UNIVERSITÀ DEGLI STUDI
DEL SANNIO Benevento

UNIVERSITÀ DEGLI STUDI DEL SANNIO

DIPARTIMENTO DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica
per l'Automazione e le Telecomunicazioni

Implementazione su FPGA di un circuito elettronico per il controllo di un braccio meccanico

Relatore
Ch.mo Prof.
Marco Pisco

Candidato
Antonio Rapuano
Matr. 862000695

Anno Accademico 2020/2021

INDICE

Introduzione

Il recente sviluppo dell'elettronica digitale e il suo impiego nel controllo dei processi hanno condotto al rinnovamento del concetto di automazione. Nell'accezione odierna quest'ultima consiste nell'impiego di dispositivi elettronici e meccanici per manipolare grandezze fisiche con lo scopo di ridurre la necessità dell'intervento umano, quindi per l'esecuzione di operazioni ripetitive, complesse, rischiose o che richiedono un'elevata accuratezza.
Al fine di realizzare una macchina in grado di controllare un determinato processo in modo che si comporti nella maniera desiderata, è necessario costituire un sistema di controllo, il quale è formato dall'insieme del processo da controllare e dei dispositivi utili al suo controllo (come sensori e calcolatori). In genere un sistema di controllo è governato da uno specifico algoritmo, il quale può essere implementato per mezzo di microcalcolatori dedicati o di controllori logici programmabili (PLC).
Uno degli esempi più evidenti di automazione è la robotica industriale: all'interno delle linee di produzione è ormai divenuto canonico l'impiego di sistemi robotici, in grado di affiancare e sostituire l'essere umano in modo efficace. In questo contesto, la maggior parte dei dispositivi impiegati rientra nella categoria dei bracci meccanici, i quali sono generalmente in grado di compiere gran parte dei compiti di manipolazione delle merci e dei materiali con un grado di efficacia, rapidità e qualità di lavoro ben superiore alla controparte biologica.[1]
L'utilizzo di queste macchine risulta estremamente utile anche in altri ambiti: si pensi ad esempio al lander *InSight*, dispiegato dalla NASA su Marte nel novembre del 2018, che vanta un braccio robotico chiamato *Instrument Deployment Arm*, utile ad adoperare un sensore per la misura del flusso termico; oppure al sistema chirurgico *da Vinci*, il quale dispone di quattro bracci pilotabili da un medico per svolgere diversi tipi di procedure chirurgiche miniinvasive.

Il presente lavoro di tesi riguarda lo studio, la progettazione e la realizzazione di un sistema di controllo in grado di pilotare degli attuatori con lo scopo di controllare il movimento di un semplice braccio meccanico flessibile. In particolare, per soddisfare questo proposito è stato ritenuto opportuno fare impiego di alcuni piccoli servomotori (della tipologia da modellismo), i quali possono essere governati tramite specifici segnali impulsivi modulati in larghezza (definiti PWM), facilmente generabili attraverso circuiterie digitali. Per l'implementazione circuitale è stato scelto di avvalersi della logica programmabile FPGA, e in particolare della scheda di sviluppo Altera DE2-115, corredata da un kit compatibile per il pilotaggio dei servomotori.

Questa sezione introduttiva fa da preambolo al corpo della tesi, diviso in due parti: *panoramica del progetto* e *realizzazione del prototipo*. Nella prima parte, composta da tre capitoli, vengono presentati e descritti le risorse, i componenti e gli strumenti, sia hardware che software, alla base del funzionamento del sistema di controllo; nella seconda si passa alla realizzazione pratica vera e propria della circuiteria digitale e di un prototipo di braccio robotico, arrivando ad un totale di cinque capitoli. Nelle pagine finali si propongono alcune conclusioni tratte dall'attività di tesi e i riferimenti bibliografici utilizzati.

Parte prima: panoramica del progetto

Capitolo 1

Controllare un braccio meccanico

Il controllo di un braccio meccanico flessibile è un compito che può essere portato a termine soltanto se si dispone dei mezzi adeguati. Sorvolando sulla progettazione della circuiteria digitale di controllo, relegata alla seconda parte della tesi, è chiaro che la struttura fisica della macchina debba essere messa in movimento da un qualche tipo di attuatore. Se l'obiettivo è l'emulazione dei gradi di libertà di un braccio umano, allora è necessario che gli attuatori impiegati riescano a ruotare un supporto rigido nei limiti di un angolo sufficientemente grande. In questo modo sul medesimo supporto rigido è possibile installare un secondo attuatore dello stesso tipo, così da garantire al braccio un'ampia gamma di possibili movimenti, e quindi la capacità di rivelarsi utile per lo svolgimento di diversi compiti.

1.1 Servomotori

Tipicamente per applicazioni di questo tipo la scelta cade tra due categorie di dispositivi: i motori passo-passo e i servomotori. Il vantaggio di entrambi i dispositivi è che generalmente possono essere adoperati a ciclo aperto; dunque, per il controllo della posizione non c'è bisogno che il sistema che li pilota sia dotato di retroazione (o feedback), riducendo complessità e costi. D'altra parte, la differenza sta principalmente nelle prestazioni. Un motore passo-passo è in grado di ruotare in modo continuo con un controllo preciso del numero di passi da compiere, ed è indicato per applicazioni a basso costo, senza particolari esigenze di velocità, coppia o accuratezza. Al contrario un servomotore è caratterizzato da un costo maggiore, motivato dal fatto che si compone di un sofisticato sistema per il raggiungimento e il mantenimen-

to dell'uscita desiderata, in genere una posizione angolare compresa tra 0 e 180°; inoltre, spesso può garantire prestazioni superiori a quelle di un motore passo-passo appartenente alla stessa fascia di prezzo.[2]
Dunque, dato che per molti possibili scenari è auspicabile che il braccio sia dotato di buoni livelli di forza e accuratezza (basti pensare all'ambito industriale), e visto che non c'è bisogno che le articolazioni ruotino in modo continuo, per questo progetto è stato ritenuto opportuno fare uso di servomotori.[3]

Come accennato, un servomotore è un attuatore in grado di controllare la posizione, la velocità o l'accelerazione di rotazione di un albero motore in modo conforme al suo input. Fondamentalmente consiste in un sistema di controllo a ciclo chiuso composto da tre elementi fondamentali:

- un motore, in genere elettrico, la cui tipologia varia in base alle specifiche di progetto (può essere in corrente continua o alternata, con spazzole o senza, e così via). Il compito del motore è quello di trasformare un segnale d'ingresso nel movimento dell'albero
- un sensore di rotazione (come un potenziometro, un encoder rotativo o un tachimetro), in grado di misurare la rotazione dell'albero per comunicarne il valore in retroazione
- elettronica di controllo, che essenzialmente implementa un controllore in grado di confrontare la misura con un segnale di riferimento (proporzionale all'input del dispositivo) per pilotare il motore, ovvero il processo, in modo da ottenere l'uscita desiderata.

I modelli più avanzati possono prevedere molteplici rami di retroazione, ognuno riferito ad una grandezza d'interesse (corrente assorbita dal motore, velocità o posizione dell'albero), come si può osservare in Figura 1.1. Normalmente ogni grandezza viene misurata attraverso uno o più sensori idonei.[4]

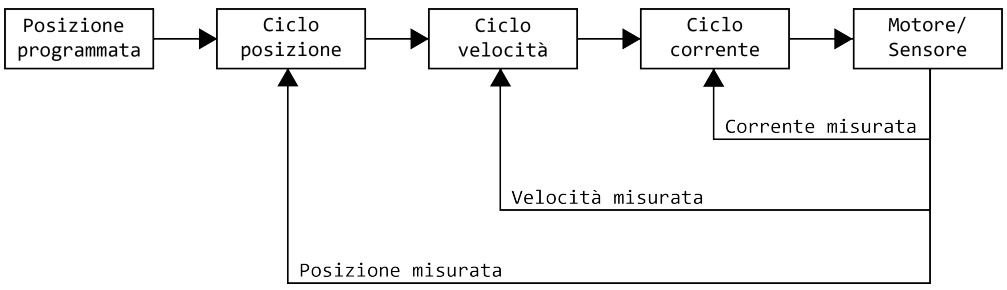

Figura 1.1: Diagramma a blocchi del sistema di controllo di un servomotore.

Al contrario, i prodotti hobbistici che si trovano in commercio per pochi euro (i cosiddetti servomotori RC, Figura 1.2) sono molto più semplici. In linea di massima sono composti da un comparatore, il cui compito è quello di confrontare il segnale d'ingresso con una forma d'onda che dipende dalla tensione in uscita da un potenziometro solidale all'albero del servomotore, che costituisce il feedback del sistema. La differenza tra il segnale di controllo e quello di feedback è l'errore, il quale pilota un circuito che guida il flusso di corrente che attraversa il motore.

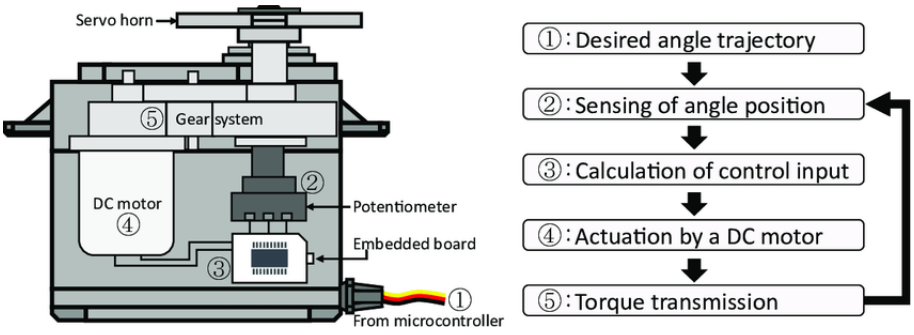

Figura 1.2: Struttura e funzionamento di un servomotore RC. Da Y. Hwang; Y. Minami; M. Ishikawa, *Virtual Torque Sensor for Low-Cost RC Servo Motors Based on Dynamic System Identification Utilizing Parametric Constraints*, «Sensors», 18, 3856 (2018).

Ad oggi lo standard di collegamento utilizzato per larga parte dei servomotori RC consiste in tre linee parallele: l'alimentazione (o positivo, con valori tipici tra 4 e 8 V), la massa (o negativo) e il segnale di pilotaggio. Per quanto riguarda quest'ultimo, la trasmissione della posizione angolare desiderata avviene mediante impulsi modulati in larghezza.[5]

1.2 Modulazione di larghezza d'impulso (PWM)

Nel settore delle telecomunicazioni la modulazione è una tecnica che in sostanza permette di codificare un'informazione mediante la variazione di un parametro di un segnale continuo (detto portante) adatto alla trasmissione all'interno di un canale (via cavo o wireless). A partire dalla natura del segnale che contiene l'informazione (chiamato modulante) si distinguono modulazioni analogiche o numeriche. Tra le modulazioni numeriche si possono individuare le modulazioni impulsive, le quali prevedono che la codifica avvenga sotto forma di una serie di impulsi. In base al parametro dell'impulso oggetto della modulazione si distinguono:

- modulazione d'ampiezza d'impulso, o PAM
- modulazione di larghezza d'impulso, o PWM
- modulazione codificata d'impulso, o PCM
- modulazione di fase d'impulso, o PPM.

Alcune caratteristiche chiave della PWM fanno sì che essa sia una delle modulazioni maggiormente utilizzate nel mondo dell'elettronica.
Innanzitutto, trattandosi di una modulazione numerica, i livelli logici di un segnale PWM sono soltanto due, cioè quello basso (0) e quello alto (1). La modulazione avviene mediante la modifica del parametro noto come *duty cicle*, definito come il rapporto (solitamente espresso in percentuale) tra la durata dell'impulso e il periodo del segnale, che in genere è fissato. Naturalmente allo 0% corrisponde la trasmissione continua del valore logico basso, mentre al 100% viene trasmesso quello alto. In Figura 1.3 è possibile osservare degli esempi di segnali con diversi duty cicle.

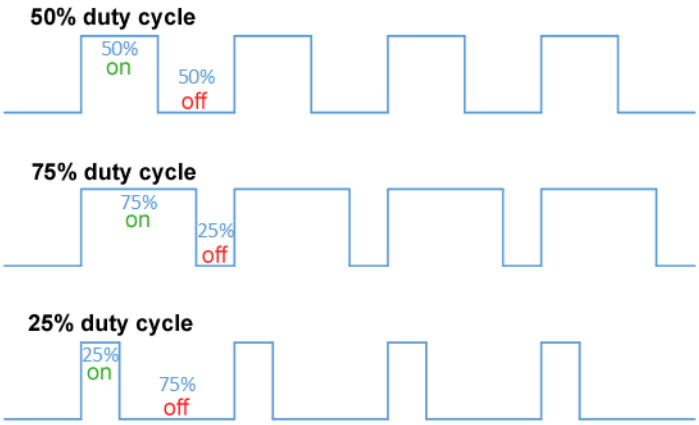

Figura 1.3: Esempi di duty cicle al 50%, al 75% e al 25%. Da Thewrightstuff, `https://commons.wikimedia.org/wiki/File:Duty_Cycle_Examples.png` (2018).

Ne consegue che l'interfaccia tra la circuiteria digitale che contiene l'informazione e il canale di trasmissione non necessita di un convertitore digitale-analogico, visto che il segnale può essere generato direttamente in digitale tramite un'opportuna temporizzazione degli impulsi. Per fare questo basta servirsi di un contatore con una risoluzione sufficientemente elevata.
Ciò costituisce un vantaggio in termini di robustezza al rumore: in primo luogo non c'è introduzione di rumore di quantizzazione, il quale in genere

è dovuto alla conversione tra analogico e digitale; inoltre la trasmissione è numerica, quindi il rumore sulla linea può deteriorare l'informazione soltanto se è talmente elevato da modificare il valore logico del segnale.
Ulteriori benefici riguardano il trasferimento di potenza ad un carico. Si immagini di avere un semplice circuito costituito da un generatore di tensione, un interruttore e una lampadina (Figura 1.4). Chiudendo l'interruttore la potenza del generatore si trasferisce alla lampadina, che si accende. Se invece l'interruttore viene chiuso e aperto alternativamente ogni 5 s, come è ovvio la lampadina si accende e si spegne in sequenza. Ciò corrisponde ad azionare il carico tramite un segnale PWM con un duty cicle del 50% e con un periodo pari a 10 s. Se invece la frequenza con cui si adopera l'interruttore sale, in modo che il periodo del segnale scenda ben al di sotto della costante di tempo dell'utilizzatore, la luce emessa dalla lampadina diviene continua, ma la sua luminosità si abbassa. Il comportamento di un carico di questo tipo è conforme a quello di un filtro passa-basso o di un integratore reale, e la medesima situazione ricorre anche con gran parte dei motori elettrici, compresi quelli all'interno dei servomotori RC.

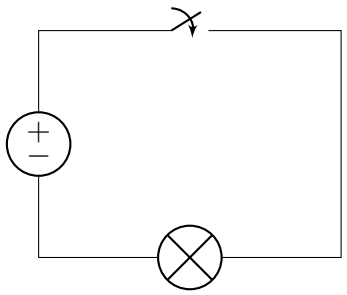

Figura 1.4: Circuito per l'accensione di una lampadina.

Infine, sempre per quanto riguarda il trasferimento di potenza, è doveroso sottolineare che l'impiego della PWM porta un vantaggio anche in termini di efficienza. Solitamente un segnale di controllo (che ad esempio può essere generato da un microcontrollore) non è adatto a fornire la potenza di cui l'utilizzatore ha bisogno; dunque lo si utilizza per pilotare un transistor, che a sua volta regola la potenza proveniente da una linea di alimentazione adeguata. In un semiconduttore la potenza dissipata è pari al prodotto tra la corrente che lo attraversa e la tensione ai suoi capi. Se il transistor fosse comandato tramite un segnale analogico, in ogni istante si avrebbero tensione e corrente generalmente non nulle, e di conseguenza una considerevole dissipazione. Diversamente, facendo uso di segnali digitali i casi sono due: il transistor può essere completamente conduttivo, quindi manifestare una differenza di potenziale nulla, oppure può essere totalmente interdetto,

bloccando il passaggio della corrente. Ne consegue che, almeno dal punto di vista ideale, viene dissipata potenza soltanto negli intervalli di transizione tra stati logici.[6]

1.3 Scelta dei componenti

La scelta dei componenti con cui lavorare per realizzare il progetto parte dai servomotori, dei quali è opportuno valutare le seguenti caratteristiche:

- Costo: dato che il progetto non richiede prestazioni particolarmente elevate, non c'è bisogno di investire una grande quantità di denaro.
- Dimensioni e peso: è meglio che i servomotori siano piccoli e leggeri, in modo da non ostacolare i movimenti del braccio e allo stesso tempo favorirne l'assemblaggio e il trasporto.
- Coppia di stallo: consiste nel carico di coppia tale che il servomotore vada in stallo, cioè non riesca più a ruotare. Generalmente si desidera un valore alto, in modo che il braccio sia in grado di muoversi e di interagire con degli oggetti.
- Angolo massimo di rotazione: come già accennato, tipicamente questi apparecchi possono ruotare (approssimativamente) tra 0 e 180°. Si tratta di limiti accettabili per il caso in questione.
- Velocità di rotazione: è auspicabile che le articolazioni del braccio ruotino con una velocità sufficientemente elevata ma non eccessiva, in modo da non compromettere la stabilità dell'intera struttura.
- Temperatura d'esercizio: il dispositivo deve poter funzionare a temperatura ambiente, ovvero in un intervallo sufficiente ampio attorno al valore nominale di 20 °C.

Il modello scelto è il Tower-Pro MG996R (Figura 1.5), il quale si presta all'obiettivo non solo perché soddisfa le caratteristiche menzionate, ma anche perché è facilmente reperibile ed esiste un'ampia documentazione a riguardo. In Tabella 1.1 ne sono riportate alcune specifiche, mentre sul manuale del Terasic Servo Motor Kit[7] (del quale si discute nel Paragrafo 2.3) si legge che deve essere pilotato tramite segnali PWM a 5 V con periodo di 20 ms (che corrisponde a una frequenza di 50 Hz). Se l'impulso dura 0.5 ms (cioè con un duty cicle del 2.5%) l'albero ruota a 0°, con una durata di 2.5 ms (duty cicle del 12.5%) si raggiungono i 180°.

Figura 1.5: Servomotore Tower-Pro MG996R

Peso	55 g
Dimensioni	40.7×19.7×42.9 mm
Tensione operativa	4.8 - 6.6 V
Coppia di stallo	9.4 kg cm^{-1} (4.8 V); 11 kg cm^{-1} (6 V)
Velocità di rotazione	0.19 s/60° (4.8 V); 0.15 s/60° (6 V)
Ampiezza della banda morta	1 µs
Temperatura d'esercizio	0 - 55 °C
Tipo di ingranaggi	Ingranaggi in metallo
Connettore	JR
Lunghezza del cavo	32 cm
Corrente a riposo	10 mA
Corrente senza carico	170 mA
Corrente di stallo	1400 mA

Tabella 1.1: Specifiche del servomotore Tower-Pro MG996R. Da `https://torqpro.com/product/mg996r/`.

Naturalmente per la costruzione di un prototipo di braccio meccanico non bastano i soli servomotori, ma c'è bisogno anche di una struttura rigida adeguata e di un organo di presa. Il vantaggio di aver scelto un modello così comune sta anche nella semplicità con la quale è possibile procurarsi dei supporti compatibili e la relativa bulloneria, come quelli in Figura 1.6.
Infine, la scelta dei componenti circuitali fa parte del prossimo capitolo, mentre per quanto riguarda la selezione dei dispositivi di alimentazione si rimanda al Paragrafo 5.1.

Figura 1.6: Staffa e *gripper* compatibili con il servomotore Tower-Pro MG996R.

Capitolo 2

Piattaforma hardware

In accordo con quanto messo in evidenza nel capitolo precedente, è necessario che il circuito elettronico per il controllo del braccio meccanico governi ognuno dei servomotori che lo compone indipendentemente, fornendo sia la potenza necessaria, sia il segnale PWM che codifica le informazioni di pilotaggio. Inoltre, quest'ultimo deve necessariamente dipendere dall'input di un utente, che quindi deve essere correttamente acquisito e interpretato.
In sostanza bisogna ricorrere a un supporto elettronico digitale in grado di elaborare molteplici segnali in contemporanea e con un buon livello di prestazioni (intese soprattutto come qualità di gestione degli ingressi, nonché come efficacia e accuratezza nella manipolazione delle uscite). In più le linee di output devono essere facilmente interfacciabili con il modello di servomotore scelto, il Tower-Pro MG996R. In aggiunta, per le fasi di progettazione e di diagnostica conviene che il supporto sia caratterizzato da un buon livello di flessibilità, in modo che all'evenienza le funzionalità del circuito possano essere modificate agevolmente.

2.1 FPGA

Le opzioni verosimilmente adatte per realizzare un sistema di controllo di questo tipo sono principalmente quattro:

- ASIC (*application specific integrated circuit*): si tratta di circuiti integrati fabbricati per un'applicazione specifica, e che di conseguenza possono raggiungere prestazioni elevatissime. Il costo di sviluppo è particolarmente elevato, dunque sono indicati per grandi volumi di produzione.

- DSP (*digital signal processor*): è una tipologia di processori ottimizzati per eseguire operazioni di elaborazione numerica di segnali (ad esempio codifica/decodifica audio e video), e che implementano permanentemente funzioni e algoritmi utili allo scopo.

- FPGA (*field programmable gate array*): sono dispositivi logici programmabili costituiti da blocchi logici e interconnessioni che possono essere modificati direttamente dall'utente finale, limitando costi e tempi di produzione senza rinunciare a un buon livello di performance, con lo svantaggio di un'elevata ridondanza di transistor.

- MCU (*microcontroller unit*): sostanzialmente rappresentano l'evoluzione dei microprocessori, rispetto ai quali oltre alla funzione di elaborazione delle istruzioni integrano ulteriori periferiche (come memorie, interrupt e timer). Sono particolarmente adatti alle applicazioni di controllo che prevedono una parte di elaborazione dati.

Nel caso in esame è immediato rendersi conto che le prime due tecnologie non sono particolarmente convenienti: l'elaborazione numerica da implementare non giustifica il ricorso a un DSP, e realizzare un ASIC comporterebbe costi e tempi incompatibili con quelli del progetto.[8]
Al contrario, FPGA e MCU sono entrambe alternative più che valide per il controllo di servomotori. Un microcontrollore è indubbiamente più economico e più semplice sia da programmare che da testare, a discapito del fatto che la sua struttura hardware è prefissata. Il suo funzionamento è regolato da un processore, dunque è orientato a lavorare in modo sequenziale. Viceversa, adoperando una scheda FPGA il sistema di controllo può essere modellato nella maniera desiderata, a patto di conoscere ogni aspetto della sua configurazione circuitale, a partire dalle strutture logiche che lo compongono. Inoltre, per via della sua struttura interna, un FPGA vanta buone capacità di lavoro in parallelo.[9]
Come già anticipato nell'introduzione, la valutazione di queste caratteristiche ha portato alla scelta di realizzare la circuiteria di controllo del braccio meccanico mediante una scheda di sviluppo FPGA.

Volendone approfondire l'architettura, un FPGA è un device a semiconduttori essenzialmente formato da una matrice di blocchi logici programmabili (o CLB) circondati da una rete di interconnessioni, anch'esse configurabili. Ai margini della matrice vengono posizionati dei blocchi di ingresso/uscita (IOB) per l'interfacciamento con l'esterno (Figura 2.1). Inoltre, nelle famiglie più avanzate la strutture matriciale può essere arricchita tramite funzionalità aggiuntive fisse, come blocchi DSP, moltiplicatori, anelli agganciati in fase

(PLL), memorie e processori integrati, le quali conducono all'incremento le prestazioni e le capacità complessive della scheda.
Come è possibile osservare negli esempi in Figura 2.2 e 2.3, ogni CLB è composto da alcune celle logiche, le quali tipicamente sono costituite da una tabella di verità a quattro ingressi (parte combinatoria), un flip-flop D (elemento di memoria), un full-adder e svariati multiplexer.

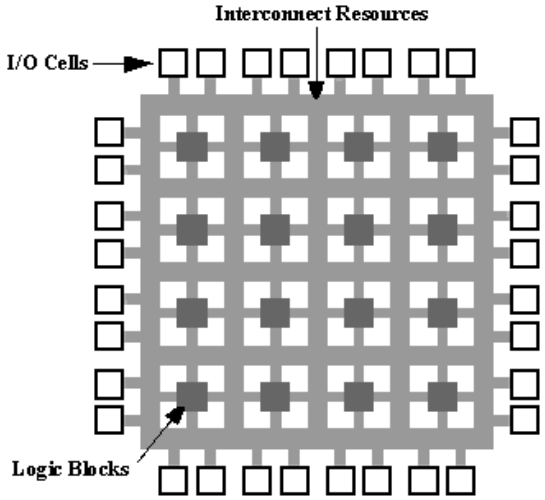

Figura 2.1: Architettura FPGA. Da W.T.Freeman, `https://commons.wikimedia.org/wiki/File:Fpga1a.gif` (2009).

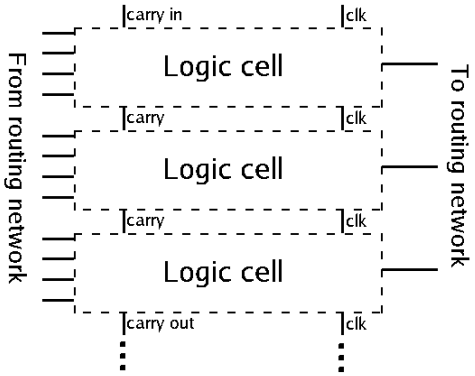

Figura 2.2: Esempio di connessione delle celle logiche in un CLB. Da Petter.kallstrom, `https://commons.wikimedia.org/wiki/File:FPGA_cells-in-block_example.png` (2010).

Facendo uso di uno specifico linguaggio di programmazione detto linguaggio di descrizione hardware (HDL), i CLB possono essere configurati per eseguire

operazioni logiche combinatorie e sequenziali, connettendosi tra di loro e con i pin di ingresso e uscita tramite le piste programmabili. In particolare, il codice HDL viene verificato, simulato e riscritto sotto forma di *netlist* (ovvero un file che contiene la descrizione di ogni componente logico elementare e di tutti i collegamenti all'interno del circuito) da parte di un software, tipicamente fornito dal produttore del FPGA. Per concludere, la configurazione vera e propria genericamente avviene facendo utilizzo di memoria flash (non volatile, che consente la conservazione dei dati anche senza l'alimentazione) oppure di RAM (viceversa volatile), in modo che la struttura matriciale possa essere modificata un elevato numero di volte. L'alternativa è l'impiego di fusibili e antifusibili, chiaramente un meccanismo che non consente la riprogrammazione.[10][11][12]

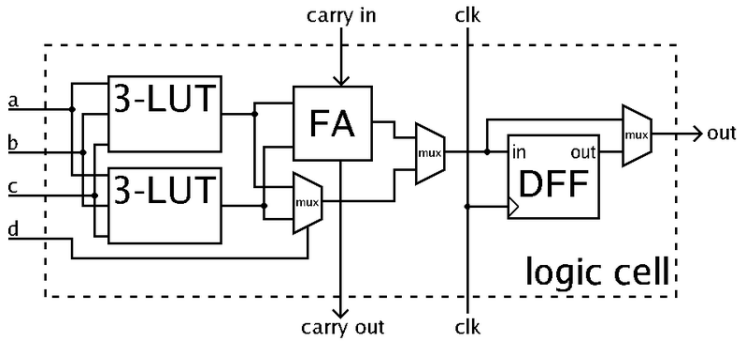

Figura 2.3: Esempio di struttura interna di una cella logica di un FPGA.

Al momento le quattro aziende leader mondiali nella produzione di schede FPGA sono Xilinx, Altera (acquisita da Intel nel 2015), Actel (parte di Microsemi dal 2010) e Lattice Semiconductor.

2.2 Altera DE2-115

Il presente paragrafo riporta alcune caratteristiche della piattaforma FPGA scelta per l'implementazione del circuito di controllo, ovvero la scheda di sviluppo educativa Altera DE2-115, distribuita da Terasic.
Come è possibile osservare in Figura 2.2, il prodotto in questione racchiude una grande quantità di periferiche e funzionalità:

- Device FPGA: il cuore della scheda è il dispositivo FPGA EP4CE115F29 della famiglia Cyclone, il quale è composto da 114 480 elementi logici, 3888 kbit di memoria integrata, 4 PLL, 528 IOB e 266 moltiplicatori.

- Configurazione: la matrice logica può essere programmata secondo due modalità: JTAG e AS. JTAG sta per *joint test action group*, e prevede che il flusso di dati sia caricato direttamente nel FPGA in modo volatile. AS è l'acronimo di *active serial*, e fa utilizzo della periferica di configurazione seriale EPCS64 per conservare i dati anche con l'interruzione dell'alimentazione. La modalità va selezionata azionando l'interruttore posto vicino ai display a sette segmenti.

- Memoria: la memoria è costituita da 128 MB di SDRAM, 2 MB di SRAM, 8 MB di memoria flash e 32 kbit di EEPROM.

- Clock: il segnale di clock, fondamentale per il funzionamento dei circuiti sequenziali, viene generato da tre oscillatori alla frequenza di 50 MHz. Il clock può essere trasmesso o prelevato attraverso connettori SMA.

- Alimentazione: la tensione d'alimentazione deve essere continua a 12 V, e va introdotta collegando un trasformatore adatto al connettore specifico. Una serie di regolatori *step-down* LM3150MH hanno il compito di adeguare il livello di tensione a quelli richiesti dai diversi componenti.

- Interruttori e indicatori: sono installate alcune periferiche che permettono all'operatore di interagire direttamente con la logica FPGA. Si tratta di 18 interruttori a scorrimento, 4 interruttori a pulsanti, 18 LED rossi, 9 LED verdi, 8 display a sette segmenti e un modulo LCD 16×2.

- Connettori: per la connessione verso l'esterno si possono sfruttare porte ethernet, porte USB (necessarie per collegare la scheda a un PC per la configurazione), diversi jack audio, connettori PS/2 per mouse e tastiera, eccetera. Per gran parte di questi il livello di tensione corrispondente a 1 logico può essere regolato spostando dei jumper.

- Espansione: la rosa di possibilità offerte dal FPGA può essere ulteriormente ampliata ricorrendo a moduli esterni interfacciabili (denominati *daughter card* nei manuali) attraverso una porta di espansione GPIO (*general purpose input/output*) a 40 pin. Per esempio, è possibile collegare moduli contenenti fotocamere, schermi tattili, porte DVI e così via.

- Altro: sono integrate funzionalità aggiuntive come encoder/decoder di segnali audio e televisivi, un lettore SD e un ricevitore a infrarossi.

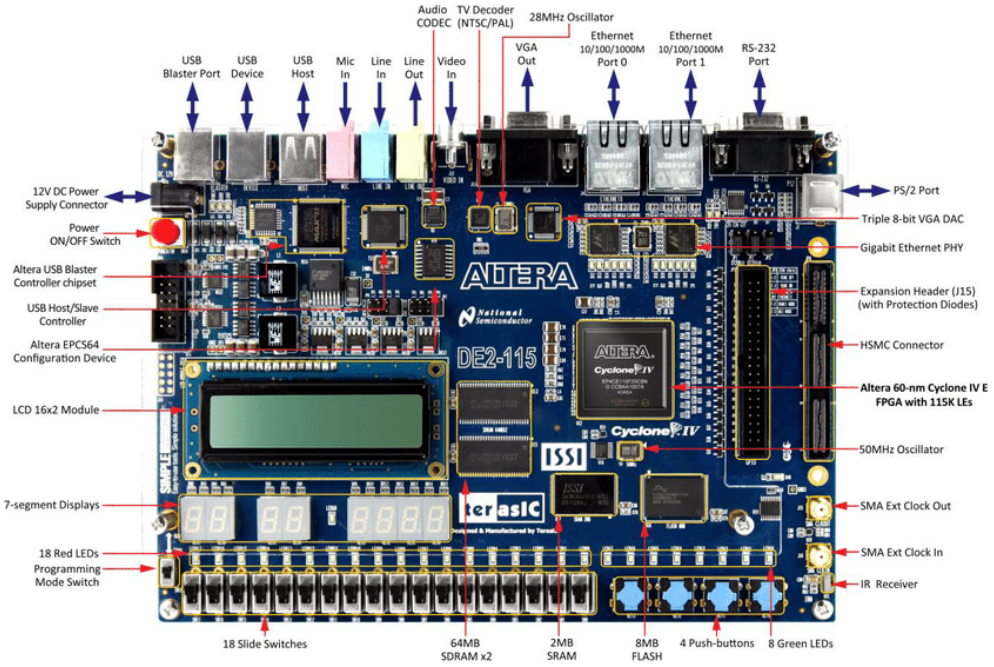

Figura 2.4: Layout della scheda di sviluppo Altera DE2-115. Dal manuale della scheda[13].

Per la programmazione della scheda DE2-115, il manuale[13] consiglia di utilizzare il software Intel Quartus Prime, il quale, come riportato nel Paragrafo 3.1, permette di curare ogni aspetto della progettazione, dalla compilazione del codice HDL all'analisi di temporizzazione del circuito, fino alla scrittura della netlist sul dispositivo tramite USB.

2.3 Terasic Servo Motor Kit

Per consentire alla scheda Altera DE2-115 di interfacciarsi con i servomotori che compongono le articolazioni del braccio meccanico occorre adoperare un modulo d'espansione opportuno. A riguardo il catalogo di Terasic offre il Terasic Servo Motor Kit, un pacchetto che include la daughter card in questione, un servomotore Tower-Pro MG996R e un cavo di collegamento a 40 pin.
Naturalmente il fulcro di questo paragrafo è proprio la daughter card, la quale svolge le seguenti funzioni:

- alimenta i servomotori a partire da un input in corrente continua compreso tra 6.2 e 12 V

- converte i segnali PWM provenienti dalla scheda FPGA da 3.3 a 5 V e provvede a trasferirli ai servomotori
- protegge ognuna delle linee di collegamento con i servomotori tramite fusibili
- monitora la corrente circolante totale e il livello di tensione di un'eventuale batteria. Se rileva valori fuori dai limiti, disconnette l'alimentazione
- può gestire un massimo di 24 servomotori simultaneamente.

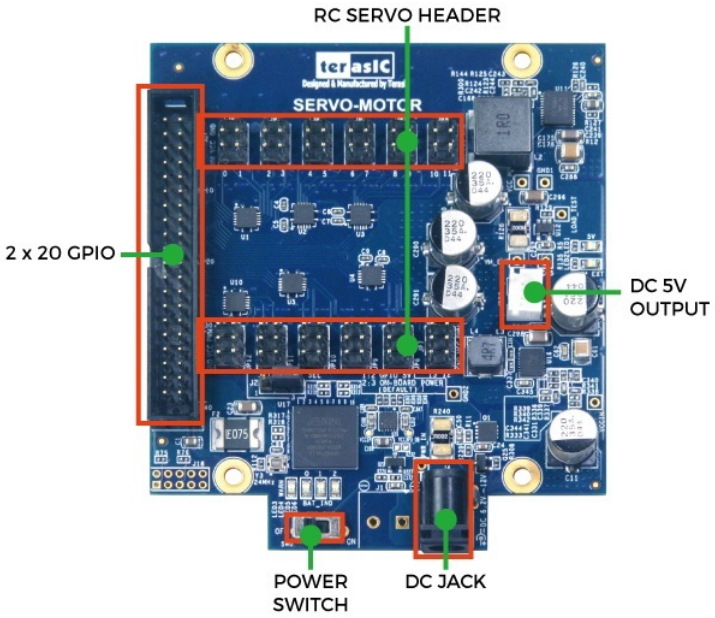

Figura 2.5: Layout della daughter card Terasic SMK. Dal manuale del kit[7].

Invece, per quanto riguarda la sua struttura, si faccia riferimento alla Figura 2.5. È costituita innanzitutto dalla medesima interfaccia GPIO a 40 pin presente anche nella scheda madre, alla quale va collegata tramite il cavo incluso nel kit. Poi è possibile osservare il jack DC e il relativo interruttore per la connessione alla linea d'alimentazione; parte della potenza può anche essere ricondotta in uscita, tramite il connettore bianco a 5 V, nel caso ci fosse bisogno di alimentare un ulteriore apparato. Infine, ci sono i 24 header a 3 pin destinati al collegamento dei servomotori, compatibili con lo standard JR adottato da questi ultimi.
L'utilizzo del modulo è molto semplice: connesse le varie interfacce, basta

azionare l'interruttore dell'alimentazione e trasmettere i segnali PWM attraverso i pin GPIO della scheda FPGA. La corrispondenza tra gli header per i servomotori e i canali GPIO è resa in Figura 2.6.

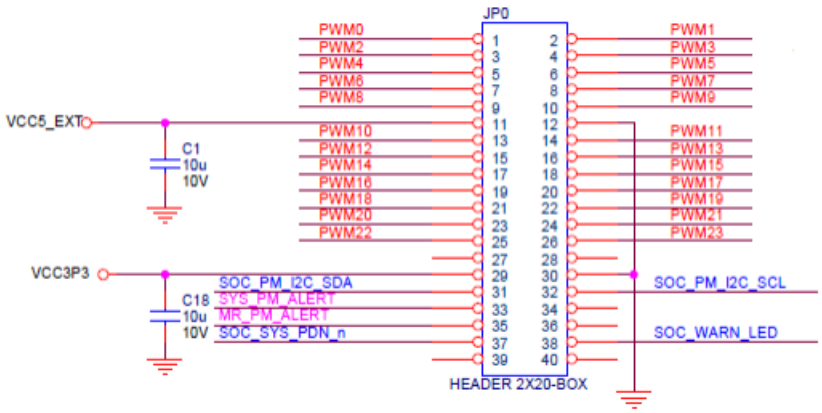

Figura 2.6: Mappatura dei pin GPIO della daughter card Terasic SMK. Dal manuale del kit[7].

Capitolo 3

Ambiente di sviluppo

In accordo con quanto evidenziato durante l'illustrazione del supporto hardware, la configurazione di una scheda FPGA è un processo articolato. Difatti, pur supponendo di aver già progettato su carta il circuito finale, per ottenere un sistema funzionante bisogna necessariamente servirsi di un ambiente di sviluppo, con l'aiuto del quale è essenziale far fronte a una scaletta univocamente definita. Quest'ultima prevede un totale di sei fasi:

1. *Design entry*: l'hardware che compone il circuito va descritto tramite il supporto di una *schematic*, di un grafo orientato (particolarmente utile per le macchine a stati finiti) o di un codice scritto in HDL (come Verilog o VHDL).

2. *Translate and mapping*: l'ambiente di sviluppo traduce il progetto del circuito, scritto in un linguaggio di alto livello, sotto forma di un'unica netlist di basso livello, la quale viene poi mappata rispetto alle risorse effettivamente disponibili sulla scheda FPGA. Questa fase è detta anche sintesi.

3. *Functional simulation*: si tratta di una prima simulazione del circuito, che avviene prima della scrittura sul supporto fisico con lo scopo di mettere in evidenza eventuali errori di progettazione.

4. *Place and route*: a questo punto il software scompone ulteriormente gli elementi circuitali nei blocchi logici fondamentali che li costituiscono (ad esempio un registro viene diviso in un insieme di flip-flop). In seguito decide in quali punti della matrice di CLB posizionare ogni singolo blocco e come realizzare le interconnessioni, in modo da ottimizzare le prestazioni e l'impiego di risorse.

5. *Timing analysis*: prima di configurare il circuito su FPGA è necessario verificare la sua funzionalità in termini di vincoli di temporizzazione, i quali nascono a partire da alcune specifiche imposte dall'utente (ad esempio è possibile valutare la corretta propagazione di un segnale di clock a una determinata frequenza).

6. *Programming*: l'iter si conclude con l'effettiva programmazione della scheda.

3.1 Intel Quartus Prime

Come anticipato nel Paragrafo 2.2, per lavorare con la scheda Altera DE2-115 conviene servirsi del software Quartus Prime di Intel (Figura 3.1). Essenzialmente si tratta di un'applicazione per la progettazione e la configurazione di dispositivi logici programmabili, primi tra tutti i sistemi FPGA.
Quartus Prime viene distribuito in tre edizioni: pro, standard e lite. La versione lite è l'unica gratuita, ma convenientemente è compatibile con le schede che implementano FPGA della famiglia Cyclone IV, dunque anche con la DE2-115. Lo svantaggio è che bisogna rinunciare ad alcune funzioni, comunque non strettamente necessarie ai fini del progetto in questione.[14]

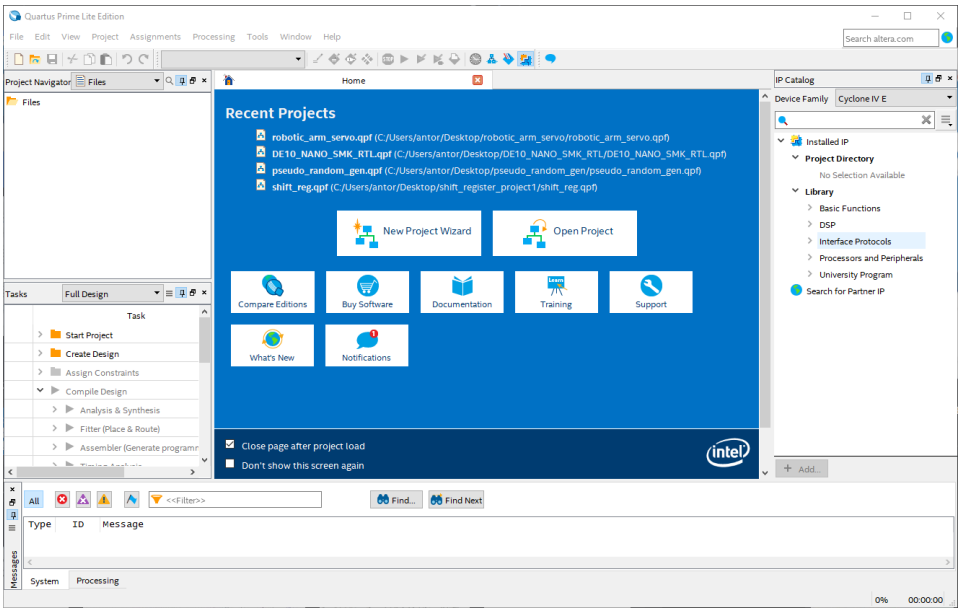

Figura 3.1: Schermata iniziale di Intel Quartus Prime.

Di seguito viene illustrato un tipico *workflow* riguardante la realizzazione di un circuito su FPGA tramite Quartus:

1. Il primo step consiste nella creazione di un nuovo progetto tramite l'apposito *wizard*. L'utente è tenuto a inserire alcune informazioni, come il modello di FPGA selezionato.

2. A seguire c'è la fase di design entry, che prevede l'importazione o la creazione di uno o più file di progetto. Quartus permette di leggere o generare progetti in diversi formati: sono supportati svariati tipi di HDL (Verilog, SystemVerilog, VHDL, AHDL), il linguaggio TCL, diagrammi a blocchi, diagrammi degli stati di un automa, file Qsys. Per semplificare il lavoro di progettazione viene offerta la possibilità di utilizzare delle librerie.

3. Fatto ciò, occorre definire dei vincoli, i quali consistono sostanzialmente in due aspetti: l'assegnazione dei pin della scheda agli ingressi e alle uscite del progetto, e l'impostazione delle specifiche di compilazione (come la scelta del tool per la simulazione del design).

4. Segue la compilazione del progetto. Il software dapprima analizza il design per mettere in evidenza avvisi o errori, i quali devono essere gestiti in modo adeguato dall'utente. Successivamente vengono le fasi di sintesi, place and route e timing analysis. Se tutto va a buon fine, la compilazione si conclude con la scrittura della EDA netlist destinata al trasferimento sulla scheda FPGA.

5. Prima di applicare il design sul dispositivo fisico può risultare opportuno simularlo, in modo da capire nell'immediato se il circuito svolge correttamente le funzioni che si desiderava implementare, o se si nasconde qualche errore nella sua progettazione. A tal proposito è necessario scrivere un ulteriore file, detto *test bench*, contenente informazioni sugli stimoli ai quali deve essere sottoposto il circuito (banalmente, se si volesse testare una porta AND a due ingressi, si potrebbe pensare di introdurvi in sequenza le quattro possibili combinazioni di input). La simulazione vera e propria avviene su due livelli: RTL, cioè *register transfer level*, con un grado di astrazione limitato ai registri, e *gate-level*, che al contrario si spinge fino al test delle singole porte logiche.

6. Per concludere, non resta che programmare la scheda FPGA (fase di programming). Lo strumento di programmazione permette di scegliere tra le modalità JTAG e AS, già presentate nel Paragrafo 2.2, e di selezionare il canale di comunicazione. Per configurare il dispositivo tramite USB, occorre installare e scegliere il driver USB-Blaster.

Dato che l'architettura FPGA mette a disposizione del progettista un oceano di possibilità, è scontato che il flusso di lavoro appena delineato è semplicemente uno scheletro, il quale può essere arricchito da ulteriori passaggi non ancora menzionati qualora ve ne sia l'evenienza. Tra gli altri strumenti messi a disposizione da Quartus Prime, a seguire si evidenziano i principali:

- *RTL Viewer*, che permette di mostrare a schermo una rappresentazione della EDA netlist sotto forma di una rete di blocchi logici e registri
- *Platform Designer*, tool che consente di integrare in maniera immediata proprietà intellettuali (come il processore Nios II) generando automaticamente la logica di interconnessione
- *Power Analyzer*, il quale aiuta a determinare una stima dei consumi del sistema finale con lo scopo di rispettare un'eventuale *power budget*
- *Signal Tap Logic Analyzer*, in grado di registrare e mostrare in tempo reale all'utente il comportamento logico del circuito a posteriori rispetto alla configurazione su FPGA, senza dover modificare il progetto per includere input o output appositi.

3.2 Verilog HDL

Al giorno d'oggi, col grado di complessità che ha raggiunto l'elettronica non conviene più che la descrizione dei circuiti avvenga semplicemente tramite schematic. Al contrario gli strumenti più potenti ed efficienti a disposizione del progettista sono i linguaggi di descrizione hardware, o HDL. Questi aiutano a semplificare e ottimizzare il processo e, integrandosi con altri strumenti, permettono anche di verificare la funzionalità del circuito in modo automatico.
Il mercato degli HDL è dominato da due linguaggi rivali, ovvero Verilog e VHDL. Verilog è caratterizzato una sintassi concisa e familiare agli utenti alle prime armi, perché simile a quella del linguaggio C; si tratta di un HDL debolmente tipato e comune in ambito aziendale. Viceversa, VHDL ha una sintassi prolissa, affine a quella del linguaggio ADA; fa uso rigoroso dei tipi e in genere viene preferito in ambito accademico.
Per il presente progetto è stato scelto Verilog. Tuttavia, prima di procedere con la descrizione del codice relativo al circuito di controllo (Capitolo 4), per poterlo comprendere al meglio occorre approfondire alcuni suoi aspetti, compresa la sintassi.

Innanzitutto, i principali *data type* di Verilog sono due: *wire* e *reg*. Una variabile wire corrisponde a una connessione fisica nel circuito, dunque può essere assegnata (con l'istruzione *assign*) oppure letta, ma non può memorizzare un dato. Al contrario reg costituisce un'unità di memoria, la quale mantiene l'informazione finché non viene sovrascritta; ne deriva che le variabili reg sono particolarmente utili per l'implementazione di circuiti sequenziali.
Per quanto riguarda le costanti numeriche, queste possono essere espresse in notazione binaria, decimale, ottale, esadecimale e così via. Quando la notazione differisce da quella decimale va indicata, assieme al numero di bit occupati dal dato numerico (ad esempio il numero esadecimale C5 va indicato come 8'hC5).
Inoltre, molto spesso per descrivere alcuni particolari tipi di circuiti con semplicità si ricorre ad un livello d'astrazione più elevato, che consiste nell'impiego dei processi *initial* e *always*. Entrambi sono blocchi di istruzioni che partono nel momento in cui inizia l'esecuzione della simulazione o del circuito. La differenza è che initial si ripete soltanto una volta, quindi è utile per inizializzare variabili oppure nei test bench, mentre always si ripete di continuo. Quando always viene seguito da una chiocciola (@) e una condizione, l'istruzione che contiene si ripete soltanto quando la condizione è verificata. Si osservi in Figura 3.2 l'utilizzo del processo always per la realizzazione di un flip-flop: all'interno di blocchi procedurali simili una *rule of thumb* è quella di utilizzare assegnazioni non bloccanti (corrispondenti al simbolo <=) al posto di quelle bloccanti (=). Un ulteriore carattere molto comune in tali processi è il cancelletto (#), che simboleggia un ritardo temporale.
Il livello di astrazione in questione può consistere nell'utilizzo di costrutti tipici dei linguaggi di programmazione software. Si tratta di espressioni che permettono al progettista di configurare le risposte di un circuito in dipendenza da alcune condizioni: ad esempio la ben nota struttura alternativa *if-else*, oppure il *case* (che in base a come gestisce i *don't care* si divide in case semplice, *casex* e *casez*), il quale definisce un'uscita a partire da molteplici combinazioni di variabili come accade in una tabella di verità.

```
1   module d_flipflop(clk, rst, D, Q);
2
3   input clock, reset, D;
4   output reg Q;
5
6   always @(posedge clock, posedge reset) begin
7       if(reset)
8           Q <= 1'b0;
9       else
10          Q <= D;
11  end
12
13  endmodule
```

Figura 3.2: Esempio di flip-flop D in Verilog.

Per concludere, quando si fa uso di un HDL per la descrizione di un circuito complesso è bene suddividere quest'ultimo in svariati moduli elementari. Si tratta di una pratica che comporta diversi benefici: in primo luogo un problema articolato viene frazionato in una serie di operazioni semplici che il progettista può affrontare più facilmente; inoltre, nel caso dovessero verificarsi errori o malfunzionamenti, risulta molto più rapido individuarne la causa e correggerla; infine, i moduli possono essere riutilizzati nell'ambito dello stesso circuito, portando a risparmiare linee di codice e tempo.
Verilog permette tutto questo: all'interno di un progetto è possibile creare numerosi file sorgente (in formato .v), ognuno relativo ad un modulo del circuito. In seguito occorre generare tutte le istanze necessarie all'interno di un file principale (chiamato *top-level design*), semplicemente dichiarando le connessioni verso i pin d'ingresso e da quelli d'uscita. In Figura 3.3 ne viene riportato un esempio elementare. Quando si creano più copie di uno stesso modulo, può sorgere la necessità di modificarne di volta in volta alcuni aspetti. In tal caso il modulo deve contenere dei *parameter* con valori di default, i quali possono essere modificati nel momento dell'inizializzazione.[15][16]

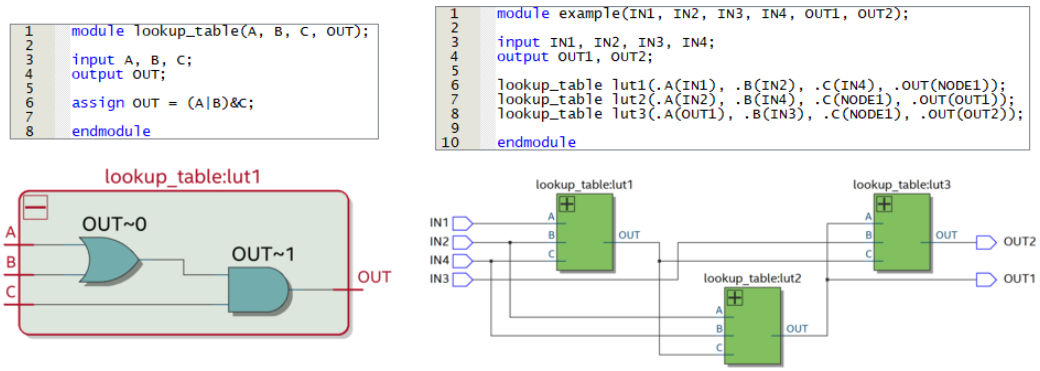

Figura 3.3: Esempio di modularizzazione in Verilog. In alto: codici sorgente. In basso: schemi logici ottenuti tramite RTL Viewer.

Parte seconda: realizzazione del prototipo

Capitolo 4

Implementazione del circuito di controllo

Naturalmente prima di implementare qualsivoglia circuito su FPGA è necessario avere in mente un'idea ben precisa di come sia fatto e del suo funzionamento. Per quanto riguarda il progetto corrente, come già anticipato il compito principale della scheda FPGA è quello di interfacciare l'utente con i servomotori che compongono il braccio meccanico. Dunque, in prima approssimazione il circuito di controllo può essere visto come una scatola nera che prende in ingresso i segnali prodotti dagli input fisici della scheda e in uscita esibisce un segnale PWM destinato alla daughter card del Terasic SMK.
Partendo proprio dall'uscita, occorre implementare un modulo in grado di produrre un segnale PWM adeguato, ovvero con periodo costante e duty cicle variabile. Il duty cicle deve dipendere dallo stato corrente e dalla storia del sistema; dato che è legato alla posizione dei servomotori, bisogna prevedere la possibilità di modificarlo in tempo reale agendo sui comandi della scheda, cioè gli ingressi del circuito. In Figura 4.1 viene proposto un primo diagramma a blocchi del sistema di controllo. Seguono cinque paragrafi incentrati sull'analisi, la realizzazione e la simulazione di ognuno dei moduli che lo compongono, compreso il top-level design.

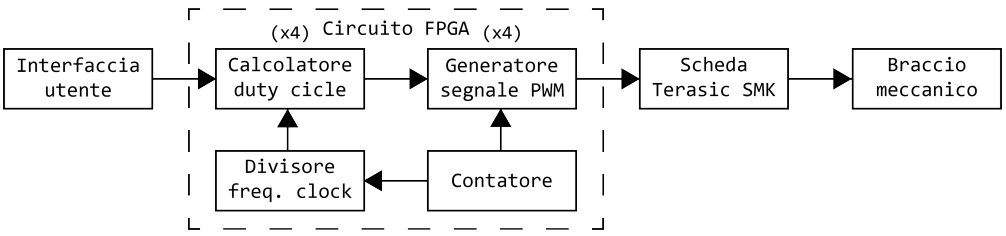

Figura 4.1: Diagramma a blocchi del sistema di controllo del braccio meccanico.

4.1 Contatore

Un contatore è un circuito sequenziale in grado di contare quante volte si verifica un determinato evento. Allo stesso modo può venire utilizzato per effettuare misure di tempo a partire da un clock: conoscendone il periodo, basta moltiplicarlo per il numero di volte che se ne rileva un fronte positivo o negativo nell'intervallo da misurare. In tal modo l'errore che si commette è al massimo pari a un periodo di clock.
Nel circuito in analisi bisogna disporre un contatore che faccia da base sia al divisore di frequenza di clock che ai generatori di segnali PWM. Difatti, come riportato nei paragrafi a seguire, i segnali PWM in uscita, e dunque anche il clock che pilota i calcolatori di duty cicle, devono avere periodo pari a 20 ms. Conviene che il contatore sia unico, questo fondamentalmente per due motivi: innanzitutto così facendo si ottimizzano le risorse, dato che si tratta di un modulo alquanto dispendioso in termini di flip-flop e porte logiche; in più, gli altri moduli del sistema possono utilizzarlo come riferimento temporale in modo da lavorare in sincronia.
Considerando che da manuale[13] la scheda Altera DE2-115 può generare un clock a 50 MHz (denominato CLOCK_50), il contatore in questione deve essere in base 1 000 000. Per la realizzazione in Verilog, un buon punto di partenza è il *decade counter*: il conteggio deve essere incrementato a ogni ciclo di clock finché non si raggiunge la base, punto nel quale viene resettato. Inoltre la variabile di conteggio va riportata in uscita tramite un bus a 20 bit, abbastanza largo da codificare correttamente tutti i numeri fino a 1 000 000. Segue, in Figura 4.2, il relativo codice sorgente. Si osservi che, oltre al clock e alla variabile di conteggio, in ingresso si preleva anche un segnale di reset: quando si crea un circuito sequenziale è buona norma prevedere la possibilità di resettarlo manualmente.

```
1   module counter(clock, reset, count);
2
3   input clock, reset;
4   output reg [19:0] count = 0;
5
6   always @(posedge clock) begin
7       if(reset || count >= 1_000_000)
8           count <= 1;
9       else
10          count <= count+1;
11  end
12
13  endmodule
```

Figura 4.2: Codice sorgente Verilog del contatore.

Una volta compilato il file sorgente può essere utile dare uno sguardo alla rete logica generata dallo strumento RTL Viewer, sia per verificare che non

ci siano errori di progettazione, sia per rendersi conto di come avviene effettivamente l'implementazione all'interno del FPGA. Se ne riporta la schermata in Figura 4.3.

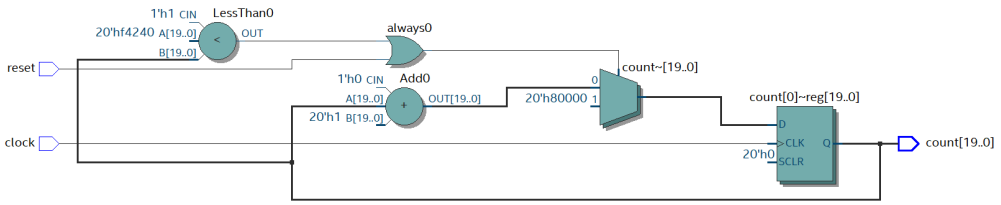

Figura 4.3: Vista RTL del contatore.

Quando si progettano circuiti su FPGA, ennesima buona abitudine è quella di simulare ogni singolo modulo separatamente. Ciò permette di evidenziare e correggere fin da subito eventuali malfunzionamenti, in modo da risparmiare il tempo e gli sforzi che altrimenti sarebbero da impiegare per la loro ricerca. In accordo con quanto affermato nel Paragrafo 3.1, per simulare un circuito su Quartus Prime occorre realizzare un test bench. La prova più semplice consiste nell'introdurre un segnale di clock fittizio (con periodo di 20 ns) ed osservare il bus d'uscita del contatore per un tempo sufficientemente lungo, appurando che si comporti nella maniera desiderata. In Figura 4.4 viene presentato il codice Verilog del test bench.

```
1   `timescale 1ns/1ps
2
3   module counter_tb();
4
5   reg clock, reset;
6   wire [19:0] count;
7
8   parameter period = 20;
9
10  counter UUT(.clock(clock), .reset(reset), .count(count));
11
12  always begin
13      clock = 1'b0; #(period/2);
14      clock = 1'b1; #(period/2);
15  end
16
17  initial begin
18      reset = 1'b1; #period;
19      reset = 1'b0; #(1_100_000*period); $stop;
20  end
21
22  endmodule
```

Figura 4.4: Codice sorgente Verilog del test bench del contatore.

Impostato il file da simulare nelle impostazioni di Quartus, non resta che far partire la simulazione RTL (con questi tipi di circuiti non ci sono differenze sostanziali con la simulazione gate-level). I risultati sono presentati in Figura 4.5: dato che l'intera simulazione è lunga e ridondante, sono mostrate solo

le sezioni più rilevanti. Si osservi che ogni qual volta la tensione del clock si alza, il contatore incrementa la variabile di conteggio; superato il limite, cioé la base 1 000 000, ritorna a 1. Si può concludere che il modulo è stato progettato in modo corretto.

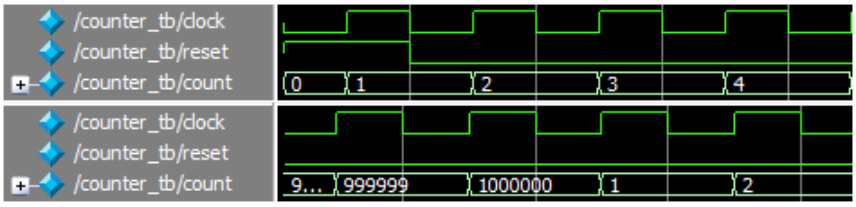

Figura 4.5: Sezioni della simulazione RTL del contatore. In alto: inizio del conteggio. In basso: reset del conteggio dopo aver raggiunto il valore di base.

4.2 Divisore di frequenza di clock

A partire dal contatore è possibile progettare anche il divisore di frequenza di clock, sinteticamente chiamato *downclock*. Un divisore di frequenza di clock è un dispositivo digitale che, prelevando un clock a una certa frequenza in ingresso (ad esempio generato da un oscillatore), è in grado di generare in uscita un secondo segnale di clock a frequenza minore, tipicamente mediante una rete di flip-flop. La necessità di un clock a frequenza ridotta nasce dal calcolatore di duty cicle: in breve, per far sì che il generatore di segnale PWM legga sempre un valore di duty cicle costante e stabile, serve che quest'ultimo si aggiorni soltanto quando non viene letto, tramite un registro che funzioni alla stessa frequenza del segnale PWM ma con uno sfasamento opportuno (tale concetto viene richiamato anche nei paragrafi successivi). Nel codice Verilog (Figura 4.6) la prima cosa da fare è dichiarare gli ingressi e le uscite: in accordo con quanto detto, oltre al clock in ingresso, il segnale di reset e il clock, in uscita va prelevato anche il bus di conteggio del contatore. Il conteggio è in base 1 000 000 e viene incrementato ad intervalli pari al periodo di clock, cioè 20 ns: ciò significa che si resetta con una frequenza di 50 Hz, la quale corrisponde proprio alla frequenza desiderata per il clock in uscita. Non resta che determinare la tensione d'uscita: per garantire lo sfasamento tra i segnali PWM e il clock in questione, è meglio che il secondo semiperiodo sia al livello logico alto. Dunque, l'uscita è alta se il conteggio supera la metà del fondo scala. In Figura 4.7 segue anche la raffigurazione della rete logica generata dal RTL Viewer.

```
 1  module downclock(clockIn, reset, count, clockOut);
 2
 3  input clockIn, reset;
 4  input [19:0] count;
 5  output reg clockOut;
 6
 7  always @(posedge clockIn) begin
 8     if(reset)
 9        clockOut <= 1'b0;
10     else
11        clockOut <= (count > 500_000);
12  end
13
14  endmodule
```

Figura 4.6: Codice sorgente Verilog del divisore di frequenza di clock.

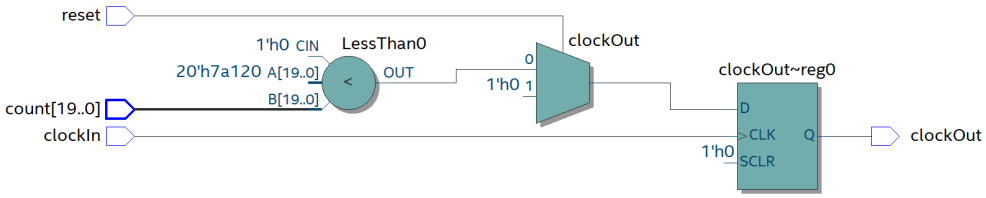

Figura 4.7: Vista RTL del divisore di frequenza di clock.

Non resta che verificare le funzionalità del circuito mediante una simulazione. Il file Verilog che descrive il test bench è riportato in Figura 4.8: innanzitutto, dopo aver dichiarato le variabili utili, bisogna importare sia il divisore di frequenza di clock che il contatore, visto che per costruzione il primo non può funzionare senza il secondo. Infine, non bisogna fare altro che attendere un intervallo sufficiente affinché il clock d'uscita manifesti almeno un periodo.

```
 1  `timescale 1ns/1ps
 2
 3  module downclock_tb();
 4
 5  reg clockIn, reset;
 6  wire clockOut;
 7  wire [19:0] count;
 8
 9  parameter period = 20;
10
11  counter counter(.clock(clockIn), .reset(reset), .count(count));
12  downclock UUT(.clockIn(clockIn), .reset(reset), .count(count), .clockOut(clockOut));
13
14  always begin
15     clockIn = 1'b0; #(period/2);
16     clockIn = 1'b1; #(period/2);
17  end
18
19  initial begin
20     reset = 1'b1; #period;
21     reset = 1'b0; #(3_000_000*period); $stop;
22  end
23
24  endmodule
```

Figura 4.8: Codice sorgente Verilog del test bench del divisore di frequenza di clock.

In Figura 4.9 è possibile osservare il risultato della simulazione RTL a cui è stato sottoposto il modulo in questione. Naturalmente, per poter valutare il segnale di clock d'uscita è necessario fare uso di una risoluzione temporale tale da rendere indistinguibili i valori logici del clock in ingresso e della variabile di conteggio. Tramite l'utilizzo di tre cursori (in giallo) si individuano le durate dei due semiperiodi (e dunque del periodo) del segnale di clock in uscita. I requisiti imposti sono soddisfatti alla perfezione, dunque non ci sono errori nella progettazione del circuito.

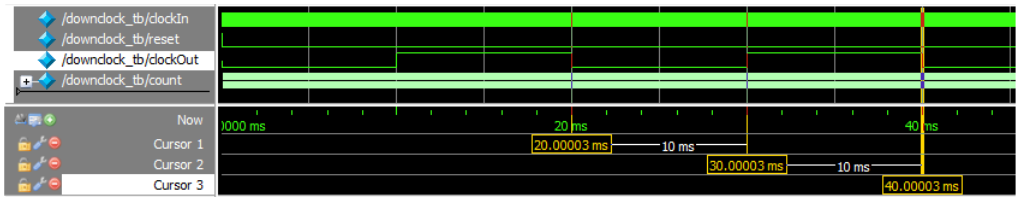

Figura 4.9: Simulazione RTL del divisore di frequenza di clock.

4.3 Calcolatore di duty cicle

Il calcolatore di duty cicle è un blocco circuitale che si occupa fondamentalmente dell'interpretazione degli ingressi introdotti dall'utente. Pertanto, in primo luogo bisogna chiarire in che modo gli input acquisiti influenzano le variabili tramite le quali si generano i segnali di controllo. Dato che l'interfaccia utente in ingresso è costituita principalmente da pulsanti e interruttori, una possibilità è quella prevedere che l'angolo di rotazione dell'albero motore incrementi con la pressione prolungata di un determinato pulsante, facendo sì che lo stato di uno specifico interruttore stabilisca il verso della rotazione. Visto che, come descritto nel Paragrafo 1.3, modificare l'angolo di un servomotore corrisponde a variare il duty cicle del segnale PWM che lo pilota, chiaramente l'uscita del modulo in questione deve essere proprio il duty cicle (o almeno deve dipenderne).
Sebbene realizzare un circuito digitale di questo tipo in modo tradizionale sia estremamente complesso, l'implementazione in Verilog (Figura **??**) risulta sicuramente più agevole. Relativamente a quanto affermato, l'uscita del circuito deve essere un bus abbastanza grande da codificare il duty cicle del segnale PWM. Dato che il periodo di quest'ultimo è fisso a 20 ms, per semplicità il duty cicle può essere espresso soltanto con la durata dell'impulso (nel codice tOn). Per quanto dichiarato in precedenza, tale durata può andare da 0.5 a 2.5 ms: per la codifica del massimo, se espresso in microsecondi, sono necessari 12 bit, che quindi è l'ampiezza del bus in output. Al contrario, in

ingresso si prevedono quattro segnali a bit singolo. Oltre al clock e al reset, fondamentali in qualsiasi circuito sequenziale, ci sono i comandi provenienti dall'interfaccia utente, uno per l'interruttore e uno per il pulsante. Si prevedono anche tre parametri, i quali possono essere alterati relativamente ai vincoli fisici di ognuno dei servomotori: esprimono in microsecondi il passo di incremento, il minimo e il massimo del duty cicle.
Il fulcro del modulo è composto da un blocco procedurale contenente costrutti if e casex. Se l'utente sta azionando il pulsante per la rotazione del servomotore, e se il servomotore può effettivamente ruotare nella direzione indicata, il duty cicle si modifica secondo le modalità descritte. Il problema è che il valore del duty cicle non può essere aggiornato mentre viene letto per la generazione dei segnali PWM. Come già anticipato, una possibile soluzione è quella di ridurre la frequenza del clock alla medesima dei segnali in questione (50 Hz) facendo uso del contatore e del divisore di frequenza di clock. Se il duty cicle viene calcolato al fronte di salita del clock, i periodi dei segnali PWM devono iniziare in corrispondenza del fronte di discesa.

```
1   module dutycicle_calculator(clock, reset, reverse, rotate, tOn);
2
3   input clock, reset, reverse, rotate;
4   output reg [11:0] tOn = 1500;
5
6   parameter step = 20;
7   parameter minTOn = 500;
8   parameter maxTOn = 2500;
9
10  always @(posedge clock) begin
11      if(reset)
12          tOn <= 1500;
13      else if(rotate) begin
14              casex({reverse, tOn <= minTOn, tOn >= maxTOn})
15                  3'b0x0: tOn <= tOn+step;
16                  3'b10x: tOn <= tOn-step;
17                  default: ;
18              endcase
19          end
20  end
21
22  endmodule
```

Figura 4.10: Codice sorgente Verilog del calcolatore di duty cicle.

In Figura 4.11 è possibile apprezzare, tramite la vista RTL del modulo in esame, uno dei vantaggi che derivano dall'impiego di un linguaggio di descrizione hardware come Verilog. Quello che manualmente sarebbe stato un circuito esageratamente complesso da ideare può essere sintetizzato in pochissime linee di codice e senza grandi sforzi. Oltre a questo, la rete logica generata può essere ispezionata nel dettaglio per verificare che il modulo realizzi appieno il suo obiettivo, come già fatto per il contatore. L'aspetto principale da sottolineare è il fatto che l'uscita è gestita da un registro comandato dal clock come desiderato.

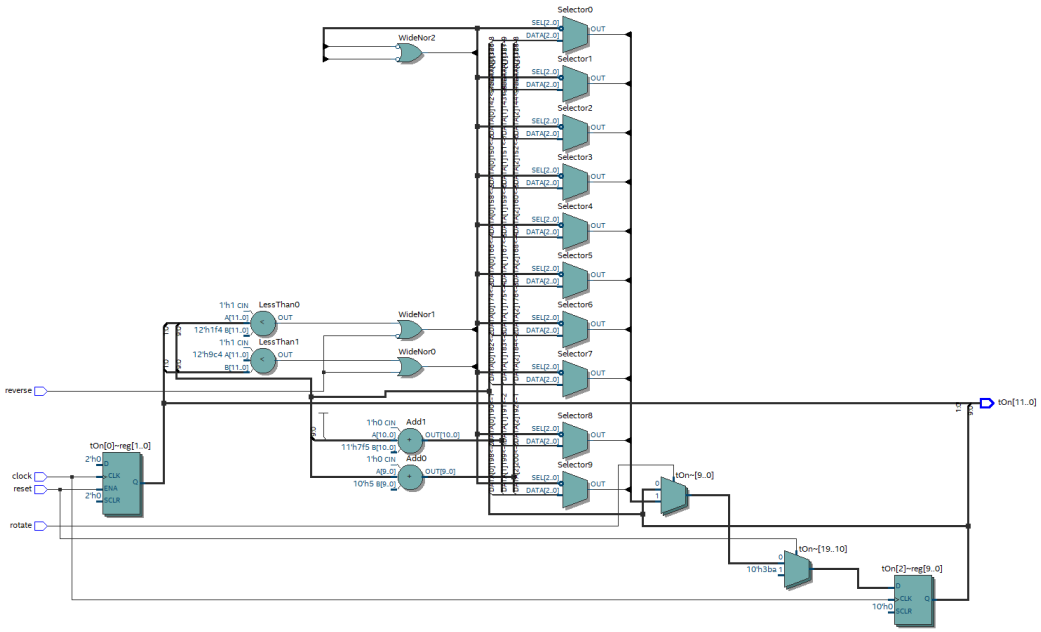

Figura 4.11: Vista RTL del calcolatore di duty cicle.

Infine occorre realizzare il test bench per la simulazione del circuito. Il suo scopo è quello di verificare che il calcolatore di duty cicle riesca a interpretare correttamente tutte le combinazioni di ingressi rispettando i limiti imposti. Considerando un clock con un periodo di 20 ms e lasciando invariati i parametri del modulo, bastano 100 periodi di clock per passare dal minimo al massimo del duty cicle (cioè 2 s, per una velocità di rotazione dei servomotori di circa $90\,^\circ\,\mathrm{s}^{-1}$). Considerando che quest'ultimo viene inizializzato al valore medio di 1500 µs, non resta che sottoporre la *unit under test* a differenti stimoli in modo da poter esaminare nel dettaglio il suo comportamento. Il codice sorgente del test bench viene riportato in Figura 4.12: dapprima si incrementa il duty cicle fino a raggiungere il valore massimo, per poi testare anche il decremento mediante l'azionamento del segnale di inversione.
Il risultato della simulazione RTL viene presentato in Figura 4.13. Se il segnale di rotazione è alto e l'inversione è disattiva, il valore del duty cicle aumenta di 20 µs a ogni fronte di salita del clock, fino a raggiungere il limite imposto di 2.5 ms. Se invece si attiva il comando di inversione, il duty cicle diminuisce col medesimo passo fino al minimo di 0.5 ms. Naturalmente con l'abbassamento della tensione del segnale di rotazione l'incremento viene sospeso il bus d'uscita resta costante. Si conclude che il circuito analizzato risponde correttamente agli stimoli ai quali è sottoposto.

```
1   `timescale 1ms/1us
2
3   module dutycicle_calculator_tb();
4
5   reg clock, reset, reverse, rotate;
6   wire [11:0] tOn;
7
8   parameter period = 20;
9
10  dutycicle_calculator UUT(.clock(clock), .reset(reset), .reverse(reverse),
11                           .rotate(rotate), .tOn(tOn));
12
13  always begin
14      clock = 1'b0; #(period/2);
15      clock = 1'b1; #(period/2);
16  end
17
18  initial begin
19      reset = 1'b1; reverse = 1'b0; rotate = 1'b0; #period;
20      reset = 1'b0; rotate = 1'b1; #(60*period);
21      reverse = 1'b1; #(110*period);
22      reverse = 1'b0; #(50*period);
23      rotate = 1'b0; #(10*period);
24      rotate = 1'b1; #(50*period); $stop;
25  end
26
27  endmodule
```

Figura 4.12: Codice sorgente Verilog del test bench del calcolatore di duty cicle.

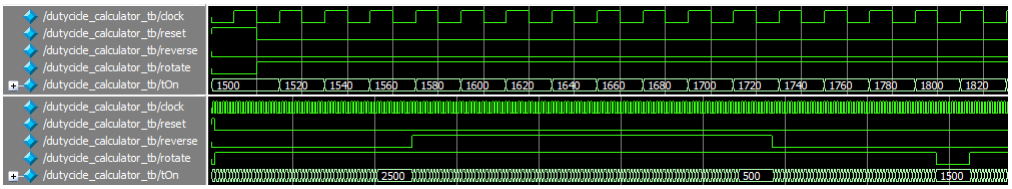

Figura 4.13: Sezioni della simulazione RTL del calcolatore di duty cicle. In alto: incremento del duty cicle. In basso: azionamento del comando per l'inversione della rotazione.

4.4 Generatore di segnale PWM

Il principio di funzionamento di un generatore di segnale PWM è elementare, oltre che molto simile a quanto già visto per il divisore di frequenza di clock. Anche stavolta l'obiettivo è generare un segnale periodico di 20 ms, dove però la durata del livello logico alto è stabilita da un bus a 12 bit proveniente da un calcolatore di duty cicle. Dato il conteggio proveniente dal contatore e conoscendo il suo periodo di incremento, basta che in ogni istante lo si confronti con il duty cicle richiesto. Se dall'inizio del periodo del segnale è trascorso un tempo minore della durata dell'impulso desiderata, l'uscita deve diventare o rimanere alta; viceversa, bisogna che raggiunga o mantenga il livello logico basso. Verilog può codificare tale meccanismo attraverso

poche righe di codice, come si osserva in Figura 4.14. La variabile tOn va moltiplicata per 50 in modo da convertirla in unità di conteggio (50 è il risultato della divisione tra l'unità di tOn, cioè 1 µs, e il periodo di clock con cui viene incrementato il conteggio, ovvero 20 ns).

```
1   module pwm_generator(clock, reset, tOn, count, pwm);
2
3   input clock, reset;
4   input [11:0] tOn;
5   input [19:0] count;
6   output reg pwm;
7
8   always @(posedge clock) begin
9       if(reset)
10          pwm <= 1'b0;
11      else
12          pwm <= (count <= tOn*50);
13  end
14
15  endmodule
```

Figura 4.14: Codice sorgente Verilog del generatore di segnale PWM.

Stavolta la vista RTL rispecchia la semplicità del relativo codice sorgente: in Figura 4.15 è possibile esaminare la rete logica sintetizzata dal RTL Viewer. Effettivamente, dopo aver analizzato lo schema nel dettaglio, si conclude che il codice viene interpretato correttamente da Quartus.

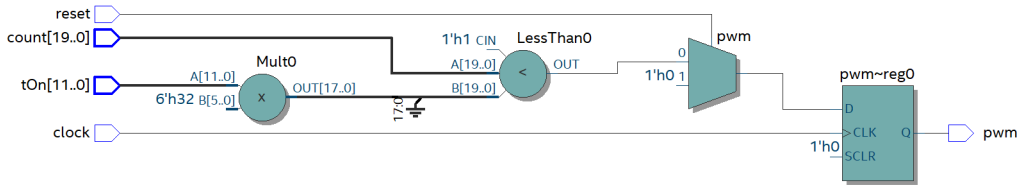

Figura 4.15: Vista RTL del generatore di segnale PWM.

Passando alla scrittura del test bench, per verificare che il circuito si comporti secondo quanto desiderato basta stimolarlo con diversi valori di duty cicle. Dato che la generazione del segnale PWM dipende dallo stato della variabile di conteggio del contatore, occorre che il codice sorgente incorpori anche quest'ultimo modulo. Come si può osservare in Figura 4.16, dopo aver importato i componenti d'interesse e aver generato il clock fittizio a 50 MHz, basta modificare tOn a intervalli sufficientemente lunghi e osservare l'uscita mediante il simulatore RTL (Figura 4.17). La simulazione conferma la corretta progettazione del circuito: utilizzando due cursori, è possibile misurare la durata di tOn, che corrisponde a quella letta in ingresso (2 ms).

```
1   `timescale 1ns/1ps
2
3   module pwm_generator_tb();
4
5   parameter period = 20;
6
7   reg clock, reset;
8   reg [11:0] ton;
9   wire pwm;
10  wire [19:0] count;
11
12  counter counter(.clock(clock), .reset(reset), .count(count));
13  pwm_generator UUT(.clock(clock), .reset(reset), .ton(ton), .count(count), .pwm(pwm));
14
15  always begin
16      clock = 1'b0; #(period/2);
17      clock = 1'b1; #(period/2);
18  end
19
20  initial begin
21      ton = 1000; reset = 1'b1; #period;
22      reset = 1'b0; #(2_000_000*period);
23      ton = 2000; #(2_000_000*period); $stop;
24  end
25
26  endmodule
```

Figura 4.16: Codice sorgente Verilog del test bench del generatore di segnale PWM.

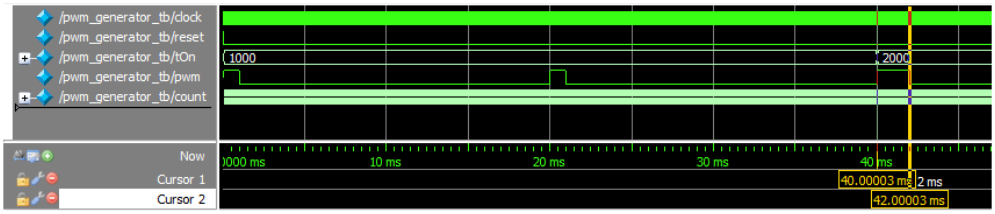

Figura 4.17: Sezione della simulazione RTL del generatore di segnale PWM.

4.5 Configurazione finale

In ultima istanza va analizzato il top-level design. Si tratta del modulo principale del progetto, ed è il primo che viene inizializzato dalla scheda FPGA. Il top-level design svolge essenzialmente due compiti: per prima cosa si interfaccia direttamente con i pin del dispositivo reindirizzandone i segnali ai sotto-moduli del circuito; in secondo luogo gestisce le interconnessioni tra questi ultimi. Prima di procedere bisogna chiarire la quantità di servomotori, quindi di linee d'ingresso e d'uscita, che il dispositivo deve controllare: per garantire al braccio meccanico una buona mobilità, come viene approfondito nel Capitolo 5, è opportuno connettere quattro servomotori. Per differenziarli ci si può riferire ad ognuno con il nome di una diversa articolazione del braccio umano: spalla, gomito, polso e mano.
Il codice sorgente Verilog viene mostrato nella Figura 4.18. I pin di input sono quelli relativi alla linea di clock a 50 MHz, ai quattro pulsanti e ai primi cinque interruttori presenti sulla scheda. Uno degli interruttori funge da reset

dei calcolatori di duty cicle, in modo da riportare il braccio alla posizione di partenza non appena viene abbassato e da non consentire il movimento del braccio finché non viene alzato (è invertito in modo da risultare *active low*). L'uscita viene trasmessa tramite quattro pin GPIO appartenenti alla porta d'espansione della Altera DE2-115 (le denominazioni dei pin vanno ricercate all'interno del manuale della scheda[13]). Inoltre vengono realizzati anche i collegamenti interni: un wire per il clock a 50 Hz, quattro bus a 12 bit per i duty cicle dei servomotori e un ultimo bus a 20 bit per la variabile di conteggio del contatore.
Ad ogni calcolatore di duty cicle vengono passati specifici parametri che dipendono dalle caratteristiche della singola articolazione. Per prima cosa va regolata l'apertura massima del gripper: il modello scelto (Paragrafo 1.3, Figura 1.6) è composto da un meccanismo a due ingranaggi, quindi l'angolo di apertura equivale al doppio dell'angolo di rotazione dell'albero del relativo servomotore. Allora, per evitare che la pinza collida con la staffa sulla quale è montata, la rotazione va interdetta al di sotto dei 90°. Allo stesso modo, onde scongiurare la possibilità che la struttura si scontri con la base d'appoggio, va limitato anche il movimento dell'articolazione gomito. Inoltre, essendo che l'articolazione polso è deputata al movimento del solo gripper, è opportuno aumentarne la velocità modificando il passo corrispondente.
Infine, vanno collegati i pin e costituite le interconnessioni tra i moduli: si tenga a mente che i pulsanti (KEY) sono active low, dunque vanno invertiti.

```
 1  module arm_control_circuit(CLOCK_50, SW, KEY, GPIO);
 2
 3  input CLOCK_50;
 4  input [3:0] KEY;
 5  input [4:0] SW;
 6  output [3:0] GPIO;
 7
 8  wire clock50Hz;
 9  wire [11:0] handTOn, wristTOn, elbowTOn, shoulderTOn;
10  wire [19:0] count;
11
12  counter counter(.clock(CLOCK_50), .reset(1'b0), .count(count));
13
14  downclock downclock(.clockIn(CLOCK_50), .reset(1'b0), .count(count), .clockOut(clock50Hz));
15
16  dutycicle_calculator #(.minTOn(1500)) handDriver(.clock(clock50Hz), .reset(~SW[4]),
17                                  .reverse(SW[3]), .rotate(~KEY[3]), .tOn(handTOn));
18  pwm_generator handPwm(.clock(CLOCK_50), .reset(1'b0), .tOn(handTOn), .count(count),
19                                                                  .pwm(GPIO[3]));
20
21  dutycicle_calculator #(.step(30)) wristDriver(.clock(clock50Hz), .reset(~SW[4]),
22                               .reverse(SW[2]), .rotate(~KEY[2]), .tOn(wristTOn));
23  pwm_generator wristPwm(.clock(CLOCK_50), .reset(1'b0), .tOn(wristTOn), .count(count),
24                                                                  .pwm(GPIO[2]));
25
26  dutycicle_calculator #(.minTOn(650), .maxTOn(2350)) elbowDriver(.clock(clock50Hz),
27                .reset(~SW[4]), .reverse(SW[1]), .rotate(~KEY[1]), .tOn(elbowTOn));
28  pwm_generator elbowPwm(.clock(CLOCK_50), .reset(1'b0), .tOn(elbowTOn), .count(count),
29                                                                  .pwm(GPIO[1]));
30
31  dutycicle_calculator shoulderDriver(.clock(clock50Hz), .reset(~SW[4]), .reverse(SW[0]),
32                                                  .rotate(~KEY[0]), .tOn(shoulderTOn));
33  pwm_generator shoulderPwm(.clock(CLOCK_50), .reset(1'b0), .tOn(shoulderTOn),
34                                                  .count(count), .pwm(GPIO[0]));
35
36  endmodule
```

Figura 4.18: Codice sorgente Verilog della configurazione finale.

I collegamenti possono risultare più chiari dando uno sguardo alla vista RTL in Figura 4.19. Si prenda nota del fatto che Quartus interpreta ognuno dei sotto-moduli come una scatola nera con i suoi ingressi e le sue uscite. In effetti facendo doppio click su di queste viene mostrato in dettaglio il singolo circuito in questione, dunque è possibile apprezzare anche le differenze tra le istanze degli stessi moduli dovute alla modifica dei parametri. Da questo punto di vista, il circuito appare correttamente sintetizzato.

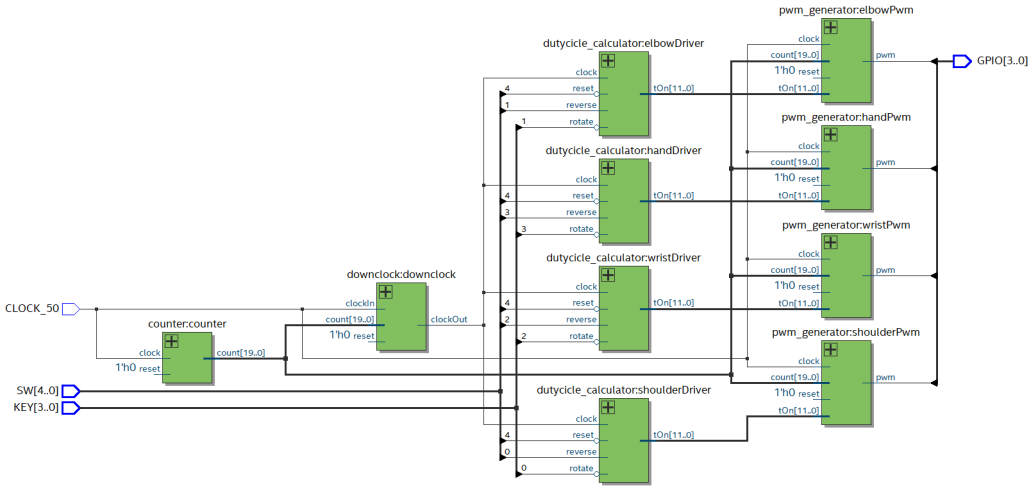

Figura 4.19: Vista RTL della configurazione finale.

Per concludere, non resta che scrivere il test bench (Figura 4.20) e simulare la configurazione finale. Non è necessario che venga simulato l'intero circuito, ma basta un singolo ramo connesso ad un unico servomotore. Per via di tale modifica rispetto al codice così come è stato scritto, e per avere l'opportunità di visualizzare anche i segnali intermedi (il clock a 50 Hz, la variabile di conteggio e il valore del duty cicle), non viene importato direttamente il modulo principale, ma al suo posto i sotto-moduli che lo compongono, ovviamente interconnessi in modo adeguato. Generato anche stavolta il clock, nel processo initial la circuiteria viene stimolata richiedendo l'incremento e il decremento della posizione angolare dell'albero motore mediante l'azionamento dei segnali che rappresentano gli ingressi fisici della scheda, il tutto per durate temporali opportune (similmente a quanto fatto nel Paragrafo 4.3 con il calcolatore di duty cicle).
In Figura 4.21 è riportata la schermata della simulazione RTL relativa. Il circuito finale risponde correttamente agli stimoli corrispondenti agli ingressi fisici, dato che il duty cicle aumenta oppure diminuisce quando il segnale del pulsante assume valore logico basso e relativamente alla tensione dell'ingres-

so equivalente all'interruttore. Come desiderato, inoltre, il segnale di clock a 50 Hz e il segnale PWM in output hanno lo stesso periodo e sono sincroni tra di loro. Ne consegue che tOn, che si aggiorna soltanto ai fronti di salita del clock a 50 Hz, resta costante per un lungo intervallo temporale attorno alla finestra in cui effettivamente viene letto, ovvero quando l'uscita sul pin GPIO è alta: come già discusso, così facendo si evitano malfunzionamenti dovuti a una possibile cattiva lettura da parte del generatore di segnale PWM.

```
1   `timescale 1ns/1ps
2
3   module arm_control_circuit_tb();
4
5   reg clock, sw, key, reset;
6   wire clock50Hz, gpio;
7   wire [11:0] tOn;
8   wire [19:0] count;
9
10  parameter period = 20;
11
12  counter counter(.clock(clock), .reset(reset), .count(count));
13  downclock downclock(.clockIn(clock), .reset(reset), .count(count),.clockOut(clock50Hz));
14  dutycicle_calculator dutycicle_calculator(.clock(clock50Hz), .reset(reset), .reverse(sw),
15                                            .rotate(~key), .tOn(tOn));
16  pwm_generator pwm_generator(.clock(clock), .reset(reset), .tOn(tOn), .count(count),
17                                                          .pwm(gpio));
18
19  always begin
20      clock = 1'b0; #(period/2);
21      clock = 1'b1; #(period/2);
22  end
23
24  initial begin
25      reset = 1'b1; sw = 1'b0; key = 1'b0; #period;
26      reset = 1'b0; #(3_000_000*period);
27      sw = 1'b1; #(3_000_000*period); $stop;
28  end
29
30  endmodule
```

Figura 4.20: Codice sorgente Verilog del test bench della configurazione finale.

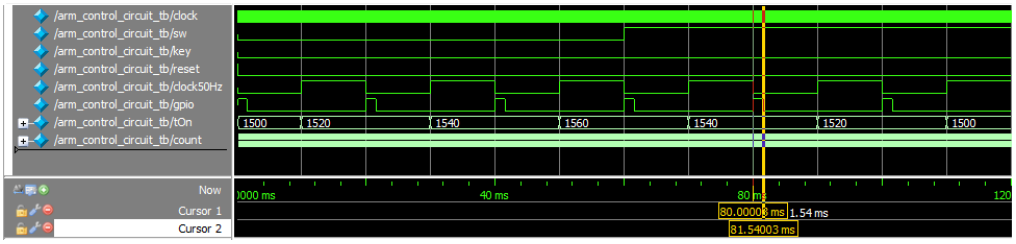

Figura 4.21: Simulazione RTL della configurazione finale.

Capitolo 5

Costruzione del braccio meccanico

Naturalmente, per completare il prototipo di sistema di controllo occorre corredarlo di un opportuno supporto fisico che faccia da attuatore. L'obiettivo del presente capitolo è quello di descrivere il processo di assemblaggio e connessione dei componenti tangibili del progetto, dalla linea di alimentazione della scheda DE2-115 alle articolazioni del braccio meccanico. Fatto questo, per concludere è necessario verificare che il dispositivo finale si comporti in modo conforme a quanto dichiarato nel capitolo precedente, evidenziando eventuali criticità riscontrate durante il suo funzionamento.

5.1 Assemblaggio dei componenti

Come accennato, uno degli aspetti fondamentali che bisogna curare è l'alimentazione dei dispositivi elettronici. In accordo con quanto descritto nel Capitolo 2, le periferiche da alimentare sono la scheda FPGA e il modulo d'espansione del Terasic Servo Motor Kit. Nel primo caso il device viene venduto assieme ad un alimentatore stabilizzato AC/DC da 12 V e 2 A, mentre il SMK non ne è provvisto poiché l'alimentazione va dimensionata in base al progetto. Per quanto riguarda quest'ultimo, nel manuale[7] sono indicate le specifiche circa le dimensioni del jack e la tensione in ingresso, la quale va da 6.2 a 12 V. Visto che i servomotori da pilotare sono quattro, e dato che ognuno può assorbire da 10 a 1400 mA di corrente, e tenendo anche conto delle apparecchiature a disposizione, è stato scelto di utilizzare un alimentatore da 12 V e 0.5 A. Entrambi gli alimentatori vanno collegati alla rete domestica e ai jack sulle rispettive schede, le quali vanno inoltre interconnesse tra di loro tramite un cavo a 40 pin come illustrato nel Paragrafo 2.3.

In riferimento a quanto affermato nel Paragrafo 1.3, i servomotori Tower-Pro MG996R sono dotati di cavi lunghi 32 cm che terminano con connettori JR. In genere bisognerebbe semplicemente collegare questi ultimi ai pin maschi sulla daughter card del Terasic SMK, prestando particolare attenzione a non invertirne erroneamente il verso. Il problema è che la lunghezza standard dei cavi in questione può ostacolare i movimenti del braccio: allora è necessario utilizzare delle prolunghe adeguate, come i cavi in Figura 5.1.

Figura 5.1: Cavo JR maschio/femmina.

Il punto fondamentale del setup prototipale è la realizzazione del braccio meccanico vero e proprio. Se il compito dei servomotori MG996R è quello di ruotare le articolazioni, lo scheletro deve essere composto da una struttura adatta, ad esempio costituita da staffe compatibili come quelle rappresentate nel primo capitolo in Figura 1.6. Il dispositivo finale può essere ritenuto realmente utile soltanto se in grado di compiere alcune semplici azioni come il sollevamento e lo spostamento di oggetti: nella medesima figura è rappresentato anche l'organo di presa scelto, ovvero il gripper, anch'esso compatibile con il modello MG996R.
In totale il braccio deve disporre di (almeno) quattro gradi di libertà, corrispondenti a quattro articolazioni mosse da altrettanti servomotori, in modo da avere la capacità di manipolare con efficacia oggetti posti in qualsiasi direzione entro la sua portata. L'illustrazione in Figura 5.2 ne descrive concettualmente la struttura finale e, considerando che i servomotori MG996R possono ruotare fino a 180° e tenendo conto delle limitazioni imposte per via circuitale (Paragrafo 4.5), riporta indicativamente l'estensione dei suoi gradi di libertà (si ricorda che con il gripper scelto una rotazione di 90° equivale ad un'apertura di 180°). I componenti mobili del braccio vanno installati su di una massiccia base: tale accorgimento aiuta a prevenire che la struttura si destabilizzi conseguentemente alla rotazione delle articolazioni (a causa della conservazione del momento angolare).

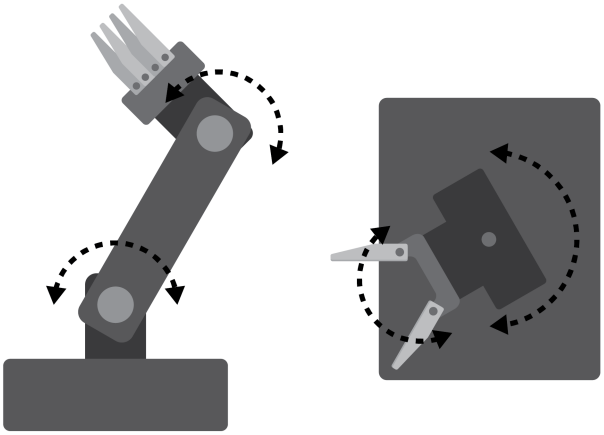

Figura 5.2: Illustrazione della struttura finale e dei gradi di libertà del braccio meccanico. A sinistra: piano laterale. A destra: piano orizzontale.

Generalmente i componenti strutturali scelti sono venduti disassemblati, quindi dopo averli acquistati bisogna rimetterli insieme facendo uso di svariati tipi di viti, bulloni e distanziali. Al contrario la base è realizzata artigianalmente. Nella fase finale di montaggio è necessario prestare particolare attenzione ai versi e agli assi di rotazione dei servomotori relativamente al posizionamento delle staffe. Una volta sistemati i cavi con delle fascette da elettricista, non resta che collegarli al circuito di controllo: il risultato complessivo viene mostrato in Figura 5.3.

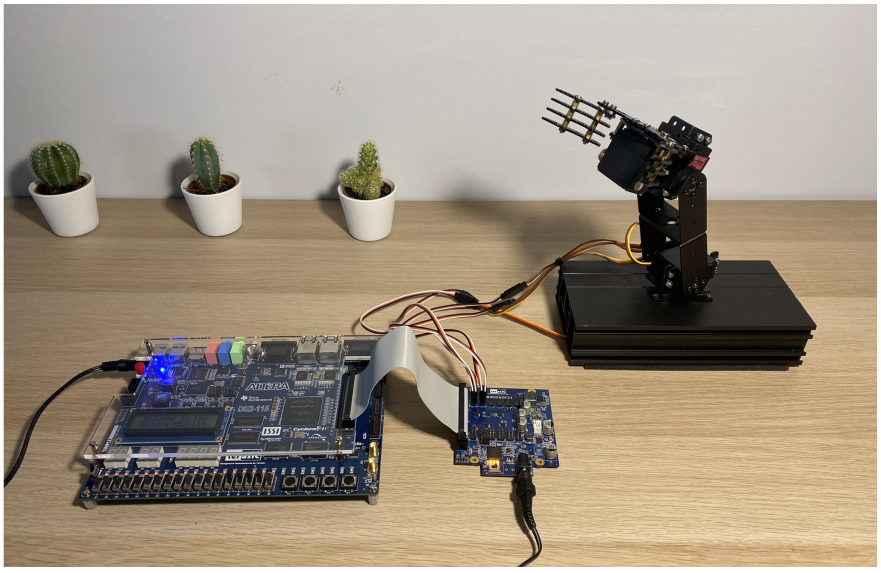

Figura 5.3: Setup finale del prototipo.

5.2 Verifica del funzionamento

Anche se nel Capitolo 4 ogni modulo del circuito di controllo è stato simulato nel dettaglio, non è scontato che il braccio meccanico risponda correttamente ai segnali che riceve. Quindi occorre che il suo funzionamento sia verificato a dovere, andando ad appurare che il dispositivo finale rispecchi con fedeltà il comportamento che si desiderava ottenere nella fase di progettazione. Durante questa analisi potrebbero sorgere problemi che precedentemente non sono stati considerati: in tal caso è necessario evidenziarli, in modo da poterli correggere nelle versioni future.
Riprendendo la configurazione mostrata nel paragrafo precedente, dopo aver attivato l'alimentazione e configurato il dispositivo FPGA in modalità JTAG è necessario disinnescare il reset (ovvero sollevare il quinto interruttore da destra) per abilitare il comando del braccio. Una volta fatto ciò, è possibile adoperare pulsanti ed interruttori per azionare individualmente i servomotori. Di seguito (Figura 5.4, 5.5, 5.6 e 5.7) sono riportate alcune foto che illustrano i movimenti della struttura; osservando questi ultimi in relazione agli input impartiti mediante l'interfaccia utente, risulta immediato giungere alla conclusione che il dispositivo funziona correttamente e soddisfa le aspettative in termini di velocità e limiti di rotazione. Tuttavia, va evidenziata una criticità: quando il braccio incontra un vincolo fisico (ad esempio quando si cerca di sollevare un oggetto fisso o troppo pesante) occorre che l'operatore presti estrema attenzione, in quanto il circuito di controllo non ha modo di rendersi conto del fatto che il servomotore adoperato non riesce a raggiungere la posizione desiderata, quindi continua ad inviare il segnale PWM che vi corrisponde causando un eccessivo assorbimento di corrente, che a sua volta può portare al danneggiamento dei componenti elettrici.

Figura 5.4: Rotazione dell'articolazione spalla.

Figura 5.5: Rotazione dell'articolazione gomito.

Figura 5.6: Rotazione dell'articolazione polso.

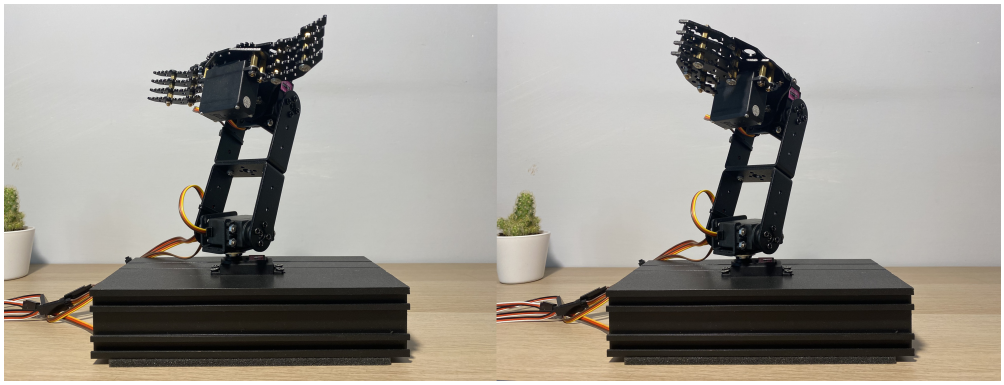

Figura 5.7: Apertura e chiusura del gripper.

Per finire, in Figura 5.8 sono riportate alcune istantanee di un video che dimostra l'azionamento del prototipo per lo spostamento di un oggetto, a

testimonianza del fatto che, come desiderato, effettivamente il dispositivo finale è capace di rendersi utile per lo svolgimento di semplici compiti di manipolazione di oggetti dalle dimensioni contenute.

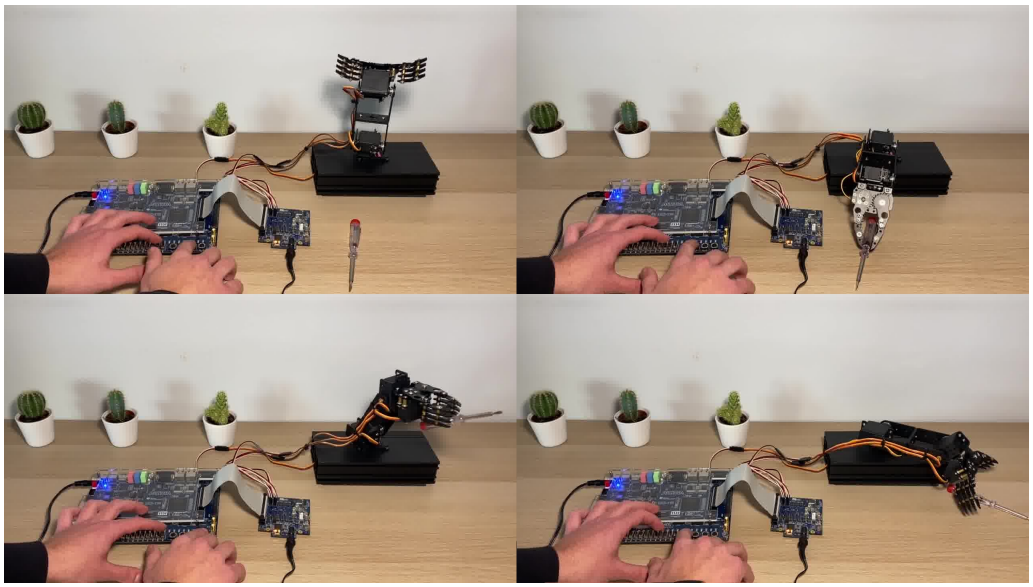

Figura 5.8: Utilizzo del braccio meccanico per il sollevamento di un oggetto.

Conclusioni

Naturalmente il risultato ottenuto è soltanto un punto di partenza per quello che potrebbe diventare un progetto ben più articolato. In questo senso, un possibile aspetto da migliorare riguarda l'interfaccia utente, la quale al momento è grezza ed essenziale: pulsanti e interruttori andrebbero sostituiti con periferiche di input più intuitive, come una leva di comando oppure una tastiera. Inoltre, per risolvere le criticità riguardanti il movimento della struttura del braccio meccanico si potrebbero installare dei sensori per la misura di prossimità o di corrente assorbita (e la relativa circuiteria di retroazione), in modo da evitare che i servomotori vadano in stallo cercando di ruotare in presenza di un vincolo fisico. Oltretutto questi ultimi potrebbero essere sostituiti con dei modelli più affidabili e prestanti, ridimensionando a dovere le relative linee di alimentazione. Esiste un ampio margine di perfezionamento anche per quanto riguarda la struttura del braccio, la quale potrebbe essere disegnata in CAD e stampata in 3D in base ai requisiti di applicazione. Infine, una volta definita la versione finale del circuito di controllo, sarebbe opportuno migrare il supporto tecnologico da FPGA ad ASIC, in modo da ottenere un device più compatto ed efficiente (magari unificando le funzioni della scheda DE2-115 e del Terasic Servo Motor Kit, e congiungendo le due linee di alimentazione a 12 V).

Al giorno d'oggi i bracci meccanici più avanzati sono mossi da micromotori estremamente precisi, a loro volta comandati da complesse circuiterie in grado di implementare sofisticati algoritmi di controllo che elaborano misure acquisite in tempo reale da una rete di sensori molto accurati. Sebbene sotto svariati punti di vista questa tecnologia sia ancora acerba, probabilmente in un futuro non troppo distante, con l'evoluzione del controllo dei servomeccanismi e il perfezionamento delle interfacce neurali, essa sarà in grado di migliorare le condizioni di lavoro o di vita di molte persone (basti pensare al settore del manifatturiero pesante oppure alla medicina protesica). Come scrive Rozum Robotics[3], nel mondo di domani le persone dovrebbero abbracciare la creatività, mentre i robot dovrebbero agire come i loro collaboratori instancabili e sempre attenti. In altre parole, ci si aspetta che le persone diano alla luce idee e le macchine collaborino per realizzarle, e per questo è indispensabile che merci e materiali vengano manipolati con l'aiuto di bracci meccanici sempre migliori.

Bibliografia

[1] *Automazione Industriale in "Enciclopedia Italiana"*. URL: https://www.treccani.it/enciclopedia/automazione-industriale_%28Enciclopedia-Italiana%29/.

[2] Bill Lackey. *What's the difference between servo and stepper motors?* 2018. URL: https://www.machinedesign.com/mechanical-motion-systems/article/21836868/whats-the-difference-between-servo-and-stepper-motors.

[3] Rozum Robotics. *Robotic Arm with Servo Motors — A Step Closer to Future*. 2019. URL: https://rozum.com/servos-as-robot-components/.

[4] Kollmorgen Experts. *How Servo Motors Work*. 2020. URL: https://www.kollmorgen.com/en-us/blogs/_blog-in-motion/articles/how-servo-motors-work/.

[5] Darren Sawicz. *Hobby Servo Fundamentals*. URL: https://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf.

[6] Michael Barr. *Introduction to Pulse Width Modulation (PWM)*. 2001. URL: https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation.

[7] *Terasic Servo Motor Kit user manual*. URL: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=1025&FID=d1d10684aa5c6a87efc407b52d504104.

[8] Peter Bishop. *A tradeoff between microcontroller, DSP, FPGA and ASIC technologies*. 2009. URL: https://www.eetimes.com/a-tradeoff-between-microcontroller-dsp-fpga-and-asic-technologies/.

[9] Jessica Hopkins. *Comparing FPGA vs Microcontroller – Which is Best for Your Needs?* 2021. URL: https://www.totalphase.com/blog/2021/04/comparing-fpga-vs-microcontroller/.

[10] Antonio De Rosa Catello. *Introduzione alle Logiche Programmabili*. Edizioni dell'Ambrosino, 2003.

[11] Jan M. Rabaey, Anantha Chandrakasan e Borivoje Nikolic. *Circuiti integrati digitali. L'ottica del progettista*. Pearson, 2020.

[12] Rodolfo Zunino. *Dispense del corso di Sistemi Elettronici Dedicati 1*. 2004. URL: http://sealab.diten.unige.it/sitoDidattica/Files/fpga.pdf.

[13] *Altera DE2-115 user manual*. URL: https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf.

[14] *Intel Quartus Prime Design Software Brochure*. URL: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/quartus-prime-software-brochure.pdf.

[15] Marco Pisco. *Dispense del corso di Circuiti Elettronici Programmabili*. 2021.

[16] Ettore Napoli. *Progetto di Sistemi Elettronici Digitali Basati Su Dispositivi FPGA*. Esculapio, 2011.