POLITECNICO
DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Progettazione e sviluppo di un'architettura tollerante alle radiazioni basata su FPGA riconfigurabile dinamicamente

per applicazioni in ambito spazio

Relatore
prof. Maurizio Zamboni

Candidato
Nicola Tisat

Aprile 2018

Abstract

Questa tesi descrive lo sviluppo di una metodologia in grado di aumentare l'affidabilità di dispositivi elettronici commerciali esposti alle radiazioni dello spazio. L'obiettivo di questo lavoro è implementare una soluzione che permetta di ridurre i costi dei sistemi di elaborazione impiegati nei satelliti, aumentandone allo stesso tempo le prestazioni computazionali. Il sistema fa uso della ridondanza hardware e della riconfigurabilità parziale dinamica per individuare, mascherare e correggere gli errori causati dalle radiazioni. Per l'implementazione si è fatto uso di un System On Chip Zynq XC7Z020 prodotto da Xilinx, composto da un processore e una parte di logica programmabile. Infine, è stato eseguito un test di validazione utilizzando una tecnica di fault injection, dimostrando la capacità di auto-riparazione dell'architettura implementata.

Sommario

Le future missioni di esplorazione nello spazio profondo utilizzeranno strumenti scientifici in grado di generare grandi quantità di dati. Per poterle elaborare, i computer di bordo richiederanno elevate potenze computazionali. Il maggior limite all'aumento delle performance di questi dispositivi è rappresentato dall'ambiente estremo in cui essi devono lavorare. Infatti, lo spazio interplanetario è continuamente attraversato da particelle cariche ad alta energia, principalmente generate dal vento solare e dai raggi cosmici. Queste possono colpire e distruggere i circuiti elettronici di bordo, oppure introdurre errori nelle elaborazioni. Il pericolo è presente anche per i satelliti in orbita all'interno dello scudo naturale offerto dalla magnetosfera terrestre, dove le fasce di Val Allen intrappolano queste particelle cariche, creando delle zone altamente radiative.
I componenti elettronici devono quindi essere protetti utilizzando particolari tecniche di *radiation hardening* (chiamate radhard techniques). Esse prevedono di schermare i componenti con dei fogli metallici (*shielding*), utilizzare speciali tecnologie di fabbricazione degli Integrated Circuit (IC), oppure impiegare il concetto di ridondanza e aumentare il numero di risorse computazionali disponibili in modo da poter tollerare eventuali malfunzionamenti. Queste soluzioni comportano però un aumento del peso e una riduzione dello spazio disponibile (nel caso dello shielding e della ridondanza) oppure una riduzione della densità dei transistor implementati negli integrati e quindi una diminuzione delle funzionalità e performance del dispositivo (nel caso delle tecnologie di fabbricazione degli IC). Inoltre questi dispositivi presentano un alto costo, in forte contrasto ai componenti Commercial Off-The-Shelf (COTS), i quali permettono alte performance e ridotto ingombro e peso ad un prezzo inferiore di anche tre ordini di grandezza. I componenti commerciali non sono progettati per resistere alle radiazioni presenti nello spazio, ma le loro prestazioni e caratteristiche risultano essere molto interessanti per le future missioni d'esplorazione.

L'obiettivo principale di questa tesi è lo sviluppo di una soluzione alternativa al problema delle radiazioni. La soluzione individuata deve permettere di ridurre il costo dei sistemi di elaborazione elettronici impiegati a bordo delle piattaforme satellitari. Inoltre, per anticipare le richieste delle future missioni spaziali, il sistema sviluppato deve essere dotato di un'elevata potenza computazionale. Questi traguardi possono essere raggiunti impiegando componenti COTS. Per poter utilizzare dispositivi commerciali su satelliti in orbita è necessario individuare delle tecniche che li rendano tolleranti alle radiazioni. La soluzione sviluppata utilizza la capacità di alcune Field Programmable Gate Array (FPGA) di essere riconfigurate parzialmente e dinamicamente. Questa tecnologia permette di intervenire in zone precise del IC, modificando solo alcune delle funzionalità implementate nel dispositivo, mentre il restante circuito può continuare ad elaborare dati senza interruzioni. Questo, in unione alla ridondanza hardware, permette di rivelare e correggere autonomamente gli effetti delle radiazioni, senza che il sistema subisca alcun arresto delle funzionalità oppure generi risultati errati. Lo scopo di questa tesi è quindi di sviluppare ed implementare il prototipo di un'architettura in grado di aumentare l'affidabilità di un sistema elettronico realizzato con componenti commerciali, sfruttando la tecnologia della riconfigurabilità parziale dinamica.

Questa tesi è organizzata come segue.
Il **Capitolo 1** introduce l'azienda nella quale questa ricerca è stata svolta, Argotec. Vengono quindi descritti i dispositivi FPGA e il concetto di riconfigurabilità e infine viene presentata una panoramica sulle radiazioni presenti nello spazio e gli effetti che queste causano ai circuiti elettronici.
Il **Capitolo 2** descrive le possibili soluzioni applicabili al problema delle radiazioni, soffermandosi in particolare sulla ridondanza Hardware (HW). Viene inoltre presentato lo stato dell'arte riguardo alla riconfigurabilità applicata nel campo aerospaziale.
Il **Capitolo 3** introduce gli obiettivi che la tesi si pone e dai quali viene elaborata una soluzione. Il sistema sviluppato e la relativa architettura vengono quindi descritti.
Il **Capitolo 4** illustra le risorse hardware e software utilizzate per implementare il sistema studiato. Vengono inoltre descritti i dettagli implementativi dell'architettura, sia dal lato HW sia da quello Software (SW).
Il **Capitolo 5** presenta i test effettuati sull'architettura, in particolare i test funzionali eseguiti durante la fase di implementazione e la tecnica di fault injection utilizzata per validare il sistema.
Infine, il **Capitolo 6** raccoglie le conclusioni e gli sviluppi futuri del sistema descritto in questa tesi.

Ringraziamenti

Vorrei cogliere l'occasione per ringraziare tutte le persone che, con il loro aiuto, mi hanno permesso di raggiungere questo importante obiettivo.
Innanzitutto ringrazio David Avino, Managing Director di Argotec, per avermi permesso di sviluppare questo interessante progetto. I miei ringraziamenti vanno anche al mio tutor aziendale Eugenio Scarpa, per avermi guidato durante questo percorso ed aver speso tempo e pazienza per permettermi di concluderlo. Vorrei quindi ringraziare tutti i colleghi di Argotec, in particolare Emilio e Claudio: in questi mesi ho imparato molto e ciò mi ha permesso di crescere sia professionalmente che come persona.
Un grazie va ai ragazzi di Feltre, sempre vicini anche se lontani, e agli amici di Torino con i quali ho condiviso le gioie e i dolori della vita universitaria da studente fuori sede. Inoltre vorrei ringraziare Elena per essermi stata vicina nei momenti più difficili di questi anni.
Infine, un grazie speciale va ai miei genitori e a mia sorella per avermi sempre sostenuto ed aver creduto in me anche quando pensavo di non farcela, accompagnandomi in questo lungo viaggio.

Indice

Elenco delle tabelle

Elenco delle figure

Elenco degli acronimi

ACE Advanced eXtensible Interface (AXI) Coherency Extensions

AMBA Advanced Microcontroller Bus Architecture

APU Application Processing Unit

ARTE Advanced Research Thermal Passive Exchange

AXI Advanced eXtensible Interface

CLB Configurable Logic Block

CNSA China National Space Administration

COTS Commercial Off-The-Shelf

CPU Central Processing Unit

DMA Direct Memory Access

DSP Digital Signal Processor

DWC Duplication With Comparison

ECC Error-Correcting Codes

FPGA Field Programmable Gate Array

GNC Guidance Navigation and Control

HLS High Level Syntesis

HW Hardware

IC Integrated Circuit

ICAP Internal Configuration Access Port

IDE Integrated Development Environment

IP Intellectual Property

ISS International Space Station

KB Kilobyte

LEO low Earth orbit

LFA Linear Frame Address

LUT Look-Up Table

MB Megabyte

MOS Metal-Oxide Semiconductor

MOSFET Metal-Oxide Semiconductor Field Effect Transistor

NASA National Aeronautics and Space Administration

OBC On-Board Computer

PCAP Processor Configuration Access Port

PL Programmable Logic

PS Processing System

RAM Random Access Memory

RM Reconfigurable Module

RP Reconfigurable Partition

SDK Software Development Kit

SEE Single Event Effects

SEL Single Event Latchup

SEM Soft Error Mitigation

SET Single Event Transient

SEU Single Event Upset

SOC System On Chip

SRAM Static Random Access Memory

SSI Stacked Silicon Interconnect

SW Software

TID Total Ionizing Dose

TMR Triple Modular Redundancy

VHDL VHSIC Hardware Description Language

Capitolo 1

Introduzione

Questo capitolo introduce il contesto in cui la tesi è stata sviluppata. Innanzitutto viene presentata l'azienda Argotec. In seguito viene descritto il problema delle radiazioni nello spazio. Infine viene illustrato il concetto di riconfigurabilità computazionale.

1.1 Argotec

Questa tesi è stata sviluppata in collaborazione con Argotec [30], azienda ingegneristica aerospaziale torinese. Argotec progetta e sviluppa prodotti ed esperimenti utilizzati sulla Stazione Spaziale Internazionale (International Space Station (ISS)) con l'obiettivo di applicare nella vita di tutti i giorni le tecnologie sviluppate. L'azienda si occupa inoltre del training del personale di terra e degli astronauti europei, preparandoli alle missioni sulla ISS.

Argotec è composta da diverse unità:

- Payload and Tech Unit (PTU)
- Training & Simulation Unit (TSU)
- Small Satellite Unit (SSU)
- Space Food (RTL)

1.1.1 Payload and Tech Unit

Argotec si occupa di attività di ricerca e sviluppo in diverse aree dell'industria aerospaziale. Inoltre, l'azienda ha preso parte in molte attività riguardanti la progettazione e la

realizzazione di sistemi termo-fluido dinamici per la ISS. Questi dispositivi non hanno come unico obiettivo l'applicazione spaziale, ma sono pensati per sviluppare tecnologie innovative utilizzabili direttamente su sistemi terresti. Con questo obiettivo sono stati realizzati ISSpresso, Advanced Research Thermal Passive Exchange (ARTE) e altri payload per la stazione spaziale.

ISSpresso ISSpresso, la prima macchina del caffé basata su capsule in grado di lavorare anche nelle condizioni estreme dello spazio, è il frutto di una collaborazione fra Argotec, Lavazza e l'Agenzia Spaziale Italiana. L'obiettivo è di dare la possibilità di gustare un autentico espresso anche agli astronauti sulla ISS.
Il 14 Aprile 2015, ISSpresso viene lanciata a bordo della missione CRS-6 di SpaceX. Il primo espresso nello spazio è stato bevuto dall'astronauta della agenzia spaziale europea (European Space Agency) Samantha Cristoforetti il 3 Maggio 2015, come mostrato in Figura 1.1.
ISSpresso è progettata per preparare non solo il caffè, ma anche altre bevande calde quali tea e brodo. In particolare, il brodo permette di reidratare il cibo destinato agli astronauti. Oltre a permettere un piacevole momento di relax, questo particolare payload offre l'opportunità di studiare alcuni fenomeni fisici relativi alla fluido dinamica dei liquidi ad alte pressioni e temperature in condizioni di microgravità. Questi studi hanno portato a realizzare un sistema in grado di minimizzare la quantità di acqua necessaria per ottenere il caffè.

Figura 1.1 *Samantha Cristoforetti beve il primo caffè espresso prodotto nello spazio da ISSpresso (dietro di lei). Fonte: Argotec srl [30]*

ARTE Nel 2014 Argotec inizia la progettazione di un dimostratore tecnologico con l'obiettivo di studiare più approfonditamente le caratteristiche delle heat pipe in condizioni di microgravità. Le heat pipe sono dispositivi dal peso ridotto ma in grado di trasferire passivamente il calore con grande efficienza. Questi sistemi vengono già utilizzati sulla stazione spaziale con ottimi risultati, ma richiedono l'uso di sostanze tossiche che possono causare seri pericoli in caso di perdite. ARTE, mostrata in Figura 1.2, sfrutta invece sostanze a bassa tossicità, permettendo quindi di sviluppare sistemi efficienti ed adatti a veicoli abitati.

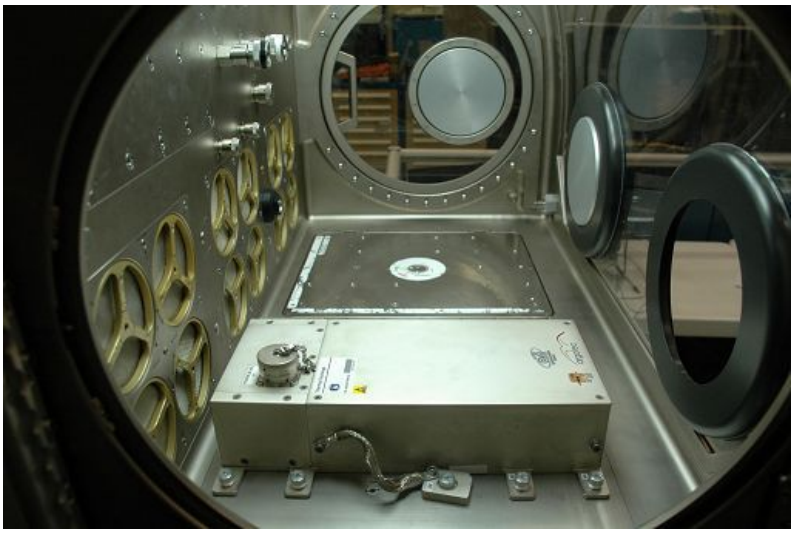

Figura 1.2 *ARTE a bordo della ISS. Fonte: Argotec srl [30]*

1.1.2 Training & Simulation Unit

Argotec vanta una lunga e riconosciuta storia nel settore del training per l'agenzia spaziale europea. All'European Astronaut Centre a Cologna, personale certificato prepara gli astronauti e i controllori di volo ai protocolli di comunicazione, alle operazioni sulla ISS e alle regole di volo. Gli istruttori organizzano anche delle simulazioni per verificare il comportamento dei futuri astronauti in situazioni di emergenza a bordo della stazione.

1.1.3 Small Satellite Unit

Satelliti di dimensioni sempre minori sono diventati comuni negli ultimi anni. Essi sono caratterizzati da una massa compresa fra 0.1 kg e 500 kg. Dispositivi di queste dimensioni sono utili per effettuare esperimenti scientifici, testare nuove tecnologie oppure fornire

servizi commerciali mantenedo il costo basso. In questo contesto Argotec sta sviluppando Argomoon, un satellite che raggiungerà l'orbita lunare.

ArgoMoon (Figura 1.3) sarà l'unico rappresentante europeo fra i dodici satelliti che voleranno nella Exploration Mission 1 del nuovo razzo sviluppato dalla National Aeronautics and Space Administration (NASA), lo Space Launch System. L'obiettivo di Argomoon è scattare fotografie storicamente significative della missione e testare dei sistemi di comunicazione innovativi. L'orbita translunare in cui verrà inserito gli permetterà di raggiungere lo spazio profondo, un ambiente particolarmente ostile in termini di radiazioni in quanto la protezione del campo magnetico terrestre viene meno. Questa particolare condizione permetterà agli ingegneri di Argotec di testare nuove tecnologie che potranno essere applicate nei futuri viaggi interplanetari.

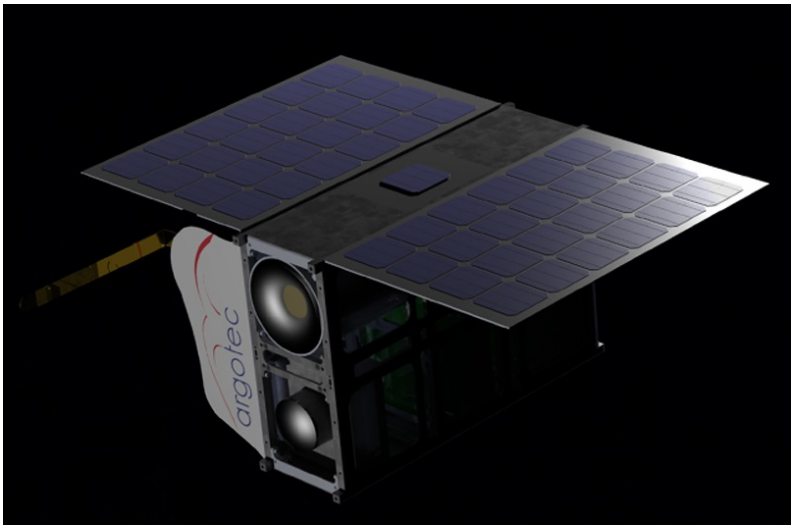

Figura 1.3 *Rappresentazione del satellite ArgoMoon. Fonte: Argotec srl [30]*

1.1.4 Space Food

Dal 2010 Argotec è la responsabile europea per lo sviluppo e il rifornimento del cibo per gli astronauti sulla stazione spaziale internazionale. L'azienda agisce quindi per conto di ESA come interfaccia verso NASA e i più importanti fornitori di generi alimentari europei. È inoltre incaricata del *bonus food*, un pasto pensato per i gusti personali di ogni astronauta e che viene consumato in occasioni particolari e per sostenere psicologicamente l'equipaggio a bordo della stazione.

Sulla ISS non sono presenti congelatori, quindi è necessario usare tecniche particolari per conservare il cibo per periodi fino a 24 mesi. Per affrontare questa sfida tecnologica, Argotec ha sviluppato indipendentemente una nuova area di ricerca per eliminare la presenza di patogeni senza intaccare le caratteristiche nutrizionali ed organolettiche delle pietanze.

1.2 FPGA e riconfigurabilità parziale dinamica

In questa sezione viene presentato il dispositivo Field Programmable Gate Array (FPGA), un tipo di circuito integrato utilizzato nello sviluppo della tesi. Viene inoltre introdotto il concetto di riconfigurabilità parziale dinamica.

1.2.1 FPGA

Una FPGA è un circuito integrato digitale progettato per essere configurato da un progettista senza la necessità di un intervento da parte del produttore, ovvero *on the field*. È composto da una matrice di elementi logici programmabili e dalle connessioni necessarie a collegarli fra loro e verso le porte di ingresso/uscita, come riportato in Figura 1.4.

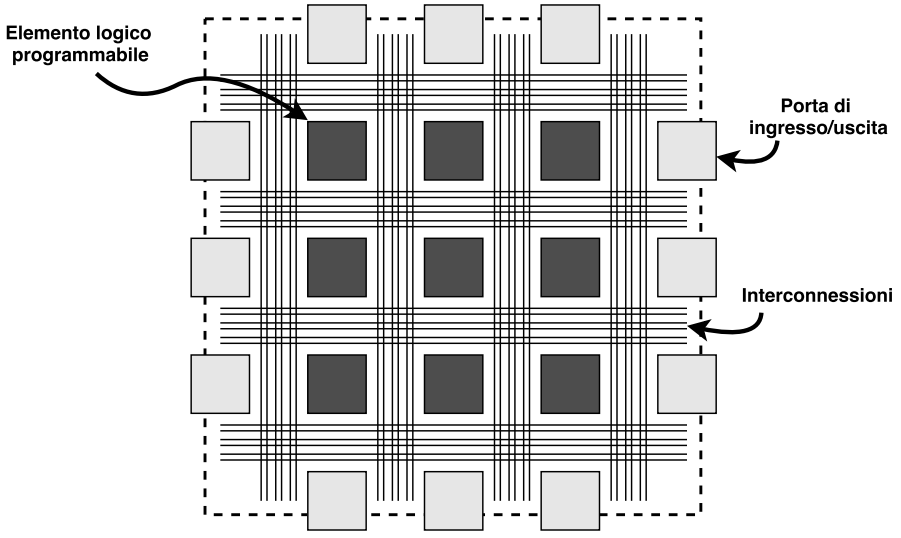

Figura 1.4 *Struttura di una FPGA*

Il file (chiamato *bitstream*) che descrive la configurazione del dispositivo è memorizzato in una *configuration memory* sfruttando diverse tecnologie quali antifusibile, flash e Static Random Access Memory (SRAM):

- **Antifusibile:** l'informazione viene memorizzata creando dei cortocircuiti negli elementi di memoria. Un antifusibile è costituito da due contatti metallici separati da pochi nanometri di silicio amorfo, il quale presenta caratteristiche dielettriche. Se viene applicata una tensione sufficientemente alta ai due contatti, il silicio fonde e si crea una lega policristallina conduttiva silicio-metallo. Questa tecnica non è reversibile e viene utilizzata in dispositivi *one time programmable*. Un produttore di FPGA con tecnologia ad antifusibile è *Microsemi* [32].

- **Flash:** è una tipologia di memoria non volatile, in cui vengono sfruttati dei *floating gate Metal-Oxide Semiconductor Field Effect Transistor (MOSFET)* (Figura 1.5) per memorizzare il dato. Questi MOSFET presentano un secondo gate isolato detto *floating gate*, posto fra il canale e il gate di controllo (*control gate*). Applicando un campo elettrico è possibile iniettare della carica nel floating gate, modificando la tensione di soglia del transistor. Essendo il gate isolato, questa carica rimane intrappolata consentendo così di immagazzinare un'informazione. Questo tipo di memoria può essere scritto e cancellato, permettendo la riprogrammazione della FPGA. Un produttore di FPGA con tecnologia flash è *Altera* [33].

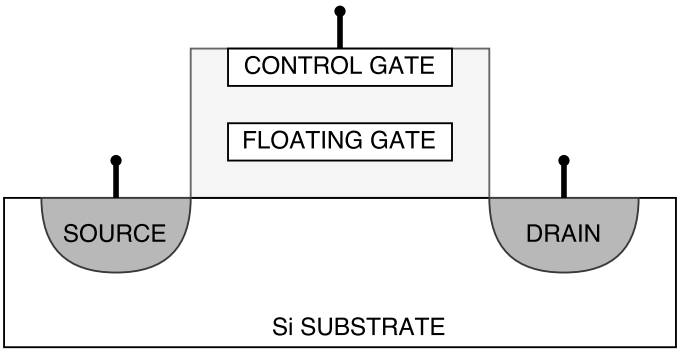

Figura 1.5 *Struttura di un MOSFET Floating Gate*

- **SRAM:** è un tipo di memoria volatile. Una cella di memoria è composta da due invertitori logici connessi in retroazione. Sbilanciando gli ingressi di questa cella, è possibile imporre un valore logico che verrà mantenuto finchè il sistema viene alimentato. Una FPGA basata su questa tecnologia può essere riprogrammata un numero illimitato di volte, ma richiede una memoria non volatile aggiuntiva per conservare la configurazione in assenza di alimentazione. Un produttore di FPGA con tecnologia SRAM è *Xilinx* [34].

Esistono inoltre altre tecnologie, ad esempio Programmable Read-Only Memory, Erasable Programmable Read-Only Memory e a fusibile, ma sono considerate obsolete.
Ogni elemento programmabile della FPGA permette di realizzare delle funzioni logiche elementari, ma combinandoli assieme è possibile implementare algoritmi estremamente complessi. Queste unità base sono composte da delle Look-Up Table (LUT) e da elementi di memoria, anche se i dettagli implementativi sono diversi per ogni produttore. Xilinx definisce questi elementi logici come Configurable Logic Block (CLB). Un esempio molto semplificato di CLB è illustrato in Figura 1.6. La LUT permette di implementare qualunque funzione logica a tre ingressi utilizzando gli input per selezionare un valore da un vettore contenente $2^{(\#\ input)}$ bit memorizzato nella memoria di configurazione. L'output della LUT è quindi portato in uscita al CLB. Un multiplexer permette inoltre di indirizzare verso un flip-flop un ingresso esterno al CLB oppure il risultato generato dalla LUT.

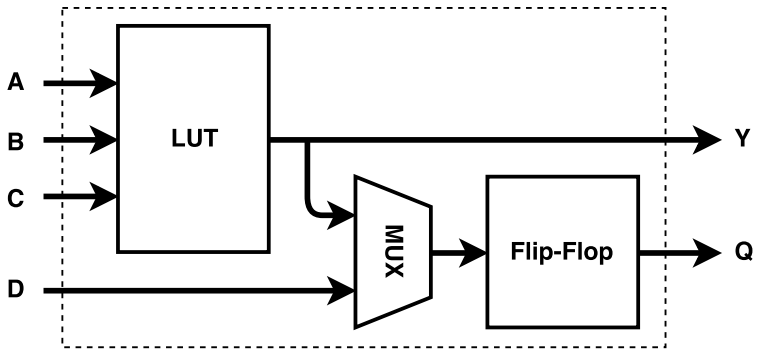

Figura 1.6 *Struttura di un CLB*

I CLB sono collegati fra loro da una rete di interconnessioni configurabili, sfruttando delle *matrici di scambio*. Queste permettono di indirizzare un segnale utilizzando dei pass-transistor, i quali mettono in comunicazione oppure isolano le linee. Per collegare CLB lontani fra loro, sono inoltre previste delle connessioni che non attraversano le matrici di scambio, in modo da ridurre considerevolmente il ritardo introdotto dagli switch.
Una FPGA si interfaccia con dispositivi esterni tramite dei blocchi di ingresso/uscita, chiamati Input-Output Block. Questi elementi permettono di configurare i pin del circuito integrato come porte di ingresso, di uscita, bidirezionali oppure tri-state. Inoltre è possibile attivare delle resistenze di pull-up o di pull-down, programmare le soglie di ingresso per poter dialogare con diverse famiglie logiche e modificare la rapidità di commutazione dei segnali d'uscita (detto *slew rate*).
Questi dispositivi vengono programmati utilizzando un linguaggio di descrizione dell'hardware, il quale permette di descrivere esattamente come le risorse fisiche della FPGA

devono essere configurate. Attualmente i liguaggi più utilizzati sono il VHSIC Hardware Description Language (VHDL) e il Verilog. È inoltre possibile utilizzare una modalità di programmazione chiamata *schematic-entry*, la quale permette di descrivere un sistema in maniera grafica utilizzando i simboli degli elementi logici elementari. Questa modalità diventa però difficilmente gestibile per sistemi complessi e inoltre risulta essere poco parametrizzabile.

1.2.2 Riconfigurabilità parziale dinamica

La possibilità di programmare una FPGA sul campo e per un numero illimitato di volte (come nel caso di dispositivi basati su SRAM) ha permesso di sviluppare l'idea di riconfigurabilità nata nel 1960 ad opera di Gerald Estrin [4]. Con l'obiettivo di migliorare le performance dei computer, Estrin propose di sviluppare un'architettura composta da due sezioni. La prima, detta "fixed", è composta da un elaboratore general purpose mentre la seconda, detta "variable", implementa diversi moduli hardware specializzati per eseguire una determinata funzione. Quest'ultimi possono essere attivati dalla sezione fixed quando necessario, permettendo di aumentare la velocità del sistema rispetto alla sola elaborazione software.
Lo sviluppo della tecnologia dei semiconduttori ha consentito di implementare architetture molto complesse all'interno di un singolo chip. La grande flessibilità offerta dalle FPGA può quindi essere sfruttata per applicare il concetto di riconfigurabilità. Un sistema composto da un processore connesso tramite un bus ad alta velocità ad una FPGA permette di utilizzare quest'ultima come un coprocessore, aumentando la potenza computazionale in maniera decisiva.
Un'estensione del concetto di riconfigurabilità è rappresentata dalla *riconfigurabilità parziale dinamica* [5]. Normalmente, durante una riconfigurazione è necessario mantenere in reset l'intera FPGA. Questo avviene anche nel caso in cui vengano modificati solo alcuni dei moduli implementati, interrompendo quindi l'elaborazione dei dati da parte delle altre unità. Nei dispositivi che la supportano, la riconfigurabilità parziale dinamica permette invece di intervenire solamente su alcuni elementi, mantendo operativo il resto del sistema. Per applicare questa proprietà è necessario seguire un particolare flusso di progetto, ponendo attenzione alla modularità del sistema. Nei dispositivi riconfigurabili prodotti da Xilinx vengono definite delle Reconfigurable Partition (RP) sulla matrice della FPGA, all'interno delle quali vengono implementati i Reconfigurable Module (RM) (Figura 1.7). Le RP permettono di imporre dei vincoli fisici ai moduli, individuandoli e isolandoli dal resto del sistema durante il processo di riconfigurazione. In ogni RP possono essere usati diversi RM, purché quest'ultimi presentino le stesse interfacce fisiche previste dalla RP.

Durante la fase di progetto viene creato un *partial bitstream* per ogni RM. Durante processo di riconfigurazione, questo bitstream viene scritto nella porzione di configuration memory relativa alla RP che si vuole riconfigurare.

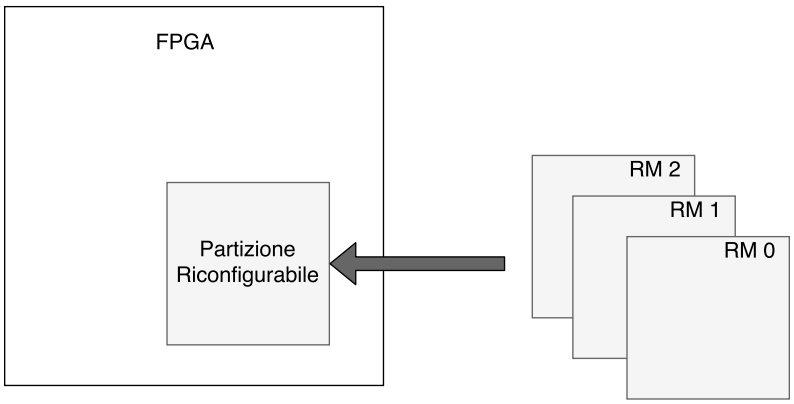

Figura 1.7 *Principio base della riconfigurazione parziale dinamica. All'interno di una FPGA viene definita una RP, nella quale possono essere caricati diversi moduli riconfigurabili*

1.3 Radiazioni e relativi effetti

In questa sezione vengono descritte le radiazioni presenti nello spazio e i loro effetti sui circuiti elettronici.

1.3.1 Raggi Cosmici

I raggi cosmici sono composti da particelle subatomiche ad alta energia che attraversano lo spazio. Essi sono costituiti in gran parte da protoni e nuclei di elio (particelle α)[1], come riportato nel grafico in Figura 1.8. La loro origine è stata individuata principalmente nelle supernovae [2], ovvero nell'esplosione che avviene al termine della vita di una stella. Un'altra sorgente di queste particelle ad alta energia è rappresentata dal vento solare, causato dalle eruzioni della corona del Sole, le quali proiettano nello spazio interplanetario elettroni, protoni e nuclei di elio. Queste particelle vengono accelerate dai campi magnetici generati dalle supernovae o dalle stelle fino a velocità prossime a quella della luce. Questa alta velocità le rende particolarmente energetiche, con valori rilevati fino a 10^{20} eV [3].
Il campo magnetico terrestre cattura tali particelle nelle *fasce di Van Allen*. Queste regioni della magnetosfera permettono di schermare la Terra dal vento solare, creando però

un ambiente radiativo pericoloso per gli esseri umani e per l'elettronica di bordo di un satellite.

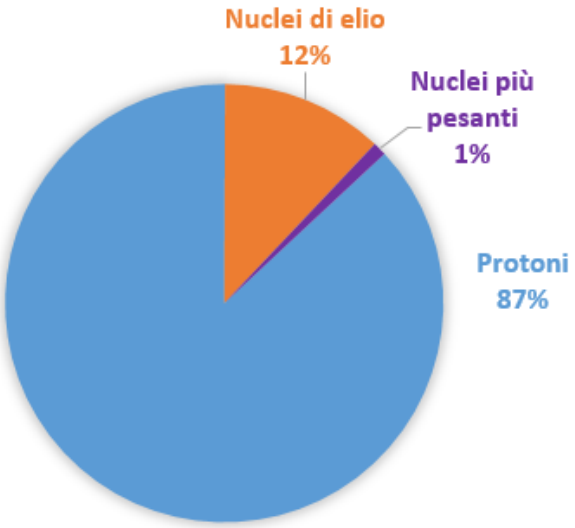

Figura 1.8 *Composizione dei raggi cosmici*

1.3.2 Effetti delle radiazioni sui componenti elettronici

I componenti elettronici risultano essere particolarmente sensibili alle radiazioni, con effetti che possono essere divisi in due tipologie:

- Effetti cumulativi a lungo termine, conosciuti anche come Total Ionizing Dose (TID)
- Effetti transitori, noti come Single Event Effects (SEE)

Effetti cumulativi a lungo termine - Total Ionizing Dose

Gli effetti a lungo termine danneggiano la struttura cristallina del semiconduttore, degradando le caratteristiche fisiche del dispositivo. Quando una particella carica colpisce un transistor si può creare una coppia elettrone-lacuna. In particolare, se questa coppia si genera all'interno dell'ossido di gate di un MOSFET, il forte campo elettrico presente separa l'elettrone dalla lacuna. Mentre il primo si allontana velocemente grazie alla sua maggiore mobilità, la lacuna, più lenta, può essere catturata all'interno delle impurità presenti nell'ossido. Il risultato è un accumulo di carica nel dispositivo (Figura 1.9), che porta ad un cambiamento dei parametri elettrici del transistor. Il principale effetto è una variazione della tensione di soglia del MOSFET. Questo parametro influenza pesantemente la capacità di controllo del dispositivo, che nel caso peggiore può non essere più in

grado di gestire il flusso di corrente fra source e drain, rimanendo sempre in interdizione (nel caso dei p-channel Metal-Oxide Semiconductor (MOS)) oppure sempre in conduzione (nel caso degli n-channel MOS). Si ha inoltre un incremento della corrente di *leakage* [7], con conseguente aumento del consumo di potenza.

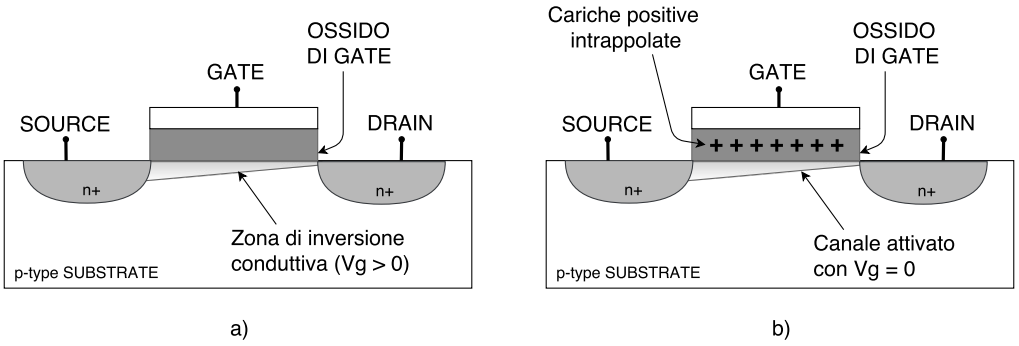

Figura 1.9 *Effetto TID su un NMOS prima (a) e dopo (b) l'esposizione alle radiazioni. È possibile notare l'accumulo di carica all'interno dell'ossido di gate.*

Effetti transitori - Single Event Effects

I SEE sono generati dall'attraversamento in un semiconduttore di una particella carica. Perdendo energia, questa genera una scia di atomi ionizzati, come rappresentato in Figura 1.10. Questi effetti transitori possono essere divisi in *distruttivi* (hard errors) e *non distruttivi* (soft errors).

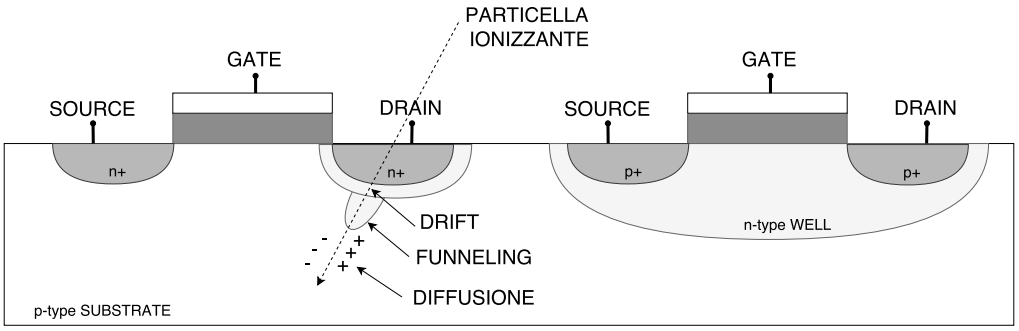

Figura 1.10 *Effetto di un SEE su un MOS. È possibile notare la scia di atomi ionizzati generata dalla particella carica*

Soft errors Nel caso dei soft errors, la carica risultante dall'attraversamento viene raccolta nelle giunzioni dei dispositivi. Se questa è maggiore rispetto a quella immagazzinata in un nodo sensibile del circuito, la relativa informazione è irrimediabilmente persa. Appare quindi evidente come i circuiti digitali siano particolarmente sensibili a questo tipo di effetti delle radiazioni, in quanto il risultato è la variazione dello stato logico di un bit. Nel caso in cui questo glitch venga memorizzato da un elemento di memoria, esso può ripercuotersi sulle future elaborazioni eseguite dal circuito. Questo tipo di errore può essere risolto eseguendo un reset del sistema oppure sovrascrivendo l'informazione.
I soft errors possono quindi essere suddivisi nelle seguenti categorie:

- **Single Event Transient (SET)** avviene quando la carica risultante del SEE si scarica attraverso il circuito, assumendo la forma di un segnale spurio. L'effetto è lo stesso di una scarica elettrostatica.
- **Single Event Upset (SEU)** è il cambiamento di stato di un bit in una memoria o in un registro in seguito all'interazione di una particella carica con il dispositivo. In particolare, un SET che si propaga all'interno di un circuito digitale fino a raggiungere un elemento di memoria viene considerato come SEU. Esso non causa nessun danno fisico, ma può portare a gravi errori se il sistema non è in grado di individuare e correggere il problema. In dispositivi particolarmente sensibili, una singola particella può causare errori multipli (Multiple Bit Upset) in diverse celle di memoria adiacenti. Inoltre, i SEU possono diventare Single-event Functional Interrupts nel caso in cui colpiscano circuiti di controllo quali macchine a stati. Questo può portare il sistema in uno stato indefinito, con la necessità di applicare un reset per ripristinare il corretto funzionamento.

Hard errors Gli hard errors portano alla distruzione fisica del dispositivo. Essi si dividono in:

- **Single Event Latchup (SEL)**, che possono colpire i chip che presentano una struttura parassita PNPN fra le alimentazioni, come i dispositivi con tecnologia Bulk Complementary MOS. Se uno ione pesante oppure un protone ad alta energia attraversa una delle giunzioni di questa struttura, essa può attivarsi mettendo in cortocircuito l'alimentazione del dispositivo. L'unica soluzione è effettuare un power-cycle. L'effetto è conosciuto come *latch-up*, e se non gestito efficacemente può portare alla distruzione del semiconduttore a causa delle alte correnti che attraversano la struttura parassita. Soluzioni quali Silicon On Insulator oppure Silicon On Sapphire permettono di risolvere il problema.

- **Single-Event Snapback**, il quale risulta essere molto simile al latch-up, ma è causato da una moltiplicazione a valanga dei portatori attivata da uno ione. Infatti, se la radiazione colpisce la giunzione di drain di un transistor di potenza a canale N, quest'ultimo entra in conduzione e non è più possibile spegnerlo.
- **Single Event Burnout**, che può accadere nei MOSFET di potenza, quando il substrato vicino alla regione di source viene polarizzato direttamente e la tensione drain-source è maggiore di quella di breakdown dei diodi parassiti. Il risultato è un'alta corrente che surriscalda il transistor fino a portarlo alla distruzione.
- **Single Event Gate Rupture**, il quale avviene quando uno ione pesante colpisce la regione di gate di un MOSFET mentre vi è applicata un'alta tensione. Il forte campo elettrico porta alla rottura del dielettrico di gate distruggendo il transistor, con conseguente passaggio di corrente e relativo surriscaldamento.

Capitolo 2

Possibili soluzioni al problema delle radiazioni e stato dell'arte

Questo capitolo introduce le principali tecniche utilizzate per permettere ai componenti elettronici di resistere alle radiazioni presenti nello spazio. In particolare verranno analizzate le tecniche di ridondanza hardware. Viene inoltre presentato lo stato dell'arte relativo all'applicazione della riconfigurabilità nei sistemi spaziali.

2.1 Shielding

Per ridurre l'esposizione alle radiazioni è possibile schermare i componenti e i sistemi più sensibili. Il materiale che compone la schermatura dipende dal tipo di particella dalla quale è necessario proteggere. I raggi cosmici sono in gran parte composti da protoni e particelle alpha, i quali possono essere arrestati da uno strato di metallo. Il potere di arresto è direttamente proporzionale al numero atomico dell'elemento utilizzato, per questo motivo vengono spesso impiegati il piombo e il tantalio. L'alluminio risulta essere meno efficace, ma viene spesso utilizzato in quanto ha un peso minore a parità di spessore. Per schermare i dispositivi dai neutroni è invece possibile utilizzare materiali ad alto contenuto di idrogeno. Il polietilene (CH_2) ha dimostrato essere più efficace di alcuni metalli contro questo tipo di radiazioni [6].
Il principale vantaggio di questa tecnica è costituito dalla ridotta complessità con cui può essere applicata. Inoltre permette di contrastare sia gli effetti TID che SEE. Porta però ad un aumento dell'ingombro e del peso, risorse particolarmente critiche in un satellite.

2.2 Tecnologie di realizzazione degli IC

Gli effetti delle radiazioni possono essere mitigati applicando delle soluzioni tecnologiche nella realizzazione dei circuiti integrati [8]. Alcune delle tecniche più usate sono le seguenti:

- **Silicon-On-Insulator:** questa tecnologia consiste nel creare uno strato di semiconduttore sopra un livello di dielettrico, chiamato Buried Oxide. Ciò permette di isolare il substrato di ogni MOSFET, eliminando così i transistor parassiti. Questa soluzione risulta quindi particolarmente efficace nei confronti dei SEL, dei SET e dei SEU. Essa permette inoltre di ridurre drasticamente le correnti di perdita, diminuendo la potenza consumata dal dispositivo.

- **Triple wells:** per isolare i MOSFET è possibie introdurre dei layer di drogaggio aggiuntivi. Nella struttura ottenuta, i MOSFET a canale P sono implementati in un N-well (come nel caso di un processo a singolo o doppio well), mentre il P-well dei MOSFET a canale N è a sua volta inserito in un N-well più profondo. Questo permette di isolare dal substrato entrambi i transistor tramite una giunzione polarizzata inversamente, limitando i SEL e i SEU.

- **Buried layers:** questa tecnica prevede di proteggere i MOSFET implementati sulla superficie del semiconduttore attirando in una zona altamente drogata l'eccesso di carica depositato dalle radiazioni. Questa zona è inserita all'interno del substrato o del well, lontano dal transistor. In questo modo è necessaria una maggiore quantità di carica per attivare il latch-up, risultando in un incremento dell'immunità ai SEL. Inoltre anche i SET e i SEU vengono mitigati.

- **Enclosed layout transistor:** per diminuire la sensibilità di un MOSFET ai SEE e a TID, è possibile ridurre la cross-section del transistor. Una soluzione prevede di ridurre l'area occupata dal drain, implementandolo al centro del MOS e non lateralmente. Questo permette inoltre di ridurre la corrente di perdita, considerando che il drain non presenta più punti di contatto diretto con il substrato.

- **Guard rings:** l'immunità al latch-up può essere incrementata introducendo degli anelli di guardia. Questi sono connessi direttamente all'alimentazione e permettono di catturare le cariche generate dalle radiazioni, riducendo la possibilità i transistor parassiti vengano attivati causando un SEL.

L'impiego di queste tecniche porta ad un aumento dell'area occupata da ogni transistor, e quindi ad una minore densità di MOSFET nei circuiti integrati. Questo significa una riduzione delle funzionalità che l'Integrated Circuit (IC) può implementare.

2.3 Ridondanza

Un modo per poter tollerare ed eventualmente correggere i soft errors causati dai SEE è aumentare la ridondanza del sistema. Questo richiede di incrementare il numero di risorse disponibili, in modo da poter tollerare la perdita di qualcuna di queste in caso di guasto. Il concetto di ridondanza può essere applicato a diverse sottocategorie:

- Ridondanza hardware (o spaziale);
- Ridondanza software (o temporale);
- Ridondanza dell'informazione.

2.3.1 Ridondanza hardware

La ridondanza hardware richiede di replicare più volte dei moduli funzionali e di confrontarne gli output con l'obiettivo di individuare eventuali discrepanze, indici di errori (faults). Il più semplice esempio di ridondanza è dato da Duplication With Comparison (DWC), rappresentato in Figura 2.1, nel quale il modulo funzionale viene duplicato.

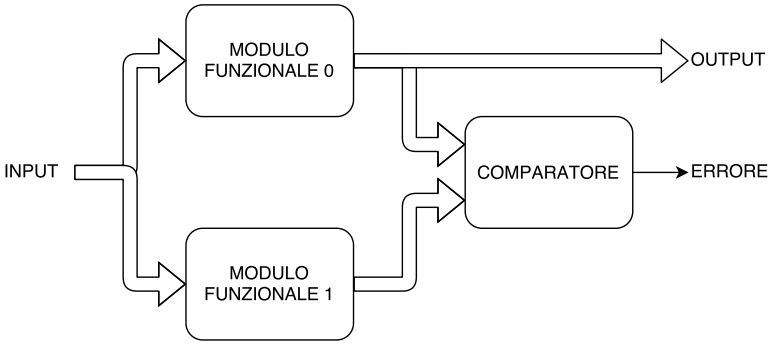

Figura 2.1 *Architettura DWC*

Confrontando i dati elaborati dai due moduli è possibile individuare la presenza di un errore e quindi agire di conseguenza, segnalando la situazione ed evitando che l'errore si propaghi nel sistema. L'architettura non è però in grado di mascherare il fault. Questo limite può essere superato aumentando il numero di repliche seguendo la legge

$$\#\ repliche = 2N + 1,\ con\ N = 1,2,3,...$$

e confrontando i risultati dei singoli moduli funzionali con un voter, il quale applica un algoritmo di maggioranza come riportato nella Tabella 2.1.

IN 0	IN 1	IN 2	Out
A	A	A	A
B	A	A	A
A	B	A	A
A	A	B	A
A	B	C	Output corrotto

Tabella 2.1 *Tabella logica di un voter. A, B e C rappresentano dei generici valori di ingresso*

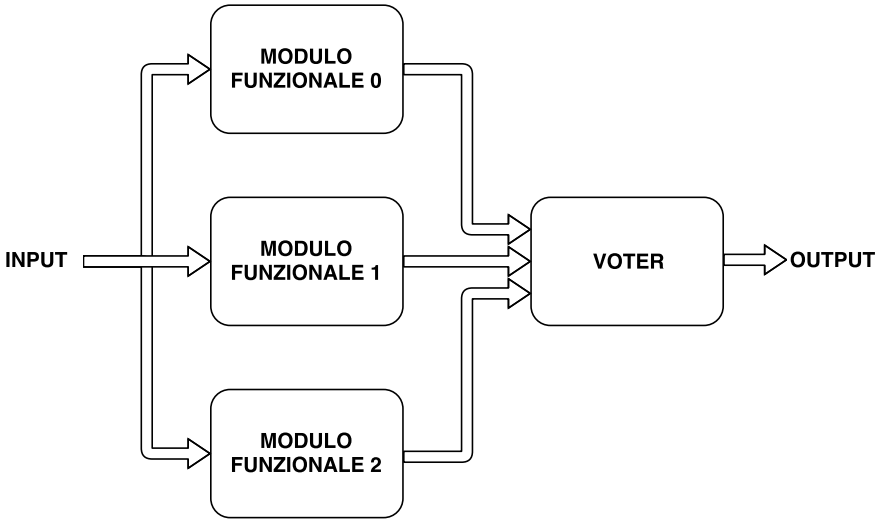

Figura 2.2 *Architettura TMR*

Prendendo ad esempio il caso $N = 1$, il numero delle repliche risulta essere pari a 3. L'architettura che ne risulta è chiamata TMR [9] ed è schematizzata nella Figura 2.2. Con riferimento al numero di errori nei moduli funzionali, è possibile identificare tre situazioni:

- Nessun errore: gli ingressi del voter sono uguali, l'algoritmo di maggioranza non rileva alcun fault;
- Un modulo funzionale subisce un errore: uno degli ingressi al voter è discorde rispetto agli altri, il voter applica quindi l'algoritmo di maggioranza e individua il modulo fallato. L'algoritmo permette di mascherare l'errore, ignorando il dato non corretto;
- Due (o tre) moduli funzionali subiscono un errore: gli ingressi al voter risultano essere discordi, l'algoritmo di maggioranza non è in grado di individuare la locazione dei fault e determinare un dato correttamente elaborato.

Analizzando queste situazioni, è possibile dedurre che il sistema è in grado di mascherare fino a

$$\#\ fault\ mascherabili = 2N - 1$$

faults mentre può rivelare

$$\#\ fault\ rivelabili = 2N$$

faults. È importante notare come il voter rappresenti un *single point of failure*, ovvero un elemento che, se colpito, causa il fallimento dell'intero sistema. Anche se quest'ultimo occupa spesso un'area minore rispetto ai moduli funzionali ed è quindi soggetto in misura inferiore ai SEE, un fault in questo elemento vanificherebbe l'intera architettura. Per risolvere questo problema è possibile replicare anche questo elemento, aumentando l'affidabilità del sistema ma anche il costo e la complessità. Un sistema TMR con voter ridondati è mostrato in Figura 2.3.

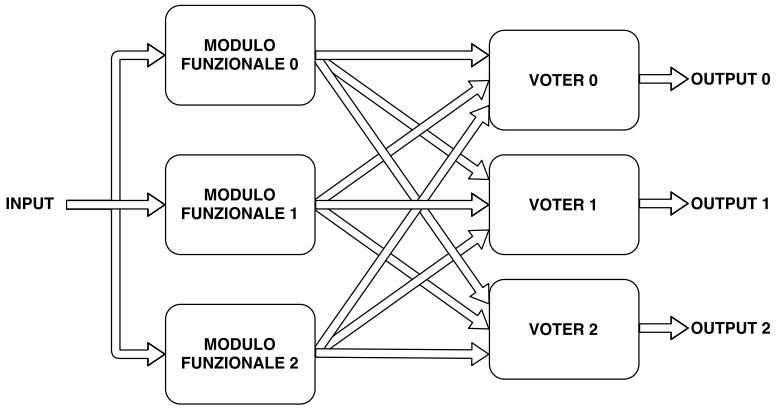

Figura 2.3 *Architettura TMR in cui anche i voter sono stati triplicati*

Quanto visto può essere impiegato nella realizzazione di sistemi più complessi, detti a *ridondanza attiva*. In questi sistemi le istanze del modulo funzionale sono connesse ad uno switch (Figura 2.4). Il sistema analizza i dati elaborati dai moduli, rivelando un eventuale fault (ad esempio applicando DWC). Quando un errore viene generato da un modulo, lo switch lo rileva analizzando i risultati dei DWC e il modulo viene quindi isolato. Questa architettura permette di tollerare fino a $M - 1$ fault, con M pari al numero di repliche del modulo funzionale. Lo svantaggio di questi sistemi è dato dalla complessità implementativa. Oltre all'aumento del numero di moduli funzionali è infatti necessario realizzare anche i sistemi DWC e lo switch. Inoltre, questi ultimi elementi potrebbero essere soggetti ad errori a loro volta, come nel caso del voter nel TMR. Ridondare anch'essi porterebbe ad un ulteriore incremento della complessità di questa soluzione.

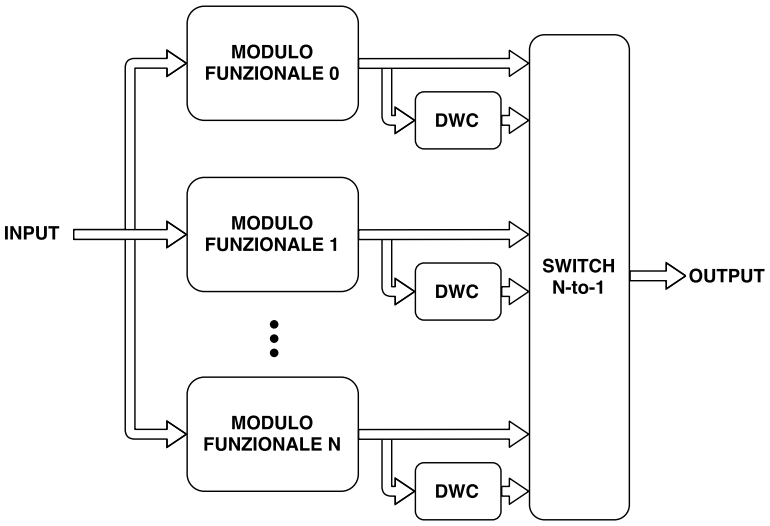

Figura 2.4 *Architettura di un sistema a ridondanza attiva*

La ridondanza Hardware (HW) può essere applicata su diversi livelli:

- **Livello sistema (alto livello)**, replicando interi sistemi quali il computer di bordo e confrontandone i risultati solo al termine dell'elaborazione. Un esempio è l'On-Board Computer (OBC) dello Space Shuttle [10], il quale era ridondato cinque volte;
- **Livello dispositivo**, ad esempio aumentando il numero di Central Processing Unit (CPU) di un elaboratore;
- **Livello gate (basso livello)**, applicando questa soluzione fra diversi stadi di porte logiche.

È possibile analizzare queste diverse soluzioni in termini di complessità, performance ed affidabilità, come mostrato nel grafico 2.5.

Ad alto livello le performance risultano essere più elevate in quanto viene introdotto un numero inferiore di voter, i quali allungano il percorso critico e aumentano la latenza. D'altro canto, ad un livello inferiore il maggior numero di stadi di voting permette di isolare in maniera più puntuale i fault. Questo permette di tollerare complessivamente più errori e quindi aumentare l'affidabilità, al prezzo di una maggiore complessità e prestazioni più limitate.
In generale, la ridondanza hardware permette di rivelare e correggere SEU e SET. Gli svantaggi sono rappresentati da un overhead considerevole sull'area (più del 200% nel caso

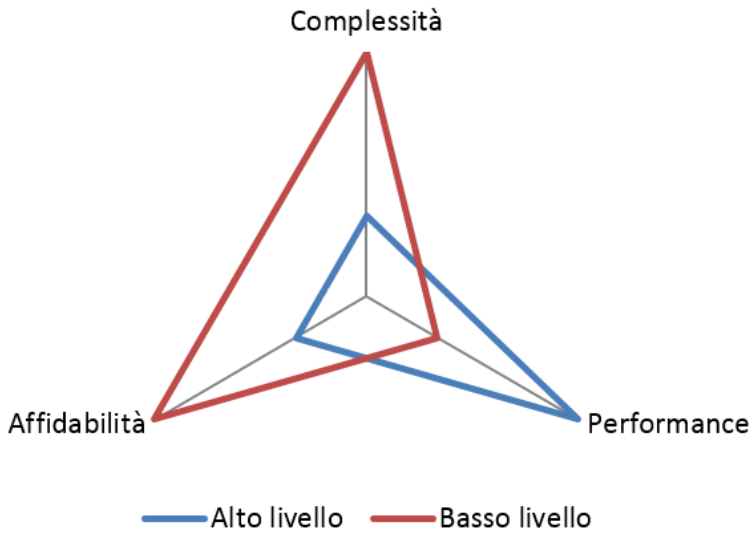

Figura 2.5 *Confronto fra i livelli a cui è possibile applicare il TMR*

del TMR) e quindi sui consumi e un incremento della latenza a causa dell'inserimento dei comparatori.

2.3.2 Ridondanza software

La ridondanza software prevede di replicare il numero di esecuzioni di una stessa funzione per poi confrontarne i risultati (Figura 2.6). Questo permette di rivelare e correggere un eventuale soft error avvenuto durante l'elaborazione. Mentre non sono richieste risorse HW aggiuntive, questa tecnica introduce un overhead sul tempo necessario a completare il calcolo. Similmente alla ridondanza HW, se un'operazione viene eseguita due volte è possibile rivelare un fault, mentre per poterlo correggere è necessario applicare lo stesso algoritmo almeno tre volte.
Anche la ridondanza Software (SW) può essere applicata a diversi livelli. È possibile replicare l'esecuzione di intere applicazioni, di solo alcune funzioni e algoritmi oppure delle singole istruzioni. Generalmente, il principale vantaggio è un'ottima copertura dai fault nell'elaborazione dei dati. Gli svantaggi sono invece rappresentati da un aumento del tempo impiegato a completare un'operazione (un incremento da 2 a 4 volte [8]) e una maggiore occupazione di memoria, necessaria a memorizzare i risultati da confrontare e un codice più lungo.

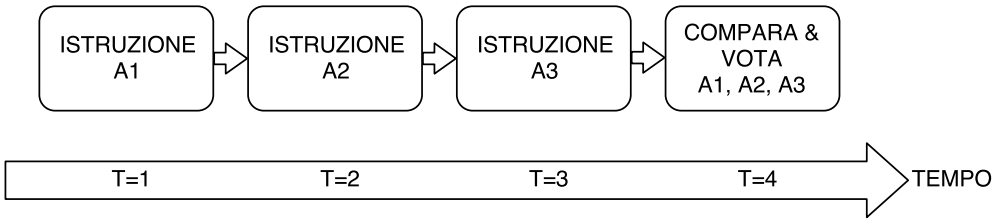

Figura 2.6 *Applicazione della ridondanza SW. Le singole operazioni vengono ripetute tre volte e infine i risultati vengono confrontati*

2.3.3 Ridondanza dell'informazione

Questa forma di ridondanza richiede di utilizzare più simboli di quanti effettivamente necessari a codificare un'informazione. Essa viene utilizzata per proteggere i dati memorizzati in memorie o in registri applicando dei codici a correzione d'errore (Error-Correcting Codes (ECC)) ai dati. Esistono diversi algoritmi per implementare questa tecnica, ad esempio il *controllo di parità* e i *codici di Hamming*. Ogni algoritmo è caratterizzato da diverse possibilità di individuare e correggere uno o più fault, ma in generale questo tipo di ridondanza richiede un aumento di area nelle memorie e un overhead temporale necessario ad elaborare ed eventualmente correggere i dati.

2.4 Riconfigurabilità nello spazio: stato dell'arte

Negli ultimi anni la comunità scientifica ha rivolto particolare attenzione ai sistemi riconfigurabili. Nell'ambito spaziale sono stati eseguiti vari studi, soprattutto a livello accademico, con risultati incoraggianti. Da un'analisi dello stato dell'arte sono state individuate alcune pubblicazioni scientifiche, tra cui si riportano le più significative per i risultati ottenuti e per l'attinenza con la tecnologia in esame.

In "Characterization of Partial and Run-Time Reconfigurable FPGAs" (Emilio Fazzoletto) [11] è stato eseguito uno studio sulle performance dei dispositivi riconfigurabili, con particolare riguardo all'overhead introdotto dal processo di riconfigurazione e ai trade-off e limiti di questi sistemi. I risultati raccolti permettono di sfruttare al meglio le possibilità offerte da questi dispositivi, in particolare nel campo delle applicazioni real-time.

In "Enabling Technologies for Distributed Picosatellite Missions in LEO" (T. Vladimirova et al.) [12] viene presentata una architettura riconfigurabile basata su FPGA applicata al computer di bordo di pico-satelliti. In particolare l'articolo illustra come l'utilizzo della riconfigurabilità parziale possa rappresentare, nel campo delle costellazioni

di pico-satelliti in orbita bassa, una via per incrementare le funzionalità e la vita operativa delle piccole piattaforme satellitari. Grazie alla capacità di riconfigurabilità parziale dinamica, una costellazione di satelliti potrebbe infatti riconfigurarsi durante la fase operativa, garantendo l'aggiornamento delle funzionalità eseguite dal satellite stesso.

Tra i lavori più recenti si evidenziano le attività di ricerca e sviluppo del Fraunhofer Institute [31] e della NASA. Il progetto "Fast In-Orbit FPGA Reconfiguration via In-Band TM/TC" ha lo scopo di creare un OBC per applicazioni satellitari nel settore delle telecomunicazioni, il quale sarà testato in orbita nel 2020 a bordo del satellite tedesco *Heinrich Hertz*. Le tecnologie impiegate nel settore delle telecomunicazioni diventano spesso obsolete a causa della continua evoluzione degli standard satellitari. Dotando tali satelliti di FPGA riconfigurabili, e usando tali dispositivi per il coding, il decoding e il processing dei dati trasmessi sul canale di trasmissione, è possibile riconfigurare le radio stesse, così come i sistemi di processamento. In questo modo, sovradimensionando opportunamente le FPGA utilizzate, è possibile aumentare la vita di un satellite per telecomunicazioni.

Il lavoro pubblicato in "A Novel Fault Tolerant and Runtime Reconfigurable Platform for Satellite Payload Processing" da L. Sterpone et al. [13] offre un'analisi più approfondita sulle potenzialità dell'hardware riconfigurabile qualora utilizzato nel campo delle applicazioni spaziali. In questo caso gli autori analizzano anche le possibilità di sfruttare la riconfigurazione parziale al fine di aumentare l'affidabilità e la robustezza del dispositivo in ambiente irradiato, concentrandosi in particolare sulla mitigazione dei SEU e dei Multiple-Event Upset. Nonostante il buon livello di dettaglio con il quale il progetto è stato condotto e i significativi risultati ottenuti, la piattaforma descritta è molto complessa presentando una pluralità di CPU e FPGA, anche radhard, interconnesse.

In "An FPGA-based Radiation Tolerant SmallSat Computer System" (Brock J. LaMeres et al.) [14] viene presentato un OBC sviluppato in collaborazione con NASA nel progetto "Radiation Tolerant Computer Mission on the ISS (RTcMISS)". Lo scopo è di studiare la resistenza alle radiazioni di dispositivi auto-riconfiguranti, dotati di ridondanza architetturale, qualora implementati con FPGA Commercial Off-The-Shelf (COTS) e operanti in orbita bassa. Tale studio è attualmente in fase di svolgimento e potrà fornire dati importanti circa il livello di competitività dei computer di bordo basati su questo tipo di tecnologia.

China National Space Administration (CNSA) ha testato a bordo della sonda *Chang'e*

5-T1 un sistema di Guidance Navigation and Control (GNC) combinando il concetto di TMR su più livelli e di riconfigurabilità, come descritto nel paper "Architecture design for reliable and reconfigurable FPGA-based GNC computer for deep space exploration" (Yang MengFei et al.) [15]. L'orbita seguita dalla missione ha portato la sonda all'interno della sfera d'influenza lunare, quindi nello spazio profondo. Per questo motivo i dati riportati risultano essere particolarmente significativi. Mentre le unità non protette con questa soluzione hanno subito diversi errori, il sistema GNC non ha riportato nessun fault pur essendo realizzato con dispositivi commerciali non radhard, dimostrando la validità di questa tecnica.

CNSA sta inoltre progettando lo Space Solar Telescope, un satellite studiato per l'osservazione solare. Il paper "A Dynamically Partial-reconfigurable FPGA-based Architecture for Data Processing on Space Solar Telescope" (Zhuo Ruan et al.) [16] riporta lo studio effettuato su un prototipo di computer di bordo realizzato con componenti commerciali. Il telescopio utilizzerà sensori multispettrali, generando fino a 1728 Gigabyte di dati al giorno. Per processare efficacemente questi dati è stato previsto l'utilizzo di una FPGA in modo da sfruttare la concorrenzialità delle operazioni. Considerando che la sonda lavorerà in un ambiente fortemente irradiato dal vento solare, per aumentarne l'affidabilità verrà inoltre applicata la riconfigurabilità. Lo studio effettuato dimostra quindi che questa tecnologia può essere applicata anche in sistemi ad alta densità di dati, come nel caso dei satelliti di osservazione.

Risultano quindi evidenti i diversi vantaggi presentati dalla riconfigurabilità, quali un incremento dell'affidabilità e delle funzionalità delle piattaforme satellitari. Questo è un campo di ricerca molto promettente e diversi enti, sia industriali che accademici, stanno impiegando le loro risorse per esplorarne le possibilità.

Capitolo 3

Descrizione del sistema

Questo capitolo introduce gli obiettivi che la tesi si pone. A partire da questi viene elaborata una soluzione, arrivando quindi a descrivere il sistema sviluppato e la relativa architettura.

3.1 Obiettivi

L'ambiente estremo in cui i satelliti devono operare richiede di applicare tecniche radhard ai componenti elettronici di bordo. Questo comporta però diversi svantaggi, quali un aumento dei costi, del peso e degli ingombri. In particolare, questi ultimi risultano essere risorse critiche nella progettazione e realizzazione di piattaforme satellitari. Per questi motivi, i componenti radhard presentano performance inferiori rispetto ai COTS, i quali possono sfruttare a pieno le ultime tecnologie di realizzazione degli IC.
Poter utilizzare componenti commerciali in applicazioni spaziali porterebbe quindi ad un risparmio significativo su diversi fronti. È però necessario sviluppare una soluzione al problema delle radiazioni, in particolare i SEE, contro i quali i COTS non dispongono di alcuna protezione.
L'obiettivo di questa tesi è di progettare ed implementare un sistema basato su IC commerciali in grado di tollerare i SEU, con lo scopo finale di sviluppare una tecnologia che possa ridurre il costo dei computer di bordo dei satelliti mantenendo delle performance elevate.

3.2 Descrizione del sistema e dell'architettura sviluppata

Per raggiungere gli obiettivi posti è stato necessario individuare delle tecnologie che permettano a componenti commerciali di tollerare i soft errors. La soluzione individuata si applica alle FPGA e prevede di sfruttare la riconfigurabilità parziale dinamica per correggere i SEU. Questi errori possono colpire la configuration memory oppure i dati elaborati. Nel primo caso, la funzionalità implementata può cambiare portando a comportamenti imprevisti dell'IC. Questa situazione può essere corretta utilizzando la riconfigurabilità parziale per riscrivere la parte di configuration memory colpita. Per effettuare la correzione è possibile utilizzare una *golden copy*, ovvero con una copia originale non fallata del bitstream.
Nel caso in cui vengano colpiti i dati elaborati, gli errori introdotti possono essere mascherati implementando un'architettura di tipo TMR. Questa inoltre permette di individuare la posizione del fault tramite l'algoritmo di maggioranza, consentendo di applicare efficacemente la riconfigurabilità parziale. Infatti, i moduli che compongono il sistema TMR possono essere implementati come RM. In questo modo è possibile agire solamente sul modulo che ha subito il SEU, riconfigurandolo con la relativa golden copy e quindi riparandolo, senza interrompere l'elaborazione nel resto del dispositivo grazie alla tripla ridondanza e alla riconfigurabilità dinamica.
La soluzione individuata fa quindi un uso congiunto della riconfigurabilità parziale dinamica e della ridondanza TMR, con l'obiettivo di mascherare, individuare e correggere eventuali fault.
Al costo di una maggiore complessità dell'architettura, anche i voter vengono triplicati e resi riconfigurabili. Questo permette di aumentare decisamente la robustezza del sistema. È quindi necessario analizzare gli output dei voter per estrarre il dato correttamente elaborato e mascherato dai fault. Per questo compito viene impiegato un modulo chiamato *voter controller*, il quale permette di individuare un eventuale errore nei moduli riconfigurabili.
L'indicazione di quale RM ha subito un fault viene quindi utilizzata da un *reconfiguration controller*, che si occupa di gestire il processo di riconfigurazione, sovrascrivendo la parte di configuration memory relativa al modulo che ha subito il fault con il relativo partial bitstream. I file di configurazione degli RM vengono conservati in una memoria esterna detta *golden copy memory*.
L'architettura è stata divisa in due sezioni:

- **Sezione TMR:** implementa i moduli triplicati e riconfigurabili (moduli funzionali e voter);

- **Sezione di controllo:** controlla lo stato della sezione TMR (voter controller) e, se necessario, esegue la riconfigurazione (reconfiguration controller).

La sezione TMR è implementata su una FPGA con capacità di riconfigurazione parziale dinamica, mentre la parte di controllo su CPU. È importante notare che quest'ultima sezione può essere implementata anche su FPGA. Un processore risulta però essere più facilmente programmabile in quanto l'uso di un linguaggio di programmazione SW permette un'astrazione maggiore rispetto ad un linguaggio di descrizione HW. Inoltre, in questa applicazione le migliori performance della FPGA nel calcolo concorrente non sono rilevanti, in quanto non è richiesta un'elevata potenza computazionale.
L'architettura finale è riportata in Figura 3.1. Per una descrizione dettagliata dei moduli e delle interconnessioni si rimanda al Capitolo 4. In questa implementazione la CPU si occupa inoltre di prelevare i dati da elaborare dalla memoria di sistema e di quindi salvare i risultati. In futuri sviluppi del sistema, questa operazione aggiuntiva potrà essere effettuata direttamente nella sezione di TMR, riducendo ulteriormente la complessità della sezione di controllo.

Per meglio comprendere il funzionamento dell'architettura si segua il flusso dei dati. La CPU preleva i valori in ingresso dalla *data memory* e li trasferisce ai moduli funzionali nella FPGA. Una volta elaborati, essi vengono inviati ai voter, i quali applicano un algoritmo di maggioranza per individuare i fault presenti nei moduli a monte. I risultati dei voter vengono quindi acquisiti dalla CPU, la quale li analizza via software. Quest'ultimo implementa due funzioni principali: voter controller e reconfiguration controller. Il voter controller identifica eventuali errori nella sezione TMR, segnalandone la locazione al reconfiguration controller il quale corregge il modulo fallato tramite una riconfigurazione parziale dinamica. Inoltre il controller estrae il risultato dell'elaborazione dei moduli funzionali mascherando gli eventuali errori. I dati così processati vengono quindi salvati nella data memory.

L'architettura sviluppata presenta dei *single point of failure*, in particolare nella sezione di controllo. Quest'ultima deve quindi essere implementata utilizzando tecniche radhard convenzionali. Considerando però la limitata potenza computazionale necessaria per eseguire le funzioni richieste, un microprocessore esterno di ridotte dimensioni (e quindi di costo contenuto) può risultare sufficiente. Un altro importante punto critico è rappresentato dalla golden copy memory. È infatti fondamentale che i partial bitstream relativi agli RM non vengano corrotti dai SEE, altrimenti l'intero sistema perderebbe la sua capacità di tollerare gli effetti delle radiazioni. Pertanto, questa memoria deve essere

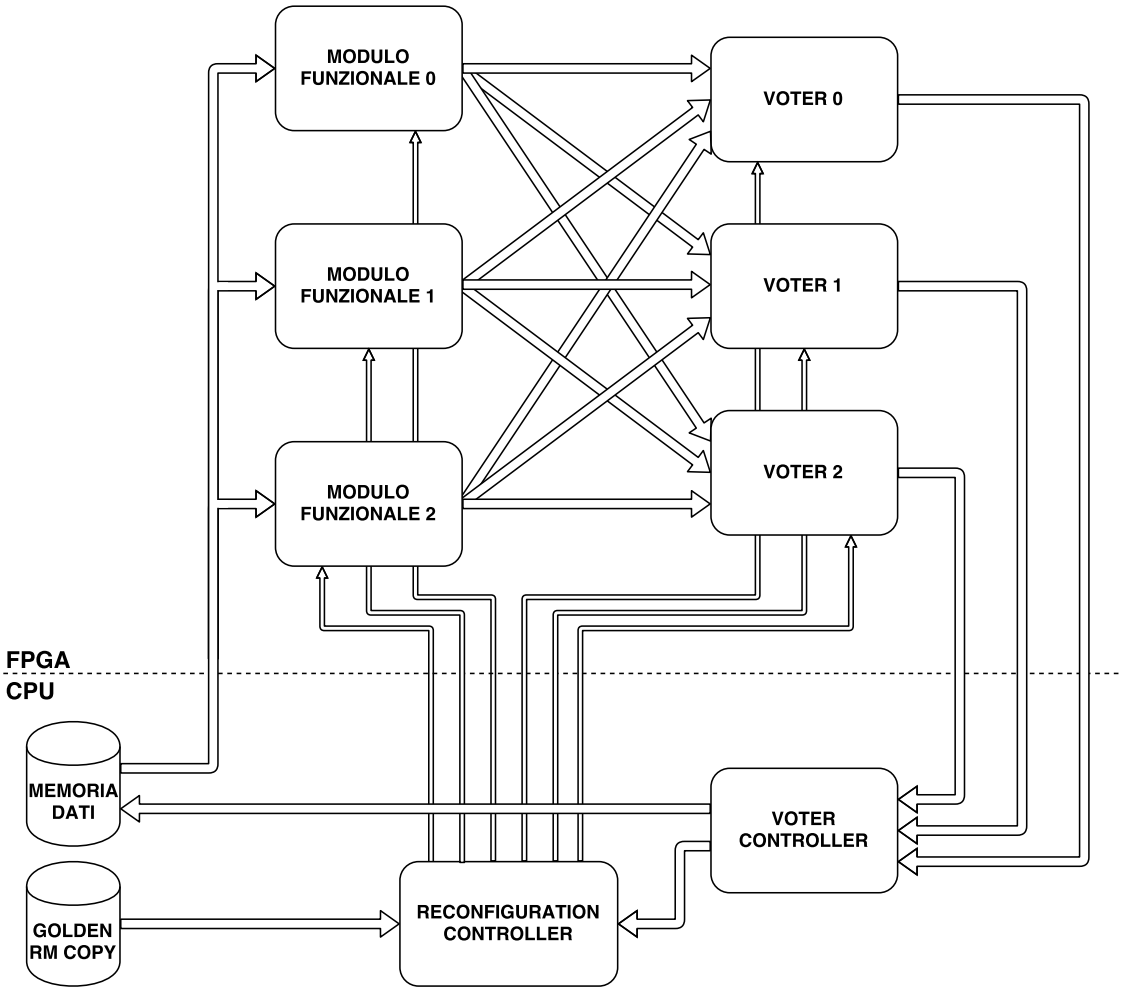

Figura 3.1 *Architettura del sistema. In alto è rappresentata la sezione TMR, mentre in basso trova posto la sezione di controllo*

realizzata con tecnologie radhard. L'impatto sul costo totale del sistema è comunque limitato, in quanto questa memoria deve contenere solamente pochi Megabyte (MB) (fino a 4 MB nel caso dell'implementazione presentata nel capitolo 4).
Un limite dell'architettura proposta è dato dall'assenza di un meccanismo di ripristino del contesto, in mancanza del quale non è possibile implementare degli algoritmi sequenziali all'interno dei moduli funzionali. Una riconfigurazione causerebbe infatti la perdita delle informazioni memorizzate nei registri del modulo che ha subito il fault. Una soluzione consiste nel copiare queste informazioni dagli altri moduli al termine della riconfigurazione, ripristinando così lo stato interno.

Capitolo 4

Implementazione

Questo capitolo descrive le risorse hardware e software utilizzate per implementare il sistema studiato. Vengono inoltre descritti i dettagli implementativi dell'architettura, sia dal lato HW sia da quello SW.

4.1 Xilinx Zynq-7020 SOC

Il sistema è stato sviluppato utilizzando una *development board*, la quale permette di concentrare il lavoro sull'implementazione dell'architettura piuttosto che sullo sviluppo dell'hardware. È stata impiegata la scheda *Zedboard* (Figura 4.1), realizzata da *Avnet* [35]. Questa include un System On Chip (SOC) *Zynq-7020* [17] realizzato da Xilinx. A supporto di questo IC sono presenti sulla board 512 MB di memoria Random Access Memory (RAM), 256 MB di memoria Flash e diverse interfacce quali Universal Serial Bus e un lettore di memorie SD-Card.

Zedboard è stata scelta per il suo SOC, ovvero un IC che integra nello stesso chip i diversi componenti di un sistema di elaborazione, quali ad esempio un processore, un controller delle memorie e i circuiti di input/output. In particolare, Zynq è composto da un Processing System (PS) e una sezione di logica programmabile chiamata Programmable Logic (PL), la quale può essere riconfigurata parzialmente e dinamicamente. Un diagramma a blocchi semplificato di Zynq è mostrato in Figura 4.2.

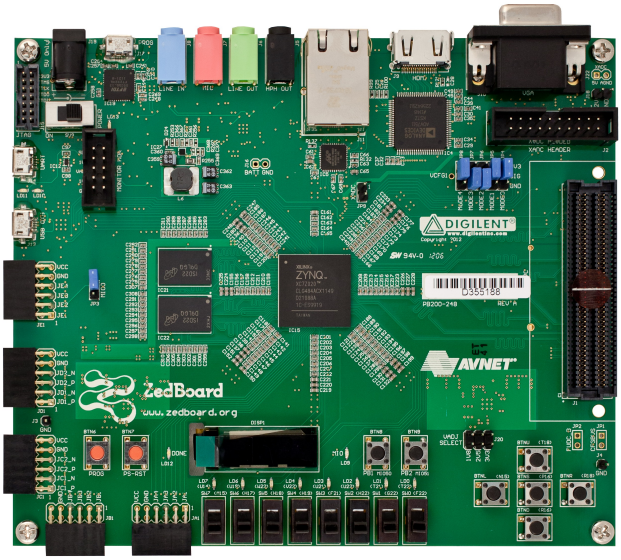

Figura 4.1 *Zedboard, al centro della quale è posizionato il SOC Zynq. Fonte: [37]*

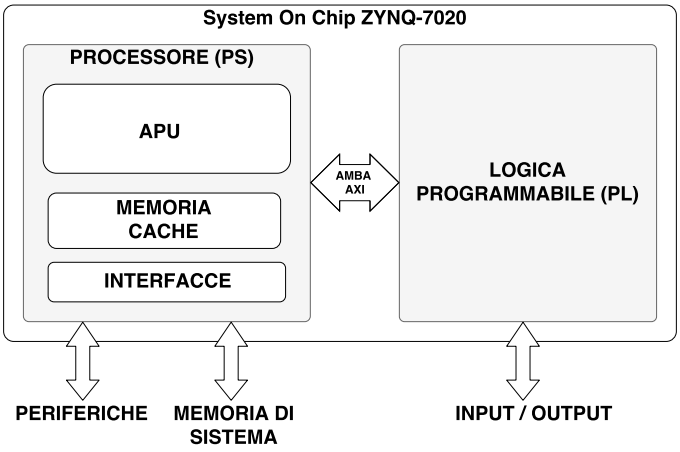

Figura 4.2 *Schema a blocchi semplificato del SOC Zynq*

La combinazione di PS e PL risulta essere molto interessante in quanto permette ad un'applicazione software di sfruttare efficacemente le potenzialità offerte dall'hardware. Nell'architettura esposta in questa tesi, la sezione TMR viene implementata nel PL mentre la sezione di controllo sfrutta il PS.
In particolare, il PS è composto da:

- Application Processing Unit (APU);
- Interfacce verso le periferiche;

- Memoria cache;
- Interfacce verso le memorie esterne;
- Circuiti di gestione del clock.

APU è composta a sua volta da un processore dual core ARM [36] *Cortex-A9*[18], una Memory Management Unit, una memoria cache di livello 1 e un Digital Signal Processor (DSP) *Neon* [17]. È inoltre presente una cache di livello 2 e una On Chip Memory di 256 kB connesse tramite una Snoop Control Unit ai processori ARM. Le interfacce verso le periferiche del PS comprendono porte Serial Peripheral Interface, Universal Asynchronous Receiver-Transmitter e General Purpose I/O.
Il PL è basato sulla FPGA *Artix-7* [19] di Xilinx. Questa permette la riconfigurazione parziale dinamica, la quale può essere gestita sia dalla FPGA stessa, sia dal PS. Sono inoltre presenti dei moduli DSP e Block RAM da 36kb. I primi possono essere impiegati per eseguire calcoli aritmetici ad alte performance, mentre utilizzando i secondi è possibile implementare RAM, Read-Only Memory e buffer First In First Out.
PS e PL sono interconnessi con delle interfacce standard Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) [20], le quali permettono collegamenti ad alta banda e bassa latenza. Lo standard AMBA (versione 4) [20] definisce due gruppi di interfacce: AXI Coherency Extensions (ACE) e AXI. Il protocollo ACE è progettato per mantenere la coerenza delle cache nei sistemi a multiprocessore, utilizzando un'interfaccia a cinque canali. AXI viene utilizzato per trasferire dati all'interno di un IC ed è a sua volta composto da AXI4, AXI4-Lite e AXI4-Stream:

- **AXI4** è pensato per interfacce del tipo memory mapped e permette dei burst fino a 256 cicli di trasferimento dei dati;
- **AXI4-Lite** implementa un'interfaccia memory mapped a singola transazione. Il burst non è permesso e questo si traduce in una soluzione leggera per il trasferimento dei segnali di stato e controllo fra i diversi elementi di un sistema;
- **AXI4-Stream** permette un burst illimitato e può quindi essere efficacemente impiegato quando è necessario trasferire grandi quantità di dati.

Tutte e tre queste interfacce possono essere utilizzate per trasferire dati fra PS e PL.

4.2 Suite Vivado

Xilinx mette a disposizione degli sviluppatori una potente suite software chiamata *Vivado Design Suite*, composta da tre SW:

- **Vivado Integrated Development Environment (IDE)**;
- **Software Development Kit (SDK)**;
- **Vivado High Level Syntesis (HLS)**.

4.2.1 Vivado IDE

Vivado IDE [21] assiste lo sviluppatore in ogni fase della programmazione di una FPGA. Questo processo può essere diviso in quattro step principali:

1. **Descrizione dell'HW:** l'architettura da implementare viene descritta utilizzando un linguaggio di descrizione quale VHDL oppure Verilog. È anche possibile utilizzare un editor grafico. In quest'ultima modalità (chiamata block diagram), le entità che compongono il sistema sono visualizzate come blocchi ed è possibile tracciare le connessioni semplicemente con l'uso del mouse;
2. **Sintesi:** l'architettura viene sintetizzata, ovvero il design viene convertito in una netlist di porte logiche;
3. **Implementazione (o Place & Route):** ad ogni porta logica della netlist viene individuata una locazione nelle risorse della FPGA. Inoltre vengono tracciate le connessioni fra le risorse utilizzate.
4. **Generazione del bitstream:** infine viene generato il file di configurazione con il quale è possibile programmare la FPGA.

Vivado IDE permette inoltre di simulare il design prima di procedere con la sintesi dell'architettura. In questo modo è possibile ridurre il tempo richiesto nello sviluppo del sistema, in quanto sintesi ed implementazione richiedono molto tempo essendo processi molto complessi.

4.2.2 SDK

SDK è basato su *Eclipse*, un IDE usato per compilare e debuggare software. Questo programma permette di sviluppare il codice per il PS dello Zynq. Offre inoltre la possibilità di utilizzare un terminale seriale, utile per comunicare con il SOC dal computer. Un progetto su SDK può essere iniziato a partire da un progetto di Vivado IDE: in questo modo il SW in esecuzione sul processore può avere facile accesso alle interfacce HW dell'architettura implementata in FPGA, in quanto i relativi driver vengono generati in maniera automatica.

4.2.3 Vivado HLS

Vivado HLS [22] permette al progettista di descrivere un modulo in VHDL o in Verilog partendo da una funzione scritta in un linguaggio di programmazione a più alto livello, quale il *C* o il *C++*. Vivado HLS effettua la traduzione fra i due linguaggi, permettendo di ridurre drasticamente lo sforzo richiesto per lavorare con le FPGA. È inoltre possibile ottimizzare il design tramite l'uso di direttive, ad esempio imponendo l'uso di pipeline.

4.3 Creazione di un progetto riconfigurabile

La sezione PL di Zynq permette di essere riconfigurata parzialmente e dinamicamente. Il processo di riconfigurazione può essere gestito sia dal PL stesso attraverso un *Partial Reconfiguration Controller* Intellectual Property (IP) core [23] appositamente progettato oppure dal PS, come nel caso dell'architettura in esame. Nel primo caso è possibile sfruttare l'interfaccia Internal Configuration Access Port (ICAP), la quale permette alla logica programmabile di accedere alla sua stessa configuration memory, mentre nel secondo caso è possibile utilizzare l'interfaccia Processor Configuration Access Port (PCAP). L'accesso alla memoria di configurazione della FPGA è condiviso fra queste due porte. Di default PCAP è attiva, ma è sufficiente agire sul bit di controllo PCAP_PR (bit 27) del registro devc.CTRL del PS (indirizzo 0xF8007000) per garantire l'accesso a ICAP, come mostrato in Figura 4.3.

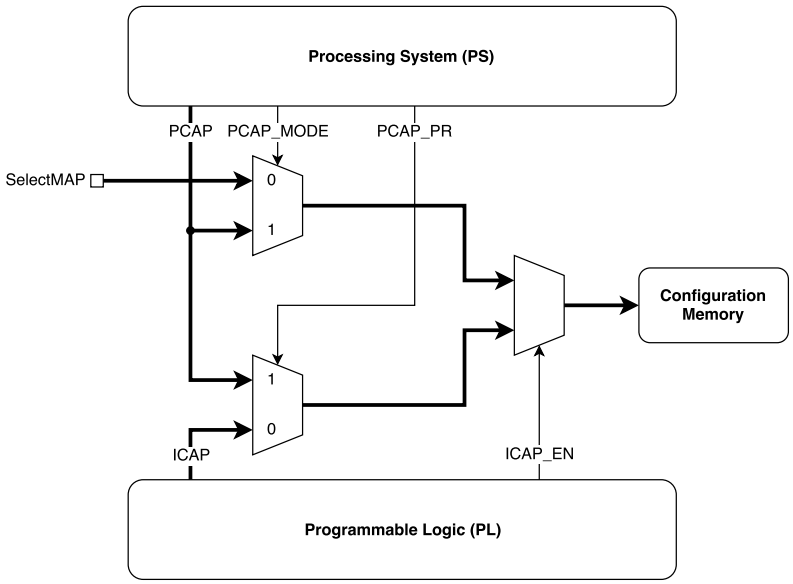

Figura 4.3 *Accesso alla configuration memory [25]*

4.3.1 Flusso di progetto in Vivado IDE

Per poter sfruttare la capacità di riconfigurazione parziale e dinamica offerta dal PL dello Zynq è necessario seguire un determinato flusso di progetto in Vivado IDE [24]. In particolare, la versione 2017.3 (versione usata nello sviluppo di questa tesi) mette a disposizione del progettista un wizard che permette di impostare facilmente le RP e i relativi RM. Gli step indicati di seguito si riferiscono a Vivado IDE 2017.3
Una volta descritta l'architettura che si desidera implementare nella FPGA, è possibile abilitare il progetto alla riconfigurazione parziale selezionando la relativa voce dal menu *Tools*. Ciò rende disponibile l'uso del wizard, ma è prima necessario creare almeno una *partition definition* (una RP) selezionando un modulo del design che si vuole rendere riconfigurabile. Il programma chiede quindi di assegnare un nome alla RP appena creata come mostrato in Figura 4.4. Ora è possibile attivare il wizard. I vari step del wizard

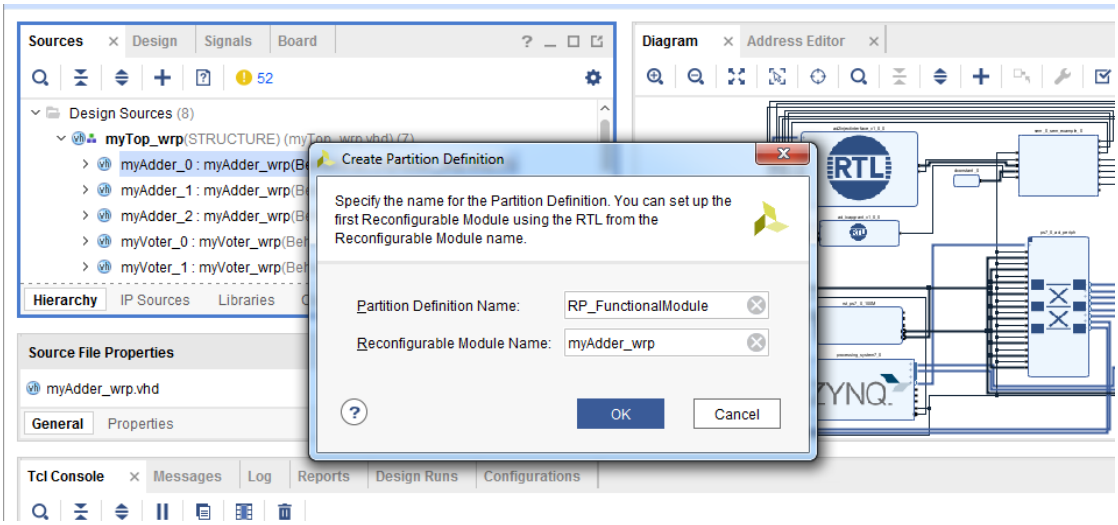

Figura 4.4 *Dettaglio del processo utilizzato per impostare la riconfigurabilità: creazione di una partition definition*

sono riportati di seguito:

1. **Edit Reconfigurable Modules:** in questo step è possibile associare i moduli riconfigurabili progettati alle relative RP (Figura 4.5);

2. **Edit Configurations:** (Figura 4.6) permette di creare delle configurazioni. Queste consentono di specificare quali RM verranno effettivamente utilizzati per ogni RP presente del design, in modo tale da poterne generare i relativi partial bitstream. È importante notare che queste configurazioni non impongono alcun vincolo alla modalità con cui le partizioni riconfigurabili devono essere utilizzate. Le RP possono

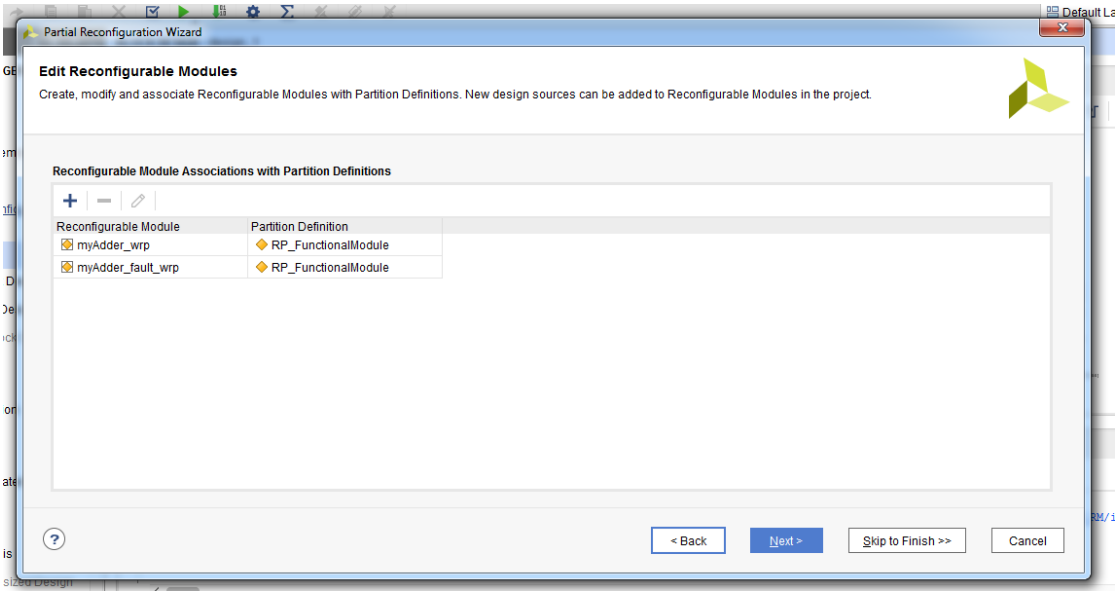

Figura 4.5 *Dettaglio del processo utilizzato per impostare la riconfigurabilità: associazione di degli RM alle relative RP. Nell'immagine sono stati assegnati due moduli alla stessa partizione*

essere infatti riconfigurate indipendentemente l'una dall'altra. Tuttavia, una delle configurazioni create in questo step verrà selezionata come default nella fase successiva e utilizzata per generare il bitstream dell'intera architettura caricato durante l'inizializzazione della FPGA. È inoltre possibile selezionare l'opzione *greybox*, la quale implementa nella partizione scelta un RM senza alcun contenuto;

3. **Edit Configuration Runs:** (Figura 4.7) è possibile indicare in quale ordine le configurazioni create nella fase precedente verranno sintetizzate ed implementate. In particolare, i risultati dell'implementazione della configurazione di default verranno utilizzati dalle altre configurazioni per permettere di rispettare i vincoli fisici sulle connessioni in ingresso/uscita dalle RP.

Una volta concluso il wizard è possibile procedere con la sintesi del progetto, al termine della quale possono essere imposti i constraints fisici relativi alle partizioni riconfigurabili. Queste possono essere dimensionate direttamente sulla rappresentazione del die dell'IC, selezionando nella finestra *Netlist* gli RM e utilizzando lo strumento *draw Pblock* (Figura 4.8). Quando tutte le RP sono state create, è consigliabile eseguire *report Design Rule Check* dal menu Tools. In particolare la regola relativa alla riconfigurazione parziale deve essere attivata. Se non viene riportato nessun errore, è possibile eseguire l'implementazione del design. Questo implementa ogni RM indicato nel wizard. Infine la generazione dei bitstream crea i seguenti file:

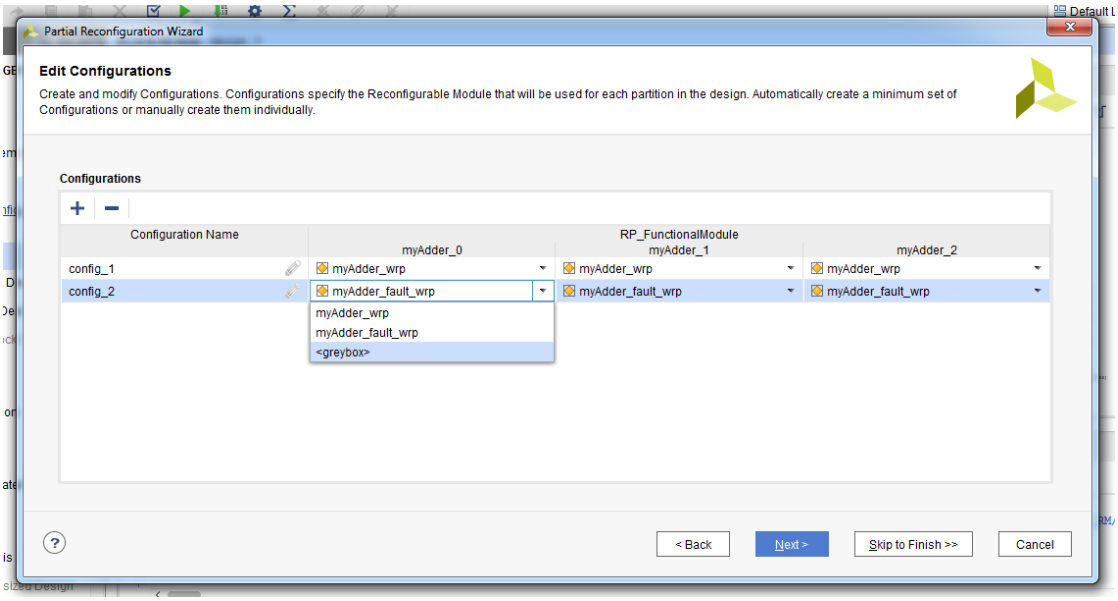

Figura 4.6 *Dettaglio del processo utilizzato per impostare la riconfigurabilità: creazione delle diverse configurazioni*

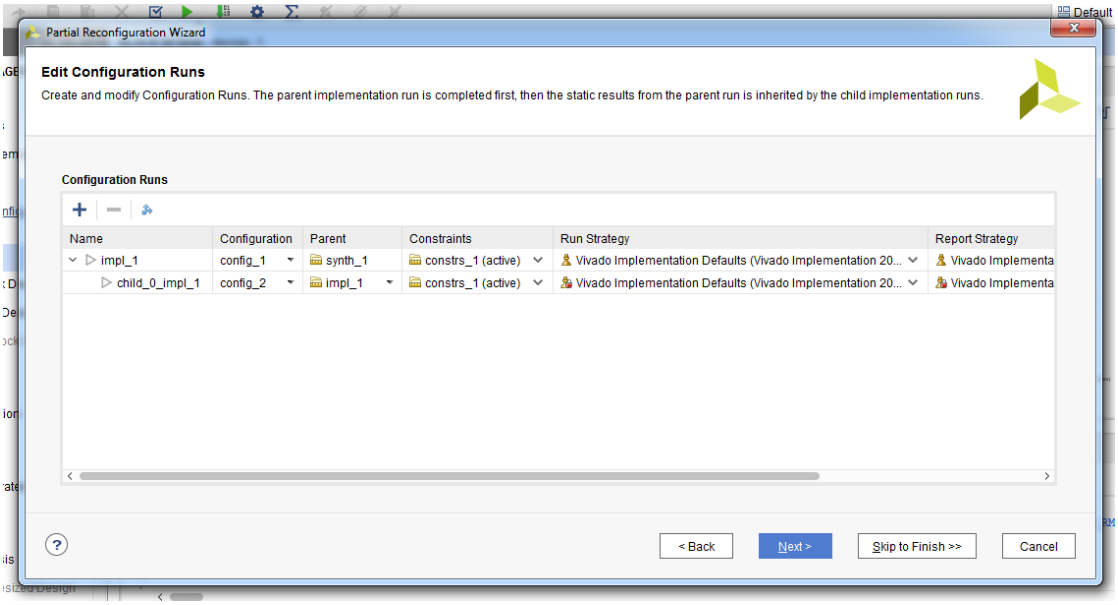

Figura 4.7 *Dettaglio del processo utilizzato per impostare la riconfigurabilità: impostazione dei settings per sintesi ed implementazione*

- Un bitstream completo, utilizzabile per programmare l'intera FPGA durante l'inizializzazione. Per generare questo file viene utilizzata la configurazione di defualt specificata nel wizard;

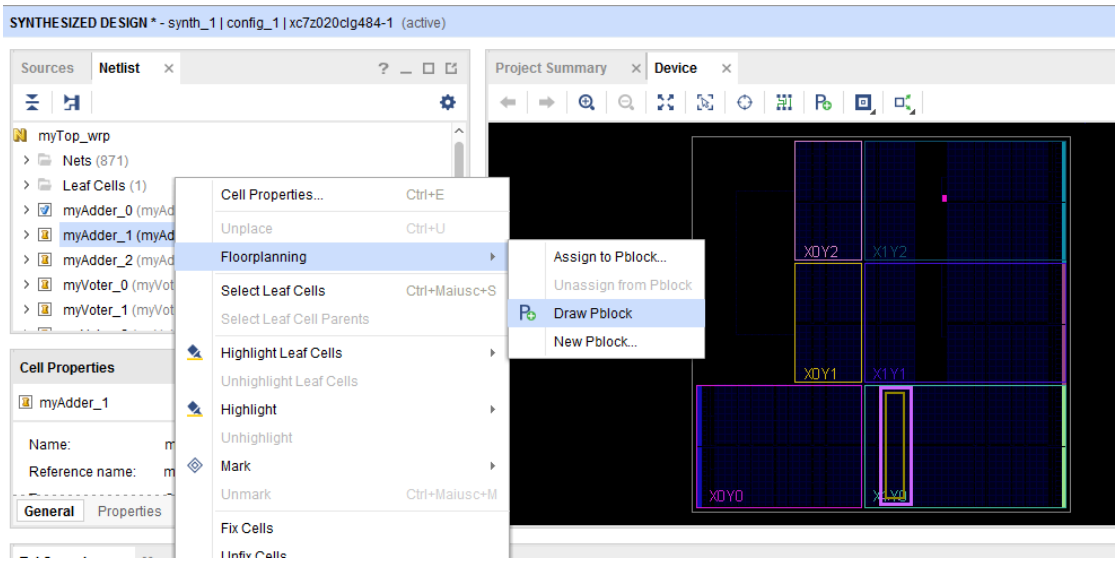

Figura 4.8 *Dettaglio del processo utilizzato per impostare la riconfigurabilità: dimensionamento fisico delle RP. In particolare è visibile in violetto in basso a destra, sulla rappresentazione del die dello Zynq, una partizione già creata*

- I partial bitstream relativi agli RM precisati nel wizard, utilizzabili per riconfigurare le RP.

I bitstream generati (con estensione *.bit*) possono essere utilizzati per programmare la FPGA attraverso la suite Vivado, ma non possono essere trasferiti direttamente nella configuration memory (ad esempio dal PS) in quanto contengono un header e una speciale formattazione. Risulta quindi necessario convertirli in file con estensione *.bin*. Questo può essere fatto in Vivado IDE con il seguente comando TCL:

```
write_cfgmem -force -format BIN -interface SMAPx32
    -disablebitswap -loadbit "up 0 myRM_partial.bit" myRM.bin
```

4.3.2 Uso della riconfigurabilità in SDK

La riconfigurazione può essere effettuata sotto il controllo del PS, come richiesto dall'architettura sviluppata in questa tesi. Questo è possibile utilizzando l'interfaccia PCAP, la quale permette al processore di accedere direttamente alla configuration memory del PL e di trasferire i partial bitstream utilizzando un Direct Memory Access (DMA) controller. È quindi necessario sviluppare un programma in SDK per gestire questo processo, i cui steps possono essere riassunti come segue:

- Nella fase di inizializzazione, i partial bitstream salvati nella golden copy memory vengono traferiti nella memoria di sistema con lo scopo di aumentare la velocità di accesso a questi dati e quindi ridurre il tempo necessario alla riconfigurazione. Inoltre l'interfaccia PCAP viene attivata;

- Quando viene richiesta una riconfigurazione, PCAP utilizza il DMA controller per trasferire il bitstream del RM da riconfigurare dalla memoria di sistema alla configuration memory;

- Il processo di riconfigurazione si conclude quando termina il trasferimento dei dati. Questo viene controllato utilizzando la tecnica del polling.

Xilinx riporta un problema dovuto al ritardo di propagazione del clock di riferimento all'interno delle partizioni riconfigurabili [38]. Per risolvere questo problema è richiesto di mantenere attivo il reset del RM per ulteriori tre millisecondi al termine della riconfigurazione, in modo da permettere al segnale di clock di assestarsi. Se questa soluzione non viene applicata, i moduli potrebbero non riconfigurarsi correttamente. È stato quindi introdotto il ritardo richiesto al termine del ciclo di polling.

4.4 Moduli funzionali

I moduli funzionali possono implementare qualunque algoritmo, considerando che, in tal senso, l'architettura proposta risulta essere generica. Al livello di sviluppo raggiunto in questa tesi è assente un meccanismo di ripristino del contesto interno dei moduli, escludendo la possibilità di implementare algoritmi sequenziali. Questo limite verrà rimosso nei futuri sviluppi del sistema. Possibili esempi di funzionalità sono algoritmi di compressione di immagini oppure di controllo dell'assetto di un satellite. Nei moduli funzionali è possibile inserire anche interi microprocessori, realizzando così un OBC tollerante alle radiazioni realizzato con componenti commerciali.
In generale, questi moduli possono avere diverse interfacce per i dati in ingresso e in uscita. Per diminuire la complessità del sistema è preferibile ridurre il numero degli output, semplificando di conseguenza anche i voter che dovranno analizzare un minor numero di dati.
Per validare l'architettura, la quale risulta essere indipendente dal modulo implementato, nell'ambito di questa tesi sono stati utilizzati dei moduli funzionali che eseguono l'elevamento a potenza del dato in ingresso tramite una serie di moltiplicazioni. Considerando che nell'architettura sviluppata è assente un meccanismo di ripristino del contesto interno ai moduli, è infatti possibile implementare solamente funzioni combinatorie. Per

raggiungere un adeguato utilizzo delle risorse del PL sono quindi stati impiegati 10 moltiplicatori a 32 bit. Per interfacciarsi con il PS, è stata utilizzata un'interfaccia AXI4-Lite e un registro memory mapped da 32 bit, nel quale viene scritto il dato in ingresso. Questa interfaccia permette un semplice metodo di trasferimento dei dati da elaborare, ma può rappresentare un collo di bottiglia in caso di grandi quantità di informazioni da trasferire. Implementazioni future del sistema potranno usufruire dell'interfaccia AXI4-Stream per aumentare il data rate. Il risultato dell'elaborazione è presentato su un output parallelo a 32 bit. È inoltre previsto un segnale di *valid*, utilizzato per permettere ai voter di campionare correttamente il risultato e quindi implementare un handshake elementare. La Figura 4.9 riporta le interfacce implementate.

Figura 4.9 *Interfacce implementate dai moduli funzionali*

4.5 Voter

I voter ricevono in ingresso i dati elaborati dai moduli funzionali. Dispongono quindi di tre gruppi di ingressi. Ogni gruppo può essere formato da una o più interfacce, in base al numero di output di ogni functional module. I dati in ingresso vengono analizzati applicando l'algoritmo di maggioranza descritto in 2.3.1, il quale consiste in una serie di confronti bitwise. Queste operazioni generano infine i segnali *Data Out, Fault Indication* e *Channel Indication*:

- **Data Out**: dato elaborato dai moduli funzionali una volta mascherato l'eventuale fault. In caso di fault multipli l'output viene messo a 0 in quanto non è possibile mascherare gli errori;
- **Fault Indication**: segnale che viene asserito nel caso in cui venga rilevato un fault;
- **Channel Indication**: segnale che, in caso di errore, indica quale dei moduli funzionali ha subito il fault. In caso di fault multipli viene generato uno speciale codice d'errore.

I voter implementati consentono quindi di mascherare, rivelare e localizzare i fault. La tabella 4.1 ne descrive la logica, mentre la Figura 4.10 riporta i segnali di ingresso e di uscita.

Le interfacce di ingresso dei voter implementati sono composte da degli input paralleli su

IN 0	IN 1	IN 2	Data Out	Fault Indication	Channel Indication
A	A	A	A	0	0
A	A	B	A	1	2
A	B	A	A	1	1
A	B	B	B	1	0
B	A	A	A	1	0
B	A	B	B	1	1
B	B	A	B	1	2
B	B	B	B	0	0
A	B	C	0	1	codice d'errore

Tabella 4.1 *Tabella logica di un voter implementato nell'architettura. A, B e C rappresentano dei generici valori di ingresso*

32 bit e da un segnale di valid. Quest'ultimo viene asserito dai moduli funzionali quando il dato elaborato è valido, permettendo quindi ai voter di effettuare un campionamento corretto. I valori generati dall'analisi degli ingressi sono quindi trasferiti al PS utilizzando una interfaccia AXI4-Lite e tre registri memory mapped da 32 bit, come riportato in tabella 4.2. Come nel caso dei moduli funzionali, questa interfaccia può rappresentare un collo di bottiglia in caso di grandi quantità di dati da trasferire. Implementazioni future del sistema potranno aumentare il data rate impiegando delle interfacce AXI4-Stream.

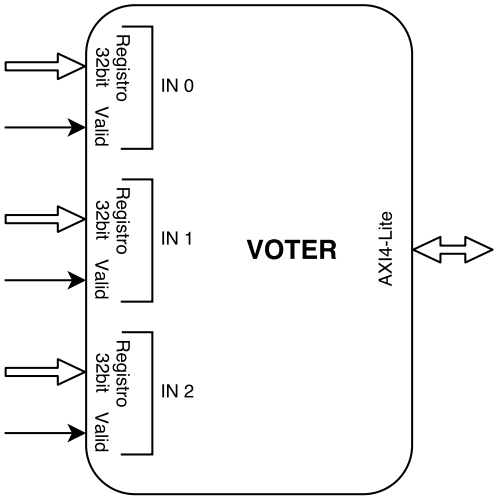

Figura 4.10 *Interfacce implementate dai voter*

Indirizzo	Registro
Indirizzo_base + 0	Data Out
Indirizzo_base + 4	Fault Indication
Indirizzo_base + 8	Channel Indication

Tabella 4.2 *Mappatura dei registri di interfaccia dei voter*

4.6 Implementazione della sezione TMR

La sezione TMR dell'architettura è stata implementata utilizzando il software Vivado IDE e seguendo gli step descritti in 4.3 per creare un progetto che permetta la riconfigurabilità. Per le interfacce con il PS si è fatto uso dell'editor grafico (creando un *block design*), mentre i moduli funzionali e i voter sono stati istanziati assieme al block design all'interno di un file con livello gerarchico superiore (chiamato *TOP.vhd*), come schematizzato in Figura 4.11. La necessità di un livello gerarchico superiore all'interno del quale implementare direttamente i moduli riconfigurabili è dovuta alla richiesta del partial reconfiguration wizard di non utilizzare come RM dei moduli presenti all'interno di un block diagram [24].

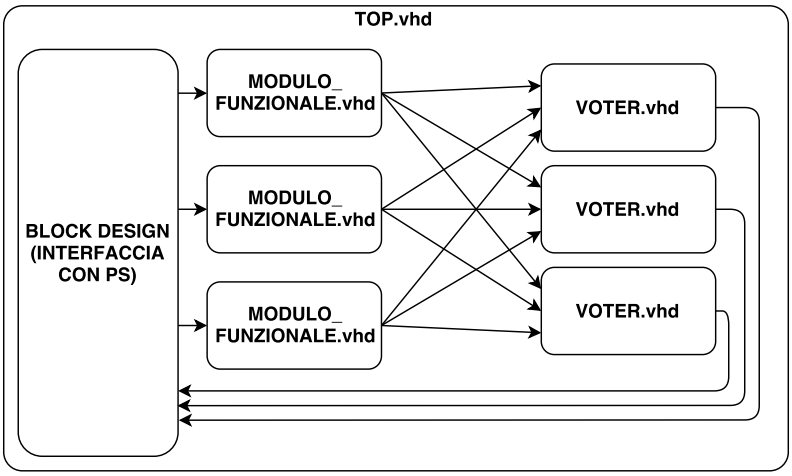

Figura 4.11 *Organizzazione del file TOP.vhd*

Nel block design è possibile interfacciare facilmente le due sezioni di PS e PL tramite l'uso dell'IP core *Zynq7 processing system* [26]. Vivado IDE permette di automatizzare alcune funzioni quando si lavora su un block design, ad esempio la creazione dei segnali di reset oppure la connessione delle interfacce AXI. Infatti, per poter connettere più interfacce slave (i moduli funzionali e i voter) ad un solo master (in questo caso rappresentato

dal PS) viene automaticamente instanziato un IP core chiamato *AXI Interconnect* [27], il quale permette di gestire le connessioni. Sono state quindi create sei porte AXI4-Lite in output al block design, le quali sono collegate ai moduli funzionali e ai voter. In Figura 4.12 è mostrato il block design implementato. In basso sono visibili gli IP core Zynq7 processing system, AXI Interconnects e il modulo per la gestione del reset. A destra è possibile notare le interfacce AXI, connesse all'esterno del block diagram. In alto sono invece presenti i moduli utilizzati per il fault injection, descritti nel Capitolo 5.2.

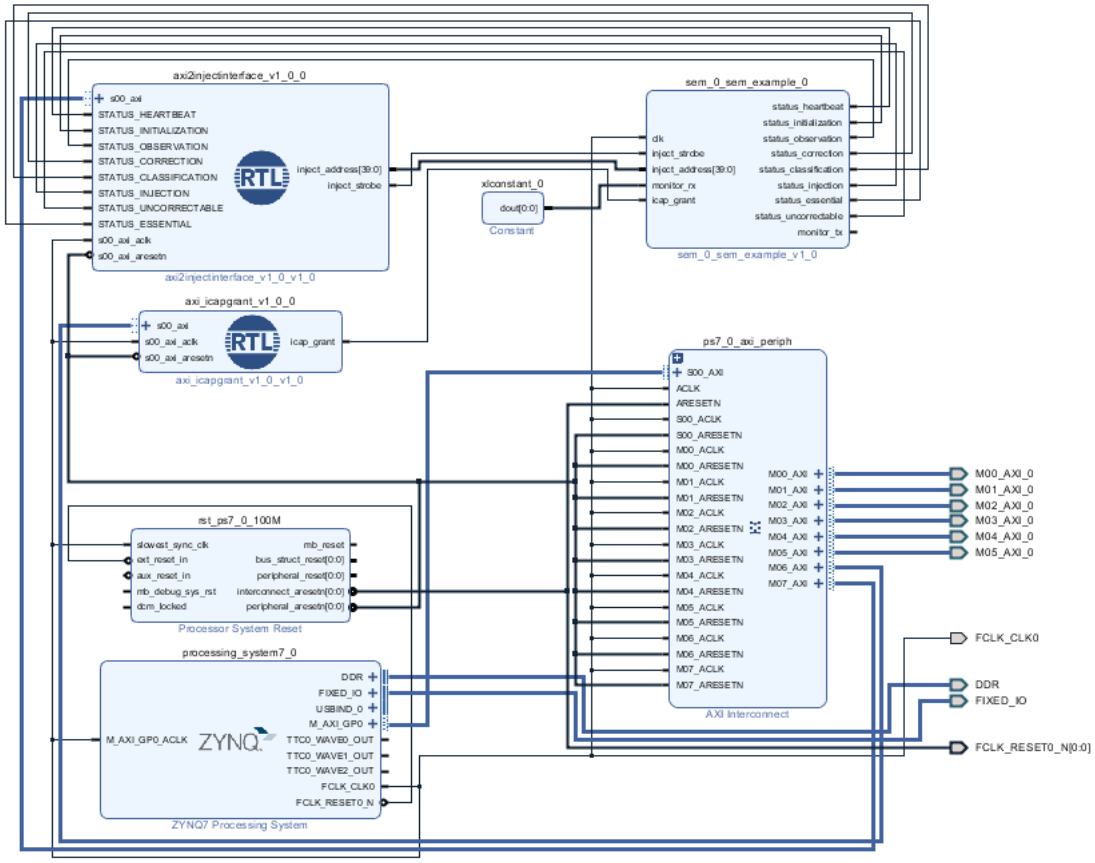

Figura 4.12 *Block design usato per descrivere le interconnessioni fra il PS e la sezione TMR*

Dopo aver eseguito la sintesi del file TOP.vhd sono state definite le sei RP. Il PL dello Zynq-7020 è suddiviso in sei domini di clock. Per ogni modulo funzionale ne è stato allocato uno intero, mentre le tre partizioni dei voter sono state inserite in un unico dominio, essendo queste di dimensioni sufficientemente ridotte. È quindi stata avviata la fase di implementazione, il cui risultato è visibile nella Figura 4.13. Questa immagine rappresenta il die dello Zynq. In particolare è possibile distinguere in alto a destra un

rettangolo nero, all'interno del quale è allocato il PS. La rimanente area in blu è occupata dal PL. In verde sono rappresentate le risorse utilizzate per i moduli funzionali, mentre in rosso quelle per i voter. Al centro è infine presente un'ulteriore partizione evidenziata in giallo, questa non configurabile, all'interno della quale sono implementate le risorse necessarie ad eseguire la tecnica di fault injection.

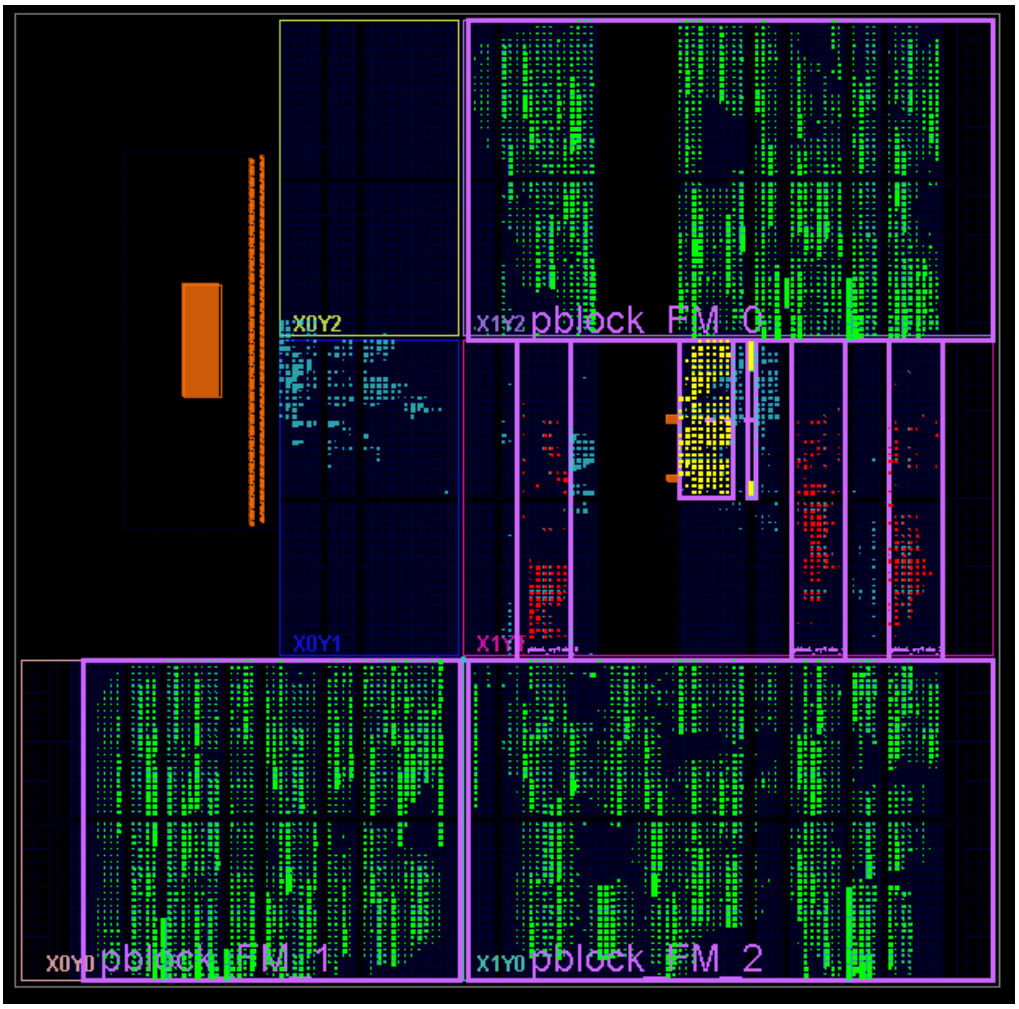

Figura 4.13 *Rappresentazione del die dello Zynq. In verde sono evidenziate le risorse del PL utilizzate per i moduli funzionali, mentre in rosso quelle per i voter*

Al termine dell'implementazione sono stati generati i bitstream necessari a programmare il PL. In particolare, nel caso dello Zynq-7020, il processo genera un bitstream di 3.85 MB necessario a programmare l'intera FPGA, e un partial bitstream per ogni RP. Le dimensioni di questi ultimi dipendono dalle dimensioni fisiche delle partizioni

riconfigurabili. Nell'architettura implementata i file di configurazione dei voter occupano 205 Kilobyte (KB), mentre quelli dei moduli funzionali 1327 KB, 1120 KB e 1497 KB, rispettivamente per i moduli funzionali 0, 1 e 2.
Il progetto è stato inoltre esportato verso SDK, per permettere al SW eseguito dal processore di interfacciarsi agevolmente con la sezione TMR creando automaticamente i driver necessari.

4.7 Descrizione della sezione di controllo

La funzione della sezione di controllo è l'elaborazione gli output generati dai voter e, in caso di fault, la riconfigurazione del modulo colpito. Inoltre essa si occupa di inviare ai moduli funzionali i dati da elaborare e salva i risultati. Questa sezione è implementata tramite un SW che viene eseguito dal PS dello Zynq.

4.7.1 Gestione dei dati

Nel sistema implementato, i dati in ingresso al sistema sono memorizzati in un file di testo all'interno di una memoria esterna (SD-Card). Il processore legge questo file e trasferisce i dati in un vettore nella memoria di sistema. Una volta acquisito l'intero file, l'applicazione inizia ad inviare i valori da elaborare ai moduli funzionali. I risultati, una volta estratti dal voter controller, sono quindi salvati all'interno di un secondo file di testo, memorizzato anch'esso nella memoria esterna.

4.7.2 Voter controller

Il voter controller analizza gli output generati dai voter, applicando la stessa strategia di maggioranza ma su di un set di dati maggiore (composto dai segnali di fault indications, channel indications e data outputs). Questo permette di rivelare e mascherare un eventuale fault, indipendentemente dal fatto che siano stati colpiti i moduli funzionali oppure i voter. Una volta che l'errore è stato localizzato, il voter controller ne indica la posizione al reconfiguration controller, in modo da poter ripristinare il corretto funzionamento del modulo fallato. L'algoritmo di decisione implementato dal voter controller è riportato nella Figura 4.14, e consiste in una serie di confronti sui valori di ingresso. Possono essere individuate tre situazioni:

- **Tutti gli ingressi sono uguali e i segnali di fault indications sono deasseriti:** nessun errore è stato rivelato e il dato elaborato dai moduli funzionali può essere salvato nel file di output;

- **Tutti gli ingressi sono uguali e i segnali di fault indications sono asseriti:** almeno un modulo funzionale ha subito un fault. Il segnale channel indication permette di localizzare il modulo fallato. In caso di fault multipli è necessario eseguire una riconfigurazione totale di tutti i moduli funzionali, in quanto non è possibile applicare l'algoritmo di maggioranza per individuare la posizione del fault. In questa situazione è inoltre necessario eseguire nuovamente l'elaborazione dei dati, considerando che non è possibile mascherare l'errore nei valori calcolati;

- **Gli ingressi sono diversi:** almeno un voter ha subito un fault. Applicando l'algoritmo di maggioranza è possibile individuare quale voter deve essere corretto. In caso di fault multipli, similmente a quanto è richiesto nei moduli funzionali, tutti i voter devono essere riconfigurati. Inoltre i risultati dei moduli funzionali potrebbero essere stati corrotti ed è quindi necessario eseguire nuovamente l'elaborazione.

4.7.3 Reconfiguration controller

Il reconfiguration controller gestisce il processo di riconfigurazione degli RM come descritto in 4.3.2. Esso utilizza un DMA controller per trasferire nella configuration memory il partial bitstream relativo al RM da riconfigurare. Per il corretto funzionamento del controller è necessario specificare la posizione nella memoria di sistema del bitstream e la sua lunghezza (espressa in numero di words). Il trasferimento non richiede di conoscere a priori quali locazioni della configuration memory verranno scritte: questa informazione è già codificata all'interno del partial bitstream.

4.7.4 Applicazione SW

Il codice eseguito dal PS implementa le funzionalità di voter controller e di reconfiguration controller. Esso è stato scritto in C, realizzando un'applicazione *bare metal*, ovvero che agisce direttamente sulle risorse del processore senza l'intervento di un sistema operativo. Il flusso di operazioni può essere riassunto dal flow chart in Figura 4.15. Esso mostra le fasi in cui l'esecuzione dell'applicazione è divisa:

1. **Inizializzazione:** i dati da elaborare vengono letti dal file esterno e caricati in un vettore. L'interfaccia PCAP viene impostata per permettere la riconfigurazione parziale e dinamica;

2. **Lettura di un dato dal vettore:** viene selezionato un elemento dal vettore contenente i dati da elaborare;

3. **Invio ai moduli funzionali:** il valore estratto dal vettore viene inviato ai moduli funzionali tramite l'interfaccia memory mapped AXI4-Lite;

4. **Acquisizione dei risultati dai voter:** i dati elaborati dai voter vengono letti attraverso l'interfaccia AXI4-Lite;

5. **Analisi con voter controller:** il voter controller viene impiegato per analizzare i dati ottenuti dai voter ed individuare un eventuale fault. I risultati generati dai moduli funzionali vengono salvati nella memoria esterna;

6. **Riconfigurazione:** se il voter controller ha rivelato un fault, viene attivato il reconfiguration controller per correggere il modulo della sezione TMR colpito dal SEU.

Una volta analizzati i risultati dei voter ed eseguita l'eventuale riconfigurazione, il programma legge un nuovo dato dal vettore, iniziando così un nuovo ciclo di elaborazione che ha termine quando non sono più presenti valori da elaborare.

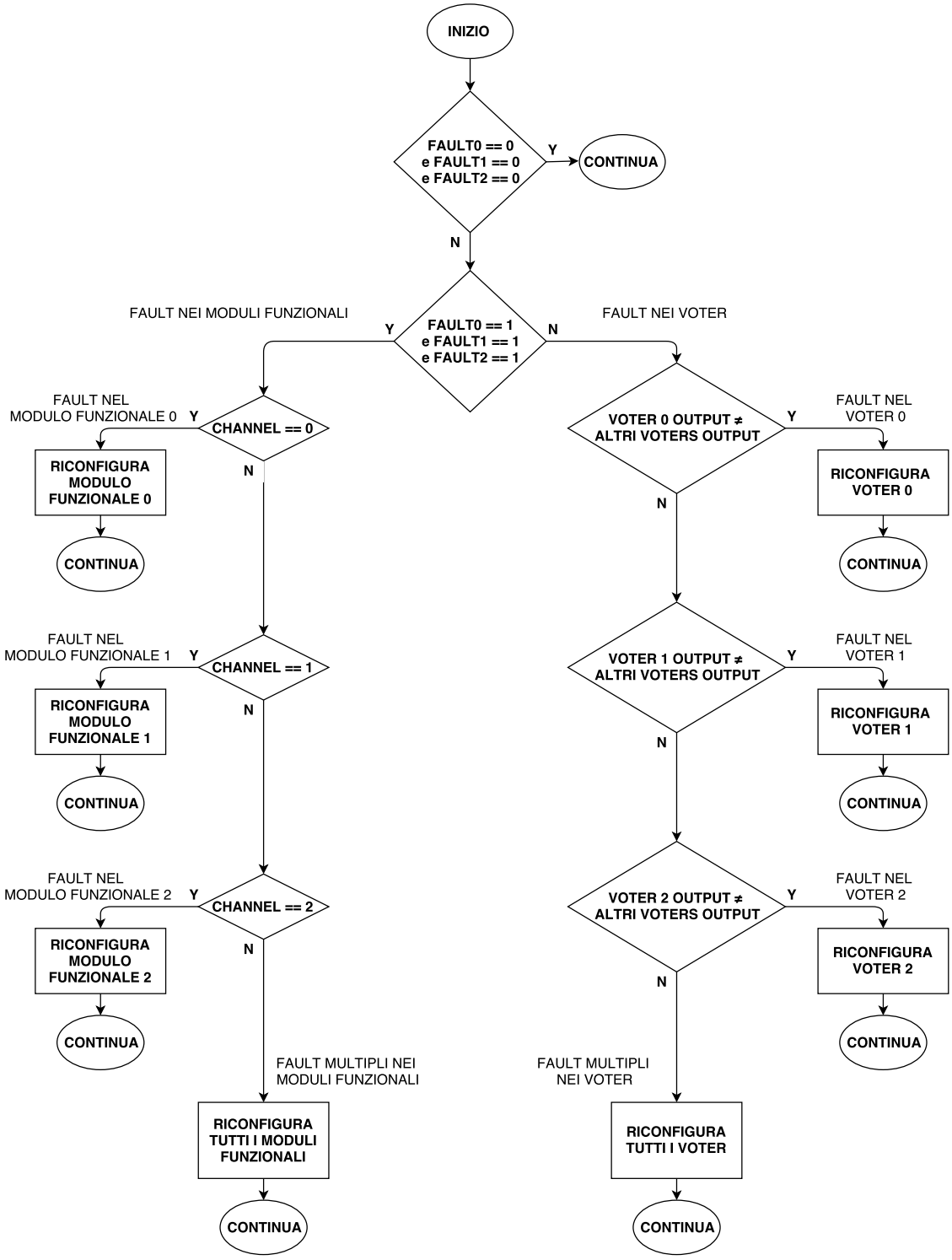

Figura 4.14 *Diagramma di flusso dell'algoritmo implementato dal voter controller*

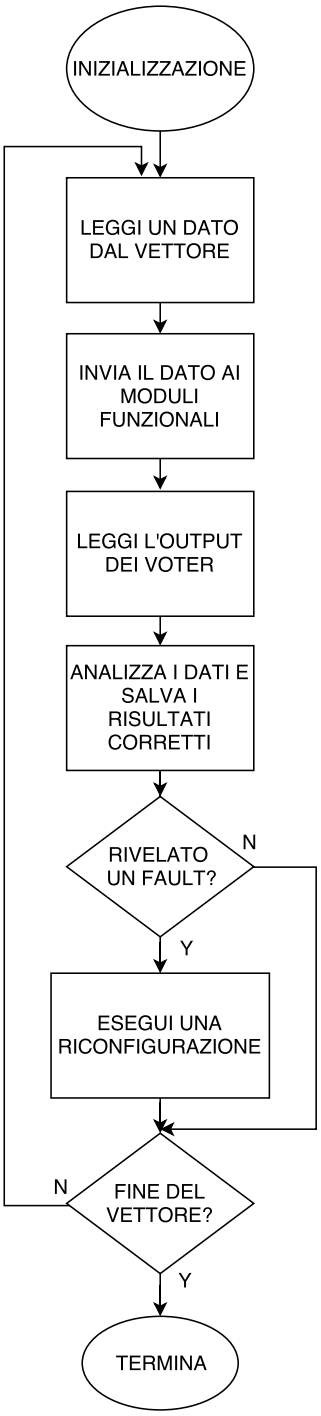

Figura 4.15 *Diagramma di flusso dell'applicazione SW*

Capitolo 5

Validazione della soluzione

Questo capitolo presenta i test effettuati sull'architettura. In particolare sono descritti i test funzionali eseguiti durante la sua implementazione e la tecnica di fault injection utilizzata per validarla.

5.1 Test funzionali

Lo sviluppo di un'architettura complessa richiede l'utilizzo di test estensivi per poter individuare gli errori commessi nell'implementazione del sistema. In particolare si è rivelato necessario poter stimolare la sezione a ridondanza tripla per verificarne il corretto funzionamento e la capacità di mascherare gli errori. Una seconda tipologia di test ha permesso invece di simulare l'iniezione di errori nei moduli riconfigurabili e quindi di esercitare l'intero sistema.
Entrambe le tipologie di test richiedono di intervenire nel flusso di operazioni eseguite dal processore per stimolare il sistema, interrompendo momentaneamente l'elaborazione. Questo permette, nel caso della seconda tipologia di test, di simulare fault multipli iniettando errori in più moduli prima di riprendere la normale esecuzione del codice.

5.1.1 Test architetturale

Nell'architettura sviluppata, il PS fornisce i dati da elaborare ai tre moduli funzionali in maniera indipendente, interagendo con un solo modulo alla volta. Questo viene sfruttato per testare la sezione TMR. Se ad un modulo vengono forniti dati diversi rispetto a quelli inviati agli altri, questo genererà risultati diversi. I voter individueranno questa discrepanza e la interpreteranno come un fault, segnalandolo al PS. Questa tecnica permette

di eseguire una verifica approfondita dei voter, considerando che possono essere generate facilmente diverse combinazioni di output dai moduli funzionali. È inoltre possibile testare parte delle funzionalità del voter controller, verificando che esso indichi correttamente il modulo da correggere al reconfiguration controller.

5.1.2 Test funzionale

Questo test permette di esercitare le capacità dell'intero sistema simulando l'iniezione di errori nei moduli riconfigurabili. Esso sfrutta il concetto di riconfigurabilità per sostituire un RM con un modulo difettato. Durante il flusso di progetto, questi moduli fallati sono progettati allo stesso modo di quelli effettivamente impiegati nel sistema. Tuttavia, le funzionalità implementate vengono modificate. Ad esempio, mentre un modulo funzionale corretto esegue un'operazione di somma fra due valori, un modulo funzionale fallato può eseguire una somma e aggiungere una costante al risultato. Un voter fallato potrebbe invece segnalare costantemente la presenza di fault, anche se i moduli a monte non commettono alcun errore. Queste modifiche portano i moduli a comportarsi in modo diverso rispetto alle altre unità dello stesso tipo, dando quindi origine ad un fault. In particolare, sostituire un modulo funzionale permette di testare il corretto funzionamento dei voter, mentre riconfigurare un voter permette la verifica del voter controller. In entrambi i casi viene testato anche il reconfiguration controller, il quale deve correggere il modulo fallato. È quindi possibile testare la funzionalità dell'intera architettura, con lo svantaggio principale di un overhead nella memoria di sistema, che deve contenere anche i partial bitstream relativi ai moduli fallati. Poter eseguire test multipli richiede di generare un file di configurazione per ogni fault da inserire, portando ad una campagna estensiva di debug molto costosa sia in termini di risorse fisiche che temporali (ogni RM deve essere infatti sintetizzato ed implementato).
La riconfigurazione di un modulo (e quindi l'iniezione di un fault) viene effettuata prima di inviare i dati ai moduli funzionali, in modo stimolare il sistema e raccogliere i risultati in un solo ciclo di elaborazione. Se l'iniezione venisse infatti eseguita dopo aver inviato i dati, il fault potrebbe non essere rivelato nell'attuale ciclo. Per comprendere questa affermazione è sufficiente supporre che il soggetto della riconfigurazione sia un modulo funzionale. Si supponga altresì che l'elaborazione dei dati avvenga in $10ns$, dopodichè i risultati vengono trasferiti ai voter. Se il processo di riconfigurazione ha inizio $100ns$ dopo l'invio dei dati, questi ultimi non risentiranno dell'errore introdotto e quindi il sistema non rivelerà alcun fault almeno fino all'elaborazione successiva.

5.2 Test di validazione

Il sistema è stato validato implementando una tecnica di fault injection, con l'obiettivo di simulare i SEU nella configuration memory del PL. Questi possono modificare in maniera persistente le funzionalità dei moduli, mentre eventuali SEU che colpiscono i dati elaborati dalla FPGA possono venire mascherati dal sistema TMR, limitando così il loro impatto sulle successive elaborazioni. La sezione di controllo implementata nel PS verrà realizzata utilizzando soluzioni radhard convenzionali e non è quindi stata considerata nel test di validazione effettuato.
È possibile definire un sottoinsieme dei bit della memoria di configurazione, chiamati *essential bits*. Non tutti i bit descrivono delle risorse effettivamente utilizzate dal design implementato nel PL. Gli essential bits sono definiti come i bit della configuration memory che, se modificati, alterano la funzionalità del sistema. Per simulare un SEU è quindi sufficiente agire su di essi cambiandone il livello logico. Questo è possibile accedendo alle relative locazioni della configuration memory da interfacce quali PCAP o ICAP. Il risultato è un fault molto simile a quelli causati dalle radiazioni, permettendo a questa tecnica di fault injection di essere realistica e in grado di sostituire metodi di validazione molto più complessi e costosi.
Vivado IDE permette di individuare gli essential bits all'interno del bitstream, generando un file con estensione .edb. Per ottenere questo file è necessario specificare il seguente constraint:

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

Il file .edb contiene una matrice rappresentante i bit della configuration memory. Gli essential bits sono identificati come elementi della matrice posti a valore uno, mentre gli altri bit sono posti a valore zero. La relazione fra i bit della matrice e i moduli implementati nel PL non è nota, rendendo così molto difficile iniettare errori in una specifica zona della FPGA. È pertanto necessario agire su ogni essential bit e valutarne gli effetti per identificare a quale modulo il bit sia associato. Questo processo può richiedere una grande quantità di tempo per design complessi, considerando che la configuration memory dello Zynq-7020 contiene più di 25 milioni di bit.

5.2.1 SEM controller

Per implementare la tecnica di fault injection appena descritta è necessario istanziare un sistema che possa operare parallelamente al resto dell'architettura sotto test. Il Soft Error Mitigation (SEM) controller IP core [25] messo a disposizione da Xilinx permette di iniettare faults nella configuration memory attraverso ICAP. Questo core permette

inoltre di correggere gli effetti delle radiazioni effettuando periodicamente una verifica di tipo ECC alla memoria (tecnica chiamata *scrubbing*). Questa funzionalità esula dagli scopi di questa tesi. È possibile comunicare con SEM attraverso due interfacce:

- **Monitor Interface:** interfaccia di tipo seriale full duplex con protocollo compatibile RS-232. Permette di inviare comandi e ricevere un rapporto dettagliato sullo stato del core;
- **Error Injection Interface:** interfaccia composta da un ingresso parallelo a 40 bit e un segnale di strobe. Permette solamente di inviare comandi.

Le altre interfacce presenti sul controller sono l'interfaccia ICAP verso la configuration memory e una *status interface*, la quale indica lo stato in cui il controller si trova. Status interface è composta da 8 bit, ognuno di quali rappresenta uno stato del controller (descritti in seguito), ad eccezione del segnale *status_heartbeat* il quale viene asserito per un colpo di clock ogni 150 cicli. Quets'ultimo segnale è pensato per permettere ad un watchdog esterno di controllare il corretto funzionamento del core, considerando che esso stesso può essere soggetto ai SEU. Le interfacce *FRAME_ECC* e *Fetch* sono opzionali e vengono usate solo quando SEM è abilitato alla correzione dei SEU. La Figura 5.1 mostra le porte appena illustrate. Si è scelto di connettere il controller al PS tramite la

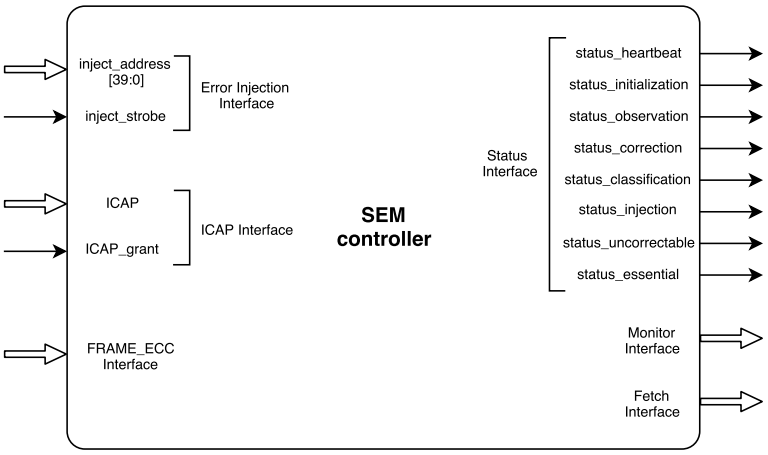

Figura 5.1 *Interfacce del SEM controller IP core [25]*

Erorr Injection Interface, le cui funzionalità risultano sufficienti per gli scopi richiesti. Un modulo chiamato *AXI2InjectInterface* è stato sviluppato per interfacciare il processore al IP core. Questo ha come ingresso un'interfaccia AXI4-Lite e come uscita un registro a 40 bit e un segnale di strobe. La porta memory mapped indirizza quattro registri da 32 bit, le cui funzioni sono riportate nella Tabella 5.1.

Indirizzo	Register
Indirizzo_base + 0	Inject_address (0 ÷ 31)
Indirizzo_base + 4	Inject_address (32 ÷ 39)
Indirizzo_base + 8	Inject_strobe
Indirizzo_base + 16	Status interface

Tabella 5.1 *Mappatura dei registri di interfaccia del modulo AXI2InjectInterface*

In particolare, i primi due registri vengono concatenati per generare l'output a 40 bit, mentre il terzo permette di attivare il segnale di strobe. Il quarto registro riporta al PS i segnali della status interface, rendendo possibile ottenere un feedback sulle operazioni eseguite dal controller. La Figura 5.2 rappresenta il modulo e le sue interfacce.

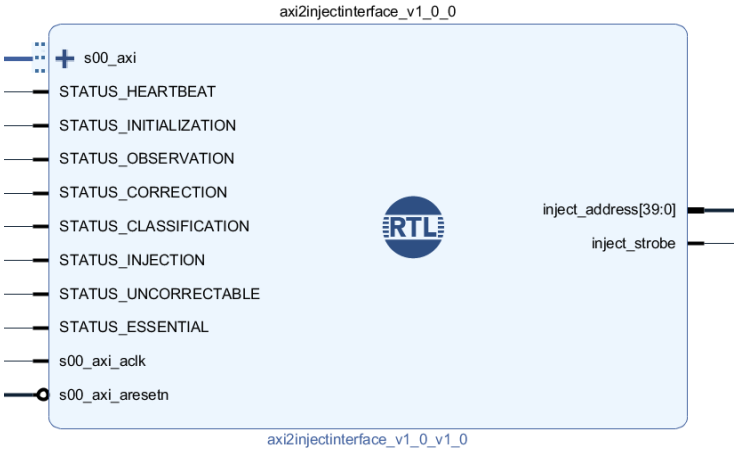

Figura 5.2 *Interfacce del modulo AXI2InjectInterface*

Un secondo modulo chiamato *AXI_ICAPgrant* è stato implementato per controllare il segnale *ICAP_grant*. Questo segnale esterno è necessario in quanto il core non ha modo di percepire autonomamente se la porta ICAP abbia accesso alla configuration memory oppure se quest'ultima sia connessa a PCAP, di fatto escludendo SEM. Prima di utilizzare il controller, il PS deve quindi attivare ICAP deasserendo il bit di controllo PCAP_PR come indicato in Figura 4.3, e successivamente portare a livello logico alto il segnale ICAP_grant per notificare al controller la disponibilità della porta di accesso. Il segnale deve essere asserito per l'intera durata delle operazioni del IP core. Il modulo implementato per controllare questo segnale presenta un'interfaccia di ingresso AXI4-Lite ad un registro, il quale permette di imporre uno stato logico al segnale di uscita. La

Figura 5.3 mostra il modulo implementato.

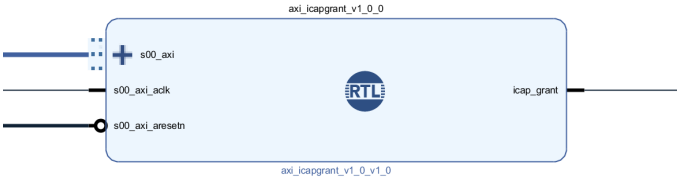

Figura 5.3 *Interfacce del modulo AXI_ICAPgrant*

Per poter utilizzare il controller è necessario comandarne la macchina a stati finiti, la quale prevede sette stati:

- **Initialization:** questo stato è raggiunto automaticamente quando il segnale ICAP_grant viene portato al livello logico alto e quindi attivato. Il segnale della status interface *status_initialization* è asserito. Una volta completata l'inizializzazione del core, esso entra nello stato di osservazione senza bisogno di alcun comando esterno;
- **Observation:** il controller osserva la memoria di configurazione alla ricerca di errori. Il segnale della status interface *status_observation* è asserito. Da questo stato è possibile andare progredire Idle oppure in Correction;
- **Correction:** il controller tenta di correggere un errore rivelato nello stato Observation. Il segnale della status interface *status_correction* è asserito;
- **Classification:** il controller classifica l'errore come *correctable* o *uncorrectable*. Il segnale della status interface *status_classification* è asserito;
- **Idle:** il controller non svolge nessuna funzione, rimanendo in attesa di comandi. Tutti i bit della status interface sono deasseriti. Da questo stato si evolve in Observation oppure in Injection;
- **Injection:** il controller inietta un errore, invertendo lo stato logico di un bit della configuration memory specificato tramite un indirizzo. Il segnale della status interface *status_injection* è asserito. Al termine dell'iniezione il controller torna automaticamente in Idle;
- **Fatal Error:** il controller ha rilevato un problema di consistenza interna. Tutti i bit della status interface sono asseriti.

Degli stati elencati, la tecnica di fault injection utilizzata prevede l'uso dei soli stati di Initialization, Observation, Idle e Injection come mostrato nel diagramma di stato

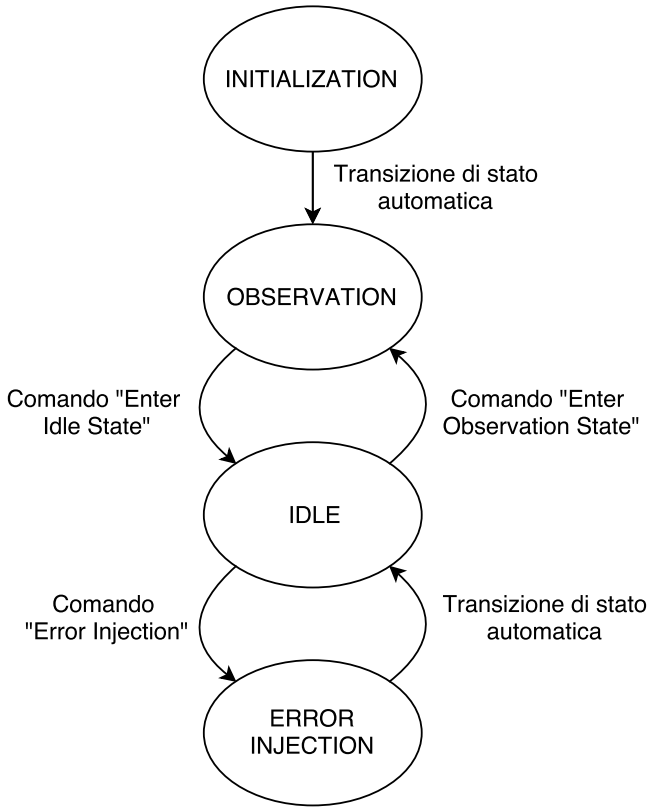

Figura 5.4 *Diagramma di stato che descrive l'evoluzione degli stati del controller durante l'iniezione degli errori*

in Figura 5.4. I quattro bit più significativi del vettore *inject_address* permettono di cambiare stato (Figura 5.5). Inoltre, nel comando *Error Injection Command*, i 31 bit meno significativi permettono di specificare l'indirizzo del bit della configuration memory su cui intervenire. Questo è composto da diversi campi: il primo da destra è identificato come Stacked Silicon Interconnect (SSI), ovvero una tecnologia di realizzazione degli IC sviluppata da Xilinx [28]. Zynq non utilizza questa tecnologia e quindi i bit SSI sono impostati a 00. I campi che seguono (Linear Frame Address (LFA), word address e bit address) permettono di localizzare un bit all'interno dell'essential bit file, il quale è organizzato come una collezione di linee. Ogni linea corrisponde ad una word di 32 bit mentre 101 linee costituiscono un frame. È possibile quindi calcolare il LFA di un particolare bit all'interno di una word come

$$\text{LFA} = \textit{linea della word} \ \% \ 101$$

dove l'operando % indica il resto della divisione intera. Il word address è riferito alla posizione della word all'interno del frame, variando da 0 a 100. Infine, bit address indica

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

Enter Idle State Command

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

Enter Observation State Command

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

Error injection Command

Dove:

- SS = Codice SLR per dispositivi SSI e 00 per dispositivi non-SSI (2 bit)
- LLLLLLLLLLLLLLLLL = linear frame address (17 bit) [0 ÷ MaxFrame]
- WWWWWWW = word address (7-bit) [0 ÷ 100]
- BBBBB = bit address (5-bit) [0 ÷ 31]

Figura 5.5 *Comandi usati per implementare la tecnica di fault injection con il SEM controller IP core [25]*

la posizione all'interno della word del bit sul quale si desidera agire. Unendo questi campi è possibile ottenere inject address, come riportato nella seguente espressione:

$$inject\ address\ =\ 0xC000000000\ |\ (\ \mathrm{LFA}\ \ll\ 12)\ |\ (word\ addr\ \ll\ 5)\ |\ bit\_addr$$

dove l'operando $\ll$ è utilizzato per indicare lo shift dei singoli bit verso sinistra, mentre l'operando | identifica l'operazione logica *or* bitwise. Il prefisso 0x denota che il numero che segue è espresso in notazione esadecimale, mentre il carattere C è richiesto per generare un comando di error injection.
Per poter controllare il core ed iniettare i fault è necessario intervenire nel codice eseguito dal PS. Sono state perciò create due funzioni:

- **Init_SEM()** viene richiamata prima del ciclo di elaborazione dei dati. Per permettere al controller di entrare nello stato di inizializzazione, l'accesso alla configuration memory viene assegnato ad ICAP e il segnale ICAP_grant viene asserito. Una volta terminata questa fase, la funzione impone al core di entrare nello stato Idle, inviando il relativo comando. Infine ripristina l'accesso alla memoria di configurazione a PCAP. Il segnale ICAP_grant non viene deasserito in modo che il controller non venga disabilitato ma rimanga nello stato Idle.

- **Err_inj(int err_address)** viene richiamata ogni volta in cui è necessario invertire lo stato logico di un bit nella memoria di configurazione e quindi iniettare un fault. L'accesso alla configuration memory viene assegnato ad ICAP e il comando di Error Injection viene inviato al core, completandolo nei bit meno significativi con l'indirizzo passato alla funzione. Viene quindi eseguito un polling sui bit della status interface finchè il core non torna nello stato Idle. Infine l'accesso alla memoria di configurazione viene ripristinato a PCAP, permettendo così al PS di eseguire la riconfigurazione parziale del modulo fallato.

5.2.2 Test di fault injection

Per validare la resistenza ai SEU dell'architettura implementata è stato eseguito un test di fault injection, sfruttando l'IP core SEM controller per agire sulla configuration memory e simulare l'effetto delle radiazioni. Come descritto in 4.4, i moduli funzionali utilizzati nel test implementano una funzione di elevamento a potenza con 10 moltiplicatori. L'area occupata da questi moduli è valutata al 47.8% dell'area totale a disposizione del PL, mentre i voter occupano complessivamente il 2.1%, con riferimento ad un SOC Zynq XC7Z020. Il test eseguito consiste nell'iniezione di un numero significativo di fault in indirizzi scelti casualmente con distribuzione di probabilità uniforme, in modo da simulare la distribuzione spaziale con cui le radiazioni colpiscono i circuiti integrati. Considerando che la configuration memory dello Zynq-7020 è composta da circa 25 milioni di bit, essa può subire in media 93 SEU al giorno se posta in un'orbita low Earth orbit (LEO) [29]. Questo significa che, in una missione della durata media di 10 anni, possono avvenire circa 350 000 fault. Pertanto, è stato scelto di iniettare un milione di fault, permettendo così di testare esaustivamente il sistema. L'iniezione avviene prima di inviare dei nuovi dati da elaborare ai moduli funzionali per lo stesso motivo esposto in 5.1.2. In ogni caso questo non è un limite e l'errore può essere iniettato in qualunque altro istante.

Il sistema ha rivelato e corretto 22 328 fault, 21 431 dei quali sui moduli funzionali mentre 897 sui voter. Questi ultimi occupano infatti un'area minore rispetto ai moduli funzionali impiegati nell'architettura implementata. Le rimanenti iniezioni hanno colpito bit della configuration memory non essenziali, non causando quindi modifiche delle funzionalità del sistema. Nessun errore ha corrotto i dati elaborati, dimostrando le capacità di auto-correzione della tecnica proposta. Tuttavia, il test ha anche evidenziato un punto debole dell'architettura. Infatti, il protocollo AXI utilizzato per collegare il PS ai moduli nel PL implementa un handshake. Se un fault colpisce una delle interfacce AXI, può accadere che il PS, nel tentativo di trasferire dei dati, resti indefinitamente in

attesa di una risposta a causa del processo di handshake non completato correttamente. Questo porta allo stallo del processore. Nel test effettuato sono stati rilevati 36 stalli, corrispondente al 0.0036% dei fault iniettati. Si noti come questo valore sia significativo, in quanto risulta essere indipendente dalle funzionalità implementate dai moduli. Infatti le interfacce introducono un overhead costante sull'area, sia nel caso in cui i moduli funzionali eseguano semplici elaborazioni sia nel caso in cui applichino ai dati complessi algoritmi. Questo problema può essere risolto utilizzando un watchdog, come descritto nel capitolo 6.2. Infine, la tabella 5.2 riassume i risultati del test di validazione effettuato.

Totale fault iniettati	1 000 000
Fault su bit non essenziali	977 636
Fault rivelati e corretti	22 328
nei moduli funzionali	21 431
nei voter	897
Stalli del processore	36

Tabella 5.2 *Risultati del test di validazione dell'architettura*

Capitolo 6

Osservazioni conclusive

6.1 Conclusioni

In questa tesi è stata presentata una soluzione in grado di aumentare l'affidabilità dei sistemi elettronici COTS esposti ad ambienti radiativi. L'utilizzo di questa tecnica permette l'impiego di componenti commerciali nei computer di bordo dei satelliti, riducendo il costo di quest'ultimi.
L'architettura è stata validata con una tecnica di fault injection in grado di emulare realisticamente i SEU. In particolare sono stati iniettati un milione di errori, numero sufficiente a simulare la quantità di fault che colpisce un integrato quale lo Zynq-7020 in un'orbita LEO. Il sistema ha corretto con successo gli errori introdotti senza che questi abbiano causato la perdita dell'informazione elaborata.
Il test eseguito ha portato in evidenza un punto debole del sistema rappresentato dalle interfacce di comunicazione fra i vari elementi. Se queste subiscono un fault, è possibile che parte del sistema si blocchi e necessiti di un reset. Questo evento risulta però essere raro, con lo 0.0036% di probabilità di manifestarsi, ed indipendente dalla funzionalità implementata dal sistema.

6.2 Sviluppi futuri

Il sistema presentato è ancora in una fase iniziale di sviluppo e diversi miglioramenti verranno applicati. Questa sezione riporta alcuni dei punti sui quali si concentreranno gli sforzi futuri.

L'architettura implementata permette di utilizzare solamente moduli funzionali combinatori, in quanto non è presente un meccanismo per il ripristino del contesto del modulo fallato. Questa limitazione verrà superata permettendo al reconfiguration controller l'accesso ai registri interni dei moduli, in modo da trasferire una copia del loro contenuto dai moduli non fallati verso il modulo difettato una volta eseguita la riconfigurazione.

Per aumentare il throughput del sistema saranno impiegate delle interfacce AXI4-Stream in unione ad un DMA controller per alimentare i moduli della sezione TMR, sostituendo quindi le più lente interfacce AXI4-Lite. L'uso del DMA ridurrebbe inoltre il carico computazionale richiesto alla CPU, non più necessaria per trasferire i dati da elaborare.

L'uso del PS verrà ulteriormente ridotto impiegando una strategia ad interrupt per gestire il processo di riconfigurazione, in luogo del polling attualmente utilizzato. In questo modo il PS sarà in grado di analizzare i risultati dei voter anche durante il processo di riconfigurazione.

Il test di fault injection ha evidenziato come le interfacce fra i moduli siano dei punti critici. Se queste sono soggette a fault, il PS potrebbe entrare in stallo. Una soluzione a tale problema consiste nell'utilizzo di un IC radhard esterno con funzione di watchdog per effettuare un reset del processore in caso di blocco. Inoltre, per evitare che al momento del ripristino il sistema entri nuovamente in stallo, è necessario riconfigurare il modulo che ha causato il blocco.

In caso di hard error, le risorse fisiche del PL possono subire un danno fisico non riparabile effettuando una riconfigurazione. Una soluzione a questo problema è implementare RP con maggiori risorse di quante strettamente necessarie e prevedere più configurazioni di uno stesso modulo. Ogni configurazione esegue la stessa funzione, ma utilizza risorse diverse all'interno della RP. Nel caso in cui, in seguito ad una riconfigurazione, il fault sia ancora presente, questo viene identificato come hard error e il sistema segnala il RM inserito come non più utilizzabile. Viene quindi caricata una nuova configurazione del modulo, la quale non impiega le risorse colpite dal SEE, permettendo così al sistema di riprendere a funzionare correttamente.

Per sfruttare ulteriormente le possibilità offerte dalla riconfigurazione parziale si possono implementare diversi set di moduli funzionali, ognuno dei quali esegue un diverso algoritmo e viene caricato quando richiesto. Ad esempio, un sistema di elaborazione di immagini può disporre di un set di RM per eseguire un filtro gaussiano e un'altro set per effettuare edge-detection. Questa soluzione permette di incrementare sia l'affidabilità che le performance del sistema.

Bibliografia

Bibliografia

[1] Letessier-Selvon, A., Stanev, T., *Ultrahigh energy cosmic rays*, Reviews of Modern Physics 83(3), pp. 907-942, 2011

[2] Luke O'C. Drury, *Origin of Cosmic Rays*, Dublin Institute for Advanced Studies, School of Cosmic Physics, 2012

[3] D.J. Bird et al., *Detection of a Cosmic Ray with Measured Energy Well Beyond the Expected Spectral Cutoff Due to Comsic Microwave Radiation*, October 1994

[4] Gerald Estrin, *Organization of computer systems - the Fixed plus Variable Structure computer*, western joint IRE-AIEE-ACM computer conference, Pages 33-40, May 3-5, 1960

[5] Xilinx, UG702 Partial Reconfiguration User Guide (v14.5), April 26, 2013

[6] S. Guetersloh et al., *Polyethylene as a radiation shielding standard in simulated cosmic-ray environments*, Nuclear Instruments and Methods in Physics Research, Section B: Beam Interactions with Materials and Atoms 252(2), pp. 319-332, November 2006

[7] Mr. Abhishek Verma et al, *Effect of Threshold Voltage on Various CMOS Performance Parameter*, Int. Journal of Engineering Research and Applications, Vol. 4, Issue 4 (Version 8), pp.21-28, April 2014

[8] ECSS-Q-HB-60-02A, *Techniques for radiation effects mitigation in ASICs and FPGAs handbook*, ECSS Secretariat, ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands, 1 September 2016

[9] R. E. Lyons, W. Vanderkulk, *The use of triple-modular redundancy to improve computer reliability*, IBM Journal of Research and Development (Volume: 6, Issue:

2), Pages: 200 - 209, April 1962

[10] J. R. Sklaroff, *Redundancy Management Technique for Space Shuttle Computers*, IBM Journal of Research and Development 20(1), pp. 20-28, 1976

[11] Emilio Fazzoletto, *Characterization of Partial and Run-Time Reconfigurable FPGAs*, Degree project in information and communication technology, KTH Royal Institute of Technology, Stockhol, Sweden, 2016

[12] T. Vladimirova et al., *Enabling Technologies for Distributed Picosatellite Missions in LEO*, Proceedings - First NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2006, pp. 330-337, 2006

[13] L. Sterpone et al., *A Novel Fault Tolerant and Runtime Reconfigurable Platform for Satellite Payload Processing*, IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 8, AUGUST 2013

[14] Brock J. LaMeres et al., *An FPGA-based Radiation Tolerant SmallSat Computer System*, Aerospace Conference, 2017 IEEE, 4-11 March 2017

[15] YANG MengFei et al., *Architecture design for reliable and reconfigurable FPGA-based GNC computer for deep space exploration*, Science China Technological Sciences 59(2), pp. 289-300, 2016

[16] Zhuo Ruan et al., *A Dynamically Partial-reconfigurable FPGA-based Architecture for Data Processing on Space Solar Telescope*, 2007 Symposium on Industrial Embedded Systems Proceeedings, SIES'2007, pp. 194-199, 2007

[17] Xilinx, UG585 Zynq-7000 All Programmable SoC Technical Reference Manual, December 2017

[18] ARM Cortex A9 Technical Reference Manual, 2016

[19] Xilinx, 7 Series FPGAs Data Sheet: Overview, August 2017

[20] AMBA, AXI, and ACE Protocol Specification, 2013

[21] Xilinx, UG902 Vivado Design Suite User Guide - Using the Vivado IDE, 2017

[22] Xilinx, UG902 Vivado Design Suite User Guide - High-Level Synthesis, 2017

[23] PG193 Partial Reconfiguration Controller v1.2 LogiCORE IP Product Guide, 2017

[24] Xilinx, UG909 Vivado Design Suite User Guide Partial Reconfiguration, v2017.1, April 5, 2017

[25] Xilinx, PG036 Soft Error Mitigation Controller v4.1 LogiCORE IP Product Guide, 2017

[26] Xilinx, PG082 Processing System 7 v5.5 LogiCORE IP Product Guide, 2017

[27] Xilinx, PG059 AXI Interconnect v2.1 LogiCORE IP Product Guide, 2017

[28] Kirk Saban, *Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency*, WP380 (v1.2) December 11, 2012

[29] A.L. Vampola et al., *Single Event Upsets correlated with environment*, IEEE Transactions on Nuclear Science, Volume: 41, Issue: 6, Dec. 1994

Sitografia

[30] Sito web Argotec srl, `http://www.argotec.it/online/`, 2018

[31] Sito web Fraunhofer Institute, `https://www.fraunhofer.de/`, 2018

[32] Sito web Microsemi, `https://www.microsemi.com/`, 2018

[33] Sito web Altera, `https://www.altera.com/`, 2018

[34] Sito web Xilinx, `https://www.Xilinx.com/`, 2018

[35] Sito web Avnet, `https://www.avnet.com/wps/portal/us`, 2018

[36] Sito web ARM, `https://www.arm.com/`, 2018

[37] Sito web Zedboard, `http://zedboard.org/product/zedboard`, 2018

[38] AR#65199 7 Series Reference Clock Propagation Delay Mitigation, `https://www.xilinx.com/support/answers/65199.html`, 08/26/2015