th Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD’2002), D-5*, 2002.

[5] M. Caffrey, K. Morgan, D. Roussel-Dupre, S. Robinson, A. Nelson, A. Salazar, M. Wirthlin, W. Howes, and D. Richins, “On-orbit flight results from the reconfigurable cibola flight experiment satellite (cfesat),” in *17th IEEE Symposium on Field Programmable Custom Computing Machines*, pp. 3–10, 01 2009.

[6] Dan Friedlander, “Cots in space applications: Evolution overview.” Available online at: `https://www.doeeet.com/content/cots-components/cots-in-space-applications-evolution-overview/`, last accessed on 06.03.2023.

[7] 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 *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 127–134, 2018.

[8] 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. 59891–59905, 2020.

[9] A. Rodríguez, L. Santos, R. Sarmiento, and E. De La Torre, “Scalable hardware-based on-board processing for run-time adaptive lossless hyperspectral compression,” *IEEE Access*, vol. 7, pp. 10644–10652, 2019.

[10] 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. 131877–131892, 2021.

[11] X. Iturbe, D. Keymeulen, E. Ozer, P. Yiu, D. Berisford, K. Hand, and R. Carlson, “An integrated SoC for science data processing in next-generation space flight instruments avionics,” in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 134–141, 2015.

[12] V. Leon, I. Stamoulias, G. Lentaris, D. Soudris, R. Domingo, M. Verdugo, D. Gonzalez-Arjona, D. Merodio 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)*, pp. 1–8, 2021.

[13] L. Kosmidis, I. Rodriguez, Álvaro Jover, S. Alcaide, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, “GPU4S: Embedded GPUs in space - latest project updates,” *Elsevier Microprocessors and Microsystems*, vol. 77, pp. 1–10, 2020.

[14] C. Adams, A. Spain, J. Parker, M. Hevert, J. Roach, and D. Cotten, “Towards an integrated GPU accelerated SoC as a flight computer for small satellites,” in *IEEE Aerospace Conference*, pp. 1–7, 2019.

[15] 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 *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–4, 2022.

[16] J. Goodwill *et al.*, “NASA SpaceCube Edge TPU SmallSat Card for Autonomous Operations and Onboard Science-Data Analysis,” in *AIAA/USU Conf. on Small Satellites*, pp. 1–13, 2021.

[17] G. Giuffrida, L. Fanucci, G. Meoni, M. Batič, L. Buckley, A. Dunne, C. Van Dijk, M. Esposito, J. Hefele, N. Vercruyssen, G. Furano, M. Pastena, and J. Aschbacher, “The □-Sat-1 mission: the first on-board deep neural network demonstrator for satellite earth observation,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022.

[18] 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*, vol. 20, no. 3, pp. 1–23, 2021.

[19] 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.

[20] 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 *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 1–5, 2021.

[21] AMD Xilinx, San Jose, CA, USA, "Space-grade virtex-5qv fpga."

[22] Xiphos Systems Corporation, "Q8 processor." Available online at: https://xiphos.com/products/q8-processor/, last accessed on 06.03.2023.

[23] KP Labs, "Leopard-technical-sheet." Available online at: https://kplabs.space/wp-content/uploads/Leopard-technical-sheet.pdf, last accessed on 06.03.2023.

[24] AMD Xilinx, San Jose, CA, USA, "Zcu106 board user guide." Available online at: https://docs.xilinx.com/v/u/en-US/ug1244-zcu106-eval-bd, last accessed on 06.03.2023.

[25] D. Binder, E. C. Smith, and A. B. Holman, "Satellite anomalies from galactic cosmic rays," *IEEE Transactions on Nuclear Science*, vol. 22, no. 6, pp. 2675–2680, 1975.

[26] V. Leon, E. A. Papatheofanous, G. Lentaris, C. Bezaitis, N. Mastorakis, G. Bampilis, D. Reisis, and D. Soudris, "Combining fault tolerance techniques and cots soc accelerators for payload processing in space," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, 2022.

[27] AMD Xilinx, San Jose, CA, USA, "Ultrascale architecture configurable logic block user guide (ug574)." Available online at: https://docs.xilinx.com/v/u/en-US/ug574-ultrascale-clb, last accessed on 06.03.2023.

[28] AMD Xilinx, San Jose, CA, USA, "Ultrascale architecture soft error mitigation controller v3.1 logicore ip product guide." Available online at: https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/sem_ultra/v3_1/pg187-ultrascale-sem.pdf, last accessed on 06.03.2023.

[29] AMD Xilinx, San Jose, CA, USA, "Article: 41197- soft error mitigation controller - what is the difference between the ebc and the ebd file?." Available online at: https://support.xilinx.com/s/article/41197?language=en_US, last accessed on 06.03.2023.

[30] AMD Xilinx, San Jose, CA, USA, "Article:70684 - ultrascale+ - sem ip - how to use the sem ip error report to look up essential bit error locations using the ebd (essential bit data) file?." Available online at: https://support.xilinx.com/s/article/70684?language=en_US, last accessed on 06.03.2023.

[31] J. S. George, “An overview of radiation effects in electronics,” *AIP Conference Proceedings*, vol. 2160, no. 1, pp. 1–8, 2019.

[32] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M. S. Reorda, M. Violante, and P. Zambolin, “Evaluating the effects of seus affecting the configuration memory of an sram-based fpga,” *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, pp. 584–589 Vol.1, 2004.

[33] Wikipedia, “Fast fourier transform.” Available online at: `https://en.wikipedia.org/wiki/Fast_Fourier_transform`, last accessed on 06.03.2023.

[34] J. M. Johnson and M. J. Wirthlin, “Voter insertion algorithms for fpga designs using triple modular redundancy,” in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA ’10, (New York, NY, USA), p. 249–258, Association for Computing Machinery, 2010.

[35] S. Mitra and E. McCluskey, “Word-voter: a new voter design for triple modular redundant systems,” in *Proceedings 18th IEEE VLSI Test Symposium*, pp. 465–470, 2000.

[36] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, “A comparison of tmr with alternative fault-tolerant design techniques for fpgas,” *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065–2072, 2007.

[37] M. Carmichael, C. Caffrey and A. Salazar, “Correcting single-event upsets through virtex partial configuration.,” in *Xilinx Application Notes, XAPP216*, 2000.

[38] AMD Xilinx, San Jose, CA, USA, “Xilfpga: Baremetal drivers and libraries.” Available online at: `https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841910/Xilfpga`, last accessed on 06.03.2023.

[39] Real Digital, “Axi4-lite interface.” Available online at: `https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081`, last accessed on 06.03.2023.

[40] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. LaBel, M. Friendlich, H. Kim, and A. Phan, “Effectiveness of internal vs. external seu scrubbing mitigation strategies in a xilinx fpga: Design, test, and analysis,” in *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, pp. 1–8, 2007.

[41] 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. 1–5, 2019.

[42] AMD Xilinx, San Jose, CA, USA, “Soft error mitigation using prioritized essential bits.” Available online at: `https://www.eeweb.com/wp-content/uploads/articles-app-notes-files-soft-error-mitigation-using-prioritized-essential-bits-1339781673.pdf`, last accessed on 06.03.2023.