Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Tesi di laurea magistrale

Generazione automatica di modelli SystemC RTL di hardware riconfigurabile dinamicamente

Candidato:
Enrico Giordano
Matricola VR386687

Relatore:
Prof. Graziano Pravadelli

Anno Accademico 2015-2016

Indice

Capitolo 1

Introduzione

I sistemi embedded sono un particolare tipo di hardware progettato per eseguire una specifica funzione; per questo sono definiti "special purpose" per differenziarli dai sistemi "general purpose", che invece sono sistemi generici in grado di svolgere diverse funzioni generiche. I sistemi embedded stanno diventanto sempre più utilizzati in differenti contesti (domotica, automotive, comunicazione, ecc.) grazie ai loro punti di forza, ovvero il costo e il consumo (molto inferiore rispetto ai sistemi general purpose).

Il flusso di progettazione di questi sistemi è basato sulla comprensione delle specifiche del progetto, per poi creare un modello del sistema tramite modelli di computazione (FSM, Reti di Petri, ecc.) e ricavarne un sistema effettivamente implementabile. Una volta scelto il modello e il linguaggio di specifica, si procede nell'implementazione del sistema, per poi ottenere un sistema da verificare e testare, ovvero rispettivamente dimostrando differenti proprietà del sistema e identificare problemi durante la sintesi del sistema su silicio.

Una tecnologia utilizzata per la progettazione di sistemi embedded è la FPGA, un circuito integrato le cui funzionalità sono programmate via software ("Field Programmable Gate Array"). Questi dispositivi possono essere riprogrammati ogni volta che si desidera rappresentare un sistema diverso dal precedente. Esistono molteplici tipi di FPGA, in base al sistema che si vuole rappresentare.

In particolare esistono le FPGA riconfigurabili dinamicamente che sono in grado di riconfigurarsi a runtime secondo le regole scelte dal progettista. Attualmente non esiste un metodo standard per la progettazione di questo tipo di FPGA.

Con questa tesi, si vuole proporre una metodologia di sviluppo di sistemi embedded basato sull'utilizzo di FPGA riconfigurabili dinamicamente, proponendo un flusso di progettazione e un tool di generazione automatica di codice utilizzando un linguaggio appositamente progettato per questo scopo. Usando questo mezzo, si vuole progettare un sistema che presenta aspetti embedded (quindi specifici per il target finale) e aspetti più generici e, calato in un contesto particolare, sarà in grado di modificarsi in base alle esigenze ed essere più performante per il contesto in cui viene utilizzato, ottenendo diversi benefici. Basandosi sul concetto di "evoluzione", secondo cui un oggetto si modifica in base all'ambiente, si otterrà per ogni ambiente un particolare sistema embedded.

Si presenteranno quindi i seguenti aspetti nei vari capitoli:

- nel Capitolo 2 verrà introdotto, a livello descrittivo , lo stato dell'arte delle tecnologie utilizzate per la rappresentazione dei sistemi embedded e in particolare delle FPGA riconfigurabili;

- nel Capitolo 3 verranno esplicitati gli obiettivi di questa tesi;
- nel Capitolo 4 verrà mostrata la metodologia proposta per la progettazione dei sistemi basati su FPGA riconfigurabili;
- nel Capitolo 5 verrà presentato il caso di studio creato appositamente per dimostrare l'efficacia della metodologia;
- nel Capitolo 6 verranno mosse le relative conclusioni di questo progetto;
- nel Capitolo 7 si mostrerà la bibliografia.

Capitolo 2

Stato dell'arte

2.1 Tecnologia attuale del silicio

Attualmente esistono 3 tecnologie differenti riguardo l'utilizzo del silicio per scopi computazionali: Hardware General Purpose, Hardware Embedded e Hardware Riprogrammabile. Queste 3 tipologie differiscono sia per la composizione, sia per la progettazione ma anche per il settore di utilizzo.

2.1.1 Hardware General Purpose

L'Hardware General Purpose identifica quel tipo di hardware che serve per risolvere problemi generali e quindi non è dedicato ad una specifica funzione.

Questo tipo di Hardware lo si può trovare in processori e microprocessori utilizzati in strumenti di calcolo generici, come PC, moderni telefoni e televisori, che non rispondono a una particolare esigenza ma si adattano a quelle dell'utilizzatore. La caratteristica peculiare di questi dispositivi hardware è la generalità: non essendo stati progettati per un sistema specifico, devono permettere di eseguire più operazioni possibili ed essere riprogrammati a piacimento.

Questo tipo di hardware è molto utile per implementare sistemi generici: la filosofia alla base di questa tecnologia è l'utilizzo più generico possibile. Un esempio di questa tecnologia è la CPU, unità di elaborazione di un PC domestico, che deve essere in grado di eseguire programmi di vario tipo, non limitandosi ad una specifica funzione.

2.1.2 Hardware Embedded

Con Hardware Embedded si identificano genericamente tutti quei sistemi elettronici di elaborazione digitale a microprocessore progettati appositamente per una determinata applicazione (special purpose) ovvero non riprogrammabili dall'utente per altri scopi, spesso con una piattaforma hardware ad hoc, integrati nel sistema che controllano ed in grado di gestirne tutte o parte delle funzionalità richieste.

Questo tipo di Hardware è contraddistinto dalla specificità dei suoi componenti interni in base al suo utilizzo; infatti si cerca di utilizzare questo hardware in ambienti o condizioni specifiche, in cui sono richieste operazioni particolari ma soprattutto è necessario ridurre i costi e il consumo dell'applicazione finale. Esistono vari tipi di tecnologie che implementano questo tipo di hardware, come ASIC, SoC, ASIP, PLC, ecc, e difficilmente si trova un'architettura standard, proprio perché l'architettura è specifica per un certo comportamento. Esistono in commercio

dei sistemi embedded basati su microprocessori RISC, come ad esempio i Cortex-M della ARM, che hanno un'architettura molto simile a quella delle CPU. In generale comunque ogni sistema embedded ha un'architettura diversa, in quanto viene mappato su silicio uno specifico algoritmo o un'architettura più complessa.

Un sistema embedded deve costare il meno possibile e deve consumare il meno possibile, in base alle esigenze richieste; ovviamente potrà eseguire solo il compito per cui è stata progettata; ha quindi una filosofia di progettazione opposta a quella dell'Hardware General Purpose.

2.1.3 Hardware Riprogrammabile

L'ultimo tipo di Hardware è quello riprogrammabile, ovvero permette di essere configurato a livello di celle di memoria più volte per implementare diversi comportamenti. Di solito viene utilizzato sia in ambito embedded, per permettere più libertà di progettazione e più efficienza, ma anche a scopo sperimentale, perchè è utile per testare un sistema in fase di sviluppo.

La tecnologia che implementa questo tipo di hardware è la FPGA. Questa è un circuito integrato le cui funzionalità sono programmabili via software. Tali dispositivi consentono l'implementazione di funzioni logiche anche molto complesse, e sono caratterizzati da un'elevata scalabilità. Questo tipo di tecnologia ha assunto un ruolo sempre più importante nell'elettronica industriale così come nella ricerca scientifica.

Esistono diversi tipi di FPGA, che comprendono sia dispositivi programmabili una sola volta, sia dispositivi riprogrammabili un grande numero di volte. I primi, detti OTP (One Time Programmable), sono costituiti da componenti il cui stato di funzionamento cambia in modo permanente, permettendo di mantenere la configurazione allo spegnimento del dispositivo. Alla seconda categoria appartengono i dispostivi basati su tecnologia SRAM (Static Random Access Memory), i quali devono essere riprogrammati ad ogni accensione, avendo una memoria di configurazione volatile.

I circuiti FPGA sono elementi che presentano caratteristiche intermedie rispetto ai dispositivi ASIC (Application Specific Integrated Circuit) da un lato e a quelli con architettura PAL (Programmable Array Logic) dall'altro. L'uso di componenti FPGA comporta alcuni vantaggi rispetto agli ASIC: si tratta infatti di dispositivi standard la cui funzionalità da implementare non viene impostata dal produttore che quindi può produrre su larga scala a basso prezzo. La loro genericità li rende adatti a un gran numero di applicazioni come consumer, comunicazioni, automotive eccetera. Essi sono programmati direttamente dall'utente finale, consentendo la diminuzione dei tempi di progettazione, di verifica mediante simulazioni e di prova sul campo dell'applicazione. Il grande vantaggio rispetto agli ASIC è che permettono di apportare eventuali modifiche o correggere errori semplicemente riprogrammando il dispositivo in qualsiasi momento. Per questo motivo sono utilizzati ampiamente nelle fasi di prototipazione, in quanto eventuali errori possono essere risolti semplicemente riconfigurando il dispositivo. L'ambiente di progettazione è anche più user-friendly e di relativamente facile apprendimento. Di contro, per applicazioni su grandi numeri sono antieconomici perché il prezzo unitario del dispositivo è superiore a quello degli ASIC (che di converso hanno elevati costi di progettazione).

Il costo di tali dispositivi è oggi in rapida diminuzione: ciò li rende sempre di più una valida alternativa alla tecnologia standard cell. Usualmente vengono programmati con linguaggi come il Verilog o il VHDL. Molte case costruttrici (ad esempio Xilinx e Altera) forniscono gratuitamente sistemi di sviluppo che supportano quasi tutta la loro gamma di prodotti.

La struttura di una FPGA è in generale una matrice di blocchi logici configurabili, detti CLB (Configurable Logic Blocks), connessi fra loro attraverso interconnessioni programmabili. Ai margini di tale matrice vi sono i blocchi di ingresso/uscita, detti IOB (Input Output Block). I CLB realizzano le funzioni logiche, l'insieme di interconnessioni li mette in comunicazione, mentre

gli IOB si occupano dell'interfacciamento del circuito con l'esterno. All'interno di tale matrice sono presenti anche altri tipi di risorsa, come i DCM (Digital Clock Manager), che generano il segnale di clock, la rete che trasporta il segnale di clock dai flip-flop ai CLB ed altre risorse di calcolo, come ad esempio le ALU (Arithmetic Logic Unit), e risorse di memoria distribuita. Ciascuno di questi elementi costitutivi ha un modello di funzionamento specifico, che riveste notevole importanza nella comprensione del corretto funzionamento del dispositivo.

Attorno alla FPGA sono presenti vari moduli standard che possono servire per il consueto utilizzo in ambito di controlli, ma non sempre sono necessarie, in quanto si possono descrivere all'interno della parte riconfigurabile. Parte necessaria per il funzionamento corretto di una FPGA è la logica di controllo della parte riprogrammabile: deve esserci un hardware dedicato per rimappare questa parte, in modo che il processo di riconfigurazione avvenga più velocemente possibile; questo si occuperà quindi di riconfigurare, in base al bitstream che rappresenta l'hardware, la parte riconfigurabile.

In base al suo utilizzo può assumere aspetti di un hardware general purpose e di un hardware embedded; essendo una tecnologia piuttosto costosa, ha un mercato più ristretto rispetto ai tipi di hardware precedentemente descritti.

2.2 Hardware Metamorfico e tecnologia associata

Si vedrà ora nel dettaglio il significato di hardware metamorfico e in particolare la tecnologia che implementa questo concetto.

2.2.1 Il concetto di Hardware Metamorfico

Il concetto di "Hardware Metamorfico" nasce nei primi anni del 1990, per poi perdere parzialmente interesse in quanto veniva presentato come idea, non come tecnologia pratica. Con questo concetto si intende un tipo di Hardware capace di cambiare il proprio comportamento ("evolvere") utilizzando un "metodo" di riconfigurazione: i termini "evolvere" e "metodo" non sono mai stati definiti in maniera standard, l'articolo che cerca di porre chiarezza su tutto questo ambito [1] focalizza l'attenzione sulle diverse sfaccettature del termine "evoluzione", parlando di evoluzione "intrinseca" , "estrinseca" e "mixata", rimandando la definizione ad un articolo scientifico precedente [2].

Questo tipo di Hardware è tipico del mondo naturale, in quanto tutti gli oggetti e gli esseri viventi sono in grado di modificarsi in base all'ambiente che li circonda.

Il ciclo di vita di un Hardware Metamorfico può essere rappresentato come nella figura 2.1: il superciclo che compone questo sistema, chiamato "ciclo innato", ha un sotto-ciclo di acquisizione di stimoli, chiamato "ciclo di acquisizione", che, tramite il sottostrato di algoritmi di esecuzione, cambia il funzionamento del sistema in base al "comportamento attuale" e il "comportamento desiderato"; con questi dati, tramite un modulo di valutazione degli stimoli e un meccanismo di investigazione (per valutare il comportamento desiderato rispetto al comportamento attuale), viene generato dal modulo di "decisione" il risultato che fa modificare l'intero sistema. Il ciclo innato successivamente fa rieseguire questo procedimento, in modo da permettere, potenzialmente all'infinito, la metamorfosi del sistema.

2.2.2 Hardware riconfigurabile

Per rendere utile l'hardware metamorfico, è necessario trovare una tecnologia che lo rappresenti, in modo da rendere concreto l'utilizzo di tale idea. Per renderla concreta, è necessaria la presenza

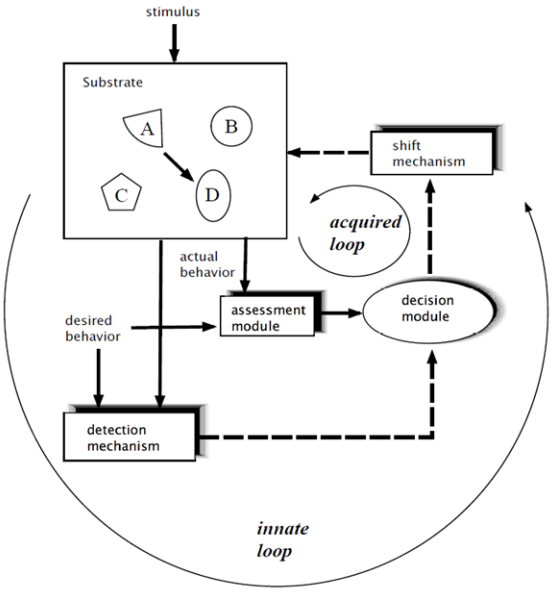

Figura 2.1: Ciclo evolutivo di un sistema metamorfico

di un oggetto che possa riprodurre sia il comportamento di silicio, sia il comportamento evolutivo dell'hardware metamorfico. Nel 1985, la Xilinx aveva creato le prime FPGA, rilasciando in commercio il primo modello "XC2064", utilizzabile perloppiù per la prototipazione di circuiti programmabili. Questa tecnologia sembra prestarsi coerentemente con l'idea di Hardware metamorfico, in quanto è in grado di poter cambiare il proprio comportamento a livello hardware cambiando la propria struttura interna. Il suo carattere riconfigurabile però è stato progettato e utilizzato per la prototipazione, non per l'effettivo mutamento di comportamento durante il suo ciclo di vita.

Nei primi anni '90 nasce una tecnica di progettazione chiamata "Riconfigurabilità", da qui poi la tecnologia di applicazione, appunto le FPGA. Questa tecnica consiste nell'utilizzare Hardware con tecnologia riprogrammabile che, tramite la memoria messa a disposizione dentro il sistema, riesce a modificare il suo comportamento. Non è un comportamento appreso completamente dall'ambiente, ma è un comportamento previsto e quindi implementato che va a sostituire un comportamento esistente; il ciclo di vita quindi non sarà composto da un vero e proprio "apprendimento", ma sarà una modifica prevista dal progettista.

Quindi, più precisamente, si può parlare di "riconfigurabilità parziale" e "riconfigurabilità run-time": il primo termine pone enfasi sulla parziale capacità del dispositivo di modificarsi, in quanto ci sarà una parte progettata che non deve modificarsi ma soprattutto perché il dispositivo non si modificherà completamente, avrà delle modifiche consentite prestabilite; il secondo termine indica il fatto di potersi modificare durante il suo ciclo di vita, cambiando la propria configurazione di porte logiche ottenendo quindi una forma diversa.

In particolare, questo tipo di sistema è composto da 2 parti: la parte statica e quella dinamica. La parte statica è quella parte del sistema che non si modifica durante l'esecuzione del sistema e solitamente dirige la riconfigurazione della parte dinamica; la parte dinamica invece è la parte in cui avviene la riconfigurazione, ovvero dove vengono caricate le varie configurazioni del sistema descritte dal progettista (ovvero dove avviene la riconfigurazione parziale). Si può interpretare

come architettura Master-Slave, in cui la parte statica (Master) controlla la parte dinamica (Slave) affinchè esegua correttamente tutte le riconfigurazioni previste.

2.2.3 Metodi di riconfigurazione

Essendo la Riconfigurabilità una tecnica di progettazione, prevede vari metodi per essere implementata, che possono essere scelti sia per la tecnologia sia in base alle risorse disponibili su essa. Questi metodi sono stati proposti dall'Altera [3] e da vari cultori della materia [4].

La prima tecnica è la "Riconfigurazione parziale dinamica", che consiste nel cambiare una parte della FPGA mentre il rimanente circuito continua il suo ciclo di vita. Per fare ciò, esiste una partizione riconfigurabile e in questa si carica il bitstream che corrisponde al nuovo circuito da una memoria esterna. Per far comunicare la parte statica con quella dinamica, è posto un bridge fisico tra di esse.

Il bridge fisico consiste in un bus, solitamente a 32 bit, che permette lo spostamento di file sintetizzati che rappresentano la nuova configurazione hardware. Questi file si trovano o nella memoria esterna o nella memoria interna, quindi vengono caricati runtime in base a condizioni scelte dal progettista. Solitamente il bus è di tecnologia DMA in modo da essere il più veloce possibile e il trasferimento avviene tramite diversi protocolli gestiti a livello hardware (solitamente il protocollo PCAP a 32 bit, ma dipende dall'architettura ed è trasparente al progettista).

Questa tecnica è molto efficace quando si ha a disposizione poca memoria per memorizzare il bitstream che rappresenta l'hardware e può essere utilizzata in quelle architetture che non dispongono di sufficiente memoria per caricare al completo tutto il sistema descritto.

La seconda tecnica è la "Rilocazione parziale di bitstream", che consiste nello "spostare" i moduli compilati che descrivono l'hardware (che rappresentano la parte riconfigurabile) da un'area della FPGA all'altra con le stesse dimensioni e proprietà. Questa tecnica è più limitante della precendente, un quanto si pone il vincolo di avere due moduli con stesse dimensioni e proprietà, però può essere utile per questioni di ottimizzazione di codice, in quanto i due moduli possono rappresentare la stessa unità ma al loro interno eseguono lo stesso algoritmo in maniera differente.

La parte cruciale quindi di queste tecniche è proprio la reallocazione dei moduli, in quanto la riconfigurabilità consiste proprio nello "spostare" un modulo da una sezione attiva a una non attiva. Esistono due metodi di reallocazione dei moduli riconfigurabili: usando dei tool offerti dalla Xilinx e usando le direttive del bus.

I tool della Xilinx sono dei software che fungono da middleware per gestire lo spostamento da un'area all'altra e vengono dati in dotazione in base al sistema acquistato. Le direttive del bus invece sono delle macro da utilizzare durante l'esecuzione del codice per istruire il bus prima di far comunicare i diversi moduli e collegarli.

I metodi descritti creano un "ponte" tra la zona statica e la zona dinamica della logica programmabile, in modo da interconnetterle per generare una nuova descrizione hardware.

Entrambi i metodi si basano sulla riconfigurazione runtime della parte di logica programmabile tramite lo stesso hardware della FPGA che si occupa della riconfigurazione statica, ovvero quel tipo di riconfigurazione che viene attuata da parte del progettista ogni volta che vuole riconfigurare manualmente una FPGA standard, ma in maniera automatica: il modulo di controllo di riconfigurazione sceglie quale può essere la zona di logica programmabile corretta (la prima che ha abbastanza spazio per contenere la nuova logica) e genera il segnale di riconfigurazione; al momento del segnale, si esegue la procedura di riconfigurazione, ma il bitstream da caricare non deriva da un programma esterno via JTAG bensì dalla memoria interna. Quindi ciò che viene caricato non viene generato runtime, è già presente all'interno della memoria, ma il comportamento globale del sistema cambia in quanto viene caricata una porzione di bitstream che andrà

ad alterare il circuito generato dal bitstream statico (che non cambia mai) e quello dinamico (caricato runtime).

2.2.4 Aree di reallocazione

La parte statica e la parte dinamica del sistema si trovano all'interno della zona di logica programmabile della FPGA, denominata "Programmable logic", dove risidede effettivamente il codice che descrive il sistema. Le due parti del sistema normalmente possiedono una frammentazione interna ed esterna, in quando il codice non sempre occupa tutti i settori di memoria; proprio per questo si può deframmentare e ottenere dei settori vuoti in cui inserire runtime il codice.

Spostare un nuovo settore dentro la parte di logica programmata significa quindi configurare hardware precedentemente inattivo, dando quindi una nuova configurazione hardware e quindi un diverso comportamento del sistema. In base alla grandezza delle aree da spostare, possono coesistere più codici di parti dinamiche all'interno della zona di logica programmabile, ma si sconsiglia l'utilizzo di questa strategia in quanto si utilizza molto più silicio rispetto ad aver allocato un unico codice.

2.2.5 Progettazione di sistemi riconfigurabili dinamicamente

Esistono diversi metodi di sviluppo di sistemi riconfigurabili dinamicamente, però essendo un tipo di sistema storicamente recente, non è stato ancora decisa una metodologia standard per la progettazione. Attualmente sono state proposte delle metodologie basate sull'analisi dell'applicativo finale (quindi un approccio Top-Down) e sulla generazione di modelli TLM in SystemC [5] [6].

Il metodo normalmente utilizzato per la progettazione di questi sistemi è lo stesso dei sistemi embedded, ovvero si crea un modello di sistema, scegliendo il linguaggio e l'ambiente di simulazione più adatto alla situazione e, dopo aver verificato il sistema, si procede con il test e la sintesi. Per quest'ultima parte, in base alla FPGA scelta, si utilizza l'apposito software dato in dotazione dalla casa produttrice della FPGA.

Perciò, si vuole creare un nuovo flusso di progettazione per gestire al meglio la fase di sviluppo. Poiché questi sistemi fanno parte di un sottoinsieme dei sistemi embedded (a causa della specificità dell'applicazione finale di questi sistemi), si utilizzerà ciò che è noto durante la fase di sviluppo di un normale Sistema Embedded per poi aggiungere ciò che serve per specializzare tale sistema in uno Metamorfico.

2.3 Simulazione e sintesi di sistemi embedded

Esistono vari metodi per progettare un sistema embedded; l'approccio ingegneristico più avanzato vuole che si progetti in maniera astratta il sistema, per poi calarlo in un contesto di descrizione tramite HDL per poi eseguire la sintesi.

2.3.1 Simulazione

Per la descrizione dei sistemi e la loro simulazione, solitamente si utilizza SystemC, passando da una descrizione mirata a osservare il comportamento dei diversi moduli durante la loro interazione (TLM), fino alla descrizione a livello di EFSM cycle accurate o analogica (RTL e AMS). I vantaggi dell'utilizzo di SystemC sono molteplici, soprattutto grazie allo scheduler interno si riesce a simulare correttamente i vari comportamenti a livello di segnali e attivazione dei moduli.

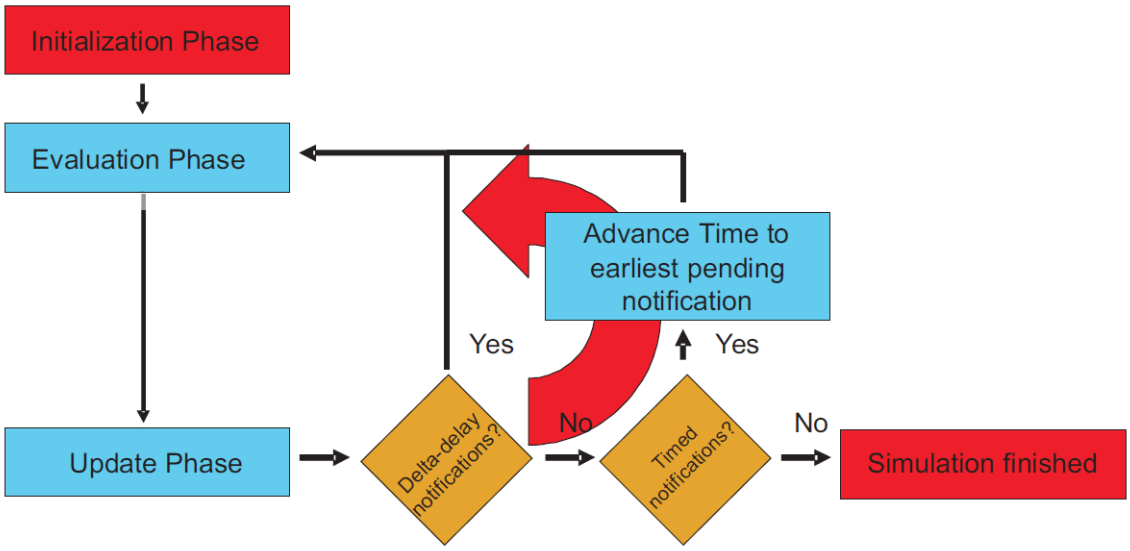

Figura 2.2: Lo scheduler di SystemC

2.3.2 SystemC

Il SystemC è un insieme di librerie e macro scritte in C++ con cui è possibile simulare processi concorrenti, ovvero che avvengono nello stesso momento, ognuno descritto attraverso la sintassi C++. Nell'ambiente di lavoro del SystemC, gli oggetti descritti possono comunicare in un contesto simulato real-time usando segnali di tutti i tipi di dato offerti dal C++, insieme ad altri offerti dalle librerie SystemC, oltre a quelli definiti dall'utilizzatore.

Il linguaggio offre una semantica simile a quella di VHDL e Verilog, ma al costo di un aggravamento sintattico rispetto a questi. D'altro canto, permette un maggiore libertà espressiva, come programmazione orientata agli oggetti e classi template. Più in generale, SystemC è sia un linguaggio di descrizione sia un sistema di simulazione che permette di generare un eseguibile che si comporta come il modello descritto al momento dell'esecuzione. Le prestazioni del sistema di simulazione sono difficilmente paragonabili a quelle dei simulatori VHDL/Verilog commercializzati attualmente, specialmente per via delle differenze di ambito applicativo.

2.3.3 Scheduler di SystemC

Lo scheduler di SystemC è stato progettato in modo da rispettare una gerarchia di wakeup dei moduli, in quanto l'hardware solitamente è pilotato sia da un clock interno sia da segnali provenienti da diversi moduli, quindi è stato necessario modellare questo parallelismo tramite uno scheduler. Lo schedule eseguito dallo scheduler è diviso in passi:

1. Elaborazione: vengono create le strutture dati e viene fatto il binding delle porte di input e di output;

2. Inizializzazione: vengono inizializzate porte e segnali e vengono fatti eseguire tutti i processi fino alla loro terminazione o al primo wait, in modo da creare una coda d'esecuzione iniziale;

3. Evaluation: viene fatto eseguire il primo processo pronto in coda d'esecuzione fino alla sua terminazione o ad un wait. Se vengono notificati eventi (con delle notify()), i processi interessati vengono messi in coda d'esecuzione. Se ci sono altri processi in coda pronti ad essere eseguiti ripeto le operazioni precedenti, altrimenti proseguo;

4. Update: vengono aggiornate porte e segnali ed aggiunti in coda d'esecuzione i processi sensibili ad essi;

5. Delta: vengono aggiunti alla coda d'esecuzione tutti i processi sensibili ad eventuali delta notification (notify(SC_ZERO_TIME)) Se ci sono altri processi in coda pronti ad essere eseguiti ripeto le operazioni precedenti a partire dalla fase di evaluation, altrimenti proseguo;

6. Timed: Vengono aggiunti alla coda d'esecuzione tutti i processi sensibili alla prima timed notification (notify(SC_TIME(n,SC_NS))). Aggiorno il tempo di simulazione (clock). Se ci sono altri processi in coda pronti ad essere eseguiti ripeto le operazioni precedenti a partire dalla fase di evaluation, altrimenti termino la simulazione.

Grazie a questo scheduler, si ha uno strumento che simula correttamente un sistema embedded, rispettando accuratamente anche i tempi di esecuzione, osservabili tramite timestamp.

2.3.4 Processi SystemC

SystemC, per la rappresentazione dei moduli hardware da implementare, utilizza un oggetto chiamato "processo", che a sua volta può essere di 3 tipi: "sc_method", "sc_thread" e "sc_clocked_thread"; il primo appare come una funzione in C++ che quando viene chiamata viene eseguita completamente e viene riattivata secondo la sensitivity list; il secondo differente dal primo tipo può essere bloccato durante la sua esecuzione; il terzo funziona come un sc_thread ma è associato ad un oggetto "clock", ovvero un contatore che può riattivare questo tipo di processo anche se è stato precedentemente messo in pausa. Tra questi tipi di processi, l'unico sintetizzabile è il "sc_method", in quando gli altri processi possono essere bloccati durante l'esecuzione, cosa non possibile per un dispositivo hardware.

2.3.5 Sintesi

Una volta preparato il modello del sistema in SystemC e simulato correttamente, si procede con la descrizione hardware in HDL, simulazione ed infine sintesi. Solitamente si utilizza VHDL e Verilog, due linguaggi di descrizione hardware che permettono sia di simulare sia di sintetizzare. Solitamente se viene descritto correttamente il sistema in SystemC, il passaggio ad una descrizione HDL è quasi automatico, in quanto si rappresentano gli stessi concetti semanticamente; infatti questo processo dovrebbe essere una traduzione sintattica tra linguaggi.

2.3.6 Simulazione e sintesi di sistemi metamorfici

Poichè non esiste un metodo standard per progettare sistemi metamorfici, bisogna definire *ex novo* un metodo efficace. Intuitivamente sarebbe consigliato utilizzare la stessa metodologia di progettazione di sistemi embedded standard, in modo da rendere facile l'apprendimento a chi è già abituato a progettare sistemi embedded consuetudinari, oltre a sfruttare le potenzialità di un metodo già assodato e verificato nel tempo.

2.3.7 Simulazione con SystemC ReChannel

Per la simulazione si vorrebbe utilizzare SystemC, in quanto offre una descrizione fedele dell'hardware da simulare e sfrutta la conoscenza di C++ per descrivere i sistemi. Attualmente esiste una libreria di SystemC per la descrizione di sistemi riconfigurabili chiamata ReChannel.

ReChannel è una libreria sviluppata nell'Università di Bonn nel 2007 [4], ceduta a GreenSocs con licenza open-source l'anno successivo e attualmente non più mantenuta, a seguito degli aggiornamenti di SystemC 2.0 che non permettevano la compilazione. La libreria è stata aggiornata e riscritta durante la stesura di questa tesi ed ora è compilabile ed utilizzabile.

La libreria si basa sul concetto di **switch**, ovvero l'elemento cardine di un sistema riconfigurabile: deve esserci la possibilità di cambiare un modulo con un altro, possibilmente di stessa dimensione e con la stessa quantità di input e output. Lo switch, in questa libreria, viene rappresentato come una sorta di multiplexer, che tiene collegati i diversi moduli della parte riconfigurabile con il modulo della parte statica; al momento opportuno, deciso dal progettista, sia dinamicamente sia staticamente in fase di compilazione, si imposta in modo che attivi il canale di comunicazione tra il modulo attivato e la parte statica, mentre disattiva i moduli della parte dinamica che vengono sostituiti. Lo switch permette la comunicazione sia da parte statica a parte dinamica sia viceversa: la prima viene chiamata *portal*, la seconda viene chiamata *exportal*.

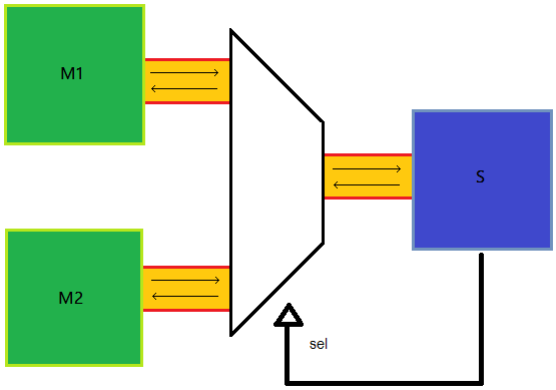

Figura 2.3: Rappresentazione dello switch

Come si può vedere nella figura 2.3, lo switch ha lo stesso funzionamento di un multiplexer: i moduli riconfigurabili M1 ed M2 vengono collegati entrambi allo switch, che a sua volta è collegato al modulo statico S; in base al segnale "sel", pilotato da S, lo switch attiverà il corrispondente modulo riconfigurabile, in modo che abbia un collegamento diretto e unico con il modulo statico. Il processo non attivo rimarrà bloccato fino alla prossima riattivazione. Questo rappresenta di fatto ciò che avviene in una FPGA riconfigurabile dinamicamente, anche se in forma diversa, ovvero mentre qui il modulo riconfigurabile viene disattivato, nella FPGA viene deallocato per poi allocare il modulo riconfigurabile da attivare.

La comunicazione tra parte statica e parte dinamica può essere semplice (a un solo valore) o multivalore (coda fifo). Questo modella correttamente il concetto di bridge descritto precedentemente, ossia quel componente che permette la comunicazione tra le parti della FPGA. Questo si può vedere come una sorta di bus, che dirige il traffico point to point tra i diversi moduli.

L'utilizzo di questo componente per la comunicazione è un aspetto che si distanzia molto dalla solita progettazione di hardware in SystemC, poiché solitamente si definiscono i moduli con vari input e output e si effettua un collegamento punto-punto all'interno del main; con questa

libreria invece si definiscono le modalità di comunicazione tra i moduli di diversa natura, per poi far gestire a questo componente i diversi collegamenti. Questo fatto modella correttamente le FPGA riconfigurabili moderne.

Il problema principale ora, conoscendo lo scheduler di SystemC, è gestire le modalità di schedule dei moduli della parte riconfigurabile; è necessario quindi estendere lo scheduler nativo con delle nuove opzioni.

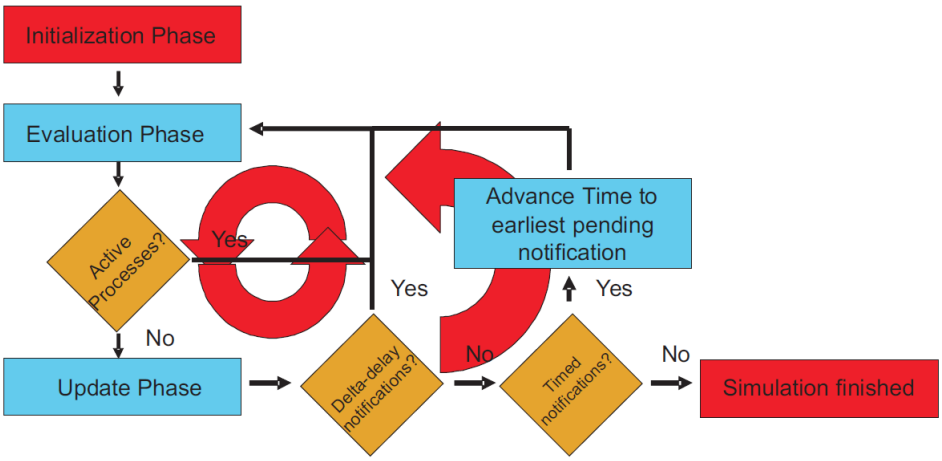

Figura 2.4: Scheduler esteso con ReChannel

Come si può notare con la figura 2.4, è stato esteso lo scheduler dando priorità alla riconfigurazione, quindi nella fase di Evaluation si aggiunge un'ulteriore fase, ovvero il controllo sui processi SystemC attivi, oltre ad aggiungere passaggi nella fase di inizializzazione. Di fatto i moduli riconfigurabili sono processi SystemC che vengono bloccati se un modulo deve essere disattivato (deallocato) e riattivati quando vengono ricaricati (reallocati).

Durante la fase di inizializzazione, vengono istanziati tutti i moduli riconfigurabili e disattivati; è compito del progettista decidere quale modulo inizialmente deve essere attivato, ma non è necessario che ci siano moduli attivi. Successivamente, un modulo può essere attivato in qualunque momento; lo scheduler lo attiverà appena entrerà nella parte finale della fase di Evaluation, controllando che ci siano processi SystemC associati ai moduli riconfigurabili (quindi pronti per essere attivati).

Solitamente, come controllore dei moduli dinamici, si utilizza un modulo "statico" di SystemC Rechannel, che ti fatto è un normale processo SystemC che rappresenta un circuito, che si occupa di fare da transattore tra la parte statica e la parte dinamica del sistema riconfigurabile; solitamente viene chiamato processo "Top" e si occupa sia di descrivere lo switch sia di effettuare la commutazione dei moduli della parte dinamica. Anche in questo caso, viene modellata correttamente l'architettura riconfigurabile, in quando deve esserci un'entità del sistema che si occupa di effettuare la riconfigurazione, che può essere il processore nella board della FPGA o lo stesso progettista se si descrive una riconfigurazione statica.

Un ultimo aspetto molto importante che simula correttamente la realtà è la possibilità di impostare il tempo necessario per la riconfigurazione: ogni modulo, in base alla FPGA target, può metterci tempi differenti per essere attivato e disattivare gli altri, quindi è neccesario un meccanismo per modellare il tempo di riconfigurazione. Questa libreria modella il tempo di riconfigurazione in base al caso pessimo di tempistiche, ovvero permette al progettista di impo-

stare il tempo massimo che impiega il modulo ad essere attivato (infatti non sempre impiega lo stesso tempo per essere riconfigurato, per questo bisogna ragionare per caso pessimo). Può essere deciso questo tempo secondo le specifiche della FPGA target proporzionali all'area che il circuito ricopre (si possono effettuare delle stime in base a quante e quali operazioni vengono eseguite).

2.3.8 Sintesi

La sintesi di questi sistemi è molto simile alla sintesi dei normali sistemi embedded, poiché si utilizza HDL compilato e sintetizzato tramite tool automatici.

La traduzione da SystemC Rechannel a VHDL è molto semplice: si rappresentano i diversi moduli come moduli statici, contrassegnando i moduli dinamici con un nome riconoscibile ed incrementale (es: reconfigurable_module1, reconfigurable_module2, ecc...), ci si assicura che tutto sia stato scritto correttamente e si importano i file nel programma di sintesi prescelto.

Solitamente i programmi di sintesi vengono offerti direttamente dai produttori della FPGA target (ad esempio per la Virtex 4 danno in dotazione il programma) e permettono di impostare facilmente i diversi moduli scritti in HDL tramite interfaccia grafica. Tutti i tool automatici partono quindi dalla descrizione in HDL e, tramite un'ulteriore sforzo del progettista nell'impostare il comportamento dei moduli, acquisiscono la particolarità di essere statici o dinamici.

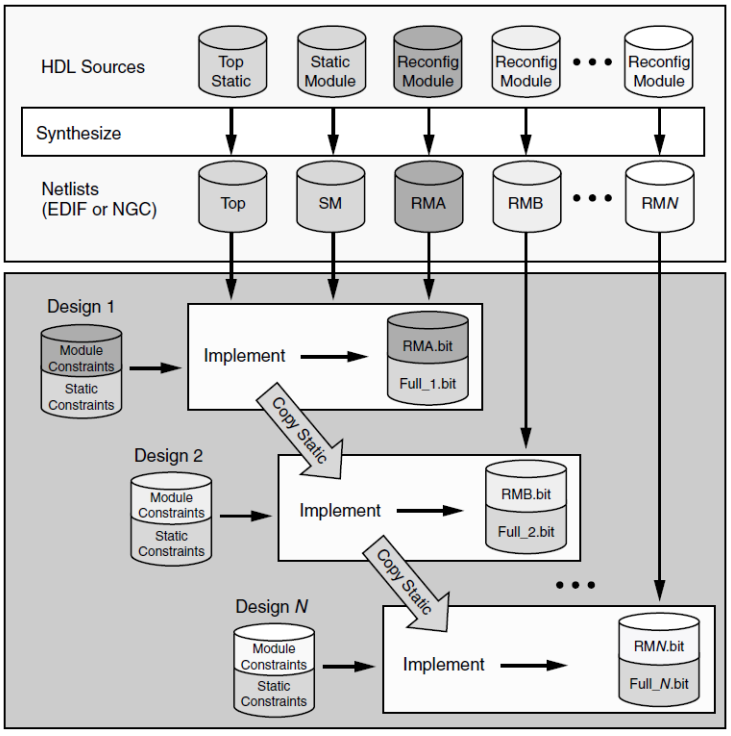

Figura 2.5: Processo di sintesi eseguito da un tool automatico

Come si può notare con la figura 2.5, il tool automatico identifica i sorgenti come moduli che devono essere sintetizzati; possiamo notare i diversi insiemi di moduli tra cui il modulo "Top", già visto precedentemente con SystemC ReChannel, che identifica il transattore tra la parte

statica e la parte dinamica del sistema. Una volta sintetizzati i sorgenti, si crea una Netlist, ovvero si creano i collegamenti interni ed esterni tra moduli, in modo da definire il metodo di comunicazione. Fatto ciò, il tool identifica in automatico tutti i differenti design che si creeranno durante il ciclo di vita del sistema, in base alle configurazioni possibili definite nella Netlist; ogni Design è una copia statica presente in memoria, quindi una volta sintetizzata esistono le configurazioni in memoria già pronte per essere nuovamente sintetizzate in hardware (solitamente risiedono nella memoria Flash).

L'ultimo passaggio del tool automatico è la creazione delle partizioni della FPGA, perché deve essere possibile identificare dove caricare la parte riconfigurabile e dove deve rimanere staticamente il codice; viene creato quindi il file delle partizioni, solitamente chiamato *XPARTITION.xml*, che risiede in memoria e viene consultato ogni volta che deve avvenire una riconfigurazione.

2.3.9 YAML

YAML (pronunciato 'jaemel, in rima con "camel") è un formato per la serializzazione di dati utilizzabile da esseri umani. Il linguaggio sfrutta concetti di altri linguaggi come il C, il Perl e il Python e idee dal formato XML e dal formato per la posta elettronica (RFC2822).

Proposto da Clark Evans nel 2001, è stato sviluppato da quest'ultimo e Brian Ingerson. Il nome definisce l'acronimo ricorsivo "YAML Ain't a Markup Language". Nella prima fase di sviluppo l'acronimo veniva definito come "Yet Another Markup Language", significato che è andato perso in favore di un nome che specificasse la natura orientata alla memorizzazione di dati del linguaggio, contrapposto all'utilizzo consono dei linguaggi di markup.

É quindi un linguaggio di serializzazione dati che permette facilmente di descrivere con item e sequenze dati e parti di codice.

Capitolo 3

Obiettivi

I sistemi riconfigurabili dinamicamente sono una tecnologia piuttosto giovane e vedono come campo di applicazione, per ora, solo quello della ricerca, quindi non sono sistemi ancora commercialmente diffusi, mancando di un metodo di progettazione semplice che possa aiutare nella progettazione e comprensione di questi sistemi. Si vuole quindi creare un modo per rendere più facilmente sfruttabile questa tecnologia tramite una metodologia rigorosa di sviluppo, in grado di aiutare anche chi non conosce a pieno questi sistemi.

Si vuole quindi creare un metodo per la progettazione di questi sistemi, basato sull'utilizzo di un modello di sistema standardizzato e di un tool creato appositamente per sfruttare tale modello in maniera automatica. Utilizzando il modello e il tool automatico proposti, il progettista sarà in grado di sviluppare ed implementare l'intero sistema in poco tempo e con piena consapevolezza del progetto.

Il metodo per risolvere il problema della progettazione efficace si concretizza nello sviluppo del modello del sistema con il tool automatico di generazione di codice: inizialmente si cercherà di creare un modello che potesse generalizzare al meglio questi sistemi, in modo che, al momento della progettazione, si possa identificare subito cosa è necessario implementare; successivamente, creato tale modello, si potrà generare in maniera automatica, a partire da questo modello, il codice che lo rappresenta. Si creerà il tool e il linguaggio interpretato da tale tool in modo che si possa facilmente generare codice SystemC in maniera efficace e senza obbligare lo sviluppatore ad imparare l'utilizzo di nuovi software per la progettazione: infatti il linguaggio interpretato da questo tool risulta molto facile da imparare e sfrutta una formattazione tipica di un linguaggio già noto, ovvero YAML.

Quindi, viste le problematiche dello stato dell'arte, viene proposto un modello di sistema caratterizzato da un'architettura Master-Slave (parte statica Master con parte dinamica Slave) composta da una "descrizione comune" di una EFSM composta da 3 stati: "Init", "Work", "Reset". Con "descrizione comune" si intende la descrizione della parte statica e della parte dinamica utilizzando un modello che li rappresenta con la stessa forma (ovvero con gli stati Init, Work e Reset), ma con il codice differente per i 3 stati. Il modello poi verrà convertito tramite il tool creato, chiamato Pyngu, descrivendolo prima con il linguaggio interpretato dal tool, chiamato PynguLang, per poi ottenere come output del tool il codice finale SystemC.

Successivamente a ciò, per testare l'efficacia della metodologia progettata, si implementerà un caso di studio apposito piuttosto complesso, in modo da verificare se effettivamente risulta

utile durante la progettazione.

Nei prossimi capitoli verrà descritta nel dettaglio la metodologia proposta e il caso di studio.

Capitolo 4

Metodologia

Lo scopo di questo documento è di proporre una metodologia efficiente di progettazione e simulazione per hardware riconfigurabile dinamicamente. Attualmente non esiste un metodo standard per la progettazione di sistemi basati su questa architettura, quindi si è cercato di proporre un metodo più efficiente possibile per garantire anche a chi non conosce approfonditamente questa tecnologia di poterla sfruttare al meglio. Per decidere uno stile di progettazione, si è deciso di basarsi su uno stile già noto nei sistemi consuetudinari embedded, ovvero la progettazione di macchine a stati finiti. La tecnica delle macchine a stati finiti garantisce chiarezza di progettazione e un flusso chiaro di istruzioni che vengono eseguite durante la "vita" del sistema, evidenziando anche i possibili blocchi ed anomalie che possono esserci durante l'esecuzione. Inoltre ci si è serviti del noto concetto di "macchine a stati finiti non deterministiche": è possibile rappresentare un sistema a stati che può comportarsi "contemporaneamente" in molteplici modi, quindi avere più comportamenti tramite lo stesso input in ingresso (solitamente evitato durante la progettazione delle macchine a stati finiti, in quanto non è implementabile).

La progettazione proposta quindi si differenzia in diverse fasi:

- *analisi del sistema*, evidenziando cosa potrebbe essere trattato come sistema standard e sistema riconfigurabile; si analizza quindi cosa deve rimanere statico per tutta l'esecuzione di un algoritmo e cosa può cambiare durante l'esecuzione;

- *analisi degli stati*, ovvero si comincia a creare il sistema a stati evidenziando come evolve il sistema dinamico dividendolo in fasi; si deve quindi creare una macchina a stati generale creando il flusso di esecuzione del sistema, ma non rappresentando ciò che succede all'interno degli stati (perché si suppone che per diverse configurazioni ci saranno diversi comportamenti all'interno delle suddette fasi);

- *definizione degli stati*, ovvero si "disegnano" gli stati del sistema riconfigurabile, evidenziando le diverse macchine a stati che vengono prodotte: in questa fase si devono ottenere più macchine a stati duplicate, con lo stesso numero di stati il cui contenuto però cambia tra macchine, perché il flusso di esecuzione sarà lo stesso ma ciò che avviene negli stati deve essere differente;

- *definizione del Top*, il componente che deve guidare lo svolgimento delle attività della parte riconfigurabile; questo deve avere anch'esso la stessa macchina a stati dei moduli riconfigurabili, ma sarà l'unico componente che può decidere in quale stato deve trovarsi il sistema;

- *implementazione e simulazione*, tramite il tool SystemC Rechannel;
- *validazione e sintesi*.

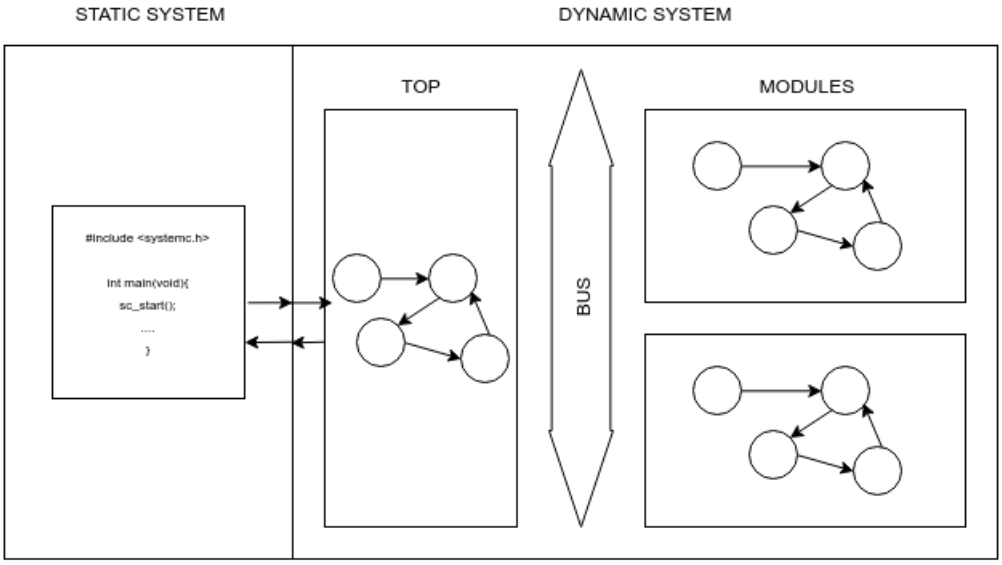

Figura 4.1: Struttura finale del sistema progettato

Il sistema finale dovrebbe essere simile alla figura 4.1: si può notare che il sistema risulta modulare e ben diviso tra parte statica e parte dinamica: questo, oltre a facilitare la comprensibilità del progetto, garantisce anche una buona analisi in fase di simulazione e validazione, in quanto è possibile analizzare nel dettaglio le singole parti del sistema e garantire l'analisi di ogni stato e di ogni modulo. La divisione netta tra parte statica e parte dinamica garantisce anche il parallelismo di esecuzione tra le diverse parti del sistema: potenzialmente la parte statica può eseguire codice in maniera del tutto asincrona rispetto alla parte dinamica, garantendo quindi più potenza di calcolo in termini di velocità. Infine, rappresentare il sistema con macchine a stati finiti permette di effettuare tutte le ottimizzazioni note per diminuire il numero degli stati del sistema, guadagnando ancor più spazio. Si vedrà nel dettaglio con il caso di studio il guadagno ottenuto da questa architettura proposta in contesto di vincoli realtime hard.

È quindi necessario garantire che il sistema sia sempre in stato "safe": come tutti i sistemi asincroni, se sono presenti le 4 condizioni necessarie (mutua esclusione, hold and wait, no preemption, attesa circolare), il sistema può andare in deadlock, quindi è necessario adottare tutte le strategie necessarie per evitare ciò. Verrà esplicitata la strategia per permettere di rimanere nello stato safe nella sezione di presentazione del tool automatico. Fino ad ora la garanzia di stato safe del sistema dal punto di vista di asincronicità era lasciata al progettista, con questo progetto si propone invece un tool automatico per generare il software per SystemC Rechannel adottando questo metodo di progettazione, in modo da lasciare al progettista solo il compito di pensare e progettare il comportamento dei moduli del sistema.

Con questo progetto quindi non si propone esclusivamente una metodologia di progettazione, ma anche un tool automatico per progettare correttamente questi sistemi.

4.1 Rappresentazione della metodologia di progettazione

Il processo di progettazione proposto per questi sistemi deve seguire il Sequence Diagram proposto nella figura 4.2.

Si può notare che il processo proposto risulta lineare, senza cicli e con un'unica diramazione: questo è molto importante dal punto di vista dello sviluppo in quanto non costringe lo sviluppatore ad eseguire delle scelte in base alla situazione e non deve iterare varie parti della progettazione in base al tipo di progetto. La complessità di progettazione quindi è pressochè costante, mentre il punto critico della diramazione consiste in un singolo passo quindi non può portare problemi di sviluppo (dal punto di vista di tempistiche e di utilizzo di risorse).

Tutto parte dalla comprensione delle specifiche da parte del progettista: queste, descritte in qualsiasi forma, devono essere ben comprese dal progettista per identificare la "forma" del sistema futuro. Ogni sistema può essere rappresentato con una macchina a stati finiti estesa, quindi è necessario che il progettista riesca ad indentificare come formalizzare il sistema tramite una FSM estesa. Una volta formalizzata la EFSM, si può descrivere l'intero sistema disegnandolo come fosse una EFSM.

La EFSM può essere descritta a proprio piacimento, ma deve essere abbastanza eloquente in modo da far capire quali siano le parte predestinate alla riconfigurazione. Questo significa che ogni parte del sistema deve essere descritta in maniera generica, tralasciando gli aspetti implementativi (ovvero la descrizione di ciò che avviene all'interno degli stati del sistema), ma bisogna descrivere nel dettaglio ogni potenziale stato del sistema.

Una volta ottenuta la EFSM del sistema, si esegue manualmente il partizionamento della EFSM candidata alla parte statica del sistema e alla parte dinamica: per farlo si possono seguire le seguenti euristiche:

- considerare le parti ripetute della EFSM (dal punto di vista semantico): quando si nota che una parte della EFSM viene ripetuta, ovvero ha le stesse transizioni nel grafo e il comportamento interno ha lo stesso significato (ma l'implementazione futura genererà un comportamento diverso), allora si può inserire quella parte di FSM nella parte dinamica futura della EFSM, mentre il restante grafico si implementerà nella parte statica;

- identificare i potenziali punti in cui il sistema dovrebbe riconfigurarsi e cambiare comportamento (rimanendo semanticamente uguale tra le configurazioni): in alcune situazioni è ben chiaro quando il sistema deve cambiare la propria descrizione interna in base alla situazione ed eseguire un task che contiene delle piccole variazioni rispetto ad altri.

Quindi si può capire che ciò che caratterizza la parte dinamica del sistema è la semantica: devono essere implementati comportamenti diversi, ma la semantica del task deve essere la medesima per ogni riconfigurazione. Basti pensare un forno con diverse modalità di cottura, ognuna ha il significato di "cuocere qualcosa", ma la modalità di cottura è differente, quindi il codice che implementa la cottura sarà differente.

In aggiunta a ciò, si propone anche uno standard di EFSM presentato nella figura 5.6: poiché questo metodo di progettazione è molto efficiente, si può utilizzare uno standard anche di rappresentazione della macchina a stati, in modo da rendere più semplice e chiara la partizione tra sistemi e più semplice la descrizione degli stati.

Questa EFSM è composta da almeno 3 stati, uno di questi può essere esteso a quanti stati si vuole. Lo stato iniziale è lo stato "Init" (nella figura, "I"), in cui avviene l'inizializzazione del sistema riconfigurabile (quindi si resettano i valori delle variabili, si aspetta che la configurazione sia pronta e si prepara il sistema ad eseguire il futuro algoritmo); lo stato successivo è lo stato di

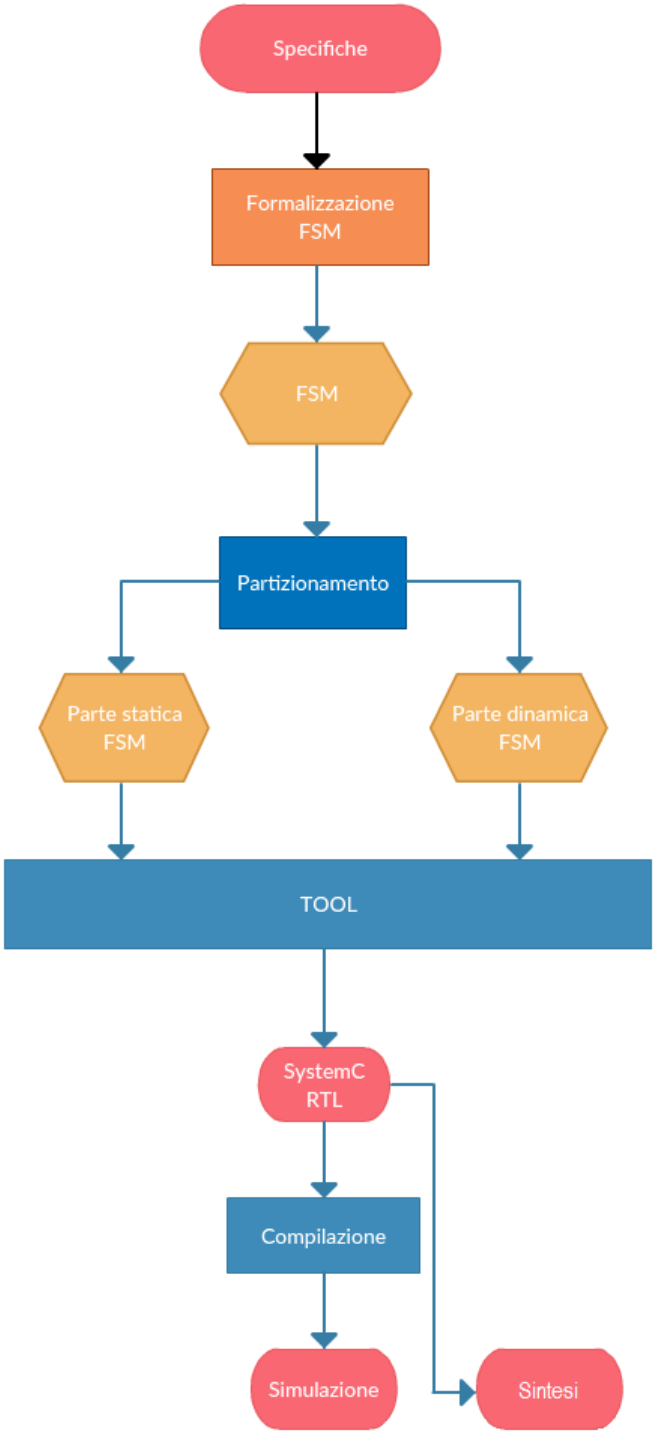

Figura 4.2: Sequence diagram del processo di progettazione

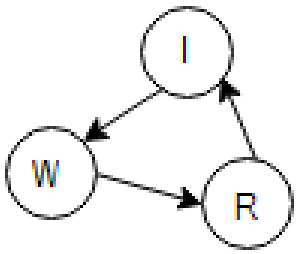

Figura 4.3: Descrizione della FSM standardizzata

"Work" (nella figura, "W"), in cui si esegue effettivamente l'algoritmo per cui il sistema è stato progettato; questo stato può essere esteso a quanti stati si vuole, in base al modello di sistema che si è rappresentato precedentemente. Infine, lo stato "Reconfigure" (nella figura, "R"), esegue la riconfigurazione del sistema per ottenere un comportamento diverso dal precedente; si può dire che questo stato è l'opposto della fase di Init, in quando prepara il sistema alla futura fase di Init di un'altra configurazione.

Non è necessario rappresentare la FSM nel modo precedentemente descritto, ma è consigliato in quanto aiuta a comprendere meglio la partizione del sistema e a creare gli stati correttamente ed in maniera ordinata.

Una volta eseguito il partizionamento si può ottenere il codice del sistema finale utilizzando il tool automatico che genera il codice SystemC partendo da una descrizione ad alto livello. Il tool automatico verrà descritto nella prossima sezione.

Infine, avendo il codice SystemC, è possibile sia simulare il sistema, tramite compilazione, sia generale l'effettivo codice finale per eseguire il deploy su FPGA dedicata. Per convertire il codice da SystemC a Netlist, si può utilizzare un qualsiasi tool automatico di conversione da codice a Netlist.

4.2 Tool automatico (Pyngu)

Poichè il processo di generazione di codice è standard e meccanico, è possibile implementare con un tool automatico tutto il processo di scrittura di codice utilizzando un programma di supporto che possa aiutare il progettista.

Poiché non tutti i progettisti di sistemi embedded conoscono la teoria e la parte implementativa dell'hardware riconfigurabile, è utile astrarre il più possibile dall'utilizzo di SystemC Rechannel e dallo studio dei sistemi riconfigurabili e concentrarsi sulla semantica del sistema. Nasce quindi la necessità di rendere disponibile al progettista un tool automatico che utilizza un mezzo semplificato per descrivere ed implementare il sistema, senza perdere comunque l'espressività di SystemC Rechannel. Utilizzando questo tool, si implementa, secondo lo standard proposto precedentemente, l'intero sistema finale, ottenendo il codice SystemC Rechannel senza sapere i dettagli implementativi della libreria.

Il tool sviluppato si chiama Pyngu, un interprete e compilatore code-to-code scritto in Python; il nome deriva dal noto personaggio dei cartoni, Pingu, che parlava una lingua incomprensibile all'uomo ma riusciva a farsi capire dai suoi simili, oltre al fatto di capire il linguaggio umano (ciò che fa anche questo tool, ovvero trasforma una descrizione ad alto livello del sistema in

un linguaggio che non sempre è noto al progettista); l'aggiunga del suffisso "Py" è dovuto allo standard Python dei progetti.

Il tool prende in input una descrizione ad alto livello del sistema completo, tenendo conto della divisione tra parte statica e parte dinamica: in questo modo, il tool genererà i differenti file in base al modo in cui è stato descritto il sistema, tenendo separata la parte statica dalla parte dinamica.

Come linguaggio di base ad alto livello è stato scelto YAML; è stato scelto questo linguaggio perché è molto semplice da imparare e non richiede nessuna conoscenza di programmazione, in modo da renderlo comprensibile anche a chi non conosce i sistemi riconfigurabili.

Seguendo le regole del linguaggio di base ed aggiungendo delle parole chiave che hanno significato solo per il tool automatico, il tool automatico genera il sistema descritto. In particolare, il tool esegue gli stessi passi che esegue il progettista mentre progetta il sistema: identifica la descrizione della parte statica e dinamica del sistema, trasforma la descrizione in EFSM in SystemC e infine genera il codice del sistema finale.

Figura 4.4: Schema di funzionamento di Pyngu

Il tool fa sia da interprete sia da compilatore code-to-code: prima di tutto controlla che non ci siano errori nel codice scritto dal progettista, analizzando i tag che riesce a riconoscere, poi, partendo da dei template di file SystemC che contengono la struttura del sistema secondo lo standard, popola i file finali SystemC trasformando ciò che ha descritto il progettista in codice.

Inoltre il tool rispetta lo standard di SystemC Rechannel, aggiungendo lo standard proposto in questa tesi, oltre al fatto di gestire le condizioni di safety descritte precedentemente: utilizzando correttamente questo tool e il relativo linguaggio, si eviteranno futuri errori di implementazione che potrebbero compromettere l'intero funzionamento del sistema.

Per risolvere il problema della safety, il tool genera un'architettura Master-Slave dividendo la parte statica del sistema in 3 processi SystemC:

- Producer, che si occupa di produrre i dati ed inviarli al modulo riconfigurabile attualmente attivo;
- Controller, che si occupa di gestire la macchina a stati dell'intero sistema;
- Monitor, che si occupa di leggere i dati di risposta del modulo riconfigurabile attualmente attivo.

Durante il funzionamento del sistema, il Producer invia al modulo riconfigurabile il comando che dovrà eseguire; nel frattempo il Monitor rimane in attesa della risposta del modulo riconfigurabile e il Controller gestisce periodicamente la EFSM in base agli stimoli esterni ed istruendo il Producer a generare il rispettivo output in base allo stato in cui ci si trova. Il modulo infine possiede una coda fifo di comandi in modo che non si possa perdere nessun comando inviato dal Producer. Come si può notare, non si ha mutua esclusione e nemmeno attesa circolare, quindi si è sicuri di essere in uno stato safe e perciò non sono possibili deadlock di sistema.

4.2.1 Il linguaggio (PynguLang)

Il linguaggio riconosciuto dal tool automatico serve per descrivere sia la parte statica sia la parte dinamica del sistema ad alto livello basandosi sulla conoscenza dello standard YAML. Il progettista dovrà essere in grado, dopo lo studio di questo linguaggio, di creare un file ".yaml" contenente la descrizione del sistema secondo la sintassi del linguaggio, in modo da astrarsi completamente dalla conoscenza e studio di SystemC Rechannel.

4.2.1.1 Sintassi

```
port:
    num,type
top-init:
    variables:
        - type, name, value
    channels:
        - type, name
    portals:
        - type, io-type, name
reconfigurable-modules-init:
    num: #
    port:
        num: #
    channels_in:
        - type, name
    channels_out:
        -type,name
    configuration-names: []
    create:
        - type, name, activation_delay, activation_delay_measure
    first-module: *
fsm-init:
    num: #
    states: []

common-other-variables:
    - type, name
specularity-code: *

top-code:
    pre-code: *
    debug-activation: t/f
    producer:
        variables
            -type, name
        code: *
    controller:
        variables:
            -type, name

        code: *
    monitor:
        code: *
modules-code:
    descriptions:
        - type
        fsm:
            - state, code
        other-code: *
common-code: *
main:
    includes: *
    init: *
    code: *
```

Figura 4.5: Struttura del file YAML

Il linguaggio presenta la stessa sintassi del linguaggio YAML ma possiede delle keyword predefinite per permettere al tool automatico di creare i componenti per SystemC Rechannel. Il file dovrà avere genericamente la struttura presentata nella figura 4.5; essendo basato su YAML, è necessario mantenere la gerarchia della descrizione rappresentata come da figura. Come convenzione, il simbolo "-" prima di una proprietà indica un elenco, il simbolo "[]" indica un array o elenco, il simbolo "*" indica un qualsiasi tipo di stringa e il simbolo "#" indica un numero.

4.2.1.1.1 port

```
port:
        num: #
        type: #
```

Indica le porte di comunicazione tra la parte statica del sistema e la parte dinamica. Si può definire la quantità di queste porte tramite la proprietà *num* e il tipo tramite la proprietà *type*. *type* rappresenta una lista di valori e la quantità di valori dipende dalla proprietà *num*.

4.2.1.1.2 top-init

```
top-init:
        variables:
                - type: *
                name: *
                value: *
        channels:
                - type: *
                name: *
        portals:
                - type: *
                io-type: *
                name: *
```

Indica come inizializzare la parte statica del sistema, ovvero i modulo "Top". Di questo modulo, è necessario dichiarare le variabili che utilizza, i canali di input e output e i canali di comunicazione per comunicare con la parte dinamica del sistema; questi sono rispettivamente le proprietà "*variables*", "*channels*" e "*portals*".

Per la proprietà *variables* si deve definire una lista composta dal tipo di variabile tramite la proprietà *type*, il nome della variabile tramite *name* e il valore iniziale tramite *value*.

Per la proprietà *channels* si deve definire una lista composta dal tipo di dato che si vuole trasmettere o ricevere tramite la proprietà *type* e il nome del canale tramite *name*.

Per la proprietà *portals* si deve definire una lista composta dal tipo di dato che si vuole trattare tramite la proprietà *type*, il tipo di canale (ovvero se di input o di output) tramite *io-type* e il nome del canale di comunicazione tramite *name*.

4.2.1.1.3 reconfigurable-modules-init

```
reconfigurable-modules-init:
        num: #
        port:
                num: #
        channels_in:
                - type: *
                name: *
        channels_out:
                - type: *
                name: *
        configuration-names: [*]
        create:
                - type: *
                name: *
                activation_delay: #
```

```
                    activation_delay_measure: *
            first-module: *
```

Indica come inizializzare la parte dinamica del sistema, ovvero i moduli riconfigurabili. Di questi moduli bisogna dichiarare quanti possono essere, la struttura comune e alcune proprietà specifiche di ogni modulo, oltre ad identificare quale sarà il primo modulo ad essere caricato all'avvio del sistema.

La proprietà *num* indica quanti moduli sono previsti per questo sistema, mentre la proprietà *port* indica quanti canali di comunicazione hanno i vari moduli, definendone il numero tramite la proprietà *num*. I canali di comunicazione vengono poi definiti in maniera specifica tramite la proprietà *channels_in* per gli ingressi e *channels_out* per le uscite; entrambe le proprietà devono contenere una lista composta dal tipo di dato che viene inviato o ricevuto tramite la proprietà *type* e dal nome del canale di comunicazione tramite *name*.

Si devono definire poi i nomi concettuali delle configurazioni tramite la proprietà *configuration-names*; i nomi sono importanti in quanto verranno utilizzati per identificare i vari moduli, quindi è importante essere coerenti con le successive dichiarazioni. Queste verranno poi utilizzate come macro nel codice SystemC, quindi si consiglia di scriverle in maiuscolo.

Si definiscono poi nello specifico i vari moduli tramite la proprietà *create*, in cui si definisce una lista composta dal tipo di configurazione (nome mnemonico a scelta del progettista) tramite la proprietà *type*, il nome del modulo riconfigurabile tramite *name*, il tempo di attivazione del modulo (ovvero il tempo che passa per allocarlo nella zona dinamica della FPGA) tramite *activation_delay* e l'unità di misura dell'attivazione tramite *activation_delay_measure*.

4.2.1.1.4 fsm-init

```
    fsm-init:
            num: #
            states: [*]
```

Indica che forma deve avere la EFSM, ovvero quanti stati deve avere e quali sono i nomi di questi stati. Tramite la proprietà *num* si indica quanti stati deve avere la EFSM, mentre la proprietà *states* si elencano i nomi degli stati di questa EFSM. Il numero degli stati deve essere coerente con la quantità di stati effettivamente descritti.

4.2.1.1.5 common-other-variables

```
    common-other-variables:
            - type: *
            name: *
```

Indica tutte le variabili che possono essere usate all'interno del codice in maniera globale. Tramite la proprietà *common-other-variables* si crea una lista contenente il tipo di varibile definibile tramite la proprietà *type* e il relativo nome tramite *name*.

4.2.1.1.6 specularity-code

```
    specularity-code: *
```

Con questa proprietà si definisce il codice in comune tra la parte statica e la parte dinamica del sistema. Il codice in comune possono essere funzioni (che non necessitano di doppia dichiarazione), macro o componenti base da includere nei moduli.

4.2.1.1.7 top-code

```
top-code:
        debug-activation: true, false
        producer:
                variables:
                        -type: *
                        name: *
                code: *
        controller:
                variables:
                        - type: *
                        name: *
                code: *
        monitor:
                code: *
```

Questa proprietà identifica il codice del modulo statico chiamato Top; con questa si definiscono i 3 sottomoduli precedentemente descritti (Producer, Controller e Monitor) con le relative variabili e codice.

Si può definire l'attivazione delle stampe di debug tramite la proprietà *debug-activation*, che può avere come valori true o false.

La proprietà *producer* identifica la descrizione del modulo Producer, si possono definire tramite le sue proprietà le variabili del modulo tramite *variables* definendo una lista di tipi di variabili con *type* e il relativo nome tramite *name*, per poi definire il codice di questo modulo tramite la proprietà *code*.

La proprietà *controller* identifica la descrizione del modulo Controller, si possono definire tramite le sue proprietà le variabili del modulo tramite *variables* definendo una lista di tipi di variabili con *type* e il relativo nome tramite *name*, per poi definire il codice di questo modulo tramite la proprietà *code*.

La proprietà *monitor* identifica la descrizione del modulo Monitor, si può definire il codice di questo modulo tramite la proprietà *code*.

4.2.1.1.8 modules-code

```
modules-code:
        descriptions:
                - type: *
                fsm:
                        -state: *
                        code: *
                other-code: *
```

Questa proprietà identifica il codice dei moduli dinamici; per ogni modulo si identificano gli stati della EFSM e il relativo codice.

Per ogni tipo di configurazione (precedentemente elencate nella proprietà *reconfigurable-modules-init*), si descrive la rispettiva EFSM, quindi tramite la lista di proprietà *type*, si definisce la EFSM tramite la proprietà *fsm*, identificando la lista di stati che si vogliono descrivere, scrivendo il nome nella proprietà *state* e il relativo codice nella proprietà *code*.

Se per ogni modulo si vuole aggiungere codice in più rispetto alla EFSM, si può definire la proprietà *other-code* e scrivere codice al suo interno.

4.2.1.1.9 common-code

```
common-code: *
```

Per il codice comune alla parte statica e dinamica o tra moduli riconfigurabili, si può definire la proprietà *common-code* e descrivere al suo interno le funzioni comuni. Si otterrà codice globale sfruttabile da ogni parte del sistema.

4.2.1.1.10 main

```
main:
        includes: *
        init: *
        code: *
```

Si deve descrivere il main del progetto tramite la proprietà *main*, in cui si definiscono gli includes di SystemC tramite la proprietà *includes*, l'inizializzazione delle variabli tramite la proprietà *init* e il codice del main tramite la proprietà *code*.

4.2.1.2 Metodologia di programmazione

La descrizione del sistema tramite il linguaggio consiste in 2 passaggi: la descrizione della parte di struttura delle singole parti del sistema (quindi parte statica e parte dinamica) e la descrizione del comportamento interno delle singole parti.

La prima parte consiste nel descrivere le porte di ingresso, le uscite, le variabili interne e i tipi di segnali che compongono le parti del sistema. Le parole chiave sono contraddistinte dalla presenza della stringa "*init*"

In questa parte si possono descrivere:

- il Top, ovvero la parte statica del sistema (tramite il tag "top-init"), di cui si possono definire le variabili ("variables"), i canali di input e output ("channels") e i canali di comunicazione con la parte dinamica ("portals") ;
- i moduli riconfigurabili (tramite il tag "reconfigurable-modules-init"), di cui si possono definire la quantità di moduli che il sistema può avere ("num"), il numero di porte di ingresso e uscita che può avere ("port:num"), il tipo di canali di comunicazione che può avere ("channels_in" e "channels_out"), il nome che si vuole dare alle diverse configurazioni ("configuration_names"), il primo modulo che deve attivarsi e le modalità di caricamento dei

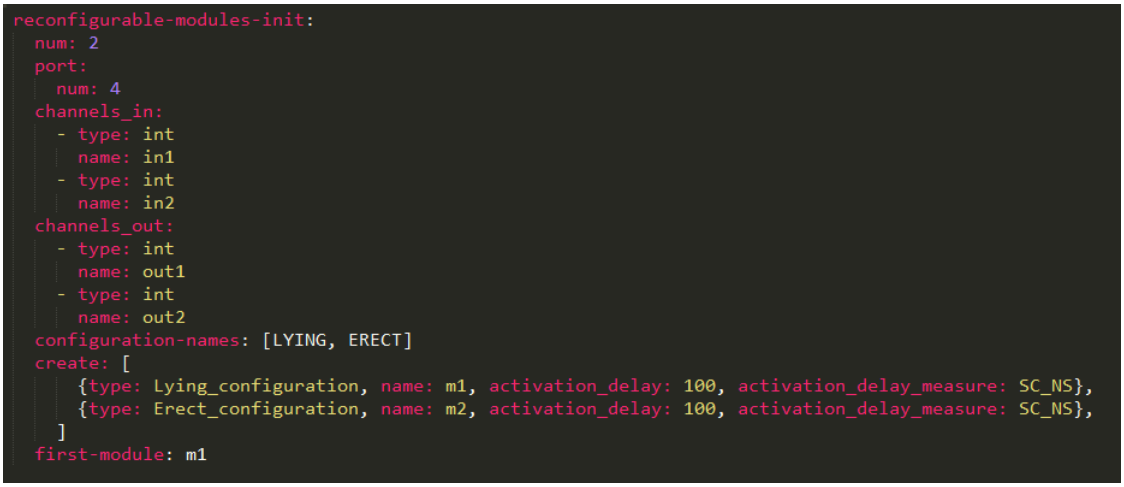

Figura 4.6: Esempio della prima parte della descrizione del sistema

moduli ("create", con cui si può definire il tempo di attivazione tramite "activation_delay" e l'unità di misura in cui viene espresso questo tempo tramite "activation_delay_measure");

- la FSM in comune tra i due sistemi (tramite il tag "fsm-init"), di cui si definiscono il numero di stati ("num"), e il nome degli stati ("states");

- altre variabili di supporto definite dal progettista (tramite il tag "common-other-variables"), di cui si identifica il tipo (tramite "type"), il nome (tramite "name") e il relativo valore se necessario (tramite "value");

Nella prima parte quindi si dà particolare enfasi alla parte di specifiche strutturali, dove è fondamentale capire gli input e gli output del sistema, oltre ad implementare il metodo corretto di comunicazione tra parte statica e parte dinamica del sistema.

La seconda parte invece consiste nello scrivere l'effettivo codice che verrà eseguito all'interno dei diversi sistemi. Le parole chiave sono contraddistinte dalla presenza della stringa "*code*" e il linguaggio permette l'inserimento di codice puro (raw) di SystemC.
In questa parte si possono descrivere:

- il codice in comune tra i due sistemi (tramite il tag "specularity-code");

- il codice del Top (tramite il tag "top-code"), in cui si può definire il codice di preparazione ("pre-code"), il codice del produttore ("producer"), del controller ("controller") e del monitor ("monitor");

- il codice dei moduli (tramite il tag "modules-code"), in cui si descrive, per ogni modulo associato ad un nome ("type") si descrivono gli stati della FSM ("fsm");

- il main del programma di simulazione (tramite il tag "main").

Dopo la scrittura del codice, si otterrà il file YAML che rappresenta il sistema ed è pronto per essere trasfomato nella descrizione in SystemC Rechannel tramite il tool automatico.

```yaml
modules-code:
  descriptions:
  - type: Lying_configuration
    fsm:
    - state: S_INIT
      code: "abbassati();"
    - state: S_WALK
      code: "striscia(1);"
    - state: S_STOP
      code: "wait_move(20);"
    - state: S_RECONFIG,
      code: "wait_move(20);"
    - state: S_LEFT
      code: "rotea_sx();"
    - state: S_RIGHT
      code: "rotea_dx();"
    other-code: |
     RC_COUTL("Lying_configuration: executed " << STATUS << " (t=" << sc_time_stamp() << ")");
  - type: Erect_configuration
    fsm:
    - state: S_INIT
      code: "abbassati();"
    - state: S_WALK
      code: "cammina(1);"
    - state: S_STOP
      code: "wait_move(10);"
    - state: S_RECONFIG
      code: "wait_move(10);"
    - state: S_LEFT
      code: "rotea_sx();"
    - state: S_RIGHT
      code: "rotea_dx();"
    other-code: |
      RC_COUTL("Erect_configuration: executed " << STATUS << " (t=" << sc_time_stamp() << ")");
```

Figura 4.7: Esempio della prima parte dell'implementazione di codice del sistema

4.3 Utilizzo del tool durante la progettazione

Il tool automatico risulta essere particolarmente efficace durante la progettazione, avendo fatto le giuste scelte implementative seguendo il flusso proposto. In particolare, il linguaggio riconosciuto dal tool, basato appunto su YAML, ha massimo potere espressivo nel contesto del flusso di progettazione, in quanto permette di descrivere ogni componente del sistema riconfigurabile da produrre. Se sono state fatte le scelte giuste (precedentemente descritte), la conversione delle FSM ottenute sarà molto facile seguendo il linguaggio, mentre scelte sbagliate renderanno la descrizione tramite il linguaggio leggermente più complesse (o comunque non efficaci).

Si suppone che comunque il tempo necessario per apprendere il linguaggio PynguLang sia molto minore rispetto ad apprendere il funzionamento della libreria SystemC Rechannel, quindi un guadagno di apprendimento si ha, oltre al fatto di ottenere codice che risulta essere safe e corretto per costruzione.

Capitolo 5

Caso di studio

Per verificare le potenzialità di questa architettura e del metodo di progettazione proposto, è stato implementato un progetto complesso che comprende molteplici aspetti avanzati dell'applicazione dell'informatica; in questo modo, si è proposto un esempio robusto per far notare i pregi (anche dal punto di vista della complessità di progettazione) di questa architettura. Il caso di studio consiste in un robot *intelligente* e *riconfigurabile* che deve essere in grado di affrontare un percorso ad ostacoli per raggiungere una fonte di calore, quindi calato in un contesto di robot di salvataggio che è in grado di ricercare persone in ambienti avversi o identificare fonti di incendi e tentare di domare le fiamme. Il tutto è stato implementato con diversi tool di simulazione che sono stati fatti comunicare e rispettando i vincoli temporali di un sistema reale (quindi utilizzando clock, tempi fisici realistici con tutti i problemi di hard realtime annessi). Sono stati utilizzati quindi i seguenti tool di simulazione:

- *Blender* per la progettazione e implementazione della parte fisica ed estetica del robot;
- *V-REP* per la simulazione del robot implementato in Blender, utilizzando il più possibile una fisica realistica;
- *SystemC Rechannel* per la simulazione dell'hardware riconfigurabile, che comanda i motori del robot in modo che possa muoversi e riconfigurarsi secondo l'ambiente;
- *Linux Ubuntu* per simulare il programma di intelligenza artificiale che dovrebbe essere eseguito su un qualsiasi processore e che deve interfacciarsi con la parte riconfigurabile (implementata in SystemC Rechannel).

Il funzionamento del sistema è il seguente: il robot può avere 2 differenti configurazioni, ovvero configurazione a quadrupede e configurazione a doppio serpente; queste configurazioni vengono pilotate e mosse dal sistema riconfigurabile in base al percorso che il robot deve eseguire. Il robot è sensibile alle fonti di calore, ovvero tramite sensore di visione di calore, è in grado di percepire dove si trova una fonte di calore e capire quanto è distante da esso. Per capire quanto è distante dalla fonte di calore, viene fatto uno studio statistico sulla quantità di pixel rossi che appaiono nell'immagine acquisita. Il robot non conosce l'ambiente in cui si trova, si deve solo basare sui sensori di distanza posti su di esso, uno per ogni lato, e sul sensore di visione; non deve interagire con l'ambiente ma deve solo essere in grado di non essere ostacolato nei suoi movimenti. Il task del robot consiste nel raggiungere o di avvicinarsi il più possibile alla zona di calore e, per farlo, deve riuscire a trovare il percorso migliore (ovvero quello fattibile). La strategia del robot consiste nel "provare" quale delle 2 sue configurazioni può essergli utile per raggiungere l'obbiettivo: durante

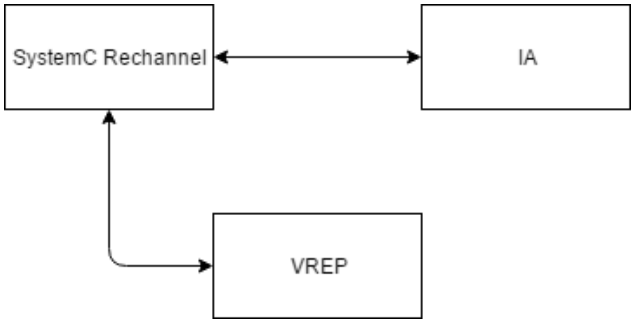

Figura 5.1: Struttura del caso di studio

il suo tragitto, potrebbe trovare un ostacolo che gli impedisce di proseguire, quindi deve essere in grado di oltrepassarlo o passandoci sotto oppure raggirandolo. Deve inoltre essere in grado di capire se la scelta di strategia per oltrepassare l'ostacolo è corretta: è ammesso che il robot esegua degli sbagli durante le sue scelte, in quanto non ha una mappatura dell'ambiente, ma deve essere in grado di tornare indietro nelle sue scelte, capire quando è avvenuto l'errore della sua scelta e quindi adottare una strategia diversa per raggiungere il target. La scelta del tragitto viene sempre attuata in base a dove si trova la fonte di calore: l'euristica è appunto quella di minimizzare il tragitto andando in direzione sempre dove la concentrazione di pixel rossi è maggiore. Ovviamente in questo modo il robot non sa dove si trovano gli ostacoli in quanto è in grado di "vedere" soltanto le fonti di calore, quindi il suo tragitto sarà pieno di scelte.

A causa della difficoltà reale del progetto, sono state fatte diverse assunzioni:

- il terreno di simulazione è liscio e piatto;
- gli ostacoli sono semplici e facilmente individuabili dai sensori del robot (quindi composti da figure solide semplici);
- l'ambiente risulta statico, quindi non ci sono ulteriori interazioni col robot;
- i movimenti del robot sono rettilinei, a meno di problemi dovuti alla progettazione manuale del robot (giunti non perfettamente allineati, misure approssimative, errori del simulatore).

Si vedranno ora nel dettaglio le diverse implementazioni dei componenti.

5.1 Hardware riconfigurabile

Il sistema riconfigurabile svolge la funzione di moto, riconfigurazione e sensoristica del robot: tutta la parte di cinematica viene delegata a questa parte, con i relativi controlli sui sensori. Si potrebbe paragonare ad un sistema nervoso adattivo: esso è in grado di adattarsi all'ambiente percependolo, ma non comprendendolo. La comprensione di tutti i segnali viene delegata alla parte di intelligenza artificiale, mentre l'hardware riconfigurabile si occupa di acquisire dati e di inviarli al "sistema di controllo e comprensione" e riconfigurarsi in base all'ambiente circostante. Questo è stato implementato in simulazione su SystemC Rechannel ed è composto da 2 entità:

- *Top*, che contiene la parte di controllo del sistema riconfigurabile che si interfaccia direttamente con i moduli riconfigurabili; è in grado di dare il comando di riconfigurazione ai moduli riconfigurabili e di acquisire i segnali dai sensori del robot per trasformarli in dati comprensibili alla parte di intelligenza;

- *Moules*, che contiene i moduli riconfigurabili che eseguono gli algoritmi di movimento.

L'algoritmo di funzionamento è il seguente: il Top controlla i sensori del robot e li invia al sistema di intelligenza artificiale che li elabora; successivamente il Top riceve il comando in base ai valori dei sensori e il Top, in base alla sua macchina a stati, invia l'azione da eseguire al modulo riconfigurabile attualmente attivo. Parallelamente il Top controlla se deve essere eseguita una riconfigurazione: se deve essere riconfigurato il sistema, dealloca il modulo riconfigurabile e attiva quello prescelto, in modo che venga eseguita la prossima azione su quel modulo. Contemporaneamente il modulo riconfigurabile attualmente attivo esegue l'azione che gli è stata precedentemente inviata dal Top, per poi attendere il prossimo comando.

Il Top è composto quindi dai seguenti componenti:

- *activator*, che si occupa di attivare e deattivare i moduli riconfigurabili;
- *producer*, che invia i dati ai moduli riconfigurabili;
- *controller*, che si occupa di tenere traccia della riconfigurazione attuale e di quella futura;
- *monitor*, che si occupa di leggere i dati di risposta dei moduli riconfigurabili.

Si ha quindi anche parallelismo interno al modulo Top, in quanto deve essere in grado di gestire contemporaneamente i diversi aspetti della riconfigurazione. Questo parallelismo è gestito da SystemC Rechannel, però lo scambio di messaggi (che risulta bloccante secondo l'architettura Client-Server rispettivamente dei moduli riconfigurabili e del Top) viene gestito dal progettista.

5.2 Robot riconfigurabile

È stato creato un robot riconfigurabile, ovvero capace di mutare la propria forma in modo da presentare una cinematica differente per ogni riconfigurazione. La caratteristica principale di questi robot è di essere composti da moduli semplici che, uniti tra loro, creano una struttura più complessa.

5.2.1 I robot riconfigurabili

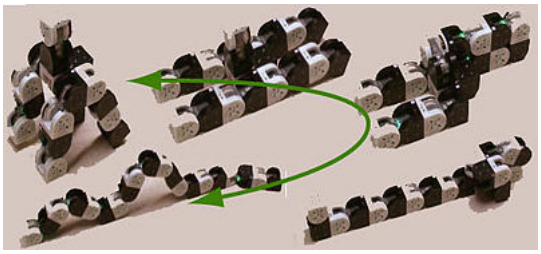

Figura 5.2: Un robot riconfigurabile (M-TRAN III)

I robot riconfigurabili sono macchine con cinematica autonoma che possiedono una morfologia variabile. Con morfologia variabile si intende la capacità di mutare forma e quindi la configurazione dei giunti, in modo da permettere al robot di compiere movimenti diversi, quindi avere una cinematica diversa. La caratteristica principale di questi sistemi è quindi di riarrangiare la connessione tra le parti in base a determinate circostanze, in modo da eseguire diversi task, per

tornare in situazioni safe, o per ripararsi. Questi si dicono anche "robot modulari" in quanto sono composti da piccoli componenti (moduli) che si riassemblano tra loro e creano il robot nella sua completezza.

Questi sistemi uniscono diversi aspetti della robotica e di altre discipline, come la progettazione di sistemi embedded, le reti di sistemi embedded e l'intelligenza artificiale, poiché questi sistemi hanno bisogno di una visione completa del mondo circostante ma soprattutto dei pezzi di cui sono composti.

Un robot riconfigurabile è composto da un insieme omogeneo di moduli, che sono tutti simili di forma e di cinematica tra loro (solitamente con 1 grado di libertà) che sono in grado di comunicare e "organizzarsi", ovvero coordinarsi per collaborare. La collaborazione serve ai moduli per "capire" come collegarsi tra loro e creare una struttura differente per ogni situazione.

Esistono differenti tipi di robot riconfigurabili, tutti raggruppati in 2 macro aree:

- *parzialmente riconfigurabili*, che sono composti da moduli precedentemente assemblati che, con delle forme prestabilite si riconfigurano in modo da cambiare la morfologia del robot (come in questo caso di studio);

- *completamente riconfigurabili*, che non hanno una forma prestabilita e tutti i moduli sono inizialmente separati, per poi comporsi tra loro completamente in base alla situazione.

Questo stile di progettazione del robot porta essenzialmente vantaggi funzionali e vantaggi economici: i primi sono dovuti alla potenziale robustezza e adattabilità del robot rispetto all'ambiente, rispetto ai robot convenzionali; i secondi derivano dal fatto di costi di produzione ridotti (si creano moduli tutti uguali in serie) e costi di progettazione sia hardware sia software ridotti (si progetta un singolo modulo e si crea un software di coordinazione che è valido per ogni modulo).

Questi robot non sono ancora diventati effettivamente utilizzati in quanto presentano degli svantaggi logistici piuttosto importanti: oltre al fatto che sono ancora in fase sperimentale, non vengono ancora sfruttati metodi ingegneristici per la progettazione, in particolare non è ancora stata studiata una tipologia di hardware e software che, accoppiati, potessero massimizzare le performance e potessero minimizzare i consumi, oltre al fatto che non esiste un modo coerente e standard per simulare e quindi testare il sistema completo. Il problema principale infatti di quando si progetta il robot è avere una rappresentazione abbastanza corretta e simile alla realtà, in modo da passare alla fase di produzione per il test reale in maniera piuttosto trasparente ed efficace (minimizzando quindi i costi di progettazione e produzione).

Un aspetto teorico legato a questi robot è la modellazione simile a quella che si utilizza quando si progetta un nuovo linguaggio. Questa infatti è stata formalizzata con aspetti di di analisi "sintattica" e "semantica": poiché sono strutture difficili da studiare, il moto di questi robot viene studiato attraverso la definizione di un linguaggio, quindi definendo la relativa grammatica (movimenti base dei moduli), sintassi (movimenti ammissibili per i moduli collegati) e semantica (il senso del movimento globale). Quindi, lo studio meccanico della cinematica si limita al semplice modulo, per poi "orchestrare" l'insieme dei moduli all'unisono [9]. Ogni robot riconfigurabile ha quindi una rappresentazione secondo un linguaggio: tramite questo, si riesce a modellare correttamente ed in maniera efficace tutte le possibili configurazioni e transizioni compatibili tra esse.

Queste informazioni si adattano perfettamente al modello di hardware riconfigurabile proposto: la sintassi e la semantica vengono gestiti dal Top, mentre la grammatica viene gestita dai moduli riconfigurabili, il tutto gestito da una macchina a stati per tenere traccia dei movimenti da eseguire (ma soprattutto quelli possibili).

5.2.2 Progettazione

Il robot è stato progettato appositamente per questo progetto, riprendendo il modello del robot serpente di esempio del simulatore V-REP. Questo non presenta una riconfigurabilità totale, in quanto, a causa del simulatore, non era possibile rappresentare un robot che presentasse giunti semovibili. In ogni caso, il robot è in grado di riconfigurarsi grazie alle direttive dell'hardware riconfigurabile.

Come primo passaggio è stato disegnato il singolo modulo per capire come poteva essere più funzionale per il progetto finale, per poi assemblarli e creare il modello finale del robot, in posizione di riposo. È stato quindi creato un file *.obj* per poter essere importato in un programma di simulazione che potesse supportare l'importazione di mesh poligonali.

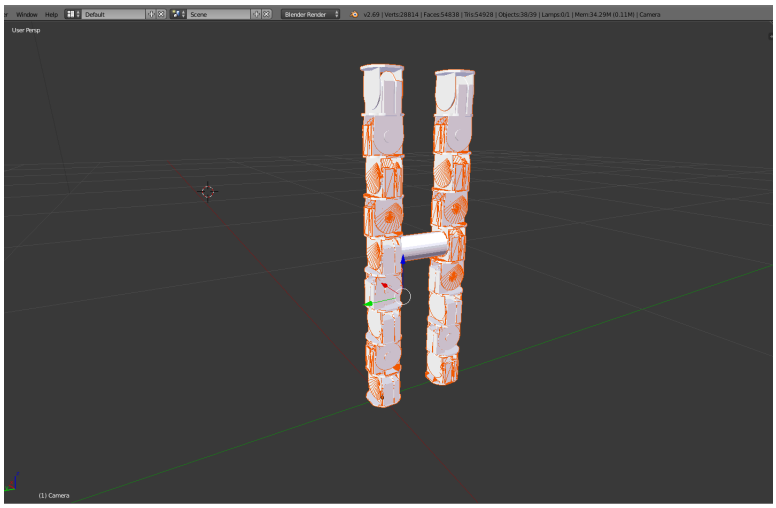

Figura 5.3: Progettazione su Blender del modello del robot

Il robot è poi stato importato nel programma di simulazione V-REP. Una volta importato, è stato necessario creare il modello dei moduli e quindi del robot completo definendo i punti di collisione, ovvero gli oggetti solidi che compongono il robot. Il modello di mesh infatti non possiede proprietà fisiche dell'oggetto, è compito del progettista (anche se si cerca di ricavarli in maniera automatica dalle mesh) generare i punti di collisione dalla mesh. Una volta creati i punti di collisione, è stato verificato che il robot rimanesse in equilibrio statico; a causa della discretizzazione (e della creazione non molto precisa con Blender), il robot rimane in equilibrio statico ma tende a "scivolare" durante la simulazione. Purtroppo non è stato possibile aumentare il coefficiente di attrito dell'oggetto, quindi si ha una simulazione poco precisa.

In ambiente VREP poi sono stati aggiunti 1 sensore di visione da 64x64 px e 5 sensori di prossimità; il sensore di visione, posto sopra il robot, è stato impostato per vedere solo le fonti di calore (che rappresentano il target di movimento) e con una bassissima risoluzione, in modo da minimizzare il calcolo da parte dell'hardware riconfigurabile, mentre i sensori di prossimità hanno il fascio cilindrico e sono stati posti ad ogni lato del robot, leggermente sopraelevati rispetto alla base del robot in modo che i movimenti del robot non possano interferire con la rilevazione degli ostacoli.

Dal punto di vista di controllo, il robot viene controllato in base alle posizioni dei giunti (controllo di posizione), quindi ogni istante della simulazione viene inviato un comando di posizione a tutti i giunti del robot per eseguire un movimento. A causa della natura del sistema

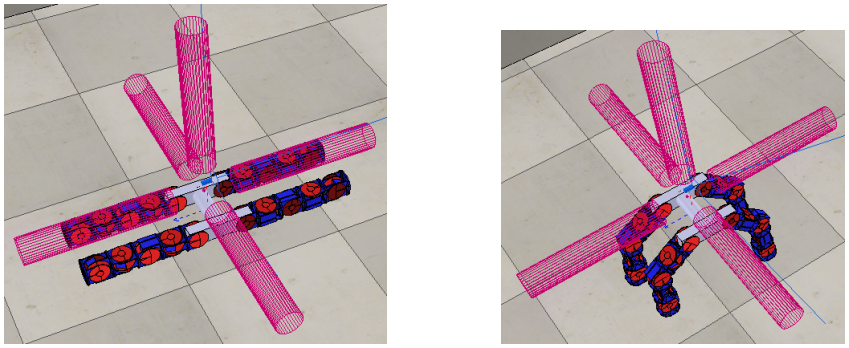

Figura 5.4: Robot nelle sue 2 configurazioni: a serpente doppio e a quadrupede

che lo comanda (hardware riconfigurabile), ogni posizione è discreta, quindi il moto non è stato descritto con un'equazione nel continuo ma da una serie di posizioni dei giunti.

Il robot è composto dalle seguenti parti:

- diversi componenti locomotori connessi in maniera opposta l'uno dall'altro, che uniti tra loro permettono di eseguire molteplici movimenti;
- un corpo centrale a forma di "H", in cui risiede tutta la circuiteria e permette di avere una base solida per alcuni tipi di movimenti.

Ogni componente pesa $40g$ e deve avere l'attrito necessario per permettere il movimento; sfortunatamente il tool di simulazione non permette di impostarlo e quindi alcuni movimenti non rappresentano correttamente la realtà.

Per le diverse configurazioni, sono stati presi diversi modelli naturali per rappresentare il moto: la configurazione a serpente doppio ha preso spunto dal moto del serpente (moto ondulatorio verticale), mentre la configurazione a quadrupede ha preso spunto dalla marcia del cavallo (passo).

5.2.3 Configurazione a serpente doppio

In questa configurazione, il robot si trova sdraiato, completamente a contatto con il terreno. Per muoversi, deve eseguire un moto ondulatorio verticale, come alcuni tipi di serpenti. Questo tipo di moto può essere descritto con queste equazioni:

$$y(t_x) = (j_{v1} \cdot (1-s) + j_{v2} \cdot s) \cdot \sin(t \cdot (v_{v1} \cdot (1-s) + v_{v2} \cdot s) + i \cdot (j_{v1} \cdot (1-s) + j_{v2} \cdot s))$$

$$y(t_{x+1}) = (j_{v1} \cdot (1-s) + j_{v2} \cdot s) \cdot \cos(t \cdot (v_{v1} \cdot (1-s) + v_{v2} \cdot s) + i \cdot (j_{v1} \cdot (1-s) + j_{v2} \cdot s))$$

la variabile y indica l'output della funzione, t_x indica un istante di tempo generico mentre t_{x+1} indica l'istante di tempo successivo al precedente. Le variabili j_{v1}, j_{v2} indicano le posizioni dei giunti verticali, indipendentemente da quanti giunti esistono (come nel caso del tempo, si indicano semplicemente giunti successivi tra loro). Le variabili v_{v1}, v_{v2} indicano invece le velocità dei giunti successivi tra loro. Si deve pensare che queste 2 equazioni si devono eseguire ciclicamente in modo da avere un movimento continuo ed ondulatorio, per permettere di strisciare.

A causa della natura del robot, poichè la parte centrale è statica, si deve eseguire il movimento con più potenza del necessario, in modo da essere in un'istantanea fase di volo che può dare la spinta necessaria per avanzare (e quindi spostare la parte statica del robot).

Per la rappresentazione spaziale, è stato adottato il modello di *Chirikjian - Burdick*[11], in cui si prende come sistema di riferimento fisso la "coda" del serpente, cioè l'ultimo giunto, e tutti i calcoli relativi ai successivi giunti del corpo serpentoide del robot verranno eseguiti in base a tale scelta. In particolare, il sistema viene visto come un insieme di 4 serpenti collegati ad un corpo rigido: in questo modo i movimenti vengono semplicemente ripetuti in maniera speculare o come prolungamento di altri serpenti, semplicemente riferendosi ai giunti sequenziali del robot.

Poichè ogni posizione di ogni giunto è nota durante l'esecuzione, è stata utilizzata la cinematica diretta per determinare la posizione e il movimento di ogni componente. Inoltre, poiché il numero dei giunti era molto elevato, è stata utilizzata la convenzione di *Denavit-Hartenberg*[12], rappresentando una trasformazione geometrica nello spazio euclideo tridimensionale con 4 parametri. Calcolando questi valori, è stato possibile quindi calcolare la posizione dei giunti in maniera ottimale per il corretto movimento del robot.

Quindi, implementando ogni posizione dei giunti, si è riusciti a far muovere il robot con il moto ondulatorio verticale.

In questa configurazione è anche possibile muoversi lateralmente, rovesciando il robot lateralmente 2 volte consecutive. Per fare ciò, è necessario piegare contemporaneamente 2 giunti dello stesso lato, in modo che il robot si possa rovesciare nel lato opposto, per poi rieseguire lo stesso movimento con gli altri 2 giunti.

5.2.4 Configurazione a quadrupede

In questa configurazione, il robot si trova in posizione eretta sui suoi 4 arti che nella configurazione precedentemente descritta erano i 4 serpenti. Per questa configurazione non si è trovato un modello analitico per descrivere il moto del robot, quindi ci si è serviti dello studio cinematico di un cavallo, in particolare della cinematica del "passo". Il passo è la camminata leggera del cavallo, che consiste nel muovere alternativamente e a coppie disallineati gli arti opposti, senza avere una fase di volo; in questo modo, si ha sempre la condizione di equilibrio una volta completato l'insieme di movimenti. La caratterstica di questo movimento però è lo sbilanciamento verso la direzione in cui ci si muove: gli arti vengono alzati in modo che avvenga uno sbilanciamento in avanti in modo che, raddrizzandoli successivamente, si è riusciti ad avanzare.

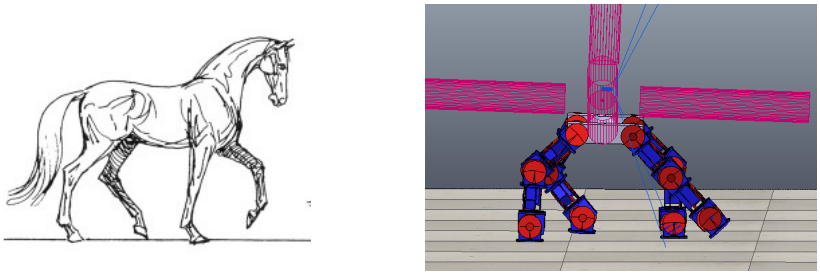

Figura 5.5: Confronto tra il "passo" del cavallo e il "passo" del robot

Per implementare il moto del robot quindi, è stato fatto uno studio cinematico del passo del cavallo, per poi riportare le posizioni dei "giunti" del cavallo nelle posizioni dei giunti del robot; in questo modo si è discretizzato il movimento del robot per poi permettergli di muoversi secondo

un modello naturale funzionante, oltre a fare uno studio dell'anatomia del cavallo per capire come renderla compatibile con la configurazione a quadrupede del robot. Utilizzando un controllo di posizione infine, si è riusciti facilmente ad implementare il corretto movimento: servendosi di uno studio di settore già stato eseguito [10], è stata semplicemente trascritta la posizione attesa dei giunti e rappresentata in simulazione.

5.3 Intelligenza Artificiale

Per completare il progetto, era necessario utilizzare un programma intelligente che potesse gestire il path planning; questo perché sarebbe troppo poco performante far eseguire il tutto all'hardware riconfigurabile, quindi si è implementato un programma esterno per gestire l'aspetto di pianificazione del percorso e il collision avoidance.

Il programma, scritto in C++, è in grado di interpretare i dati ricevuti dall'hardware riconfigurabile e prendere decisioni sul percorso da eseguire: i dati sono organizzati secondo un protocollo specifico, ovvero un ordine di segnali che corrispondono all'individuazione da parte di un sensore di un oggetto vicino. Il programma, sceglie il percorso durante l'avanzamento del robot nello spazio, memorizzando ad ogni passo la scelta eseguita; questo è molto importante per comprendere se durante il tragitto sono state fatte delle scelte sbagliate. Il concetto di "scelta sbagliata" in questo contesto è semplice: se si ha una una situazione di impossibilità di movimento (il robot non può più eseguire l'azione che stava eseguendo, ci sono troppi ostacoli, ecc), allora il programma ha scelto un tragitto sbagliato. Dal momento che il programma si rende conto che ha commesso un errore di scelta, deve essere in grado di capire quando è stato fatto lo sbaglio, ovvero deve tornare indietro eseguendo i tutti i movimenti al contrario fino all'ultimo movimento errato, ovvero quando ha iniziato a fare il movimento che l'ha portato nella situazione errata. Alla fine dell'esecuzione, il programma ha mantenuto la lista di comandi che hanno portato il robot a raggiungere l'obbiettivo, quindi la stessa lista si potrebbe utilizzare potenzialmente per eseguire il percorso senza commettere errori.

Il programma implementa quindi la tecnica di *A Star* con la variante *Fog War and no path information*: si utilizza un'euristica per raggiungere la fonte di calore, ovvero mantenere centrata la fonte di calore nel visualizzatore, senza però sapere com'è fatto il percorso, avvolto quindi da una "nebbia di guerra" che impedisce di sapere cosa si trova dinanzi. Il concetto di nebbia di guerra è stato importato dai videogiochi e quindi dall'intelligenza artificiale dei personaggi, in quanto in alcuni tipi di giochi strategici è stata implementata una mappa le cui informazioni interne non sono note, devono essere scoperte dal videogiocatore o dal nemico virtuale (che quindi deve viaggiare nella mappa pur non sapendo che ostacoli può trovare). La tecnica dell' A star si è dimostrata molto efficace, in quanto il percorso risulta discretizzato in base ai movimenti possibili del robot e la visione della mappa viene modificata dalle 2 configurazioni. Si può pensare quindi di avere una mappa a 2 livelli, i quali possono essere sostituiti a piacere in base alla situazione.

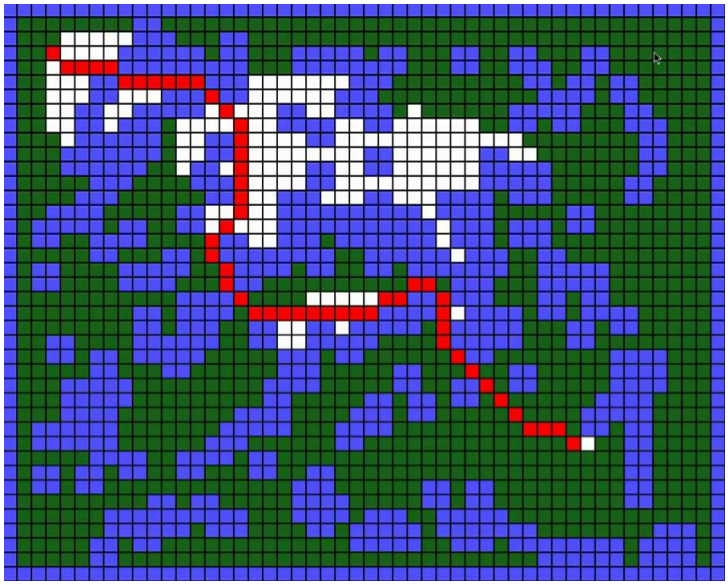

Figura 5.6: Mappa in cui utilizzare A Star (percorso in rosso, ostacoli pieni in verde, ostacoli vuoti in bianco)

5.3.1 Algoritmo

L'algoritmo intelligente definisce una serie di regole che il robot deve adottare in modo da trovare il percorso corretto per raggiungere l'obbiettivo. Questo quindi consisterà in una serie di letture dei dati derivanti dai sensori, che verranno interpretati e quindi si deciderà quale azione eseguire. Da notare che l'ordine dei controlli è molto importante: si definisce una priorità di controlli in modo che, a parità di configurazione, la priorità delle scelte possa cambiare l'esecuzione del programma.

Il programma sarà sempre sincronizzato con il sistema sottostante, non ha visione del mondo che lo circonda ma semplicemente interpreta tramite regole prestabilite i segnali che arrivano dai sensori. Infine, deve essere un algoritmo semplice e che non richieda troppo utilizzo di memoria e risorse, in quanto deve essere più veloce possibile e calato in un contesto di un microcontrollore che si trova a bordo del robot, quindi indubbiamente con non ottime capacità di calcolo.

```
Data: Robot input
Result: Movement decisions
initialization;
while not arrived do
    read current data;
    if I can go ahead then
        go ahead;
    else if I can go ahead but I'm lying then
        go ahead dragging;
    else if I'm under the obstacle then
        go ahead dragging;
        memorize that I'm under the obstacle;
    else if I'm not under the obstacle then
        go ahead dragging;
        memorize that I'm not under the obstacle;
    else if I can get up then
        get up;
    else if I can't go ahead for an obstacle then
        crouch;
    else if I can't go ahead lying (try to go right) then
        go right;
        memorize that I'm going right;
    else if I can't go ahead lying (try to go left) then
        go left;
        memorize that I'm going left;
    else if I can't try another movement to right then
        rollback actions;
    else if I can't try another movement to left then
        rollback actions;
    else
        I don't know, abort;
end
```

Algorithm 1: Algoritmo di path planning del robot

Capitolo 6

Conclusione e sviluppi futuri

Si ripercorrono ora i passi descritti precedentemente per dare rilevanza al lavoro svolto. Innanzitutto si è partiti dai diversi tipi di architetture hardware ora esistenti, focalizzandosi in particolar modo sui sistemi metamorfici, in quando si basano su un concetto di evoluzione molto utile per risolvere particolari problemi e dare valore aggiunto alle tecnologie hardware; rimane il problema di trovare una tecnologia che riesca ad implementare questo concetto, trovando nella tecnologia dell'hardware riconfigurabile una buona strategia implementativa. Contrariamente alle altre tecnologie però, non esiste una metodologia di sviluppo rigorosa ed ingegnerizzata per progettare questi sistemi, quindi si è cercato di creare un ambiente di sviluppo, analisi e sintesi in modo da sfruttare al meglio questi sistemi. Si è quindi creata, tramite una libreria caduta in disuso (SystemC Rechannel), una metodologia di sviluppo di questi sistemi basata sulla creazione di una EFSM associata ai diversi comportamenti del sistema, dividendo rigorosamente una parte dedicata alla riconfigurazione e una parte statica. Inoltre per l'implementazione velocizzata, è stato creato un linguaggio apposito per la descrizione facilitata del sistema, oltre a rendere possibile ad ogni progettista di progettare un sistema del genere senza conoscere le problematiche di fondo e senza conoscere la libreria di supporto di SystemC. È stato provato questo approccio con un progetto di esempio che rappresenta al meglio questo sistema, dimostrando che con questi sistemi si può implementare abbastanza facilmente qualcosa di molto complesso.

Come sviluppi futuri si potrà sicuramente migliorare il tool Pyngu per renderlo più efficiente, oltre a testare su FPGA l'effettivo funzionamento di SystemC Rechannel generato dal tool automatico.

Bibliografia

[1] Garrison W. Greenwood, Andy M. Tyrrell, "*Metamorphic Systems: A New Model for Adaptive System Design*", Evolutionary Computation (CEC), 2010 IEEE Congress on, 2010

[2] Sekanina, L. ; Brno Univ. of Technol., Brno ; Martinek, T. ; Gajda, Z., *Extrinsic and Intrinsic Evolution of Multifunctional Combinational Modules*, Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, 2006

[3] R. Kenny, D. Rupe, *FPGA Run-Time Reconfiguration: Two Approaches*, Altera, 2008

[4] Dirk Koch, *Partial Reconfiguration on FPGAs*, Springer, 2013

[5] Alisson V. Brito, Matthias Kuhnle, Michael Hubner, "*Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC*", IEEE Computer Society Annual Symposium on, 2007

[6] F. Cancare; C. Pilato; A. Cazzaniga; D. Sciuto; M. D. Santambrogio, "*D-RECS: A complete methodology to implement Self Dynamic Reconfigurable FPGA-based systems*", Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on, 2013

[7] Raabe A., Felke A., *A SYSTEMC language extension for high-level reconfiguration modelling*, Specification, Verification and Design Languages, 2008. FDL 2008. Forum on, 2008

[8] , Andreas Raabe , *Describing and Simulating Dynamic Reconfiguration in SystemC Exemplified by a Dedicated 3D Collision Detection Hardware*, 2008

[9] Masahiro Fujita Hiroaki Kit an0 Toshitada Doi, *Syntactic-Semantic Analysis of Reconfigurable Robot*, Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on, 1999

[10] S.Makita, N.Murakami, M.Sakaguchi, J.Furusho, *Development of Horse-type Quadruped Robot*,Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on, 1999

[11] G. S. Chirikjian; J. W. Burdick, *Kinematics of hyper-redundant robot locomotion with applications to grasping*, Proceedings. 1991 IEEE International Conference on Robotics and Automation, 1991

[12] Jacques Denavit, Richard S. Hartenberg, *A kinematic notation for lower-pair mechanisms based on matrices*, in Trans ASME J. Appl. Mech, n° 23, 1955, pp. 215-221