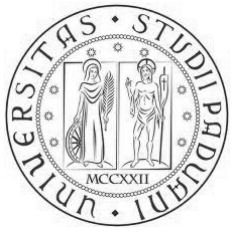

Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Elettronica

Tesi di Laurea Triennale

Sviluppo su piattaforma FPGA di interfacce per sensori video per applicazioni oftalmiche

Relatore: Prof. Daniele Vogrig

Laureando: Federico Sattin

28 settembre 2010

A mamma Rita e papà Gilberto

Sommario

Questa tesi riguarda il lavoro svolto durante il periodo di tirocinio presso NIDEK Technologies S.r.l., un'azienda con sede ad Albignasego (Padova, Italia) che progetta, produce e commercializza apparecchi oftalmici. L'azienda fa parte del gruppo giapponese NIDEK CO. LTD. dal 2001.

La collaborazione con l'azienda prevedeva lo sviluppo di interfacce per sensori video scritte in Verilog su FPGA da impiegare nei prodotti dell'azienda. Il progetto finale voluto dall'azienda prevede la creazione di una scheda prototipo per l'acquisizione ed il trattamento di immagini e video da installare sul topografo corneale Magellan Mapper. Si è scelto un FPGA per poter poi adottare la stessa scheda in tutti i prodotti dell'azienda apportando meno modifiche possibili. Le periferiche da interfacciare al *chip* FPGA sono un sensore CCD, un convertitore analogico/digitale (ADC), una memoria DDR2 ed un microcontrollore la cui funzione principale è lo scambio di dati con periferiche compatibili col protocollo USB 2.0. Normalmente queste periferiche sono calcolatori elettronici su cui eseguire programmi per l'utilizzo degli apparecchi prodotti dall'azienda o per l'elaborazione delle immagini e dei dati medici. Le interfacce da realizzare sono principalmente tre: una per il sistema di acquisizione (CCD – ADC), una per la memoria DDR2 ed una per il microcontrollore. Il tempo a disposizione per il tirocinio ha permesso di trattare prevalentemente le interfacce con il sistema di acquisizione e con il microcontrollore.

La tesi si apre con un capitolo introduttivo in cui si presentano l'azienda ed i suoi prodotti, gli FPGA e quello usato in questo progetto ed infine il Verilog come linguaggio di descrizione dell'hardware. Si prosegue con un capitolo che descrive il sistema hardware attualmente in uso e quello che si sta progettando. Due capitoli descrivono poi le due interfacce realizzate o scelte per l'utilizzo per arrivare infine al capitolo conclusivo che presenta i risultati ottenuti.

Indice

1 Introduzione

In questo capitolo viene presentata l'azienda presso la quale è stato svolto il tirocinio e sviluppato il progetto. Successivamente è introdotta la tecnologia degli FPGA e presentato il particolare dispositivo utilizzato. Infine, viene fornito qualche cenno sul Verilog come linguaggio di descrizione dell'hardware e confrontato con l'altro linguaggio presente nel mercato: il VHDL.

1.1 Informazioni su NIDEK Technologies S.r.l.

NIDEK Technologies è una società specializzata nella ricerca, sviluppo e produzione di strumenti diagnostici ad alto contenuto tecnologico e di sistemi informatici per l'oftalmologia.

La società originaria, NIDEK CO. LTD, fondata nel 1971 a Gamagori (Giappone) da Hideo Ozawa con altri sei soci, è nata con l'obiettivo di "rendere le cose da invisibili a visibili" coniugando insieme ottica ed elettronica.

NIDEK Technologies, incorporata nel Gruppo giapponese NIDEK nel 2001, si dedica alla progettazione e allo sviluppo di strumentazioni innovative e tecnologicamente avanzate, e sistemi software per la diagnostica in oftalmologia.

La filiale con sede ad Albignasego (Padova), collabora con la casa madre operando anche come base per il supporto alla vendita e la distribuzione nei mercati di Europa, Medio Oriente e Africa.

L'obiettivo principale e la missione di NIDEK Technologies all'interno del Gruppo NIDEK consiste nel realizzare prodotti innovativi e all'avanguardia, che rispecchino i più alti standard qualitativi per l'industria *Eye Care*. In questa sua missione NIDEK Technologies collabora attivamente con le istitu-

zioni accademiche e i centri di ricerca internazionale per lo sviluppo e la commercializzazione del know-how e della tecnologia.

La sede alloggia i seguenti reparti: direzione, uffici amministrativi, ricerca e sviluppo, marketing e vendite. La struttura include le linee produttive di cinque strumenti diagnostici in oftalmologia e i relativi sistemi software.

Attualmente NIDEK Technologies sviluppa e commercializza il software NAVIS (NIDEK Advanced Vision Information System), il Topografo Corneale Magellan Mapper, il sofisticato Microperimetro MP-1, il CS-4 Confocal Microscope per applicazioni di microscopia corneale e, ultimo in ordine di produzione, l'autoretinografo Orion.

Nel quartier generale a Gamagori, NIDEK sviluppa, produce e vende sistemi laser e attrezzature diagnostiche di oftalmologia, optometria, chirurgia generale e chirurgia cosmetica e dermatologica. I prodotti NIDEK per l'oftalmologia e l'optometria sono specificamente progettati per diagnosticare e curare malattie retiniche, glaucoma, terapie refrattive, retinopatie precoci, terapie pre e post interventi di cataratta. I prodotti chirurgici NIDEK sono progettati per trattare pazienti nelle crescenti applicazioni della chirurgia laser cosmetica e plastica. NIDEK distribuisce i suoi prodotti attraverso una rete distributiva formata da partner collaboratori e una forza vendita diretta.

NIDEK Technologies ha ottenuto la certificazione EN 46001 (Sistemi di Qualità – Dispositivi Medici – Particolari requisiti per l'applicazione della ISO 9001) da parte dell'organismo notificato TÜV Product Service [1], ed è quindi autorizzata ad apporre la marcatura CE (secondo la direttiva 93/42/CEE) ai propri strumenti, dispositivi ed applicazioni software di misura in diagnostica oftalmologia. La società ha inoltre ottenuto la certificazione secondo la norma ISO 9001:2000 (Sistemi di Gestione per la Qualità – Requisiti, anche nota come "Vision 2000"), in anticipo rispetto all'obbligatorietà, e la certificazione secondo la norma ISO 13485:2003 (Dispositivi Medici – Sistemi di Gestione per la Qualità – Requisiti per Scopi Regolamentari).

Grazie alla collaborazione di NIDEK Technologies con il Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Padova è nato

questo progetto nell'ambito dello sviluppo di un'evoluzione dell'attuale Topografo Corneale Magellan Mapper.

Figura 1.1.1. La sede di NIDEK Technologies S.r.l. ad Albignasego (PD).

Nel seguito vengono presentate brevemente le macchine prodotte in sede da NIDEK Technologies. Le informazioni sono state prese dal sito web dell'azienda [2], dove sono reperibili altri dettagli sulle caratteristiche delle macchine e del loro campo di utilizzo.

Orion – Autoretinal Imaging

ORION, visibile in Figura 1.1.2, e' una *fundus* camera non midriatica completamente automatica che non necessita di un operatore dedicato. Essa identifica la presenza di un paziente ed acquisisce varie immagini dei differenti campi retinici compensando il difetto ottico del paziente. La cattura automatica delle immagini, il salvataggio e la visualizzazione su un computer esterno vengono effettuati attraverso una connessione wireless. Il design è progettato perché la macchina possa operare in assenza dell'operatore.

Figura 1.1.2. La *fundus* camera ORION.

Confoscan 4

Il Confoscan 4 (Figura 1.1.4) è uno strumento diagnostico unico che combina in sé un microscopio confocale, un microscopio endoteliale ed un pachimetro[1] di precisione non a contatto. È il primo pachimetro basato su un microscopio confocale capace di eseguire misure complete dello spessore della cornea e localizzare qualsiasi struttura intra-cornea. Come microscopio endoteliale non a contatto completamente automatizzato fornisce un'analisi endoteliale veloce in qualsiasi condizione, non essendo influenzato nemmeno dall'opacità corneale.

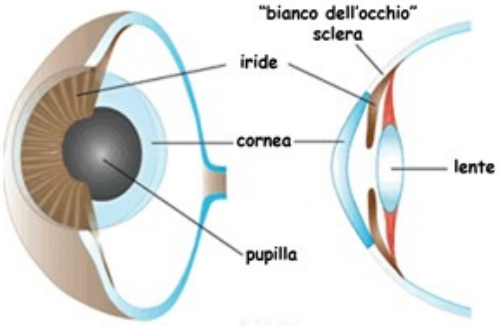

Figura 1.1.3. Una sezione dell'occhio che evidenzia le sue strutture frontali.

1 *Uno strumento che misura lo spessore della cornea. Un pachimetro può essere "a contatto", quando si appoggia alla cornea, o, in caso contrario, "non a contatto".*

Figura 1.1.4. Il microscopio confocale CS4.

MP-1 Microperimetro

Il Microperimetro MP-1 integra in un unico strumento i dati soggettivi della perimetria computerizzata ed i dati oggettivi delle immagini retiniche ottenendo misurazioni precise, ripetibili e completamente automatiche della funzione retinica e di quella maculare. Realizza quindi una combinazione di perimetria computerizzata e retinografia. Lo strumento è mostrato in Figura 1.1.5.

Figura 1.1.5. Il microperimetro MP-1.

MM1 Topografo Corneale Magellan Mapper.

Il topografo corneale Magellan Mapper (Figura 1.1.6) fornisce delle informazioni sulla curvatura e potere della cornea, permettendo la valutazione dell'astigmatismo e la diagnosi di svariate patologie corneali. È il primo topografo con funzionalità *Plug and Play*; si connette infatti via USB a qualsiasi computer portatile o desktop operante con Windows 2000 o XP e che utilizzi la nuova applicazione software di topografia corneale NIDEK. Le caratteristiche peculiari di questo strumento sono le seguenti:

- 60 anelli analizzati per un numero di punti di misurazione fino a 21.600.
- Cono con illuminazione a bassa intensità.
- Allineamento *Software* servito.
- *Plug and Play.*
- Corneal Navigator™.
- *Corneal Aberrometry.*
- *Orthokeratology.*
- *TMS Import tool.*
- *Contact Lens Fitting software.*

Figura 1.1.6. Il topografo corneale Magellan Mapper.

La cornea rivestita dal film lacrimale costituisce la prima superficie ottica del globo oculare ed è responsabile di circa i due terzi del potere ottico totale [3]. Le lenti a contatto o la chirurgia refrattiva possono quindi modificare le caratteristiche refrattive dell'occhio. La modificazione della geometria corneale, congenita o acquisita, fisiologica o patologica è responsabile di cambiamenti della qualità della visione.

Tra i metodi di indagine qualitativa vi è la cheratoscopia col disco di Placido (Figura 1.1.7). Questo è uno strumento in cui sono disegnati dei cerchi concentrici di colore bianco alternati ad altri di colore nero. Nel centro è posto un foro attraverso il quale l'osservatore rileva l'immagine che si riflette sulla superficie corneale.

Figura 1.1.7. Un esempio di disco di Placido.

L'andamento di regolarità (quindi se i cerchi riflessi sono deformati o meno, e quanto lo sono) e la distanza tra i cerchi determinano le caratteristiche morfologiche corneali. La foto-cheratoscopia è un'ulteriore evoluzione della tecnica e consistente nella fotografia dell'immagine riflessa dalla cornea degli anelli di appositi cheratoscopi (si veda Figura 1.1.8).

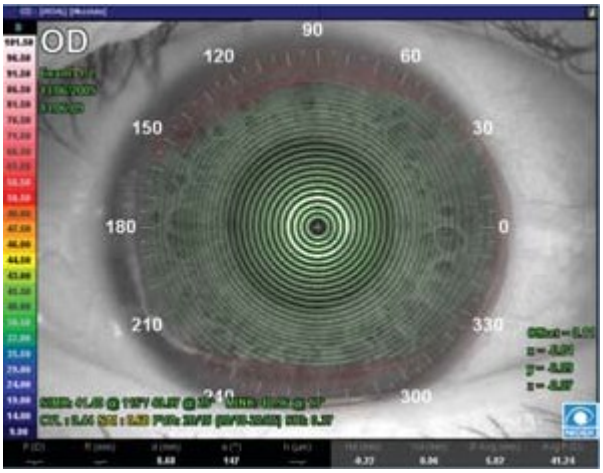

Figura 1.1.8. Gli anelli proiettati sulla cornea.

Il vantaggio di una documentazione fotografica delle condizioni corneali permette di valutare meglio e di monitorare i cambiamenti della cornea in presenza di patologie quali il cheratocono o, in contattologia, per valutare i cambiamenti di curvatura indotti ad esempio dalle lenti a contatto.

Grazie ad una rete neurale software ed utilizzando le più aggiornate statistiche corneali sviluppate dal Dr. Stephen Klyce [4] (della cui collaborazione NIDEK Technologies si avvale), il software Corneal Navigator™ è utilizzato dal Magellan Mapper per lo screening su diverse patologie corneali, quali: Cheratocono (KC), Cheratocono Sospetto (KCS), Degenerazione Marginale Pellucida (PMD), Astigmatismo (AST), Chirurgia Refrattiva Miopica (MRS), Chirurgia Refrattiva Ipermetropica (HRS) e Cheratoplastica Perforante (PKP).

NAVIS

Il *Nidek Advanced Vision Information System* (NAVIS) è il sistema operativo per la gestione integrata in rete del database dei pazienti e di tutti gli esami strumentali usato dai prodotti di NIDEK Technologies. NAVIS è un *software* completo e complesso, che tra le funzioni annovera:

- gestione database pazienti;
- gestione database esami;
- gestione cartelle cliniche;

- storia clinica;
- esame obiettivo;
- gestione appuntamenti e pianificazione delle visite;
- stampa dei dati e della documentazione diagnostica;
- connessione in rete locale;
- connessione remota via internet;
- applicazioni *software* per analisi di immagini, di misura e diagnostiche;
- *data exchange* con altre applicazioni.

Il software include molte altre applicazioni specifiche, per esempio di acquisizione dati da molteplici tipologie di strumenti oftalmici. Per ulteriori informazioni si veda l'apposita sezione del sito web dell'azienda [5].

In tutti gli strumenti oftalmici prodotti da NIDEK Technologies sono presenti sensori per l'acquisizione di immagini. La possibilità di collegare tali strumenti a calcolatori e la disponibilità di *software* specifici fanno sì che l'hardware necessario ad elaborare le immagini all'interno delle macchine possa essere contenuto. Questo, finora, ha comportato evidenti vantaggi in termini di spazio occupato dalle macchine e di costi di produzione, dato che si delega l'onere di parte dell'hardware all'utente finale. Il continuo progresso della tecnologia e le crescenti esigenze di mercato spingono però a realizzare prodotti sempre più competitivi e performanti. In quest'ottica, le capacità di archiviazione e di elaborazione delle immagini da parte degli strumenti determinano un importante valore aggiunto ai prodotti, che risultano così più competitivi. L'intento di questo progetto è iniziare a sviluppare il prototipo di un circuito che permetta l'acquisizione, l'archiviazione temporanea e la trasmissione attraverso una porta USB 2.0 dei dati prodotti dallo strumento che alloggerà la scheda. Inizialmente è stato deciso che lo strumento-destinazione di questo prototipo fosse il Magellan Mapper, ma l'obiettivo più ampio, a lungo termine, prevede l'impiego di questa scheda in tutti i prodotti dell'azienda. L'idea fondamentale del progetto è quindi lo sviluppo di un sistema che sia

quanto più possibile adattabile ai vari strumenti, col minor numero di modifiche hardware. Con queste premesse, ed in considerazione del fatto che i volumi di vendita dei prodotti in questione sono limitati, risulta logico pensare all'utilizzo di un FPGA come nucleo principale del sistema. L'attuale offerta e livello tecnologico di FPGA propone infatti prodotti abbastanza performanti da essere adatti allo scopo. Essendo circuiti completamente programmabili via *software*, oltre a gestire periferiche come quelle previste da questo progetto, essi possono essere anche sede di processori, e quindi integrare funzionalità di elaborazione dei dati estremamente personalizzabili. Questo, ovviamente, non preclude la possibilità di continuare ad usare un calcolatore esterno allo strumento, anzi ne estende le potenzialità. Per queste caratteristiche e per i motivi descritti in precedenza la piattaforma FPGA è risultata la scelta su cui basare il prototipo per un futuro sistema di gestione e trattamento (o pretrattamento) dei dati di tutti gli strumenti di NIDEK Technologies.

1.2 Cenni sugli FPGA

In questo paragrafo sono presentati gli FPGA (*Field Programmable Gate Array*, *array* di *gate* programmabili sul campo) e viene fornito qualche accenno alla loro tecnologia. Le immagini ed i contenuti sono tratti dalle dispense del corso "Laboratorio di Elettronica Digitale" tenuto dal professor Daniele Vogrig dell'Università degli Studi di Padova.

Nel campo degli ASIC (*Application Specific Integrated Circuit*, circuiti integrati specifici per un'applicazione) ci sono differenti tecnologie ed approcci che ne differenziano le tipologie. La Figura 1.2.1 propone una panoramica sugli approcci alla realizzazione di un ASIC inquadrando così la collocazione degli FPGA.

Figura 1.2.1. Approcci alla realizzazione di un ASIC.

Un FPGA è un circuito *semi-custom* (ha prestazioni ottimizzate rispetto alle specifiche del sistema) e *array-based* (contiene una matrice di celle prefabbricate senza interconnessioni). È un insieme di celle ed interconnessioni prefabbricate riconfigurabili via software, e questo costituisce una caratteristica fondamentale per questo progetto. La funzione logica svolta da ciascuna cella e i collegamenti tra i terminali delle celle sono determinati commutando interruttori programmabili. Gli elementi fondamentali dell'architettura di un FPGA sono visibili in Figura 1.2.2.

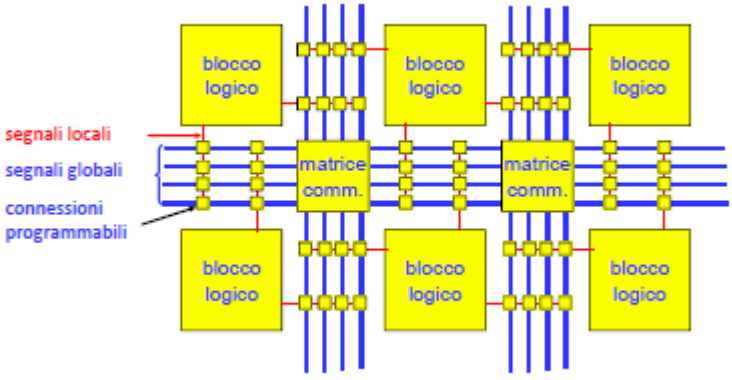

Figura 1.2.2. Architettura di un FPGA: blocchi logici e matrici di commutazione.

I singoli blocchi logici (CLB, *Complex Logic Block*) sono moduli di logica combinatoria con uno o più registri la cui funzione è programmabile. Essi sono inseriti tra le matrici di commutazione e collegati alle loro linee. Le matrici permettono di determinare a piacere le connessioni fra i terminali; in

questo modo è possibile realizzare qualsiasi circuito. La Figura 1.2.3 fornisce uno sguardo sulla struttura interna di un blocco logico e ne evidenzia le peculiarità.

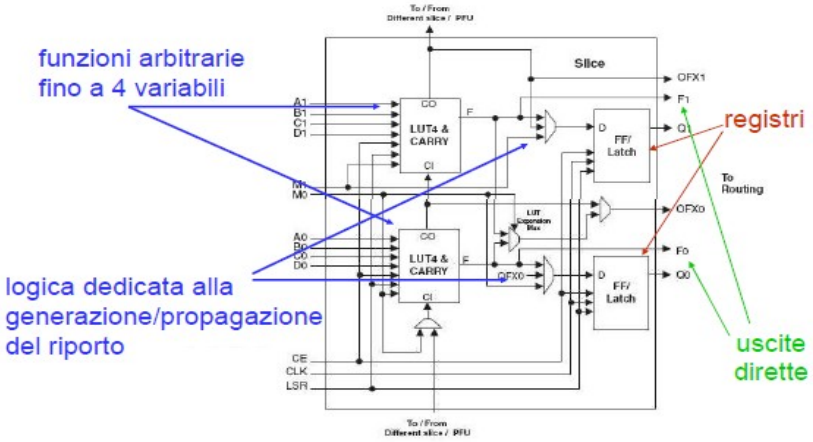

Figura 1.2.3. Struttura interna di un blocco logico (CLB).

Il vero punto di forza di ogni blocco logico sono le LUT (*Look-Up Table*), elementi costituiti da una memoria SRAM a 8 bit e da un *multiplexer* (si veda Figura 1.2.4). Le LUT permettono di realizzare una qualsiasi funzione combinatoria a N ingressi semplicemente cambiando la sequenza di bit che viene scritta nella memoria La LUT della figura, ad esempio, può realizzare qualsiasi funzione combinatoria a tre ingressi.

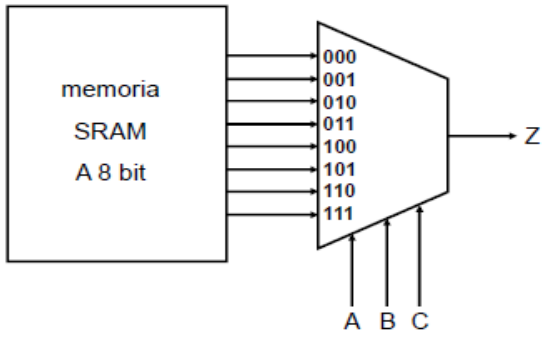

Figura 1.2.4. Struttura di una Look-Up Table (LUT).

I principali vantaggi di degli FPGA sono i seguenti:

- si compera il componente finito e lo si programma sul campo, senza ulteriori passi di fabbricazione;
- la progettazione è assistita da strumenti semi-automatici;

- terminato un progetto, la programmazione del componente richiede pochi secondi;
- il componente può essere riprogrammato.

D'altro canto, ovviamente, ci sono anche degli svantaggi rispetto ad altre tecnologie:

- utilizzo incompleto di celle ed interconnessioni;
- prestazioni ridotte rispetto ai potenziali della tecnologia;
- costi non competitivi per produzioni in grandi numeri (20000 ÷ 50000 pezzi).

L'analisi dei vantaggi e degli svantaggi conferma che la scelta della tecnologia FPGA per questo progetto è adatta. A titolo di esempio l'azienda, producendo una quantità limitata di apparecchi, non risente del maggior costo per unità rispetto ad altre soluzioni.

1.3 Il dispositivo utilizzato: Xilinx XC6SLX16

Lo FPGA utilizzato in questo progetto è uno Xilinx Spartan-6 XC6SLX16-CS324 montato su una scheda di sviluppo SP601 [6]. La scheda usata è mostrata in Figura 1.3.1, dove sono anche indicati i componenti più importanti. Una spiegazione esaustiva delle caratteristiche e delle funzionalità del dispositivo usato non rientra negli scopi di questa tesi. Per queste ragioni in questa sede ci si limita ad elencare le caratteristiche principali di questo FPGA e si rimanda al sito web del produttore per informazioni più approfondite [7].

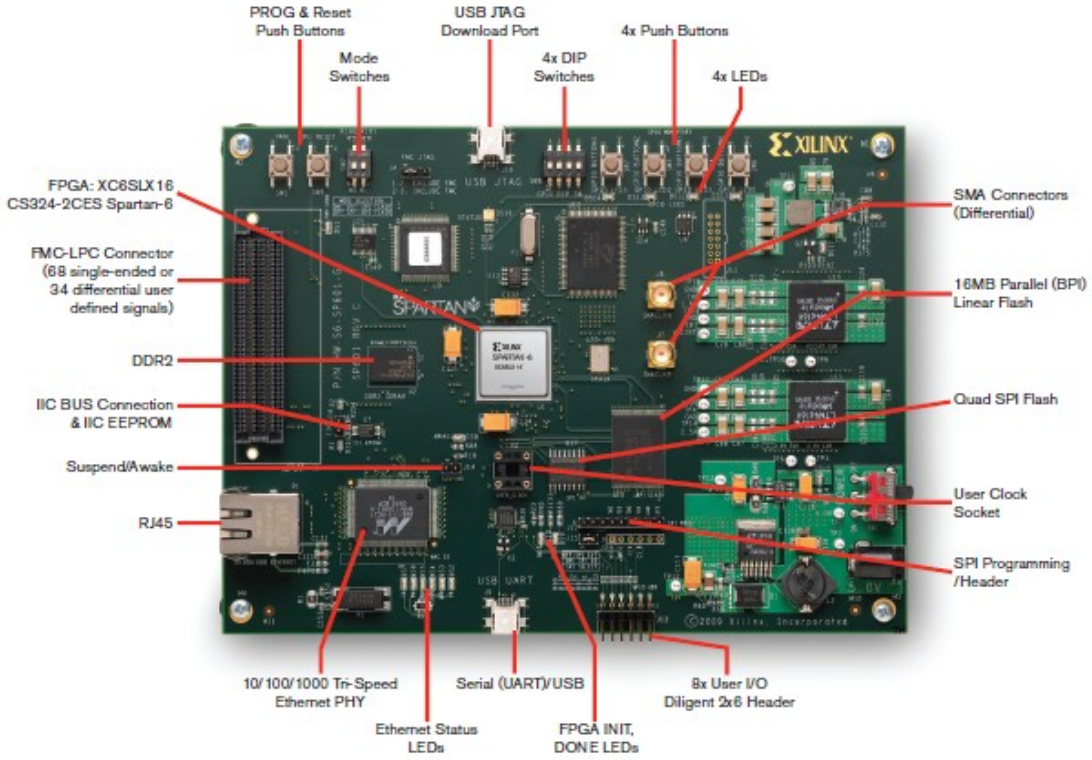

Figura 1.3.1. La scheda di sviluppo Xilinx Spartan-6 SP601.

Xilinx XC6SLX16-CS324 Spartan-6LX FPGA	
Slices	2278
Logic Cells	14579
CLB Flip-Flops	18224
Maximum Distribuited RAM (kb)	136
Block RAM (18 kb each)	32
Total Block RAM (kb)	576
Clock Management Tiles (CMT)	2
Maximum Single-Ended Pins	232
Maximum Differential Pairs	116
DSP48A1 Slices	32
Endpoint Block for PCI Express®	–
Memory Controller Blocks	2
GTP Low-Power Transceivers	–
Commercial	-1L, -2, -3, -3N
Industrial	-1L, -2, -3, -3N
Configuration Memory (Mb)	3,7

Tabella 1. Caratteristiche peculiari del *chip* FPGA montato sulla scheda Spartan-6.

1.4 Il Verilog, linguaggio di descrizione dell'hardware

Il Verilog è un linguaggio di descrizione dell'hardware (HDL, *Hardware Description Language*). Nato come linguaggio proprietario, successivamente fu reso pubblico e più tardi divenne uno standard (IEEE 1364). Fu concepito appositamente con una sintassi simile al C per favorirne la diffusione. In quanto linguaggio HDL, differisce dai linguaggi di programmazione convenzionali nell'esecuzione delle istruzioni perché potendo descrivere processi paralleli e concorrenti (come sono i segnali che viaggiano in un circuito elettronico) l'esecuzione non è strettamente lineare (se non dentro certe strutture di codice).

Il "concorrente" principale del Verilog è il VHDL (da VHSIC-HDL, *Very High Speed Integrated Circuit-HDL*), difatti il mercato, per quanto riguarda l'uso dei linguaggi HDL, è sostanzialmente diviso a metà tra Verilog e VHDL. Il VHDL è stato sviluppato per conto dell'esercito statunitense e standardizzato dalla IEEE [8]. È un linguaggio più recente ma più rigido rispetto al Verilog, e viene usato prevalentemente dall'industria Europea. Il Verilog, al contrario, è stato sviluppato da industrie elettroniche ed è diffuso soprattutto negli Stati Uniti.

A differenza dei programmi di *schematic capture* in cui si disegna il circuito da implementare, con un HDL si usa un apposito linguaggio per descrivere l'hardware del sistema. Una volta scritto il programma con un *editor* di testo esso viene sintetizzato, cioè viene tradotto nelle strutture del componente utilizzato. In pratica, viene creato un *file* che contiene i bit di configurazione da copiare nel *chip* FPGA per realizzare la stessa funzione del circuito desiderato.

Tramite la creazione di un *testbench* (un particolare modulo Verilog) si possono simulare gli stimoli provenienti dal mondo esterno al modulo sotto test e ricevere segnali da quest'ultimo agendo di conseguenza, verificando così il comportamento del circuito descritto.

Un progetto Verilog consiste di una gerarchia di moduli in cui ciascuno è definito da un insieme di ingressi, uscite ed eventualmente porte bidirezionali. Internamente contiene una lista di fili e registri. I processi paralleli e sequen-

ziali ne definiscono il comportamento stabilendo la relazione tra le porte, i registri ed i fili. Le istruzioni sequenziali sono poste all'interno di un blocco *begin–end* in ordine sequenziale valido solo all'interno del blocco.

2 Il progetto sviluppato

Questo capitolo espone l'hardware elettronico che caratterizza il sistema montato nella prima versione dell'apparecchio topografo corneale Magellan Mapper di NIDEK Technologies e successivamente descrive il progetto del sistema che dovrebbe rimpiazzarlo nei nuovi prodotti. Il lavoro svolto in questo durante il tirocinio e presentato in questa tesi è parte integrante progetto.

2.1 L'hardware esistente attualmente

L'hardware preesistente a questo progetto è montato sulla versione attuale del topografo corneale Magellan Mapper. Si tratta delle due schede visibili in Figura 2.1.1. A sinistra, il lato posteriore della scheda che alloggia il sensore CCD ed il cui lato frontale è visibile in Figura 2.1.2.

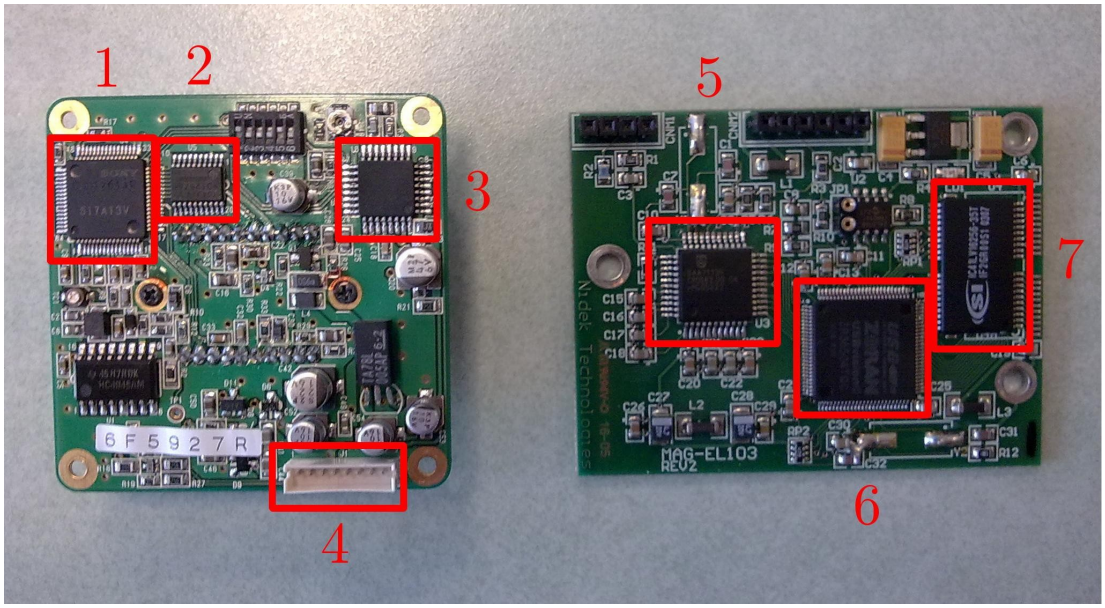

Figura 2.1.1. Schede dell'attuale topografo corneale Magellan Mapper.

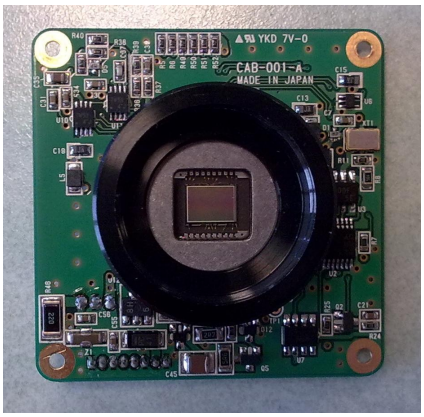

Figura 2.1.2. Scheda della telecamera dell'attuale Magellan Mapper (vista frontale).

Sul lato posteriore della scheda del sensore i componenti principali sono il controller (1, in Figura 2.1.1), il convertitore analogico/digitale (ADC, *Analog-to-Digital Converter*) (2) e lo encoder (3). Il controller gestisce l'acquisizione dei *frame* e la loro conversione in digitale tramite il convertitore analogico/digitale. I dati digitali vengono poi inviati ad un encoder che li codifica nel formato analogico NTSC (*National Television System Comitee*) [9] con risoluzione 768x494 pixel e portati in uscita al connettore (4). La seconda scheda (a destra in Figura 2.1.1) si occupa di elaborare i dati per adattarli alla trasmissione via USB 1.0. Il segnale analogico proveniente dalla scheda della telecamera viene decodificato da un decoder che ricostruisce il *frame* in formato digitale. Attraverso un bus i dati vengono quindi inviati ad un controller USB (6) che si occupa di gestirne la trasmissione rispettando l'omonimo protocollo. Nella scheda è anche presente una memoria DRAM da 4 Mbit (7) su cui il controller USB può contare.

Come commento al sistema attuale, per prima cosa si può evidenziare che lo standard NTSC è per certi aspetti (ad esempio per la risoluzione) inferiore a quello della controparte PAL che, infatti, caratterizzerà il nuovo sensore. Inoltre, la doppia conversione analogico/digitale – digitale/analogico presente nel sistema nella fase di acquisizione e di successiva codifica in formato NTSC implicano una perdita di qualità evitabile scegliendo altre soluzioni.

Infine, il sensore CCD utilizzato attualmente, visibile in Figura 2.1.2, non fornisce immagini a colori.

2.2 Descrizione del progetto

L'obiettivo del tirocinio era sviluppare delle interfacce su piattaforma FPGA in Verilog per poter interfacciare alcune periferiche al *chip* FPGA stesso.

Come già accennato, il progetto generale in cui s'inserisce questo lavoro ha lo scopo di creare un prototipo di un sistema elettronico per l'acquisizione di immagini e video che sostituisca quello attualmente presente nei prodotti dell'azienda. Inizialmente, per comodità progettuali, si ipotizza di progettarlo per il topografo corneale Magellan Mapper, ma sempre ricordando l'obiettivo ultimo che è quello di creare un sistema abbastanza adattabile da poter essere inserito col minor numero di modifiche in qualsiasi prodotto dell'azienda. Si vuole quindi migliorare, aggiornandolo, il sistema di acquisizione delle macchine. In Figura 2.2.1 è mostrato lo schema generale del progetto in cui s'inserisce il lavoro oggetto di questa relazione. Inizialmente la modalità di collegamento tra il *chip* FPGA ed il sensore CCD era diversa; era prevista un'ulteriore interfaccia (con un bus SPI, *Serial Peripheral Interface* [10]) per comandare il CCD. Durante lo sviluppo del progetto questa parte è stata però modificata, ed il nuovo schema di collegamento fornito è visibile in Figura 2.2.2. L'obiettivo che ci si è proposti è di realizzare quante più interfacce possibili nel tempo a disposizione, tenendo conto che durante lo sviluppo imprevisti, ritardi nelle consegne o difficoltà a reperire alcuni pezzi hardware possono rallentarlo.

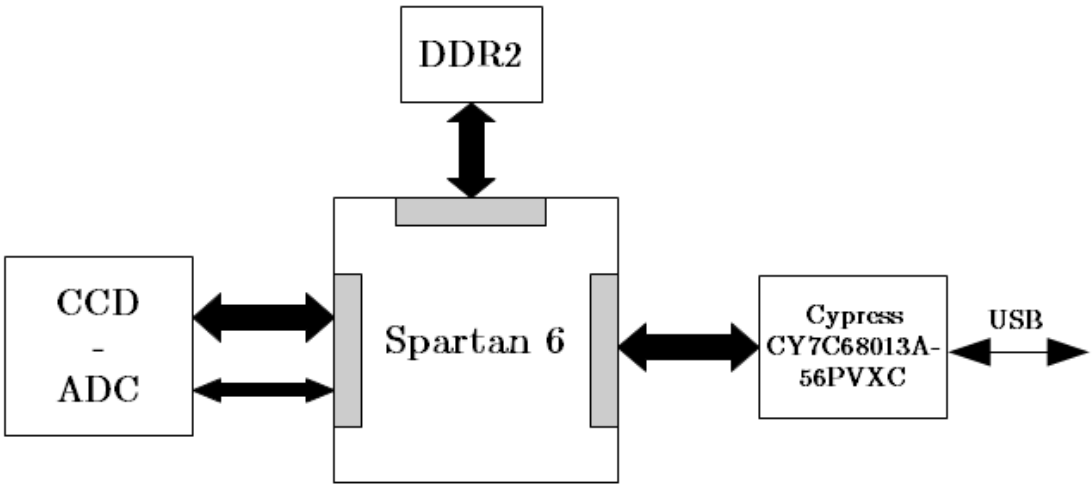

Figura 2.2.1. Schema del progetto completo.

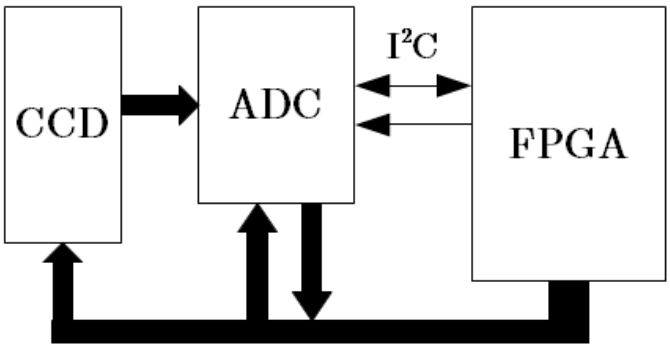

Figura 2.2.2. Schema delle connessioni tra FPGA, CCD e ADC.

Il sistema d'acquisizione è composto da un sensore CCD e da un ADC. Il CCD, in quanto tale, fornisce i dati in formato analogico; essi vengono poi convertiti in digitale dal convertitore analogico/digitale e trasferiti alla scheda Spartan-6 tramite un bus I^2C. In Figura 2.2.2 sono visibili le due linee che compongono questo tipo di bus: una linea dati bidirezionale (che però in questa applicazione serve solo a trasmettere i *frame* convertiti alla scheda di sviluppo) ed una unidirezionale per il segnale di clock. Linee aggiuntive non facenti parte di un bus particolare trasportano inoltre altri segnali di controllo verso il CCD o di controllo/dati da e verso il convertitore analogico/digitale. L'interfaccia con il sistema d'acquisizione trattata in questa relazione riguarda solo il bus I^2C. È da sottolineare che non sono noti la marca ed il modello né del sensore CCD né del convertitore. In questa relazione si supporrà per-

tanto che il convertitore abbia prestazioni conformi a quelle desiderate. Il lavoro per questa interfaccia prevede anche l'analisi delle prestazioni del bus (in via teorica, vista l'assenza di modelli specifici di componenti che poi ne implementeranno la generazione e gestione dei segnali) ed il confronto con quelle del bus USB.

Una seconda interfaccia prevede la connessione del *chip* FPGA con un microcontrollore Cypress CY7C68013A-56PVXC, il quale contiene al suo interno delle strutture gestite come code FIFO ed una CPU che tra le altre funzioni permette anche di gestire in uscita comunicazioni secondo il protocollo USB 2.0.

Infine, un'interfaccia con una memoria DDR2 dovrebbe permettere l'archiviazione temporanea dei dati per il loro successivo riutilizzo.

Per gestire tutto il sistema servirà poi un'unità “controllore” attività che verrà svolta successivamente a questo tirocinio.

3 Interfaccia con il sistema di acquisizione

In questo capitolo si tratta dell'interfaccia tra il *chip* FPGA e il sistema di acquisizione, costituito a sua volta dal convertitore analogico/digitale e del sensore CCD, e si descrivono le scelte fatte a partire dalle specifiche fornite, prevalentemente hardware. Il primo paragrafo fornisce qualche nozione teorica sui sensori CCD per presentarne la tecnologia. Nel secondo paragrafo si conduce uno studio di fattibilità che ha l'obiettivo di mettere a confronto le prestazioni in termini di banda che l'hardware a disposizione per la trasmissione via USB permette di ottenere con quelle desiderate, allo scopo di capire se sono sufficienti e di determinare i limiti del sistema. Il terzo ed ultimo paragrafo di questo capitolo presenta il bus I^2C e la soluzione in Verilog scelta per interfacciarlo con il *chip* FPGA.

3.1 Introduzione ai CCD

Il CCD (*Charge Coupled Device*) è un circuito integrato formato da una griglia (o solo da una riga) di elementi semiconduttori (*photosite*) capaci di accumulare una carica elettrica (*charge*) proporzionale all'intensità della radiazione elettromagnetica che li colpisce [11]. Questi elementi si comportano quindi come particolari condensatori. Nella struttura di un CCD c'è una regione fotoattiva (uno strato epitassiale di silicio) ed una regione di trasmissione fatta da un "registro a scorrimento" [12]. Quando un'immagine viene proiettata attraverso una lente sulla griglia fotoattiva, ogni elemento-condensato-

re si carica in base all'intensità di luce ricevuta. Mentre una linea di elementi può catturare solo una fetta di immagine, una griglia può acquisire l'intera immagine bidimensionale proiettata sul piano focale del sensore. Gli elementi sono accoppiati (*coupled*) in modo che, una volta che il sensore è stato esposto alla luce, un circuito di controllo determini il trasferimento della carica contenuta in ogni elemento a quello successivo, realizzando quindi un registro a scorrimento di carica. L'ultimo elemento deposita la carica in un amplificatore di carica, che converte la carica in una tensione elettrica [13]. Inviando al dispositivo una sequenza temporizzata di impulsi si ottiene in uscita un segnale elettrico grazie al quale è possibile ricostruire la matrice dei pixel che compongono l'immagine proiettata sul CCD. In un dispositivo digitale le sequenze di tensioni sono campionate, quantizzate e di solito salvate in una memoria; in un dispositivo analogico esse vengono trasformate in un segnale analogico che può essere poi trasmesso o, ad esempio, registrato.

Scendendo un po' più a fondo nei dettagli del principio di funzionamento si vede che gli elettroni (colorati in blu nella Figura 3.1.1) sono confinati in "serbatoi di potenziale" (in chiaro sotto ad ogni elettrodo), creati applicando una tensione positiva agli elettrodi di *gate* (G). Quando si toglie tensione ad un elettrodo il serbatoio cessa di esistere. Applicando tensione a due elettrodi vicini si crea un unico serbatoio sotto di essi che consente la distribuzione degli elettroni in tutto il suo volume. Ripetendo questa operazione nella giusta sequenza si realizza il trasferimento di carica, riuscendo a spostare la carica contenuta nei serbatoi di ogni riga di volta in volta di una posizione. Nella Figura 3.1.1, da sinistra a destra, è mostrata la sequenza con la quale si può trasferire la carica di un elemento (il primo a sinistra) al successivo (centrale).

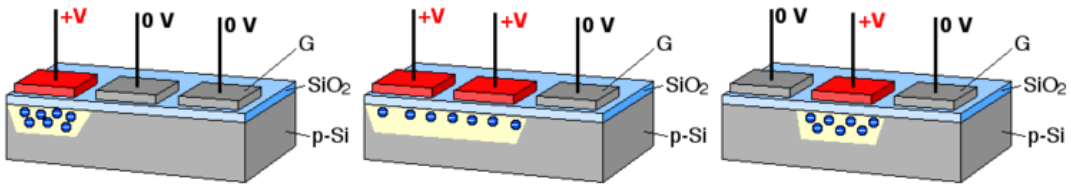

Figura 3.1.1. La sequenza di spostamento della carica in un CCD.

Sensori a colori: il filtro di Bayer

La struttura base dei sensori CCD vista finora non è in grado, da sola, di rilevare il colore (cioè la frequenza) della radiazione luminosa che incide ogni pixel; essa può rilevare solo l'intensità luminosa. I pixel, cioè, rilevano solo la luminanza della luce incidente, e non la crominanza. Per ovviare a questo problema e permettere alla griglia di un CCD di rilevare i colori base (rosso, verde e blu) ci sono diverse possibilità. In questa relazione si presenta solo quella più diffusa ed inerente al dispositivo scelto. Sopra alla griglia di fotodiodi si applica un'altra griglia con funzione di filtro, denominata con la sigla CFA (*Color Filter Array*). Ogni pixel di questa nuova griglia permette il passaggio di una sola frequenza della luce, quindi di un solo colore (Figura 3.1.2). Esistono diversi tipi di griglie, ma differiscono solo per lo schema secondo cui sono disposti i vari colori. Una delle griglie più utilizzate è il filtro di Bayer [14]. Il filtro di Bayer è una matrice di filtri di colore la cui disposizione è mostrata in Figura 3.1.3.

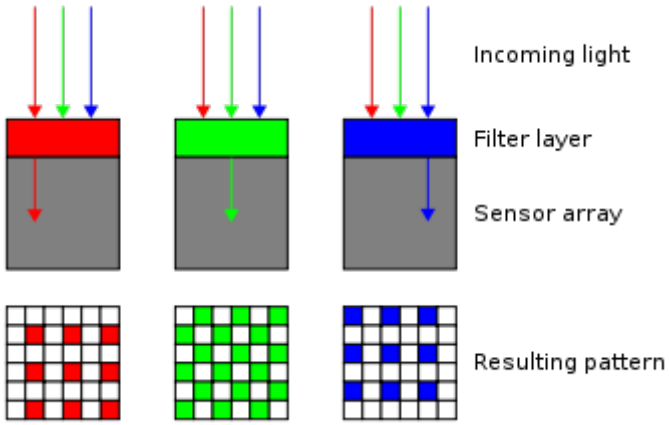

Figura 3.1.2. Il diverso comportamento dei tre filtri.

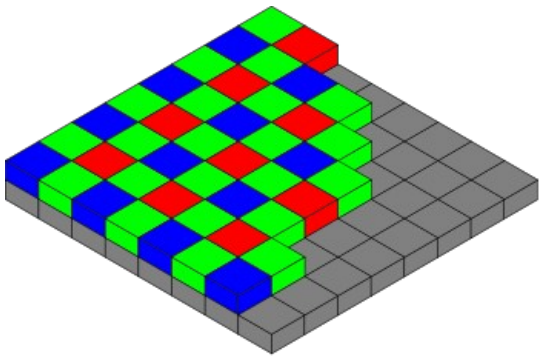

Figura 3.1.3. La disposizione dei colori in un filtro di Bayer.

Come si può vedere il filtro di Bayer ha il 50% di filtri verdi, un ulteriore 25% di filtri rossi e il restante 25% di filtri blu. Questa disparità non è casuale. Bryce Bayer, l'inventore di questo tipo di filtro, chiamò i foto-sensori verdi *luminance-sensitive elements* mentre quelli blu e quelli rossi *chrominance-sensitive elements*. Usò il doppio di elementi verdi rispetto a quelli rossi o blu per imitare la fisiologia dell'occhio umano che, in effetti, è più sensibile alla luce verde. La retina dell'occhio umano è composta di due tipi di cellule: i coni e i bastoncelli. I bastoncelli, molto più numerosi e molto più sensibili alla luce rispetto ai coni, hanno d'altro canto solo un pigmento fotosensibile a differenza dei tre (per i tre colori base) dei coni, e sono più lenti nella risposta, ma la caratteristica più importante in questo caso è la loro maggiore sensibilità alle lunghezze d'onda attorno al verde. Dato che ogni pixel di un CCD filtrato da un filtro di Bayer rileva solo uno dei tre colori, i dati di un solo pixel non possono determinare il colore esatto della luce incidente in quel punto. È necessario allora usare le informazioni di un numero maggiore di pixel per avere un'informazione più accurata sul vero colore della luce; così facendo, però, l'informazione che si ottiene si riferisce ad un'area più grande, pari alla somma delle aree dei pixel usati. Premettendo che ci si riferisce allo *output* non elaborato (o elaborato in maniera minima) dei CCD con il termine "*output* grezzo" (*raw output*), la non equità della spartizione dell'area fra i tre colori rende però necessarie delle scelte su come comporre il colore del pixel vero e proprio, quello cioè dell'immagine non grezza. Per questo esistono diversi algoritmi di demosaicizzazione[2] che vengono usati per interpolare valori completi di rosso-verde-blu per ciascun punto. Queste elaborazioni possono essere eseguite direttamente *in-camera*, producendo un'immagine in formato JPEG [15] o TIFF [16], oppure da applicazioni esterne usando il formato grezzo.

CCD vs. CMOS

Attualmente esistono due tipi di sensori in commercio: i CCD e i CMOS (*Complementary Metal-Oxide Semiconductor*). Entrambi si basano su fotodio-

2 *Algoritmi che permettono di ricostruire la rappresentazione a colori di un'immagine partendo dai dati grezzi ottenuti da un sensore che utilizza un CFA.*

di che convertono in una carica elettrica la luce ricevuta, tuttavia esistono delle differenze. Un sensore CCD è un dispositivo analogico. Le cariche elettriche immagazzinate nei vari elementi vengono convertite in tensioni un pixel alla volta man mano che vengono lette. Circuiti esterni al sensore convertono poi le tensioni in dati digitali. Un sensore CMOS è un sensore a pixel attivi costruito con il processo di fabbricazione dei semiconduttori CMOS. Ogni elemento ha il proprio amplificatore ed anche il convertitore analogico/digitale è interno al sensore, eliminando la necessità di interfacciarne uno esterno. Entrambe le tecnologie presentano vantaggi e svantaggi. Solo per citarne alcuni, la tecnologia dei CCD è più matura, ma quella CMOS in questo campo ha fatto notevoli passi avanti, ed ha dalla sua parte anche minori costi di produzione (i sensori CMOS possono essere prodotti dalle stesse fabbriche che producono altri *chip*, basta che la tecnologia sia sempre CMOS), minori consumi, maggior velocità e, potenzialmente, può essere implementata con meno componenti. Per contro, i sensori CMOS sono più sensibili alle interferenze rispetto ai CCD a causa della presenza degli innumerevoli amplificatori e convertitori implementati nella matrice dei *chip* CMOS, i parametri dei quali possono discostarsi anche di pochissimo l'uno dall'altro determinando comunque effetti non trascurabili. Nei CCD invece la conversione avviene tramite un unico *chip* dedicato, quindi questo problema non sussiste. La tendenza attuale è comunque ancora quella seguita finora, cioè quella di preferire sensori CCD per applicazioni di alta qualità.

L'azienda, al momento dello sviluppo di questo progetto, non aveva ancora deciso che tipo di sensore utilizzare, ma ha comunicato che molto probabilmente si tratterà di un CCD. In questa relazione si considererà quindi che le specifiche di progetto impongano l'uso di un CCD di non meglio specificate caratteristiche, se non che avrà una risoluzione di circa 5 megapixel.

3.2 Studio di fattibilità

Questo paragrafo ha l'obiettivo di analizzare le prestazioni ottenibili sotto certe ipotesi dall'hardware utilizzato. Le ipotesi per questo studio sono le seguenti:

- la risoluzione del CCD è circa 5 megapixel. Precisamente, è stato consigliato di considerare la risoluzione di 2,592 pixel orizzontali per 1,944 pixel verticali;
- il microcontrollore utilizzato, secondo alcuni test eseguiti dalla casa madre [17], configurato per operare uno streaming di dati raggiunge un *throughput*[3] di circa 43,8 MB/s.

Le variabili che si vogliono considerare sono invece:

- la **profondità di colore**: il colore associato ad ogni bit può essere rappresentato con un numero di bit. Tanto più alto è il numero di bit tante più tonalità di colore si riescono a rappresentare. Per questo studio si sono considerate profondità di colore di 8 bit (corrispondente ai 256 livelli) per immagini in tonalità di grigio, e di 16 e 24 bit per immagini a colori;
- il ***frame rate***: un *frame* corrisponde ad una singola immagine acquisita dal sensore. L'azienda ha richiesto di considerare tre valori di *frame rate*. Il minimo accettato è 25 *frame*/s, il valore desiderato è 40 *frame*/s mentre quello ottimo è 50 *frame*/s;
- Il ***binning***: per *binning* (rappresentato in Figura 3.2.1) s'intende il raggruppamento delle informazioni di più pixel per formarne uno unico. Il pixel risultante ha dimensioni pari alla somma delle dimensioni dei pixel da cui è stato creato ma ha le loro stesse caratteristiche (ad esempio la medesima profondità di colore). È evidente quindi che eseguire un *binning* comporta una perdita d'informazione, ma allo stesso tempo consente di diminuire il volume (e quindi la banda) dei dati da trattare. Effettuare un *binning*

3 *Termine che indica la banda effettiva misurata in un certo periodo, tenuto conto del flusso dei dati e del percorso di instradamento.*

3x3 significa “fondere” una matrice di 3x3 pixel; in modo analogo si possono eseguire, per esempio, *binning* 4x4 o 2x2. Un *binning* 1x1 non ha alcun effetto, poiché in pratica non raggruppa alcun bit. Come ultima nota, si segnala che un *binning* 2x2, 4x4, 8x8, eccetera viene di solito eseguito quando le dimensioni del *frame*, espresse come numero di bit, sono numeri divisibili per due. Altrimenti, alcune righe o colonne del *frame* potrebbero non avere dimensioni sufficienti per eseguire un *binning* di pari ampiezza eseguito per le altre e dovrebbero essere scartate. Ad esempio, se fosse eseguito un *binning* 3x3 ed in una delle due dimensioni del *frame* rimanessero due colonne, non sarebbe possibile effettuare un *binning* 3x3 tra di esse. In questo caso si potrebbe decidere di cambiare ampiezza del *binning* oppure di scartare le colonne rimaste.

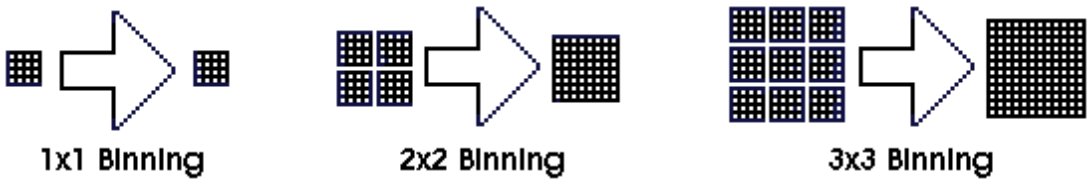

Figura 3.2.1. Esempi di *binning*.

Per calcolare la banda necessaria nei vari casi è sufficiente moltiplicare tra loro il numero di pixel che compongono un *frame*, la profondità di colore di ciascun bit, il *frame rate*, e dividere infine il risultato per il numero di pixel della matrice relativa al *binning* considerato. Ad esempio, considerando un sensore avente risoluzione 2,592x1,944 pixel, profondità di colore 8 bit, e considerando un *frame rate* di 25 *frame*/s con un *binning* 2x2 (quindi di 4 pixel) si ottiene una banda di circa 30 MB/s.

In Tabella 2 sono riassunti i calcoli eseguiti. Le celle evidenziate corrispondono a valori di banda minori del limite del microcontrollore (43,8 MB/s) e quindi raggiungibili dall'hardware considerato.

NUMERO PIXEL	PROFONDITÀ DI COLORE (bit)	FRAME RATE (*frame*/s)	BINNING	BANDA (Gbit/s)	BANDA (MB/s)
5038848	8	25	1	1,01	120,1
			4	0,25	30,0
			9	0,11	13,4
		40	1	1,61	192,2
			4	0,40	48,1
			9	0,18	21,4
		50	1	2,02	240,3
			4	0,50	60,1
			9	0,22	26,7
	16	25	1	2,02	240,3
			4	0,50	60,1
			9	0,22	26,7
		40	1	3,22	384,4
			4	0,81	96,1
			9	0,36	42,7
		50	1	4,03	480,5
			4	1,01	120,1
			9	0,45	53,4
	24	25	1	3,02	360,4
			4	0,76	90,1
			9	0,34	40,1
		40	1	4,84	576,7
			4	1,21	144,2
			9	0,54	64,1
		50	1	6,05	720,8
			4	1,51	180,2
			9	0,67	80,1

Tabella 2. Calcolo della banda richiesta dal sistema al variare dei parametri.

Come si può vedere, solo sette tra le ventisette combinazioni provate sono supportate dall'hardware considerato. Le configurazioni più interessanti per l'utilizzo specifico che se ne farà risultano essere quella con profondità di colore 8 bit, 50 *frame*/s, *binning* 3x3 e quella con profondità di colore 16 bit, 40 *frame*/s, *binning* 3x3.

3.3 Il bus I²C e l'interfaccia utilizzata

Come visto nel paragrafo 2.2, in questo progetto il *chip* FPGA comunica direttamente con il convertitore analogico/digitale attraverso un bus I²C per ricevere i dati delle immagini restituite dal sensore, mentre i dati di controllo viaggiano su linee separate. La situazione è chiarita in Figura 2.2.2. In questo paragrafo si analizza il bus I²C per verificarne le modalità operative ed i limiti, ed infine proporne una soluzione implementativa. Gli altri aspetti del sistema d'acquisizione, come ad esempio le modalità di comando del sensore e del convertitore analogico/digitale, non sono oggetto di questa relazione.

Il bus-protocollo I²C [18] (*Inter-Integrated Circuit* bus, scritto anche IIC o I2C) è un sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati che permette di collegare periferiche a bassa velocità con schede madri, sistemi *embedded* e quant'altro [19]. Creato nel 1982 da Philips Semiconductors (l'attuale NXP Semiconductors), col passare del tempo è diventato uno *standard* mondiale *de facto*, caratterizzato da successivi aggiornamenti che ne hanno via via migliorato le prestazioni. La diffusione di questo bus ha avuto un'accelerazione quando si è affermata la tendenza, da parte dei costruttori, di produrre *chip* sempre più piccoli e con meno *pin* per motivi di risparmio e/o di utilizzo in sistemi mobili. Dall'1/10/2006 il protocollo può essere liberamente usato, tuttavia ai costruttori di periferiche commercializzabili che vogliono implementarlo nei propri prodotti, avendo ogni tipo di periferica un proprio codice di riferimento è richiesto il pagamento di un diritto alla NXP Semiconductors.

Prima di iniziare a descrivere il protocollo I²C è bene precisare il significato di alcuni termini:

- **trasmettitore**: un dispositivo che trasmette dati sul bus;
- **ricevitore**: un dispositivo che riceve dati dal bus;
- ***master***: il dispositivo che può iniziare un trasferimento, genera il segnale di clock e termina il trasferimento;
- ***slave***: il dispositivo indirizzato dal *master*.

Il bus I²C mette in comunicazione almeno un *master* ed uno *slave*, ma più frequentemente sono collegati tramite un unico bus un *master* e più *slave*; è comunque possibile realizzare anche architetture *multi-master* e *multi-slave*. Il termine *multi-master* indica che più di un *master* può tentare di controllare il bus senza corrompere i dati sul bus stesso. In Figura 3.3.1 è mostrato un esempio di collegamento tra periferiche *slave* ed un unico *master*.

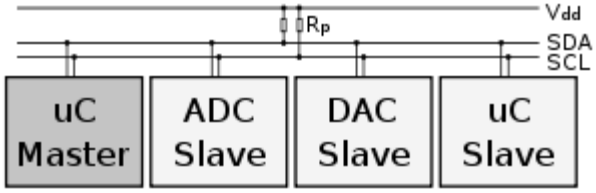

Figura 3.3.1. Esempio di periferiche collegate tramite un bus I²C.

Il bus I²C è dichiarato come un bus bifilare perché necessita di due linee di comunicazione: SDA (*Serial DAta Line*) per i dati e SCL (*Serial Clock Line*) per il clock (che caratterizza questo tipo di bus come bus sincrono). In realtà, però, c'è almeno un'altra linea (non visibile in Figura 3.3.1), talvolta chiamata impropriamente GND, che condivide il potenziale di riferimento. Oltre a queste linee normalmente ce n'è anche una di alimentazione (V_{dd}) che però può anche non essere comune a tutti i dispositivi. In tal caso, i dispositivi che tollerano la tensione di alimentazione comune possono essere collegati direttamente al bus, mentre per gli altri occorrerà inserire dei circuiti adattatori di tensione. Le tensioni di alimentazione tipiche sono +5V e +3,3V, ma sono ammesse anche tensioni più basse o più alte. In Figura 3.3.1 sono presenti dei resistori R_p detti di *pull-up*; essi sono necessari perché le linee SCL e SDA sono di tipo *open-drain* (od *open-collector* a seconda della tecnologia usata, rispettivamente MOSFET o BJT). Si ricorda che una linea o un *pin* *open-drain* sono particolari tipologie di uscite di circuiti integrati. Il segnale di uscita del circuito invece di essere portato direttamente al *pin* esterno del *chip* è applicato alla base di un MOSFET (o di un transistor NPN nel caso *open-collector*, come in Figura 3.3.2), il cui terminale di *drain* è invece collegato al piedino del *chip*. Il *source* del MOSFET (o l'emettitore del BJT) è collegato a massa [20].

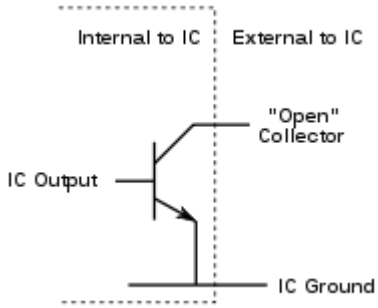

Figura 3.3.2. Schema di un *pin* "*open-collector*".

Tornando alle linee del bus I²C, quando un dispositivo collegato al bus attiva la sua uscita esso forza la linea ad assumere il livello logico basso, mentre disattivandola il livello logico della linea viene riportato alto dal resistore di *pull-up*. In questo modo non si possono avere conflitti hardware perché nessun dispositivo può forzare la linea al livello logico alto.

Il bus I²C è un bus "8-bit *oriented*", perché i pacchetti di bit che vengono letti o scritti (quindi escludendo i comandi singoli come ad esempio quelli di inizio e fine trasmissione) sono sequenze di otto bit. All'inizio di ogni comunicazione invece, almeno nelle prime versioni del protocollo, i primi sette bit inviati sono l'indirizzo del dispositivo interrogato e successivamente ne segue uno di scelta dell'operazione voluta (lettura o scrittura). Con sette bit di indirizzo si codificano centoventotto possibili indirizzi diversi (corrispondenti ai nodi, cioè ai dispositivi presenti sul bus). Di questi indirizzi però, sedici sono riservati, quindi sullo stesso bus possono essere collegati al massimo centododici dispositivi. Il numero di bit di indirizzo non è l'unico fattore che limita il massimo numero possibile di nodi, su cui influiscono anche le capacità parassite introdotte da ciascun dispositivo. Secondo le specifiche, la capacità totale presentata da SDA e da SCL deve essere al massimo 400 pF.

Per quanto concerne la modalità di comunicazione ci sono, come già accennato, due tipi di nodi: *master* e *slave*. Il nodo *master* è il dispositivo che emette il segnale di clock, lo *slave* è il nodo che si sincronizza su tale segnale senza poterlo controllare. Perché la comunicazione sul bus possa avvenire correttamente ci deve essere un solo dispositivo che in ogni istante funge da *master*; più dispositivi possono essere *master*, ma non contemporaneamente. In

fase di scrittura del *firmware* si può decidere che il nodo *master* sia fisso oppure che cambi ad ogni comunicazione; in quest'ultimo caso un'apposita procedura detta *arbitrazione* stabilisce ogni volta quale dei nodi deve essere il *master*. Ogni nodo, in generale, può quindi operare secondo uno di questi modi:

- *master* trasmittente: emette il segnale di clock ed invia dati agli *slave*;
- *master* ricevente: emette il clock e riceve dati dagli *slave*;
- *slave* trasmittente: non controlla il clock ed invia dati al *master*;
- *slave* ricevente: non controlla il clock e riceve dati dal *master*.

In Figura 3.3.3 è riportato un esempio di comunicazione sul bus I^2C. In questo caso il *master* scrive i byte 0x50 e 0x0F in una memoria EEPROM[4] che è il dispositivo avente indirizzo 0x51. La comunicazione inizia ovviamente per iniziativa del *master*, il quale lancia il comando di START previsto dal protocollo, cioè effettua una transizione alto→basso sulla linea SDA mentre quella SCL è a livello alto. In questo modo esso informa gli *slave* che una comunicazione sta avendo inizio. I sette bit che seguono sono l'indirizzo dello *slave* che il *master* vuole interrogare, che in questo caso è 0x51. Gli indirizzi, così come i dati, sono trasmessi iniziando dal bit più significativo. L'ottavo bit identifica la richiesta del *master*: se è uno "0" il *master* vuole scrivere sul dispositivo, se è "1" esso vuole scrivere. A questo punto il protocollo prevede che lo *slave* debba rispondere con un segnale ACK (*acknowledge*, o ACQ – *acquired*), e deve farlo dopo ogni byte ricevuto. In questo caso, visto che sono stati trasmessi un indirizzo ed una richiesta di operazione, la risposta dello *slave* serve ad informare il *master* che il dispositivo interrogato esiste ed ha ricevuto la richiesta. Un segnale è un ACK quando la linea SDA viene portata a livello basso mentre quella SCL è a livello alto. Inizia quindi un nuovo ciclo di impulsi di clock A conferma di ricezione avvenuta seguono otto bit di dati (0x50 nell'esempio) ed un ulteriore ACK da parte dello *slave*, che in questo caso serve per comunicare al *master* che il byte è stato ricevuto e che un altro

4 *Electrically Erasable Programmable Read-Only Memory: tipo di memoria non volatile in cui le operazioni di scrittura, cancellazione e riscrittura vengono eseguite elettricamente.*

ne può essere inviato. Lo *slave* "rilascia" quindi il bus (linea SDA), che quindi va a livello logico alto per effetto del resistore di *pull-up*. Al successivo impulso di clock inizia il trasferimento di un nuovo byte, seguito infine da un altro ACK in caso di successo. Il segnale di ACK è definito dalle specifiche come segue: il trasmettitore rilascia la linea SDA durante l'impulso di clock di *acknowledge* così il ricevitore può porre la linea SDA a livello basso ed essa rimane stabile durante la parte alta di tale impulso di clock (devono essere rispettati dei tempi di *set-up* e di *hold*, ma per questo si rimanda alle specifiche [21]). Se invece l'operazione richiesta non va a buon fine per qualsiasi motivo, deve essere inviato un segnale di NACK (*not acknowledge*) invece di quello di ACK. Un segnale di NACK si ha quando la linea SDA rimane alta durante il nono ciclo di clock, ed in questo caso il *master* può generare un segnale di STOP per abortire il trasferimento oppure generarne uno di START per iniziarne uno nuovo. Ci sono cinque condizioni che portano alla generazione di un NACK:

1. Nessun ricevitore con l'indirizzo indicato è presente sul bus, quindi non c'è nessun dispositivo che risponde con un ACK.
2. Il ricevitore non può ricevere o trasmettere perché sta eseguendo qualche compito *real-time* e non è quindi pronto per iniziare una comunicazione con il *master*.
3. Durante il trasferimento al ricevitore arrivano dati o comandi che non capisce.
4. Durante il trasferimento il ricevitore non può più accettare altri byte.
5. Un *master*-ricevitore necessita di segnalare la fine del trasferimento ad uno *slave*-trasmettitore.

Infine, sempre in riferimento all'esempio di Figura 3.3.3, quando il *master* vuole terminare la comunicazione invia un segnale di STOP, definito come una transizione basso→alto sulla linea SDA mentre SCL è a livello alto.

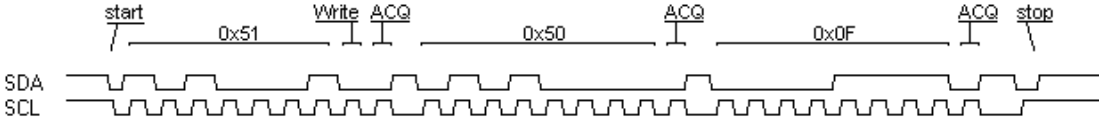

Figura 3.3.3. Esempio di scrittura su una EEPROM attraverso un bus I²C.

In generale, se un *master* vuole scrivere ripetutamente su uno *slave* deve inoltrare un ACK dopo ogni byte spedito. Se invece esso vuole ricevere ripetutamente più byte da uno *slave* deve inoltrare un ACK dopo ogni byte tranne dopo l'ultimo; a quel punto il *master* può porre fine alla trasmissione con uno STOP o mantenere il controllo del bus iniziandone un'altra con uno START.

In Figura 3.3.4 sono mostrate le condizioni di START e di STOP.

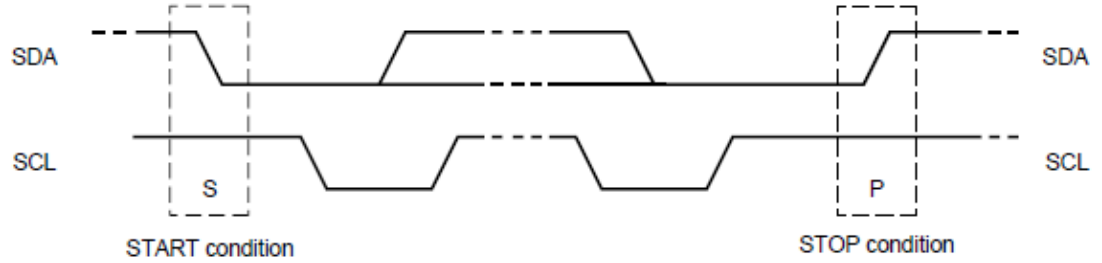

Figura 3.3.4. Condizioni di START e di STOP in un bus I²C.

I dati sulla linea SDA devono essere stabili durante la parte alta del periodo del clock. Il livello logico della linea può cambiare solo quando il segnale di clock su SCL è basso (si veda). Quindi, viene generato un ciclo di clock per ogni bit del dato da trasferire.

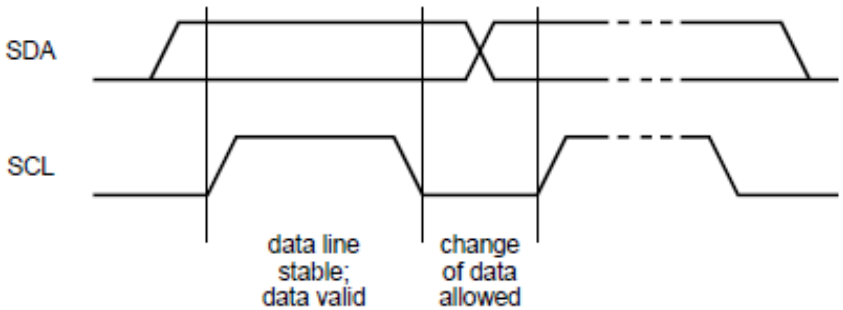

Figura 3.3.5. Validità dei dati sulla linea SDA rispetto al clock.

Nel corso delle varie revisioni sono state introdotte quattro modalità di funzionamento caratterizzate da altrettante velocità di trasmissione dati. Nell'ultima revisione (19 giugno 2007, [21]) sono definite le seguenti modalità:

- ***Standard mode***: fino a 100 kbit/s;
- ***Fast-mode***: fino a 400 kbit/s;
- ***Fast-mode plus (Fm+)***: fino a 1 Mbit/s;
- ***High-speed mode***: fino a 3,4 Mbit/s.

Inoltre, anche le possibilità di indirizzamento di dispositivi sono state aumentate, permettendo un indirizzamento a 10 bit.

Passando ora all'interfaccia in Verilog, essa si colloca come mostrato molto sinteticamente in Figura 3.3.6. In questa figura si sono considerati come elementi di memoria registri a 8 bit per la memorizzazione ed il successivo trasferimento verso altri circuiti. In questo progetto, l'interfaccia sarà comandata da un blocco "controllore" non presente in questa relazione come già spiegato nel paragrafo 2.2, e non visibile nella figura.

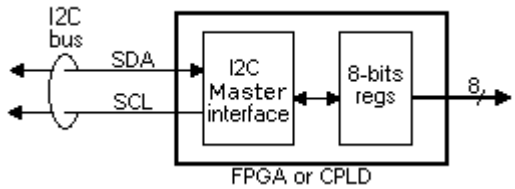

Figura 3.3.6. Esempio di interfacciamento del bus I^2C con un FPGA.

Tra gli svantaggi principali del bus I^2C c'è sicuramente la lentezza del trasferimento dati, soprattutto al giorno d'oggi in cui la richiesta di prestazioni sempre più spinte aumenta ogni giorno. Un altro punto debole di questo sistema di comunicazione è l'assegnamento degli indirizzi degli *slave* [22]; i bit di indirizzamento sono troppo pochi per prevenire conflitti di indirizzi tra le migliaia di disponibili.

Tra i vantaggi, il bus I^2C può sicuramente annoverare l'esiguo numero di linee fisiche necessarie per collegare i dispositivi, normalmente tre o quattro linee (SDA, SCL, GND e V_{dd}). Inoltre, la semplicità di funzionamento del pro-

tocollo e la considerevole diffusione ne fanno la scelta ideale per sistemi che non hanno particolari esigenze in termini di velocità di trasferimento dati.

Visti i dati riassunti in Tabella 2 (paragrafo 3.2), e dato che la massima velocità ottenibile dal bus I²C è 3,4 Mbit/s (*High-speed mode*), sembra che la scelta di un bus I²C imposta dalle specifiche per l'analisi rappresenti il "collo di bottiglia" del sistema. Se la destinazione d'uso del sistema fosse acquisire singole fotografie queste limitazioni sarebbero meno critiche, ma l'acquisizione del flusso di dati prodotto da una registrazione video scegliendo un bus con queste prestazioni sarebbe sicuramente insostenibile. Bisogna considerare però che i limiti del protocollo I²C sono forniti anche in base alla capacità parassita totale presente sul bus, che aumenta al numero di dispositivi collegati ed alla lunghezza del bus. Probabilmente in questo caso, essendo il sistema di acquisizione e il *chip* FPGA molto vicini ed essendo gli unici dispositivi sul bus, la capacità parassita totale potrebbe avere valori più bassi e permettere quindi velocità di trasferimento dati più alte. Le informazioni a disposizione sono però insufficienti per calcolare una stima. Nell'ottica del futuro utilizzo del sistema in cui si inserisce questo lavoro, cioè non solo l'acquisizione di singole immagini ma anche di video a colori, con le informazioni attuali è possibile che si debba scartare la soluzione ipotizzata a favore di una più performante. Tuttavia, per futuri test si propone una soluzione per l'interfaccia con il bus I²C.

La popolarità di questo sistema di comunicazione, la sua semplicità e, da qualche anno, l'assenza di diritti da pagare hanno favorito lo studio e lo sviluppo di molte implementazioni diverse sia nel linguaggio con cui sono state create sia per la piattaforma di destinazione. Con queste premesse è logico cercare una soluzione già creata che realizzi l'interfaccia. Dopo una ricerca in internet si è scelto un progetto [23] ospitato sul sito *opencores.org* [24], un sito web ed una fondazione la cui missione è rendere disponibili gratuitamente e documentati quanti più progetti *open-core* possibile. I progetti sono rilasciati

sotto una licenza per hardware modellata sulla LGPL (*Lesser General Public License* [25]) per software.

Il progetto scelto fornisce un'interfaccia tra un *master* compatibile con un bus Wishbone [26] ed un bus I²C. Un bus Wishbone è un bus *open-source* che ha lo scopo di mettere in comunicazione parti diverse di un circuito integrato. Non fornendo specifiche di tipo elettrico, questo bus è definito come un bus logico; piuttosto, le specifiche sono date in termini di segnali, cicli di clock, e livelli alti e bassi. Questa caratteristica è intenzionale, difatti il bus Wishbone è pensato per permettere ai progettisti di combinare progetti diversi scritti in Verilog, VHDL od altri linguaggi di descrizione dell'hardware.

La Figura 3.3.7 illustra l'architettura di questo progetto e di seguito ne sono elencate le caratteristiche principali:

- compatibile con il bus standard I²C di Philips;
- possibilità di operare in modalità multi-*master*;
- temporizzazione programmabile via software;
- "*clock stretching*" e generazione dello stato d'attesa;
- rilevamento e generazione di segnali Start e Stop ripetuti;
- supporto all'indirizzamento a 7 e 10 bit.

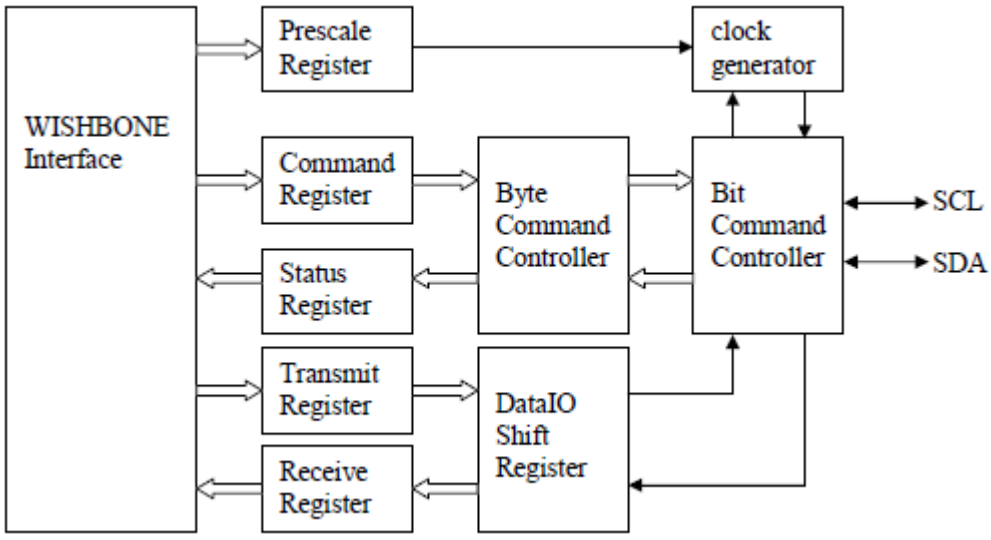

Figura 3.3.7. Architettura interna del progetto scelto (I²C Master Core).

Il progetto è costruito attorno a quattro blocchi principali: il *Clock Generator*, il *Byte Command Controller*, il *Bit Command Controller* ed il *Data-IO Shift Register*. Tutti gli altri blocchi vengono usati per l'interfacciamento o per memorizzare dati temporanei. In Tabella 3 è riportata l'occupazione delle risorse utilizzate dall'interfaccia dopo l'implementazione del progetto.

Non è stato possibile eseguire il test dell'interfaccia su scheda sia per mancanza dell'hardware che costituisce il sistema d'acquisizione (il sensore CCD ed il convertitore analogico/digitale) sia per insufficienza di tempo durante lo svolgimento di questo tirocinio breve.

Device Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	127	18,224	1%
Number used as Flip Flops	127		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	165	9,112	1%
Number used as logic	165	9,112	1%
Number using O6 output only	117		
Number using O5 output only	1		
Number using O5 and O6	47		
Number used as ROM	0		
Number used as Memory	0	2,176	0%
Number of occupied Slices	61	2,278	2%
Number of LUT Flip Flop pairs used	178		
Number with an unused Flip Flop	66	178	37%
Number with an unused LUT	13	178	7%
Number of fully used LUT-FF pairs	99	178	55%
Number of unique control sets	10		
Number of slice register sites lost to control set restrictions	25	18,224	1%
Number of bonded IOBs	33	232	14%
Number of RAMB16BWERs	0	32	0%
Number of RAMB8BWERs	0	64	0%

Device Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%
Number of BUFG/BUFGMUXs	1	16	6%
Number used as BUFGs	1		
Number used as BUFGMUX	0		
Number of DCM/DCM_CLKGENs	0	4	0%
Number of ILOGIC2/ISERDES2s	0	248	0%
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	248	0%
Number of OLOGIC2/OSERDES2s	0	248	0%
Number of BSCANs	0	4	0%
Number of BUFHs	0	128	0%
Number of BUFPLLs	0	8	0%
Number of BUFPLL_MCBs	0	4	0%
Number of DSP48A1s	0	32	0%
Number of ICAPs	0	1	0%
Number of MCBs	0	2	0%
Number of PCILOGICSEs	0	2	0%
Number of PLL_ADVs	0	2	0%
Number of PMVs	0	1	0%
Number of STARTUPs	0	1	0%
Number of SUSPEND_SYNCs	0	1	0%
Average Fanout of Non-Clock Nets	4.09		

Tabella 3. Occupazione delle risorse del progetto I²C Master Core.

4 Interfaccia con il bus USB

Questo capitolo espone l'interfaccia con il bus USB che sfrutta una coda FIFO presente all'interno del microcontrollore Cypress CY7C68013A, il quale a sua volta gestisce la comunicazione via porta USB con un qualsiasi dispositivo che supporti il relativo protocollo, tipicamente un calcolatore. Viene poi presentato e descritto il collegamento dell'intero sistema, scheda per scheda, dalla Spartan-6 al calcolatore attraverso una scheda dedicata per il microcontrollore. Infine, sono riportate le simulazioni del comportamento dell'interfaccia e il test finale eseguito su scheda.

4.1 Scopo e struttura dell'interfaccia

Lo scopo di questa interfaccia è permettere il collegamento e la comunicazione tra il *chip* FPGA e il microcontrollore Cypress CY7C68013A-56P-VXC, attraverso un bus di comunicazione. Il microcontrollore, descritto più in dettaglio nel paragrafo 4.1.2, gestisce poi la comunicazione via porta USB con un qualsiasi dispositivo dotato di tale porta. L'ambito del progetto nel quale questo circuito andrà ad operare prevede che la porta USB sia collegata ad un calcolatore per scambiare dati tra esso e la macchina che monterà il circuito stesso. Al momento della progettazione del circuito oggetto di questa tesi il tipo di questi dati è sconosciuto; potrebbe trattarsi, ad esempio, di segnali di controllo dell'apparecchio inviati da calcolatore, di un'immagine acquisita dall'apparecchio e trasmessa per essere elaborata o stoccata, o di qualsiasi altro dato. In ogni caso, comunque, il tipo di dati non influenza la progettazione dell'interfaccia.

L'interfaccia è un modulo scritto in Verilog e interno al *chip* FPGA. Essa verrà usata (e quindi comandata) da un circuito "controllore" che deciderà che azione richiedere all'interfaccia (lettura o scrittura) e che dati ricevere o presentare sul bus. Questo circuito non rientra negli scopi di questo progetto né quindi in questa relazione, essendo un elemento decisionale e non un'interfaccia. A titolo di esempio, il circuito in questione potrebbe essere un *chip* esterno a quello FPGA oppure un processore MicroBlaze™ interno ad esso [27]. In ogni caso, al momento della consegna delle specifiche non era noto di che tipo fosse questo circuito, né quindi se fosse asincrono o sincrono. A causa di questa mancanza di informazione le specifiche fornite impongono di considerare sincrono (con il clock deciso successivamente in sede di progetto) il dominio che riguarda le comunicazioni tra l'interfaccia ed il microcontrollore, mentre di considerare "asincrono" il dominio tra il resto dello FPGA (il circuito a monte dell'interfaccia che la comanda) e l'interfaccia stessa. Questo, in sostanza, per evidenziare che i segnali entranti nell'interfaccia come nEN e rW potrebbero presentarsi (o commutare) in qualsiasi momento e non necessariamente in sincronia col clock dell'interfaccia. Tuttavia, essi vengono comunque campionati al primo fronte positivo del clock che temporizza l'interfaccia. La Figura 4.1.1 illustra i segnali con cui l'interfaccia comunica con il resto dello FPGA. Questo schema costituisce una specifica di progetto fornita dall'azienda.

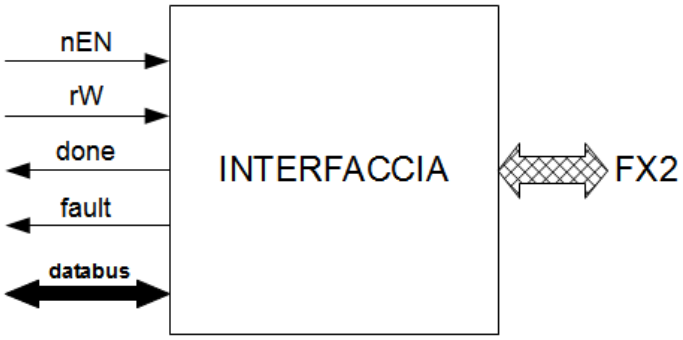

Figura 4.1.1. Schema dell'interfaccia secondo le specifiche.

Le specifiche indicano chiaramente i segnali che l'interfaccia deve possedere verso il lato FPGA, che concretamente verrà utilizzato dallo stesso *chip*

FPGA tramite qualche suo modulo; i segnali per l'altro lato, invece, sono già completamente determinati dallo FX2. I segnali richiesti sono descritti dalle specifiche iniziali in questo modo:

- **nEN**: è il segnale di *enable*. Quando è alto l'interfaccia deve porre in alta impedenza i due bus bidirezionali: databus e il bus collegato allo FX2 (non visibile esplicitamente in figura). La "n" davanti al nome evidenzia la natura attiva bassa del segnale. È compito del circuito esterno mantenere a livello basso nEN finché non ottiene una risposta sull'esito dell'operazione da esso;
- **rW**: è il segnale che indica all'interfaccia se deve leggere o scrivere sulla FIFO. Quando vale 0 è richiesta una scrittura sulla FIFO, quando vale 1 una lettura da essa. Il tipo di operazione implica anche una precisa gestione dei bus bidirezionali; ad esempio, se è richiesta una scrittura, l'interfaccia non deve impegnare il bus, perché il dato deve essere presentato dall'esterno;
- **done**: questo segnale attivo alto comunica che l'operazione richiesta è andata a buon fine;
- **fault**: anche questo segnale è attivo alto, e ha lo scopo di indicare che l'operazione richiesta ha incontrato un problema ed è quindi fallita. Questo può succedere quando si vuole scrivere su una coda piena o leggere su una coda vuota;
- **databus**: è il bus bidirezionale che collega l'interfaccia al resto dello FPGA. Fornisce all'interfaccia i dati da scrivere e offre in uscita i dati letti.

Ulteriori specifiche precisano quanto segue:

- sia in caso di riuscita dell'operazione richiesta che di fallimento, l'interfaccia deve essere disabilitata dall'esterno prima di poter operare nuovamente;
- il segnale **pktend** (si veda il paragrafo 4.1.2) deve avere valore pari alla negazione di done, che equivale a non utilizzarlo pur prevedendolo per un possibile utilizzo futuro.

4.1.1 Cenni al concetto FIFO

Il termine *FIFO (First In First Out)* ed esprime il concetto di una coda; una struttura, cioè, in cui il primo elemento ad entrare è anche il primo ad uscire. Un modello esemplificativo classico è la coda che si può avere ad una cassa di un supermercato: il primo cliente che arriva viene immediatamente servito (esce dalla coda), mentre l'ultimo deve aspettare il suo turno. Questo sistema si contrappone alla modalità *LIFO (Last In First Out)*, in cui il primo elemento ad uscire è l'ultimo arrivato. Il corrispondente esempio classico è una pila di libri. Immaginando che l'unico elemento accessibile sia quello in cima ad essa: l'ultimo ad essere inserito (sopra alla pila) è infatti anche il primo disponibile ad essere estratto.

Il concetto di coda FIFO in elettronica è utilizzato, ad esempio, per poter accoppiare due sistemi con domini di clock diversi. Concretamente, si gestiscono delle memorie con questa logica, utilizzandole come *buffer* tra due periferiche a diverse velocità.

4.1.2 Il microcontrollore Cypress CY7C68013A

Questo microcontrollore, prodotto da *Cypress Semiconductors Corporation*, è un controllore USB programmabile ad alta velocità. Il dispositivo, nel seguito talvolta nominato con la sigla FX2, fa parte della famiglia EZ-USB FX2LP™, che caratterizza microcontrollori a bassa potenza per gestire periferiche collegabili via bus di comunicazione. La scelta di questo modello di microcontrollore è stata una specifica dell'azienda presso la quale è stato svolto questo lavoro, in quanto già disponibile oltre che adatto all'impiego.

Nel seguito viene fornita una presentazione generale di tale dispositivo, focalizzando successivamente l'attenzione su alcune caratteristiche di interesse per questo progetto.

Il modello usato è alloggiato in un *package* a 56 piedini di tipo *SSOP (Shrink Small Outline* Package[5]*)*, la cui piedinatura è mostrata in Figura 4.1.2. L'insieme dei *pin* consta, tra gli altri, di ventiquattro piedini generici di I/O, divisi nelle porte A, B e D. Sedici di questi *pin* possono essere configurati come interfaccia a 16 bit verso le code FIFO interne di pari ampiezza. Il *chip* con 56 *pin* ha le seguenti caratteristiche principali:

- tre porte a 8 bit: PORTA, PORTB e PORTC;
- un bus compatibile I^2C;
- una *General Programmable Interface (GPIF)* a 8 o 16 bit;
- quattro FIFO da 8 o 16 bit in modalità *slave*.

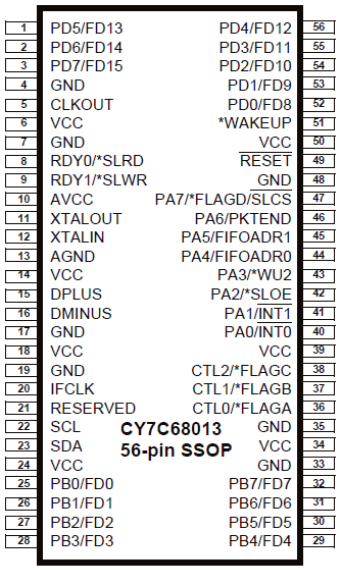

Figura 4.1.2. Piedinatura del Cypress EZ-USB FX2LP™ CY7C68013 SSOP

L'architettura del CY7C68013A, schematizzata sinteticamente in Figura 4.1.3 e più in dettaglio in Figura 4.1.4, evidenzia i blocchi principali di cui è composto il dispositivo. Si può notare la presenza del connettore USB, di un bus dati bidirezionale da 16 fili, delle linee di controllo e comunicazione da e per l'unità di elaborazione centrale e dei principali blocchi logici.

5 un tipo di involucro per *microchip* a montaggio superficiale con *pin* ad ala di gabbiano distanti 0,025" (0,635 mm). Per ulteriori informazioni sul package SSOP si veda [28]. Per informazioni su altri tipi di package si veda [29].

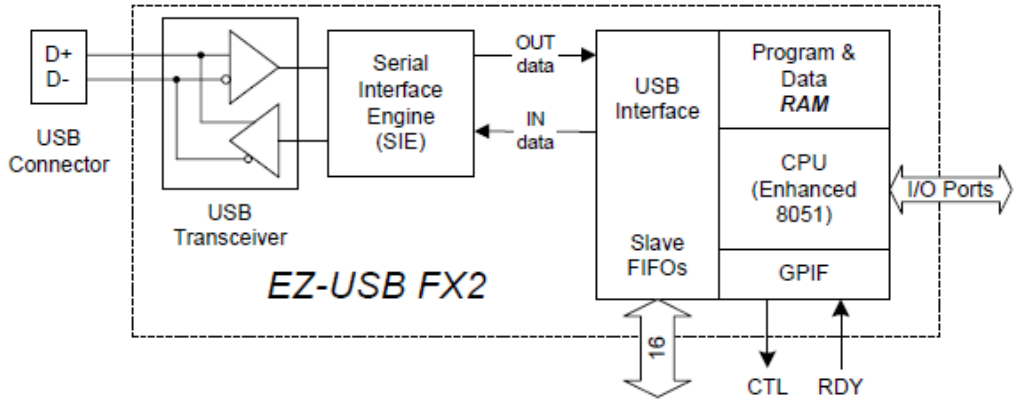

Figura 4.1.3. Architettura semplificata del CY7C68013A – package con 56 *pin*.

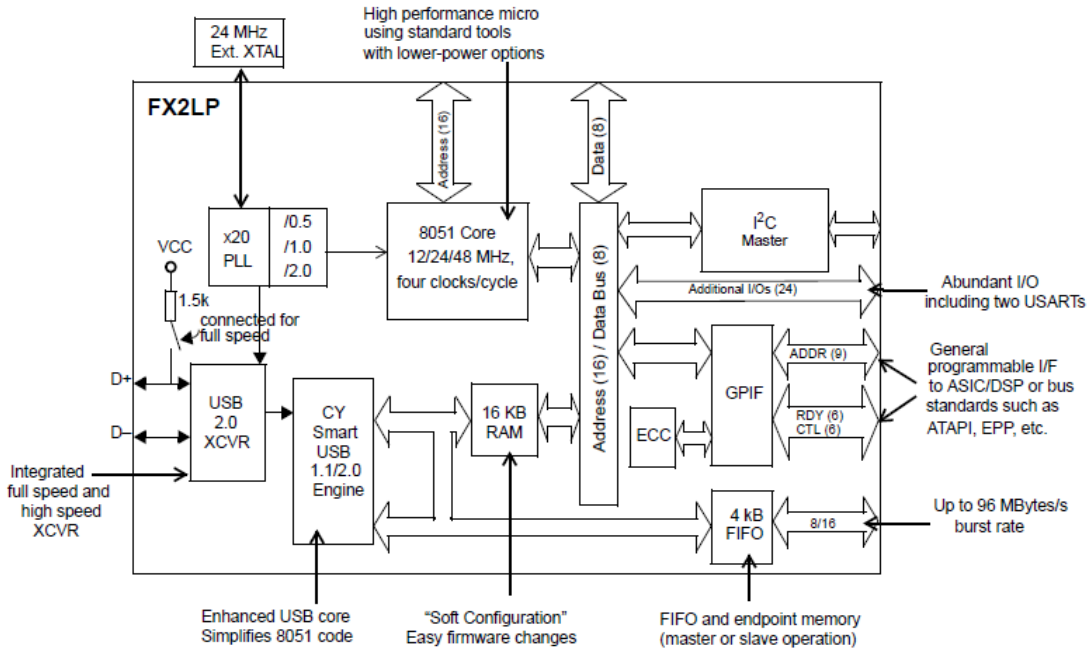

Figura 4.1.4. Architettura dettagliata del CY7C68013A – package con 56 *pin*.

Il microcontrollore racchiude in un singolo circuito integrato tutto l'hardware necessario per un'interfaccia ad una periferica USB.

Come illustrato in Figura 4.1.3, un ricetrasmettitore mette in comunicazione i *pin* D+ e D- del bus USB con la *SIE (Serial Interface Engine)*; quest'ultima codifica e decodifica i dati seriali ed esegue altri compiti richiesti dal protocollo USB. Inoltre, la SIE trasferisce dati paralleli da e verso l'interfaccia USB. Nei dispositivi della famiglia FX2 la SIE può operare a due velocità:

Full Speed (12 Mbit/s) o *High Speed* (480 Mbit/s). Il CY7C68013A, concordemente con la sua compatibilità USB 2.0, rientra nel secondo caso. Da qui la denominazione completa del microcontrollore: *EZ-USB FX2LP™ USB Microcontroller – High Speed USB Peripheral Controller*.

Lo FX2 contiene quattro strutture gestite come code FIFO, corrispondenti agli *endpoint* EP2, EP4, EP6 ed EP8 (si veda Figura 4.1.5). Questo permette al dispositivo di gestire fino a quattro periferiche USB. Per questo progetto c'è una sola periferica USB, quindi verrà usato un solo *endpoint*.

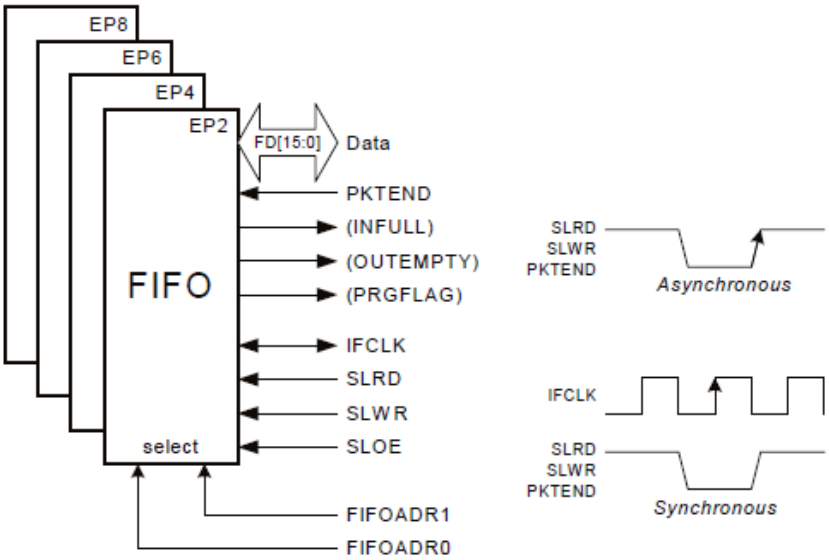

Figura 4.1.5. Le FIFO presenti nello FX2 in modalità *slave*.

Dalla Figura 4.1.5 si evince tra l'altro che lo FX2 può operare sia in modalità asincrona che sincrona. Nel primo caso l'aggiornamento dei dati e l'azione richiesta avvengono in corrispondenza del fronte positivo del segnale interessato, nel secondo mentre tale segnale è abilitato (basso) ma in corrispondenza del fronte positivo del clock.

La CPU è una 8051 che usa una memoria RAM interna per immagazzinare i dati e il programma. Il ruolo della CPU in questo sistema è duplice:

- implementa il protocollo USB ad alto livello servendo le richieste dello *host* che riguardano lo *endpoint* selezionato;

- è disponibile per un uso generico, ovviamente previa opportuna programmazione.

Quest'ultima caratteristica è importante perché amplia le possibilità del microcontrollore ed apre quindi le porte a numerosi altri scenari di utilizzo.

In Figura 4.1.6 è illustrato lo schema per il collegamento del microcontrollore con il *chip* FPGA. Il dispositivo può essere configurato in modalità *master* o *slave*, e in questo progetto è stato scelto di configurarlo in modalità *slave*. È lo FPGA, infatti, l'iniziatore delle trasmissioni dati, quindi è ragionevole configurarlo in modalità *master*.

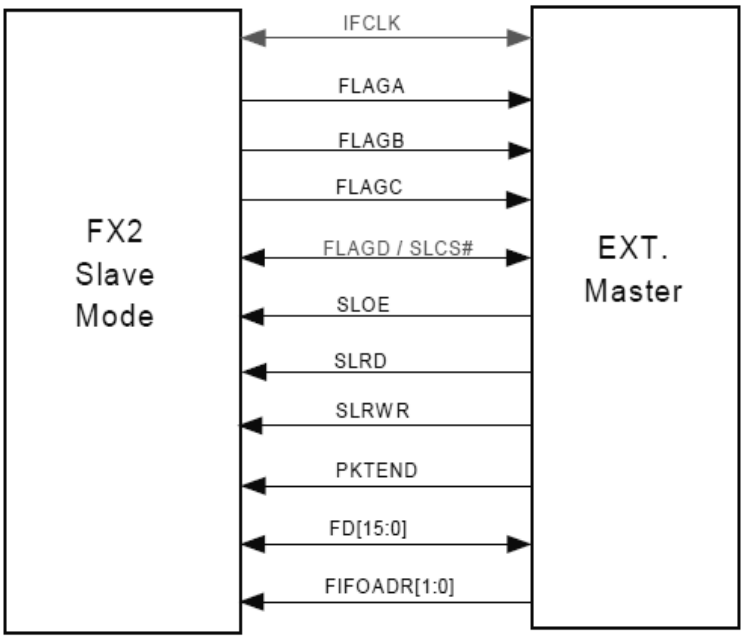

Figura 4.1.6. *Pin* dell'interfaccia FX2-FPGA (*slave mode*).

La programmazione del dispositivo attraverso la scrittura del *firmware* è necessaria in quanto non vi è un'unica modalità di funzionamento. Alcuni *pin*, infatti, possono essere usati per diversi scopi; tali *pin*, ad esempio, possono essere usati per la programmazione della CPU interna e successivamente diventare *flag*[6] o *pin* per segnali di dati.

Nel seguito è descritta la funzione dei segnali dell'interfaccia:

6 *Particolari pin/segnali che indicano lo stato dei relativi registri (in questo caso delle code FIFO).*

- **IFCLK**: è il segnale di clock e il nome sta per *"Interface clock"*. Come si evince dalla doppia freccia, il clock può essere generato dal chip FPGA e inviato allo FX2 o viceversa. Per questo lavoro è stato scelto di usare il segnale a 24 MHz generato dallo FX2.
- **FLAGA**: *flag* che indica che la FIFO selezionata è stata impostata dall'utente in una modalità di programmazione.
- **FLAGB**: *flag* attivo basso che segnala se la FIFO correntemente in uso è piena (*full flag*).
- **FLAGC**: *flag* attivo basso che segnala se la FIFO correntemente in uso è vuota (*empty flag*).
- **FLAGD/SLCS#**: *flag* programmabile oppure *chip select*. Se impostato (via *firmware*), come *chip select*, permette alla logica esterna di abilitare o disabilitare lo FX2. Se disabilitato, lo FX2 ignora i segnali SLRD, SLRWR, SLOE e PKTEND, e il bus può essere impegnato da un altro dispositivo. Questa funzione permette quindi la condivisione di uno stesso bus su più dispositivi in modalità *slave*.
- **SLOE**: segnale attivo basso di *output enable* che attiva o disattiva il bus dati. Durante la lettura (il passaggio di dati dalla FIFO selezionata al *chip* FPGA) deve essere abilitato, quindi basso; durante la scrittura, invece, SLOE deve essere disabilitato in modo che lo FX2 "rilasci" il bus (mettendolo in stato di alta impedenza) e permetta allo FPGA di prenderne il controllo per scriverci.
- **SLRD**: in modalità sincrona, se questo segnale (anch'esso attivo basso) è affermato, ad ogni fronte positivo del clock il puntatore al dato corrente della FIFO selezionata viene incrementato di una unità. In modalità asincrona l'incremento avviene ad ogni transizione abilitato/disabilitato di SLRD (si veda Figura 4.1.5). Sostanzialmente l'affermazione di questo segnale da parte dello FPGA indica che il dato è stato letto e si può passare al dato successivo.

- **SLRWR**: il comportamento di questo segnale attivo basso è simile a quello di SLRD; i dati sul bus vengono scritti nella FIFO (e il puntatore viene incrementato) ad ogni fronte positivo del clock quando SLRWR è affermato. In modalità asincrona la scrittura e l'incremento avvengono ad ogni transizione abilitato/disabilitato di SLRWR.
- **PKTEND**: il *master* afferma questo segnale, attivo basso, per spedire un pacchetto di dati alla USB avente lunghezza diversa da quella di *default* impostata negli appositi registri tramite programmazione. Usualmente PKTEND viene utilizzato quando il *master* vuole inviare un pacchetto "corto" senza dover spedire inutilmente l'intero pacchetto di lunghezza predefinita. In questo progetto non è usato in quanto l'interesse dell'azienda nel momento dello sviluppo è di trasmettere singoli pacchetti ad ogni lettura o scrittura, quindi si è scelto di non gestire questo segnale fissandolo ad un valore "casuale" (si veda il paragrafo 4.2.3). Nel caso si volessero spedire più pacchetti ad ogni trasmissione *(burst mode)* basterebbe modificare il codice Verilog dell'interfaccia.
- **FD**: è il bus dati bidirezionale. È composto da sedici linee.
- **FIFOADR**: è un segnale a due bit tramite il quale il *master* seleziona la FIFO su cui vuole operare. In questo progetto si usa una sola FIFO, quindi anche questo segnale assume un valore fisso.

Normalmente i segnali relativi allo FX2 sono quindi attivi bassi; in realtà questa proprietà si può cambiare attraverso la programmazione. In questo progetto non c'era motivo di cambiare l'impostazione predefinita, quindi è stata mantenuta.

Il microcontrollore è montato su una scheda apposita (Figura 4.1.7) che ha come porte di comunicazione un connettore *header* maschio da due file con 20 *pin* ciascuna e una porta USB-B, idonei al collegamento rispettivamente con la scheda di sviluppo e con un apparecchio esterno.

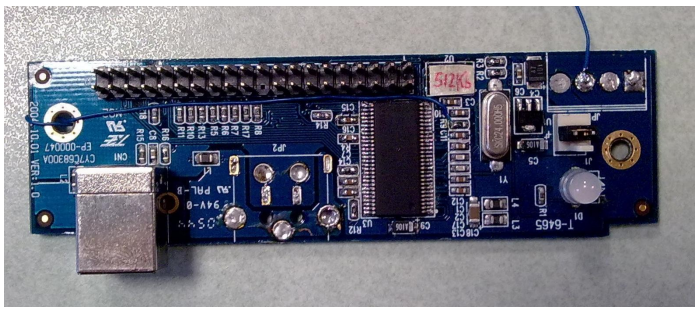

Figura 4.1.7. Scheda che monta il microcontrollore ed il connettore *header*.

La porta USB del dispositivo implementa il protocollo USB 2.0 che rispetto alla prima versione include in particolare:

- velocità massima teorica di trasferimento dati di 480 Mbit/s, quaranta volte superiore rispetto ai 12 Mbit/s delle precedenti specifiche USB 1.1;
- piena compatibilità con dispositivi e cavi USB 1.1;
- una nuova architettura *hub* che supporta *downstream* multipli a 12 Mbit/s per più dispositivi USB 1.1.

Una precisazione riguardo allo schema di Figura 4.1.3. Un concetto fondamentale del protocollo USB è che in un sistema USB c'è un solo *master*, ed è il computer *host*. Di conseguenza è sempre lo *host* che inizia una comunicazione, mai le periferiche, tranne nel caso in cui una di esse sia stata messa dallo *host* in uno stato di sospensione a bassa potenza. In quest'ultimo caso, infatti, la periferica può inviare un segnale di *wakeup* remoto. Ciò detto, è facile capire e ricordare il verso delle connessioni USB, come quelle mostrate in Figura 4.1.3; *OUT* significa dallo *host* al dispositivo, *IN* dal dispositivo allo *host*.

4.1.3 Il connettore VITA 57.1 FMC–LPC

Il connettore in oggetto è parte integrante della scheda di sviluppo Xilinx Spartan 6. In questo progetto serve a collegare la Spartan-6 alla scheda su cui è montato il microcontrollore, fornendo un accesso più agevole ai *pin* utilizzati. Il connettore maschio compatibile con quello in oggetto non era dispo-

nibile, quindi si è optato per ordinare un connettore simile ma con più *pin*, usandone poi solo una parte. La scheda che alloggia il microcontrollore monta anche un connettore *header*, non collegabile direttamente al connettore FMC; perciò si è reso necessario trovare un modo per mettere in comunicazione i due connettori. Si è valutato sufficiente, trattandosi di un prototipo sperimentale, utilizzare un cavo *IDE* avente il relativo connettore da un lato, e fili liberi dall'altro. I fili sono poi stati saldati su una basetta millefori, assieme ad altri fili di sezione minore che sono infine stati saldati sul connettore FMC. La scelta di interporre la basetta millefori invece di saldare direttamente il cavo IDE sul connettore FMC è stata dettata da considerazioni di robustezza e di comodità di saldatura.

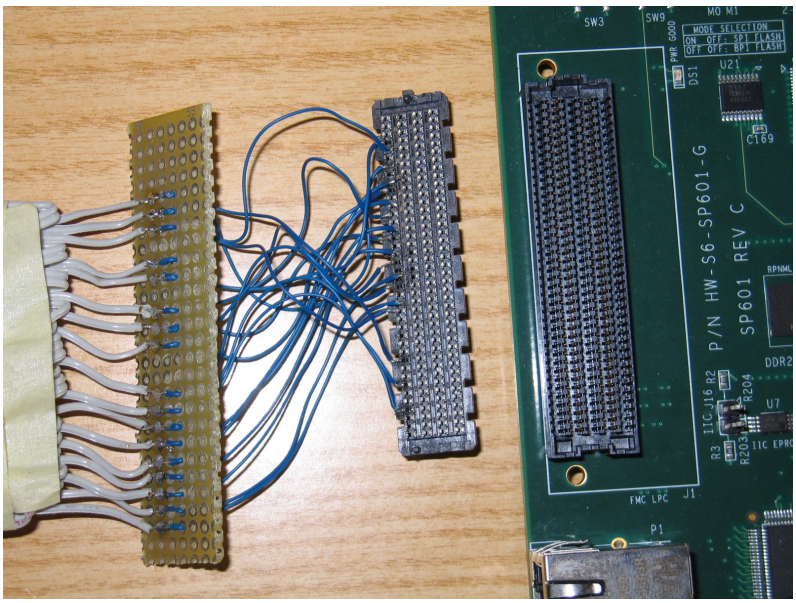

Figura 4.1.8. Da sinistra: la basetta millefori, il connettore maschio ed il connettore VITA 57.1 FMC-LPC.

Il connettore VITA 57.1 FMC–LPC (*Low Pin Count*) permette l'utilizzo di 34 segnali differenziali definibili dall'utente oltre ad alcuni segnali specifici. Il fattore di forma è quello di un connettore a 10 file per 40 posizioni ciascuna, per un totale di 400 *pin* possibili (nella versione *High Pin Count, HPC*), ma la versione LPC ne contiene solo 160, disposti in 4 file. La struttura del connettore, la disposizione dei *pin* e il significato dei segnali associati sono

mostrati in Figura 4.1.9. I *pin* marcati con la sigla NC (non connesso) si riferiscono alle file mancanti che differenziano la versione LPC dalla HPC.

Volendo qui presentare semplicemente e sommariamente il connettore usato, si rimanda al sito web apposito per informazioni più dettagliate [30]. Per informazioni sulle connessioni tra il VITA 57.1 FMC–LPC e il *chip* FPGA si rimanda alla documentazione della scheda di sviluppo [31].

	K	J	H	G	F	E	D	C	B	A
1	NC	NC	VREF_A_M2C	GND	NC	NC	PG_C2M	GND	NC	NC
2	NC	NC	PRSNT_M2C_L	CLK1_M2C_P	NC	NC	GND	DP0_C2M_P	NC	NC
3	NC	NC	GND	CLK1_M2C_N	NC	NC	GND	DP0_C2M_N	NC	NC
4	NC	NC	CLK0_M2C_P	GND	NC	NC	GBTCLK0_M2C_P	GND	NC	NC
5	NC	NC	CLK0_M2C_N	GND	NC	NC	GBTCLK0_M2C_N	GND	NC	NC
6	NC	NC	GND	LA00_P_CC	NC	NC	GND	DP0_M2C_P	NC	NC
7	NC	NC	LA02_P	LA00_N_CC	NC	NC	GND	DP0_M2C_N	NC	NC
8	NC	NC	LA02_N	GND	NC	NC	LA01_P_CC	GND	NC	NC
9	NC	NC	GND	LA03_P	NC	NC	LA01_N_CC	GND	NC	NC
10	NC	NC	LA04_P	LA03_N	NC	NC	GND	LA06_P	NC	NC
11	NC	NC	LA04_N	GND	NC	NC	LA05_P	LA06_N	NC	NC
12	NC	NC	GND	LA08_P	NC	NC	LA05_N	GND	NC	NC
13	NC	NC	LA07_P	LA08_N	NC	NC	GND	GND	NC	NC
14	NC	NC	LA07_N	GND	NC	NC	LA09_P	LA10_P	NC	NC
15	NC	NC	GND	LA12_P	NC	NC	LA09_N	LA10_N	NC	NC
16	NC	NC	LA11_P	LA12_N	NC	NC	GND	GND	NC	NC
17	NC	NC	LA11_N	GND	NC	NC	LA13_P	GND	NC	NC
18	NC	NC	GND	LA16_P	NC	NC	LA13_N	LA14_P	NC	NC
19	NC	NC	LA15_P	LA16_N	NC	NC	GND	LA14_N	NC	NC
20	NC	NC	LA15_N	GND	NC	NC	LA17_P_CC	GND	NC	NC
21	NC	NC	GND	LA20_P	NC	NC	LA17_N_CC	GND	NC	NC
22	NC	NC	LA19_P	LA20_N	NC	NC	GND	LA18_P_CC	NC	NC
23	NC	NC	LA19_N	GND	NC	NC	LA23_P	LA18_N_CC	NC	NC
24	NC	NC	GND	LA22_P	NC	NC	LA23_N	GND	NC	NC
25	NC	NC	LA21_P	LA22_N	NC	NC	GND	GND	NC	NC
26	NC	NC	LA21_N	GND	NC	NC	LA26_P	LA27_P	NC	NC
27	NC	NC	GND	LA25_P	NC	NC	LA26_N	LA27_N	NC	NC
28	NC	NC	LA24_P	LA25_N	NC	NC	GND	GND	NC	NC
29	NC	NC	LA24_N	GND	NC	NC	TCK	GND	NC	NC
30	NC	NC	GND	LA29_P	NC	NC	TDI	SCL	NC	NC
31	NC	NC	LA28_P	LA29_N	NC	NC	TDO	SDA	NC	NC
32	NC	NC	LA28_N	GND	NC	NC	3P3VAUX	GND	NC	NC
33	NC	NC	GND	LA31_P	NC	NC	TMS	GND	NC	NC
34	NC	NC	LA30_P	LA31_N	NC	NC	TRST_L	GA0	NC	NC
35	NC	NC	LA30_N	GND	NC	NC	GA1	12P0V	NC	NC
36	NC	NC	GND	LA33_P	NC	NC	3P3V	GND	NC	NC
37	NC	NC	LA32_P	LA33_N	NC	NC	GND	12P0V	NC	NC
38	NC	NC	LA32_N	GND	NC	NC	3P3V	GND	NC	NC
39	NC	NC	GND	VADJ	NC	NC	GND	3P3V	NC	NC
40	NC	NC	VADJ	GND	NC	NC	3P3V	GND	NC	NC

Figura 4.1.9. *Pinout* del connettore VITA 57.1 FMC-LPC.

4.2 Implementazione dell'interfaccia

L'interfaccia è stata realizzata tramite l'implementazione di una *Macchina a Stati Finiti* (*MSF*, o *FSM – Finite State Machine*, in inglese). In particolare è stata implementata una MSF secondo Moore. Nel paragrafo 4.2.1

vengono presentate sinteticamente le caratteristiche principali delle macchine a stati finiti.

4.2.1 Cenni alle Macchine a Stati Finiti nei linguaggi HDL

Altre differenze secondarie sono illustrate nel seguito.

Gli strumenti software per la descrizione dell'hardware contengono algoritmi di estrazione per l'identificazione ed ottimizzazione del modello di MSF ad essi sottoposto; perché questo avvenga correttamente è bene seguire degli stili di codifica ben precisi. Le linee guida generali per la scelta delle caratteristiche da assegnare ad una macchina in fase di progetto sono presentate in questo paragrafo, ma normalmente i parametri più importanti sono l'architettura di destinazione e le specifiche di dimensione e di comportamento della macchina stessa.

Moore vs. Mealy

La principale caratteristica di una MSF è l'essere una realizzazione del modello di Moore o di quello di Mealy Nelle macchine di Moore i valori che assumono le uscite dipendono solo dallo stato presente (inteso come stato in cui si trova la macchina nell'istante considerato), invece nelle macchine di Mealy le uscite sono calcolate sia in base allo stato presente che al valore degli ingressi. Di solito per gli FPGA sono preferite le macchine di Moore con codifica *One-Hot* a causa della minor logica necessaria per le uscite, mentre con una macchina di Mealy a codifica Binaria è possibile che si ottenga una macchina a stati più compatta e, talvolta, più veloce; tuttavia questo non è sempre vero, e non è facile determinarlo senza avere specifiche più precise.

Codifica *One-Hot* vs. Binaria

Questi due metodi di codifica sono i più diffusi per descrivere macchine a stati finiti per FPGA o CPLD[7] benché non gli unici. Per la maggior parte degli FPGA la codifica *One-Hot* è la migliore per l'abbondanza di flip-flop e

7 *Complex Programmable Logic Device: dispositivo logico programmabile e cancellabile. A differenza degli FPGA, i CPLD mantengono la programmazione anche se non alimentati grazie a delle memorie non volatili.*

le minori esigenze in termini di *fan-in*[8] per la logica di calcolo dello stato successivo (viene mappato meglio nelle LUT) [32]. Per i CPLD di solito si usa la codifica Binaria, per la struttura logica e la minor quantità di registri. I moderni strumenti di sintesi contengono algoritmi di estrazione capaci di analizzare il codice della macchina a stati e scegliere il miglior metodo di codifica per l'architettura di destinazione, tuttavia a volte può essere più conveniente codificare manualmente la macchina per permetterne un miglior controllo e facilitarne il debug.

Safe vs. Fast

Come si evince dai nomi di queste modalità, esse si differenziano per la caratteristica di essere una più sicura ma meno veloce, e l'altra più veloce a scapito della sicurezza (intesa come robustezza). Più precisamente, la modalità Safe introduce uno stato "sicuro" in cui la macchina va ad operare nel caso in cui, per qualsiasi motivo, gli ingressi la porterebbero ad operare in uno stato non previsto. La modalità Fast, d'altro canto, non prevedendo questo stato, di solito dà luogo ad una macchina con meno logica da sintetizzare, e quindi più veloce.

In conclusione, tipicamente, le MSF di Moore con codifica *One-Hot* sono indicate per architetture FPGA, mentre macchine di Mealy con codifica Binary sono consigliate per dispositivi come i CPLD. In base a queste informazioni ho deciso di realizzare la macchina a stati finiti oggetto di questa interfaccia secondo il modello di Moore con codifica *One-Hot*.

4.2.2 I moduli aggiuntivi

Il modulo *top* dell'interfaccia richiama e utilizza due moduli elementari che sono descritti in questo paragrafo. Dopo vari tentativi di scrivere l'interfaccia in un unico blocco di codice si è visto che la strategia migliore è separare le funzioni basilari, quando possibile e conveniente, e scriverle in moduli separati, richiamandoli poi nel codice principale. Questo modo di procedere ha

8 *Il numero di input accettabili da un circuito.*

il vantaggio di permettere una migliore gestione del codice così come un miglior debug. Infatti, avendo un modulo semplice, con poche righe di codice, è più facile creare un *testbench* che lo simuli correttamente e correggere eventuali errori in ogni dettaglio. Avendo un codice complesso, invece, scoprire dove si trova l'errore è più complicato, anche perché esso potrebbe risiedere nel *testbench* invece che nel (o in aggiunta al) codice.

Il primo modulo, **bidir_port**, il cui schema è mostrato in Figura 4.2.1, è una semplice porta bidirezionale applicata ad un bus invece che ad un singolo segnale. Nella figura, il segnale di *enable* è l'unico a direzione singola, le altre frecce rappresentano bus a 16 linee.

Il modulo, comandato dall'esterno dal segnale di *enable*, sostanzialmente non fa altro che abilitare il bus bidirezionale bidirBus collegandolo al bus di ingresso busIn, oppure disabilitarlo. Più precisamente, quando *enable* è a livello logico basso le linee di bidirBus assumono gli stessi valori di quelle di busIn, quando è a livello logico alto si trovano in stato di alta impedenza; busOut è un bus di sola uscita ed ha sempre lo stesso valore di bidirBus.

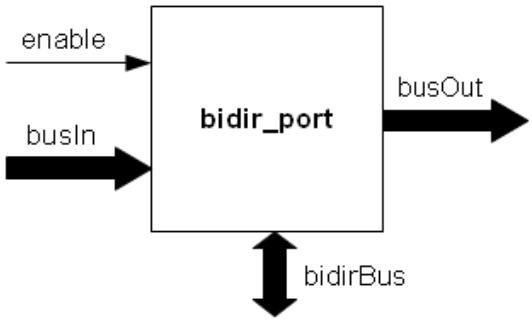

Figura 4.2.1. Il modulo bidir_port.

Con questo schema, collegando due blocchi assieme come mostrato in Figura 4.2.2, è possibile gestire la lettura e la scrittura su un bus bidirezionale, e comunicare i dati all'esterno in entrambe le direzioni. La macchina a stati dell'interfaccia infatti utilizza due istanze di bidir_bus collegate come in figura Figura 4.2.2 per comunicare con il microcontrollore, delegando ad esse questo onere.

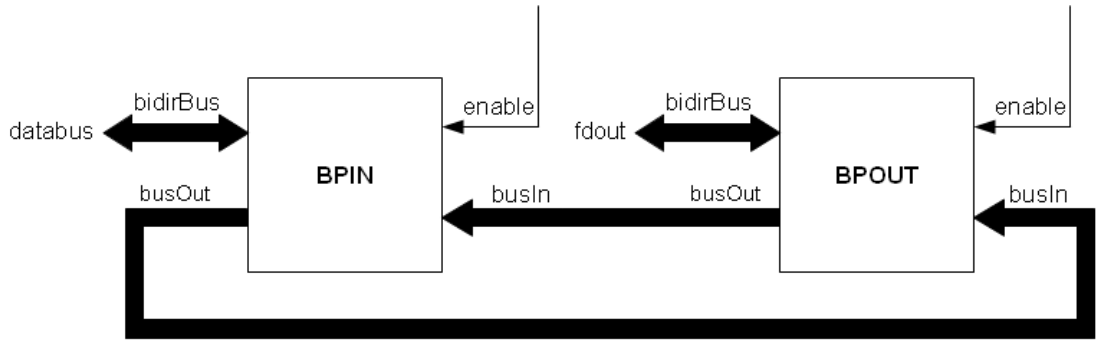

Figura 4.2.2. La gestione del bus bidirezionale usando due moduli bidir_bus.

Il funzionamento è il seguente. I due segnali di *enable* vengono controllati separatamente dalla macchina a stati finiti, ricordando che sono attivi bassi e che il bus bidirezionale collegato a BPIN (databus) comunica con il resto dello FPGA (deciderà l'utilizzatore dell'interfaccia come collegarlo), mentre quello collegato a BPOUT (fdout) è effettivamente il bus di uscita collegato al microcontrollore. Sostanzialmente le modalità in cui questo circuito opera, e che sono le stesse con cui la macchina a stati lo comanda sono tre:

- **disattivo**: quando la macchina a stati vuole mettere in stato di alta impedenza i due bus bidirezionale non deve far altro che porre a livello logico alto i due *enable*. In questo modo databus è isolato da fdout;
- **scrittura**: se la macchina deve scrivere su fdout, BPIN deve essere in lettura e BPOUT in scrittura, quindi lo *enable* di BPIN dovrà essere alto e quello di BPOUT basso. In questo modo la macchina può scrivere su databus (e di conseguenza su busOut), i dati compaiono in ingresso a BPOUT il quale essendo abilitato scrive infine su fdout;
- **lettura**: in lettura il circuito deve essere comandato esattamente all'opposto del caso di scrittura. Il funzionamento è quindi identico, seppur speculare, al caso precedente.

Il secondo modulo richiamato dalla macchina a stati finiti è il modulo **counter**, schematizzato in Figura 4.2.3. Questo modulo conta il numero di ci-

cli di clock e pone a livello logico alto l'uscita count_end quando il conteggio ha raggiunto la soglia impostata Oltre al segnale di clock in ingresso e all'uscita, lo schema presenta due ingressi di controllo (reset e clock_enable) e due parametri tramite i quali si può cambiare la dimensione del contatore e la soglia di fine conteggio (counter_bit_number e threshold). Grazie a questi due parametri il contatore è generico, adattabile secondo necessità ogni volta che lo si richiama all'interno di un altro modulo. In particolare si possono variare il numero di bit (tramite counter_bit_number), quindi il numero massimo al quale può arrivare il conteggio e la soglia (tramite threshold), cioè il numero al quale l'uscita viene abilitata. Ai fini di questo singolo progetto non era strettamente necessario che il contatore fosse adattabile, ma si è scelto di realizzarlo così in modo che fosse riutilizzabile anche in futuro con poche modifiche o addirittura senza, in previsione dell'implementazione su altre macchine e sistemi dell'azienda.

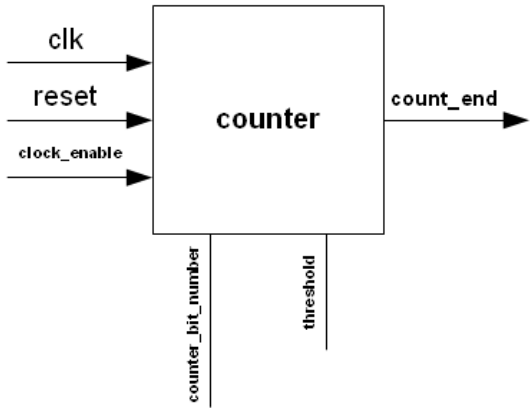

Figura 4.2.3. Il modulo counter.

La macchina a stati richiede questo contatore perché, come si vedrà nel paragrafo 4.2.3, il suo funzionamento prevede un ritardo di due cicli di clock da quando viene abilitata a quando i dati vengono presentati sul bus; questi ultimi devono essere stabili prima di poter abilitare i segnali *read* o *write* che comandano il microcontrollore. È necessario quindi contare i cicli di clock da quando lo *enable* viene attivato per essere certi di eseguire l'operazione con dati validi. Il segnale reset è attivo basso, quindi finché è basso il contatore è azzerato e l'uscita rimane bassa; quando è alto il funzionamento del contatore

dipende da clock_enable, anch'esso attivo basso. Se il segnale clock_enable, è alto mentre reset è alto, il contatore rimane fermo all'ultimo valore contato, e i fronti di clock non lo fanno commutare. Quindi il contatore opera quando reset è alto e clock_enable è basso, incrementando il conteggio ad ogni fronte positivo del clock; quest'ultimo è lo stesso ifclk che governa la macchina a stati.

4.2.3 La Macchina a Stati Finiti implementata

Il modulo principale dell'interfaccia è costituito da una macchina a stati finiti di Moore progettata appositamente per rispettare le specifiche fornite; essa usa la codifica *One-Hot* e la modalità *Safe*. Il funzionamento della MSF è illustrato in Figura 4.2.4, dove sono anche mostrati i valori delle uscite della macchina per ogni stato ed i valori degli ingressi che contribuiscono a determinare lo stato successivo. In tale figura i nomi dei segnali sono quelli effettivamente scritti nel codice Verilog; è stato scelto di non sostituirli con i nomi finora citati sia per evidenziare la differenza con i segnali di ingresso, che in alcuni casi hanno significato simile (per esempio gli *enable*), che per esplicitare la gestione dei sotto-moduli che gestiscono i bus bidirezionali.

Prima di esporre in dettaglio il funzionamento della MSF è opportuno ricordare o spiegare il significato dei segnali di ingresso e di uscita dell'interfaccia. Per i dettagli sui segnali di comunicazione tra il microcontrollore ed il *chip* FPGA si rimanda al paragrafo 4.1.2, Figura 4.1.6, ed al paragrafo 4.1, Figura 4.1.1. Si avverte inoltre che alcuni segnali presenti nelle figure appena citate non sono visibili in Figura 4.2.4 perché generati all'esterno della macchina a stati, seppur nello stesso modulo, oltre che per ragioni di chiarezza espositiva. La gestione di questi segnali sarà trattata nel seguito di questo paragrafo.

- Il segnale di *enable* è **nEN**, ed è lo stesso descritto nel paragrafo 4.1.
- **nEN_IN** e **nEN_OUT** sono, rispettivamente, lo *enable* del modulo BPIN e lo *enable* di BPOUT; entrambi i moduli sono di tipo bidir_port. Sono controllati dalla MSF perché servono a control-

lare la direzione del dato e quindi la gestione della scrittura o della lettura.

- Il segnale **rW** indica alla macchina l'operazione che deve eseguire: se rW="0" è richiesta una scrittura, quindi chi comanda l'interfaccia deve passare ad essa un dato che verrà scritto nella coda FIFO del microcontrollore. Se invece rW="1" è richiesta una lettura, quindi l'interfaccia deve restituire un dato.
- **ifclk** è il segnale di clock; si è programmato il microcontrollore perché lo fornisca a 24 MHz. Viene passato tramite un *pin* apposito al *chip* FPGA ed è il clock che temporizza l'interfaccia.
- **full**, corrispondente a FLAGB, è il segnale attivo basso proveniente dal microcontrollore che segnala se la FIFO utilizzata è piena.
- **empty**, analogo al segnale precedente, è un segnale attivo basso proveniente dal microcontrollore che segnala che la FIFO utilizzata è vuota. Corrisponde a FLAGC.
- **done** (non visibile in Figura 4.2.4) è un'uscita attiva alta che la macchina a stati restituisce al modulo che la comanderà. Essa viene abilitata quando uno dei due comandi *read* o *write* viene abilitato, cioè quando viene effettivamente ordinata un'azione al microcontrollore.
- **fault** è un segnale attivo alto analogo a *done*, solo che segnala il fallimento dell'operazione richiesta. Ovviamente, nessun comando viene impartito al microcontrollore in tal caso.
- **fx2_output_enable** corrisponde a SLOE, è un'uscita della MSF e si comporta esattamente come descritto al paragrafo 4.1.2.
- ***read*** (non visibile in Figura 4.2.4) è il comando di lettura che viene effettivamente dato al microcontrollore. Corrisponde a SLRD, e viene calcolato da logica combinatoria fuori dalla MSF, anche se ovviamente è dipendente dalle uscite della stessa.

- ***write*** (non visibile in Figura 4.2.4) è l'effettivo comando di scrittura impartito al microcontrollore. Corrisponde a SLRWR ed è calcolato come il segnale *read*.
- **pktend** (non visibile in Figura 4.2.4), la cui funzione è stata già spiegata nel paragrafo 4.1.2, nel codice dell'interfaccia è stato posto al valore negato di *done*, valore "casuale" che non causa problemi all'interfaccia. Si ricorda che per gli scopi iniziali di questo progetto non ne è previsto l'utilizzo, ma si voleva comunque implementarlo per una possibile futura gestione dei "pacchetti corti" (si veda il paragrafo 4.1.2).
- **fifoadr** (non visibile in Figura 4.2.4) è l'indirizzo a due bit della FIFO che si vuole usare. In questo caso, usandone una sola, il segnale è stato fissato al valore di un parametro (SELECT_FIFO_ADDRESS="00") in modo tale che il modulo che userà questa interfaccia in caso di necessità possa cambiare agevolmente dall'esterno questo valore per selezionare un'altra FIFO.
- **databus** (non visibile in Figura 4.2.4) è il bus a 16 bit che collega l'interfaccia al resto dello FPGA, cioè al modulo controllore che la userà e sul quale probabilmente saranno poi collegati altri dispositivi. A questo riguardo, però, le specifiche non forniscono ulteriori dettagli.
- **fdout** (non visibile in Figura 4.2.4), corrisponde a FD ed è il bus che collega il *chip* FPGA al microcontrollore.

C'è anche un segnale interno all'interfaccia, **count_end**, che è l'uscita del modulo contatore; va a livello logico alto quando il contatore raggiunge la soglia impostata e serve alla MSF per la corretta temporizzazione dei dati e per il calcolo delle uscite *write* e *read*.

Essendo una macchina di Moore, ci sono quattro stati in cui la macchina può trovarsi ad operare: A, B, C, D. Nel seguito sono descritti il significato ed il comportamento di ciascuno stato.

Stato A.

Il primo stato, A, è lo stato in cui si trova la macchina quando il circuito che essa comanda è disabilitato dall'esterno, ma è anche lo stato in cui rimane la macchina mentre aspetta un comando e quello a cui deve essere riportata in caso di errore. Per questi motivi, allo stato A ci si può riferire con i nomi "IDLE" o "SAFE", cioè come stato di *default* o stato "sicuro". In tale stato, infatti, entrambi i bus bidirezionali sono posti in stato di alta impedenza dallo FPGA, in modo che non possano verificarsi *race*[9]. Se la MSF è in questo stato e lo *enable* è disabilitato la macchina continua a rimanere in tale stato. Solo quando nEN viene abilitato la macchina cambia stato. Se rW="0" è richiesta una scrittura, ma questa deve essere effettuata solo se la FIFO non è piena; analogamente, se rW=1 è richiesta una lettura che deve essere eseguita solo se la FIFO non è vuota. Negli altri casi, cioè se viene richiesta una scrittura con FIFO piena od una lettura da FIFO vuota, la MSF entra in uno stato d'errore (D).

Stato B

Lo stato B è lo stato di scrittura. Se la MSF va a lavorare in questo stato significa che è già stato controllato che l'operazione si potesse fare. A questo punto per scrivere sulla FIFO si devono eseguire le seguenti operazioni: si deve disattivare *fx2_output_enable* perché si deve acquisire il controllo del bus (e non lasciarlo al microcontrollore), presentare il dato sul bus ed impostare i moduli BPIN e BPOUT per la scrittura sul bus *fdout*. Infine si deve disabilitare *fault*. Come si vedrà più avanti in questo paragrafo la logica esterna alla macchina provvederà ad abilitare i comandi *write* e *done* quando necessario.

Stato C

Lo stato C si comporta esattamente come B, salvo che C è lo stato in cui la macchina legge dati dalla FIFO e li rende disponibili su *databus*. Que-

9 *Condizione in cui due dispositivi cercano di porre una stessa linea a due tensioni differenti. Si creano così delle correnti che possono anche danneggiare i dispositivi.*

sta volta i moduli *bidir_port* devono essere configurati per la lettura, mentre *fx2_output_enable* deve essere abilitato perché il microcontrollore possa presentare i dati sul bus *fdout*. Anche qui *fault*="0" mentre il calcolo di *read* e *done* è lasciato alla già citata logica esterna.

Stato D

Questo stato è lo stato di *"fault"*, in cui la macchina va a lavorare quando si tenta di scrivere sulla FIFO piena o leggere dalla FIFO vuota. Come si può notare dallo schema, le uscite della MSF in questo stato sono come quelle dello stato A, tranne per il segnale *fault* che nello stato D è abilitato per segnalare l'avvenuto errore. Ovviamente in questo caso *done*, *write* e *read* non saranno abilitati dalla logica dedicata.

È interessante notare come negli stati B, C e D l'unico modo per tornare allo stato di partenza A è porre nEN="1" disabilitando di fatto l'interfaccia. Anche questa condizione è stata data come specifica di progetto. La MSF nello stato A invece riguardo a questo si comporta nel modo opposto: se nEN="1" rimane nello stesso stato.

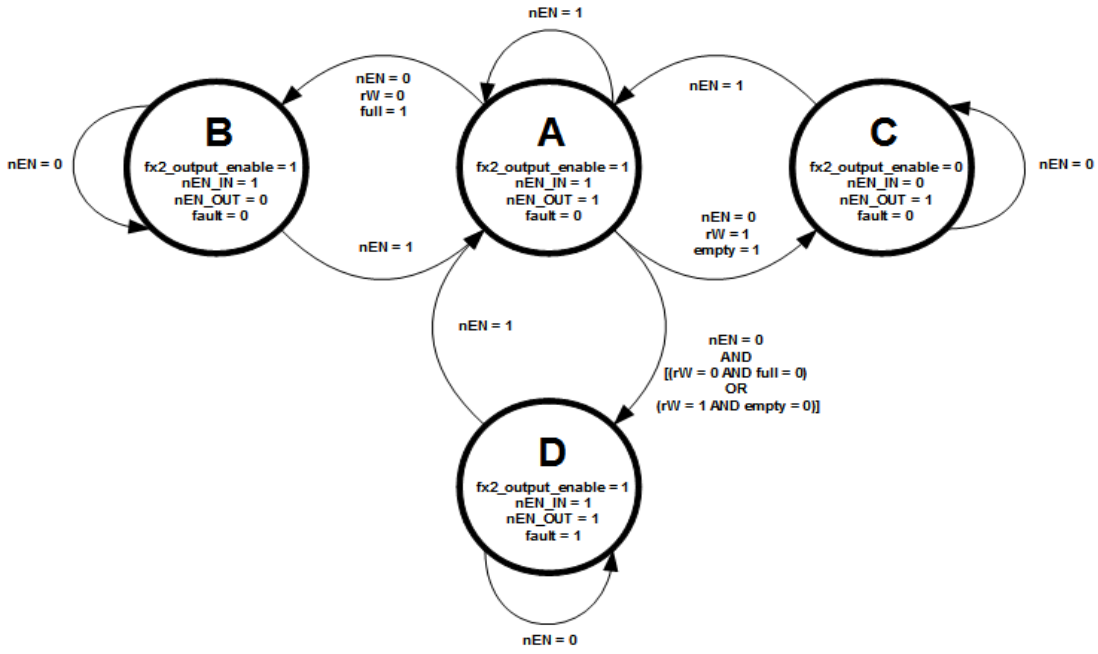

Figura 4.2.4. La macchina a stati finiti alla base dell'interfaccia.

Nel seguito è esplicitato il codice per il calcolo delle uscite *fifoadr*, *pktend*, *write* e *read*.

```
assign fifoadr = SELECT_FIFO_ADDRESS;
assign pktend = ~done;
assign write = ((count_end == 1) && (rW == 0) && (full == 1)) ? 0 : 1;
assign read = ((count_end == 1) && (rW == 1) && (empty == 1)) ? 0 : 1;
```

Queste linee di codice sono inserite nel corpo principale del programma, ed inferiscono logica combinatoria. Le prime due righe sono già state sufficientemente commentate, mentre per le rimanenti si può osservare che il rispettivo segnale viene abilitato (in questo caso posto a "0") se sono verificate tre condizioni su altrettanti segnali. In particolare perché il microcontrollore venga effettivamente abilitato in scrittura o lettura, il contatore deve aver terminato il conteggio, deve essere stato impartito il comando giusto e la FIFO non deve essere piena (in caso di scrittura) o vuota (in caso di lettura).

Per quanto riguarda la generazione di *done*, il codice è il seguente:

```
always @(posedge ifclk)
   if ((write == 0) || (read == 0))
      done <= 1;
else if (nEN == 1)
      done <= 0;
```

Come si può vedere *done*, che indica la riuscita dell'operazione richiesta, viene abilitato quando il relativo comando è effettivamente impartito al microcontrollore, mentre viene disabilitato quando l'interfaccia viene disabilitata a sua volta.

La macchina a stati è stata scritta usando un'istruzione "*parallel case*", perché i possibili valori dello stato sono mutualmente esclusivi, e "*full case*" perché tutti i possibili valori dello stato sono specificati nel *case* [33]. In aggiunta, se per qualsiasi motivo la macchina durante il funzionamento non do-

vesse entrare in uno degli stati previsti c'è un'istruzione "*default*" che di fatto pone le uscite agli stessi valori in cui le porrebbe lo stato A (stato "sicuro").

Per ultimo, si evidenzia la particolare codifica degli stati dovuta alla scelta della modalità "*One-hot*".

```
localparam A = 4'b0001;
localparam B = 4'b0010;
localparam C = 4'b0100;
localparam D = 4'b1000;
```

Il nome di questa codifica deriva dal fatto che per ogni stato codificato un solo bit è a "1", mentre gli altri sono tutti a "0". Concretamente, questo semplifica la logica di decodifica, in quanto viene inferito un solo flip-flop per stato.

4.3 Simulazioni e risultati

In questo paragrafo sono riportate e commentate le simulazioni comportamentale e *post place-and-route*.

Figura 4.3.1 Mostra la simulazione comportamentale dell'interfaccia realizzata.

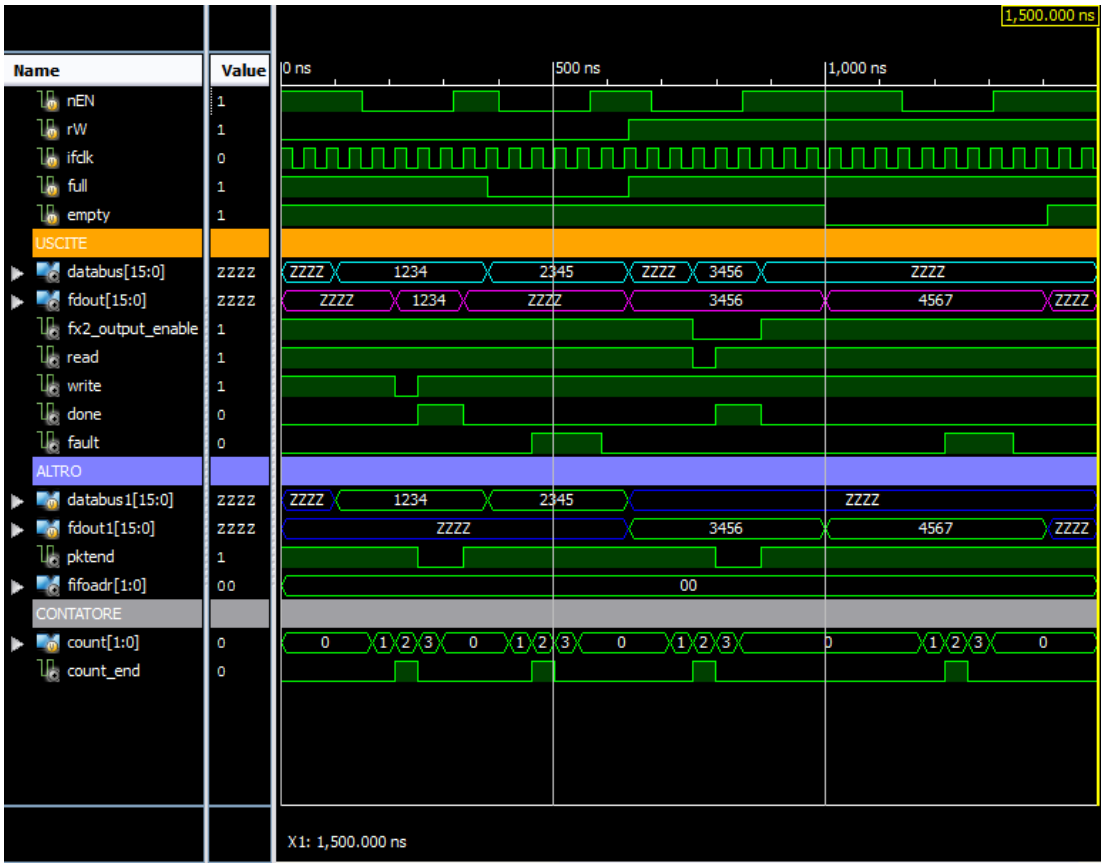

Figura 4.3.1. Schermata della simulazione comportamentale dell'interfaccia.

La prima riga mostra il comportamento del segnale di *enable*. Si vede che, creando un *testbench* apposito, sono state simulate quattro prove: le prime due di scrittura e le seconde di lettura (si veda anche il segnale rW). Il *testbench* simula contemporaneamente il comportamento del modulo "controllore" che usa l'interfaccia e quello del microcontrollore. La prima prova consiste in una scrittura con FIFO né piena né vuota. La scrittura viene eseguita correttamente, cioè il valore di *databus* viene copiato su *fdout*. La seconda prova consiste in una scrittura con FIFO piena (*full*="0"). Il dato non deve essere scritto e così è. Il bus *fdout* rimane in alta impedenza. Le ultime due prove sono del tutto analoghe, solo che riguardano la lettura. La prima prova di lettura riesce perché la FIFO non è vuota. Il dato su *fdout* viene copiato su *databus*. La seconda prova di lettura invece non riesce, com'è giusto che sia, perché si sta tentando di leggere da una FIFO vuota. Il bus *databus* non viene scritto. Si noti inoltre che i comandi *write* e *read*, come anticipato, sono abilitati con due cicli di clock in ritardo rispetto alla discesa dello *enable*. Dai

data sheet del microcontrollore si vede che l'acquisizione del comando avviene alla risalita dello stesso, e nella simulazione si può vedere che in quell'istante il dato è già stabile. In quell'istante viene anche abilitato il segnale *done* o *fault*, a seconda del caso. Questi ultimi due segnali, come da specifiche, vengono resettati alla disabilitazione dell'interfaccia (nEN="1"). Si segnala infine che *databus1* e *fdout1* sono i segnali *databus* e *fdout* così come vengono impostati dal *testbench*; è stato necessario crearli per esigenze di simulazione.

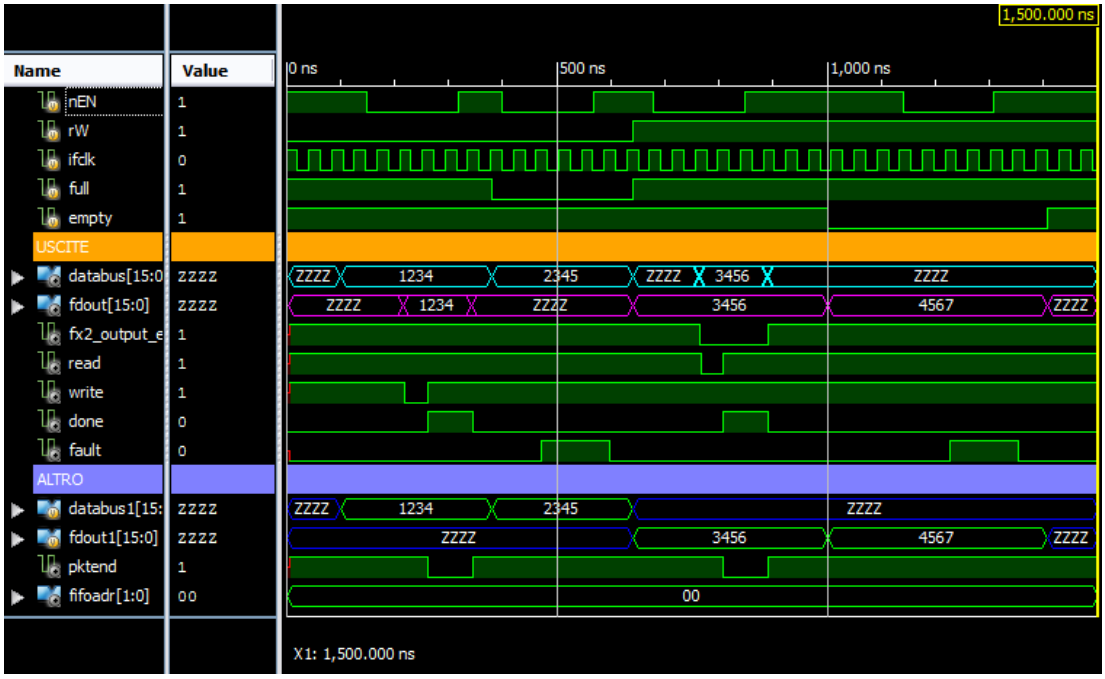

Figura 4.3.2. Schermata della simulazione *post place-and-route*.

La simulazione *post place-and-route*, visibile in Figura 4.3.2, conferma quanto detto per la simulazione comportamentale. Sono presenti ovvi ritardi dovuti alla natura stessa (più realistica) della simulazione ma tutto funziona allo stesso modo. La differenza più evidente è probabilmente il passaggio dei bus da un valore all'altro. Questo accade perché ogni linea del bus ha un diverso tempo di propagazione, ma quello che importa è che al momento del campionamento del dato esso sia costante.

In Tabella 4 si riportano le informazioni sulle risorse occupate fornite dal software usato per l'implementazione (Xilinx ISE 12.2).

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	11	18,224	1%
Number used as Flip Flops	11		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	16	9,112	1%
Number used as logic	16	9,112	1%
Number using O6 output only	14		
Number using O5 output only	0		
Number using O5 and O6	2		
Number used as ROM	0		
Number used as Memory	0	2,176	0%
Number of occupied Slices	9	2,278	1%
Number of LUT Flip Flop pairs used	16		
Number with an unused Flip Flop	5	16	31%
Number with an unused LUT	0	16	0%
Number of fully used LUT-FF pairs	11	16	68%
Number of unique control sets	3		
Number of slice register sites lost to control set restrictions	13	18,224	1%
Number of bonded IOBs	45	232	19%
Number of RAMB16BWERs	0	32	0%
Number of RAMB8BWERs	0	64	0%
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%
Number of BUFG/BUFGMUXs	1	16	6%
Number used as BUFGs	1		
Number used as BUFGMUX	0		
Number of DCM/DCM_CLKGENs	0	4	0%
Number of ILOGIC2/ISERDES2s	0	248	0%
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	248	0%
Number of OLOGIC2/OSERDES2s	0	248	0%
Number of BSCANs	0	4	0%

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of BUFHs	0	128	0%
Number of BUFPLLs	0	8	0%
Number of BUFPLL_MCBs	0	4	0%
Number of DSP48A1s	0	32	0%
Number of ICAPs	0	1	0%
Number of MCBs	0	2	0%
Number of PCILOGICSEs	0	2	0%
Number of PLL_ADVs	0	2	0%
Number of PMVs	0	1	0%
Number of STARTUPs	0	1	0%
Number of SUSPEND_SYNCs	0	1	0%
Average Fanout of Non-Clock Nets	2.51		

Tabella 4. Risorse occupate dall'interfaccia con il bus USB.

Sono stati condotti test finali con l'hardware a disposizione ed anche con un'interfaccia appositamente fornita da Cypress per scopi di test del microcontrollore. Nonostante fossero presenti segnali sul connettore della scheda *Spartan 6* l'applicazione per calcolatore non mostrava alcuno streaming di dati. È possibile quindi la presenza di un errore nella programmazione del microcontrollore, attività per altro esterna a questo tirocinio, oppure un problema hardware. Non si può del tutto escludere un problema di collegamenti, anche se questi sono stati accuratamente e ripetutamente controllati. Ulteriori test verranno condotti in una eventuale continuazione della collaborazione con l'azienda.

Si include infine una foto delle schede collegate come da test (Figura 1.1.3). Sulla scheda del microcontrollore (sulla sinistra) è visibile la porta USB-B per il collegamento col calcolatore. Il cavo grigio serve per la programmazione del chip FPGA e quello nero per l'alimentazione.

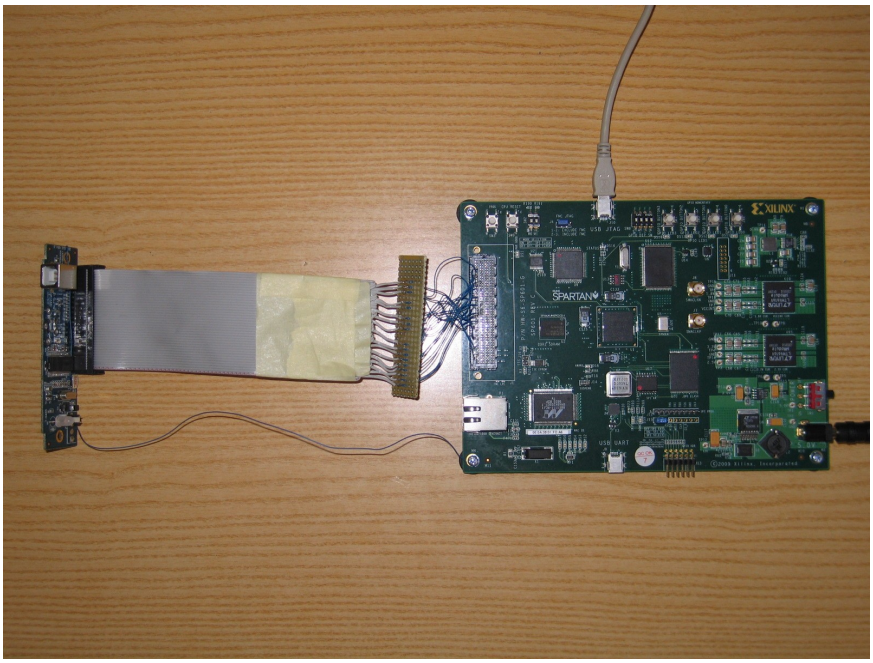

Figura 4.3.3. Le schede usate collegate per il test.

5 Conclusioni

In questo capitolo sono riassunti e commentati il lavoro svolto ed i risultati ottenuti. È poi spiegato cosa manca per completare il progetto ed infine sono esposti gli sviluppi futuri.

5.1 Commenti e conclusioni sul lavoro svolto

Il lavoro svolto in questo tirocinio è solo una parte del progetto finale che, ovviamente, per come è stato proposto e pensato dall'azienda deve portare ad un prototipo di scheda completo e funzionante. Lo sviluppo dell'intero progetto durante il tirocinio era una possibilità assai improbabile; inoltre, per le interfacce o per i dispositivi poi esclusi da questa relazione sono stati condotti degli studi iniziali che hanno inevitabilmente rallentato lo sviluppo delle parti presentate. Anche per questo, oltre che per problemi di approvvigionamento di parti hardware, non si è riusciti ad arrivare ad un test funzionante su scheda. In ogni caso, la soluzione proposta per il sistema di acquisizione potrà essere verificata non appena saranno noti marca e modello dei componenti scelti (sensore CCD e convertitore analogico/digitale). Per quanto riguarda l'interfaccia con il bus USB e la coda FIFO, il codice è completo e, in teoria, corretto e funzionante dati i risultati positivi di entrambe le simulazioni. Resta da capire perché al test pratico, che è stato possibile solo all'ultimo momento per la mancanza di un connettore, non si rileva uno streaming di dati corretti.

5.2 Completamento del progetto

Il completamento di tutto il progetto passa necessariamente dalla preventiva decisione da parte dell'azienda di quali componenti utilizzare realmente. Con tali informazioni si potrà progettare correttamente l'interfaccia di acquisizione e determinarne le prestazioni. Oltre a testare e far funzionare quanto già fatto durante questo tirocinio rimane quindi da implementare l'interfaccia con il convertitore analogico/digitale e da studiare e realizzare l'interfaccia con la memoria DDR2. Riguardo a quest'ultima, in questo tirocinio si sono fatte delle prove realizzandone una attraverso il MIG (*Memory Interface Generator*), uno strumento del pacchetto software della Xilinx. Purtroppo la preparazione derivante dagli studi universitari riguardo a questo argomento non è cosi approfondita, ed uno studio completo dell'argomento avrebbe richiesto più tempo di quello a disposizione. Per completare il prototipo, inoltre, è richiesta la programmazione del microcontrollore in linguaggio C e, soprattutto, l'implementazione di un "controllore" che gestisca tutto il sistema, compreso il comando del sensore. Questo controllore potrebbe essere, come già accennato precedentemente, un modulo MicroBlaze™ (quindi un processore costruito con le risorse del *chip* FPGA) oppure un *chip* esterno.

5.3 Sviluppi futuri

Il progetto nella sua interezza è di per sé un obiettivo a medio-lungo termine, tuttavia l'idea di fondo è quella di poter usare il circuito che deriverà dal prototipo su tutti gli apparecchi oftalmici prodotti dall'azienda. L'obiettivo ultimo è quindi costruire un sistema abbastanza adattabile da poter essere installato su tutte le macchine che, indipendentemente dalla loro funzione, possiedono un sistema di acquisizione video.

Se NIDEK Technologies darà seguito all'interesse già dimostrato per il proseguimento della collaborazione, il progetto ha le premesse per raggiungere ottimi risultati.

Bibliografia

[1] http://www.tuvglobal.com/.

[2] http://www.nidektechnologies.it/.

[3] http://www.centroculus.it/tcc.html.

[4] http://www.lsu-eye.lsuhsc.edu/faculty_detail.aspx?name=klyce_stephen.

[5] http://www.nidektechnologies.it/IT/ITProductsNavisAll.htm.

[6] http://www.xilinx.com/products/devkits/EK-S6-SP601-G.htm.

[7] http://www.xilinx.com/products/spartan6/index.htm.

[8] http://www.ieee.org/index.html.

[9] http://en.wikipedia.org/wiki/NTSC.

[10] http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

[11] http://it.wikipedia.org/wiki/Charge_Coupled_Device.

[12] http://en.wikipedia.org/wiki/Charge-coupled_device.

[13] http://en.wikipedia.org/wiki/Charge_amplifier.

[14] http://en.wikipedia.org/wiki/Bayer_filter.

[15] http://en.wikipedia.org/wiki/JPEG.

[16] http://en.wikipedia.org/wiki/TIFF.

[17] Hridya Valsaraju, "Implementing an FX2LP-FPGA Interface", http://www.cypress.com/?docID=21898, 30Apr. 2010.

[18] http://www.i2c-bus.org/.

[19] http://it.wikipedia.org/wiki/I%C2%B2C.

[20] http://en.wikipedia.org/wiki/Open_drain.

[21] NXP Semiconductors, "UM10204 I2C-bus specification and user manual", http://ics.nxp.com/support/documents/interface/pdf/i2c.bus.specificat

ion.pdf, 19 Giugno 2007.

[22] http://en.wikipedia.org/wiki/I%C2%B2C.

[23] http://opencores.org/project,i2c.

[24] http://opencores.org/.

[25] http://www.gnu.org/licenses/lgpl.html.

[26] http://en.wikipedia.org/wiki/Wishbone_%28computer_bus%29.

[27] http://www.xilinx.com/tools/microblaze.htm.

[28] http://en.wikipedia.org/wiki/Shrink_Small-Outline_Package.

[29] http://www.siliconfareast.com/ic-package-types.htm.

[30] http://www.vita.com/fmc.html.

[31] Xilinx, "SP601 Hardware User Guide", http://www.xilinx.com/support/documentation/boards_and_kits/ug518.pdf, 2010.

[32] Stephen Brown and Zvonko Vranesic, "Fundamentals of Digital Logic with Verilog Design", McGraw-Hill, pag. 311, 2003.

[33] Pong P. Chu, "FPGA PROTOTYPING BY VERILOG EXAMPLES", Wiley, pagg. 56 - 57, 2008.