Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων VLSI

Combining Mitigation Techniques for Fault-Tolerant Data Transfers in SOC FPGAs

Diploma Thesis

by

Konstantinos Larisis

Επιβλέπων: ΔΗΜΗΤΡΙΟΣ ΣΟΥΝΤΡΗΣ
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2025

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

Combining Mitigation Techniques for Fault-Tolerant Data Transfers in SOC FPGAs

Diploma Thesis

by

Konstantinos Larisis

Επιβλέπων: ΔΗΜΗΤΡΙΟΣ ΣΟΥΝΤΡΗΣ
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Ιουλίου, 2025.

.....	.....	.....
ΔΗΜΗΤΡΙΟΣ ΣΟΥΝΤΡΗΣ	ΣΩΤΗΡΙΟΣ ΞΥΔΗΣ	ΓΕΩΡΓΙΟΣ ΛΕΝΤΑΡΗΣ
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής ΠΑΔΑ

Αθήνα, Ιούλιος 2025

.....

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

Περίληψη

Η διαστημική βιομηχανία έχει βιώσει έναν μετασχηματισμό τα τελευταία χρόνια, μεταβαίνοντας από παραδοσιακές radiation-hardened συσκευές σε Commercial Off-The-Shelf (COTS) SoC FPGAs για τις διαστημικές αποστολές της. Αυτή η μετάβαση οφείλεται κυρίως σε πλεονεκτήματα, όπως το μικρό κόστος, ο μικρός χρόνος εισαγωγής στην αγορά, η μεγάλη υπολογιστική δύναμη και η αυξημένη ευελιξία στην ανάπτυξη υλικολογισμικού. Ωστόσο, τα COTS SoC FPGAs δεν έχουν κατασκευαστεί με προδιαγραφές διαστήματος, με αποτέλεσμα να είναι ευάλωτα στην ιονίζουσα ακτινοβολία. Αν και η προσοχή έχει στραφεί κυρίως στα στοιχεία επεξεργασίας, όπως οι ARM πυρήνες και οι επιταχυντές υλικού (hardware kernels), τα uncore στοιχεία, που περιλαμβάνουν και όλους τους διαύλους και διεπαφές επικοινωνίας μέσα σε ένα SoC FPGA, είναι επίσης ιδιαίτερα ευάλωτα σε σφάλματα λόγω ακτινοβολίας. Συνεπώς, είναι απαραίτητο να διασφαλιστεί η αξιόπιστη μεταφορά δεδομένων μεταξύ επεξεργαστή (CPU) και επιταχυντή υλικού.

Η παρούσα διπλωματική προτείνει μια ολοκληρωμένη, ανθεκτική σε σφάλματα αρχιτεκτονική, ειδικά σχεδιασμένη για SoC FPGAs, με σκοπό τον περιορισμό των ανατροπών μεμονωμένων συμβάντων (SEUs) που επηρεάζουν τα δεδομένα ωφέλιμου φορτίου στις μεταφορές δεδομένων, χωρίς να αλλοιώνουν τη λειτουργικότητα του συστήματος. Η προτεινόμενη αρχιτεκτονική υλοποιείται στην πλατφόρμα MPSoC UltraScale+, ενσωματώνοντας μηχανισμούς ανθεκτικότητας σε σφάλματα, τόσο στο PL όσο και στο PS υποσύστημα. Συνδυάζει τις τεχνικές Triple Modular Redundancy (TMR) και temporal redundancy, με ενσωματωμένες δυνατότητες ανθεκτικότητας σε σφάλματα του υποσυστήματος PS, ειδικότερα το dual-core lockstep του επεξεργαστή ARM Cortex R5. Για την αξιολόγηση της αξιοπιστίας και αποτελεσματικότητας της αρχιτεκτονικής, πραγματοποιήθηκε μια προσομοίωση εισαγωγής σφαλμάτων στο σύστημα, στηριζόμενη στις τεχνικές fault pruning και saboteur insertion. Τα πειραματικά αποτελέσματα υποδεικνύουν σημαντική βελτίωση στην αξιοπιστία του συστήματος, χωρίς να επιβάλλουν υπερβολική υπολογιστική επιβάρυνση (computational overhead) ή αύξηση της χρησιμοποιούμενης περιοχης (area overhead).

Λέξεις-κλειδιά — COTS SoC FPGAs, Fault Tolerance, Spatial Redundancy, Temporal Redundancy, Dual-Core Lockstep, ARM Cortex R5, SEU Error Injection, Reliability

Abstract

The space industry has experienced a transformation in recent years, transitioning from traditional radiation-hardened devices to Commercial Off-The-Shelf (COTS) SoC FPGAs in space missions. This shift is primarily driven by benefits including reduced costs, shorter time-to-market, increased processing performance, and enhanced development flexibility. However, since COTS SoC FPGAs are not specifically designed for the harsh conditions of space, they remain vulnerable to ionizing radiation. While much attention has been given to the vulnerability of processing elements such as Arm cores and hardware kernels, uncore components, including communication interfaces, are also highly susceptible to radiation-induced faults. Consequently, ensuring data integrity during transfers between the CPU and hardware kernel is challenging.

This thesis proposes a comprehensive fault-tolerant architecture specifically tailored for SoC FPGAs, aimed at mitigating Single Event Upsets (SEUs) affecting payload data during data transfers, without altering the system's functionality or components. The proposed architecture is implemented on the MPSoC UltraScale+ platform, incorporating fault-tolerant mechanisms in both PL and PS subsystems. It integrates mitigation techniques such as Triple Modular Redundancy (TMR) and temporal redundancy within the FPGA fabric, alongside built-in PS fault-tolerant features, notably the dual-core lockstep capability of the ARM Cortex R5 processor. A targeted fault injection campaign, involving fault pruning and saboteur insertion, was conducted to rigorously evaluate the architecture's resilience and effectiveness. Experimental results demonstrate significant improvements in system reliability without imposing excessive computational or area overhead.

Keywords — COTS SoC FPGAs, Fault Tolerance, Spatial Redundancy, Temporal Redundancy, Dual-Core Lockstep, ARM Cortex R5, SEU Error Injection, Reliability

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κύριο Δημήτριο Σούντρη, για την εμπιστοσύνη που μου έδειξε να εκπονήσω τη διπλωματική μου εργασία στο εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων. Επίσης, ευχαριστώ θερμά τον υποψήφιο διδάκτορα Ηλία Παπαλάμπρου, τον διδάκτορα Ιωάννη Στρατάκο, καθώς και τον καθηγητή κύριο Γεώργιο Λεντάρη, για την πολύτιμη βοήθεια και καθοδήγηση τους. Τέλος, θα ήθελα να ευχαριστήσω μέσα από την καρδιά μου τους φίλους μου, για τις όμορφες στιγμές που μοιραστήκαμε κατά τη διάρκεια των φοιτητικών χρόνων, αλλά και τους γονείς μου, Ελένη Μπουγατζέλη και Σοφοκλή Λαρίση, που με στηρίζουν σε κάθε μου βήμα και χωρίς αυτούς δεν θα είχα καταφέρει να φτάσω ως εδώ.

Κωνσταντίνος Λαρίσης, Ιούλιος 2025

Contents

List of Figures

Chapter 0

Εκτεταμένη Περίληψη στα Ελληνικά

0.1 Εισαγωγή

Η εποχή του NewSpace ορίζεται ως η περίοδος από τις αρχές του 21ου αιώνα έως σήμερα και χαρακτηρίζεται από την εμπορευματοποίηση των διαστημικών δραστηριοτήτων. Ολοένα και περισσότερες ιδιωτικές εταιρείες εισέρχονται στον διαστημικό τομέα, υιοθετώντας νέα επιχειρηματικά μοντέλα που στοχεύουν στη μείωση του κόστους και του χρόνου διάθεσης στην αγορά. Ως αποτέλεσμα των νέων αναγκών, η επιλογή Ηλεκτρικών, Ηλεκτρονικών και Ηλεκτρομηχανικών (EEE) εξαρτημάτων τείνει προς τα εμπορικά διαθέσιμα εξαρτήματα (COTS), αντί των ειδικά κατασκευασμένων και δοκιμασμένων για διαστημική χρήση. Παρόλο που τα εξαρτήματα COTS δεν προσφέρουν την ίδια αξιοπιστία σε συνθήκες ακτινοβολίας, είναι πολύ πιο οικονομικά και εύκολα προσβάσιμα. Παράλληλα, οι απαιτήσεις πραγματικού χρόνου και η χαμηλή κατανάλωση ισχύος έχουν οδηγήσει στη μετάβαση από γενικής χρήσης επεξεργαστές σε πιο εξειδικευμένες υλοποιήσεις με επιταχυντές υλικού, και ιδιαίτερα σε Field-Programmable Gate Arrays (FPGAs).

Στις SoC FPGA αρχιτεκτονικές, εμφανίζεται συχνά η έννοια του computation offloading. Η διαδικασία αυτή περιλαμβάνει την ανάθεση απαιτητικών υπολογιστικών εργασιών σε επιταχυντές υλικού, που υλοποιούνται στο προγραμματιζόμενο λογικό μέρος (PL) του SoC. Για τη λειτουργία αυτή, απαιτείται η μεταφορά δεδομένων από τη μνήμη DDR του Processing Subsystem (PS) του SoC προς τον επιταχυντή και αντίστροφα. Το μονοπάτι αυτό ορίζεται ως offloading datapath. Ωστόσο, τα COTS SoC FPGAs είναι πιο ευάλωτα στην ακτινοβολία, επειδή δεν σχεδιάζονται με γνώμονα τις ειδικές διαστημικές συνθήκες, με αποτέλεσμα η ακεραιότητα των δεδομένων κατά τη μεταφορά αυτή να μην είναι εγγυημένη.

Σφάλματα λόγω ακτινοβολίας

Μια μεγάλη κατηγορία σφαλμάτων που οφείλονται στην ιονίζουσα ακτινοβολία του διαστήματος είναι τα Single Event Effects (SEEs). Πρόκειται για σφάλματα που συμβαίνουν ξαφνικά σε ένα σύστημα και χωρίζονται σε hard errors και soft errors. Στη πρώτη περίπτωση, τα σφάλματα δημιουργούν μόνιμη βλάβη στο λογικό κύκλωμα, ενώ στη δεύτερη περίπτωση δημιουργούν προσωρινή βλάβη. Τα soft errors χωρίζονται σε Single Event Effects (SETs), τα οποία επηρεάζουν τους κόμβους του λογικού κυκλώματος και σε Single Event Upsets (SEUs), τα οποία

αφορούν αλλαγές στη λογική τιμή των αποθηκευτικών στοιχείων ενός λογικού κυκλώματος, δηλαδή τα flip-flops. Γενικότερα, η διπλωματική εστιάζει στα SEUs, τα οποία αφορούν flip flops που περιέχουν ωφέλιμο φορτίο, δηλαδή bits δεδομένων και όχι bits ελέγχου. Έτσι, η πληροφορία μπορεί να αλλοιωθεί, αλλά η λειτουργικότητα του συστήματος παραμένει ανέπαφη.

Τεχνικές ανοχής σε σφάλματα

Οι τεχνικές ανοχής σε σφάλματα χωρίζονται σε δύο κατηγορίες: τις τεχνικές error detection και τις τεχνικές recovery. Στη πρώτη περίπτωση, οι τεχνικές αποσκοπούν στην ανίχνευση των σφαλμάτων, ενώ στη δεύτερη περίπτωση διορθώνουν τα σφάλματα και βοηθούν το σύστημα να ανακάμψει έπειτα από αυτά. Σε αρκετές υπολογιστικές πλατφόρμες, υπάρχουν built-in δυνατότητες για ανοχή σε σφάλματα, όπως συμβαίνει στη περίπτωση των SoC FPGAs.

SoC FPGAs

Τα SoC FPGAs αποτελούν ετερογενή devices, καθώς συνδυάζουν τον επαναπρογραμματισμό του υλικού του FPGA, με την υπολογιστική δύναμη των ARM επεξεργαστών. Μάλιστα, σε αρκετές αρχιτεκτονικές, όπως στην Ultrascale+, υπάρχει διαθέσιμη μια κεντρική μονάδα επεξεργασίας πραγματικού χρόνου (RPU), εκτός από τη κλασσικη μονάδα επεξεργασίας γενικού σκοπού (APU). Η μονάδα αυτή, στην αρχιτεκτονική Ultrascale+, είναι βασισμένη σε μία δυάδα πυρήνων ARM Cortex R5, οι οποίοι παρουσιάζουν ενδιαφέρον ως προς τις δυνανότητες τους για ανοχή στα σφάλματα.

ARM Cortex R5

Μία δυάδα πυρήνων R5 μπορεί να λειτουργήσει σε δύο διαφορετικές διατάξεις λειτουργίας:

- **Split Configuration**: Οι δύο πυρήνες λειτουργούν ανεξάρτητα ο ένας από τον άλλον, διαθέτοντας ξεχωριστές caches και ξεχωριστές διεπαφές προς το υπόλοιπο σύστημα.
- **Lockstep Configuration**: Ο ένας πυρήνας από τους δύο αποτελεί ένα αντίγραφο της λογικής του πρώτου, εκτελώντας τις ίδιες εντολές με μία χρονική καθυστέρηση μερικών κύκλων. Μόνο ο λειτουργικός πυρήνας έχει πρόσβαση στη cache και στις διεπαφές με το υπόλοιπο σύστημα, ενώ κανείς μπορεί να εισάγει πρόσθετη λογική, η οποία να συγκρίνει τις εξόδους των δύο πυρήνων κάθε χρονική στιγμή. Με αυτό το τρόπο, είναι δυνατή η ανίχνευση σφαλμάτων που συμβαίνουν εσωτερικά σε έναν από τους δύο πυρήνες και επηρεάζουν την έξοδο του.

Συμβολές Διπλωματικής

Οι κύριες συνεισφορές της παρούσας διπλωματικής εργασίας είναι οι εξής:

- Προτείνεται μία αρχιτεκτονική ανθεκτική σε σφάλματα για τη μεταφορά δεδομένων μεταξύ της κύριας μνήμης και του επιταχυντή υλικού, στο πλαίσιο του computation offloading. Η προτεινόμενη αρχιτεκτονική αξιοποιεί τα τυποποιημένα IP cores της Xilinx, διασφαλίζοντας τη συμβατότητα με τα υπάρχοντα εργαλεία λογισμικού της εταιρείας. Η αρχιτεκτονικη συνδυάζει διαφορετικές τεχνικές ανοχής σε σφάλματα, καλύπτοντας τόσο το PL όσο και το PS υποσύστημα.

- Ανάπτυσσεται ένας μηχανισμός εισαγωγής σφαλμάτων στο σύστημα (fault injection campaign), με σκοπό την αξιολόγηση της προτεινόμενης αρχιτεκτονικής στη Zynq UltraScale+ MPSoC πλατφόρμα. Ο μηχανισμός αυτός επιτρέπει την ελεγχόμενη εισαγωγή σφαλμάτων, πραγματοποιώντας μια προσομοίωση περιβάλλοντος ακτινοβολίας.
- Αναλύεται η αξιοπιστία και η ανθεκτικότητα σε σφάλματα των εφαρμοζόμενων τεχνικών, παρέχοντας πολύτιμες πληροφορίες σχετικά με την αποτελεσματικότητα και καταλληλότητά τους για εφαρμογές υψηλής κρισιμότητας, όπου απαιτείται υψηλό επίπεδο λειτουργικής ασφάλειας.

0.2 Προτεινόμενη Αρχιτεκτονική

Η κλασσική αρχιτεκτονική ενός SoC FPGA για την μεταφορά της πληροφορίας από την DDR στον επιταχυντή υλικού, απεικονίζεται στο Σχήμα 0.2.1 και διαχωρίζεται σε πέντε κύριες ενότητες για λόγους σαφήνειας:

1. Application Processing Unit (APU): εκτελεί την κύρια εφαρμογή.
2. APU to PS Subsystem Boundary: περιλαμβάνει το υποσύστημα της μνήμης, λογικούς διακόπτες και τη διεπαφή PS-PL.
3. FPGA IP Cores: περιλαμβάνει τα IP cores που θεμελιώνουν την επικοινωνία μεταξύ της διεπαφής PS-PL και του επιταχυντή υλικού.
4. Επιταχυντής Υλικού: εκτελεί τις πράξεις επεξεργασίας δεδομένων.

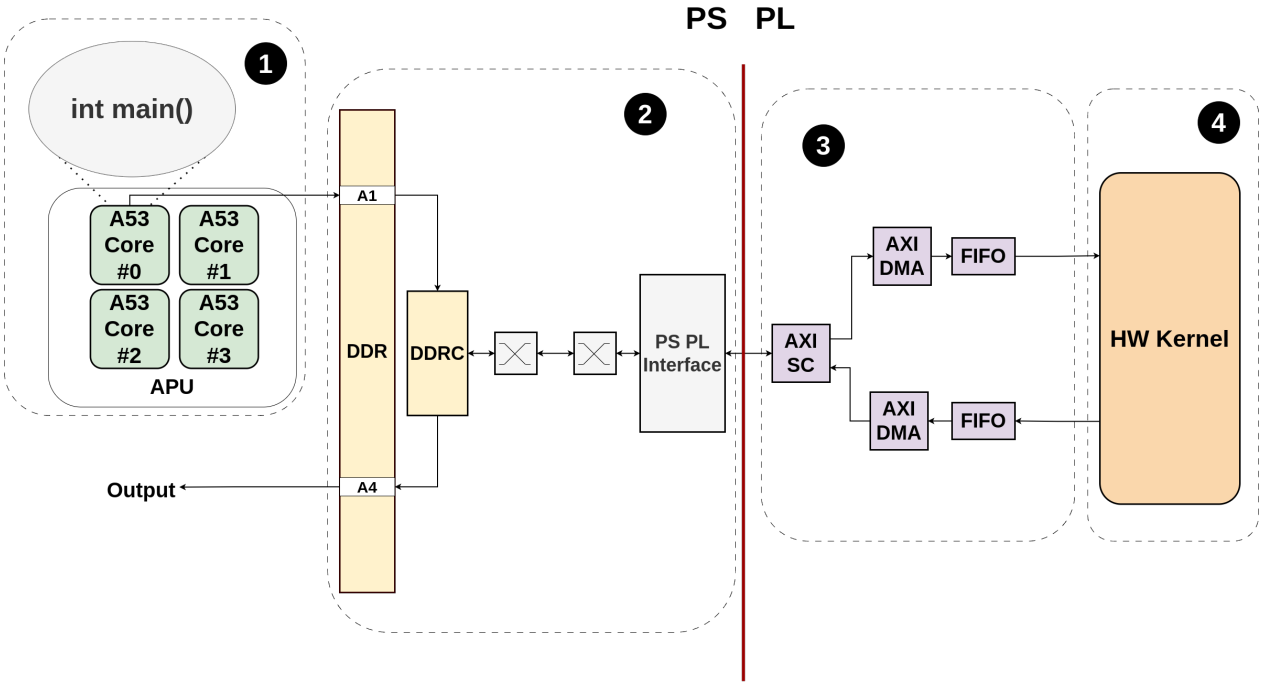


Figure 0.2.1: Αρχιτεκτονική Αναφοράς

Η προτεινόμενη αρχιτεκτονική βασίζεται στη παραπάνω κλασσική αρχιτεκτονική και την χρησιμοποιεί ως αρχιτεκτονική αναφοράς (baseline). Συγκεκριμένα, αποτελεί έναν συνδυασμό τριών

διαφορετικων τεχνικών ανοχής σε σφάλματα, εστιάζοντας στις ενότητες δύο και τρία του σχήματος, οι οποίες περιέχουν όλες τις διεπαφές επικοινωνίας σε ένα SoC FPGA.

Τεχνική 1: Triple Modular Redundancy (TMR)

Για την εφαρμογή της τεχνικής αυτής, είναι αναγκαίος ο τριπλασιασμός όλων των κόμβων που απαρτίζουν το μονοπάτι της πληροφορίας μεταξύ μνήμης και επιταχυντη υλικού. Αρχικά, δημιουργούνται τρία αντίγραφα της πληροφορίας στη μνήμη και προωθούνται μέσα από τρία διαφορετικά μονοπάτια προς τη διεπαφή PS-PL. Στη συνέχεια, τριπλασιάζεται η δομή των IP cores που θεμελιώνει την επικοινωνία στη μεριά του FPGA και προστίθεται ένα στοιχείο λογικής, το οποίο υλοποιεί μία συνάρτηση πλειοψηφίας, δηλαδή αποφασίζει κάθε χρονική στιγμή το μονοπάτι που θα προωθήσει στον επιταχυντή υλικού. Αφού η πληροφορία επεξεργαστεί, τριπλασιάζεται, κάθε αντίγραφο διέρχεται μέσα από ένα προυπάρχον μονοπάτι και αποθηκεύεται στη μνήμη. Την υλοποίηση της συνάρτησης πλειοψηφίας αναλαμβάνει ένας πυρήνας R5, προκείμενου να προωθήσει το τελικό αποτέλεσμα στη κύρια εφαρμογή. Η αναπαράσταση του TMR σχήματος απεικονίζεται στο Σχήμα 0.2.2.

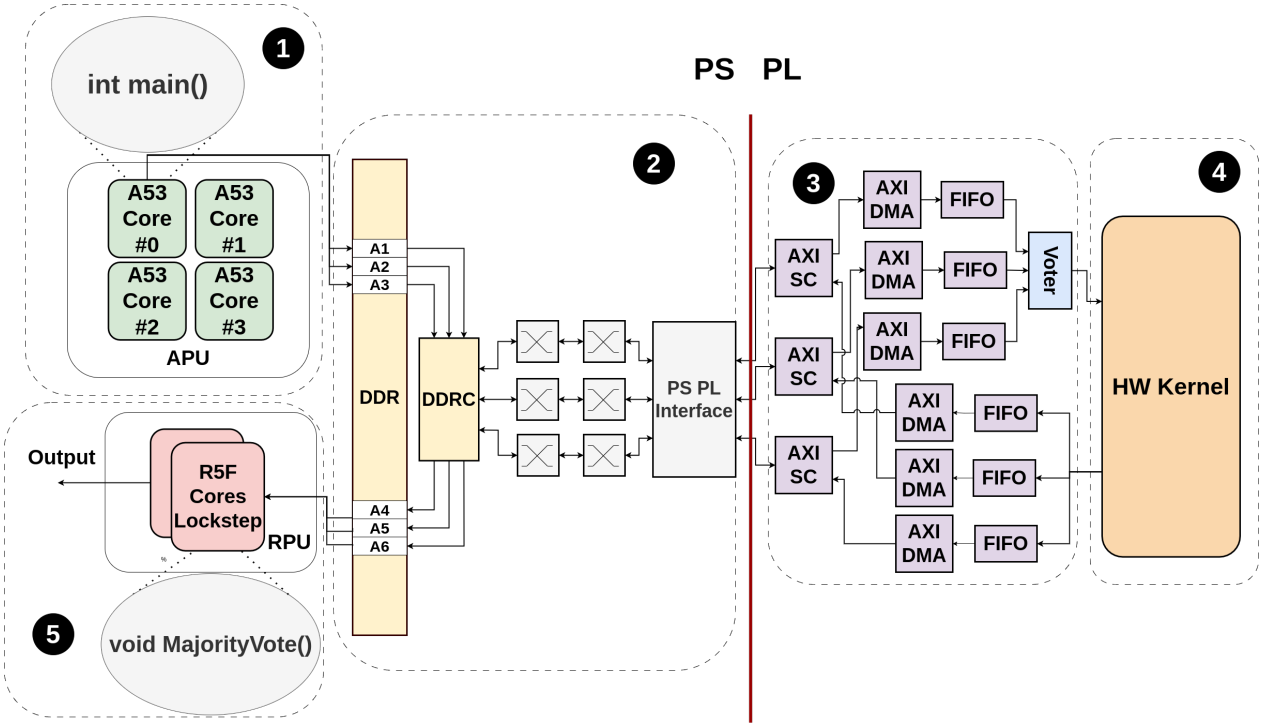


Figure 0.2.2: Προτεινόμενη Αρχιτεκτονική

Η υλοποίηση της συνάρτησης πλειοφηψίας (voting) στο PL γίνεται με ένα συνδυαστικό κύκλωμα, που αποτελείται από 32 αντίγραφα ενός 3-bit voter, όπως φαίνεται στο Σχήμα 0.2.3. Πρόκειται για ένα κύκλωμα με απλή υλοποίηση, που δεν εισάγει πρόσθετη καθυστέρηση εκτέλεσης. Αντίθετα, το voting στο PS πραγματοποιείται από μία σειρά από εντολές του R5, με αποτέλεσμα, σε αυτή τη περίπτωση να προστίθεται καθυστέρηση εκτέλεσης στο σύστημα. Μάλιστα, αυτή η καθυστέρηση εξαρτάται από τον όγκο της πληροφορίας και τη συχνότητα λειτουργίας του R5.

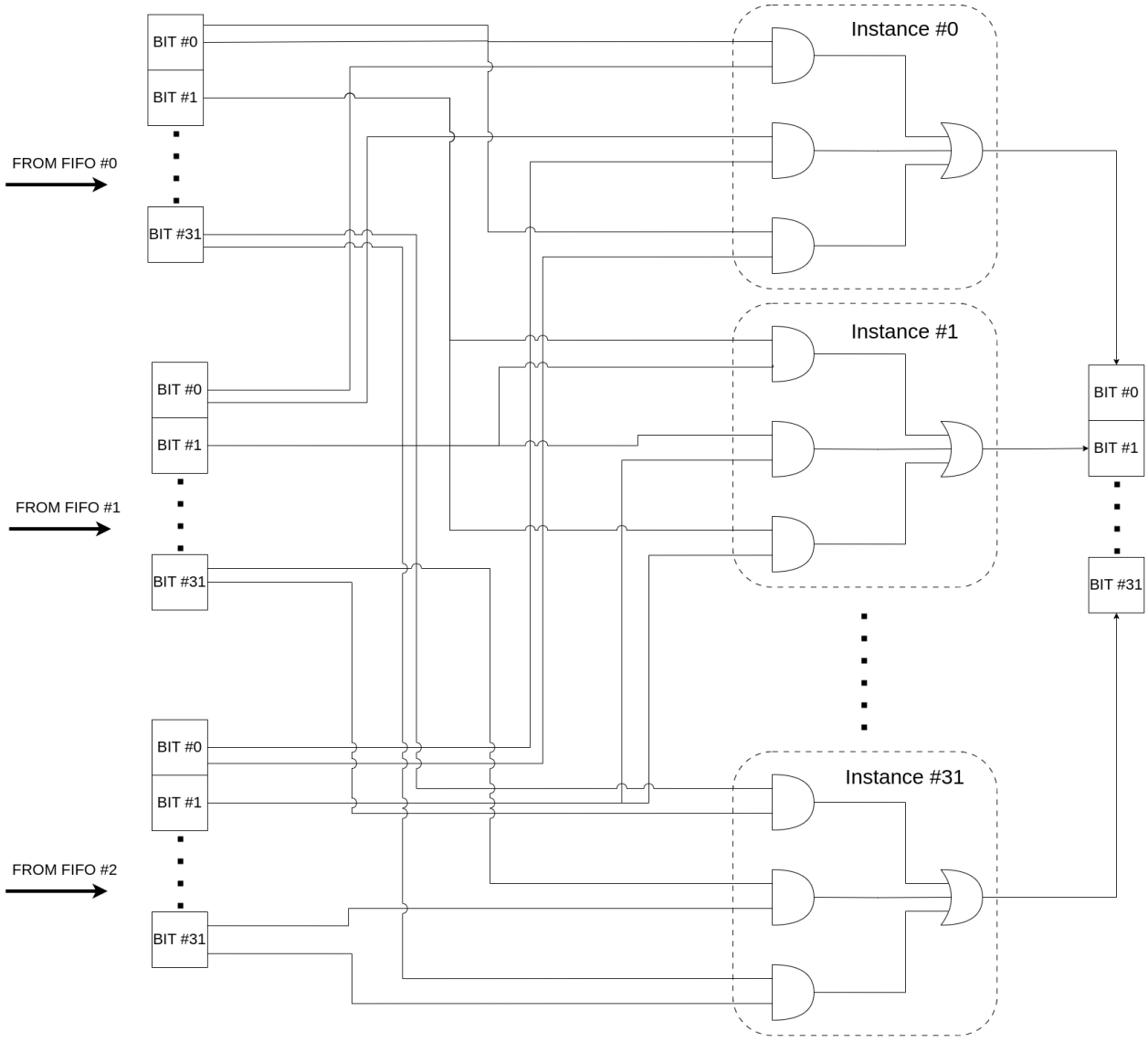


Figure 0.2.3: Υλοποίηση συνάρτηση πλειοψηφίας στο PL

Τεχνική 2: Temporal Redundancy

Η τεχνική αυτή βασίζεται στην μεταφορά κάθε αντιγράφου της πληροφορίας από ένα διαφορετικο μονοπάτι με μία χρονική καθυστέρηση. Αυτό αποσκοπεί στην εξάλειψη του κινδύνου αποτυχίας κοινού κόμβου στον controller της μνήμης, ο οποίος δεν μπορεί να τριπλασιαστεί λόγω αρχιτεκτονικής.

Τεχνική 3: Dual-Core Lockstep (DCLS)

Η τρίτη και τελευταία τεχνική που εφαρμόζεται αφορά το lockstep configuration των R5, προκειμένου να εξαλειφθεί ο κίνδυνος αποτυχίας κοινού κόμβου από τη διαδικασία του voting. Πραγματοποιείται , δηλαδή, voting και από τους δύο πυρήνες (Σχήμα 0.2.2), οι οποίοι συγκρίνουν τις εξόδους τους κάθε χρονική στιγμή. Σε περιπτώση που διαπιστωθεί ασυμφωνία μεταξύ τους, επαναλαμβάνεται η διαδικασία για τη συγκεκριμένη τριάδα αριθμών. Μάλιστα, η

υλοποίηση αυτή δεν εισάγει πρόσθετη καθυστέρηση εκτέλεσης, συγκριτικά με την υλοποίηση του ενός R5.

0.3 Αξιολόγηση και Αποτελέσματα

Fault Pruning

Για την αξιολόγηση της προτεινόμενης αρχιτεκτονικής, είναι αναγκαία η εισαγωγή σφαλμάτων σε flip-flops του συστήματος που περιέχουν ωφέλιμη πληροφορία, γεγονός δύσκολο εξαιτίας του τεράστιου αριθμού τους και της δυσκολίας πρόσβασης σε αυτά. Επομένως, εφαρμόζονται τεχνικές fault pruning, βασισμένες στην έννοια του fault-collapsing, προκειμένου να μειωθεί ο συνολικός αριθμός αναγκαίων flip-flops για εισαγωγή σφάλματων και να προσομοιωθούν σφάλματα σε μη προσβάσιμα flip-flops, με σφάλματα σε προσβάσιμα flip-flops που έχουν την ίδια επίδραση στην έξοδο του συστήματος. Έτσι, με βάση αυτές τις τεχνικές, διαπιστώνεται ότι η εισαγωγή σφαλμάτων στο σύστημα πρέπει να γίνει μεταξύ των AXI FIFOs και του επιταχυντή υλικού, αλλά και στους ενδιάμεσους καταχωρητές του R5 κατα τη διαδικασία του voting.

Εισαγωγή Saboteur

Ο μηχανισμός με τον οποίο εισάγονται σφάλματα στα συγκεκριμένα flip-flops στηρίζεται στην αλλαγή της λογικής τιμής μεταξύ δύο διαδοχικών flip-flops, το οποίο πραγματοποιείται από ειδικά κυκλώματα, τους saboteurs. Συγκεκριμένα, οι saboteurs ελέγχονται εξωτερικά, καθορίζοντας πότε αυτοί θα είναι ενεργοί και θα εισάγουν σφάλματα, με αποτέλεσμα να μπορεί να ρυθμιστεί εξωτερικά ο ρυθμός εισαγωγής σφαλμάτων στο σύστημα (SER).

Πειραματικά Αποτελέσματα

Οι κύριες μετρικές για την αξιολόγηση του συστήματας είναι η συνάρτηση αξιοπιστίας (Reliability curve) και ο μέσος χρόνο αποτυχίας (MTTF). Παράλληλα, μετριέται τόσο το resource utilization από τη μερια του PL, όσο και η καθύστερη εκτέλεσης λόγω του voting. Ως επιταχυντής υλικού στα πειράματα, επιλέγεται ένας accumulator και ένας offset adder, δύο κυκλώματα με διαφορετικά χαρακτηριστικά επεξεργασίας. Στα διαγράμματα 0.3.1, 0.3.2 και 0.3.3, απεικονίζεται η συνάρτηση αξιοπιστίας για τρια διαφορετικα configurations και για τους δύο επιταχυντές υλικού.

Παρατηρούμε ότι το σύστημα του offset adder φτανει στην αποτυχία γρηγορότερα από αυτό του accumulator, μόνο στη περίπτωση του TMR με normal PS voting. Αυτό συμβαίνει, διοτι έχοντας αυξήσει την αξιοπιστία του μονοπατιού μέσω του TMR, το τμήμα στο οποίο οφείλονται οι περισσότερες αποτυχίες του συστήματος ειναι το PS voting. Άρα, δεδομένου ότι παράγονται περισσότερα δεδομένα από τον offset adder στην έξοδο, περισσότερα δεδομένα θα υποβληθούν στη διαδικασία του voting, με αποτέλεσμα να αυξάνεται η πιθανότητα αποτυχίας σε σχέση με το σύστημα του accumulator.

Επίσης, παρατηρούμε ένα οριζόντιο τμήμα στη καμπύλη αξιοπιστίας του unmitigated συστήματος με τον accumulator. Αυτό οφείλεται στον τρόπο με τον οποίο διεξάγονται οι μετρήσεις στα πειράματα, υπάρχει σε όλες τις καμπύλες, αλλά είναι ορατό μόνο σε αυτη τη περίπτωση, γιατί η

διάρκεια του είναι συγκρίσιμη με το συνολικό χρόνο που διαρκεί το πείραμα (μέχρι δηλαδή το σύστημα να φτάσει στην αποτυχία).

Από τα διαγράμματα 0.3.4 και 0.3.5, διαπιστώνουμε πως ο μέσος χρόνος αποτυχίας του συστήματος με TMR είναι κατά 120 φορές μεγαλύτερος από αυτόν του συστήματος με την αρχιτεκτονική αναφοράς, ενώ όταν προσθέτουμε και το lockstep voting ο χρόνος αυτός γίνεται κατά 9290 φορές μεγαλύτερος, σε σχέση με το αρχικό.

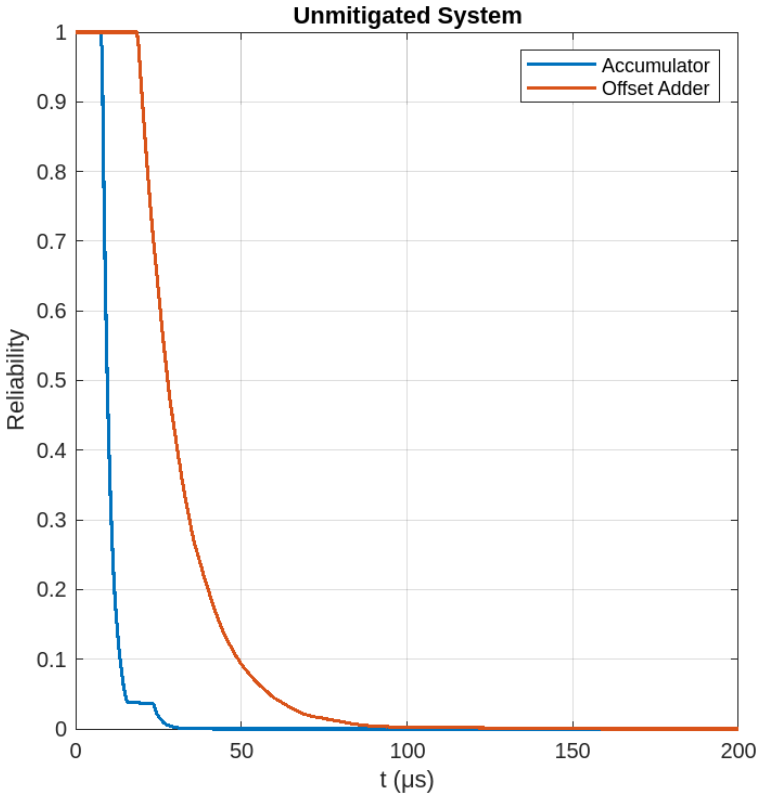


Figure 0.3.1: Unmitigated system

Οσον αφορά το resource utilization, το TMR δεν καταλαμβάνει πάνω απο το 3% ενός συγκεκριμένου resource, υποδεικνύοντας ότι η προτεινόμενη αρχιτεκτονική δεν εμποδίζει την ανάπτυξη πρόσθετων επιταχυντών υλικού στο FPGA. Τέλος, παρατηρούμε μια γραμμική σχέση μεταξύ καθυστέρηση εκτέλεσης (κύκλοι R5), λόγω voting, και όγκου πληροφορίας, το οποίο παραμένει σταθερό τόσο σε normal όσο και σε lockstep voting.

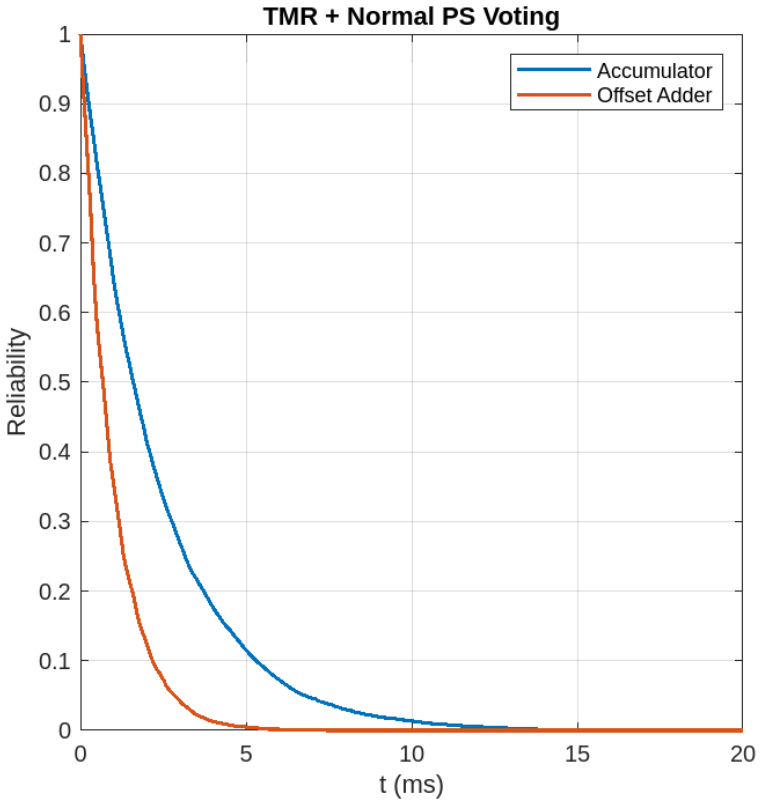


Figure 0.3.2: TMR + Normal PS Voting

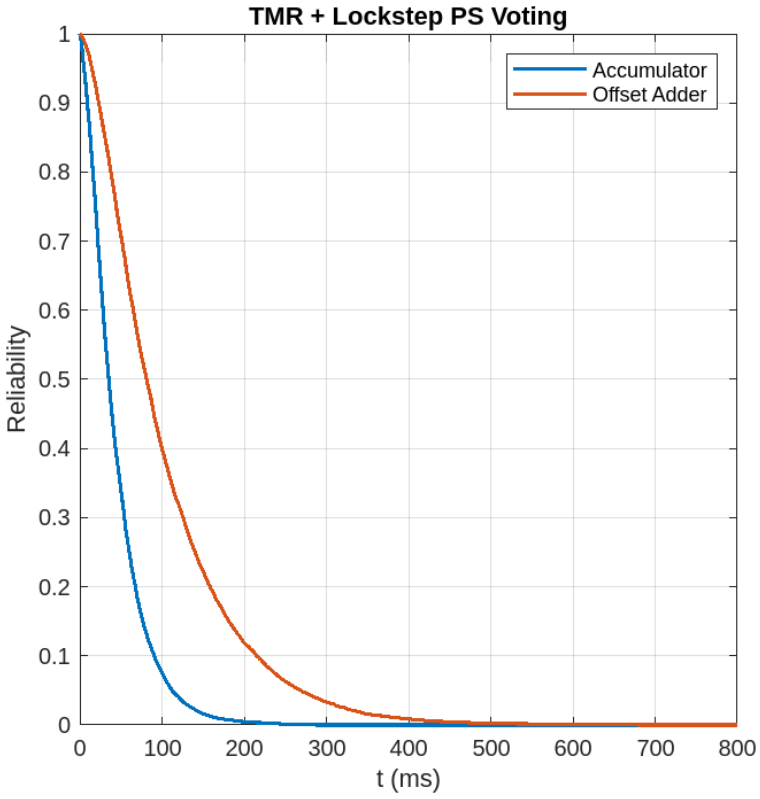


Figure 0.3.3: TMR + Lockstep PS Voting

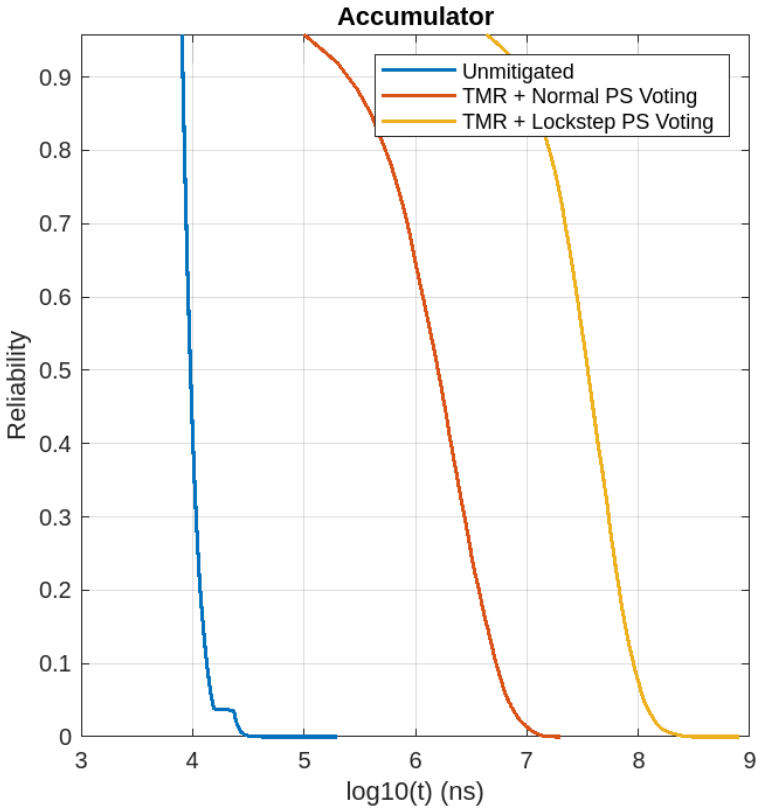


Figure 0.3.4: Three configurations compared

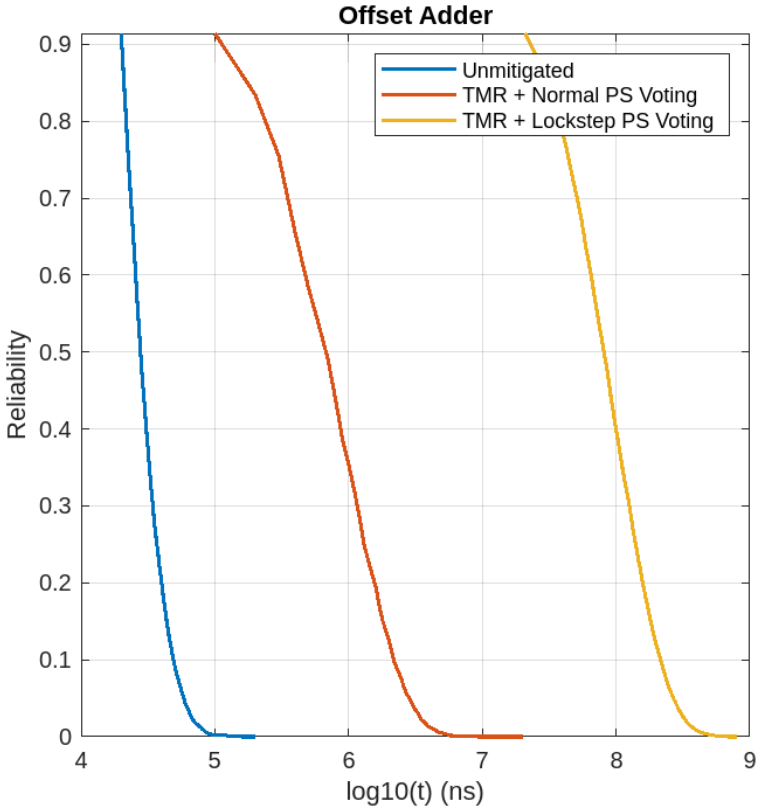


Figure 0.3.5: Three configurations compared

Configuration	Module	LUTs	FFs	BRAM
Unmitigated	AXI DMA (2)	898	1400	0
	AXI SMC (1)	1691	2509	0
	AXI DATA FIFO (2)	160	130	8
TMR + Normal PS Voting	AXI DMA (6)	2694	4200	0
	AXI SMC (3)	5061	7527	0
	AXI DATA FIFO (6)	480	390	24
TMR + Lockstep PS Voting	AXI DMA (6)	2694	4200	0
	AXI SMC (3)	5061	7527	0
	AXI DATA FIFO (6)	480	390	24

Table 1: Resource Utilization (ο αριθμός στην παρένθεση αντιπροσωπεύει τον αριθμό των αντιγράφων του συγκεκριμένου IP στο FPGA)

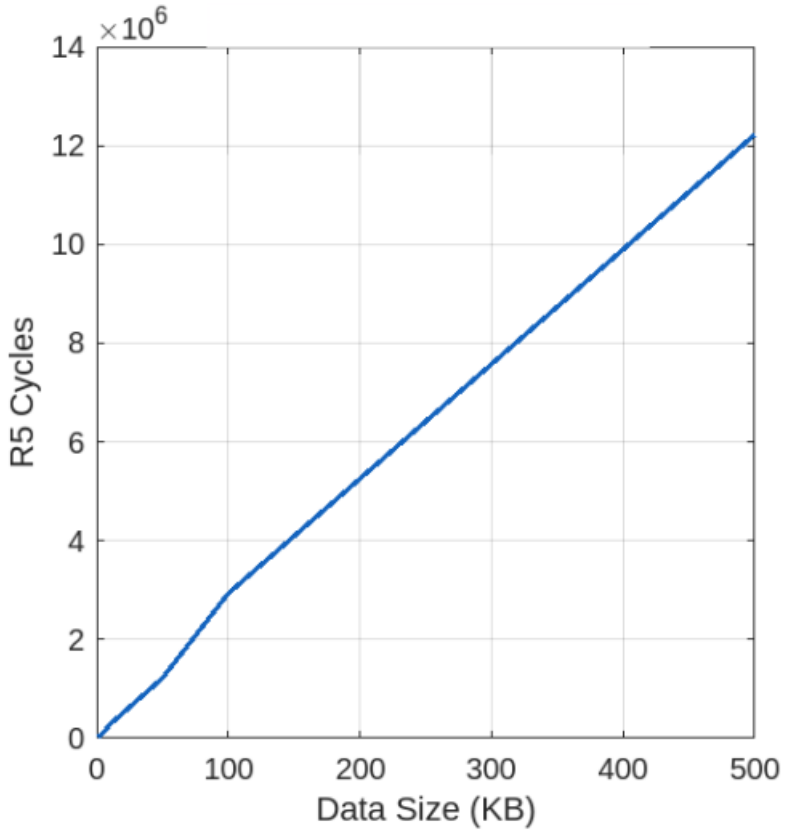


Figure 0.3.6: Execution Overhead

Chapter 1

Introduction

The NewSpace era is considered the period from the beginning of the 21st century till now and is defined by the commercialization of space activites, meaning that more and more private companies enter the space industry, prompting the development of new business models, focused on minimizing costs and shortening time-to-market. In the context of minimizing costs, a reduction in size and complexity of spacecrafts was introduced with focus on micro satellites, known as CubeSats. Furthermore, shorter operational lifespan and transition to Low Earth Orbit (LEO) took the place of 10-15 year lifespan and Geostationary Earth Orbit (GEO) of older missions [1]. As a result, the choice of Electrical, Electronic and Electromechanical (EEE) components, has favored Commercial Off-The-Shelf (COTS) parts, instead of space-qualified components which are specifically engineered to withstand the harsh conditions of space, but they are neither readily available in the market nor cost-effective [2].

The strict requirements for real-time performance and low power consumption, has forced the space industry to transition from general-purpose processors, such as CPUs and micro-controllers, to advanced hardware accelerators, like the Field-Programmable Gate Arrays (FPGAs), in order to perform intensive tasks, like Digital Signal Processing (DSP) or AI acceleration, in the context of space-specific applications e.g. Earth Observation (EO) [3–6].

However, COTS System-on-Chip (SoC) FPGAs are inherently more susceptible to radiation, as they are typically developed without considering radiation effects, so their reliability in orbit is uncertain [2]. To address this issue, space engineers have employed a practice known as up-screening, which is subjecting COTS components to rigorous testing to evaluate their suitability for space missions. Although up-screening cannot offer the same level of reliability as radiation-hardened components, it helps engineers better understand potential failure modes and causes. Also, for digital components, many system-level redundancies can be implemented due to the availability of a large programmable memory, which aim to mitigate the effects of radiation. [1, 2].

1.1 Problem Statement

In the context of SoC FPGAs, the concept of *computation offloading* is frequently encountered. This term refers to the delegation of computationally intensive tasks to a separate

processing entity, typically a hardware accelerator instantiated within the FPGA fabric. Consequently, data must be transferred from a memory location, commonly the DDR memory within the Processing System (PS), through various architectural components to the accelerator (or kernel) implemented in the Programmable Logic (PL), and back again. In the following, this data transfer route will be referred to as the *offloading datapath*.

However, under radiation-prone environmental conditions, the integrity of data traversing the offloading datapath cannot be guaranteed. This concern is substantiated by findings in the relevant literature. For instance, in work [7], a reliability analysis of the Advanced eXtensible Interface (AXI) Interconnect Intellectual Property (IP) Core is presented, revealing that this component may introduce errors during data transmission. The AXI Interconnect is a Xilinx IP Core, which facilitates communication among AXI masters and slaves with heterogeneous interface characteristics, such as differing data widths, protocols, clock domains, and connectivity patterns (e.g., one-to-many or many-to-one). It typically serves as a critical node within the offloading datapath. Further support for this observation is provided in work [8], where the authors conduct a comprehensive reliability evaluation of a complete PL subsystem designed for matrix multiplication. This subsystem includes the accelerator, its interface components, interconnect structures, and optionally, a local memory. Through a modular testing approach, the study demonstrates that interconnect structures, specifically the AXI Direct Memory Access (AXI DMA) IP Core and the AXI Interconnect IP Core, exhibit a high susceptibility to faults and should therefore be taken into account in the design of fault-tolerant systems. Similar conclusions are drawn in [9].

Moreover, the authors in [10] conducted a comprehensive reliability analysis focusing specifically on the uncore components within a SoC. Uncore components, distinct from the processing elements, encompass critical subsystems such as the DDR memory subsystem, caches, AXI interfaces, and logic interconnects. These components play pivotal roles, particularly within the offloading datapath of the system. The analysis revealed significant vulnerabilities in the crossbar interconnects and the DDR controller, highlighting their susceptibility to faults. Consequently, these components have been identified as potential sources of errors during data transfers between DDR memory and the hardware kernel, emphasizing the importance of addressing reliability concerns in these areas.

1.2 Related Work

A limited but growing body of research has focused on developing mitigation techniques for ensuring reliable data transmission in SoC architectures. Lázaro et al. [11] explored redundancy mechanisms specifically tailored for internal buses interfacing with low-speed peripherals. The authors introduced a solution based on a redundant system architecture and a custom IP Core, referred to as AXILiteRedundant. This IP Core replicates the signals from a single slave interface to three separate master interfaces, each connected to an external redundant peripheral. Additionally, it incorporates a Triple Modular Redundancy (TMR) voter, which determines the correct master value to be transmitted to the slave interface. The IP Core provides feedback to the processor, indicating whether all three masters agreed, whether one diverged but could be corrected via majority voting, or whether all three produced differing values, signaling the presence of a fault. The study details how the AXI protocol

was adapted to accommodate this redundancy scheme and elaborates on the architectural components of the proposed solution. The authors conclude that their method improves the resilience of interconnects and peripheral IPs, while maintaining resource efficiency and incurring minimal power overhead when implemented on FPGAs.

In work [12], a parity-based Dual Modular Redundancy (DMR) method is proposed to enhance the reliability of data transfers between processing cores and FPGA-based accelerators. The approach emphasizes minimal structural modifications to the interface designs generated by Vivado. The proposed architecture includes both an encoder and a decoder: the encoder duplicates the input data frame, partitions it into groups, and appends parity bits to each group. The decoder reconstructs the output data by selecting a valid channel for each group based on the recalculated and received parity bits, thus ensuring data correctness in the absence of detected errors. The results indicate that this scheme approaches the fault tolerance of TMR-based methods, while offering advantages in terms of reduced power consumption and area utilization.

Bertozzi et al. [13] conducted a comparative analysis of various coding schemes used to detect transmission errors and examined their trade-offs in energy efficiency and reliability for on-chip communication links. Their study evaluates Hamming codes, Cyclic Redundancy Check (CRC) codes, and simple parity bit schemes. Hamming codes are capable of correcting single-bit errors, while CRC and parity methods rely on retransmission to address errors. The findings suggest that, for the studied schemes, particularly CRC and Hamming codes, retransmission is generally more energy-efficient than on-the-fly error correction. CRC codes, although highly effective for error detection, do not inherently support correction and are computationally influenced by the choice of generator polynomial.

A broader perspective is presented by Mach et al. [14], who provide a comprehensive survey of existing protection techniques for mitigating transient faults in on-chip interconnects. Their work is particularly relevant to embedded processors used in safety- and mission-critical domains, such as automotive and aerospace applications. The authors classify fault-tolerant strategies into three categories: information redundancy, temporal redundancy, and spatial (hardware-based) redundancy. Information redundancy involves the use of parity bits and Error Detection and Correction (EDAC) schemes, such as Single Error Correction, Double Error Detection (SECDED), and is suitable for high-performance systems, albeit with increased complexity. Temporal redundancy, better suited for low-performance systems, relies on repeated execution over time but necessitates careful modeling of fault durations. Spatial redundancy techniques, such as TMR, are simpler to implement but incur significant overhead in terms of area and power. The study highlights that the selection of an appropriate mitigation technique should be guided by application-specific requirements, operational conditions, and the feasibility of modifying external system components.

1.3 Thesis Contributions

The contributions of this thesis are outlined as follows:

- A fault-tolerant architecture is proposed for data transmission between the main memory and the hardware kernel during computation offloading. The solution leverages

standard Xilinx IP cores, ensuring compatibility with existing Xilinx software interfaces. It combines different fault-tolerant techniques, targeting both the PS and the PL subsystem.

- A custom fault injection campaign is developed to evaluate the proposed fault-tolerant architecture on the Zynq UltraScale+ MPSoC. This fault injection mechanism enables controlled generation of errors and facilitates quantitative assessment of the design behavior under fault conditions.
- The reliability and error resilience of the proposed fault-tolerant techniques are systematically analyzed, providing insight into their effectiveness and suitability for safety-critical applications.

1.4 Thesis Outline

This thesis is organized as follows:

- **Chapter 2 – Background**: This chapter presents essential background information and foundational concepts relevant to this research.
- **Chapter 3 – Proposed Fault-Tolerant Architecture**: This chapter presents the proposed fault-tolerant architecture aimed at improving the dependability of data transfers in SoC FPGAs.
- **Chapter 4 – Evaluation**: This chapter outlines the evaluation methodology and results for the proposed fault-tolerant architecture.
- **Chapter 5 – Conclusion and Future Work**: The final chapter summarizes the key contributions and findings of the thesis. It also discusses limitations and outlines directions for future research.

Chapter 2

Background

This chapter provides essential background information and foundational concepts pertinent to the research undertaken in this thesis. The discussion begins by exploring the Space Radiation Environment, detailing the types and effects of ionizing radiation encountered in space. Emphasis is placed on differentiating cumulative radiation effects from single-event effects (SEEs), with a particular focus on the mechanisms underlying SEEs and their implications for electronics. Subsequently, the chapter delves into the concept of Dependability, introducing key terminologies critical for understanding system resilience. It clearly defines the interrelated concepts of faults, errors, and failures, and elaborates on various fault tolerance techniques employed to mitigate these issues. The metrics used to quantify dependability, including Reliability and Mean Time to Failure (MTTF), are elaborated upon to provide a comprehensive understanding of system performance evaluation. Lastly, the chapter addresses the architecture and characteristics of SoC FPGAs, particularly highlighting the Zynq UltraScale+ MPSoC used as the platform for this study. The FPGA architecture is described, including its logic blocks, programmable routing resources, and specialized hardware elements. Additionally, a detailed examination of the MPSoC's integrated processing units, memory subsystems, and interconnection infrastructure is presented, with a specific focus on the Arm Cortex-R5 processor group and its configurations for fault tolerance. The chapter concludes by introducing development tools, which facilitate hardware and software co-design.

Contents

2.1 Space Radiation Environment

2.1.1 Ionizing Radiation

Ionizing radiation is made up of particles with sufficient energy to fully dislodge an electron from its orbit, resulting in a positively charged atom. The particles associated with ionizing radiation in space are categorized into three main groups relating to the source of the radiation [15, 16]:

- Protons and electrons trapped in the Van Allen belts
- Cosmic ray protons and heavy ions
- Protons and heavy ions from solar flares.

These particles are responsible for the two general categories of radiation effects in microelectronics, cumulative effects and single event effects (SEEs).

2.1.2 Radiation Effects on Electronics

Cumulative Effects

Total Ionizing Dose (TID) results from gradual degradation caused by the accumulated energy absorbed by a material. In a metal-oxide semiconductor (MOS) device, like a transistor, electron-hole pairs generated within the gate oxide are rapidly separated due to the electric field present in the space charge region (Figure 2.1.1). The high-mobility electrons swiftly move away, while the slower-moving holes drift in the opposite direction. These holes encounter various trapping sites within the oxide (crystalline flaws), where they are readily captured. Therefore, charge buildup is induced, leading to parametric failures, such as changes in device characteristics, like leakage current and threshold voltage, or even complete functional breakdowns [16, 17].

Single Event Effects (SEE)

A Single Event Effect (SEE) is a spontaneous event, that occurs when radiation particles interact with the semiconductor lattice of transistors, transferring energy to its atomic structure. This interaction can create depletion regions within the transistor (Figure 2.1.2) by leaving behind an excess of charge carriers, either electrons or holes. The resulting imbalance can alter the flow of electrical current, either initiating or interrupting it, depending on the nature and direction of the generated charges. SEEs are generally categorized into two main types: soft errors and hard errors. Soft errors are temporary issues specific to the device, typically appearing as transient pulses or bit flips. The most common types are the Single Event Transient (SET) and the Single Event Upset (SEU), which may appear as a transient signal (spike) in combinational logic, or as a bit flip in memory cells in sequential logic. SETs can transform into SEUs, if the occurrence of the signal spike happens at the same time with the clock edge, which may store a wrong value in the memory element. Resetting the device or rewriting the data can correct the SEUs. Another type of soft error is the Single Event Functional Interrupt (SEFI), when the device ceases normal operations. SEFI generally occurs when an SEU affects the control circuitry, placing the device in an unintended state. In

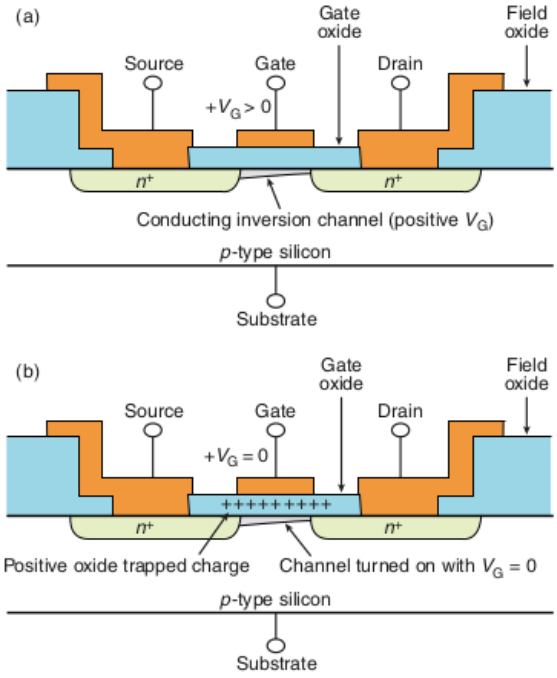


Figure 2.1.1: (a) Normal operation of n-channel MOSFET, (b) Operation under positive charge buildup (holes) due to radiation, that produces a negative threshold voltage shift.

the case of FPGA devices, it is usually related to SEUs in configuration memory and often requires a power reset to recover [18]. On the other hand, hard errors are typically more severe and may lead to permanent damage. A Single Hard Error (SHE) causes a lasting change in the device's operation, such as a stuck bit in memory [16]. TID as well as hard errors are beyond the focus of our investigation. In this thesis, we explore the effects of the SEUs, that produce corrupt data in the output of the system, without affecting its functionality.

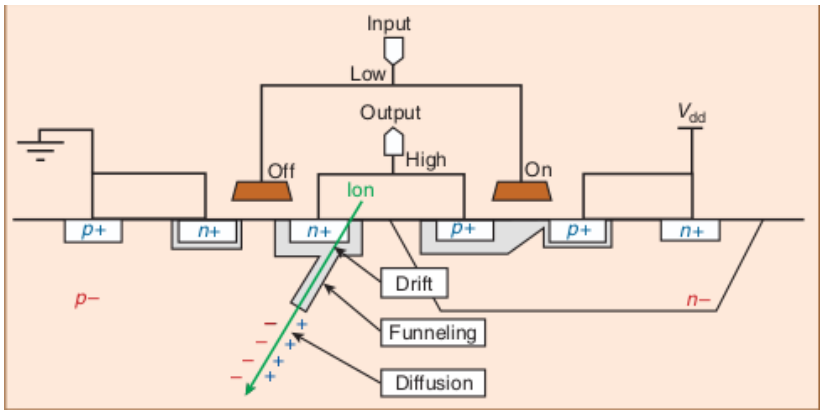


Figure 2.1.2: Single ion striking a memory cell.

2.1.3 Soft Error Rate (SER)

The Soft Error Rate (SER), which is the rate of SEU/SET occurence, is expressed in Failure in Time (FIT) units. One FIT unit equals one soft error per billion hours of operation.

Because SEUs/SETs are statistically independent, we can add multiple SERs to extract the total SER for a device. For example, a 32-bit register with SER of each bit equal to 100 FIT, has a total SER of 3200 FIT.

2.2 Dependability

2.2.1 Faults, Errors and Failures

According to [19], in fault-tolerant design, three main concepts are crucial: faults, errors, and failures. These are linked in a cause-and-effect chain, faults lead to errors, and errors can result in failures.

- A fault refers to a flaw or defect in hardware or software. This could be a physical issue, like a short in a circuit or a software problem, such as a programming loop that doesn't terminate. Essentially, it's something abnormal that has the potential to cause incorrect operation (physical universe).
- An error occurs when a fault causes the system to behave incorrectly or produce an incorrect result. For example, if a fault in a circuit changes a signal that should be a logic 0 to a logic 1, that's an error. The fault has caused the system to operate wrongly, leading to this error (information universe).
- A failure happens when an error affects the overall system's ability to function as intended. It means the system no longer meets its expected performance or behavior. For instance, a control line in a circuit may still work electrically (despite a fault), but if it doesn't turn a valve on or off as expected, then the user experiences a failure, even if the system appears operational from a technical standpoint (external universe).

2.2.2 Fault Tolerance Techniques

Fault tolerance techniques is one way to attain dependability for a system. Fault tolerance aims to failure avoidance and can be split into two categories: error detection and system recovery, as shown in Figure 2.2.1. Typically, after a fault is handled, corrective maintenance is performed to eliminate the faults that were identified. The key difference between fault tolerance and maintenance is that maintenance involves intervention by an external party. In closed systems, like the hardware on a deep space probe, removing faults is not feasible in practice. Rollback and rollforward recovery methods are triggered after an error is detected. In contrast, compensation can be used, either when needed or on a regular basis, based on specific times or events, regardless of whether an error has actually been detected. In this thesis, we apply techniques on the domain of error detection and error handling. We do not explore fault handling techniques [19].

Triple Modular Redundancy (TMR)

Triple Modular Redundancy (TMR) is a highly effective method for masking errors and enhancing the reliability. This approach involves creating three identical copies of the circuit/module. Their outputs (and sometimes inputs) are fed into a voter circuit/module,

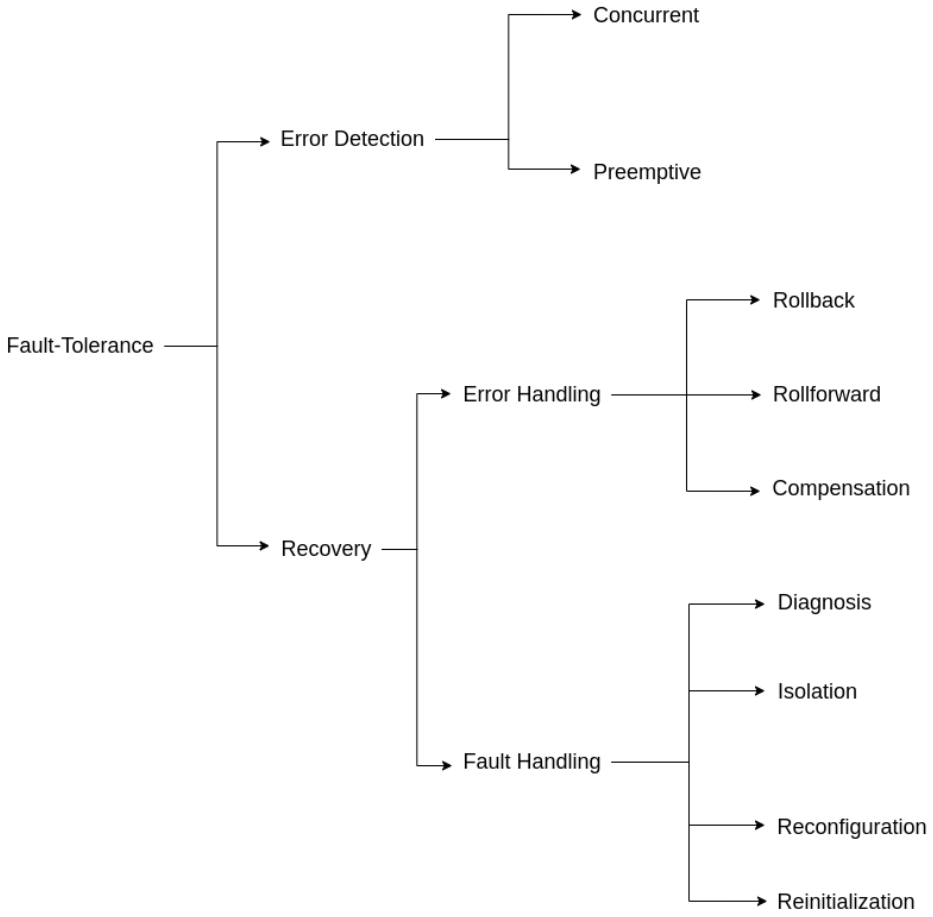


Figure 2.2.1: Fault Tolerance Techniques

which selects the majority result to determine the correct output. To eliminate the risk of a single point of failure, the voter itself is often replicated three times. TMR extends the operational reliability of the system, providing a window of time for fault handling before additional failures occur and compromise the entire system. When this technique is applied in a group of processor cores, it is referred to as Triple-Core Lockstep (TCLS).

Duplication with comparison

Duplication with comparison is a redundancy technique where two identical systems run in parallel and their outputs are compared to detect errors. While this method can identify that an error occurred, it cannot determine which module is faulty. It also has limitations, such as the potential for both modules to fail the same way due to shared input errors or comparator faults. When this technique is applied in a group of processor cores, it is referred to as Dual-Core Lockstep (DCLS).

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) is a form of redundant coding (information redundancy) widely used for detecting upsets. CRCs are capable of identifying errors across millions of

bits using just a single CRC value. Due to its high redundancy, even a single-bit change in the configuration leads to a significantly different CRC result. However, CRCs cannot pinpoint the exact location of an error, which limits their use to detection, rather than correction. CRC calculation is based on the remainder from a specialized polynomial division.

Stand-by Sparing

In this method, one module functions normally while one or more additional modules remain inactive as backups or spares. When a fault detection system identifies that a module has failed, it also determines which specific module is affected. Once the faulty module is identified, it is taken out of operation and replaced by a standby module. This reconfiguration is conceptually similar to a switch selecting output from one of the modules. The switch uses error information from the detection system to choose which module's output to use. If all modules are functioning properly, only one continues to operate, while the others remain on standby, ready to take over if needed.

Time Redundancy

The concept of time redundancy is to attempt to reduce the amount of extra hardware at the expense of additional time. The same task is performed two and three times and then the results are compared to determine if a discrepancy exists. It is often used as a method to distinguish between transient errors and permanent errors.

Watchdog Timer

A watchdog timer is a hardware or software-based timer designed to reset a system if it isn't regularly "kicked" or "fed." In embedded systems, the main code usually runs in a loop, and the timer is reset during each loop cycle to prevent an unintended system reset. Because embedded systems often run unattended and may control critical functions, developers use timer to quickly recover from issues that might cause the system to freeze. It's common for these systems to have a backup firmware or a reliable version stored on flash memory. If the timer is triggered too many times in a row, the system may automatically switch to this backup image, replacing the current one in an attempt to restore normal operation. This behavior is important to consider when debugging, as it could lead to unintentional firmware switching during system analysis.

2.2.3 Dependability Metrics

Reliability

The reliability of a system, or of an individual component, is defined as the probability that the system (or component) operates correctly, i.e., it has not experienced a failure [20]. The time to failure can be modeled using a random variable T. In continuous-time systems, T is typically represented as a continuous random variable with a sample space of $[0, \infty]$ and a measurement space of [0,1]. However, in the context of digital systems, discrete random variables are more appropriate due to the inherently quantized nature of digital time representation. Accordingly, the sample and measurement spaces are discrete. The probability

mass function (PMF) of a discrete random variable is defined as:

$$p_T(t) = P(T = t) \tag{2.2.1}$$

In this thesis, the random variable T, denotes the probability that a system failure occurs within the interval between the previous quantized time point t_{x-1} and the current time point t_x. Thus, T describes a time interval in which a failure occurs, rather than an absolute failure time. To formally derive the mathematical expression for the reliability metric, the cumulative distribution function (CDF) of T must be defined:

$$F_T(t) = P(T \leq t) = \sum_{k=1}^{t} p_T(k) \tag{2.2.2}$$

This function expresses the probability that a failure has occurred by time t. Using the CDF, the reliability function R(t) is given by:

$$R(t) := 1 - F_T(t) = P(T > t) \tag{2.2.3}$$

This represents the probability that a failure has not occured by time t.

Mean Time to Failure (MTTF)

The Mean Time to Failure (MTTF) quantifies the average operational time of a system before a failure occurs, under specified environmental conditions. It is calculated as the expected value of the random variable T. Given that T represents a time interval in discrete systems, MTTF also corresponds to an expected interval of failure occurrence rather than an absolute timestamp:

$$MTTF = E[T] = \sum_{k=1} k \cdot p_T(k) \tag{2.2.4}$$

2.3 SoC FPGA

SoC FPGAs are integrated semiconductor devices that combine embedded processor cores, often based on ARM architectures, with FPGA fabric. This heterogeneous architecture enables the coexistence of software programmability with hardware reconfigurability, offering a balance between development flexibility and computational efficiency.

2.3.1 FPGA Architecture

FPGAs are reconfigurable digital integrated circuits consisting of three primary components: logic blocks, input/output (I/O) blocks, and programmable routing resources, as illustrated in Figure 2.3.1. Logic blocks implement portions of the desired circuit functionality, while I/O blocks manage communication between the FPGA and external systems. The programmable routing infrastructure enables flexible interconnection between logic blocks and I/O blocks, allowing the FPGA to be configured for a wide range of applications.

In SRAM-based FPGAs, such as those produced by Xilinx, the configuration of logic blocks and routing resources is controlled by configuration memory implemented with SRAM cells.

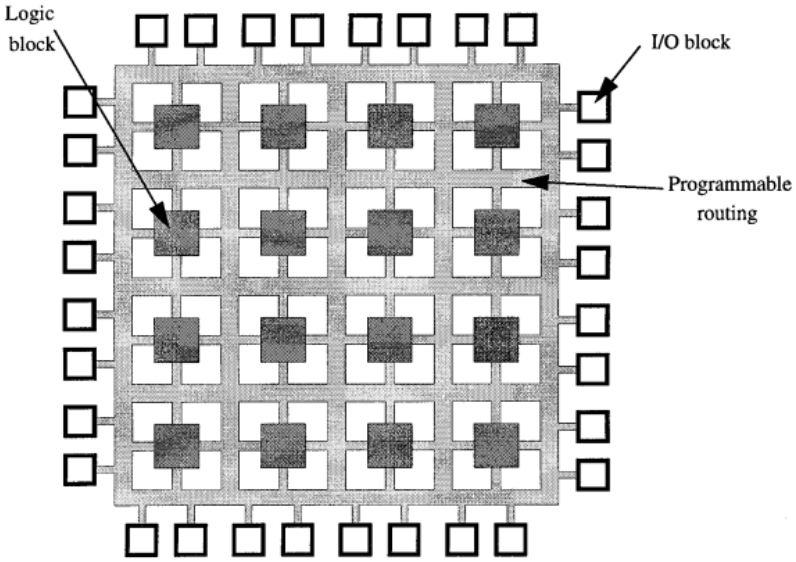


Figure 2.3.1: Simplified FPGA architecture

These SRAM cells typically function either as inputs to multiplexers or as gate control signals for pass transistors, effectively operating as configurable switches. At the core of each logic block are components known as Look-Up Tables (LUTs), which are used to implement combinational logic functions. The structure of a typical LUT is shown in Figure 2.3.2. Each LUT can realize any Boolean function of its input variables by configuring its output bits to represent the corresponding truth table. Additionally, logic blocks include flip-flops that support sequential logic operations by storing intermediate values. These flip-flops are connected to LUTs via localized interconnects [21].

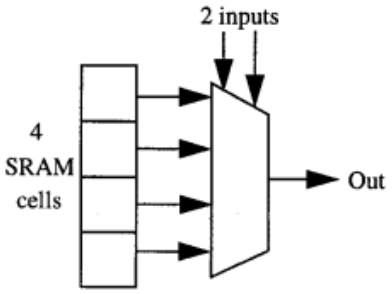


Figure 2.3.2: LUT structure

In island-style FPGAs, such as those offered by Xilinx, logic blocks are surrounded by horizontal and vertical routing channels. These channels can be connected to the inputs and outputs of logic blocks via programmable switches. At the intersection of horizontal and vertical wires, switch blocks, also composed of programmable switches, enable cross-channel connectivity, as shown in Figure 2.3.3.

Modern FPGAs also integrate specialized hardware resources beyond basic logic and routing. These include high-speed memory elements, such as Block RAMs (BRAMs), which support concurrent read and write operations, and Digital Signal Processing (DSP) blocks, which are

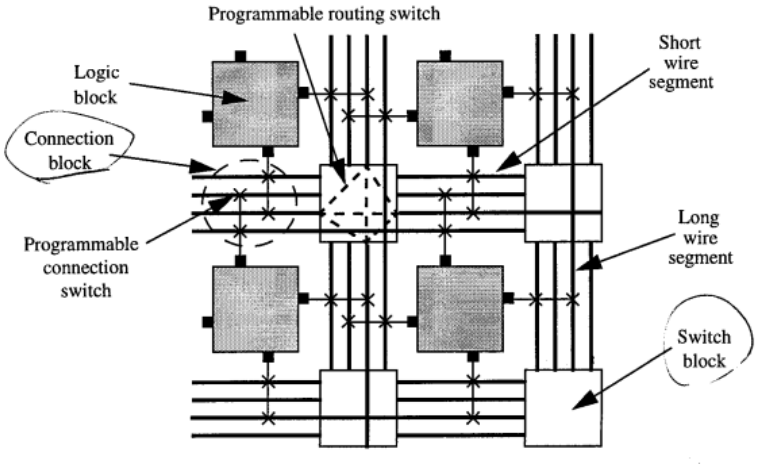


Figure 2.3.3: Programmable routing in FPGAs

optimized for performing arithmetic operations in applications such as signal processing and machine learning.

Hardware Description Language

HDLs such as VHDL and Verilog are used to model digital circuits at various abstraction levels, from high-level behavior to gate-level implementations. They support simulation, allowing designers to verify functionality before synthesis and hardware deployment. In synthesis, RTL (Register-Transfer Level) descriptions are converted into netlists suitable for FPGA implementation. RTL focuses on the flow of data between registers and the logic that processes it. Despite offering precise control over hardware, HDLs are complex and require familiarity with hardware concepts. Simulation, debugging, and synthesis are often time-consuming, particularly when resolving timing issues or working under tight deadlines.

Computer-Aided Design (CAD) Flow

The FPGA design flow begins with synthesis, where HDL code is translated into a netlist composed of basic logic gates. This netlist is then optimized for resource usage and timing constraints, preparing it for physical implementation. Next is placement, where logic elements are mapped to specific physical locations on the FPGA. Placement aims to minimize wire length, balance logic usage, and reduce critical path delays, while considering routing feasibility. Following placement, the routing phase connects the inputs and outputs of logic blocks using the FPGA's programmable interconnects. The router iteratively refines the design to meet timing and congestion constraints. The final stage is bitstream generation, which produces a binary file containing all configuration data (logic functions, placement, routing, and I/O settings) used to program the FPGA hardware.

2.3.2 Zynq UltraScale+ MPSoC

Architecture Overview

The EG family of devices, which serves as the platform for the design and evaluation of the proposed system architecture, comprises several integrated components, as illustrated in Figure 2.3.4. The following discussion focuses on the components most relevant to the scope of this thesis.

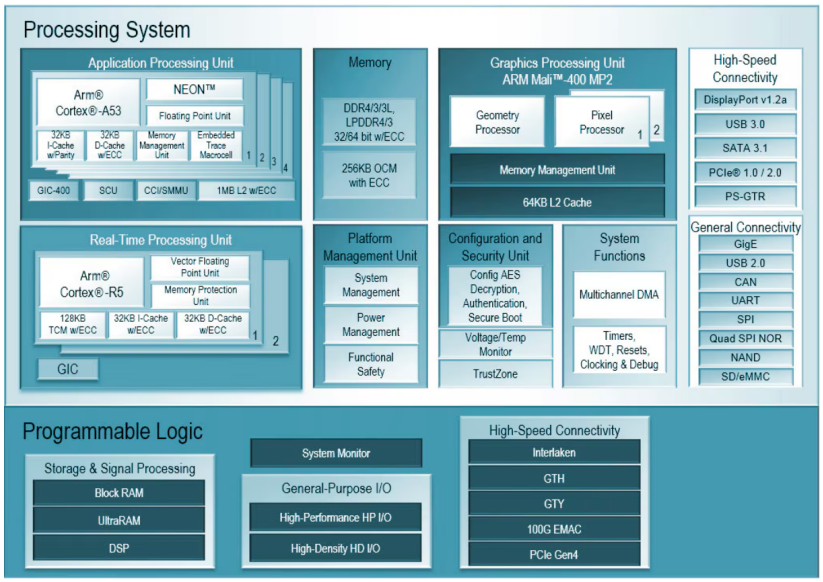


Figure 2.3.4: MPSoC EG device family

The SoC incorporates two primary processing units: the Application Processing Unit (APU) and the Real-Time Processing Unit (RPU). The APU consists of four 64-bit Arm Cortex-A53 MPCores, capable of operating in single-core, symmetric multi-processing (SMP), or asymmetric multi-processing (AMP) configurations. In contrast, the RPU includes two 32-bit Arm Cortex-R5F MPCores, designed for real-time execution [22].

In addition, the device provides a PS–PL interface, which facilitates communication between the two subsystems through AXI4 interconnects, serving as the primary channels for data exchange and includes:

- Six 128-bit/64-bit/32-bit High Performance (HP) Slave AXI interfaces from PL to PS, four HP AXI interfaces from PL to PS DDR and two high-performance coherent (HPC) ports from PL to cache coherent interconnect (CCI).
- Two 128-bit/64-bit/32-bit HP Master AXI interfaces from PS to PL.
- One 128-bit/64-bit/32-bit interface from PL to RPU in PS (PL to LPD) for low latency access to OCM.
- One 128-bit/64-bit/32-bit AXI interface from RPU in PS to PL (LPD to PL) for low latency access to PL.
- One 128-bit AXI interface (ACP port) for I/O coherent access from PL to Cortex-A53

cache memory. This interface provides coherency in hardware for Cortex-A53 cache memory.

- One 128-bit AXI interface (ACE Port) for fully coherent access from PL to Cortex-A53. This interface provides coherency in hardware for Cortex-A53 cache memory and the PL.

Beyond the processing units, the device integrates a DDR Subsystem capable of servicing read and write transactions from six application host ports. These ports interface with the DDR controller via AXI bus protocols and are implemented as six AXI slave ports:

- Two 128-bit AXI ports serve data from the Arm Cortex-A53 CPUs, RPU (Cortex-R5F and LPD peripherals), GPU, high-speed peripherals (USB3, PCIe, and SATA), and HP ports (HP0 and HP1) from the PL, all routed through the CCI.
- One 64-bit AXI port is exclusively connected to the Arm Cortex-R5F CPUs.
- One 128-bit AXI port handles transactions from the DisplayPort and the HP2 port from the PL.
- One 128-bit AXI port supports data from the HP3 and HP4 ports from the PL.
- One 128-bit AXI port serves the General DMA engine and HP5 from the PL.

The internal structure of the DDR Subsystem is depicted in Figure 2.3.5. The MPSoC also

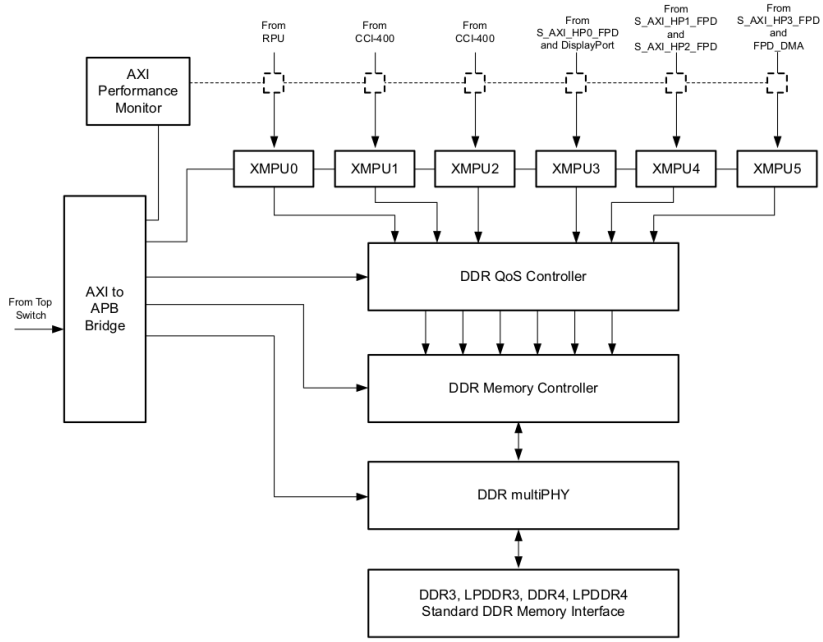


Figure 2.3.5: DDR Subsystem

includes a dedicated Platform Management Unit (PMU), a user-programmable processing subsystem tasked with power management, error handling, and optional execution of a Software Test Library (STL) for functional safety applications. Among its responsibilities is the detection and handling of lockstep errors in the Cortex-R5F cores, a topic discussed in detail in Section 2.3.2. All components within the MPSoC, including the PL, are interconnected via

a multi-layer, non-blocking Arm Advanced Microprocessor Bus Architecture (AMBA) AXI interconnect. This high-performance interconnect supports multiple simultaneous master-slave transactions, ensuring efficient communication across the system. The architecture of this interconnect is illustrated in Figure 2.3.6.

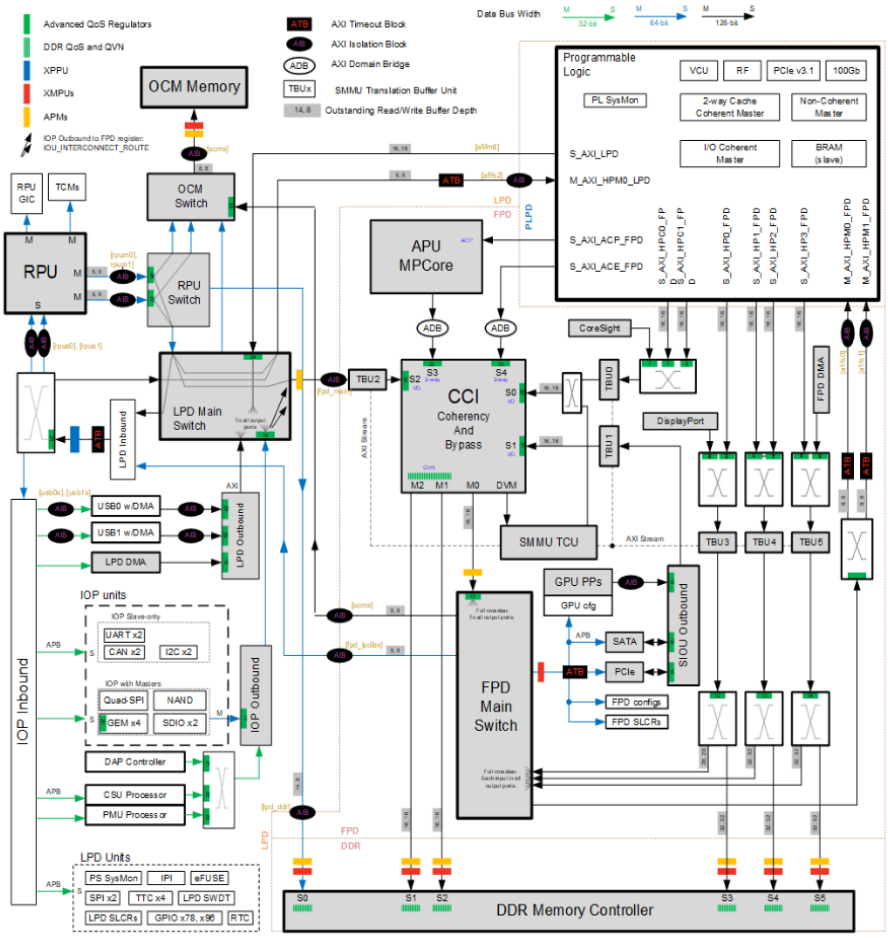


Figure 2.3.6: MPSoC Interconnect Architecture

Arm Cortex-R5

The Arm Cortex-R5 processor group can be configured to operate with either a single core or a pair of cores. Its behavior and functionality vary depending on the selected configuration, which determines the mode of operation and available fault tolerance mechanisms [23]:

- **Single CPU:** This mode features a single R5 core operating independently.
- **Twin CPU:** This configuration includes two distinct, independent Cortex-R5 cores. Each core possesses its own private cache memory, debug infrastructure, and system bus interfaces. While the cores do not interact directly within the processor group, inter-core communication can occur through other components of the SoC. Parameters such as cache size can be configured separately for each core, offering greater design flexibility.

- **Redundant CPU (Lockstep Mode):** In this configuration, the group operates with one active core accompanied by a redundant replica of the core's logic. The redundant logic receives identical input signals and shares the same cache memory with the main core, allowing a single cache structure to serve both. Although the redundant logic executes operations in synchrony with the main processor, it does not influence system behavior. All system-facing outputs are generated solely by the main core. Optional comparator circuits can be implemented to monitor consistency between the outputs of the active and redundant logic. These comparators use dedicated input and output signal lines, DCCMINP[7:0], DCCMINP2[7:0], DCCMOUT[7:0], and DCCMOUT2[7:0], to interact with the broader SoC.

- **Split/Lock:** This configuration provides two R5 cores capable of operating in one of two modes. In *Split Mode* (performance mode), both cores run independently, similar to the twin-core configuration. In *Lock Mode* (safety mode), the cores operate in lockstep for increased fault tolerance. Transitions between these modes are restricted to power-on reset states. The configuration is controlled via the SLCLAMP and SLSPLIT input signals, which dictate the operational mode during system initialization.

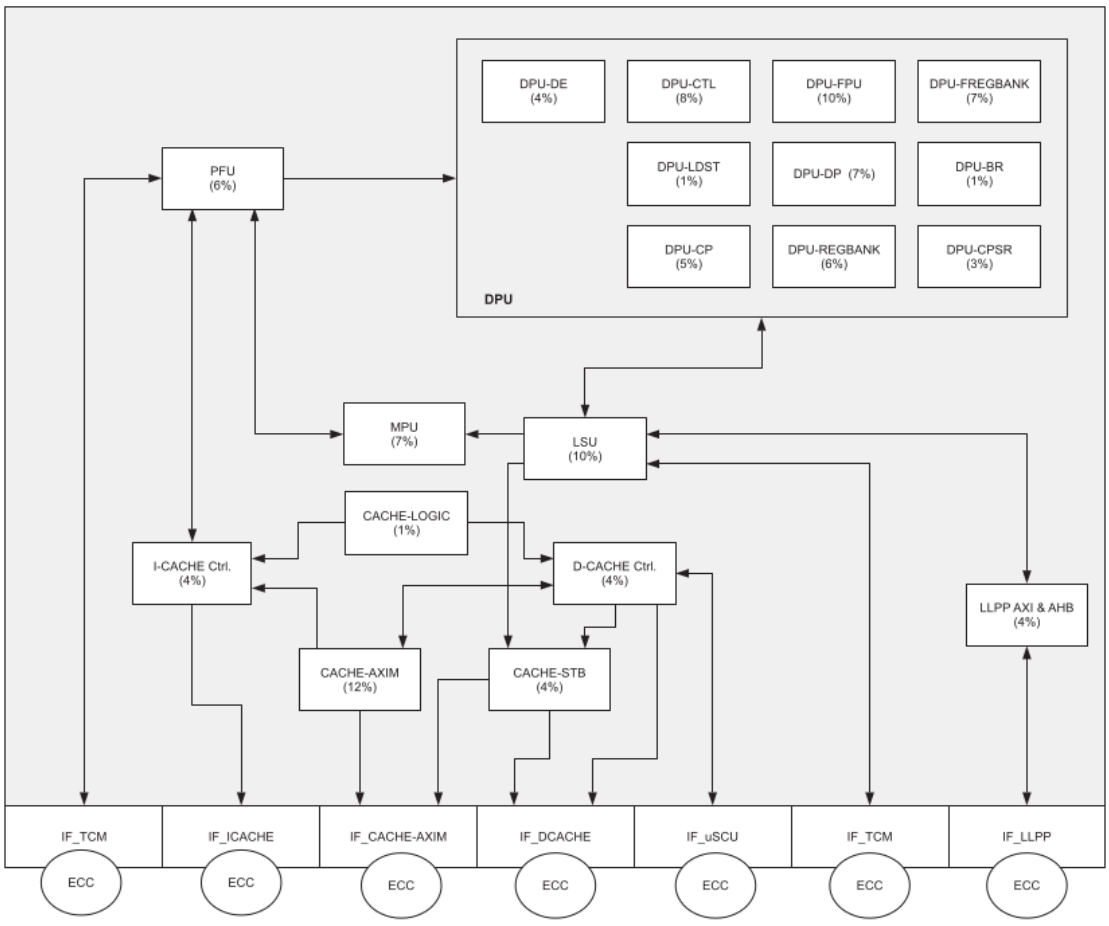


Figure 2.3.7: Arm Cortex-R5 micro-architecture and micro-components

2.3.3 Development Tools for SoC FPGAs

Xilinx provides two primary development tools for SoC FPGA design: the *Vivado Design Suite* and the *Vitis Unified Software Platform*. The Vivado Design Suite is a comprehensive toolchain for hardware design, simulation, synthesis, and implementation targeting Xilinx FPGAs and SoCs. It supports design entry through HDL languages (VHDL/Verilog) and schematic-based methods, with advanced capabilities for logic synthesis, placement, and routing. Vivado also provides detailed reports on performance and resource utilization, facilitating early design optimization. The Vitis Unified Software Platform supports application development across Xilinx devices, enabling efficient software development for embedded systems and hardware acceleration. Vitis abstracts low-level hardware complexity, allowing both hardware and software engineers to collaborate effectively within a unified co-design environment.

Chapter 3

Proposed Fault-Tolerant Architecture

This chapter introduces the proposed fault-tolerant architecture developed to enhance the dependability of SoC FPGA-based systems, specifically addressing vulnerabilities associated with the offloading datapath. Initially, the chapter provides a comprehensive overview of the overall fault-tolerant architecture, clearly segmenting it into distinct components and the baseline offloading data path in a fault-prone architecture is described to establish a reference point for evaluating fault-tolerance enhancements. Following this baseline, the chapter details three fault-tolerance techniques: Triple Modular Redundancy (TMR), Time Redundancy, and Dual-Core Lockstep (DCLS). Each technique is thoroughly examined, outlining how they are integrated into the architecture to mitigate potential faults and their specific roles within different segments of the data path. Lastly, the chapter presents the design of custom hardware kernels, an accumulator and an offset adder, implemented for systematic evaluation of the proposed fault-tolerant architecture, due to their diverse operational characteristics.

Contents

3.1 Overview

A high-level representation of the SoC FPGA incorporating the proposed fault-tolerant architecture is illustrated in Figure 3.1.1. This architecture aims to mitigate the effects of radiation in the offloading datapath, as we discussed in the previous chapter.

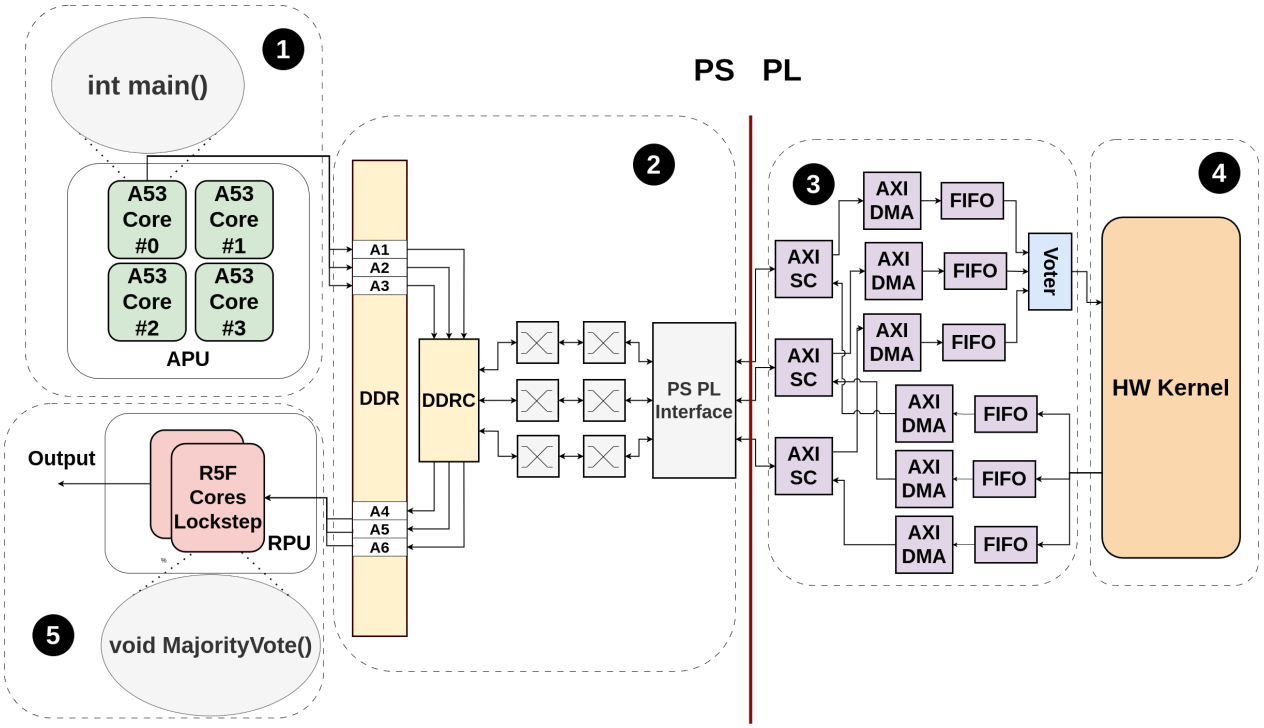


Figure 3.1.1: Proposed Fault-Tolerant Architecture

In the figure, the connectors indicate the direction of data flow: unidirectional connectors denote a single direction of data transfer, while bidirectional connectors represent data exchange in both directions. For clarity and to facilitate comprehension, the architecture is divided into distinct sections. This segmentation serves to highlight the specific areas addressed within this thesis, the application of fault-tolerant techniques, and the methodologies employed. The sections are defined as follows:

1. Application Processing Unit (APU): Executes the main application.
2. APU to PS Subsystem Boundary: Comprises uncore components, including the DDR subsystem, crossbar switches, and the PS-PL interface.
3. AXI Interconnect and IP Cores in the FPGA Fabric: Facilitates communication between the PS-PL interface and the hardware kernel. This section also incorporates additional logic to support the implementation of fault-tolerant techniques.
4. Hardware Kernel: Performs the core data processing tasks.
5. Real-Time Processing Unit (RPU): Plays a critical role in the proposed fault-tolerant architecture.

The fault-tolerant techniques presented in this thesis target Sections 2, 3, and 5. Accordingly, Sections 1 and 4 are assumed to be fault-free within the scope of this work. Although additional components, such as the AXI Interconnect and Processor System Reset IP cores are part of the system, they are not considered integral to the fault-tolerant architecture. Their role is limited to the orchestration and configuration of the offloading process and they are considered fault-free in the context of this thesis. Sections 3.3, 3.4, and 3.5 present the application of TMR, Time Redundancy, and DCLS, respectively, detailing where, how, and why each technique is integrated into the proposed architecture.

3.2 Offloading Data Path in a Fault-Prone Architecture

In the baseline fault-prone architecture shown in Figure 3.2.1, input data designated for processing by the hardware kernel are stored in a predefined DDR memory region. These data originate either from the application running on the PS or from an external source (e.g., sensor), and are transferred via DMA. The application configures an AXI DMA controller with the base memory address and transfer size, then initiates the transfer. The AXI DMA core, acting as an AXI master, fetches the data from DDR through one of the HP AXI slave interfaces of the PS–PL interconnect.

The memory request passes through the AXI SmartConnect IP Core (AXI SC), a protocol- and bandwidth-optimized interconnect that supports heterogeneous AXI interfaces and handles data width, protocol, and clock domain adaptation. The HP interface connects, via a hierarchical cascade of crossbar switches, to one of the AXI ports of the DDR subsystem. These crossbars function as switching fabrics that route memory requests from various peripherals (e.g., PL, DMA, DisplayPort) to the DDR Controller (DDRC), which returns the requested data to the DMA. The DMA temporarily buffers the data in its internal FIFO, then streams it via the AXI4-Stream protocol to an AXI4-Stream Data FIFO IP Core, which serves as a buffering interface to the kernel. The process is mirrored for output data: once processing is complete, the kernel sends the results through an output FIFO to the AXI DMA, which writes them back to DDR upon being triggered by the PS application.

Figure 3.2.2 illustrates the baseline fault-prone architecture implemented in Vivado Block Diagram. This is used as a reference design, around which the rest of the architecture will be developed. The block labeled Zynq MPSoC represents the PS in Figure 3.2.1 and is responsible for configuring all components within the subsystem (controls the AXI DMAs, as well as the input and output data streams).

3.3 FT Technique 1: TMR

3.3.1 Triplication of the Offloading Data Path

The proposed TMR scheme enhances reliability by triplicating all critical nodes in the offloading datapath between PS and PL, as illustrated in Figure 3.1.1. Initially, input data are triplicated and stored in three separate DDR regions (A1, A2, A3, Figure 3.1.1). This redundancy ensures that a single-bit fault in one memory region does not affect all replicas, which would render TMR ineffective. Regarding the DDRC, although it is a shared resource,

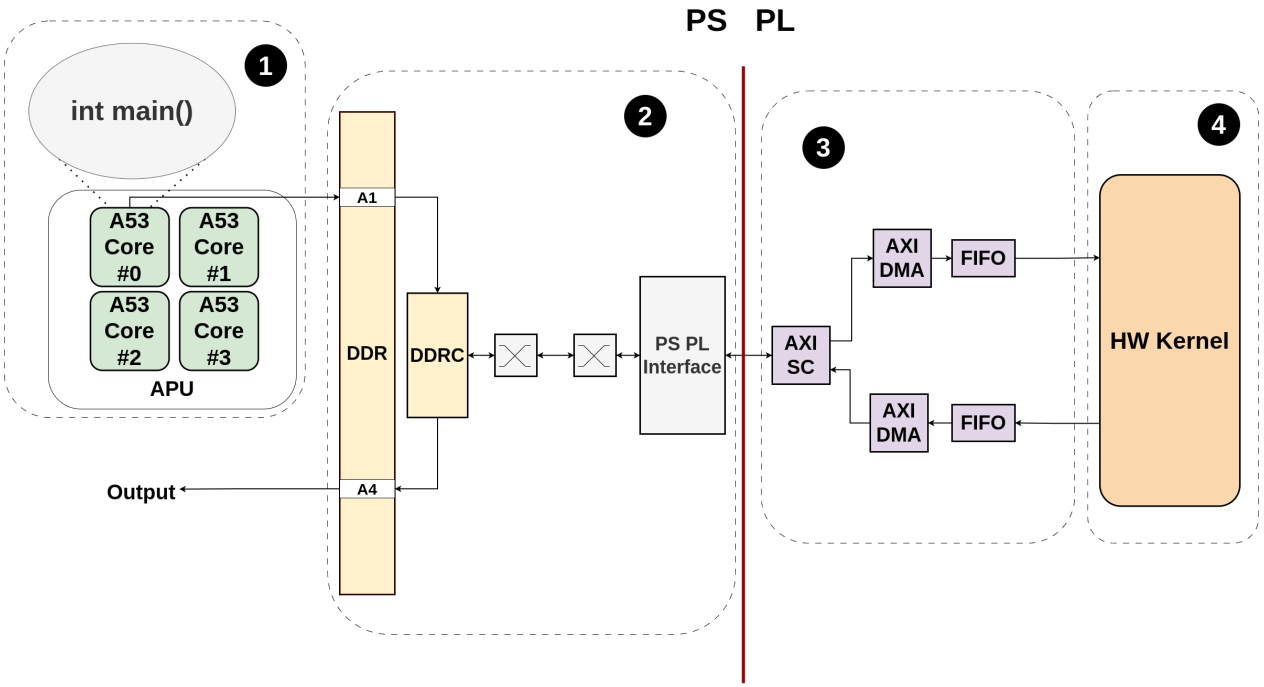


Figure 3.2.1: Baseline Fault-Prone Architecture

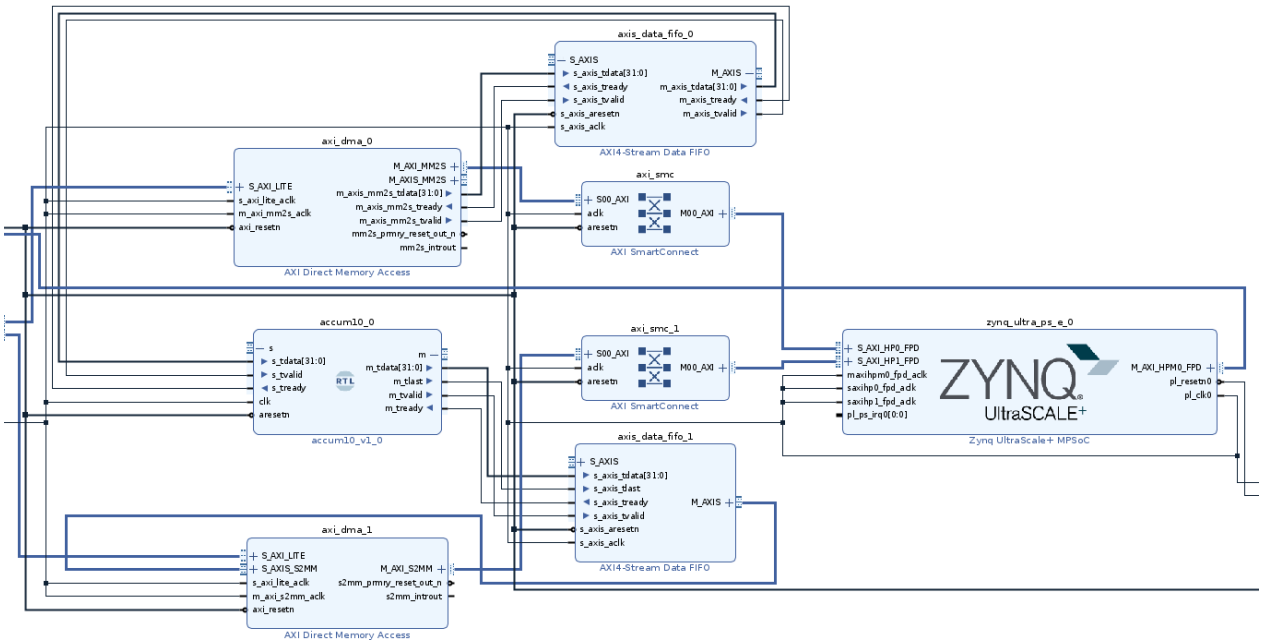


Figure 3.2.2: Block Design

it cannot be replicated or bypassed, due to MPSoC constraints. Furthermore, data isolation is reinforced by routing each data replica through a separate PS–PL AXI slave port (S0, S1, S3), ensuring traversal through independent crossbar switches cascades, utilizing all three of them, that exist in the SoC (Figure 2.3.6).

This selection avoids shared interconnects (e.g., S1 and S2 share a crossbar), eliminating potential single points of failure, since crossbar logic typically includes sequential elements such as arbiters, making it is susceptible to soft errors and therefore must be encompassed in the fault-tolerant scheme. The TMR implementation includes also, in the PL subsystem, three AXI SmartConnect IPs, six AXI DMA controllers (three for input, three for output), and six AXI4-Stream FIFOs, depicted in Figure 3.3.1. Each selected AXI slave port connects to a distinct SmartConnect, which connects to two DMA cores: one for reading from DDR, the other for writing back processed data, while each DMA is paired with a dedicated AXI4-Stream FIFO, as shown in Figure 3.1.1.

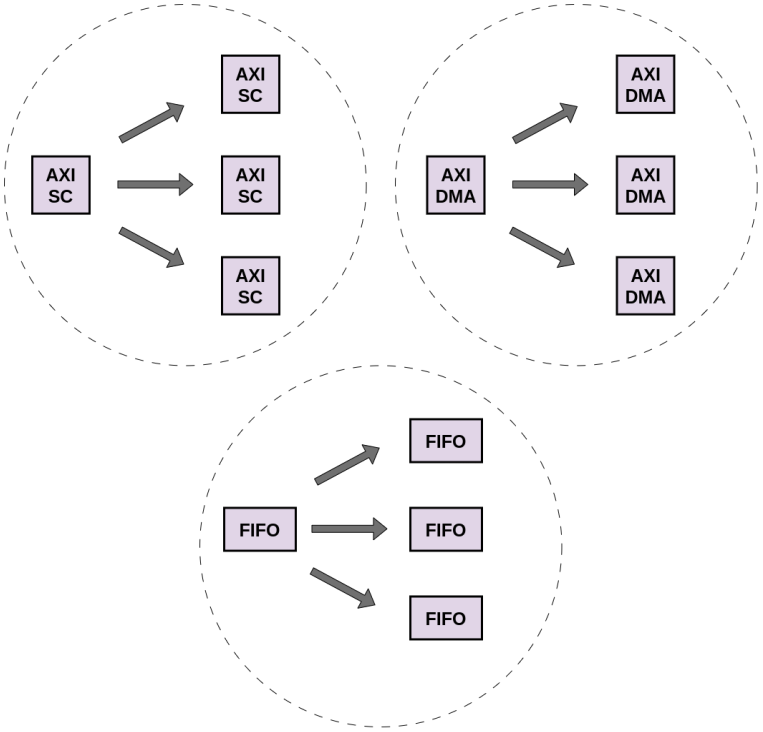


Figure 3.3.1: Triplication of Critical IP Blocks

The outputs of the three input FIFOs are routed to a majority voter module, which connects to the AXI4-Stream slave interface of the kernel. Only the payload lines of each FIFO are connected to the voter (Figure 3.3.2), while control signals are wired directly to the kernel, with additional logic to support the three-to-one, master-to-slave interface configuration. A similar strategy is applied to the kernel's output, with its AXI4-Stream master interface connected in parallel to the three output FIFOs (Figure 3.3.2).

This symmetrical design ensures full triplication in both directions. Importantly, the forward (DDR-to-kernel) and return (kernel-to-DDR) paths are independent to each other, since a fault occurring in one path is effectively overwritten or cleared when new data propagate through the same hardware in the reverse direction. This allows shared hardware resources, such as SmartConnect IPs, PS–PL interfaces, internal crossbars, and the DDRC, to be reused across both paths without cross-contamination.

Since FIFO buffers are used in the aforementioned implementation, the problem of the appropriate FIFO buffer sizing arises. The depth of the FIFO buffers depends heavily on the

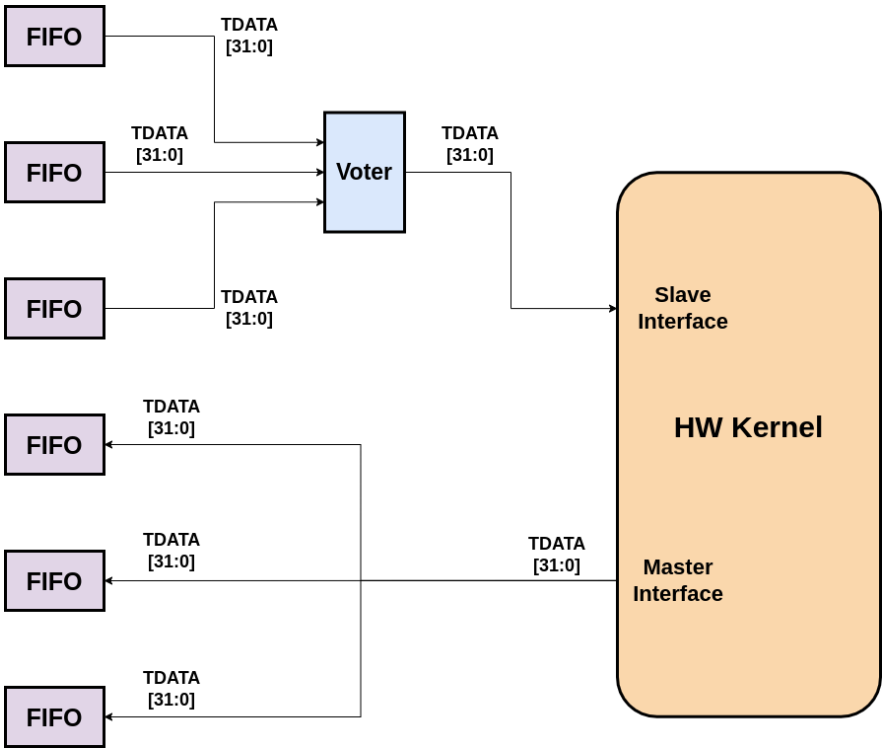


Figure 3.3.2: FIFO–Kernel Communication: detailed view of the interface between the two (Zoom-in of Figure 3.1.1)

rate at which data is produced or consumed from or to the hardware kernel and the rest of the PL memory system. Since our implementation serves as a generic framework, the FIFO sizing is out of scope of this thesis and left as a design variable to the end-user.

3.3.2 AXI4-Stream Protocol

The AXI4-Stream protocol is a standard interface designed for unidirectional data transfers between components. It enables communication from a single master, which produces data, to a single slave, which consumes it. Additionally, the protocol is scalable and supports more complex configurations involving multiple masters and slaves. AXI4-Stream facilitates the construction of generic interconnects capable of performing upsizing, downsizing, and routing operations by supporting multiple data streams over a shared set of signals [24]. The essential interface signals for AXI4-Stream are illustrated in Figure 3.3.3. In its simplest implementation, the AXI4-Stream interface includes the following signals: TVALID, TREADY, TLAST, and TDATA. Data transfer occurs when both TVALID and TREADY are asserted simultaneously. This two-way handshake mechanism allows both the master and the slave to control the data flow rate. The TVALID signal, asserted by the master, indicates that valid data is available, whereas the TREADY signal, asserted by the slave, indicates that it is ready to receive data. Importantly, once a master asserts TVALID, it must remain asserted until the handshake is completed. A master cannot delay asserting TVALID in response to TREADY. However, a slave is permitted to wait for TVALID before asserting TREADY, and it may also deassert TREADY prior to TVALID being asserted [24].

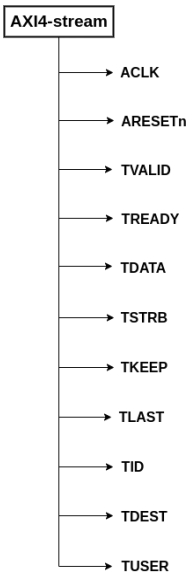


Figure 3.3.3: AXI4-Stream signals

Handshake Synchronization in TMR Systems

As discussed in the preceding section, three FIFOs must interface with the kernel using the AXI4-Stream protocol, either in a three-masters-to-one-slave or one-master-to-three-slaves configuration. These configurations are implemented without an AXI interconnect.

In the three-masters-to-one-slave configuration, the slave should receive a logic high on its TVALID input only when all three masters have asserted their respective TVALID signals, indicating that they are each ready to transmit valid data. To enforce this condition, a logical AND gate is inserted, as depicted in Figure 3.3.4. However, the routing of the TREADY signal from the slave back to the masters presents challenges. A given master should only see a logic high on its TREADY input when the slave has asserted TREADY and the other two masters have also asserted their TVALID signals. This condition is enforced by inserting three logical AND gates, with their outputs routed to the corresponding master TREADY input. This prevents scenarios where one master perceives the handshake as complete (due to high TREADY and its own asserted TVALID) while the other two masters are still idle and the slave waits for TVALID assertion, resulting in synchronization errors. Such a situation is problematic in designs where the slave does not wait for TVALID to assert before asserting TREADY.

Conversely, in the one-master-to-three-slaves configuration, the master should only perceive a logic high on its TREADY input when all three slaves have asserted their respective TREADY signals, signifying readiness to receive data. Again, a logical AND gate is required to generate the appropriate TREADY feedback to the master, as shown in Figure 3.3.5. In this setup, the TVALID signal from the master must only be considered valid by a given slave when both the master has asserted TVALID and the other two slaves have also asserted their TREADY signals. This ensures that all slaves are prepared to receive the data concurrently. Without this gating mechanism, one slave may perceive a valid handshake and read the data prematurely, while the other two are not yet ready, leading to non-synchronization. This

again highlights the necessity of synchronizing handshake signals through logical AND gates, as shown in Figure 3.3.5, to maintain consistency across all data paths.

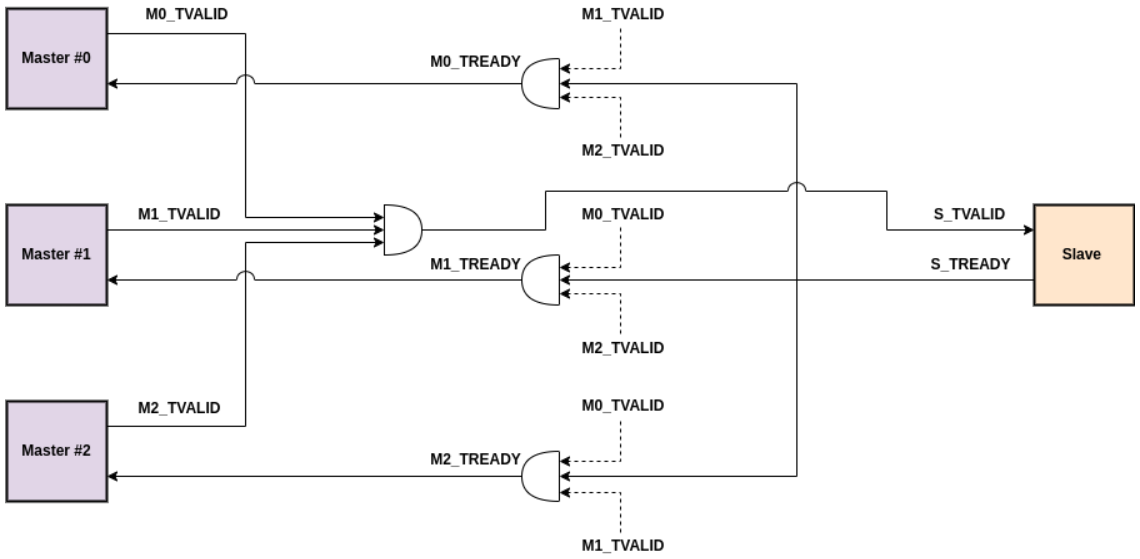


Figure 3.3.4: Three-masters-to-one-slave configuration using AXI4-Stream

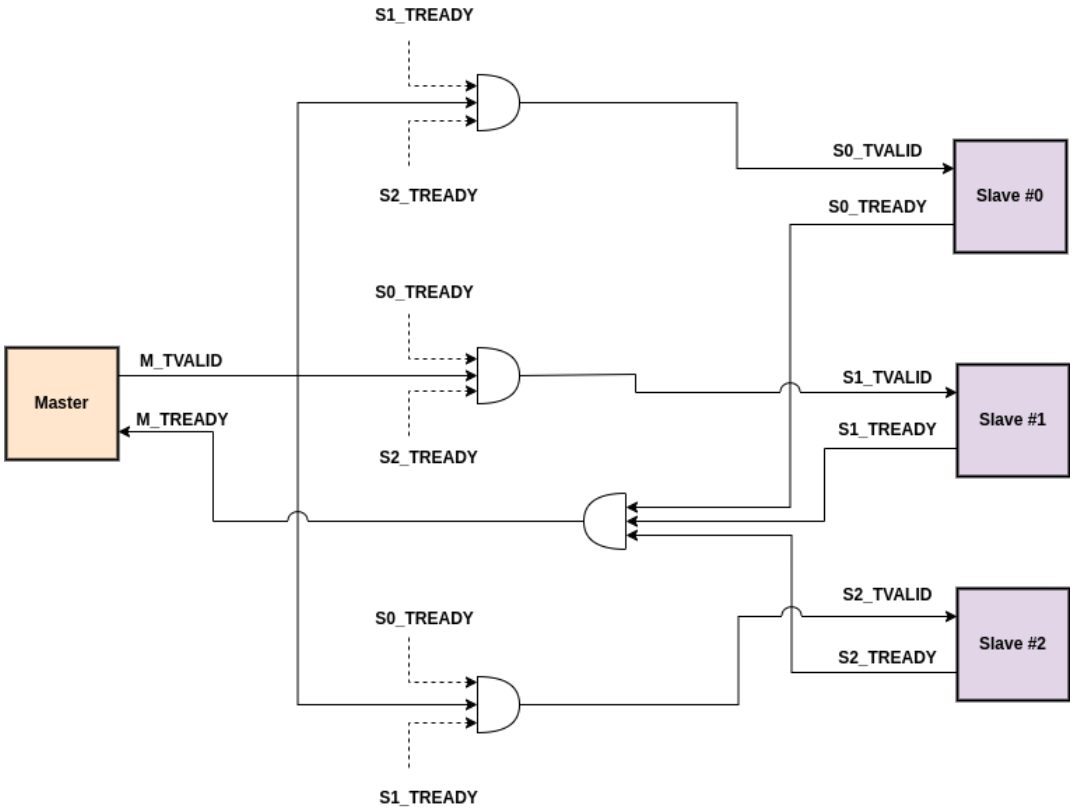


Figure 3.3.5: One-master-to-three-slaves configuration using AXI4-Stream

3.3.3 PL Voting

The Boolean majority function evaluates to logic high when at least two of its three input variables are asserted (i.e., are logic high), and to logic low otherwise. It can be mathematically expressed as:

$$V = XY + YZ + XZ \tag{3.3.1}$$

The corresponding truth table is provided in Table 3.1, and the most commonly used combinational logic implementation of the 3-input majority voter is shown in Figure 3.3.6.

X	Y	Z	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.1: Truth table for 3-bit majority voter

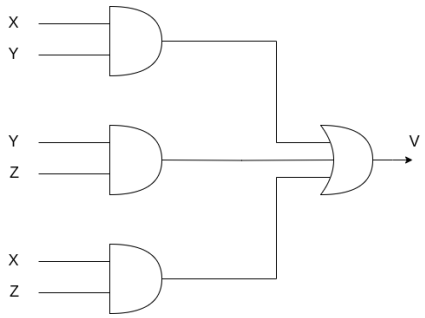


Figure 3.3.6: Most common implementation of 3-bit majority voter

Although the majority voter architecture depicted in Figure 3.3.6 may be susceptible to faults under certain conditions (and more robust alternatives exist [25]) this design is nonetheless adopted in the present work. The rationale for this choice lies in the operational context: SETs are not considered within the scope of this thesis and the voter is composed exclusively of combinational logic gates. As such, the likelihood of a transient fault manifesting and propagating through the circuit is minimal. Therefore, the majority voter is assumed to operate error-free for the purposes of this implementation. As illustrated in Figure 3.3.7, the voter module comprises 32 instances of the 3-bit majority voter. Each instance processes one bit from each of the three 32-bit data words arriving from the input FIFOs. It performs bitwise voting and writes the resulting value to the corresponding bit position in the output, as further detailed in Figure 3.3.7. It is important to highlight that this implementation does not introduce any execution overhead to the system, as the voting mechanism is completed within a single AXI clock cycle. Furthermore, the impact on the critical path is minimal, involving only two logic gates.

3.4 FT Technique 2: Time Redundancy

To overcome the architectural limitations of the DDRC, as discussed in Section 3.3.1, and to introduce redundancy at this critical point, a combination of spatial redundancy, as employed in TMR, with time redundancy is presented, forming a hybrid redundancy scheme and thereby enhancing fault tolerance in systems where duplicating memory controllers is not feasible. Specifically, the risk of a single point of failure within the DDRC can be mitigated

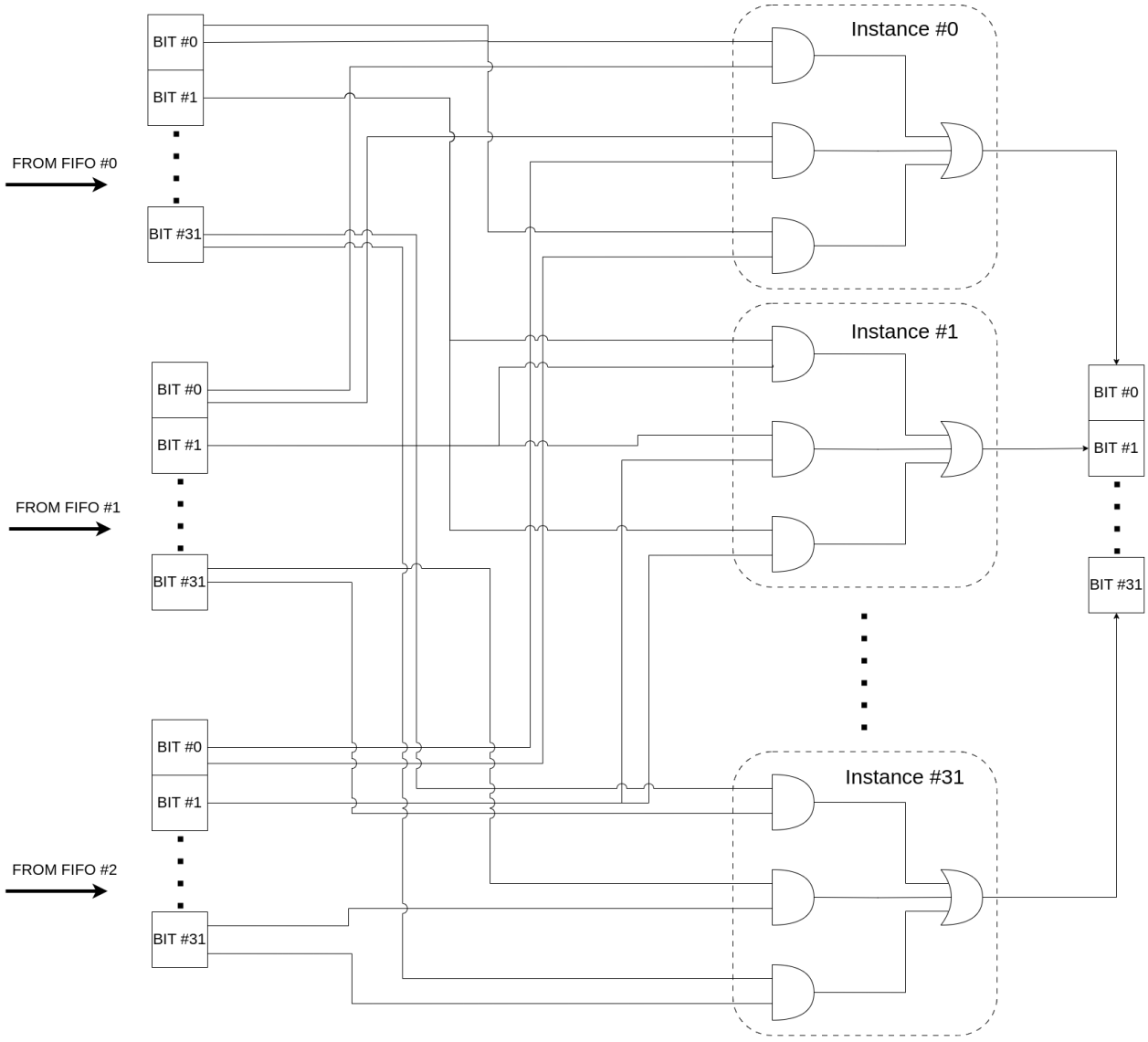


Figure 3.3.7: Implementation of 32-bit majority voter

by issuing the three data transfers (through the 3 identical paths) from DDR memory to the kernel, separated by a predefined time offset. The first data copy is routed to the first input FIFO at time t, the second copy to the second FIFO at time $t + t_{offset}$, and the third copy to the third FIFO at time $t + 2 * t_{offset}$. This redundancy is implemented by invoking the software routine responsible for configuring the AXI DMA IPs and initiating the data transfers three times, each with the appropriate delay. It is important to note that, due to the AXI4-Stream synchronization mechanism described in Section 3.3.2, the system ensures that the actual processing begins only after the third copy has arrived and has been stored in the corresponding FIFO. This synchronization guarantees data alignment across the three replicas. A similar procedure is applied in the reverse direction, from the output FIFOs back to DDR memory, maintaining the same principles of hybrid redundancy and timing separation.

3.5 FT Technique 3: DCLS

Once the data have been transmitted back to the DDR memory, each copy is stored in a distinct memory region, specifically A4, A5, and A6 (Figure 3.1.1). Similar to the PL, a dedicated hardware or software block must perform majority voting on each triad of values to derive the final, reliable result that will subsequently be utilized by the application. Since it is not feasible to introduce additional logic within the DDR subsystem, the voting operation must be delegated to an external processing core, such as the Cortex-A53 or Cortex-R5F.

However, this solution introduces certain challenges. Modern processors are increasingly susceptible to soft errors, primarily due to aggressive device scaling. Various strategies exist to protect processor cores from soft errors, spanning multiple stages of the design and manufacturing process. Some processors adopt radiation-hardened-by-process (RHBP) technologies [26], while others rely on commercial fabrication methods that incorporate fault-tolerance mechanisms such as lockstep execution [27, 28], redundant multithreading [29], or architectural-level protections integrated directly into the processor pipeline [30].

To mitigate the risk of a single point of failure during the voting process, our architecture utilizes the lockstep execution feature of the Cortex-R5F, as detailed in Section 2.3.2. In the event of a lockstep error (classified as a system-level error) the error is detected via a bit assertion in the PMU GLOBAL error status register. This assertion triggers an interrupt in the PMU, which then invokes the RPU Lockstep Error Handler routine stored in PMU RAM. The handler routine is responsible for executing a rollback-recovery technique, selectable by the user. For instance, one possible strategy involves re-executing the voting procedure. This may help to distinguish between transient faults introduced during the voting process itself and pre-existing mismatches in the input data. In the former case, a repeated voting attempt may succeed without triggering another lockstep error. In the latter case, however, the processing core lacks the information necessary to identify the faulty bits, and re-execution is likely to produce the same fault condition. In our implementation, we adopt a straightforward approach: the voting operation is retried until it completes without triggering a lockstep error. While functionally correct, this method can be time-consuming, particularly in scenarios where repeated voting attempts consistently result in errors.

3.6 Design of Hardware Kernels

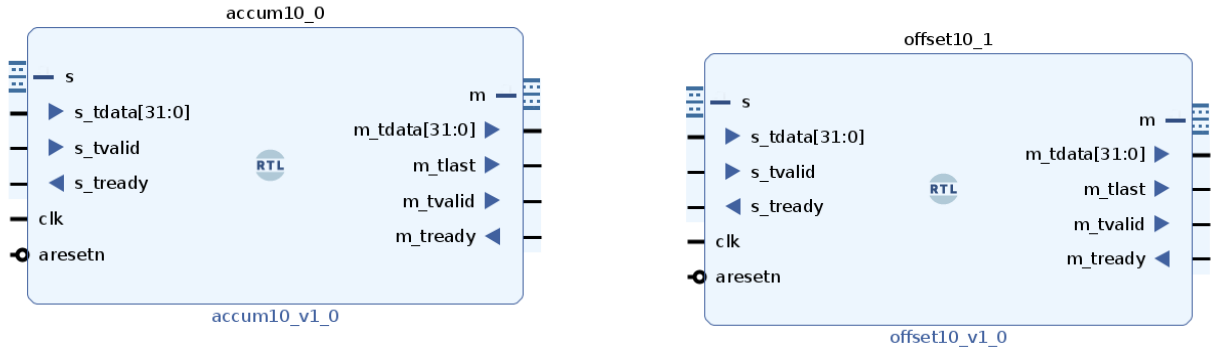
As an initial step towards the evaluation of our fault tolerant architecture, a custom hardware design was developed to support systematic analysis and experimentation. The architecture was deliberately kept simple to reduce design complexity, while still incorporating the core functionalities necessary for meaningful evaluation. To this end, two fundamental hardware circuits were implemented: an accumulator and an offset adder. This selection enables the evaluation of fault mitigation techniques on two distinct types of processing behavior, one involving temporal storage and aggregation of input values (the accumulator), and the other performing stateless, input-independent arithmetic operations (the offset adder).

Accumulator

The accumulator is designed to temporarily store a fixed number of input values (in this case, ten) and subsequently compute their sum. The module is compliant with the AXI4-Stream interface, which provides the necessary handshake signals for receiving input values and transmitting the computed result, as illustrated in Figure 3.6.1a. Functionally, the accumulator operates as a finite state machine (FSM) comprising three states: IDLE, CALCULATE, and SEND. In the IDLE state, the module accepts incoming data transactions until the internal buffer reaches its capacity. Once ten values have been received, the FSM transitions to the CALCULATE state, where the summation is performed. Finally, in the SEND state, the accumulated result is transmitted to the downstream component.

Offset Adder

The offset adder performs addition of a predefined offset to each incoming value. Like the accumulator, it is also compliant with the AXI4-Stream protocol, supporting all necessary handshake signals for data reception and transmission, as shown in Figure 3.6.1b. This module also follows a finite state machine structure with three states: IDLE, CALCULATE, and SEND. In the IDLE state, the module awaits an input value. Upon reception, it transitions to the CALCULATE state, where the offset is added to the input. The result is then forwarded, in the SEND state, to the next stage in the data path.



(a) Accumulator IP

(b) Offset Adder IP

Figure 3.6.1: Custom IP cores: Accumulator and Offset Adder

Chapter 4

Evaluation

This chapter presents the evaluation methodology and experimental results of the proposed fault-tolerant architecture. The evaluation begins with a detailed description of the fault injection campaign, employing a uniform fault injection technique targeting both the PS and the PL subsystem of the SoC FPGA. The fault injection campaign is structured into two main steps: fault pruning and saboteur insertion. Fault pruning effectively reduces the complexity of the fault space, identifying critical points within the system where faults are most impactful. Saboteur insertion involves embedding dedicated hardware components to simulate bit-flip faults, facilitating systematic and controlled fault injection. Following this, the chapter describes the experimental setup, including hardware specifics such as the Xilinx ZCU102 MPSoC development board used for implementation. The setup enables precise measurement of system dependability metrics, under varying fault-tolerance configurations. The chapter concludes by presenting and analyzing experimental results obtained from testing the two distinct hardware kernels. The results illustrate the impact of different fault-tolerant techniques on system reliability, resource utilization and execution overhead, highlighting improvements and trade-offs associated with each method.

Contents

4.1 Fault Injection Campaign

4.1.1 Overview

Existing fault injection techniques in processor-based architectures, provoked by radiation, can be classified in the following main categories:

- Radiation ground testing
- Hardware/Software Implemented Fault Injection
- Simulated Fault Injection

The first involves exposing the design under test (DUT) to a controlled radiation beam that induces physical events representative of real-world fault conditions. The second technique also operates on the physical device. However, fault events are emulated through software routines that run concurrently with the application program. These routines alter the contents of memory cells, such as registers and internal memory, which are typical targets for SEUs. Finally, the third approach relies on a HDL model of the device, wherein code modifications are implemented to simulate run-time faults across various components of the system [31].

In the context of SoC FPGAs, fault injection can target either the PS or the PL subsystem. For the PL, most FPGA-based fault injection campaigns operate by modifying the configuration memory. This is typically achieved by altering the design's bitstream to inject faults, which is then reloaded into the FPGA. The modified bitstream causes the FPGA to operate with intentional configuration errors, potentially resulting in deviations from the intended design behavior [32, 33].

To evaluate the robustness of the proposed fault-tolerant architecture, a uniform fault injection approach is employed, targeting both PS and PL subsystems. This method emulates fault occurrences by directly manipulating the logical values of selected system elements during execution. The fault injection campaign is performed on the physical device, specifically the Xilinx Zynq UltraScale+ MPSoC FPGA, where the proposed architecture is assumed to be deployed. The overall procedure consists of two primary steps:

1. Fault Pruning
2. Saboteur Insertion

The initial step in any fault injection campaign involves defining the fault space, a multi-dimensional space in which each dimension represents a specific aspect of a fault. These typically include:

- When the fault occurs (temporal dimension)
- How the fault manifests (fault type or value change)
- Where in the system the fault is located (spatial dimension)

Given that our study focuses on SEUs, the relevant fault locations are limited to D flip-flops (DFFs) within the logic blocks of the system. These are included in components along the offloading datapath, as described in Section 3.2, such as DFFs in the DDRC, AXI interfaces, and FIFOs, as well as DFFs within the register file of the Cortex-R5F cores, which are used

during PS voting. These flip-flops exclusively store data bits, excluding control bits that affect system operation and inter-block connectivity. The fault type is defined as a bit-level transition, either from logic '1' to logic '0' or vice versa. The temporal dimension consists of discrete time instants, each corresponding to individual clock cycles, starting from the initiation of the system's operation. Ideally, fault injection experiments aim to assess system behavior under every point in this fault space, with each point representing a unique fault scenario [34]. However, due to the large size and complexity of this space, exhaustive testing is often infeasible. To address this challenge, pruning techniques are employed to reduce the number of fault scenarios that need to be tested, thereby making the process tractable.

4.1.2 Step 1: Fault Pruning

In our methodology, we employ a fault pruning technique known as fault collapsing [35]. Fault collapsing is based on the principle of fault equivalence and fault dominance. Two faults are equivalent if every test detecting one also detects the other. On the other hand, Fault A dominates fault B if all tests for B are a subset of A's tests . Tests are combinations of inputs, that given a fault, result in faulty output.

In Figure 4.1.1, purple DFFs are initial targets. After pruning, gray DFFs are eliminated. For instance:

- Bitflips in FF1 and FF2 are equivalent, as both produce inverted values at the input of the logic block.
- A bitflip in FF7 dominates FF4. The former causes faulty output for all 9 input combinations, while FF4 causes faults only for a subset (e.g., 101 and 111).
- Bitflips in FF8 are unique and cannot be pruned.

Pruning also mitigates the challenge of inaccessible flip-flops in uncore components (e.g., DDRC, crossbar switches). In the forward data path (DDR to kernel), illustrated in Figure 4.1.2, logic blocks perform only non-transformational operations (e.g., data transfer or upsizing), so all flip-flops are equivalent from the SEU perspective. Thus, fault injection is applied only to the last DFF in the FIFO. For 32-bit data, and three replicas, this results in:

$$32\,\text{flip-flops} \cdot 3\,\text{replicas} = 96\,\text{fault injection targets (forward path)} \tag{4.1.1}$$

In the reverse path (kernel to DDR), shown in Figure 4.1.3, faults are injected at the first DFF in the cascade, again 96 targets in total. Importantly, injecting faults after the kernel output and before branching is invalid, as faults in each branch are neither dominant nor equivalent to faults in the shared upstream node (as seen in Figure 4.1.1, blue and red arrows).

To determine the fault injection targets during voting, we analyze the R5F assembly code for the majority voting routine:

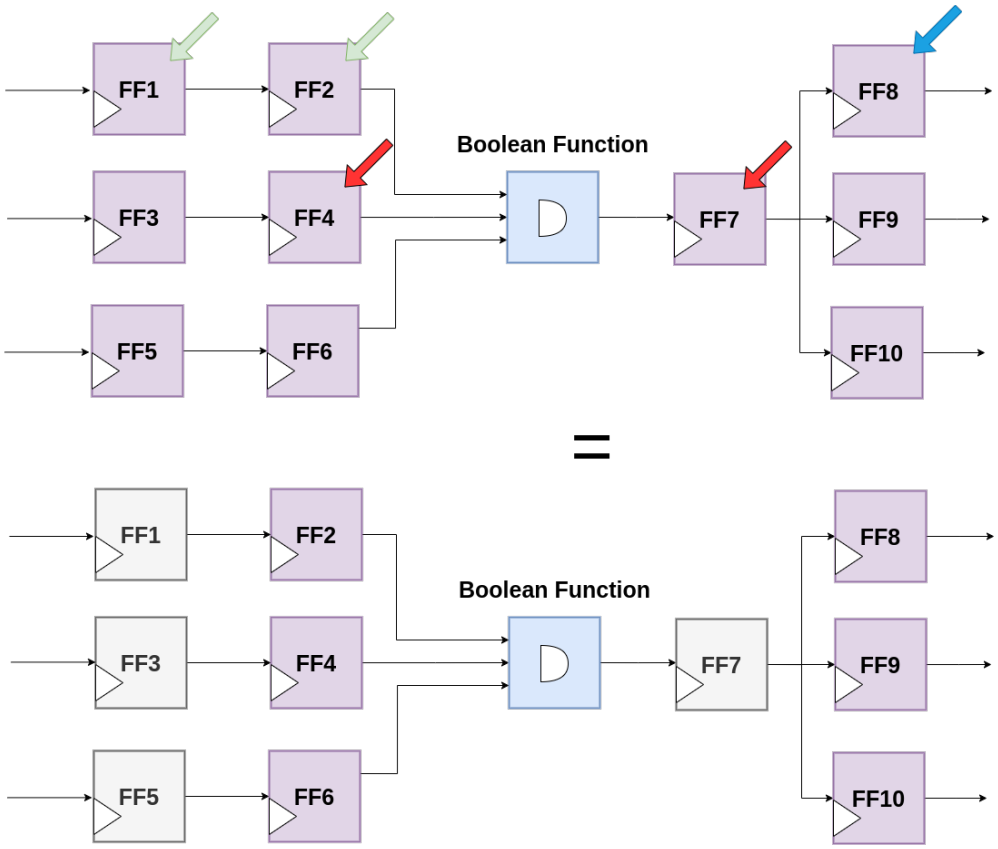


Figure 4.1.1: Fault dominance and equivalence (the Boolean function is equal to an AND gate)

```
ldr  r0, A4
ldr  r1, A5
ldr  r2, A6
and  r3, r0, r1
and  r4, r0, r2
and  r5, r1, r2
orr  r6, r3, r5
orr  r7, r4, r6
str  A7, r7
```

To better understand the relations between fault injection targets, Figure 4.1.4 illustrates the RTL description of the Assembly routine. Only registers r3, r4, r5 are retained for fault injection, as shown in Figure 4.1.4:

- r7 dominates r6: r7 can be removed.
- r6 dominates r3 and r5: r6 can be removed

Thus, 96 additional fault targets are identified in these registers (32 bits $\times$ 3 registers). In total, we have selected 288 FFs (384 FFs, in case of dual-core) for fault injection, as shown in Table 4.1.

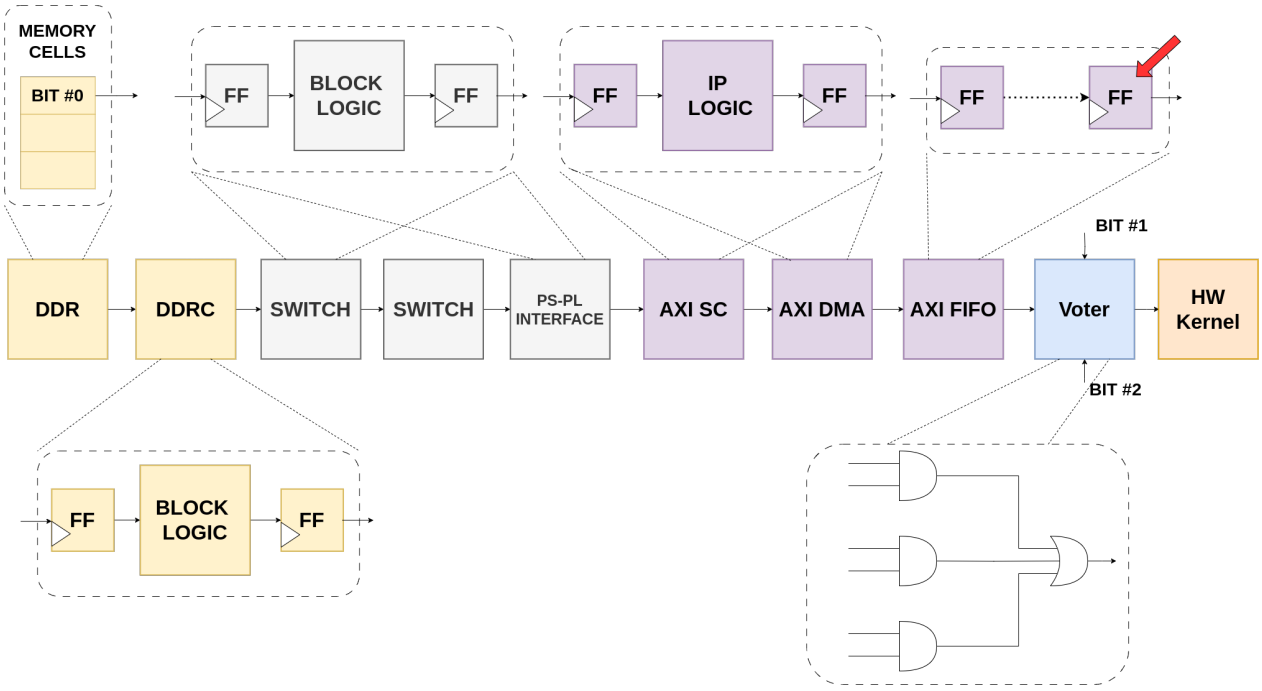


Figure 4.1.2: Single bit forward datapath (DDR-kernel) with highlighted fault injection target (red arrow)

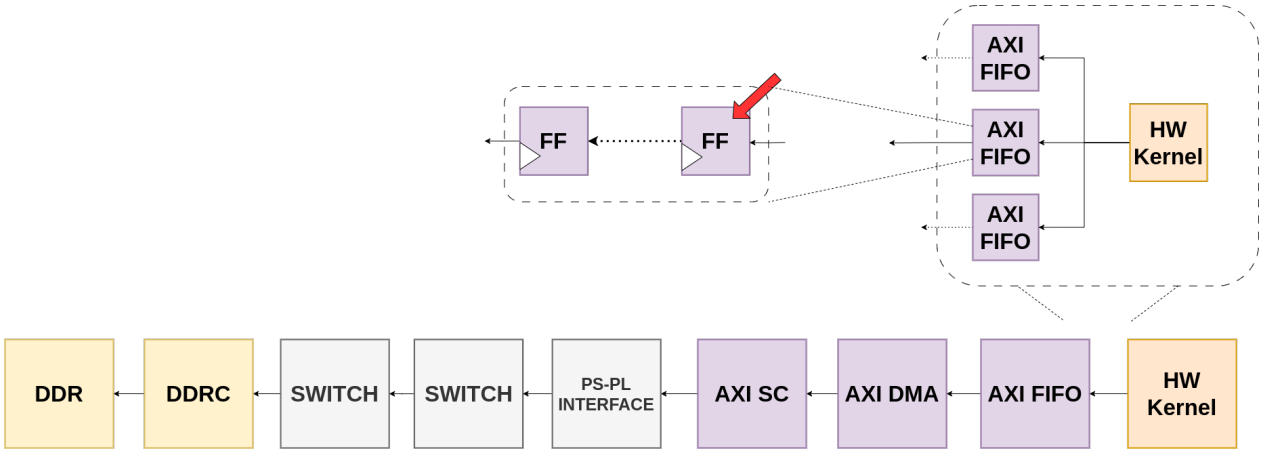


Figure 4.1.3: Single bit reverse datapath (kernel-DDR) with highlighted fault injection target (red arrow)

Location	**Flip-Flops (32-bit per replica)**	**Number of Replicas**	**Total Targets**
Forward Path	32	3	96
Reverse Path	32	3	96
Voting Registers	32	3 or 6	96 or 192
Total			**288 or 384**

Table 4.1: Summary of Fault Injection Targets

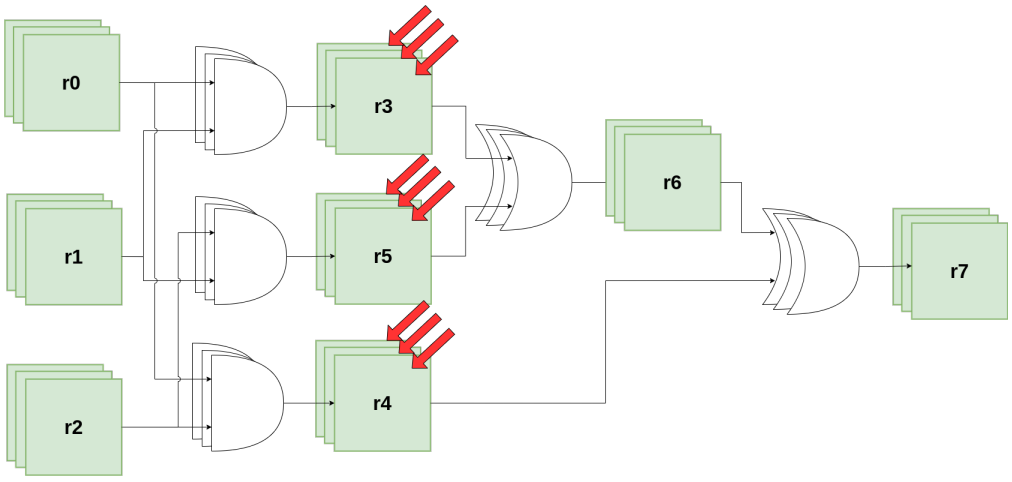
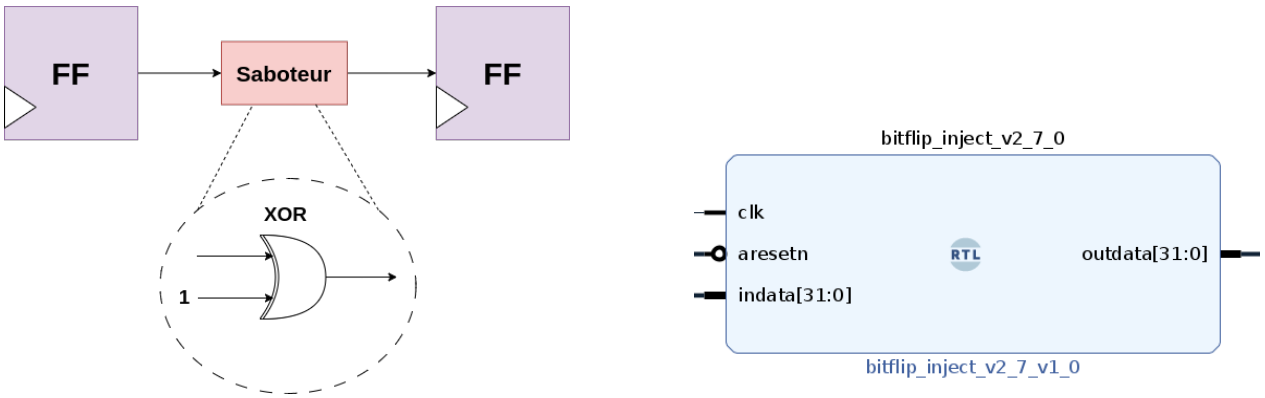


Figure 4.1.4: RTL representation of Assembly code (voting), with highlighted the fault injection targets after pruning (red arrows)

4.1.3 Step 2: Saboteur Insertion

A *saboteur* is a dedicated VHDL component integrated into the original model to introduce faults. Its role is to modify the value or timing characteristics of one or more signals upon fault injection. During normal system operation, the component remains inactive. To simulate a bitflip in a DFF during a clock cycle, the saboteur performs an XOR operation on the DFF output with a logic '1', effectively inverting its value. As illustrated in Figure 4.1.5a, the downstream DFF thus receives an inverted signal in the subsequent clock cycle.



(a) Bitflip Saboteur

(b) Custom IP: Bitflip Inject

Figure 4.1.5: Fault Injection Components

To control saboteur activation, a custom Vivado IP, referred to as *bitflip inject*, was developed (Figure 4.1.5b). This IP comprises 32 saboteur replicas, each mapped to a corresponding bit of a 32-bit input word. Accordingly, six instances of the IP are integrated into the system design. The IP remains in reset during normal execution and is released when fault injection commences. However, it does not support dynamic configuration to target specific bits. Instead, it relies on a predefined SER, determined by the system clock and the IP's internal structure, which is based on reseeded linear-feedback shift registers (LFSRs) for

pseudorandom fault injection.

An LFSR is a shift register where the input bit is generated from selected "tap" bits of the current state, typically combined through an XOR gate. In Fibonacci LFSRs, these taps define a feedback polynomial. For instance, selecting the 8^{th} and 5^{th} bits yields the polynomial $x^8 + x^5 + 1$. If this polynomial is *primitive*, the LFSR is *maximal-length*, cycling through all $2^n - 1$ non-zero states. The initial (non-zero) configuration of the register is termed the *seed*. Despite their deterministic nature, LFSRs can serve as pseudorandom number generators (PRNGs) when designed with sufficiently long periods appropriate for the application [36,37]. The sequence's randomness can be extended by reseeding, as described in [38].

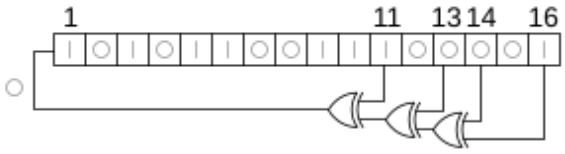


Figure 4.1.6: LFSR Representation [?]

Figure 4.1.7 presents the internal architecture of the bitflip inject IP. The LFSR initializes a down-counter, which activates the saboteur when it reaches zero. The LFSR simultaneously advances to its next state, increasing the counter by one. Once the counter reaches a value corresponding to the LFSR's full cycle length, a new seed is loaded into the LFSR, and the process repeats. The architecture ensures that the sequence of seeds lasts throughout the experiment and differs across all target flip-flops, ensuring statistical independence of bitflips. This design guarantees controlled fault injection at a defined SER. The SER for each bit is

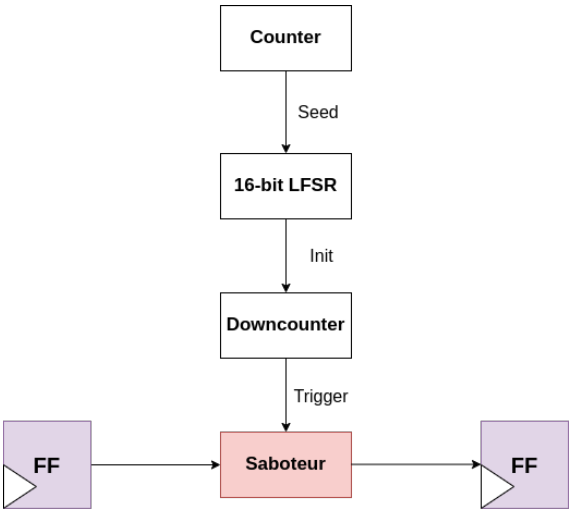


Figure 4.1.7: Internal Structure of Bitflip Inject IP

computed as follows:

In a complete LFSR cycle, the number of injected bitflips is $2^{16} - 1$, while the total number

of clock cycles is $(2^{16}-1)\cdot 2^{16}/2$. Hence:

$$SER = \frac{2^{16}-1}{(2^{16}-1)\cdot 2^{16}/2} = 2^{-15}\,\text{errors/bit/}T_{clock} \tag{4.1.2}$$

By adjusting the PL clock frequency, the effective SER can be tuned to the desired level.

For flip-flops within the Cortex-R5 register file, direct saboteur insertion is infeasible. Therefore, an emulation of the lockstep configuration is performed: in split mode configuration , both R5 cores run identical code independently. Each core performs a voting operation, illustrated in Figure 4.1.4, using shared memory locations to temporarily store intermediate values in the voting process. This setup allows external exposure of the registers to the element performing the bitflips, which in this case are two A53 cores, acting as saboteurs. The final A53 core performs the voting comparison between the outputs of the two R5 cores. It then signals a lockstep error to the Platform Management Unit (PMU) by writing to a designated global register, as depicted in Figure 4.1.8.

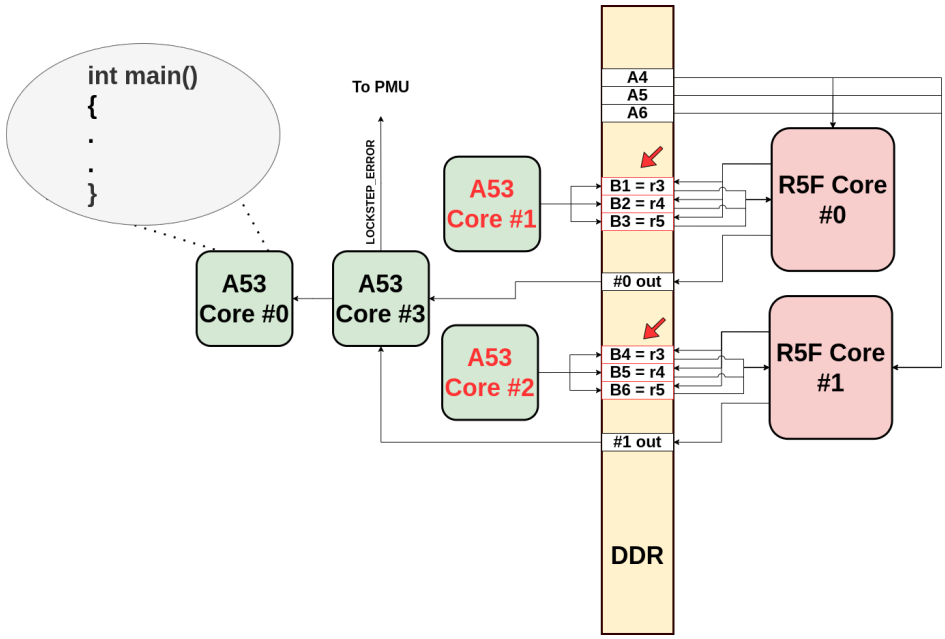


Figure 4.1.8: A53 Cores Acting as Saboteurs

4.2 Experimental Setup

We selected the Xilinx ZCU102 MPSoC development board as the implementation and evaluation platform for our study. This platform is based on the Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC, which belongs to the EG family of Xilinx's Zynq UltraScale+ devices. The PL subsystem of the device includes the resources summarized in Table 4.2:

To compute the Reliability and the MTTF of the proposed architecture, as discussed in Section 2.2.3, it is necessary to estimate the PMF of the time-to-failure distribution. This can be achieved either through analytical modeling or via empirical measurement through experimentation. In this thesis, we adopt the experimental approach. The complete experimental setup is illustrated in Figure 4.2.1.

Resource	Quantity
System Logic Cells	599,550
CLB Flip-Flops	548,160
CLB LUTs	274,080
Distributed RAM (Mb)	8.8
Block RAM (Mb)	8.1
DSP Slices	2,520
Maximum I/O Pins	328

Table 4.2: FPGA resources available on the Zynq UltraScale+ XCZU9EG device.

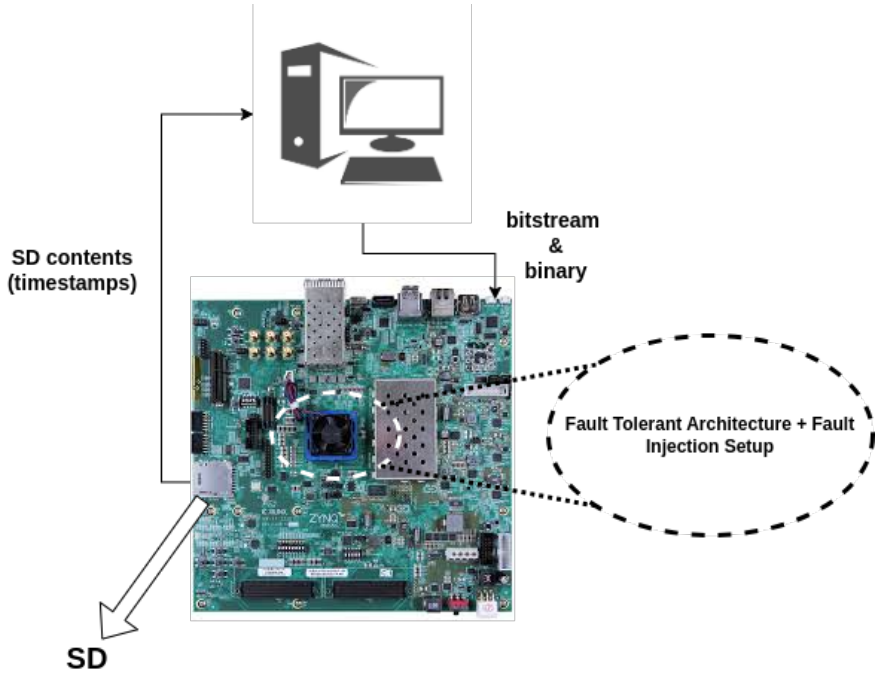


Figure 4.2.1: Experimental setup used for reliability evaluation.

Each experimental run consists of the following steps:

1. The main application offloads data to the PL.
2. The hardware kernel processes the data and returns the results to the PS.
3. The output data are evaluated using either standard voting (single-core) or lockstep voting (dual-core) mechanisms.
4. The resulting output is compared against a predefined golden output to detect discrepancies. In the event of a mismatch, the PS records the timestamp of detection (relative to the start of the experiment) onto an SD card and initiates a new run. Otherwise, the offloading process is repeated without resetting the experimental run.

It is important to note that a saboteur is activated at the beginning of the first experiment and remains enabled throughout all subsequent runs. Additionally, the timing measurements reflect only the effective processing time, excluding delays associated with simulating lockstep

operation (execution overhead from lockstep operation is measured separately) and data storage. Specifically, the overhead from accessing and writing to the SD card is omitted.

Table 4.3 shows the testing parameters, with which we run our experiments in the development board.

Parameter	**Value**
Clock Frequency (PL)	100 MHz
Clock Frequency (R5)	100 MHz
Clock Frequency (A53)	1.2 GHz
Soft Error Rate (SER)	3000 errors/bit/second
Data Size	10 KB
Number of Experimental Runs	50000

Table 4.3: Experimental configuration and fault model parameters.

The selection of the above specifications was made with the goal of optimizing the accuracy of the PMF estimation, minimizing the total experimental execution time, and ensuring correct operation during simulation.

Regarding the Soft Error Rate (SER), it is crucial to strike an appropriate balance. If the SER is too low, approaching the naturally occurring rates in real space environments, faults that lead to failures would occur too infrequently, significantly increasing the time required to complete all experimental runs. On the other hand, if the SER is too high, the system would become overwhelmed by faults, preventing the fault-tolerant architecture from effectively demonstrating its mitigation capabilities.

With respect to the operating frequencies of the A53 and R5f cores, the A53 cores are configured to run at a significantly higher frequency than the R5f. This ensures that, from the R5f's point of view, fault injection into the shared memory appears instantaneous, simulating the way SEUs occur in real-world scenarios.

The choice of 50,000 experimental runs provides a good statistical basis for estimating the PMF in our simulation. While increasing this number would improve the accuracy of the PMF, it would also lead to a proportional increase in total experimental execution time.

Finally, the data size should be treated as a user-defined parameter, as the proposed architecture is intended to serve as a generic framework. For the purpose of this study, we arbitrarily select a value of 10KB, and based on the experimental results, we analyze how varying the data size affects the reliability.

4.3 Experimental Results

As discussed in earlier sections, we evaluate the proposed fault-tolerant architecture by considering two types of hardware kernels implemented in the PL: an *accumulator* and an *offset adder*. Figure 4.3.1 illustrate the reliability of both systems in the absence of any fault-tolerant mechanisms.

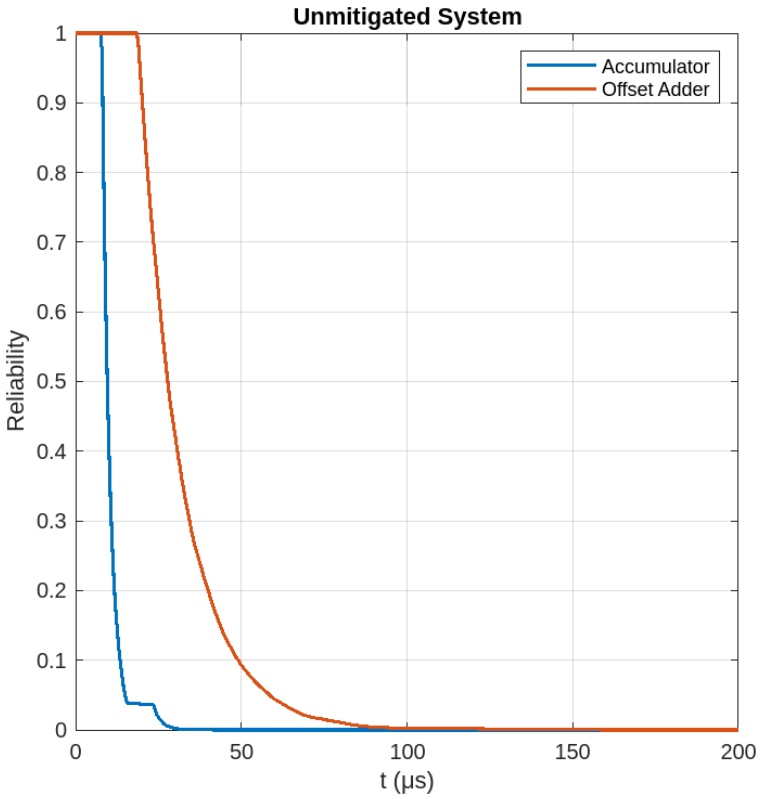


Figure 4.3.1: Unmitigated System

Based on the figure, several observations can be made. The calculated MTTF for the accumulator-based system is 10.589 μs, while the offset adder-based system exhibits an MTTF of 31.987 μs. Ideally, the MTTF should correspond to the steepest point of the reliability curve, indicating the most probable failure region. However, due to the non-Gaussian nature of the experimental PMF derived from our fault injection campaign, this condition does not hold in our case.

Furthermore, the system utilizing the accumulator kernel exhibits earlier failure under fault conditions. The accumulator kernel inherently accumulates multiple values, consequently aggregating the probability of fault occurrence across multiple input values. Conversely, the offset adder kernel processes individual numbers independently, significantly reducing the likelihood that a single faulty input will propagate through to an erroneous system state. On the other hand, the offset adder kernel demands the transfer of larger data volumes between the PS and the PL, given that no internal data reduction occurs. In contrast, the accumulator kernel performs internal data reduction, substantially decreasing the output data size relative to the input data (e.g. ten input numbers into a single output value). Thus, the offset adder kernel incurs increased data movement, which inherently exposes the system to more extensive fault-prone interactions across communication interfaces. This exposure amplifies the probability of fault occurrence in data transfer, particularly in the reverse data path (from PL to PS). Analyzing these factors together, it becomes apparent that the accumulator kernel's susceptibility predominantly arises from faults in the forward data path (input accumulation), whereas the offset adder kernel is predominantly sensitive to faults in the reverse data path (output data transfer). Although determining the dominant effect

analytically without empirical evidence is challenging, in this case, the accumulator kernel's vulnerability to forward path faults surpasses the offset adder's vulnerability to reverse path faults.

Additionally, in the case of the accumulator-based system, the reliability curve flattens between 15 μs and 27 μs. This phenomenon is a consequence of the experimental setup. During each experimental run, failures can only be detected at specific time intervals, namely, when the software compares the actual output with the golden output. In the intermediate time intervals, when data is being transmitted to the PL, processed, and returned, no output verification occurs, and thus, no failures can be observed. Although such flat segments in the reliability curve occur in all configurations, they become visible only in this specific configuration because the time to failure is particularly short and comparable to the duration of these verification gaps. As a result, such behavior is observable primarily in configurations that tend to fail shortly after the beginning of the experiment.

TMR (spatial and temporal) addition in the architecture

Figure 4.3.2 show the impact of introducing TMR in both the forward and return paths, while retaining standard voting mechanisms in the PS. Compared to the baseline configuration, the MTTF improved by an average factor of **120**. In particular, the MTTF reached 2.29 ms for the accumulator-based system and 0.96 ms for the offset adder.

Despite the overall reliability improvement, the offset adder system exhibits a shorter MTTF compared to the accumulator-based system. This is due to the influence of the implemented TMR scheme, which has shifted the most probable point of failure from the communication interface to the PS-level voting mechanism. The accumulator system benefits more from this change, as it involves less data passing through the voting stage, resulting in a greater improvement in reliability.

Lockstep PS Voting addition in the architecture

Lastly, we incorporate lockstep voting within the PS to further enhance system reliability. Figure 4.3.3 depict the resulting reliability curves when the complete fault-tolerant architecture is applied. Compared to the baseline configuration, MTTF increases by an average factor of **9290**. The MTTF reaches 110.9 ms for the accumulator system and 259.5 ms for the offset adder system.

The introduction of lockstep PS voting has shifted the most probable point of failure from the PS voting mechanism back to the communication interface. Based on the characteristics of the unmitigated system, this shift inherently favors the offset adder kernel. The MTTF values for all configurations are summarized in Table 4.4, while Figures 4.3.4 and 4.3.5 illustrate the comparative effects of the applied fault-tolerant techniques.

Resource Utilization and Execution Overhead

While the dependability metrics are very important in analysizing fault-tolerant systems, it is equally important to evaluate the system in terms of resources utilization and execution overhead. Table 4.5 presents the resource utilization in terms of LUTs, FFs and BRAM for key

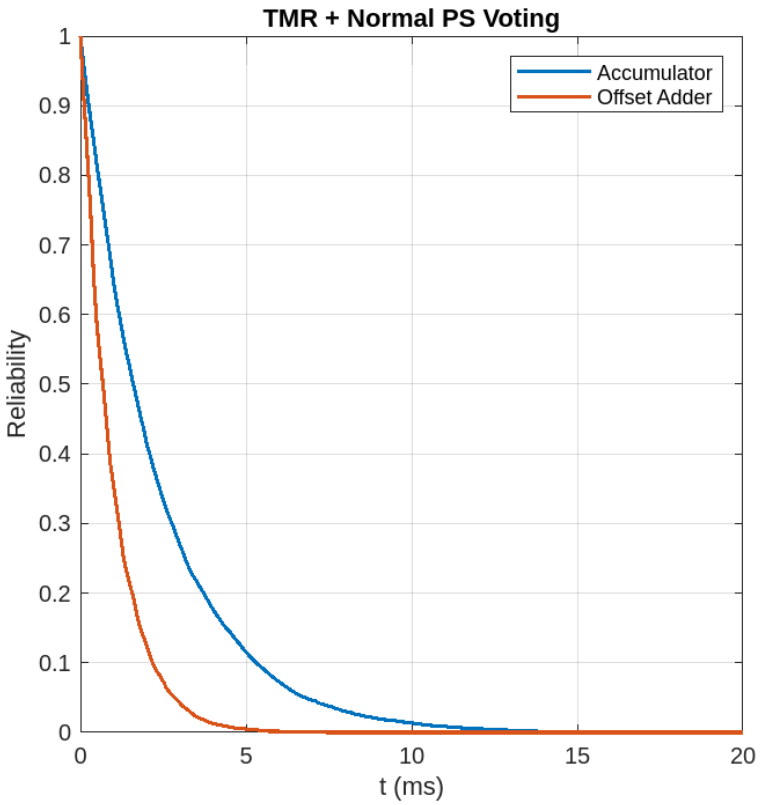


Figure 4.3.2: TMR + Normal PS Voting

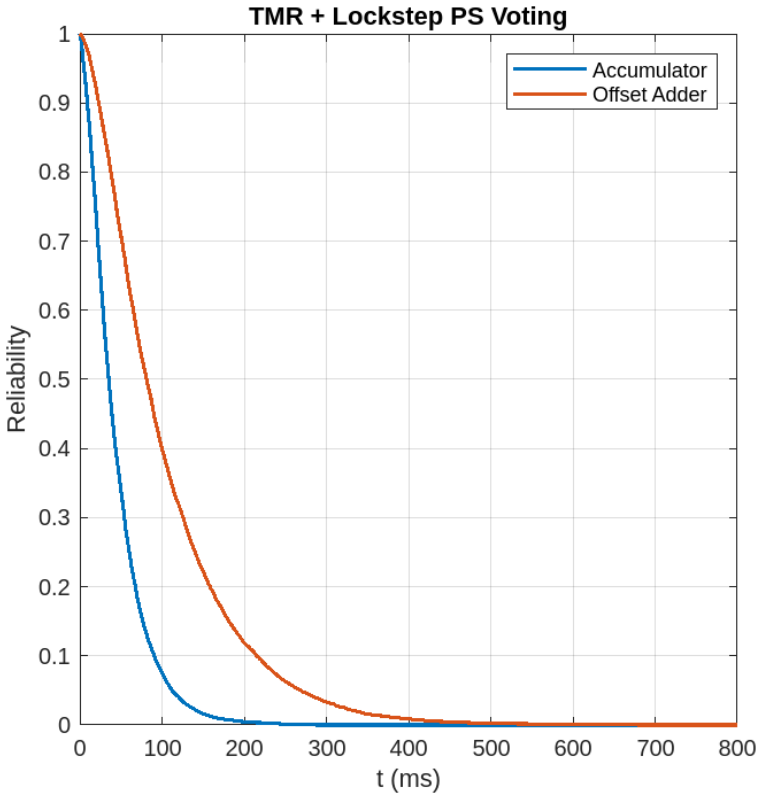


Figure 4.3.3: TMR + Lockstep PS Voting

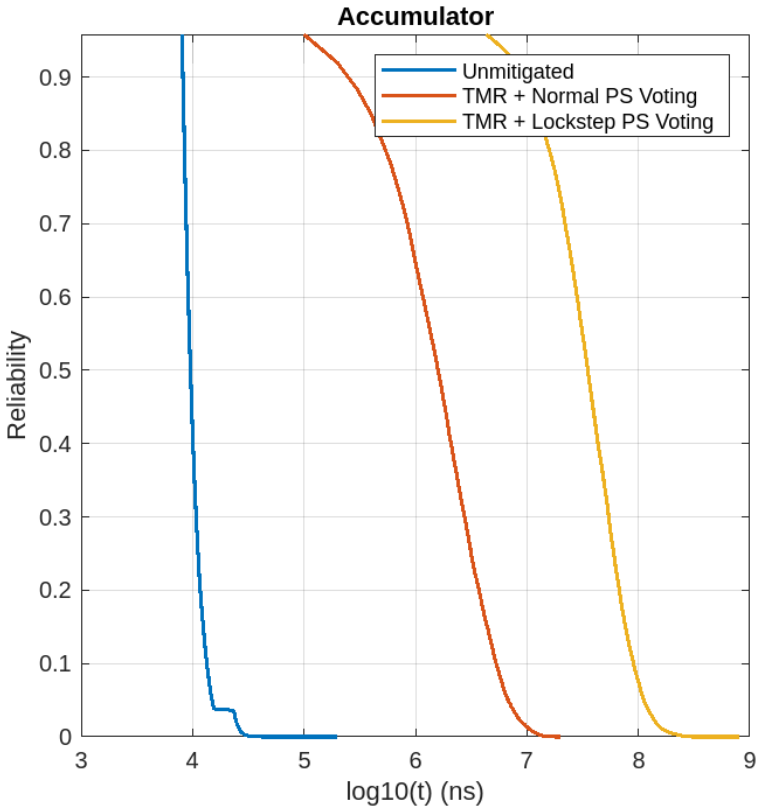


Figure 4.3.4: Three configurations compared

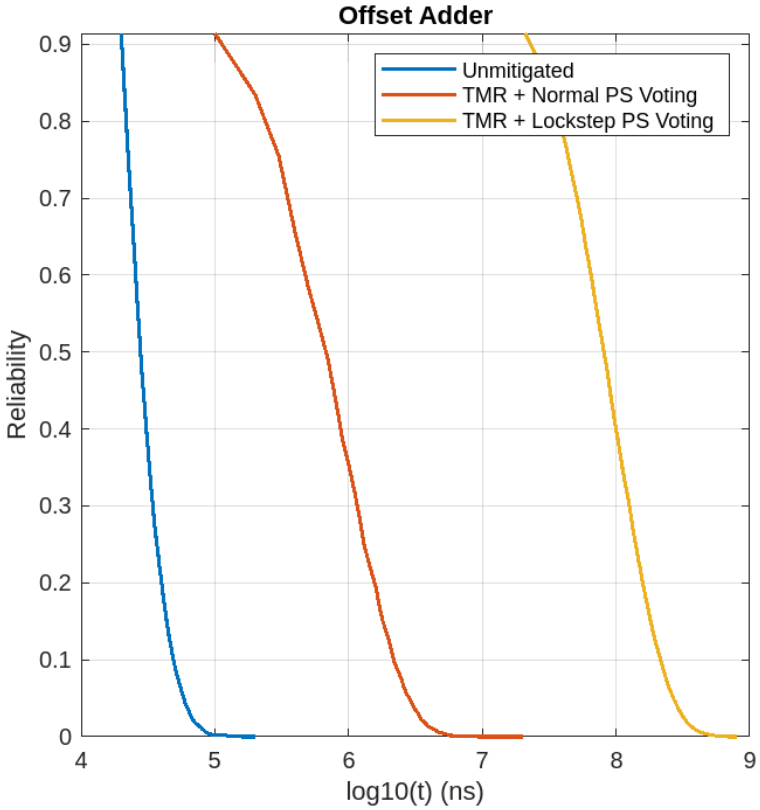


Figure 4.3.5: Three configurations compared

data movement and interconnect modules, specifically, the AXI DMA, AXI SMC, and AXI Data FIFO, across three different configurations: No Fault Tolerance, TMR Only, and TMR with PS Lockstep Voting. In all fault-tolerant configurations, the number of instantiated modules increases accordingly, leading to a rise in resource consumption.

It can be observed that no configuration utilizes more than **3%** of the device's total capacity for any specific resource. This indicates that the implementation is not resource-intensive and leaves ample space for the end user to develop additional hardware kernels within the remaining PL. Furthermore, the number of resources has increased threefold, as expected with the application of TMR schemes.

Regarding execution overhead, there are two primary contributing factors: temporal redundancy and the PS voting mechanism. In the first case, the overhead introduced by temporal redundancy depends on the time required for the processor to initiate the data transfer, specifically, the time taken to write to a designated register. Experimental measurements have shown that this process takes approximately 100 processor cycles, which corresponds to the execution time of a simple software routine. Given that this duration is negligible compared to the overhead introduced by PS voting, it is excluded from the total execution overhead calculation. In the second case, the overhead due to PS voting is a function of both the data size and the operating frequency of the R5 processor. In Figure 4.3.6, the execution overhead is shown in R5 cycles at an operating frequency of 100MHz, which is the frequency used in our experiments. A linear relationship is observed between the execution overhead and the data size. This linearity is a result of the deterministic nature of the voting process, which we enforce by optimizing cache utilization during the operation.

It is important to note that the voting algorithm is not configurable by the end user. Users cannot modify or interfere with the way data is fetched, voted upon, or written back to memory. It should also be noted that the reported execution overhead corresponds to a single voting operation over the entire data size. In the event of a lockstep error, the execution overhead may increase, depending on the recovery mechanism selected by the user. For instance, in our implementation, we choose to repeat the voting process upon detection of a mismatch, which results in additional overhead. Finally, the execution overhead remains unchanged regardless of whether single or lockstep voting is employed. This is a significant advantage, as it allows us to enhance system reliability without incurring additional execution overhead.

Kernel	**Configuration**	**MTTF**
Accumulator	No Fault Tolerance	10.597 μs
Accumulator	TMR Only	2.29 ms
Accumulator	TMR + PS Lockstep Voting	110.9 ms
Offset Adder	No Fault Tolerance	31.987 μs
Offset Adder	TMR Only	0.96 ms
Offset Adder	TMR + PS Lockstep Voting	259.5 ms

Table 4.4: Summary of MTTF Values for Different Configurations

Configuration	Module	LUTs	FFs	BRAM
Unmitigated	AXI DMA (2)	898	1400	0
	AXI SMC (1)	1691	2509	0
	AXI DATA FIFO (2)	160	130	8
TMR + Normal PS Voting	AXI DMA (6)	2694	4200	0
	AXI SMC (3)	5061	7527	0
	AXI DATA FIFO (6)	480	390	24
TMR + Lockstep PS Voting	AXI DMA (6)	2694	4200	0
	AXI SMC (3)	5061	7527	0
	AXI DATA FIFO (6)	480	390	24

Table 4.5: Resource Utilization (the number in parentheses represent the number of instances regarding each IP

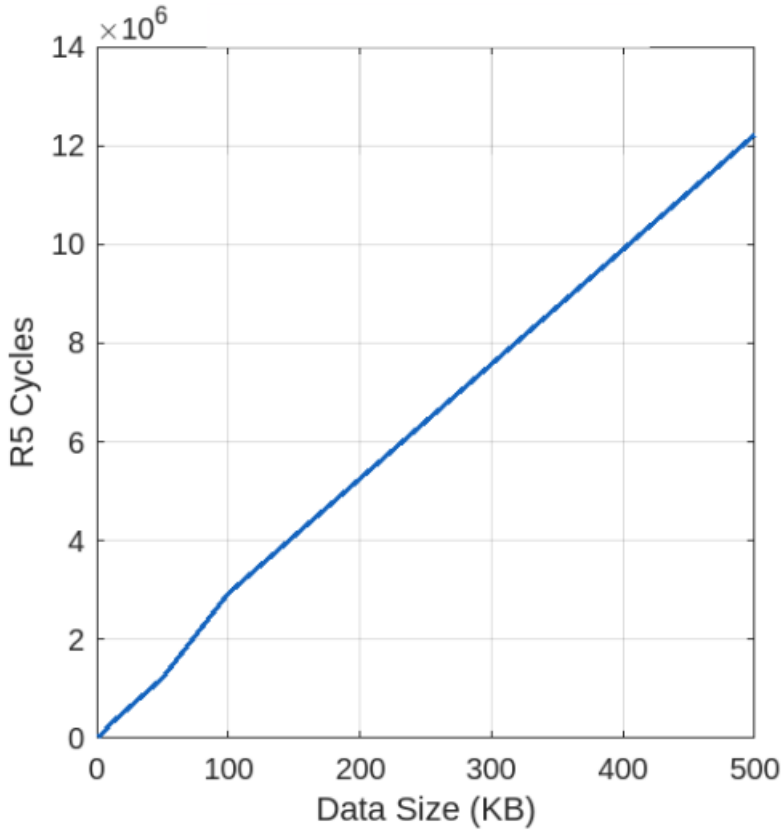


Figure 4.3.6: Execution Overhead

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis presented a comprehensive approach to achieving fault-tolerant data transfers COTS SoC FPGAs deployed in radiation-prone environments, with a particular focus on space applications. The proposed methodology leverages standard Xilinx IP cores, ensuring seamless integration and full compatibility with existing Xilinx design tools. Proof-of-concept designs were implemented on an MPSoC Ultrascale+ ZU102, featuring a quad-core Arm Cortex-A53 application processor and a dual-core Arm Cortex-R5F real-time processor. Importantly, all techniques introduced in this thesis are easily replicable on any board with similar specifications, using the same standard components and IP libraries provided by Xilinx. These resources are widely available and relatively low-cost compared to alternative solutions, enhancing the scalability and practicality of the proposed methodology. The fault-tolerant techniques developed in this work include spatial and temporal redundancy, as well as dual-core lockstep, targeting both the PL and PS subsystems. Two combinations of these techniques were evaluated in terms of reliability, MTTF and execution overhead, and compared against an unmitigated baseline design. A custom fault injection campaign was designed to systematically assess the effectiveness of the proposed fault-tolerant techniques. Experimental results demonstrated significant improvements in error resilience, validating the suitability of the architecture for safety-critical space missions using cost-effective COTS FPGA platforms.

5.2 Future Work

Although the architecture presented achieves substantial improvements in fault tolerance, several areas warrant further investigation:

- **Sampling-Based PS Voting**: Instead of performing voting over the entire data payload, a sampling-based approach could be explored, where only a subset of the data is checked for discrepancies. This method could significantly reduce the execution overhead introduced by the voting process.
- **Integration with Fault-Tolerant Processing Elements**: The current approach

focuses on securing data transfers (i.e., the communication interfaces) rather than the processing elements themselves, such as the main application or hardware kernels. Integrating the proposed techniques with fault-tolerant processing components could further enhance the system's overall reliability.

- **Real-Time Adaptive Fault Tolerance**: Investigating dynamic reconfiguration techniques may enable systems to adjust their fault-tolerance mechanisms in real time based on environmental conditions, such as measured radiation levels. This would allow for better optimization of both performance and resource usage.
- **Software Support and Abstraction**: Developing a user-friendly abstraction layer, including APIs and support libraries, could simplify the deployment of the proposed architecture. This would facilitate adoption by end users and enable easier integration into diverse applications.

Bibliography

[1] G. Brunetti, G. Campiti, M. Tagliente, and C. Ciminelli, "Cots devices for space missions in leo," *IEEE Access*, vol. 12, pp. 76478–76514, 2024.

[2] H. Bokil, "Cots semiconductor components for the new space industry," in *2020 4th IEEE Electron Devices Technology Manufacturing Conference (EDTM)*, pp. 1–4, 2020.

[3] A. Pérez, A. Rodríguez, A. Otero, D. G. Arjona, Jiménez-Peralo, M. 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.

[4] 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.

[5] V. Leon, I. Stamoulias, G. Lentaris, D. Soudris, D. Gonzalez-Arjona, R. Domingo, D. Merodio 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, 09 2021.

[6] X. Iturbe, D. Keymeulen, E. Özer, P. Yiu, D. Berisford, K. Hand, and R. Carlson, "An integrated soc for science data processing in next-generation space flight instruments avionics," pp. 134–141, 10 2015.

[7] C. De Sio, S. Azimi, and L. Sterpone, *On the Evaluation of SEU Effects on AXI Interconnect Within AP-SoCs*, pp. 215–227. 07 2020.

[8] F. Benevenuti and F. L. Kastensmidt, "Reliability evaluation on interfacing with axi and axi-s on xilinx zynq-7000 ap-soc," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pp. 1–6, 2018.

[9] F. Benevenuti and F. Kastensmidt, *Analyzing AXI Streaming Interface for Hardware Acceleration in AP-SoC Under Soft Errors*, pp. 243–254. 01 2018.

[10] H. Cho, C.-Y. Cher, T. Shepherd, and S. Mitra, "Understanding soft errors in uncore components," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[11] J. Lázaro, A. Astarloa, A. Zuloaga, J. Araujo, and J. Jiménez, "Axi lite redundant on-chip bus interconnect for high reliability systems," *IEEE Transactions on Reliability*, vol. 73, no. 1, pp. 602–607, 2024.

[12] A. C. R. Alves, L. F. Q. Silveira, M. E. Kreutz, and S. M. Dias, "A parity-based dual modular redundancy approach for the reliability of data transmission in nanosatellite's onboard processing," *IEEE Access*, vol. 12, pp. 90815–90828, 2024.

[13] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 818–831, 2005.

[14] J. Mach, L. Kohutka, and P. Cicak, "On-chip bus protection against soft errors," *Electronics*, vol. 12, 11 2023.

[15] Nasa, "Why space radiation matters." , 2017.

[16] K. LaBel, M. Gates, A. Moran, P. Marshall, J. Barth, E. Stassinopoulos, C. Seidleck, and C. Dale, "Commercial microelectronics technologies for applications in the satellite radiation environment," in *1996 IEEE Aerospace Applications Conference. Proceedings*, vol. 1, pp. 375–390 vol.1, 1996.

[17] R. Maurer, M. Fraeman, M. Martin, and D. Roth, "Harsh environments: Space radiation environment, effects, and mitigation," vol. 28, 01 2008.

[18] AMD, "Single-event upset mitigation selection guide application note," Tech. Rep. XAPP987, AMD, March 2008. Revision 1.0.

[19] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*. Reading, Massachusetts: Addison-Wesley, 1989.

[20] A. Benso and P. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. 01 2003.

[21] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Boston, MA: Springer, 2004.

[22] Xilinx Inc., *Zynq™ UltraScale+™ MPSoC Data Sheet: Overview (DS891)*. Xilinx, 2025-03-18. Document DS891, Version 1.11.1.

[23] Xilinx Inc., *Zynq UltraScale+ Device Technical Reference Manual (UG1085)*. Xilinx, 2025-03-21. Document UG1085, Version 2.5.

[24] ARM Ltd., *AMBA 4 AXI4-Stream Protocol Specification Version 1.0*, 2010.

[25] P. Balasubramanian, K. Prasad, and N. E. Mastorakis, "A fault tolerance improved majority voter for TMR system architectures," *CoRR*, vol. abs/1605.03771, 2016.

[26] N. F. Haddad, R. D. Brown, R. Ferguson, A. T. Kelly, R. K. Lawrence, D. M. Pirkl, and J. C. Rodgers, "Second generation (200mhz) rad750 microprocessor radiation evaluation," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*, pp. 877–880, 2011.

[27] X. Iturbe, B. Venu, and E. Ozer, "Soft error vulnerability assessment of the real-time safety-related arm cortex-r5 cpu," in *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 91–96, 2016.

[28] X. Iturbe, B. Venu, E. Özer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek, "The arm triple core lock-step (tcls) processor," *ACM Transactions on Computer Systems*, vol. 36, pp. 1–30, 06 2019.

[29] M. Barbirotta, F. Menichelli, A. Cheikh, A. Mastrandrea, M. Angioli, and M. Olivieri, "Dynamic triple modular redundancy in interleaved hardware threads: An alternative solution to lockstep multi-cores for fault-tolerant systems," *IEEE Access*, vol. PP, pp. 1–1, 01 2024.

[30] J. Mach, L. Kohútka, and P. Čičák, "In-pipeline processor protection against soft errors," *Journal of Low Power Electronics and Applications*, vol. 13, no. 2, 2023.

[31] G. Cardarilli, F. Kaddour, A. Leandri, M. Ottavi, S. Pontarelli, and R. Velazco, "Bit flip injection in processor-based architectures: a case study," in *Proceedings of the Eighth IEEE International On-Line Testing Workshop (IOLTW 2002)*, pp. 117–127, 2002.

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

[33] G. L. Nazar and L. Carro, "Fast single-fpga fault injection platform," in *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 152–157, 2012.

[34] M. Stojcev, "Alfredo benso, paolo prinetto, editors, fault injection techniques and tools for embedded systems reliability and evaluation, kluwer academic publishers, boston, 2003. hardcover, pp 241, plus xiv, isbn 1-4020-7589-8.," *Microelectronics Reliability*, vol. 46, pp. 1396–1397, 08 2006.

[35] L. Berrojo, I. Gonzalez, F. Corno, M. Reorda, G. Squillero, L. Entrena, and C. Lopez, "New techniques for speeding-up fault-injection campaigns," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, pp. 847–852, 2002.

[36] R. S. Durga, C. K. Rashmika, O. N. V. Madhumitha, D. G. Suvetha, B. Tanmai, and N. Mohankumar, "Design and synthesis of lfsr based random number generator," in *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 438–442, 2020.

[37] "Application notes, efficient shift registers, lfsr counters, and long pseudo-random sequence generators (xapp052)."

[38] P. S. Dilip, G. R. Somanathan, and R. Bhakthavatchalu, "Reseeding lfsr for test pattern generation," in *2019 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0921–0925, 2019.