ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

FACOLTA' DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

DIPARTIMENTO DEIS

TESI DI LAUREA

in

Calcolatori Elettronici LS

SVILUPPO E SPERIMENTAZIONE DI INTERFACCIA USB2 PER LA COMUNICAZIONE CON SISTEMI DI VISIONE STEREO BASATI SU FPGA

CANDIDATO
Arman Tanna Nazmul

RELATORE:
Chiar.mo Prof. Giovanni Neri

CORRELATORE/CORRELATORI
Prof. Stefano Mattoccia
Ing. Nanni Davide

Anno Accademico 2010/11

Sessione II

Indice

INTRODUZIONE....5
1. IL PROTOCOLLO E L'INTERFACCIA USB....8
1.1 Motivazioni....8
1.2 Obiettivi....12
1.3 Vantaggi e Svantaggi....12
1.4 Evoluzione dell'interfaccia USB....14
1.4.1 Concorrenza con Firewire ed Ethernet....16
1.4.2 USB 3.0....17
1.4.3 Concorrenza con Thunderbolt....19
1.5 Trasferimenti USB....20
1.5.1 Comunicazioni USB....20
1.5.1.1 Strati di comunicazione USB....24
1.5.2 Il protocollo USB....27
1.5.2.1 Formato dei campi di pacchetto....27
1.5.2.2 Formato dei pacchetti....29
1.5.2.3 Formato delle transazioni....31
2. ARCHITETTURA E COMPONENTI HARDWARE DEL SISTEMA....36
2.1 Il controller USB CY7C68013....37
2.1.1 Caratteristiche principali....38
2.1.1.1 CPU 8051....38
2.1.1.2 Memoria interna....39
2.1.1.3 Endpoint buffers....40
2.1.1.4 Input/Output....41
2.1.1.5 Sistema di Interrupt....42
2.1.1.6 Reset....43
2.1.1.7 General Programmable Interface....43
2.1.1.8 Modalità Slave Fifo....44
2.1.2 I2C....46
2.1.3 Circuito di generazione del clock....46
2.1.3.1 PLL....46
2.2 Sensori di immagini....48
2.2.1 MT9V032....49
2.3 Deserializer....52
2.3.1 Caratteristiche principali....52
2.4 Microcontrollore....54
2.5 FPGA....55
2.5.1 Xilinx Spartan 3....56
3. TOOL E COMPONENTI SOFTWARE DEL SISTEMA....59
3.1 Tool di sviluppo Cypress....59
3.2 VHDL....62
3.2.1 Xilinx ISE....63
3.2.2 Picoblaze....67

3.2.2.1 Architettura....67
3.2.2.2 Programmazione....69
3.3 Libreria....69
3.3.1 Libusb....70
3.3.1.1 Libusb win32....72
3.3.1.2 Libusb windows_backend....73
3.3.2 OpenCV....74
3.3.2.1 Organizzazione della libreria....75
3.3.2.2 Strutture dati....76
3.4 Microsoft Visual Studio....78
4. ASPETTI REALIZZATIVI E SPERIMENTAZIONE....80
4.1 Lo stampato del prototipo....80
4.2 Caricamento driver....83
4.2.1 Driver Cypress....83
4.2.2 Il file CyUSB.inf....84
4.2.3 Riconoscimento della periferica....86
4.3 Sviluppo e caricamento firmware....87
4.4 Test....90
4.4.1 Test delle due versioni di libusb....90
4.4.1.1 Test libusb 1.0....91
4.4.1.2 Test libusb win32....94
4.4.1.3 Test libusb windows_backend....96
4.4.1.4 Test libusb 1.0 con thread....98
5. ASPETTI PRATICI E SVILUPPO....101
5.1 Installazione driver Cypress....101
5.2 Caricamento firmware....106
5.3 Installazione driver winusb....107
5.4 Sviluppo libreria....110
5.4.1 Funzioni: about, init, connect e disconnect....112
5.4.2 Funzioni: bulk transfer, write, get e set properties 114
5.4.3 Funzioni: get_frame e get_frame_and_save....117
5.4.4 Sviluppo applicazione utente....119
CONCLUSIONI E SVILUPPI FUTURI....120
BIBLIOGRAFIA....123

INTRODUZIONE

La realizzazione dei sensori, in grado di fornire immagini in tre dimensioni, ha avuto grande interesse in ambito scientifico ed industriale per via delle innumerevoli applicazioni in cui tali sensori possono essere utilizzati. Gli strumenti teorici utilizzati a questo scopo sono parzialmente acquisiti e in continua evoluzione così come i componenti pratici che ne permettono la realizzazione fisica. Il fulcro fondamentale su cui si basano questi dispositivi è la visione stereoscopica, finalizzata alla ricostruzione della struttura tridimensionale di una scena osservata da più telecamere. In particolare, il nostro interesse si è focalizzato sulla visione binoculare che utilizza solamente due sensori d'immagine.

Il principio utilizzato dalla visione stereo è la triangolazione, con lo scopo di mettere in relazione la proiezione di uno specifico punto della scena su due o più piani immagine delle telecamere; i punti ottenuti con questa procedura vengono chiamati punti omologhi. L'individuazione dei punti omologhi (problema noto come problema delle corrispondenze o "matching") consente, attraverso la conoscenza di opportuni parametri (ottenibili mediante il processo di calibrazione), di risalire alla posizione nello spazio 3D del punto considerato.

Il matching può essere eseguito "on board", cioè direttamente sul dispositivo che si occupa dell'acquisizione immagini oppure da un Host, tipicamente un PC.

Un dispositivo molto utilizzato ed efficiente per eseguire elaborazione di immagine on board è l'FPGA (Field Programmable Gate Array). In questi componenti elettronici digitali la funzionalità è programmabile via software direttamente dall'utente finale, consentendo così la diminuzione dei tempi di progettazione, di verifica mediante simulazioni e di prova sul campo dell'applicazione.

Di fondamentale importanza risulta essere anche la scelta del protocollo di trasmissione con l'Host PC, sia in termini di velocità di trasmissione sia in termini di diffusione sul mercato.

L'evoluzione tecnologica nel campo elettronico ed informatico ha portato alla nascita nel 1995 dell'architettura USB. Tramite l'interfaccia USB è possibile connettere facilmente ad un'unica unità di elaborazione un elevato numero di dispositivi e di avere a disposizione un canale con un'ampia banda di trasmissione.

La presente tesi si colloca all'interno di un progetto complesso finalizzato allo sviluppo di una telecamera stereo con elaborazione dei dati on board e dotata di interfaccia USB 2.0.

L'obbiettivo specifico del lavoro svolto è stato lo studio con il conseguente sviluppo software di un particolare controller USB 2.0, finalizzato alla successiva integrazione del medesimo nel progetto complessivo.

Nel Capitolo 1 si introducono i concetti e gli aspetti fondamentali, ovvero una panoramica riguardante le caratteristiche principali dell'interfaccia USB, evidenziandone i pregi ed i limiti, non tralasciando l'evoluzione che tale interfaccia ha subito.

Il Capitolo 2 è dedicato interamente alla spiegazione dell'architettura hardware complessiva del progetto. E' stato analizzato il controller Cypress FX2, con una descrizione dettagliata che permette di introdurne il funzionamento basilare, la telecamera stereo e la descrizione dei singoli dispositivi che la compone.

I tool e componenti software utilizzati nel progetto è descritto nel Capitolo 3. E' stato esaminato la programmazione del FPGA, la descrizione della libreria Libusb e vengono trattati tutti gli aspetti rilevanti software per una corretta interoperabilità tra host e controller.

Nel capitolo 4 è stato riportato in dettaglio quelli che sono stati gli aspetti realizzativi della telecamera stereo, la comunicazione con il controller e la

descrizione dei risultati emersi durante la prima fase delle prove pratiche di sperimentazione.

Infine, nel capito 5 sono stati ripercorsi in dettaglio gli aspetti pratici dell'installazione dei driver, l'implementazione di una libreria ed un'applicazione utente.

1. IL PROTOCOLLO E L'INTERFACCIA USB

L’Universal Serial Bus (USB) è uno standard di comunicazione tra PC e periferiche la cui nascita risale al Gennaio 1996. Durante la WinHEC95 (Windows Hardware Engineering Conference) un consorzio di produttori, composto da Microsoft, IBM, Compaq, Digital e molti altri, ha posto le basi per la definizione di questo standard per risolvere la problematica relativa alla connessione di molteplici dispositivi al PC.

Figura 1.1: simbolo dell'USB

1.1 Motivazioni

Nessuna delle interfacce di comunicazioni utilizzate nei pc, quali porte seriali, parallele, porte per joystick, mouse e tastiere, porte midi, soddisfaceva uno dei requisiti principali, ovvero il plug-and-play (termine mutuato dall'inglese che tradotto letteralmente significa "collega e usa"). Inoltre, molte di queste connessioni erano state concepite per soddisfare solamente un sottoinsieme delle risorse disponibili, come ad esempio interrupt hardware e canali DMA. L’USB doveva essere la nuova interfaccia capace di sopperire a queste mancanze.[1.1]

Per capire meglio quali sono state le motivazioni che hanno spinto verso l'introduzione dell'interfaccia USB dovremmo comprendere le tipologie di trasmissioni presente nel campo informatico.

In informatica con il termine trasmissione si intende il processo e le modalità/tecniche finalizzate all'invio di informazione tramite segnali su un canale fisico di comunicazione da un mittente ad un destinatario. Essa è resa possibile da apparecchiature elettroniche di ricetrasmissione ovvero da un trasmettitore e da un ricevitore poste agli estremi del canale di comunicazione e che insieme definiscono genericamente il sistema di telecomunicazioni. In generale questo processo subisce le caratteristiche trasmissive del canale di comunicazione stesso che altera più o meno l'informazione trasmessa e vincola allo stesso tempo i rispettivi parametri di trasmissione. [1.2]

Le trasmissioni si suddividono in:

- **trasmissione analogica,** si indica un particolare tipo di trasmissione in cui l'informazione da trasmettere sotto forma di segnali sul canale di comunicazione dal mittente al/ai destinatario/i rimane costantemente in forma analogica a partire dalla sorgente analogica. Una trasmissione analogica può essere implementata su qualunque tipo di mezzo trasmissivo ovvero sia in comunicazioni cablate (comunicazioni elettriche e comunicazioni ottiche) sia su comunicazioni radio. [1.3]
- **trasmissione digitale o numerica,** si indicano quelle trasmissioni di dati in cui i dati stessi sono rappresentati in forma digitale cioè numerica ovvero sequenze di bit. La trasmissione digitale è possibile a partire da sorgenti reali analogiche attraverso una conversione analogico-digitale dei dati analogici. A partire da questa operazione si associano poi usualmente operazioni di elaborazione digitale del segnale numerico quali la codifica di sorgente volte alla compressione dei dati, operazioni di codifica di canale volte all'introduzione di ridondanza per la rilevazione e/o la

correzione dell'errore a valle in ricezione. Infine prima della trasmissione sul canale, che in genere è un canale ad onde continue, si attua una modulazione numerica per la conversione digitale-analogica del segnale digitale. In ricezione si opera in maniera complementare con le rispettive operazioni inverse di demodulazione numerica, decodifica di canale, decodifica di sorgente ed infine conversione digitale analogica. [1.4]

I vantaggi delle trasmissioni digitali sono proprio nella possibilità di effettuare le operazioni di elaborazione di cui sopra (codifica di canale e di sorgente) per far fronte alle problematiche di trasmissione introdotte dal canale di comunicazione quali gli errori di trasmissione e/o le limitazioni di banda offerte da quest'ultimo rendendo quindi la trasmissione più efficiente ed affidabile. Tali operazioni non sono infatti possibili su segnali di tipo analogico.

In base al numero di destinatari queste tipologie di trasmissione possono essere a loro volta:

- *Unicast*, il destinatario è solo uno di quelli raggiunti dal segnale quindi la comunicazione è punto-punto;
- *Multicast*, i destinatari devono essere solo alcuni di quelli raggiunti dal segnale quindi la comunicazione è punto-multipunto;
- *Broadcast*, i destinatari sono tutti quelli raggiunti dal segnale quindi la comunicazione è punto-tutti.

Dal punto di vista del trasporto dell'informazione sul canale una comunicazione può essere:

- *simplex*, ovvero mono-direzionale (ad es. radiodiffusione e telediffusione);
- *half-duplex*, ovvero bidirezionale, ma solo un utente alla volta (ad es. walkie-talkie);
- *full-duplex*, ovvero bidirezionale contemporanea tra due utenti (ad es. telefono). [1.2]

In ambito strettamente informatico si distinguono invece:

- **trasmissione parallela,** è una tipologia di collegamento utilizzata per trasmettere informazioni. Viene utilizzata prevalentemente per trasmettere informazioni digitali anche se esistono dei cavi capaci di trasmettere parallelamente informazioni di carattere analogico. Nella trasmissione parallela vengono utilizzati più conduttori per trasmettere simultaneamente informazioni. Quindi un cavo che effettua una trasmissione parallela a *n* bit è formato da almeno n conduttori separati. [1.5]
- **trasmissione seriale,** è una modalità di comunicazione tra dispositivi digitali nella quale le informazioni sono comunicate una di seguito all'altra e giungono sequenzialmente al ricevente nello stesso ordine in cui le ha trasmesse il mittente. [1.6]

  Esistono tre modalità di trasmissione seriale:

  1. sincrono: generatori di clock separati con capacità di aggancio di fase del ricevente;
  2. asincrono: generatori di clock separati presso mittente e destinatario, di frequenze simili e sincronizzati ad ogni carattere;
  3. isocrono: unico segnale ausiliario di sincronizzazione (clock) comune a mittente e destinatario. [1.7]

Rispetto alla trasmissione seriale, la trasmissione parallela risulta avere prestazioni più alte, a parità di frequenza, ma ovviamente è anche più costosa. Infatti normalmente per comunicazioni a lunga distanza si preferisce utilizzare delle trasmissioni seriali, poiché i cavi necessari alla trasmissione parallela renderebbero inconveniente economicamente il progetto.

L'Universal Serial Bus, appunto, è uno standard di comunicazione seriale, bidirezionale ed unicast che consente di collegare diverse periferiche ad un computer. È stato progettato per consentire a più periferiche di essere connesse

usando una sola interfaccia standardizzata ed un solo tipo di connettore, e per migliorare la funzionalità plug-and-play consentendo di collegare o scollegare i dispositivi senza dover riavviare il computer (hot swap). [1.8]

1.2 Obiettivi

L'obiettivo primario, una volta definite le specifiche, era favorirne il più velocemente possibile la diffusione, ed in tal senso si è messa a disposizione questa interfaccia a tutti i produttori hardware. Spesso infatti una nuova interfaccia, pur avendo evidenti vantaggi a livello tecnico, non viene adottata in modo immediato ed indolore da parte degli utenti, poiché essi non vedono di buon occhio una modifica così radicale del loro ambiente fino a quando non si evidenziano i problemi che rendono necessario un upgrade. Al giorno d'oggi questa interfaccia ha raggiunto una diffusione tale da avere tutti i calcolatori dotati di USB, e non ci sono motivi per pensare che l'interfaccia non continui ad essere usata diffusamente.

Comunque, tra gli obiettivi principali c'erano anche l'espansione per PC facile da usare (unico modello di cavi e connettori, periferiche ri-configurabili e collegabili dinamicamente), soluzione di basso costo ed elevate prestazioni, supporto completo per elaborazioni real-time, protocollo flessibile, robustezza (inserimento e rimozione "a caldo" dei connettori), architettura upgradable. [1.9]

1.3 Vantaggi e Svantaggi

La diffusione di un prodotto, e in questo caso dell'interfaccia USB, è determinato dai vantaggi che esso ha sia dal punto di vista dell'utilizzatore finale che da quello dello sviluppatore.

Entrando nell'ottica dell'utente finale, l'USB garantisce una grande facilità d'uso (anche grazie al supporto del Plug'n Play), flessibilità e basso consumo di energia (tema di grande attualità, in particolare nelle applicazioni mobile).

In alcune periferiche, come le chiavette USB dotate di memoria Flash,

l’alimentazione (fino a 5V con corrente 500mA) è fornita dall’interfaccia, senza la necessità di alcun alimentatore esterno.

Lo sviluppatore d’altro canto beneficia anch’esso di queste peculiarità, non dovendosi preoccupare di problemi legati alla tipologia del cavo utilizzato, del connettore e più, in generale, del problemi hardware del protocollo, dovendo solamente configurare in modo opportuno il controller esterno in base alle proprie esigenze.

La facilità d’uso è dovuta sicuramente dalla grande eterogeneità di dispositivi USB in commercio che non disorientano l’utente, come invece potrebbe accadere con una grande molteplicità di interfacce (che porrebbero anche problemi a livello di operabilità come può accadere con i caricatori dei telefoni mobili, problema che sembra superato dall’adozione del connettore micro-usb quale standard comune).

I sistemi operativi inoltre configurano in modo automatico (quando possibile) la periferica all’atto della connessione caricando il driver opportuno, oppure l’installazione è a carico dell’utente solo la prima volta.

Da non sottovalutare la grande flessibilità offerta: se infatti le porte USB disponibili non fossero sufficienti, è possibile creare una struttura a sette livelli mediante l’utilizzo di hub, lavorando cosi con un numero elevato di periferiche (al massimo 127). Quindi, una periferica può essere connessa all’host o in modo diretto oppure attraverso un hub che ha il semplice compito di moltiplicare le porte disponibili. [1.10]

Tra gli svantaggi c'è che, come in ogni rete, è fissato un limite superiore (fissato in 5 metri) per la distanza tra l’hub principale e la periferica connessa. Essendo al massimo 7 i livelli sovrapponibili con hub, la distanza massima tra il primo e l’ultimo hub sarà quindi di 30 metri. Questa limitazione in realtà non costituisce un ostacolo in particolar modo in ambiente lavorativo, quale un ufficio, lo è invece nel caso di necessità di collegamento tra periferiche di uno stesso edificio, nel qual caso si dovranno adottare tecniche differenti.

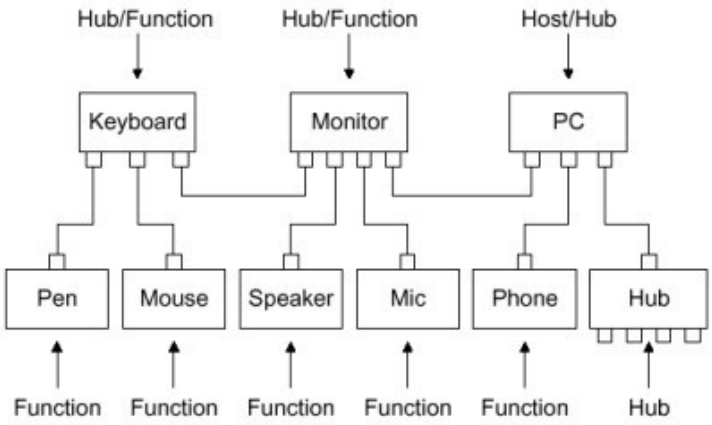


Figura 1.2: schema di connessione tra periferiche e HUB

Un altro aspetto negativo del USB è che il protocollo USB è in grado di gestire fino a 127 periferiche connesse ad albero, occorre però tenere presente che la banda viene suddivisa tra i vari dispositivi e la priorità dipende dalla posizione nella catena.

L'USB inoltre è pensato per comunicazioni punto-punto, tipicamente tra host e periferica, ed è inadatto alle comunicazioni peer-to-peer, in cui una molteplicità di host parla contemporaneamente con una molteplicità di periferiche, problema parzialmente superabile con USB OTG (On-The-Go). Non esiste comunque un sistema per mandare messaggi in modo simultaneo a più periferiche collegate sul bus, ovvero non è possibile fare broadcast. [1.11]

1.4 Evoluzione dell'interfaccia USB

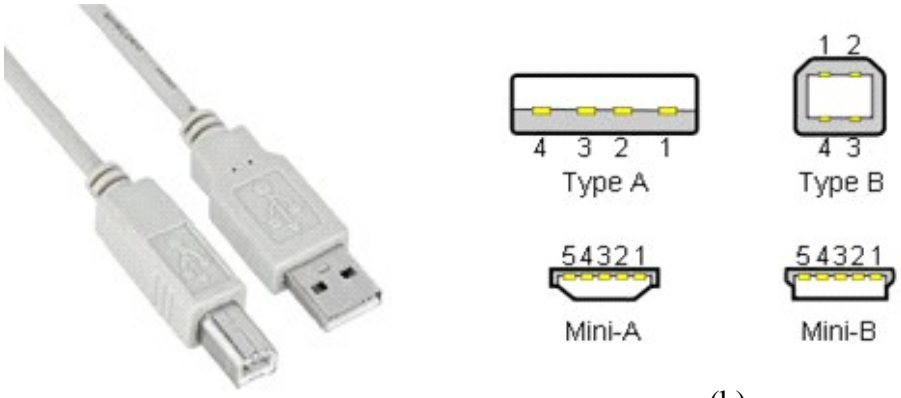
Dopo l'uscita della versione 1.0, datata Gennaio 1996 e capace di una velocità di soli 1.5 Mbit/sec utile al solo utilizzo di dispositivi lenti quali mouse e tastiere, nel settembre 1998 fu rilasciata la versione 1.1 che rese disponibile un nuovo tipo di trasferimento (interrupt) consentendo di raggiungere una velocità di 12 Mbit/sec (Full Speed).

Dopo che la versione 1.1 guadagnò una certa popolarità, si sentì il bisogno di una velocità superiore. La risposta a questa necessità fu l'uscita della versione 2.0

nell'Aprile 2000, la cui novità maggiore, che avrebbe cambiato in modo forte e deciso l'utilizzo dell'interfaccia, fu l'aumento considerevole di velocità a 480 Mbit/sec (High Speed). [1.12]

Come spesso accade nel campo dell'informatica, una nuova versione di un'interfaccia (ma più in generale di un componente) deve essere compatibile con le versioni precedenti, e l'USB in tal senso non fa eccezione. Un hub USB infatti supporta sia connessioni con dispositivi 1.0 che 1.1 in modo intercambiabile. Per poter trasferire dati in High Speed , è necessario l'utilizzo di una periferica e di un host High Speed, ed anche eventuali hub intermediari devono essere compatibili.

Una porta USB può inoltre funzionare sia da controller che da normale dispositivo, grazie ad un'estensione dell'USB, chiamata USB-On-The-Go. L'OTG sta diventando uno standard sempre più utilizzato per connettere periferiche e dispositivi di ogni settore Hi-tech, soprattutto portatili. La novità consiste nel fatto che la comunicazione tra i dispositivi avviene senza che sia obbligatoria la presenza di un pc. [1.13]



(a) (b)

Figura 1.3: (a) connettori USB di tipo A e B; (b) Pin-Out USB

Per quanto riguarda i connettori, attualmente ve ne sono 5 (rimanendo tra quelli ufficiali, in quanto ne esistono molti altri non standard):

- USB di tipo A;
- USB di tipo B;

- Mini USB di tipo A;
- Mini USB di tipo B;
- Micro USB.

L'USB richiede un cavo schermato contenente quattro fili. Due di questi, D+ e D-, sono linee di comunicazione di dati differenziali. Tali segnali sono riferiti al terzo cavo, ovvero la massa (GND). Il quarto filo è chiamato VBUS (o USB Vcc), e porta una tensione nominale di 5V che può essere utilizzata per alimentare le periferiche.

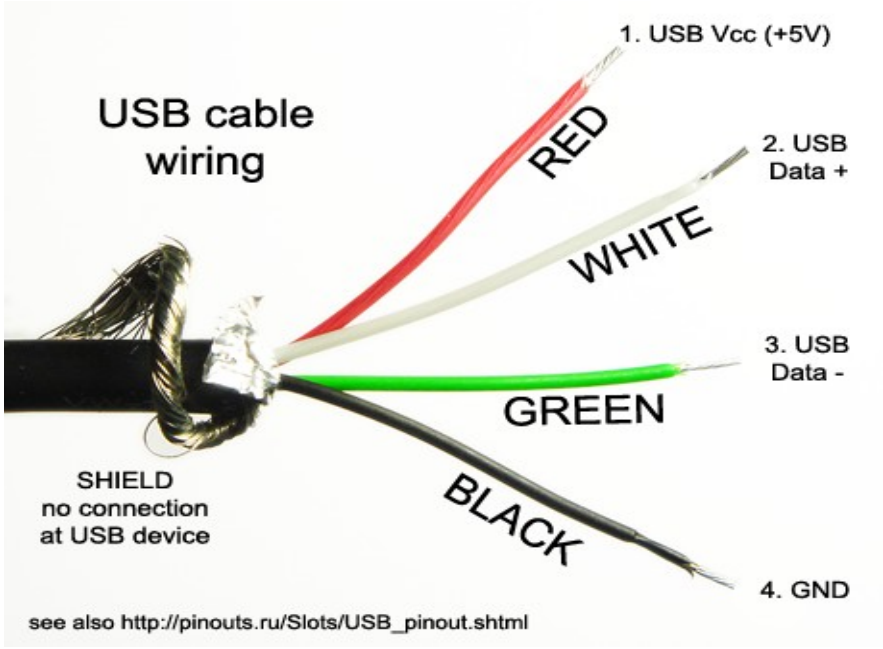


Figura 1.4: cablaggio dei cavi USB

1.4.1 Concorrenza con Firewire ed Ethernet

Il FireWire (nome con il quale è noto lo standard IEEE 1394), di proprietà della Apple Computer, ma conosciuto anche con il nome commerciale i.Link datogli dalla Sony, è un'interfaccia standard per un bus seriale. Questa supporta due diverse modalità di trasferimento dati: asincrona e isocrona. [1.14]

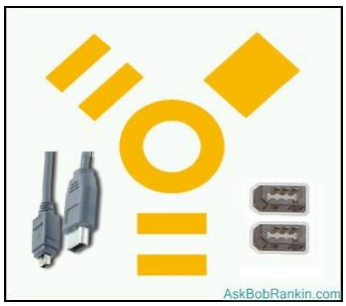


Figura 1.5: logo e connettori Firewire

L'incremento considerevole della velocità dell'USB 2.0 mise in diretta competizione con la Firewire 400 (IEEE 1394), capace di raggiungere i 400 Mbit/sec, e rese l'interfaccia utilizzabile per una gamma di dispositivi molto più ampia quali stampanti, scanner e dischi. In realtà la velocità utile dell'USB era inferiore rispetto alla Firewire, che si adatta molto bene ad applicazioni che richiedono comunicazioni molto veloci o broadcast verso più ricevitori.

L'USB inoltre ha dovuto confrontare anche con l'Ethernet, uno standard IEEE (802.3) introdotto nel commercio nel 1980 quando viene pubblicato anche la versione 1.0 dello standard stesso. Quest'ultimo può utilizzare cavi molto lunghi (100 volte più dell'USB), ma l'hardware richiesto è piuttosto complesso e costoso. [1.15]

1.4.2 USB 3.0

L'ultima versione dell'USB è la 3.0, la cui prima dimostrazione è stata effettuata nel settembre 2007. Tale versione riesce ad avere una velocità di trasferimento fino a dieci volte superiore della precedente 2.0 (4.8 Gbit/sec) che introduce il supporto di connessioni ottiche e la modalità "Superspeed" che, al netto dell'overhead del protocollo, dovrebbe riuscire a raggiungere un payload di 3.2 Gbit/sec.

L'evoluzione dell'interfaccia non si è focalizzata in maniera esclusiva sull'aumento della velocità ma anche sulla realizzazione di un'interfaccia priva di

cavi, chiamata Wireless USB, che incorpora le caratteristiche della versione 2.0 (fino ad una distanza massima di 3 metri) con la praticità della tecnologia wireless.

Figura 1.6: il connettore USB 3.0

L'altra novità è che USB 3.0 è una soluzione bi-direzionale, quindi in grado di leggere e scrivere dati simultaneamente. Questo è stato possibile aggiungendo altri 4 connettori al vecchio modello: due per la trasmissione e due per la ricezione.

Sotto il profilo energetico, l'ente certificatore ha incrementato la potenza massima gestibile, da 500mA a 900mA. In pratica i pc potranno supportare dispositivi ad alto consumo, gli hub saranno in grado di alimentare un maggior numero di periferiche, e i dispositivi con batteria ricaricabile potranno essere caricati più velocemente.

Il tutto dovrebbe essere accompagnato anche da una maggiore efficienza. Ad esempio i controller USB 2.0 accedevano continuamente ai dispositivi, per tenersi pronti in caso di trasferimento dati; con l'USB 3.0 saranno le periferiche stesse a spedire un segnale all'avvio delle operazioni. [1.16]

1.4.3 Concorrenza con Thunderbolt

La principale concorrente di USB a livello di interfaccia sembra essere Thunderbolt, tecnologia sviluppata da Intel (con il nome Light Peak) in

collaborazione con Apple, che mira ad imporsi come unica interfaccia sul mercato, compito che sembra essere molto arduo e difficile.

Figura 1.7: il logo e il connettore Thunderbolt

La larghezza di banda offerta dalla prima generazione di tale tecnologia raggiunge i 10 Gbit/s bidirezionali. Ogni connettore Thunderbolt porta due canali, quindi in teoria ogni connettore è in grado di ricevere e trasmettere 20 Gbit/s. Lo standard è stato sviluppato per poter essere espanso fino a 100 Gbit/s e lo standard prevede l'utilizzo dei cavi in fibra ottica per raggiungere le larghezze di banda più elevate. Il connettore è identico al connettore mini DisplayPort sviluppato da Apple, e difatti Thunderbolt è compatibile con i monitor DisplayPort. Lo standard combina i protocolli di trasferimento dati DisplayPort e PCI Express in un unico flusso dati; questo permette al connettore di gestire sia monitor che generiche periferiche. Nell'idea dei promotori dello standard Thunderbolt dovrebbe sostituire i diversi connettori presenti nei computer diventando l'unico connettore presente nel computer per il trasferimento dei dati. Lo standard gestisce fino a 6 dispositivi in cascata ed è in grado di alimentare i dispositivi dato che fornisce un cavo di alimentazione integrato che è in grado di fornire al massimo 10 Watt di potenza. Ogni cavo può essere lungo al massimo 3 metri.

Thunderbolt offre due canali di comunicazione indipendenti, in teoria ogni singolo canale di comunicazione offre tre volte la velocità massima del protocollo USB 3.0. In pratica, la bassa latenza nell'ordine degli 8 ns anche alla fine della

catena di comunicazione e il leggero protocollo di comunicazione PCI Express, offrono prestazioni vicine a quelle massime teoriche. Durante la presentazione ufficiale Intel ha mostrato la copia di un file da hard disk allo stesso hard disk a velocità maggiori di 6.4 Gbit/s con apparecchiature commerciali. [1.17]

1.5 Trasferimenti USB

Nei trasferimenti USB normalmente intervengono due elementi:

- HOST, dispositivo master (PC), l'unico autorizzato ad avviare una comunicazione sul bus;
- DEVICE, ogni dispositivo nel sistema USB che non è un host è necessariamente una periferica. Esistono periferiche autoalimentate oppure alimentate dal bus e periferiche full-speed (12Mbps) o low-speed (1.5Mbps).

Nei trasferimenti può essere presente un terzo elemento che è l'hub, come nelle reti di computer, che fornisce un punto di connessione a più periferiche partendo da una sola porta USB.

1.5.1 Comunicazioni USB

La comunicazione USB è concettualmente divisibile in due macro-categorie: comunicazioni all'atto della connessione del dispositivo e in fase di trasferimento dati.

Subito dopo la connessione di una periferica all'host inizia la prima fase, che viene chiamato enumerazione, durante la quale il computer interroga la periferica per avere maggiori informazioni su di essa in modo da identificarla, preparandosi a ricevere dati.

Inizialmente viene collegata la periferica ad un hub USB, che fornisce corrente alla porta con la periferica che viene a trovarsi nello stato "Powered". Successivamente, mediante una differenza di tensione tra le linee D+ e D-, l'hub

riconosce che è stata connessa una periferica. La periferica altera la tensione nelle sue linee differenziali: se la linea positiva (D+) viene portata sopra la tensione di soglia per più di 2,5μs è connessa una periferica veloce (2.0), viceversa se è la linea differenziale negativa (D-) a essere portata sopra soglia vuol dire che la periferica è a bassa velocità (1.0). Notare che nel funzionamento a regime una delle due linee D è sempre superiore a una tensione di soglia mentre l'altra è vicina ad una tensione di riferimento. L'informazione viene propagata attraverso la variazione della tensione differenziale.

Tutto ciò funziona nel seguente modo: viene scatenato un evento (tramite un endpoint di tipo Interrupt) che segnala all'host la connessione della periferica. Viene inviata una richiesta di tipo Get_Port_Status dall'host per avere maggiori informazioni su di essa. Dopo aver determinato se la periferica è in full/high speed, viene inviato un reset generale.

Al termine del reset, la periferica sarà nello stato "Default" ed è pronta a rispondere ad eventuali richieste di identificazione (ad esempio come Get_Descriptor) al Control Endpoint di default (situato all'indirizzo 00h) che permettono all'host di reperire informazioni, ad esempio sulla dimensione massima dei pacchetti per quella pipe.

A questo punto, viene assegnato dall'host un indirizzo unico (Set_Address) con lo stato del dispositivo che si modifica in "Addressed". Tale indirizzo è valido fino a quando non viene disconnesso o il sistema riavviato e non è garantito che al prossimo collegamento lo stesso indirizzo sia assegnato alla medesima periferica.

Viene inviata un'ulteriore richiesta Get_Descriptor al nuovo indirizzo. Il Device Descriptor è una struttura contenente tutte le informazioni che caratterizzano la configurazione della periferica, quale dimensione massima dei pacchetti per l'Endpoint 0.

Dopo aver finito di interrogare la periferica, viene selezionato uno dei driver presenti oppure viene richiesta l'installazione nel caso non ne vengano trovati di

compatibili.

Infine il device driver invia una richiesta Set_Configuration per caricare una particolare configurazione del dispositivo (alcuni supportano solamente una configurazione). Lo stato della periferica è ora “Configured”.

Quindi nella seconda fase di comunicazione l’host interroga ciclicamente le periferiche, seconda una certa schedulazione, inviando un pacchetto dati contenente i parametri:

- tipo di transazione;
- direzione della transazione;
- indirizzo della periferica USB interessata alla transazione;
- EndPoint number.

Ogni dispositivo che riceve il pacchetto dati USB (token packet):

- decodifica il campo indirizzo stabilendo se è l'interessato;
- definisce il verso della transazione.

L'entità che invia il pacchetto dati può rispondere con dati significativi oppure specificare che non ha nessun dato da trasmettere. Il ricevitore da parte sua risponde con un pacchetto dati di handshake per notificare al trasmittente l'avvenuta ricezione, e quindi il successo della transazione.

La comunicazione logica tra client software (host) e funzione della periferica è effettuata tramite pipes e questo deve avvenire prima che il trasferimento si possa realizzare. Quindi occorre che l’host crei una sorta di associazione, ovvero una pipe, durante l’enumerazione. Un pipe è un’associazione tra uno specifico endpoint sulla periferica e l’appropriato software sull’host.

Un endpoint è la sorgente o la destinazione dei dati trasmessi sul cavo USB. Gli Endpoint sono uno degli elementi fondamentali di tutti i trasferimenti USB, in quanto su di essi viene convogliato tutto il traffico dati. In sostanza un endpoint

non è altro che un buffer contenente byte. Un'interfaccia è composta da endpoints raggruppati. Tutti gli endpoint, fatta eccezione per l'Endpoint 0, possono supportare un flusso di dati unidirezionale.

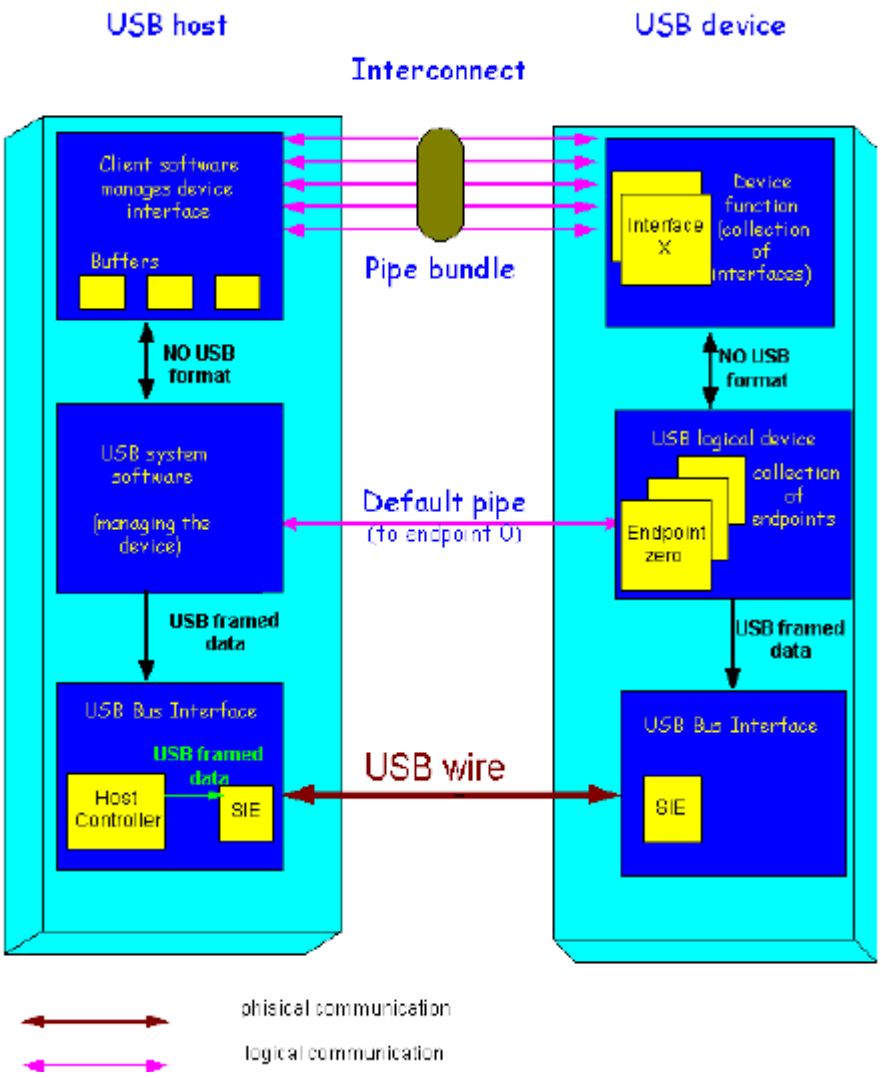


Figura 1.8: flusso di comunicazione USB

Il numero massimo di endpoint per una periferica High o Full Speed è di 30 indirizzi (quindi ad esempio 15 endpoint configurati sia IN che OUT). Il client software trasmette dati tra il buffer dell'host e l'endpoint nella periferica gestendo la specifica interfaccia.

La comunicazione può avvenire in due direzioni decisi dall'Host:

- OUT: flusso di dati da host a device,
- IN: flusso di dati da device a host.

1.5.1.1 Strati di comunicazione USB

Sono tre gli strati di comunicazione. Il primo strato di comunicazione USB è lo strato fisico che è l'interfaccia fisica, ovvero il cavo USB, ed ha il compito di trasmettere e ricevere "0" ed "1" in modo corretto. La trasmissione di uno "0" avviene portando il potenziale del terminale D+ del cavo USB a livello basso e quello del terminale D- a livello alto, e viceversa per trasmettere un "1". I bit sono inviati partendo dal LSB. I dati vengono codificati e decodificati usando il metodo NRZI (Non Return to Zero Inverted – rappresenta i cambiamenti di stato logico con "1" e la permanenza di uno stato logico con "0").

Esistono alcuni tipi di segnali sul bus, individuati come casi speciali:

- Reset signaling: l'host può resettare la periferica inviando un SE0 (single ended zero), cioè portando entrambe i potenziali D+ e D- a livello basso per più di 2,5μs;
- Suspend signaling: l'host può forzare la periferica in suspend mode, in cui il dispositivo non risponderà al traffico USB. Si attiva il suspend mode quando il bus rimane inattivo per più di 3ms e per un tempo non superiore ai 10ms;
- Resume signaling: una periferica riprende le sue operazioni quando riconosce un segnale K ("0" differenziale per periferiche full speed e "1" differenziale per periferiche low speed) per un minimo di 20ms;
- EOP (End Of Packet) signaling: corrisponde alla trasmissione di SE0 per 2 cicli di clock, seguito da un segnale J ("1" differenziale per periferiche full speed e "0" differenziale per periferiche low speed) per un ciclo di clock.

Lo strato fisico (o physical layer) ha un modulo fisico presente nell'host e nei

devices responsabile della serializzazione e deserializzazione (conversione dello stream di dati seriale in uno parallelo) delle trasmissioni USB che si chiama SIE (Serial Interface Engine). Esso codifica i dati in uscita secondo la logica NRZI e decodifica i dati in ingresso. Genera il codice di controllo CRC (Cyclic Redundancy Check) per gli stream in uscita e verifica il CRC degli stream in ingresso. Ha inoltre il compito di rivelare il PID (packet's id – è un campo nel pacchetto USB che individua il tipo di dato), così come i segnali SOP, EOP, RESET e RESUME sul bus.

L'HC (Host Controller) dello strato fisico è l'elemento hardware più importante del sistema USB, colui che inizia tutte le transazioni, controlla gli accessi ed è il motore principale di controllo del flusso. Le sue funzioni fondamentali sono:

- Frame generation: l'HC è responsabile della partizione del tempo in unità di tempo di 1ms chiamate frame mediante l'emissione di uno SOF (Start Of Frame);
- Data Processing: gestione delle richieste di dati da e per l'host;
- Protocol Engine: gestione del protocollo USB a livello interfaccia;
- Error handling: gestione degli errori (Timeout, CRC, Overflow ecc.);
- Remote wakeup: possibilità di iniziare l'USB in suspend mode e rilevare un segnale di wakeup sul bus.

Il protocollo Engine Layer è lo strato intermedio nel modello di comunicazione. Si occupa della traduzione dei dati tra applicazioni (software dell'host e funzioni della periferica) e il protocollo USB. Assume denominazione diversa a seconda che sia riferito all'host USB (USB System Software) o alla periferica USB (USB Logical Device).

Il sistema software USB (USB system sw) è responsabile dell'allocazione della larghezza di banda, della gestione dell'energia fornita al bus oltre all'abilitazione delle periferiche per l'accesso al bus. Si compone di:

- Host Controller Driver (HCD): è un'interfaccia all'host controller;
- USB driver (USBD): il client software richiede dati dall'USBD nella forma di IRPs (I/O Request Packets) che consiste nella richiesta di invio/ricezione di dati attraverso un certo pipe; fornisce al client una descrizione generale della periferica che il software sta gestendo. È necessario per creare un enumeration process.

Invece, USB logical device è composto da un insieme di endpoints indipendent. Ad ogni endpoint è dato un indirizzo unico (endpoint number) in fase di progetto e anche l'USB logical device è unicamente indirizzato alla fine dell'enumeration process. La combinazione di questi indirizzi e della direzione dei dati (da o per l'endpoint) caratterizza completamente l'endpoint.

Tutte le periferiche USB devono supportare la comunicazione con il default pipe, il quale gioca un ruolo importante nel processo di enumerazione ed è l'unico canale di comunicazione al device nel momento del collegamento. Il default pipe è associato all'endpoint zero, che è l'unico in grado di eseguire comunicazione bidirezionale.

L'ultimo stato nel modello di comunicazione è lo strato applicazioni (o application layer) è costituito dal client software per quanto riguarda l'host e dalle funzioni dal lato device. Il client software gestisce le interfacce trasferendo dati dal suo buffer agli endpoint associati alle interfacce stesse. Il client software lavora con una specifica funzione di una periferica, indipendentemente dalle altre funzioni della periferica nel sistema.

1.5.2 Il protocollo USB

Le transizioni USB avvengono tramite pacchetti. Ogni transizione è composta da tre fasi:

- Token phase: l'host inizia il pacchetto indicando il tipo di transizione;

- Data phase: i dati attuali sono trasmessi tramite pacchetti. La direzione dei dati coincide con la direzione indicata nella fase di token trasmessa precedentemente;
- Handshake phase (opzionale): è il pacchetto inviato indicante il successo o il fallimento della transizione.

USB usa un polling protocol cioè, ogni volta che l'host vuole ricevere dati da una periferica le invia un token, se la periferica ha dati da inviare li invia e l'host risponde con un pacchetto di handshake invece se la periferica non ha dati da inviare, l'host emette un token verso la periferica successiva, e se l'host vuole inviare dati ad una periferica, le invia prima un token appropriato e il successivo pacchetto di dati. La periferica risponde con un pacchetto di handshake.

1.5.2.1 Formato dei campi di pacchetto

Nel protocollo USB sono differenti i campi dei pacchetti, tra cui:

- Sync Field che si colloca all'inizio del pacchetto. Permette alle periferiche riceventi di sincronizzare il clock interno con i dati in ingresso;
- PID (Packet Identifier Field), che è il campo contenente l'identità del pacchetto, è composto da 8 bit; i primi 4 sono usati per notificare l'id del pacchetto e i 4 successivi sono usati come check bits.

Il PID si divide in quattro gruppi principali:

- Tokens:
    1. Out token indica che i dati successivi saranno trasmessi dall'host alla periferica;
    2. In token indica che i dati successivi saranno trasmessi dalla periferica all'host;
    3. SOF token indica lo start del frame;

4. SETUP token indica che i dati successivi saranno inviati dall'host al device e contengono comandi di setup usati per la configurazione.

- Data: descrive il tipo di dato (pari, dispari, ad alta o bassa velocità, isocrono, ecc.);
- Handshake: usato nei pacchetti di handshake prima di indicare il fallimento o la riuscita del trasferimento. Può essere:
    1. ACK, il ricevitore riceve pacchetti liberi da errore;
    2. NAK, il ricevitore non è in grado di ricevere dati (overflow, ecc.) o il trasmettitore non è in grado di inviare dati (underflow, ecc.);
    3. STALL, lo specifico endpoint si è arrestato o il comando di SETUP non è supportato.
- Special che può essere di due tipi:
    1. PRE, ERR e SPLIT servono per abilitare il traffico con periferiche lente;
    2. PING è un piccolo pacchetto che viene inviato per verificare la disponibilità della periferica che, se pronta, risponderà con acknowledge.

Nel formato dei campi del pacchetto un altro campo è quello di indirizzo (address field) che è diviso in due campi a sua volta:

- Address field (ADDR) contiene l'attuale indirizzo della funzione (in genere è la periferica stessa), assegnato durante l'enumeration process. L'indirizzo della funzione è univoco e possono esserci 127 diversi indirizzi nel sistema;
- Endpoint Field (ENDP) contiene il numero di endpoint assegnato ad una funzione.

Un altro campo presente nel formato dei campi del pacchetto è FNF (Frame

Number Field), composto da 11 bit che indicano il numero del frame corrente. Il campo è contenuto solo nei token SOF che indicano l'inizio del frame Data field dove contiene il dato trasmesso nella transizione. Può contenere un massimo di 1023 bytes.

Invece campo CRC (CRC field) è usato per proteggere tutti i campi e i dati di un pacchetto (tranne il campo PID). È composto da 5 bit in un token packet e da 16 bit in un data packet.

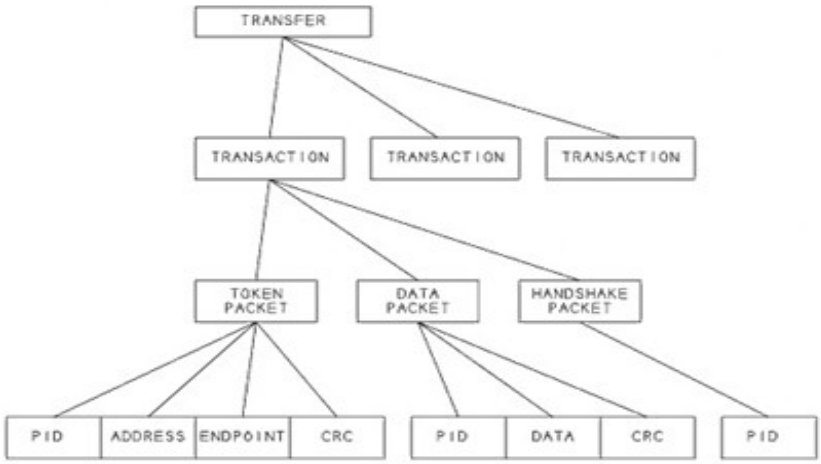


Figura 1.9: blocchi di un trasferimento USB

1.5.2.2 Formato dei pacchetti

Il formato dei pacchetti cambia a seconda del tipo, che può essere:

- Token packet: ogni transizione inizia con l'emissione di un token da parte dell'host. I campi ADDR e ENDP definiscono univocamente l'endpoint che deve ricevere i dati di SETUP o OUT o l'endpoint che deve trasmettere dati negli spostamenti IN;

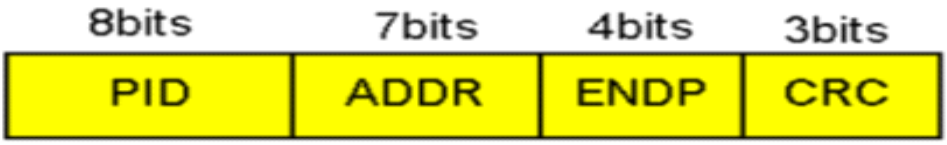


Figura 1.10: Token packet

- Start of Frame Packet: l'host emette un SOF ogni 1msec±0.0005msec. Il pacchetto contiene il campo del numero del frame. SOF può essere usato come trigger per processi di OUT isocrono;

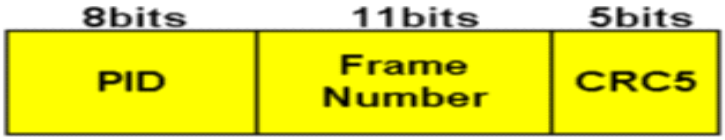


Figura 1.11: Start of Frame packet

- Data Packets: sono composti da PID (che indica che il pacchetto contiene dati), campo dei dati, e il codice CRC16 per proteggere i dati;

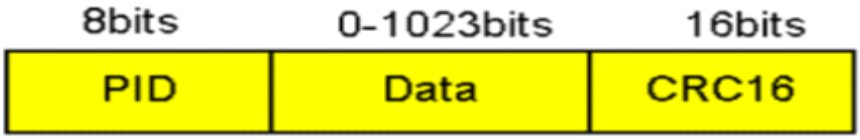


Figura 1.12: Data packet

- Handshake Packets: composto solo da PID indicante il risultato dello operazioni precedenti.

Figura 1.13: Handshake packet

Di seguito è riportata un esempio di trasmissione dati, come si può vedere per ogni blocco dati inviato (DATA0, DATA1 vanno alternati) è necessario un token packet e un handshake packet.

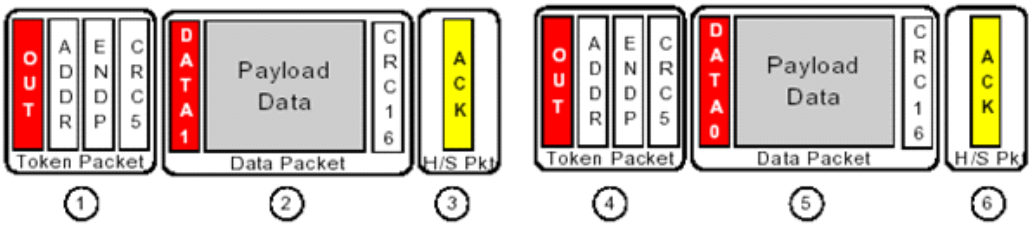


Figura 1.14: struttura dei pacchetti

1.5.2.3 Formato delle transazioni

Ci sono quattro tipi fondamentali di trasferimento USB. Il primo è control transfer che è usato per configurare le periferiche. La configurazione avviene durante il processo di enumerazione ma si può effettuare in ogni istante della comunicazione.

Il control transfer si compone di due o tre fasi: setup, data (opzionale) e status; ognuna di queste è composta da token, data, handshake. E' l'unica transazione che prevede un flusso di dati bidirezionale, questi dati hanno il compito di comandare e configurare il dispositivo, data la sua estrema importanza si implementa un sistema di controllo errori.

Comandi di setup sono:

- SET_ADDRESS: impone un indirizzo permanente ad una funzione;
- GET_DEVICE_DESCRIPTOR: comando per la ricezione di informazioni riguardo alla periferica;
- GET_CONFIGURATION_DESCRIPTOR: comando per la ricezione di informazioni riguardo alla configurazione di una periferica;
- GET_CONFIGURATION: l'host rileva quale configurazione è attiva al momento nella periferica;
- SET_CONFIGURATION: l'host impone una configurazione specifica alla periferica;

All'inizio di un'operazione di setup, l'host emette un SETUP token, seguito da un pacchetto di comando setup. La periferica risponde con un pacchetto ACK. Successivamente, il flusso dei dati viene emesso nella direzione indicata nel stage precedente. Possono essere eseguite più transazioni (IN o OUT, ma tutte nella stessa direzione), ognuna delle quali inizia con un IN/OUT token emesso dall'host e termina con l'emissione dell'eventuale handshake. Lo status stage riporta i

risultati dei precedenti stage all'host.

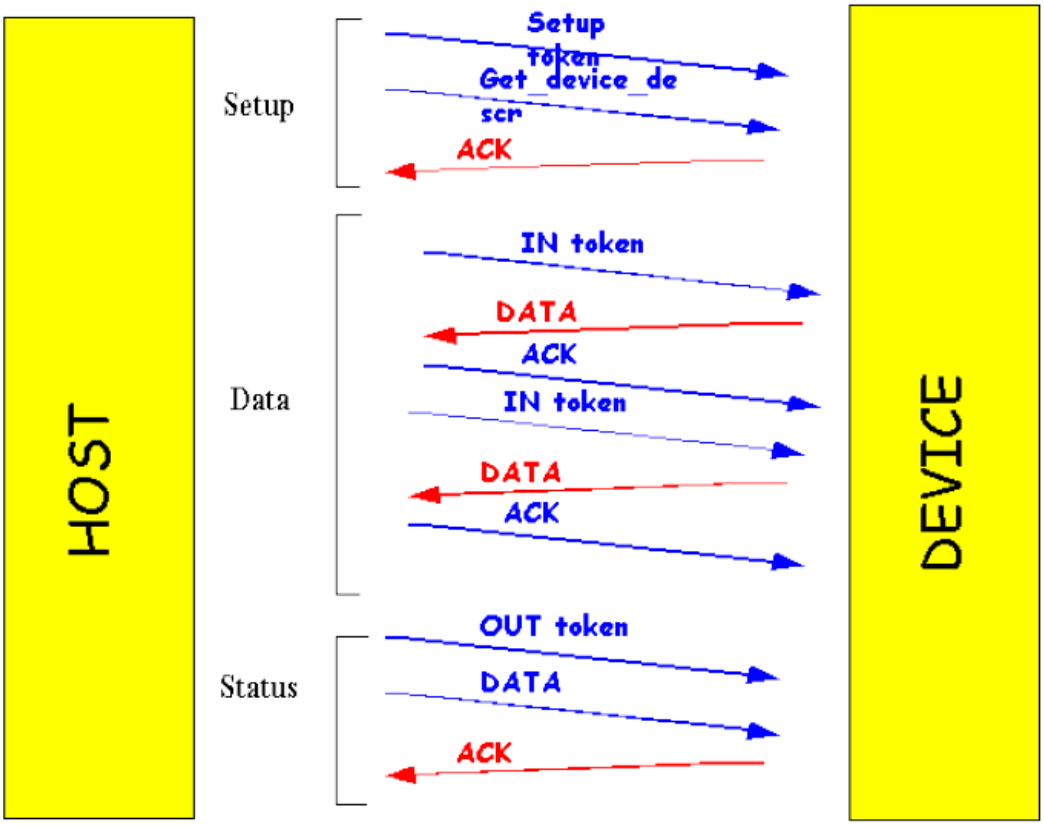


Figura 1.15: esempio di Control transfer

La seconda tipologia di trasferimento è il bulk transfer che consiste nell'invio massiccio di dati (stampanti, scanner ecc). La larghezza di banda allocata in ogni transizione varia in accordo con le disponibilità del bus nel momento del trasferimento. Garantisce pochi errori nella comunicazione.

Il bulk transfer è utilizzata per trasmissioni discontinue in cui è tollerata una latenza e non ci sono vincoli restrittivi sulla temporizzazione. Prevede pacchetti dati da 8, 16, 32 o 64 bytes (full speed) o 512 bytes (high speed). E' possibile la ritrasmissione in caso venga rilevato un errore ed è implementato il protocollo di Handshaking (il trasmettitore prima dell'invio di nuovi dati aspetta l'Handshake packet).

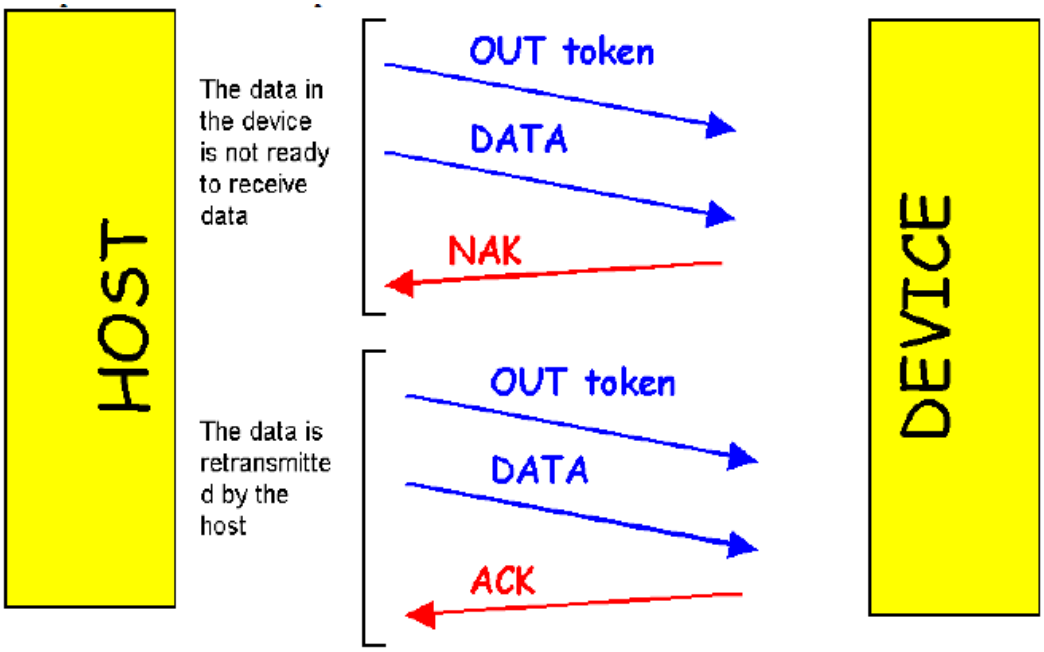


Figura 1.16: esempio di Bulk transfer

Nel Bulk Transfer la transazione si compone di tre fasi:

- invio di un token dall'host indicante la direzione della comunicazione;
- trasmissione dei dati;
- se non si rivelano errori, si invia l'handshake, altrimenti nessun pacchetto di handshake è inviato.

I trasferimenti di massa sono altamente affidabili grazie ai meccanismi di handshake e timeout. Se ci sono problemi nel sistema USB, l'host li individua e previene il blocco del sistema.

La terza modalità di trasferimento è a Interrupt ed è usato per periferiche che hanno bisogno di riportare brevi eventi (mouse, joystick ecc). In questo caso viene fatto un polling da parte dell'Host sull'interrupt endpoints che viene attivato dalla periferica quando è pronta per l'invio di nuovi dati.

Il Interrupt Transfer è un metodo di trasferimento simile al bulk transfer. L'host, che vuole conoscere lo stato di un interrupt, invia un IN token

all'appropriato endpoint. Se ci sono interrupt in attesa, la funzione invia dettagli a riguardo. L'host risponde con un segnale di ACK se il trasferimento ha avuto esito positivo. Se non ci sono interrupt in attesa, la funzione restituisce un NAK packet. Nel caso di errore nella funzione sarà inviato uno STALL packet. L'host invia un OUT token nel caso in cui vuole trasmettere dati alla periferica, seguito dal pacchetto di dati. Se il trasferimento è corretto l'host riceve un ACK o NAK o STALL packet. Se ci sono errori, non viene inviato handshake. Viene utilizzata una stream pipe (che è pipe senza un formato definito, supportano ogni tipo di trasferimento, controllabili sia da host che da periferica) unidirezionale e viene implementata una ritrasmissione in caso di verifica di errore.

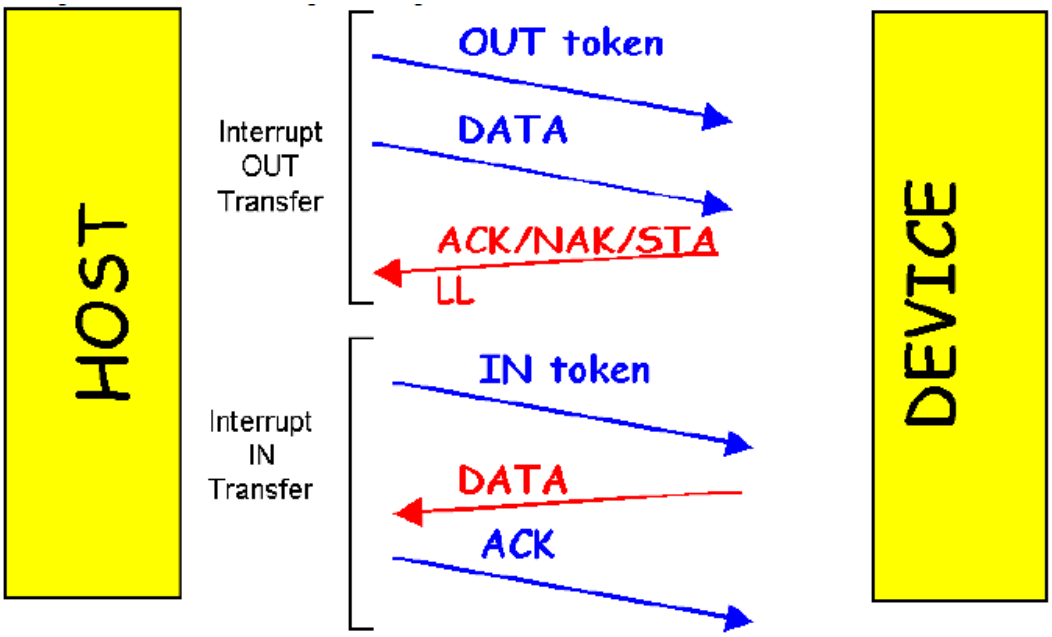


Figura 1.17: esempio di Interrupt transfer

Tale modalità prevede un pacchetto di dimensioni massime variabili dagli 8 byte per Low Speed, passando per i 64 byte in caso di Full Speed fino a 1024 byte per le periferiche High Speed.

Il quarto formato delle transazioni è il trasferimento isocrono che è utilizzata quando i dati devono arrivare a una velocità costante per potere mantenere una temporizzazione con il tempo reale; esempi tipici di dati isocroni sono musica e

voce. In questo tipo di trasferimento non è presente un sistema di ritrasmissione automatica in caso di errore altrimenti si perderebbe la sincronizzazione, tuttavia sono tollerati alcuni errori occasionali.

Nel trasferimento isocrono, a differenza di quello bulk, viene garantita una banda sul bus in termini di numero di byte ogni frame ma non è implementato l'Handshaking.

Nel Isochronous Transfer i trasferimenti sono composti da una o più fasi di transizione. L'host emette un IN token prima di ricevere dati dalla periferica, o un OUT token prima di inviare dati. Avviene l'emissione o la ricezione dei dati in base a quanto specificato precedentemente. Non esiste fase di handshake in questo tipo di trasferimento. [1.18]

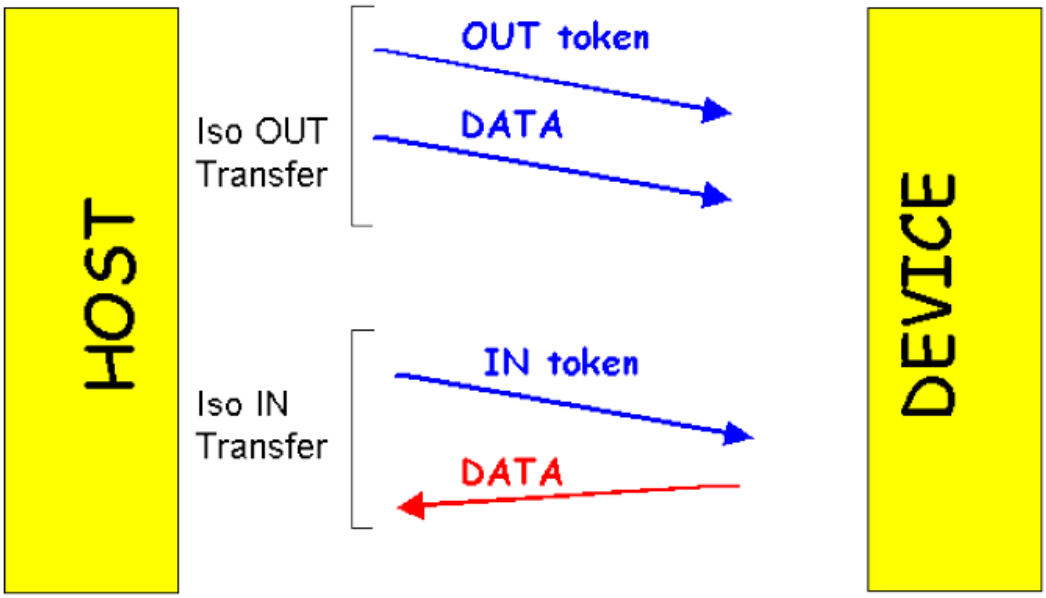


Figura 1.18: esempio di Isochronous transfer

Per applicazioni che scambiano dati in real-time con l'interfaccia USB i due tipi di trasferimento più idonei sono quindi Bulk ed Isochronous.

2. ARCHITETTURA E COMPONENTI HARDWARE DEL SISTEMA

Il progetto preso in esame prevede l'utilizzo di una telecamera stereo che dovrà interfacciarsi in modo corretto verso un generico host (un calcolatore) utilizzando l'USB, quale interfaccia di connessione, sulla quale transiteranno i dati acquisiti (ed anche eventuali parametri di configurazione inviati dal PC) mediante i due sensori d'immagine della telecamera che verranno successivamente gestiti in maniera opportuna dal calcolatore, visualizzando a video, per esempio, le immagini acquisite mediante librerie di computer vision come OpenCV.

E' meglio comprendere una descrizione, anche non dettagliata, del hardware per riuscire a capire meglio in un passo successivo sia il sistema in generale che la realizzazione software. Di seguito, è riportato lo schema a blocchi complessivo del sistema hardware che è stato progettato.

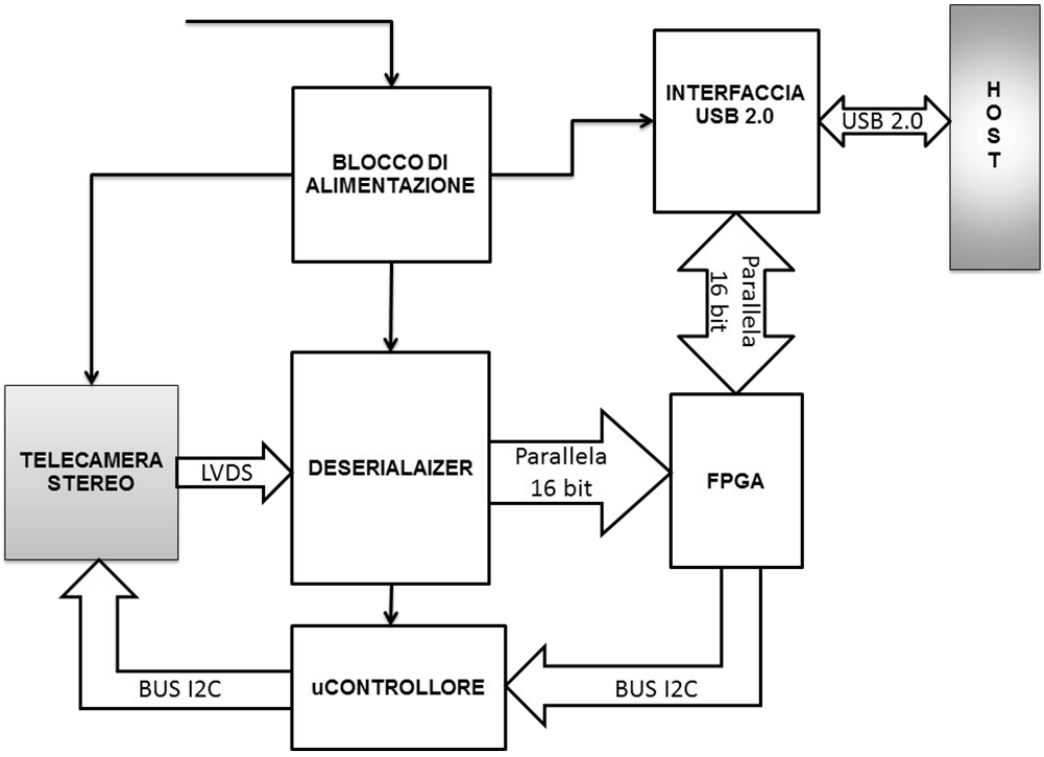


Figura 2.1: architettura logica di sistema

In primis si è reso necessario l'utilizzo di un controller USB per regolare il flusso dati (la scelta è ricaduta sul controller Cypress CY7C68013) ed è stato utilizzato un FPGA (Xilinx Spartan 3) per elaborazioni di immagini on board e due sensori (MT9V032) per l'acquisizione delle immagini.

2.1 Il controller USB CY7C68013

I controller Cypress FX2 sono tra i più popolari High-Speed USB controller sul mercato, in particolar modo per la loro flessibilità che ne permette l'uso in svariati ambiti. In particolare, per applicazioni che hanno la necessità di accedere al bus, Cypress offre i controller USB FX2LP (Low Power) con diverse piedinature.

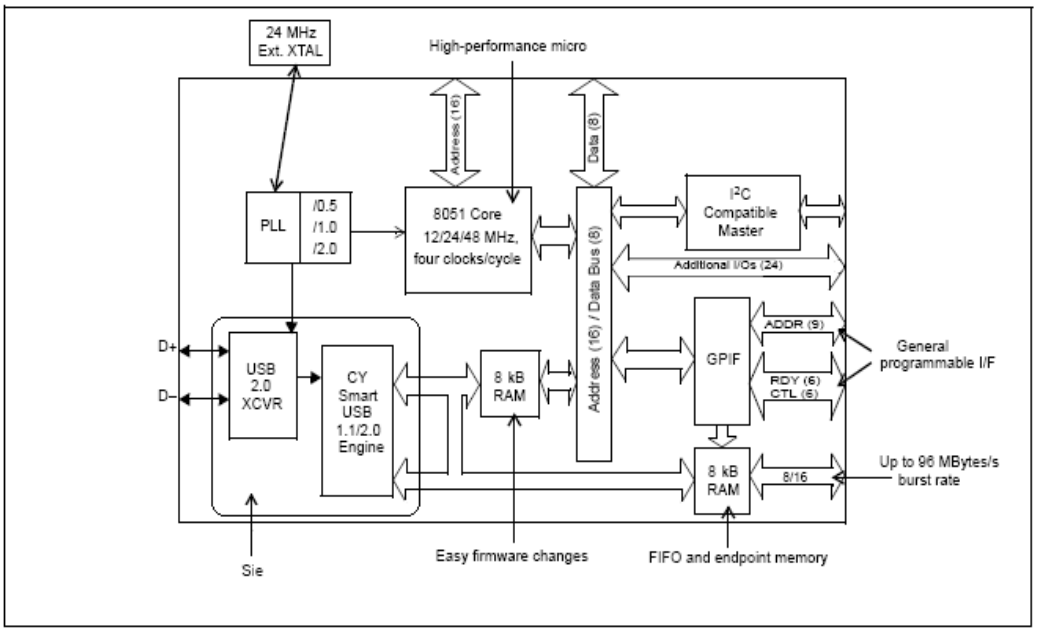


Figura 2.2: Schema a blocchi del CYC68013

L'FX2 racchiude tutte le funzioni che devono essere svolte da una periferica USB all'interno di un unico circuito integrato. Le funzionalità di cui dispone sono molto superiori a quelle necessarie per svolgere il progetto, ma è comunque utile avere una visione più generale di tale dispositivo. Il controller è disponibile in tre diverse piedinature: 56, 100 o 128 pin. Si è preferito l'utilizzo del dispositivo a 56

pin poiché è il più semplice ed idoneo per lo svolgimento dei compiti che gli saranno assegnati.

2.1.1 Caratteristiche principali

Le principali caratteristiche del FX2 a 56 pin package sono:

- tre porte di I/O A,B,D da 8 bit multiplexate per diversi utilizzi;
- FIFO configurabile con 8 o 16 bit di bus dati (porta B e D), 5 bit di controllo non multiplexate più 4 o 5 bit multiplexate con la porta A;
- GPIF per l'implementazione di diversi protocolli di trasmissione;
- un bus I^2C compatibile;
- un sistema di suspend mode per il risparmio energetico;
- una complessa gestione degli interrupt interni ed esterni.

2.1.1.1 CPU 8051

Il microprocessore 8051 presente all'interno del controller FX2 presenta caratteristiche leggermente diverse da quelle dell'8051 standard, pur utilizzando comunque lo stesso instruction set in modo da essere compatibile con gli stessi assemblatori e compilatori.

L'aspetto più rilevante dell'8051, presente sul controller, è l'aumento della velocità di esecuzione delle istruzioni dovuto alle seguenti modifiche:

- un ciclo di bus utilizza solo quattro periodi di clock contro i dodici dell'8051 standard;
- la frequenza della CPU è di 12, 24 o 48 MHz, fino a quattro volte maggiore rispetto all'8051 standard.

Inoltre, sono stati aggiunti:

- un altro data pointer;

- due porte seriali;
- una seconda usart (indirizzabili con due sfr);
- un terzo timer da 16 bit;
- un'interfaccia per memoria esterna con 16 bit di indirizzamento non multiplexati;
- due autopointer;
- vector USB and FIFO/GPIF interrupt;
- sleep mode con tre sorgenti di wakeup;
- I^2C compatibile bus serial controller da 100 a 400kHz;
- alcuni SFR.

Le funzioni svolte dall'8051 all'interno del controller sono:

- configurare gli endpoint;
- rispondere ai segnali di controllo inviati all'Endpoint 0;
- monitorare l'attività della GPIF;
- gestire la memoria fisicamente esterna;
- mettere a disposizione USARTs, counter-timers, interrupts e I/O pins.

Tipicamente l'8051 non prende parte alle comunicazioni in cui è richiesta velocità di trasmissione elevata (high-speed 480 megabit/secondo), come, ad esempio, lo streaming dati attraverso l'interfaccia USB.

2.1.1.2 Memoria interna

L'organizzazione della memoria dell'FX2 è simile, ma non precisamente identica, a quella dell'8051 standard. Sono presenti tre aree di memoria: Internal Data Memory, External Data Memory ed External Program Memory (queste

ultime due non necessariamente esterne al controller).

L'Internal Data Memory (o Internal Data RAM) è divisa in tre differenti regioni:

- Upper 128: 128 byte (indirizzi 0x80 - 0xFF) accessibili come general-purpose RAM, ma solo utilizzando indirizzamento indiretto (contenente lo stack);
- Lower 128: 128 byte (indirizzi 0x00 - 0x7F) accessibili come general-purpose RAM, utilizzando sia indirizzamento diretto che indiretto bit a bit e otto registri;
- SFR (Special Function Register) Space copre gli stessi indirizzi di Upper 128, ma l'FX2 tiene separate queste due regioni utilizzando un differente indirizzamento, ovvero SFR utilizza indirizzamento diretto.

Per quanto riguarda l'External Data Memory e l'External Program Memory l'8051 standard realizza l'architettura Harvard, ovvero data-memory e program-memory sono fisicamente separate. L'FX2 modifica questo modello, ovvero off-chip queste due memorie rimangono separate, ma on-chip sono unite secondo l'architettura di Von Neumann. Questo permette di scrivere in RAM tramite una EEPROM oppure direttamente tramite USB e successivamente di utilizzarla come program-memory. La memoria interna è costituita da due banchi di RAM entrambi da 8 Kbyte: nel primo viene allocato il firmware del dispositivo con altri dati, mentre nel secondo sono presenti gli Endpoint e la FIFO.

2.1.1.3 Endpoint buffers

Gli Endpoint sono definiti dalle specifiche USB come la fonte dei dati, ovvero una FIFO che viene riempita e svuotata in modo sequenziale con i dati provenienti dall'USB. Non possono essere presenti più di 32 endpoint (16 IN, ovvero letture lato host, e 16 OUT per scritture su periferica), che vengono selezionati tramite il loro indirizzo.

Nell'FX2 sono presenti 3 Endpoint di 64 byte accessibili solo dal firmware:

- EP0 è il Control Endpoint di default bidirezionale, è in pratica il buffer verso cui l'host invia il firmware che verrà poi caricato sulla periferica;
- EP1 IN/OUT utilizzano differenti buffer a 64 byte. Configurabili come Bulk, Interrupt o Isochronous.

A differenza di EP0 ed EP1, vi sono altri 4 endpoint (definiti come Large Endpoint) adibiti al trasferimento di grandi moli di dati anche senza l'intervento diretto del firmware, ovvero EP2, EP4, EP6 ed EP8. Sono previsti trasferimenti Bulk, Interrupt e Isochronous con direzione IN o OUT ed inoltre doppio, triplo o quadruplo buffer per rendere il più veloce possibile la trasmissione dei dati (aumentare il buffer implica tuttavia un impiego maggiore di risorse, quindi è bene dimensionarlo in base alla reali esigenze del trasferimento).

Dovendo l'FX2 supportare sia trasferimenti Full Speed che High-Speed, anche gli endpoint subiranno delle modifiche conseguenti. In particolare, verrà sicuramente alterata la dimensione massima del pacchetto (in byte) secondo le specifiche USB.

2.1.1.4 Input/Output

L'FX2 con 56 pin prevede due sistemi di Input/Output. Oltre ad un bus controller I^2C compatibile adibito all'interfacciamento con le periferiche compatibili e per l'eventuale caricamento del firmware caricato in EEPROM. Il controller presenta diversi pin di I/O programmabili per diverse funzioni (Gpif, Fifo, timer, interrupt ecc.) mediante il Configuration Register. Al momento dell'accensione, la configurazione di default prevede che tutti i pin siano adibiti all'I/O con direzione IN.

Sono disponibili al massimo 3 porte di I/O bidirezionali indicate con A, B e D (la versione a 128 pin prevede anche altre due porte, C ed E). Ad ogni porta è associata una coppia di Special Function Register:

- OEx (x=A,B,D) codifica la direzione bit a bit;
- IOx rappresenta il valore letto o scritto dal pin selezionato.

2.1.1.5 Sistema di Interrupt

Garantire una sincronizzazione tra periferica e host è uno dei punti chiave per assicurare una corretta gestione dell'I/O e questo onere grava sull'interfaccia stessa, che deve quindi comunicare le informazioni necessarie al sincronismo.

Le modalità di gestione sono principalmente due, gestione a Polling e gestione ad Interrupt. Nella gestione a Polling, l'interfaccia rende disponibili tali informazioni in uno o più registri di stato, che saranno letti dalla CPU prima dell'inizio di ogni trasferimento (ad esempio nel caso di scrittura si deve controllare che il buffer non sia pieno, nel qual caso la CPU è costretta ad attendere che venga svuotato). Tale modalità, nonostante sia la più semplice, impegna costantemente la CPU nel monitoraggio del registro e quindi ne implica una scarsa efficienza di utilizzo.

La gestione ad Interrupt pone rimedio a questo problema poiché permette alla CPU di eseguire una serie di istruzioni nell'intervallo di tempo in cui l'interfaccia non è disponibile. La CPU, infatti, riceve un Interrupt dall'interfaccia quando quest'ultima è pronta, interrompe la procedura correntemente in esecuzione e risponde alla richiesta di servizio mettendo in esecuzione un Interrupt Handler, ovvero una routine di risposta che si occuperà di effettuare il trasferimento.

Nell'FX2 è quindi l'8051 che gestisce gli interrupt controllando al termine di ogni istruzione (quindi ogni quattro cicli di clock) se sono stati settati alcuni Interrupt Flag o se sono state ricevute richieste di interrupt sui pin INT0 ed INT1 (configurabili sia a fronte che a livello). La latenza di risposta ad un interrupt dipende dallo stato del controller, ma non supera mai tredici cicli di bus.

La procedura di risposta ad un interrupt prevede l'accesso all'Interrupt Service Routine associata a quella particolare interruzione, la sua esecuzione (a meno che

non venga attivata una richiesta di interruzione di priorità superiore). Ogni routine deve terminare infine con l'istruzione RETI (Return from Interrupt) dopo la quale l'FX2 continua la sua normale esecuzione dall'istruzione successiva alla richiesta di interrupt.

2.1.1.6 Reset

In tutti i circuiti elettronici, il segnale di Reset ha una funzione di primaria importanza; esso, infatti, permette alla macchina di portare tutti i blocchi che la compongono in uno stato noto e conosciuto in modo da poterne predire il comportamento.

L'FX2 ha due reset interni:

- Power on Reset: pilotato dal pin RESET;
- CPU Reset: resetta la CPU 8051.

Inoltre, viene definito dalle specifiche USB un USB Bus Reset relativo alle periferiche collegate.

2.1.1.7 General Programmable Interface

La GPIF, ovvero General Programmable Interface, ricopre il ruolo di master nel pilotare il buffer di tipo First In First Out, allocato in Ram, relativo alla FIFO. I compiti di tale interfaccia però coprono anche la gestione di tutti i segnali di controlli per gli eventuali dispositivi slave collegati realizzando anche protocolli di comunicazione specifici.

Tramite la GPIF l'FX2 è possibile interfacciare con una qualunque periferica a 8 o 16 bit, in quanto vengono messi a disposizione in pin che possono operare come output (CTL[5:0]), input (RDY[5:0]), bus dati (FD[15:0]) e indirizzi (GPIFADDR[8:0]). Al momento dell'inizializzazione del dispositivo, il firmware copia il contenuto di quattro registri allocati in ram, chiamati Waveform Descriptor, nella memoria interna della GPIF.

2.1.1.8 Modalità Slave Fifo

Nonostante alcune applicazioni necessitino dell'intervento della CPU sui dati transitanti sul controller, sempre più spesso l'FX2 viene utilizzato come collegamento tra USB ed una generica sorgente di dati esterna, cioè la modalità Slave FIFO.

In tale modalità i dati scorrono tra l'host ed il master esterno senza che vengano in alcun modo alterati o modificati dal controller attraverso gli endpoint fifo interni. Sono presenti 4 slave Fifo (selezionate dai pin FIFOADR[1:0]), la configurazione di default non prevede la modalità slave FIFO, è necessario settare IFCONFIG[1:0] ad 1.

Sono presenti sia la modalità asincrona (il dato viene scritto su ogni fronte del segnale SLWR) sia quella sincrona (si utilizzano i fronti del clock, segnale IFCLK). Entrambe le modalità hanno dei limiti di frequenza in cui è garantito il funzionamento. Per quanto riguarda la modalità asincrone, è presente un upper bound di 8 MHz, mentre la modalità sincrona supporta frequenze comprese nell'intervallo 5 – 48 MHz.

Il clock dell'FX2 (IFCLK) può avere sia sorgente esterna che interna (frequenza configurabile a 30 o 48 MHz mediante IFCONFIG[6]). Il bus dati delle FIFO può avere ampiezza 8 o 16 bit in base al valore di WORDWIDE (pin 0 del registro EPxFIFOCFG, dove x può assumere il valore 2, 4, 6, 8). Nel caso in cui Wordwide sia 0 si lavorerà ad 8 bit, con FD[0:7] che sostituirà la porta B, in caso contrario (ovvero con 16 bit) sarà requisita anche la porta D per i bit FD[15:8].

Se tutte le FIFO sono configurate a 8 bit, la Porta D rimane disponibile per altri usi generali di I/O, mentre è sufficiente anche che solo una delle FIFO sia configurata a 16 bit perché tale possibilità venga negata. Lo stato delle FIFO viene riportato da 4 pin (FLAGA, FLAGB, FLAGC, FLAGD). Oltre ai canonici Fifo-Full e Fifo-Empty (condizioni rispettivamente di Fifo piena e vuota), un altro

segnale indica se la Fifo ha superato o meno una soglia programmabile dall'utente (chiamato Programmable Flag). Questi pin sono fondamentali per il master esterno che scrive o legge dati dal controller per ottenere una sincronizzazione tra i due in modo da evitare perdite di dati. In particolare, tipicamente nel caso di IN Endpoint, il master esterno sarà interessato al Fifo Full, mentre quando starà operando su OUT Endpoint monitorerà l'Empty Fifo.

I Flag possono operare in Indexed Mode oppure in Fixed Mode. Nel primo caso riporteranno lo stato della Fifo correntemente selezionata dai pin FIFOADR[1:0], in particolare:

- Flag A – Programmable Flag;
- Flag B – Full Status;
- Flag C – Empty Status;

Se invece si vuole associare in modo statico un Flag ad una particolare condizione (es. Flag A sarà sempre Empty Status dell'Endpoint 2) si deve utilizzare la Fixed Mode, che permette tale operazione mediante il settaggio di due registri, PINFLAGSAB e PINFLAGSCD.

Per quanto riguarda i segnali di controllo, i principali sono SLOE (enable per l'output dei pin FD), SLRD (segnale di lettura), SLWR (segnale di scrittura), FIFOADR (seleziona una delle quattro fifo) e PKTEND (se il master esterno vuole inviare un pacchetto IN prima che questo raggiunga la sua dimensione prestabilita).

Una modalità particolarmente utile è l'Auto-In/Out che configura l'FX2 per commissionare direttamente i pacchetti USB. In particolare, per gli IN Endpoint, il master esterno scrive continuamente sulla FIFO senza che il firmware dell'FX2 debba agire sui dati.

2.1.2 I^2C

Abbiamo visto che FX2 ha un bus controller I^2C compatibile adibito all'interfacciamento con le periferiche. I^2C (Inter Integrated Circuit) è un bus seriale creato dalla Philips per collegare periferiche a media-bassa velocità; oggi è in grado di trasferire dati ad una frequenza massima di 400kHz. Questa interfaccia utilizza due linee seriali: Serial Data (SDA) e Serial Clock (SCL), entrambe open drain, che necessitano quindi di due resistori per il pull up.

E' presente un solo master nel bus che ha il compito di generare il clock di linea (SCL) per tutti i dispositivi slave collegati al bus ed è l'unico che può selezionare con quale periferica dialogare e in quale direzione. SDA rappresenta la linea di scambio dati e deve rispettare una certa sincronizzazione con SCL. [2.1]

2.1.3 Circuito di generazione del clock

La generazione del clock dell'intero sistema è affidata a un apposito blocco, il cui unico scopo è fornire il clock di riferimento ai sensori e al deserializer.

Il chip che è stato utilizzato è il CY22393 prodotto dalla Cypress, questo componente implementa:

- tre PLL autonome;
- una serie di divisori/moltiplicatori programmabili via I^2C;
- un sistema di suspend mode.

2.1.3.1 PLL

PLL o Phase Locked Loop (in italiano "anello ad aggancio di fase") è un circuito elettronico progettato per generare un'onda ad una data frequenza la cui fase ha una relazione specifica con un segnale di riferimento. Una PLL è generalmente composta da tre blocchi: oscillatore controllato in tensione, divisore di frequenza e comparatore di fase. Tale tipo di circuito si usa per diversi scopi,

quali:

- sintetizzatore di frequenza, essendo in grado di sintonizzare un oscillatore controllato in tensione (Voltage Controlled Oscillator o VCO, dispositivo in grado di produrre oscillazioni ad alta frequenza ma dotato di bassa precisione) con un oscillatore al quarzo (caratterizzato al contrario da una bassa frequenza di risonanza ma anche da una precisione molto elevata);
- generatore di clock, soprattutto nei sistemi a microprocessore;
- sistema di clock recovery, finalizzato cioè all'estrazione del clock da un segnale aperiodico modulato.

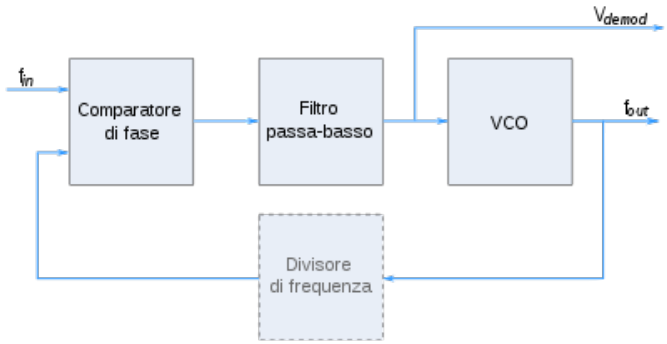


Figura 2.3: schema a blocchi di un generale PLL

Il funzionamento è il seguente: il segnale in ingresso, cioè quello che condizionerà il funzionamento del VCO, viene inserito nel comparatore di fase, che ne identifica il fronte di salita dell'onda; il VCO viene posto in oscillazione libera ad un valore prossimo a quello che sarà il suo valore di lavoro. Un divisore di frequenza programmabile ricava un segnale sottomultiplo di quello generato dal VCO e lo applica ad un secondo ingresso del comparatore. L'uscita del comparatore sarà una tensione continua che controllerà il VCO, mantenendone la frequenza rigorosamente agganciata in fase con il segnale entrante. [2.2]

2.2 Sensori di immagini

I sensori per l'acquisizione delle immagini sono elementi fondamentale del sistema (MT9V032). Le diverse modalità di funzionamento di questo sensore ne fanno un dispositivo particolarmente efficiente e versatile; può, infatti, essere utilizzato sia singolarmente nelle telecamere monoculari, sia in coppia con un altro sensore per implementare una telecamera stereo.

Per collegare fisicamente il sistema stereo all'Host utilizzeremo il controller USB 2.0 (FX2). Il compito del controller FX2 consiste solo nell'inviare, attraverso l'USB, i dati in arrivo dai due sensori verrà perciò utilizzato in modalità auto in/out auto.

Si è scelto di non utilizzare l'8051 interno all'FX2 che porta a una semplificazione del già complesso firmware, ma rende necessario l'inserimento di un altro microcontrollore (PIC18F4690) che ricoprirà il ruolo di master dell'intero sistema, che vedremo più avanti.

I due sensori di immagine, nel complesso, trasferiscono attraverso due linee seriali differenziali (LVDS); per convertire i dati da seriali a paralleli è stato inserito un deserializzatore (DS92LV16).

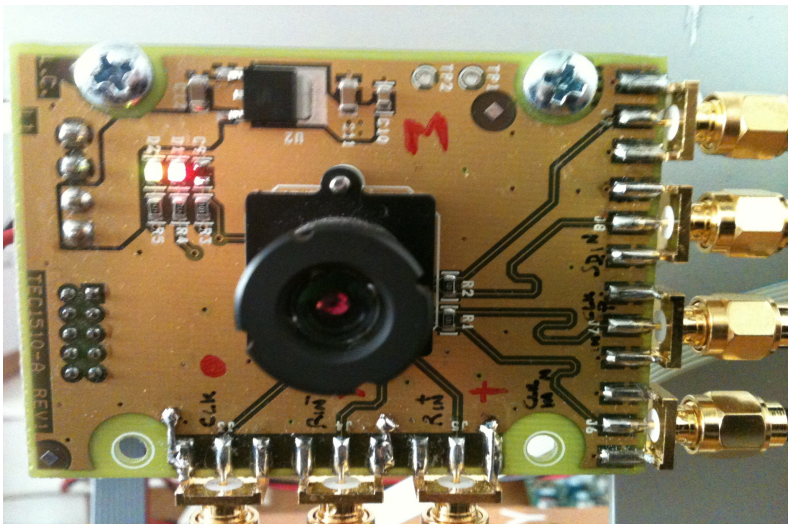


Figura 2.4: sensore per l'acquisizione delle immagini

2.2.1 MT9V032

Le diverse modalità di funzionamento del sensore MT9V032 ne fanno un dispositivo efficace e alquanto versatile; può infatti essere utilizzato sia nelle telecamere tradizionali monoculari sia nelle telecamere che sfruttano la tecnologia stereo; è proprio in quest'ultimo impiego che focalizzeremo la nostra attenzione.

Non ci soffermeremo alla descrizione dettagliata del funzionamento del sensore di immagine, ma si concentrerà l'attenzione sul funzionamento del sistema stereoscopico e come esso dialoga verso l'esterno. Come già accennato in precedenza, il sistema per l'acquisizione di immagini stereo è composto da due sensori aventi differenti funzionalità.

Il sensore secondario funzionante in stereoscopic slave ha il compito di trasmettere all'altro sensore, attraverso due linee seriali differenziali, le informazioni di uno specifico pixel letto e alcuni bit di handshaking.

Il secondo sensore funzionante in modalità stereoscopic master ha il compito di gestire la sincronizzazione con lo slave sensor e di trasmettere in uscita, attraverso due linee seriali differenziali, le informazioni ricevute allegando anche i propri dati letti (non ci è dato sapere come avviene l'operazione di sincronizzazione tra i due sensori). I due sensori di immagine sono quindi in grado di implementare un sistema stereo perfettamente sincronizzato senza nessun intervento dall'esterno a patto che siano configurati con gli stessi parametri (tempo di esposizione, risoluzione, ampiezza della finestra, tipo di scansione, frame rate ecc.).

Cominciamo con l'esaminare meglio il singolo sensore di immagine. La matrice di pixel ha una risoluzione massima di 752 x 480, l'informazione in tensione proveniente da ogni elemento fotosensibile (fotodiodo) viene convertita in formato digitale a 8/10 bit e processata al fine di essere trasmessa verso l'esterno nel formato più adeguato.

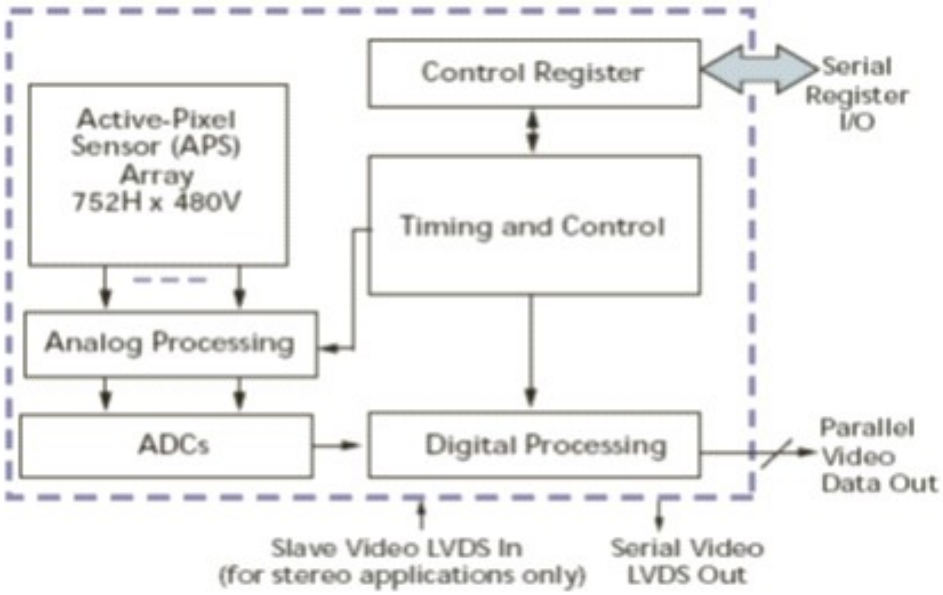


Figura 2.5: schema a blocchi del sensore MT9V032

In modalità STAND ALONE, in cui viene utilizzato un singolo sensore, possiamo scegliere tre diversi tipi di funzionamento:

- *Simultaneous mode*: in questa modalità il readout e l'esposizione avvengono in parallelo, rendendo l'operazione più veloce, ma più difficile da sincronizzare;
- *Sequential mode*: a contrario della modalità precedente, qui l'esposizione avviene successivamente al readout;
- *Snapshot mode*: in questa modalità il sensore accetta un segnale di trigger esterno che fa cominciare immediatamente l'esposizione e il successivo readout dell'immagine catturata.

Lo stream di uscita può esser in formato parallelo a 8 bit oppure seriale a 12 bit per pixel. Nel formato seriale possiamo selezionare due diverse modalità di impacchettamento: la prima prevede 10 bit di dato più lo start e lo stop bit, mentre la seconda 8 bit di dati, lo start bit, lo stop bit e sono presenti inoltre line valid e frame valid.

In modalità SERIAL STEREOSCOPIC i dati possono essere inviati solo per via seriale in pacchetti da 18 bit contenenti: il bit di start, 8 bit di dati su uno specifico pixel provenienti dallo slave sensor, 8 bit di dati riguardanti lo stesso pixel provenienti dal master sensor ed infine il bit di stop. I segnali di frame e line valid sono ricavati con l'utilizzo di parole riservate.

I bit vengono trasmessi in uscita ad una frequenza di 520 Mhz, supponendo che il tempo di esposizione della matrice di pixel sia il minimo possibile e che i sensori funzionino alla massima risoluzione.

I sensori dovranno essere configurati uno come SLAVE STEREOSCOPIC e l'altro come MASTER STEREOSCOPIC. Le linee seriali di ingresso del master sensor SER_DAT_IN andranno collegate con le linee seriali di uscita dello slave sensor per permettere la comunicazione tra i due sensori.

Per quanto riguarda la programmazione dei sensori dobbiamo settare il valore di 256 registri (da 16 bit) per ciascun dispositivo, ricordando anche che alcuni registri vanno programmati in simultanea (broadcast) per mantenere una corretta temporizzazione.

Da notare inoltre che la programmazione avviene attraverso un bus I^2C; il nostro controller USB implementa questo tipo di I/O ma, per mantenere il software di controllo più semplice possibile, si è deciso di non utilizzarlo per la programmazione ed inserire un altro processore esterno dedito ad eseguire questo lavoro.

La parte bassa del select code del bus I^2C di ogni sensore è assegnata attraverso i due pin CTRL_ADR0 e CTRL_ADR1. Per ridurre il frame rate in uscita dai sensori è possibile aggiungere dei pixel dummy, a lato ed in fondo alla finestra di immagine, queste righe e colonne vengono chiamate rispettivamente horizontal e vertical blanking.

L'ammontare del blanking verticale e orizzontale è configurabile tramite due registri R0x04, R0x05. Il segnale IMAGE VALID, quando basso, avverte che in

uscita si sta trasmettendo un immagine non valida, cioè il blanking. [2.3]

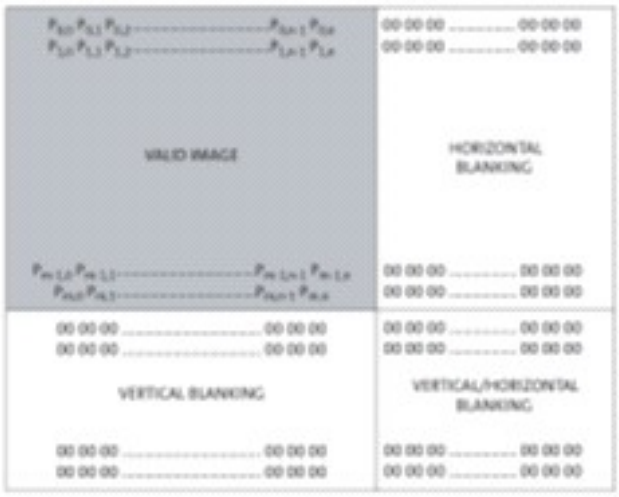


Figura 2.6: sistemazione del blanking orizzontale e verticale

2.3 Deserializer

Questo dispositivo combina un convertitore parallelo/seriale (serializer) ed un convertitore seriale/parallelo (deserializer) in un singolo chip. I due blocchi vengono mantenuti separati l'uno dall'altro sia per l'alimentazione, sia per quanto riguarda la generazione del clock. I due convertitori possono quindi lavorare anche in simultanea per avere una connessione in full duplex mode.

Di nostro interesse risulta solo il blocco deserializer, per trasformare dati seriali, provenienti dai sensori, in dati paralleli a 16 bit da inviare al controller USB.

2.3.1 Caratteristiche principali

Ora prendiamo in esame le caratteristiche e le funzionalità presenti nel blocco deserializer.

Inizialization: quando viene fornita l'alimentazione tutte le uscite sono in tristate. Appena la tensione supera i 2,2V la PLL interna cerca un clock locale (fornito sul pin REFCLK) a cui agganciarsi; le uscite rimangono ancora in trisate e il pin LOCK rimane alto.

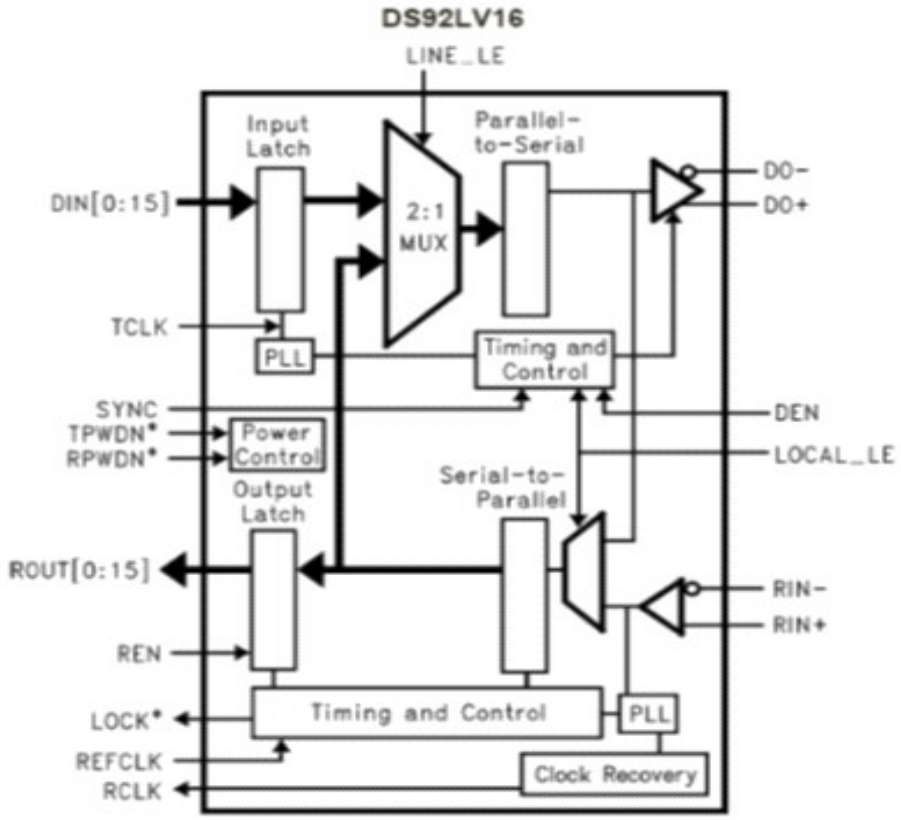


Figura 2.7: schema a blocchi del DS92LV16

Per completare la sincronizzazione la PLL del blocco deserializer deve sincronizzarsi con i dati ricevuti sui pin, RIN+ e RIN-. Per agevolare questa fase può essere inviato un pattern di sincronizzazione di bit non ripetitivi, ma è anche possibile sincronizzare la PLL con dati random; in tal caso però non è stimabile il tempo necessario per l'aggancio. Lo stato logico basso del pin LOCK segnala la corretta sincronizzazione e la validità dei dati in uscita sulle linee parallele.

Data transfer: il circuito interno di clock recover viene utilizzato per generare un segnale di strobe. Il fronte di salita di questa linea segnala la campionabilità dei dati presenti nella porta parallela.

Resynchronization: quando si perde l'aggancio (il pin LOCK va alto), il deserializer cercherà automaticamente di risincronizzarsi in modo random estraendo informazioni dai dati in arrivo.

Powerdown: è uno stato a bassa potenza in cui si può entrare abbassando TPWDN, RPWDN (notare che il serializer e il deserializer possono essere spenti autonomamente). In questo stato si spegne la PLL (si perde l'aggancio) e il dispositivo si mette quindi in attesa per una nuova inizializzazione.

Tristate: quando il pin REN viene portato basso le uscite del deserializer sono poste in alta impedenza.

Loopback Test Operation: è possibile creare una sorta di feedback collegando RIN+, RIN- con DO+, DO- e ROUT[0:15] con DIN[0:15]; in questo modo tutti i dati ricevuti dal dispositivo vengono rinvianti al mittente, che potrà quindi controllare se siano stati ricevuti correttamente dal deserializzatore.

Da notare che il tempo necessario alla sincronizzazione è molto breve (circa un micro secondo) nel caso venga inviato un sync pattern; ha, invece, durata non predicibile se si utilizza la funzione random lock. [2.4]

2.4 Microcontrollore

Per mantenere il software di controllo del controller USB il più semplice possibile, non viene utilizzato il processore 8051 messo a disposizione dalla Cypress. Si è reso quindi necessario l'inserimento di un processore esterno dedito al controllo dei reset e all'inizializzazione dei due sensori di immagine (ricordiamo che la programmazione di tali dispositivi avviene tramite un bus I^2C). Un valido candidato a svolgere questi compiti è il PIC18F2585 prodotto dalla Microchip.

Questo microcontrollore ha prestazioni modeste, compatibilmente con le mansioni che dovrà svolgere, ed è disponibile sul mercato a basso costo. Di seguito vengono brevemente elencate le sue principali caratteristiche:

- il parallelismo della ALU (Aritmetical Logical Unit) è 8 bit;
- diverse versioni 28/40/44 package (noi utilizzeremo la 44);

- 64Kbyte di Flash, 3Kbyte di SRAM, 1Kbyte di EEPROM;
- programmabile e testabile via JTAG;
- 32 linee di Input Output (I/O) multiplexate;
- Watch Dog Timer (WDT);
- modulo Pulse width Modulation (PWM);
- 4 Phase Lock Loop (PLL);
- 3 Interrupts esterni con priorità programmabile;
- è munito di interfacce: I^2C, SPI e RS232;
- 3 timer/counter a 8/16 bit programmabili;
- ADC a 10 bit con 11 canali con velocità fino a 100 Ksps;
- 2 comparatori analogici;
- modalità di stop e Idle. [2.5]

2.5 FPGA

Un Field Programmable Gate Array, solitamente abbreviato in FPGA, è un circuito integrato digitale la cui funzionalità è programmabile via software. Sono elementi che presentano caratteristiche intermedie rispetto ai dispositivi ASIC (Application Specific Integrated Circuit) da un lato e a quelli con architettura PAL (Programmable Array Logic) dall'altro. [2.6]

L'architettura di una FPGA è rappresentata da un insieme di celle logiche che comunicano tra loro e con i segnali di ingresso-uscita mediante un insieme di collegamenti programmabili disposti orizzontalmente e verticalmente. Le funzioni logiche sono ottenute combinando diverse celle logiche in differenti modi.

La struttura delle celle logiche e lo sviluppo dell'insieme dei collegamenti differiscono da costruttore a costruttore. Attualmente le tecnologie più utilizzate

con cui realizzare FPGA sono: SRAM (Static RAM) e ANTIFUSE. Ognuna di queste permette di realizzare FPGA che soddisfano a particolari esigenze di mercato.

L'architettura di una cella logica è fortemente influenzata dalle caratteristiche delle interconnessioni. Le FPGA che hanno una struttura di interconnessioni caratterizzata da molti fili e molti elementi di connessione programmabili tendono ad avere celle logiche semplici con un minor numero di ingressi; in questi casi viene perciò utilizzata la tecnologia antifuse. Invece, le FPGA che presentano strutture di interconnessione con un minor numero di fili e di interconnessioni programmabili tendono ad avere celle logiche più complesse con un maggior numero di ingressi. In quest'ultimo caso si utilizza tipicamente la tecnologia a SRAM.

2.5.1 Xilinx Spartan 3

Vediamo ora in maniera generale le caratteristiche che l'azienda costruttrice Xilinx ha introdotto nella Spartan 3.

L'FPGA della serie Spartan 3 utilizza blocchi di interconnessione programmabili di tipo SRAM con blocchi logici complessi (Look Up Table). La programmazione di tale dispositivo non è quindi permanente, ma va effettuata ogniqualvolta viene fornita alimentazione.

L'azienda costruttrice Xilinx, al fine di agevolare al massimo il compito del progettista, mette a disposizione una development board che sfrutta al meglio le caratteristiche della spartan3. Questa complessa scheda contiene già al suo interno:

- un package 320 solder ball grid array (BGA); 10.000 gate logici equivalenti; 18K byte di block ram; 18 moltiplicatori hardware a 18 bit; 4 Digital Clock Managers (DCM) con sistemi di eliminazione dello skew, moltiplicazione e divisione di frequenza e grande intervallo di frequenza di

funzionamento (da 5 MHz a 300 MHz); 232 user-defined I/O signals;

- 4 Mbit di memoria PROM configurabile mediante jumpers;
- 64 MByte di memoria sincrona DDR SDRAM bus a 16 bit;
- 16 Mbyte di NOR flash;
- 3 bit, 8 color VGA display port;
- 9 pin, RS232 serial port;
- PS2 port per mouse e tastiera;
- 10/100 ethernet inteface;
- 4 DAC e 2 ADC;
- encoder rotativo;
- display a lcd con due righe;
- 4 interruttori a slitta, 4 pulsanti;
- 8 led indirizzabili singolarmente;
- un oscillatore a 50 MHZ ed un socie libero per un oscillatore ausiliario;
- porta JTAG;
- un controller USB 2.0;
- diversi linearizzatori per fornire tensioni a 5V, 2.5V e 1.2V;
- un connettore HIROSE 100 pin verso l'esterno.

Una caratteristica saliente della Spartan 3 è una nuova struttura di gate nickel-silicide auto allineante che riduce la resistenza di gate e minimizza alcuni dei margini dovuti al processo stesso che possono altrimenti ridurre le performance del transistor stesso.

Figura 2.8: developement board Spartan 3

L'innovazione architetturale della Spartan è che Xilinx ha creato l'architettura ASMBL (Advanced Silicon Modular Block), per consentire l'assemblaggio rapido ed efficiente di piattaforme FPGA con caratteristiche differenti, garantendo la possibilità di variare il mix di caratteristiche del dispositivo, in modo da raggiungere una buona ottimizzazione per ciascun dominio di applicazione. [2.7]

3. TOOL E COMPONENTI SOFTWARE DEL SISTEMA

L'obiettivo del progetto era il corretto interfacciamento tra una telecamera stereo, collegata ad una FPGA per eventuali elaborazioni di immagini on-board, alla cui uscita USB era connesso un controller USB che gestiva il flusso dati bidirezionale verso il computer.

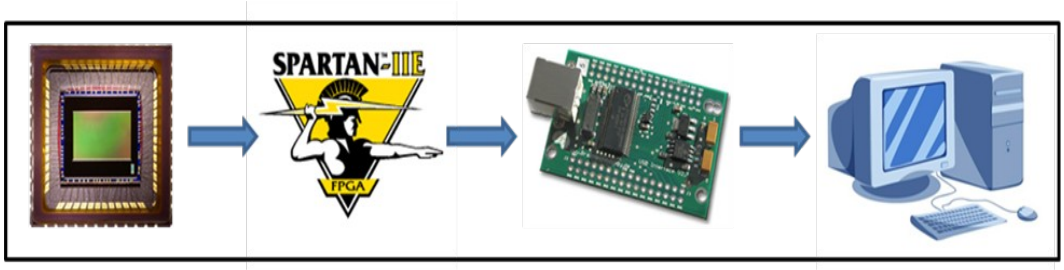


Figura 3.1: flusso dati tra sensore ed host

Nel capitolo precedente abbiamo visto l'architettura e i componenti hardware del sistema e una breve descrizione di essi. In questo capitolo cerchiamo di approfondire i tool ed i componenti software che sono stati utilizzati per la realizzazione del progetto.

3.1 Tool di sviluppo Cypress

Cypress fornisce un SuiteUSB che racchiude tutti gli elementi necessari per la configurazione ed i test relativi al controller. All'interno della SuiteUSB, Cypress fornisce diversi tools di sviluppo ed applicazioni utili alla programmazione del nostro dispositivo atti a testarne il corretto funzionamento. Sono inoltre presenti alcuni progetti di esempio che cercano di introdurre ai concetti fondamentali per iniziare lo sviluppo. Il tool fondamentale e di nostro interesse è CyControl Center.

Il Control Center è un utility che permette la comunicazione non solo con le periferiche associate al driver CyUSB.sys ma anche ai driver Microsoft.

La finestra principale si divide essenzialmente in due pannelli. In quello di sinistra è possibile selezionare una periferica, che viene esplorata mediante una struttura ad albero dalla quale è necessario selezionare un endpoint per ottenere informazioni a riguardo (cliccando sul tab "Descriptor Info") ed effettuare trasferimenti (tramite il tab "Data Transfers").

In particolare, è possibile stabilire il numero di byte che devono essere trasferiti (inserendone il valore nella textbox "Byte to transfer") e, solamente per i trasferimenti di tipo OUT, i dati da inviare verso la periferica.

Cliccando sul bottone "Transfer Data" viene fatto partire il trasferimento. Gli eventuali dati trasferiti sono visualizzati sulla box sottostante oppure scritti su un file di testo (in base alla scelta dell'utente).

In caso il trasferimento non vada a buon fine, CyConsole stampa un'indicazione di errore che riporta anche il codice relativo all'errore occorso. Nel caso si riscontrino numerosi trasferimenti falliti, tipicamente la soluzione consiste nel resettare la pipe relativa all'endpoint su cui si sta lavorando, mediante il tasto *"*Reset Pipe*"* collocato in alto a sinistra.

L'ultimo tab ("Device Class Selection") permette di filtrare le periferiche visualizzate in base alla loro classe. In particolare, oltre ad escludere o meno i dispositivi che utilizzano il driver CyUSB.sys, si possono non elencare i dispositivi delle classi Human Interface (HID) e Mass Storage.

Particolare importanza ha la funzione "Program FX2", che permette di:

- Program RAM: programma il dispositivo caricando il firmware, in formato esadecimale (.hex), nella memoria RAM;
- Program 256 B / 64 KB EEPROM: programma il dispositivo caricando un file in una delle Eeprom presenti;
- Halt: invia il segnale di Reset alla CPU;
- Run: rilascia il segnale di Reset alla CPU.

La programmazione della memoria RAM permette quindi di caricare un firmware personalizzato sul dispositivo, come nel nostro caso, il quale verrà successivamente rienumerato (processo evidente in quanto la periferica temporaneamente scompare dalla lista dei dispositivi connessi, salvo poi essere di nuovo riconosciuta dopo qualche istante secondo i parametri imposti dal nuovo firmware).

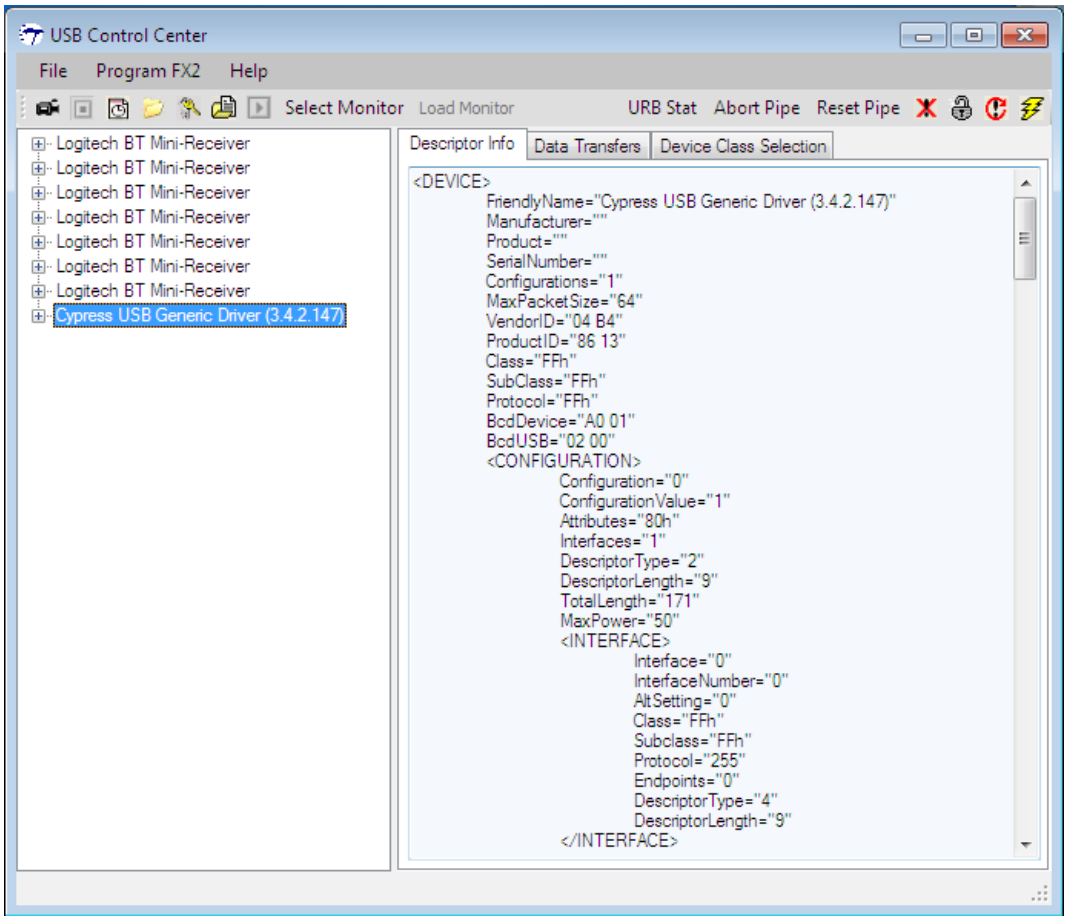


Figura 3.2: la finestra dell'USB Control Center

La prima volta che si carica un firmware personalizzato, il sistema operativo potrebbe richiedere di installare il driver appropriato. Non è comunque necessario utilizzare due diverse versioni del driver, in quanto tale file è modificabile per essere associato a più dispositivi con diversi PID e VID.

Una volta che il dispositivo viene rienumerato correttamente, la periferica avrà,

presumibilmente, una nuova configurazione visibile nella struttura ad albero.

Per controllare la corretta esecuzione dei comandi che mano a mano sono stati messi in esecuzione è possibile consultare la textbox posizionata nella parte inferiore sinistra della finestra principale che conferma la loro buona riuscita. [3.1]

3.2 VHDL

Il VHDL (Hardware Description Language) è un linguaggio di descrizione dell'hardware che consente la progettazione di circuiti integrati.

È un derivato del programma Very High Speed Integrated Circuit (VHSIC), ideato dal dipartimento della difesa Statunitense tra la fine degli anni 70 e l'inizio degli anni 80. Gli obbiettivi di questo linguaggio sono duplici: stabilire uno standard di interfacciamento tra i vari progettisti ed avere un linguaggio le cui istruzioni siano orientate alla rappresentazione circuitale.

L'insieme delle istruzioni che descrivono il progetto hanno lo scopo di modellare la realtà a cui il progettista fa riferimento. Per ritornare ai gate reali è necessario un passaggio intermedio che ha lo scopo di tradurre le istruzioni in logica combinatoria e logica sequenziale. Questo processo è chiamato sintesi e normalmente affidato a dei tool di sintesi automatici.

L'utilizzo di tool automatici conferisce la possibilità di creare circuiti sempre più complessi in un tempo molto più contenuto rispetto alla progettazione gate level.

Il rischio che si corre è quello di implementare una funzionalità logica su un area molto maggiore del necessario con prestazioni di timing deludenti o addirittura implementare delle funzionalità che non sono effettivamente sintetizzabili.

Una volta descritto il circuito in un linguaggio con una struttura formale si può procedere alla verifica della correttezza del progetto, tale fase viene chiamata simulazione e può essere solo in termini logici o può includere anche le

tempistiche. [3.2]

3.2.1 Xilinx ISE

Xilinx fornisce un IDE (Integrated Development Environment, noto anche come Integrated Design Environment, Integrated Debugging Environment o Interactive Development Environment) comprensivo di un tool software (Xilinx ISE Design Suite) che consente la programmazione dell'FPGA via USB usando come linguaggio VHDL. In questo modo è possibile programmare sia la memoria volatile (RAM) sia la memoria persistente (Flash).

Si può utilizzare questa IDE per la sintesi e l'analisi dei progetti HDL, ovvero permette allo sviluppatore di sintetizzare (o meglio compilare) i loro progetti, effettuare analisi dei tempi, esaminare i diagrammi RTL, simulare la reazione di un progetto a diversi stimoli e configurare il dispositivo. [3.3]

Interfaccia di navigatore del progetto (ISE), per impostazione predefinita, è suddivisa in tre pannelli (o sottofinestre). La prima sottofinestra, situato in alto a destra, è chiamato pannello di editor ed è un'interfaccia multidocumento (MDI) in cui è possibile inserire e visualizzare codice HDL, schemi, diagrammi di stato, report di progettazione, file di testo e la simulazione di forme d'onda.

La sottofinestra che si trova in alto a sinistra è suddiviso in due parti: la parte superiore si chiama pannello di source che mostra l'organizzazione dei file sorgenti che compongono il progetto; la parte inferiore si chiama pannello di process che ha il compito di elencare le varie operazioni che si possono eseguire su un determinato oggetto situato nel pannello source. Quest'ultima parte è sensibile al contesto e cambia in base al tipo di sorgente selezionato nel pannello source; si possono anche eseguire le funzioni necessarie per definire, eseguire e analizzare il progetto.

In questa seconda sottofinestra ci sono quattro schede che permettono di visualizzare i moduli funzionali (denominato start), file sorgenti (denominato

design), diversi snapshot (o versioni) del progetto (denominato files) e le librerie HDL per il progetto (denominato libraries). La scheda denominata design, inoltre, dà la possibilità di scegliere tra due differenti modi di visualizzazione: implementation o simulation.

L'ultima sottofinestra che si trova in basso, ed è denominato transcript, visualizza i vari messaggi del processo attualmente in esecuzione in tre schede diverse, e sono: console (visualizza i messaggi di stato), errors (visualizza i messaggi di errore) e warnings (visualizza i messaggi di avviso).

Ognuna di queste tre sottofinestre può essere ridimensionata, sganciata dal navigatore del progetto, trasferita in una nuova posizione all'interno della finestra principale del navigatore oppure chiusa. [3.4]

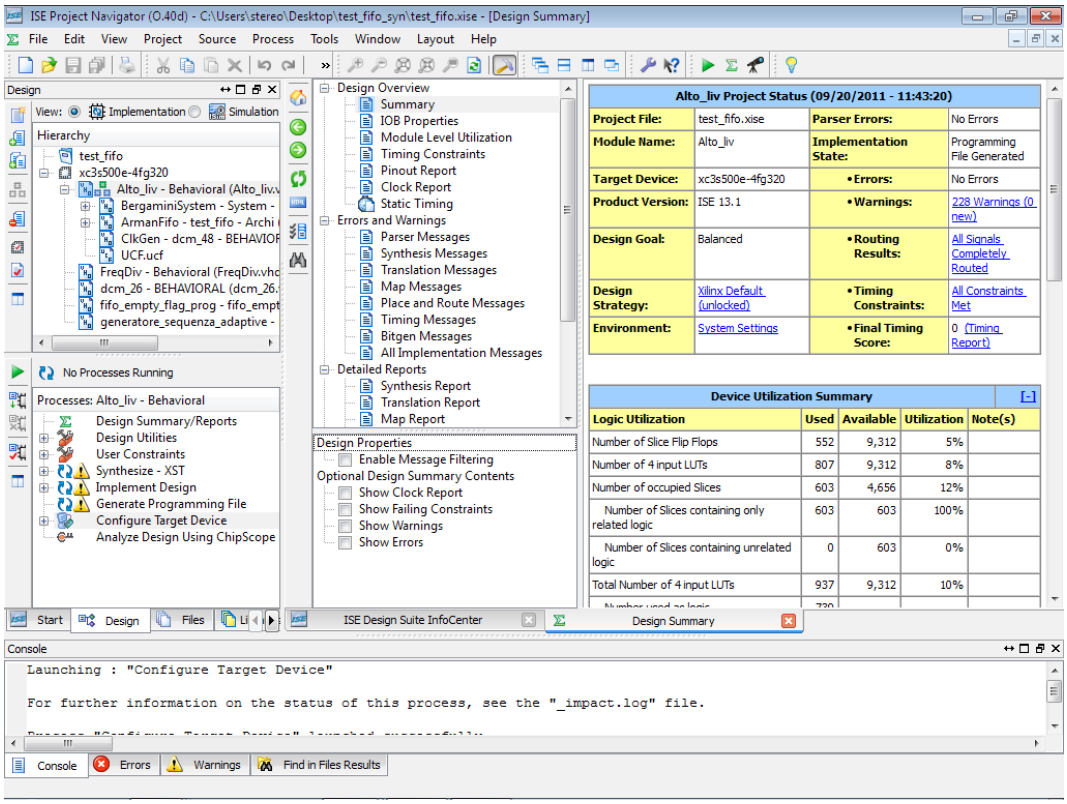


Figura 3.3: la finestra del xilinx ISE design suite

In questo progetto l'utilizzo principale del suite, oltre a quella di progettazione VHDL, è stato quello di configurare (o programmare) il dispositivo, ossia l'FPGA.

Per fare ciò, all'inizio bisogna creare il progetto software contenente la descrizione del hardware dei circuiti integrati (in VHDL), bisogna selezionare nel pannello di source il file contenente l'entità di alto livello e far eseguire le funzioni che compaiono nel pannello di process, ovvero:

- Design Summery/Reports che consente di accedere ai report di progettazione, ai messaggi ed ai sintesi dei risultati;
- Synthesis che controlla la sintassi, fa la sintesi, visualizza RTL e report della sintesi;
- Implement Design che fornisce l'accesso al tool di implementazione e al tool di analisi di post-implementazione;
- Generate Programming File che genera il bitstream (.bit);
- Configure Target Device che fornisce l'accesso al tool di configurazione per la creazione del file di programmazione ed al tool di programmazione del dispositivo.

Una volta superato con successo tutte queste fasi, si ha, a lato host, il bitstream pronto della descrizione hardware dei circuiti integrati da scrivere nel FPGA. La funzione "Configure Target Device" apre una nuova finestra chiamata ISE iMPACT, oppure è possibile aprire questa finestra direttamente dal menù (andando su tool, voce iMPACT), il quale trasferisce (o programma) il bitstream nel FPGA. Questa finestra è suddivisa in quattro pannelli.

Il pannello situato in alto a sinistra, chiamato iMPACT Flows, ha la funzione principale di "Boundary Scan". Cliccando su questa funzione compare un'area completamente bianca sul pannello in alto a destra. A questo punto, bisogna cliccare sul tasto destro del mouse rimanendo nell'area bianca del pannello e compare un menù a tendina, in cui bisogna scegliere "Inizialize Chain".

Se il dispositivo è collegato, allora, visualizzerà in questo ultimo pannello i componenti programmabili. Dopo di che basterà scegliere il componente che si

vuole programmare con il bitstream adatto.

In centro a sinistra è presente un pannello chiamato iMPACT Processes, che visualizza le possibili operazioni eseguibili sul componente selezionato. Invece, nel pannello in basso vengono visualizzati i vari messaggi del processo attualmente in esecuzione, in tre schede diverse: console (visualizza i messaggi di stato), errors (visualizza i messaggi di errore) ed warnings (visualizza i messaggi di avviso). [3.5]

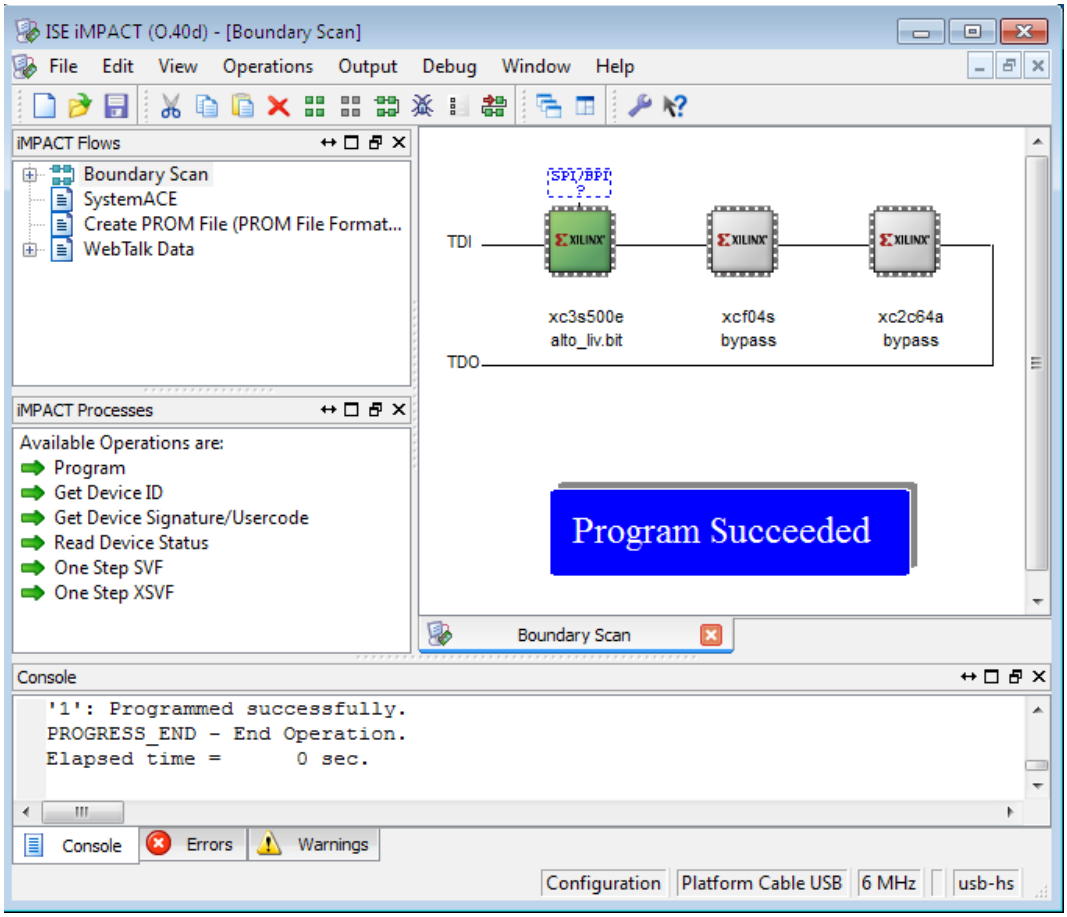


Figura 3.4: la finestra del xilinx ISE iMPACT

Se tutto è andato a buon fine, nel pannello in alto a destra, sotto i componenti, compare la scritta con lo sfondo blu “Program Succeded”, ciò significa che è stato trasferito correttamente il bitstream nel componente desiderato. I componenti possono essere programmati sia via cavo USB che mediante connettore JTAG.

3.2.2 Picoblaze

Picoblaze è il nome d'arte di un modulo scritto in VHDL da un progettista della Xilinx. Si tratta di una macchina a stati finiti programmabile che realizza in tutto le funzionalità di un piccolo microprocessore RISC a 8 bit.

Questo componente, compatto ed efficiente perché ottimizzato per le architetture FPGA, arricchisce il design della possibilità di realizzare piccoli programmi in assembler da utilizzare unitamente all'interfaccia progettata per comunicare con una serie di dispositivi on-board senza la necessità di intervento dall'esterno. [3.6]

Nel nostro progetto si è pensato di utilizzarlo per semplificare l'architettura del sistema, cioè per poter programmare via I^2C la coppia di sensori ottici e sostituire la programmazione effettuata per mezzo del PIC.

3.2.2.1 Architettura

Il processore è dotato internamente di 16 registri general purpose da 8 bit ciascuno; inoltre, è presente una ram scratch pad da 64 bytes, che permette di scrivere applicazioni estese e più complesse.

La logica di controllo gestisce il flusso delle istruzioni che possono essere memorizzate in un componente ROM; inoltre, è possibile gestire le interruzioni provenienti dall'esterno in modo piuttosto semplice, grazie alla logica di controllo degli interrupt presente nel modulo.

E' presente uno stack per chiamate a funzione in grado di gestire fino a 31 call annidate, il meccanismo di ritorno da funzione gestisce automaticamente e preserva i valori dei flag di zero e di parity.

La ALU è in grado di effettuare la maggior parte delle istruzioni aritmetico-logiche, fatta eccezione per le istruzioni in virgola mobile che richiederebbero un impiego più consistente rispetto alle dimensioni ridotte del picoblaze.

Sono inoltre presenti istruzioni di controllo del flusso del programma che, congiuntamente a istruzioni di compare e di test, permettono di realizzare comportamenti complessi come salti condizionati.

Il processore viene fornito insieme ad una serie di strumenti che permettono di scrivere un semplice programma in assembler direttamente su un editor di testo seguendo una sintassi precisa. Una volta terminato questo processo, utilizzando un compilatore, è possibile ottenere direttamente la sintesi in linguaggio vhdl che contiene un modulo di ROM al cui interno vengono mappate direttamente le istruzioni assembler di cui sopra.

Il modulo rom delle istruzioni e il modulo del processore vero e proprio sono costruiti per essere interfacciati direttamente l'uno con l'altro tramite un semplice port map nel blocco top level del design del progetto.

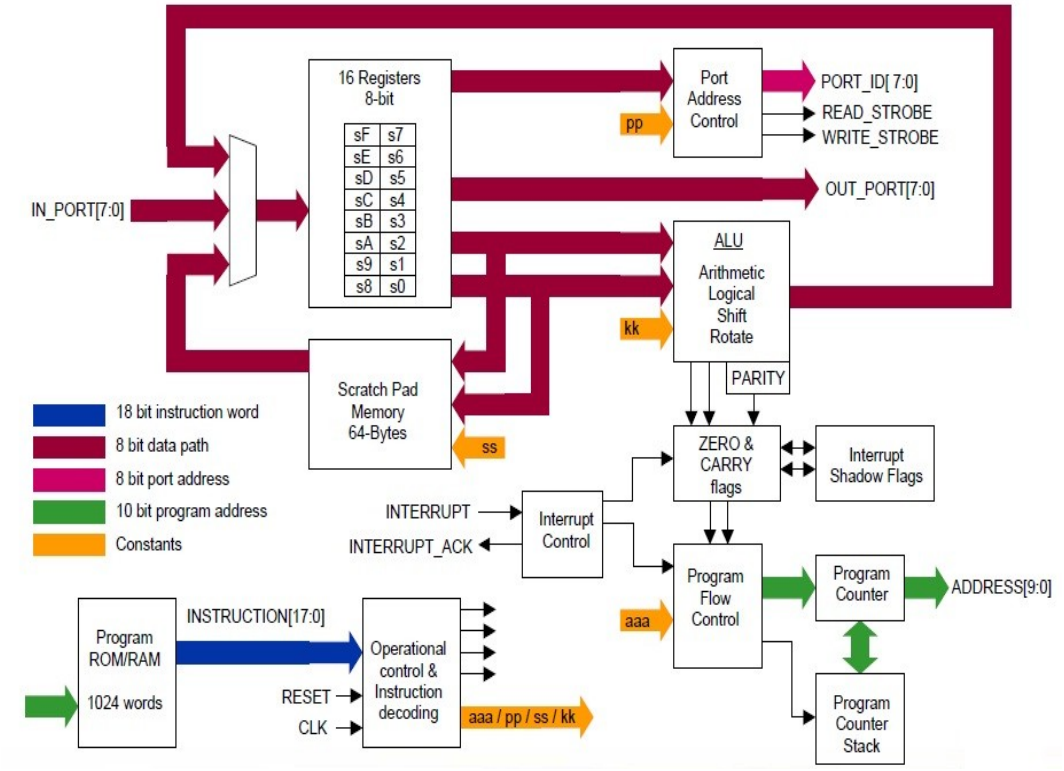


Figura 3.5: architettura del microprocessore RISC PicoBlaze

3.2.2.2 Programmazione

All'inizio bisogna scrivere e compilare il file per la programmazione del softcore Picoblaze. A tale scopo si può utilizzare un assembler gratuito offerto dalla Xilinx che permette di generare una serie di file per molte applicazioni; tra questi, quello di interesse per il design è il file con estensione .vhd (utilizza come linguaggio vhdl), che rappresenta proprio il componente con le istruzioni per il softcore all'interno. Per utilizzare l'assembler è necessario scrivere un file di testo con estensione .psm (programmable state machine) da fornire come parametro di input al programma. Il file deve seguire una serie di regole sintattiche e strutturali piuttosto semplici, ma non banali.

Il processore mette a disposizione un set di istruzioni minimale ma sufficiente a realizzare programmi piuttosto avanzati. Tramite la combinazione di istruzioni è, infatti, possibile ottenere comportamenti assimilabili a salti condizionati per la realizzazione di chiamate a funzioni e simili.

Ci sono una serie di problematiche legate al funzionamento del programma ed alla sua struttura di portata non indifferente. Primo fattore da prendere in considerazione è il fatto che il componente block memory in cui viene riversato il programma non ha dimensioni infinite. Infatti i bit di indirizzamento in ingresso a questo blocco e provenienti dal processore hanno parallelismo di 10 bit, quindi è possibile scrivere al massimo programmi di 1024 istruzioni. Un altro problema è sicuramente dato dal fatto che una volta terminate le istruzioni previste dall'esecuzione principale, il flusso del programma ritorna da capo e ricomincia l'esecuzione dall'inizio che in alcuni casi può essere un comportamento indesiderato. [3.7]

3.3 Libreria

In informatica, una libreria è un insieme di funzioni o strutture dati predisposte per essere collegate ad un programma software attraverso opportuno

collegamento. Il collegamento può essere statico o dinamico.

Il termine libreria nasce da un'errata traduzione dell'inglese library (letteralmente, biblioteca), ma ormai è così diffuso nel vocabolario dei professionisti da essere accettato quale esatta traduzione.

Lo scopo delle librerie software è quello di fornire una collezione di entità di base pronte per l'uso ovvero riuso del codice, evitando al programmatore di dover riscrivere ogni volta le stesse funzioni o strutture dati e facilitando così le operazioni di manutenzione. Questa caratteristica si inserisce quindi nel più vasto contesto del richiamo di codice all'interno di programmi e applicazioni ed è presente in quasi tutti i linguaggi. I vantaggi principali derivanti dall'uso di un simile approccio sono i seguenti:

- si può separare la logica di programmazione di una certa applicazione da quella necessaria per la risoluzione di problemi specifici, quali il calcolo di funzioni matematiche o la gestione di collezioni;
- le entità definite in una certa libreria possono essere riutilizzate da più applicazioni;
- si può modificare la libreria separatamente dal programma, senza limiti alla potenziale vastità di funzioni e strutture dati man mano disponibili nel tempo. [3.8]

3.3.1 Libusb

Libusb è una libreria che fornisce un accesso uniforme, a livello applicativo, alle periferiche USB attraverso diversi sistemi operativi. Il vantaggio nell'utilizzo di questa libreria è che, a differenze delle librerie Cypress, garantisce interoperabilità tra diversi sistemi operativi (vengono infatti distribuite diverse librerie sia per ambiente Windows che per Linux, sotto la GNU Lesser General Public License).

Libusb cerca di astrarre il più possibile i dettagli delle comunicazioni con la

periferica, offrendo un'interfaccia di alto livello con la quale lo sviluppatore accede ed opera con la periferica USB. Questo permette di apprendere in maniera rapida le operazioni basilari che si devono effettuare per realizzare un trasferimento, tralasciando particolari del protocollo che non interessano a livello applicativo. Nel progetto, libusb è stato utilizzato sia per i trasferimenti USB sia per interoperabilità del codice.

La prima versione della libreria è la 0.1, nata per sistemi basati su piatteforme linux, trascorso poco tempo viene effettuato un porting del codice anche per piatteforme windows che viene chiamato win32 ed è equivalente della versione 0.1 rilasciata per linux (le versioni della 0.1 rilasciati per windows sono denominate nel seguente modo: libusb-win32-1.x.x.x).

Lo svantaggio di libusb 0.1 è una certa mancanza di personalizzazione e di modifica rispetto alle funzioni standard presenti (ad esempio non vengono supportati i trasferimenti asincroni). Per questi motivi dopo poco tempo viene rilasciato una nuova versione per linux che è la 1.0 dove sono state aggiunte molte nuove feature come il supporto agli endpoint di tipo isochronous e sono state ottimizzate i feature precedenti, seguita dal release della stessa versione anche per windows che viene denominato windows_backend.

Nella versione 1.0 si è deciso di non tenere la compatibilità verso il basso che in questo caso con la versione 0.1. Infatti, le API (Application Programming Interface) della versione 1.0 sono differenti rispetto le API della versione 0.1.

Ad oggi le due versioni (1.0 per linux e per windows) procedono di pari passo, assicurando così la portabilità del codice da un sistema operativo all'altro. L'ultima release della versione 1.0 supporta il controller USB 3.0, viene dichiarato che è una versione esperimentale anche se maggior parte delle funzioni principali sono stati testati. [3.9]

3.3.1.1 Libusb win32

Libusb win32 è un porting di libusb versione 0.1 per sistemi operativi Windows (Windows 98 SE, Windows ME, Windows 2000, Windows XP, Windows Vista e Windows 7).

Se si desidera accedere ad una periferica USB mediante libusb win32, in primis è necessario creare un file INF, operazione immediata grazie alla presenza del utility "inf-wizard" all'interno della cartella bin, che permette di selezionare una periferica collegata e crearne il corrispettivo file INF. Infine viene offerta la possibilità di installare tale file completando cosi il processo di binding della periferica alla libreria.

La cartella bin contiene anche alcune utility correlate all'utilizzo della libreria per l'accesso ad una o più periferiche collegate. In particolare, testlibusb-win.exe permette di verificare se un dispositivo precedentemente collegato al driver creato da libusb win32 viene acceduto correttamente.

All'interno della cartella di libusb win32 (la cui ultima release è la 1.2.20) è presenta una cartella lib contenente diverse versioni della libreria (già compilate o meno) ed una cartella include con l'header file usb.h da includere in ogni progetto di applicazione utente comprendente la libreria.

Libusb win32 lega una periferica aperta ad un'interfaccia specifica. Questo significa che se si vogliono collegare più interfacce della stessa periferica, è necessaria aprirla una volta per ogni interfaccia.

Una periferica è un oggetto della classe "usb_device" aperto mediate il metodo "usb_open" che restituisce un handler per quella periferica, che sarà utilizzato in tutte le API che realizzano un trasferimento e selezionano un'interfaccia tra quelle presenti.

Prima di iniziare a leggere e/o scrivere si seleziona la configurazione e l'alternate setting per un'interfaccia (mediante le API usb_set_configuration,

usb_claim_interface e usb_set_altinterface).

E’ presente un API per la lettura dei dati ed un’altra per la scrittura (usb_bulk_read e usb_bulk_write).

Infine, è necessario rilasciare l’interfaccia (usb_release_interface) e chiudere la periferica (usb_close). [3.10]

3.3.1.2 Libusb windows_backend

Libusb windows_backend è un porting di libusb versione 1.0 per sistemi operativi Windows. Se si desidera accedere ad una periferica USB mediante libusb windows_backend, in primis è necessario installare un driver generale chiamato winusb. In questo caso a differenza di libusb win32 non c'è bisogno di creare il file INF perché si trova già dentro la cartella winusb. Necessita soltanto di essere modificato il file INF in modo opportuno che permetta di selezionare la periferica collegata. Infine, winusb offre la possibilità di installare tale file. Essendo un driver generale non offre la possibilità di verificare (tramite un tool) se i dispositivi collegati al driver viene acceduto correttamente o meno.

Una volta installato il driver generico, all'interno della cartella libusb 1.0 (o windows_backend) è presente una cartella include contenente l'header file libusb.h (dentro una sotto cartella libusb-1.0) da includere in ogni progetto di applicazione utente comprendente la libreria.

E’ bene sottolineare che le API disponibili per la versione 1.0 di libusb non sono compatibili con quelle della versione 0.1, quindi un progetto creato per una non può essere ricompilato per l’altra semplicemente cambiando la libreria collegata, ma è necessario andare a modificare tutte le API chiamate, processo piuttosto lungo che consiglia la creazione di un nuovo progetto indipendente dal primo.

Nel libusb windows_backend una periferica è un oggetto della classe “libusb_device” che può essere aperto mediate il metodo “libusb_open” che

restituisce un handler per quella periferica, che sarà utilizzato in tutte le API che realizzano un trasferimento e selezionano un'interfaccia tra quelle presenti.

Una periferica può essere aperto anche tramite il metodo "libusb_open_device_with_vid_pid" ed anch'esso restituisce un handler. Se è nota il vid e il pid della periferica con questo metodo si ha il vantaggio dell'apertura mirata del device. Mentre con il metodo precedente dovremo andare a interrogare tutte le periferiche usb collegate a host e trovare la periferica di interesse.

Prima di iniziare a leggere e/o scrivere si seleziona la configurazione e l'alternate setting per un'interfaccia (mediante le API libusb_set_configuration, libusb_claim_interface e libusb_set_interface_alt_setting).

E' presente un'unica API sia per la lettura che per la scrittura dei dati (libusb_bulk_transfer). La direzione del trasferimento è dedotta dal bit della direzione di indirizzo dell'endpoint.

Infine, è necessario rilasciare l'interfaccia (libusb_release_interface) e chiudere la periferica (libusb_close). [3.11]

3.3.2 OpenCV

OpenCV è una libreria open source di funzioni che fornisce un'implementazione adeguata di molti algoritmi di elaborazione dell'immagine sviluppata da Intel. Le funzioni messe a dispone dalla libreria sono molto superiori a quelle necessarie per svolgere il progetto, ma è comunque utile avere una visione più generale. L'utilizzo, che è stato fatto principalmente di questa libreria nel progetto, è per la visualizzazione delle immagini.

Uno dei punti di forza di OpenCV è il fatto di essere multi piattaforma. Nata nel 1999 come libreria per sistemi basati su piattaforma Windows, trascorse poco tempo prima che, nel 2000, venisse effettuato un porting del codice per piatteforme Linux. Ad oggi le due versioni procedono di pari passo, con lievissime differenze, assicurando così la portabilità del codice da un sistema

operativo all'altro.

Tutte le funzioni della libreria sono caratterizzate dal prefisso cv, mentre i nomi delle strutture hanno prefisso Cv ad eccezione della IplImage, la quale è ereditata dalla libreria IPL. Poiché le funzioni operano con immagini e con matrici numeriche, le dimensioni seguono rispettivamente la convenzione larghezza-altezza (X, Y) comune nella grafica e l'ordine righe-colonne (R,C).

Figura 3.6: logo della libreria OpenCV

3.3.2.1 Organizzazione della libreria

La libreria è strutturata in cinque sottolibrerie (dette anche moduli), ciascuna con funzionalità specifiche. Il modulo CXCORE è quello principale nonché indispensabile. Contiene le strutture dati di base con le rispettive funzioni di inizializzazione, le funzioni matematiche, le funzioni di lettura, scrittura e memorizzazione dati, le funzioni di sistema e di gestione degli errori. Il modulo CV contiene le funzioni relative all'image processing, le funzioni di analisi strutturale e del moto, di object detection e ricostruzione 3D. HIGHGUI contiene le funzioni GUI e quelle di salvataggio e caricamento immagini, nonché le funzioni di acquisizione video e di gestione delle telecamere. Il modulo ML (Machine Learning) contiene classi e funzioni relative all'implementazione e gestione di reti neurali, in particolare di tipo multilayer perceptrons (MPL), di

classificazione statistica e clustering di dati. Infine vi è il modulo CVAUX che contiene sia le funzioni basate su algoritmi ancora in fase di sperimentazione, il cui futuro è quello di migrare nel modulo CV, e sia le funzioni considerate obsolete e quindi non più supportate. Le funzionalità di tale modulo sono rivolte alla corrispondenza stereo, al tracking 3D, al riconoscimento degli oggetti (Eigen object).

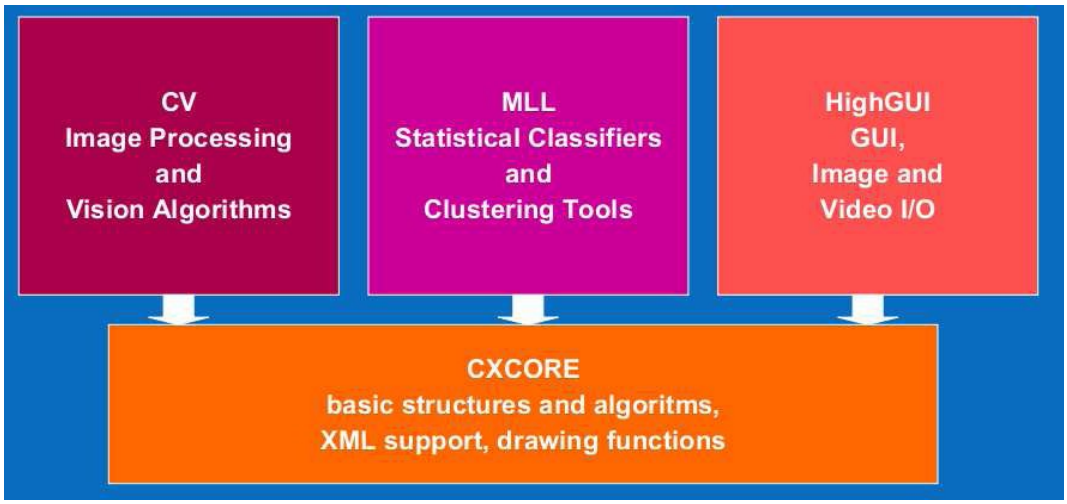


Figura 3.7: organizzazione della libreria openCV

3.3.2.2 Strutture dati

Non riusciamo sicuramente ad occuparci in modo approfondito di tutte le strutture dati presente nella libreria. Quindi introduciamo alcune delle strutture dati e funzioni base utili per iniziare ad operare con la libreria.

La maggior parte delle funzioni eseguono operazioni su strutture dati diverse tra loro, come le immagini (IplImage), le matrici (CvMat) e le liste dinamiche (CvSeq). Le funzioni che eseguono operazioni su questi tipi di dati ammettono sempre un virtuale oggetto array CvArr. La relazione esistente tra le strutture CvArr, cvMat e IplImage è di tipo object-oriented, dove la IplImage deriva dalla CvMat, la quale a sua volta deriva dalla CvArr.

La struttura più utilizzata ed importante è IplImage che è l'acronimo di Image Processing Library Image, ossia il formato standard utilizzato da Intel per

rappresentare un'immagine nelle API IPL e OpenCV. Tramite questa struttura dati è possibile gestire completamente tutto ciò che ruota intorno alle immagini, come caricamento e salvataggio, conversione di formato, elaborazione, filtraggio e visualizzazione.

La funzione che consente di creare e allocare un'immagine è la seguente:

```
cvCreateImage(CvSize size, int depth, int channels);
```

Il parametro size permette di assegnare le dimensioni relative all'ampiezza e all'altezza, il parametro depth assegna la profondità in bit ed infine channels che assegna il numero di canali.

Per poter operare con valori scalari si utilizza la struttura dati CvScalar dove l'unico membro è un array costituito da quattro elementi di tipo double. La struttura può essere utilizzata, non solo per le usuali operazioni aritmetiche ma anche, nelle operazioni di accesso ai singoli pixel di un'immagine.

Esistono due metodi per accedere, oppure modificare, i valori dei singoli pixel che sono metodo indiretto e metodo diretto.

Il metodo indiretto si basa sull'utilizzo della funzioni cvSet2D che serve per modificare il valore di uno specifico pixel.

```
void cvSet2D(const CvArr *arr, int idx0, int idx1, CvScalar value);
```

Il parametro arr è l'immagine da modificare, mentre idx0 e idx1 sono le coordinate relative allo specifico pixel, value rappresenta il nuovo valore da assegnare al pixel. Per un'immagine con un solo canale di dimensioni width x height, l'intensità del pixel di coordinate (i, j) dove $i \in [0, height - 1]$ e $j \in [0,width - 1]$, si può modifica nel seguente modo:

```
cvSet2D(img, i, j, v);
```

Per un'immagine multicanale si devono invece utilizzare tutti gli elementi del campo val, uno per ogni livello di colore:

scalar.val[0] = 'valore pixel'; scalar.val[1] = 'valore pixel'; ecc.

Invece, il metodo indiretto consiste nella manipolazione diretta di alcuni campi della struttura IplImage. Un'immagine può essere rappresentata come una matrice di dimensioni width x height. OpenCV la gestisce come un array monodimensionale, ossia le righe sono allineate una dopo l'altra. Per poter indicizzare l'array immagine si utilizza il campo widthStep, che contiene le dimensioni in bytes relative al passo con il quale le righe sono allineate. Il campo widthStep contiene quindi il numero di bytes necessari per passare dal pixel di coordinate (R, C) al pixel (R+1,C) della matrice. Per un'immagine ad un solo canale l'indice dell'array è dato da:

$$i * widthStep + j, \text{ con } i \in [0, height - 1] \text{ e } j \in [0,widht - 1]$$

Per un'immagine multicanale l'indice dell'array diventa:

$$i * widthStep + j * nChannels + k$$

dove $k \in [0, nChannels - 1]$. [3.12]

3.4 Microsoft Visual Studio

Visual Studio è un ambiente di sviluppo integrato (IDE) sviluppato da Microsoft, che supporta attualmente diversi tipi di linguaggio, quali C, C++, C#, F#, Visual Basic .Net e ASP .NET, e permette la realizzazione di applicazioni, siti web, applicazioni web e servizi web.

È inoltre un RAD (Rapid Application Development), ovvero un'applicazione atta ad aumentare la produttività aiutando il programmatore con mezzi, come l'Intellisense o un designer visuale delle forms.

Visual Studio è inoltre multipiattaforma: con esso è possibile realizzare programmi per server, workstation, pocket PC, smartphone e, naturalmente, per i browser.

Per la realizzazione del progetto è stato utilizzato Visual Studio 2008 (il nome in codice Orcas), ideato da Microsoft per programmatori che sviluppano per piattaforme Windows e .NET Framework 3.5, il quale presenta come innovazione principale introduzione di LINQ.

Nel Visual Studio 2010, l'ultimo IDE creato da Microsoft, sono stati presentati sviluppo di applicazioni per il .NET framework 4.0 e integrazione della libreria jQuery tra le principali innovazioni. [3.13]

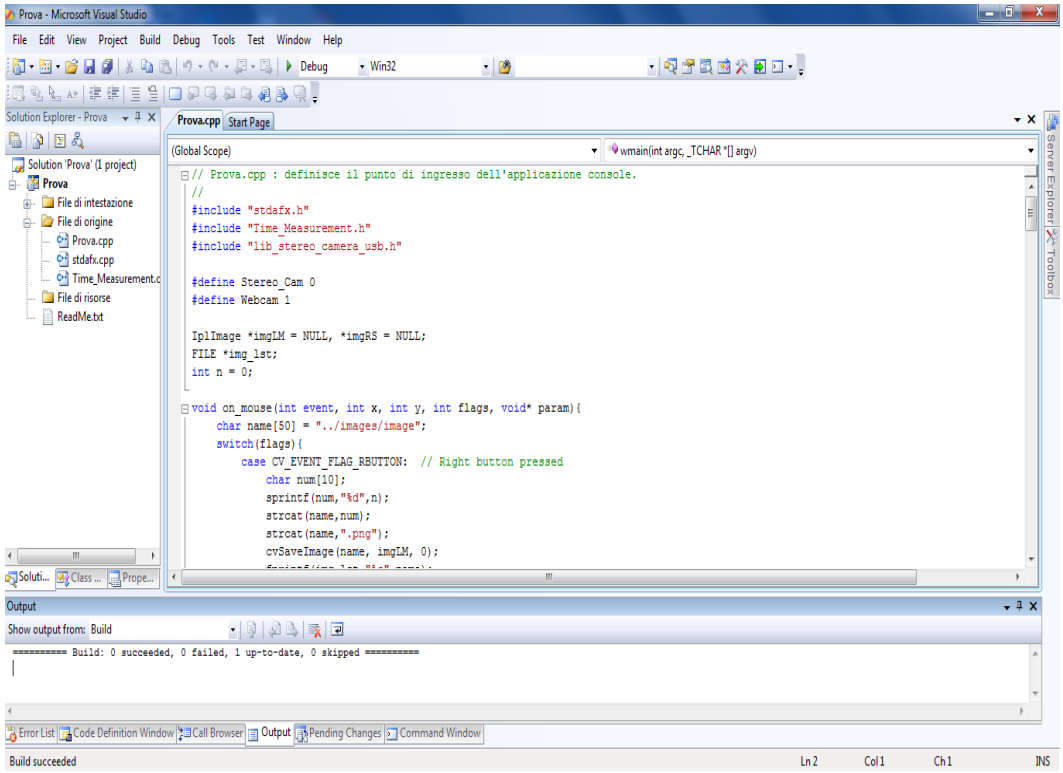


Figura 3.8: la finestra di Microsoft Visual Studio 2008

Nel nostro progetto è stato utilizzato questo IDE per scrivere i programmi a lato host per l'acquisizione e visualizzazione degli immagini in C++ utilizzando le librerie libusb e OpenCV. Tra l'altro questo IDE è stato molto utile anche nella fase di debug del codice scritto.

4. ASPETTI REALIZZATIVI E SPERIMENTAZIONE

Nel corso della tesi sono stati utilizzati diversi dispositivi; il presente capitolo ha lo scopo di descrivere le modalità con cui sono stati collegati e configurati questi dispositivi, ed anche come è stato sviluppato il codice ed i test che sono stati svolti con i relativi risultati.

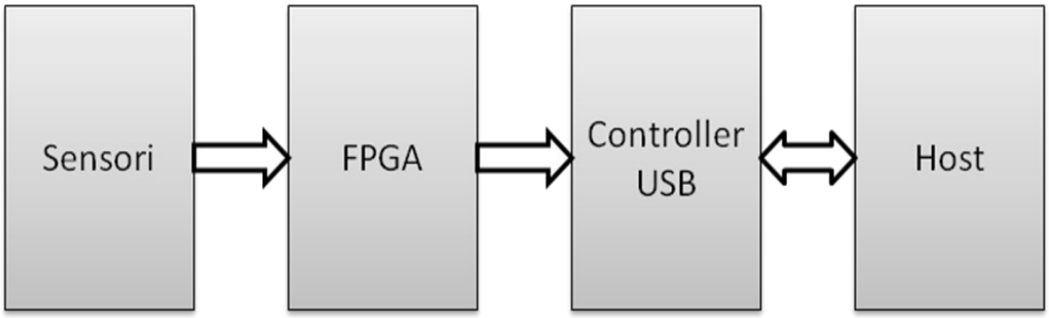


Figura 4.1: flusso dati tra sensori ed host

4.1 Lo stampato del prototipo

Questa parte del progetto, che è strettamente elettronico (hardware), è stata implementata da uno studente di ingegneria elettronica durante lo svolgimento della sua tesi. Lo riportiamo, anche se non in modo approfondito, per riuscire a comprendere meglio la parte software ed avere una visione generale del progetto.

Il prototipo stampato è stato suddiviso su due schede per mantenere distinte l'interfaccia USB 2.0 con la board di connessione verso la demo board dell'FPGA. Questa modularità consente di cambiare il tipo di interfaccia di comunicazione, senza dovere rifare completamente il layout.

Infatti è attualmente in studio da parte di altri studenti un'interfaccia GIGABIT ETHERNET e si sta inoltre vagliando l'ipotesi di usare il recente protocollo dell'USB 3.0.

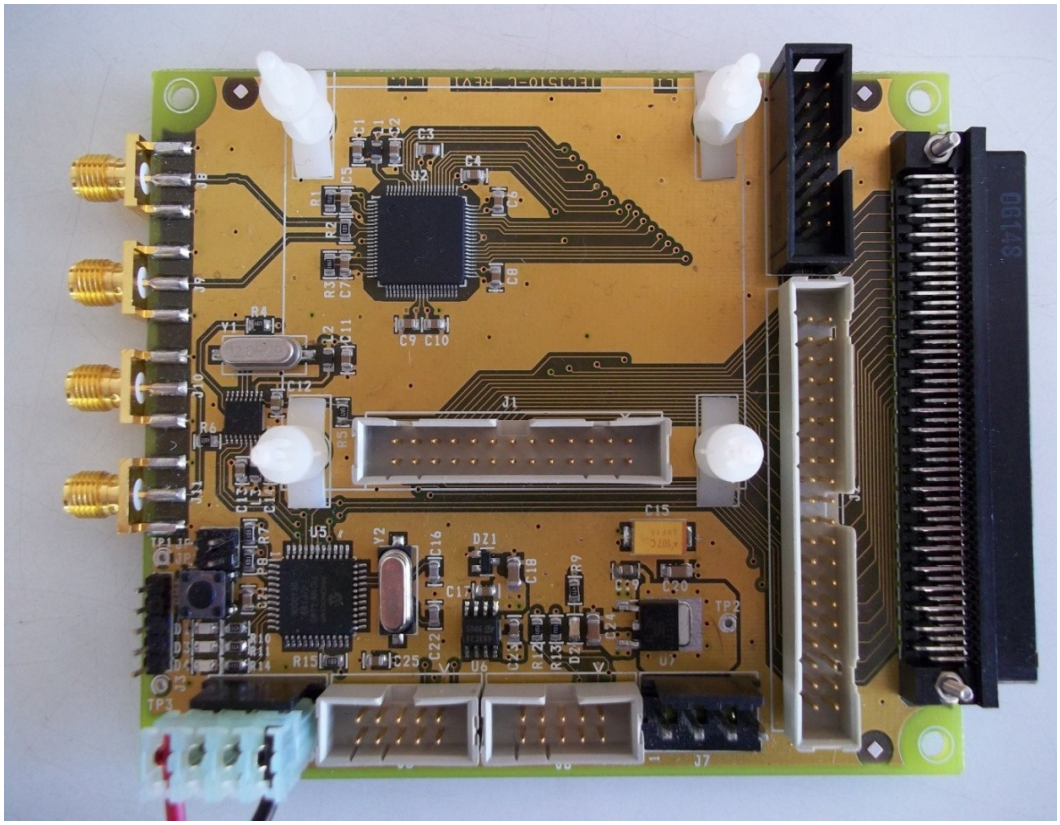


Figura 4.2: stampato della board base

La prima scheda (board base) contiene la logica di gestione dell'intero sistema. Al suo interno sono presenti il deserializer, le PLL, il microcontrollore e un linearizzatore locale di tensione che trasforma la 8,8V nella 3,3 V necessaria per alimentare tutti i dispositivi.

Questa scheda, inoltre, si interfaccia sia con i sensori, attraverso due connettori SMA, sia con entrambe le FPGA spartan3 e spartan6.

Per connettersi alla Development board SPARTAN 3 e SPARTAN 6 viene utilizzato il connettore HIROSE 100 pin.

L'interfaccia è stata implementata in modo da riuscire a connettersi con entrambe le spartan, poiché il progetto è stato prima realizzato su Spartan3 (le cui caratteristiche erano già note in precedenza) e solo successivamente sarà portato sulla più moderna e performante spartan6.

La board di comunicazione contiene il controller USB 2.0. Preso atto che l'I/O messo a disposizione sui connettori esterni delle due demo board spartan non è sufficiente al totale controllo dell'FX2, vengono inseriti dei jumper per eseguire il pull up o pull down delle linee, che non necessitano il comando diretto dell'FPGA.

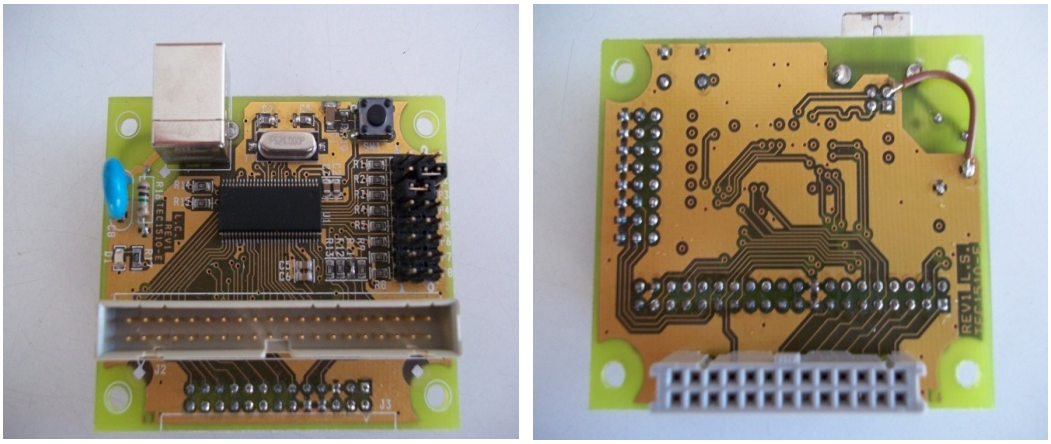


Figura 4.3: stampato del prototipo

Le due schede sono connesse in pila grazie a quattro sostegni in plastica che garantiscono la solidità meccanica.

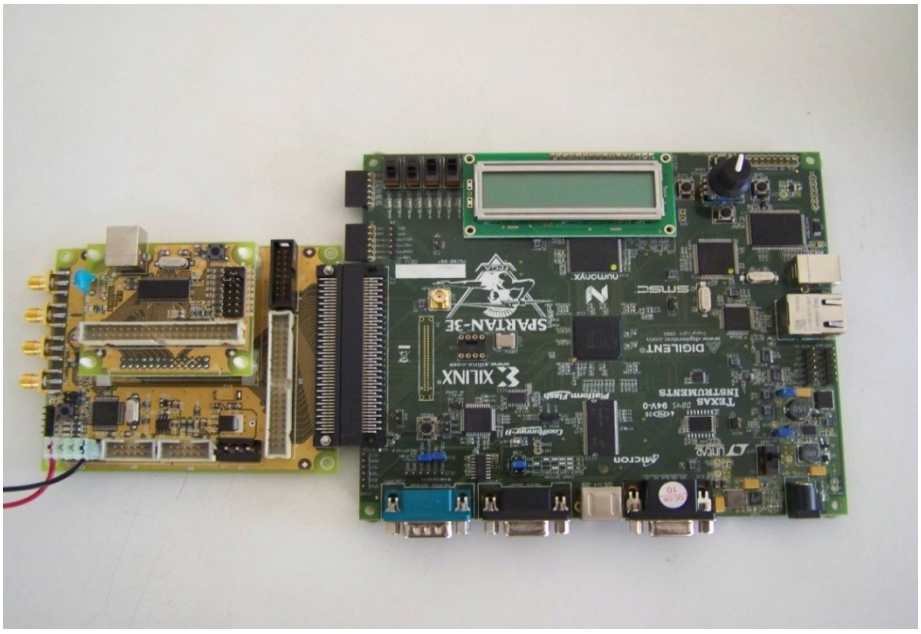


Figura 4.4: il prototipo nel complesso

L’alimentazione viene portata al prototipo da un’altra scheda, il cui unico compito è linearizzare la tensione proveniente da un trasformatore a muro e portarla ad un valore di 8V.

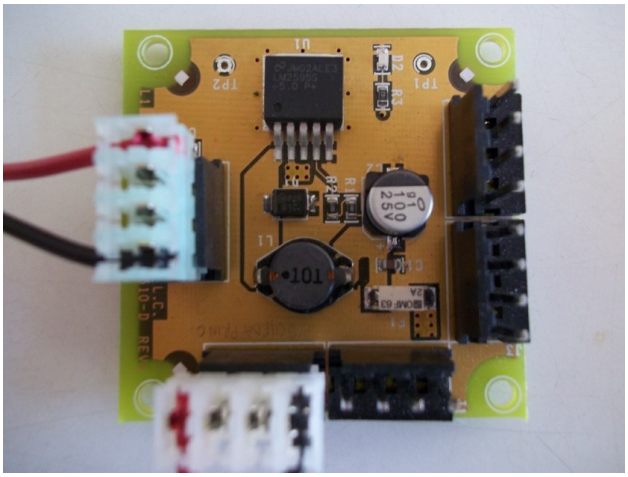


Figura 4.5: blocco di alimentazione

4.2 Caricamento driver

Quando avviene il collegamento tra il prototipo contenente il controller Cypress e l'host attraverso il cavo USB, inizia la fase di enumerazione del dispositivo. Durante questa fase, il sistema operativo deve selezionare e caricare un driver compatibile con la periferica.

Windows cerca informazioni all’interno di tutti i file INF (file di testo che contengono informazioni riguardo una o più periferiche e situati in una precisa cartella, ad esempio WINDOWS\System32\inf) confrontando il Vendor ID ed il Product ID del dispositivo con quelli memorizzati all’interno di tali file INF. Se viene trovato un match, viene caricato il driver (file in formato .sys) specificato all’interno di tale file. Nel caso in cui non venga trovato niente, Windows cerca di utilizzare driver della stessa classe, sottoclasse o protocollo della periferica.

4.2.1 Driver Cypress

All'interno della SuiteUSB viene fornito, oltre all'INF file CyUSB.inf, il driver

CyUSB.sys che offre la possibilità di comunicare con ogni dispositivo Cypress compatibile. E' un driver general purpose che riconosce i comandi USB di basso livello senza implementarne di più alto livello, dipendenti dalla classe del dispositivo per i quali sono necessarie le opportune librerie.

La funzione principale di questo driver è inizialmente quella di permettere l'invio di comandi elementari verso la periferica per i test preliminari, atti a verificarne il corretto funzionamento, come per esempio alcuni trasferimenti o la lettura della configurazione di default del nostro dispositivo, salvo poi permettere ad un'applicazione utente di comunicare con la periferica.

Le caratteristiche principali del driver sono le seguenti:

- compatibile con il Windows Driver Model (WDM);
- certificato WHDL (privo di firma);
- supporta endpoint di tipo Control, Bulk, Interrupt ed Isochronous;
- supporta il Wake Up remoto;
- supporta la connessione multipla di più dispositivi USB contemporaneamente;
- supporta la modifica del GUID senza il re-build del driver;
- supporta l'esecuzione di script eseguiti all'avvio del sistema per trasferimenti di tipo control, atti alla configurazione del dispositivo.

4.2.2 Il file CyUSB.inf

Il file CyUSB.inf fornito nella SuiteUSB necessita di essere modificato in alcune sue parti, in particolar modo devono essere aggiunti il Vendor ID (che per i prodotti Cypress sarà sempre 04B4H) ed il Product ID (dipendente dal particolare dispositivo utilizzato con cui vogliamo collegare il driver).

Ad esempio, per un dispositivo con VID 04B4H, PID 8613H, la riga

;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb, USB\VID_XXXX&PID_XXXX

dovrà essere sostituita con

;%VID_04B4&PID_8613.DeviceDesc%=CyUsb, USB\VID_04B4&PID_8613

Se si vuole far riconoscere la periferica con un nome differente rispetto a quello di default si deve modificare la stringa DeviceDesc. Sarà sufficiente sostituire la riga

VID_XXXX&PID_XXXX.DeviceDesc="Cypress USB Generic Driver (3.4.2.00)"

con

VID_04B4&PID_8613.DeviceDesc="Nome Personalizzato"

Se si desiderano apportare ulteriori modifiche al file è consigliabile creare una copia sia del file INF che del driver CyUSB.sys, rinominandoli in modo opportuno.

I parametri editabili sono tutti nella sezione [Strings], in particolare :

CYUSB_Provider = "Cypress"

CYUSB_Company = "Cypress Semiconductor Corporation"

CYUSB_Description = "Cypress Generic USB Driver"

CYUSB_DisplayName = "Cypress USB Generic"

CYUSB_Install = "Cypress CYUSB Driver Installation Disk"

Infine, è modificabile anche il Guid, ovvero il Global Unique Identifier, un numero a 16 byte utilizzato dalle applicazioni per accedere alle periferiche. L'univocità tra due Guid non è garantita, ma essendo le possibili combinazioni un numero molto alto (2^{128}), la probabilità di generare due Guid identici è trascurabile.

Per generare un nuovo Guid è disponibile l'utility GuidGen, distribuita con Microsoft Visual Studio.

Un'altra interessante opportunità offerta da questo driver è il trasferimento automatico verso il Control endpoint di default (endpoint di indirizzo 0) all'avvio della periferica.

Questo trasferimento può essere usato creando uno script, ad esempio, per caricare un firmware personalizzato ogniqualvolta la periferica venga collegata all'host, provocando una re-enumerazione del dispositivo. E' quindi fondamentale che il nuovo firmware imponga un nuovo PID, altrimenti la periferica continuerà ad essere re-enumerata in un loop infinito, in quanto verrà sempre messo in esecuzione lo script.

In alternativa, si deve configurare il file INF in modo tale che venga eseguito lo script solo se la periferica collegata ha una particolare coppia {VID, PID} (ad esempio quelli di default).

4.2.3 Riconoscimento della periferica

Dopo aver modificato il file INF, per il corretto funzionamento della periferica è necessario associarle il driver CyUSB.sys.

Questa procedura è dipendente dal sistema operativo usato, ma comunque deve essere fatta manualmente. Innanzitutto, se il sistema operativo utilizzato è a 64 bit, occorre disabilitare il caricamento dei soli driver con firma digitale. Per fare ciò, durante il boot di sistema, occorre tenere premuto il tasto F8, e successivamente selezionare l'opzione:

Disable Driver Signature Enforcement

Dopo aver collegato la periferica, il sistema operativo segnalerà che nessun driver è stato trovato. Quindi bisogna forzare manualmente l'installazione, eseguendo i seguenti passi:

- Entrare nel Device Manager;
- Una volta individuata la periferica, che si trova nella sezione

"Universal Serial Bus Controllers", cliccare due volte;

- Selezionare "Driver" nella finestra di dialogo;
- Inserire il Path del nostro driver. Per fare ciò, selezionare l'opzione:

  Don't search. I will choose the driver to install

  e successivamente

  Have Disk

- Dopo aver dato conferma, l'installazione è terminata.

L'installazione del driver libusb 0.1 e winusb per libusb 1.0 sono molto simili alla procedura vista in precedenza, per cui si è scelto di non ripetere.

4.3 Sviluppo e caricamento firmware

Anche questa parte del progetto, cioè lo sviluppo del firmware, è stato implementato da uno studente di ingegneria elettronica durante lo svolgimento della sua tesi. Per avere una visione generale del progetto e per riuscire a comprendere il software sviluppato dal sottoscritto, è necessario la conoscenza anche non approfondita di questa parte.

Per sviluppare un firmware personalizzato, occorre sicuramente avere ben chiare le modalità con cui la periferica dovrà comunicare con l'host. A tal proposito occorre pianificare quali registri saranno modificati dal firmware e decidere quali bit di tali registri saranno modificati, mediante un attento studio del Technical Reference Manual del controller. Essendo il firmware composto di più file, è utile partire da un firmware sviluppato che includa già le necessarie librerie.

Il firmware per i controller Cypress FX2 si compone di diversi file sorgenti; in primis è presente il file dscr.a51**,** ovvero il file contenente tutte le informazioni sui descrittori della periferica, sia nel caso di Full Speed che nel caso High Speed. Ogni descrittore si riconosce dalla dichiarazione mediante il suo nome, seguito da una serie di membri che ne definiscono i valori. Per identificarne la fine, vi è il

nome del descrittore concatenato con la parola End.

E' inoltre necessario includere il file relativo al framework che si intende utilizzare, chiamato tipicamente fw.c**.**

L'utilizzo di un framework permette di velocizzare lo sviluppo della periferica USB, in quanto esso implementa il codice per la CPU per l'inizializzazione del chip, gli handler per le richieste del protocollo USB e quelle relative alla gestione della sospensione della periferica.

In primis vengono inizializzate tutte le variabili di stato interne, successivamente viene chiamata la funzione Td_Init, implementata poi in un altro file sorgente, quindi vengono abilitate le interruzioni ed infine vengono ripetute, in sequenza, tre operazioni:

- invocazione della funzione Td_Poll (anch'essa da implementare);
- verifica di eventuali richieste in attesa di completamento;
- verifica di eventi che richiedano la sospensione della periferica.

Vengono inoltre inclusi alcuni header file, necessari per la compilazione:

- **fx2.h** : contiene alcune costanti relative all'EZ-USB, tipi di dato, macro ed i prototipi delle funzioni di libreria;
- **fx2regs.h** : dichiarazione dei registri dell'EZ-USB;
- **syncdly.h** : contiene la macro syncdelay, necessaria dopo il settaggio di alcuni registri in modo da rendere effettiva la modifica.

L'altro principale file sorgente (chiamato genericamente device.c) da progettare riguarda la configurazione vera e propria del nostro dispositivo. Contiene infatti l'implementazione di alcune funzioni che vengono invocate dal framework, in particolare Td_Init e Td_Poll.

Td_Init verrà eseguita allo startup della periferica e al suo interno vengono settati i valori di tutti i registri relativi agli endpoint, che altrimenti avranno una

configurazione di default. Td_Init inizializza le FIFO collegate agli endpoint ed i relativi buffer di I/O. All'interno di questa funzione è possibile, ed in alcuni casi necessario, modificare anche parametri relativi alla CPU (frequenza del clock, utilizzo di un clock interno o esterno), sulla polarità dei segnali (quali SLWR, SLRD), configurazione dei flags e soprattutto sulla modalità di funzionamento della nostra periferica (Slave FIFO o GPIF Mode)

Td_Poll invece viene invocata dal framework ogniqualvolta la periferica si trovi in uno stato di inattività. Può prevenire il framework dal rispondere a richieste di sospensione.

Altre funzioni presenti in questo file sorgente riguardano richieste di configurazione, operazioni da effettuare prima o dopo la sospensione della periferica e relative alla gestione delle interruzioni.

Una volta sviluppato il firmware il modo più semplice e veloce per caricarlo sul controller Cypress è sicuramente l'utilizzo del Control Center.

Dopo aver collegato la periferica ed aver atteso che essa sia correttamente riconosciuta dal sistema, selezionarla dalla struttura ad albero e cliccare su Program FX2 -> Program RAM.

Si aprirà una finestra che permette all'utente di selezionare quale sia il file (in formato esadecimale) con cui si intende riprogrammare il controller.

Si ricorda che è consigliata la modifica del PID all'interno del firmware, oppure è necessario continuare a caricare il firmware tramite Control Center per evitare conflitti ed incongruenze.

Continuando ad usare Control Center quindi, dopo aver caricato il firmware con PID modificato, il controller non sarà più visibile dagli applicativi Cypress in quanto viene "catturato" dal driver di libusb.

4.4 Test

La sperimentazione ha visto una prima fase in cui sono stati effettuati dei test singoli di letture e scritture tra l'host ed il controller, in modo tale da avere relativa certezza del corretto collegamento tra essi. Successivamente, è cominciato lo sviluppo di un applicativo che gestisse un flusso dati proveniente dal controller utilizzando le API fornite da libusb 0.1.

A seguito, si è elaborato un altro software che usa la libreria libusb 1.0, in modo da avere un diretto confronto tra le due versioni in termini di prestazioni e usabilità.

Dopo di che, una volta acquisita la certezza della velocità e la correttezza dei dati in ricezioni, si è sviluppato una libreria che svolgesse le seguenti funzioni: connessione alla periferica, trasferimento di una certa dimensione, individuazione dei frame (o immagini) nei dati trasferiti e chiusura della connessione.

Infine, è stato implementato un codice il quale utilizza la libreria per controllare il corretto funzionamento e visualizza le immagini acquisite.

L'ambiente operativo ha previsto l'utilizzo del controller Cypress CY7C68013A sullo stampato del prototipo, dell'FPGA Xilinx Spartan 3 e di un PC in dual boot così configurato:

Processore: Intel Pentium 2 Dual Core 2.20 Ghz

Memoria RAM : 4 GB

Sistema Operativo1: Windows 7 Professional 64 bit

Sistema Operativo2: Ubuntu 10.10

4.4.1 Test delle due versioni di libusb

In primo luogo sono stati fatti dei test per avere un'idea di quale fosse il limite della velocità di trasferimento tra l'host ed il controller, ricorrendo a libusb, e se

fosse adatto alle nostre esigenze. Era molto importante il test in quanto la velocità di trasferimento rappresenta il collo di bottiglia del progetto, ovvero se non avessimo raggiunto una certa velocità di trasferimento non sarebbe stato possibile andare avanti con il progetto stesso.

Quindi, mi sono concentrato a testare le versioni di libusb per i due sistemi operativi (linux e windows). Il primo test è stato svolto sul sistema operativo linux con la versione 1.0 di libusb poiché durante lo svolgimento dei test con la versione 0.1 con windows è uscita la versione 1.0 della libreria per linux, quindi si è deciso di testare direttamente l'ultima versione rilasciata.

4.4.1.1 Test libusb 1.0

Il codice sviluppato svolge dei passi semplici tipo connettersi al dispositivo, fare dei trasferimenti, analizzare i dati trasferiti e, infine, disconnettersi. In questo caso, attua dei trasferimenti BULK (512 byte in un unico trasferimento). L'API del libusb permette di fare trasferimenti multipli di 512. Dunque, è possibile delegare al libusb un trasferimento da 512*n, dove n rappresenta un numero intero. Ad esempio se n è pari a 100, il lato cliente, anziché fare un ciclo di 100 trasferimenti da 512 byte, delega al libusb la totale responsabilità. La libreria libusb svolgerà questi cicli, da parte del cliente, per leggere i dati; ed una volta letti, i dati ritorneranno all'utente.

Una volta trasferiti i dati si controlla la presenza degli eventuali errori e si stampano le statistiche; ad esempio, la velocità massima, minima e media di trasferimento. I dati generati dal dispositivo sono molto banali, una sequenza da 0 a 50 e la somma di questi valori; successivamente si ripete la sequenza. Facendo delle prove, si nota che la velocità di trasferimento migliora ad aumentare del valore da moltiplicare a 512, ossia la velocità di trasferimento corrisponde a 512*k, chiamato PACKET_BULK_LEN, ed all'aumentare di k aumenta questo valore (dopo una certa quantità, comunque si raggiunge la saturazione).

Così venne l'idea di creare uno script che iterasse in modo automatico il valore

di k tra 100 e 1000 ed invocasse il codice con il valore aggiornato. Ad ogni iterazione il valore k incrementava di 10. Si notò che ci potevano essere ancora dei margini di miglioramento. Perciò è stato aumentato il range di iterazione fino a 2000. Nonostante ciò il valore della velocità di trasferimento non raggiunse la saturazione, quindi si è deciso di portare il limite superiore a 5000.

Ogni lettura nel codice si ripete n volte, nel mio caso n corrisponde a 200; ovvero itero 200 volte la lettura per ogni valore di PACKET_BULK_LEN nel codice sviluppato. La ripetizione delle letture serve per una maggior precisione nel calcolo delle statistiche che sono state salvate in un file nel seguente modo:

PACKET_BULK_LEN	MASSIMO	MINIMO	MEDIA	DELTA
512*100	...	...	...	...
512*110	...	...	...	...
512*...	...	...	...	...
512*5000	...	...	...	...

Il massimo rappresenta la velocità di trasferimento più elevato raggiunto nei 200 trasferimenti per quel valore di PACKET_BULK_LEN; il minimo rappresenta la velocità di trasferimento più lento; la media rappresenta, invece, la media aritmetica dei valori di trasferimenti, mentre il delta rappresenta la differenza tra il massimo ed il minimo.

Questi sono i risultati ottenuti dall'analisi del file:

- valore massimo tra i massimi di trasferimento corrisponde a PACKET_BULK_LEN = 512*4780; massimo = 39.0139465332 MB/s; minimo = 38.3947982788 MB/s; media = 38.9292895508 MB/s; delta=0.6191482544;
- valore massimo tra i minimi di trasferimento corrisponde a PACKET_BULK_LEN = 512*4210; massimo = 38.9952774048 MB/s; minimo = 38.8466262817 MB/s; media = 38.9361132813 MB/s;

delta=0.1486511230;

- valore massimo tra i media di trasferimento corrisponde a PACKET_BULK_LEN = 512*4750; massimo = 39.0014648438 MB/s; minimo = 38.5367774963 MB/s; media = 38.9583374023 MB/s; delta=0.4646873474;
- valore minimo tra i valori del delta corrisponde a PACKET_BULK_LEN = 512*4150; massimo = 38.9805641174 MB/s; minimo = 38.8400688171 MB/s; media = 38.9375952148 MB/s; delta=0.1404953003.

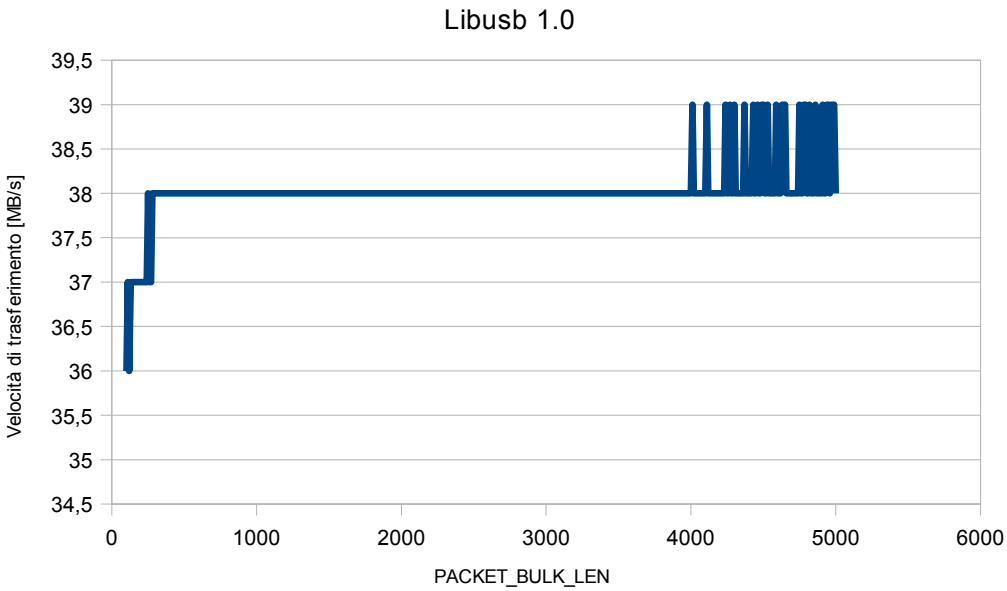


Figura 4.6: la relazione tra PACKET_BULK_LEN e velocità di trasferimento

Nella figura è stata rappresentata le prime due colonne del file, ovvero i valori presente nella colonna di PACKET_BULK_LEN (i valori riportati nella figura vanno moltiplicati per 512) e i valori presente nella colonna di massimo.

Ciò che si osserva dai risultati è che ad un certo punto la velocità di trasferimento raggiunge la saturazione (per il valore di PACKET_BULK_LEN nei d'intorni a 512*4000 circa) e per i grandi valori di PACKET_BULK_LEN la velocità media di trasferimento non cambia di molto, ma migliora notevolmente il

delta; dunque la varianza è piccola, ossia i diversi valori di velocità di trasferimento sono molto vicini al valore medio.

Bisogna tener conto che la velocità di trasferimento calcolata nel codice è approssimata e sarà difficile calcolarla in maniera precisa. Questo è dovuto al fatto che il valore viene calcolato nel seguente modo:

- si preleva il tempo dal sistema (system call);
- si fa il bulk transfer (system call);
- si preleva nuovamente il tempo dal sistema (system call).

Successivamente, si fa la differenza tra i due valori prelevati e si divide per il numero di byte trasferiti per ricavare la velocità di trasferimento.

4.4.1.2 Test libusb win32

Durante lo sviluppo del codice con il Microsoft Visual Studio 2008, per evitare errori di compilazione, è necessario includere nel progetto l'header file "usb.h", disponibile all'interno della cartella "include" di libusb win32, aggiungendo in testa al file sorgente la seguente linea di codice:

```
#include "...\libusb-win32-bin-1.2.2.0\include\usb.h"
```

Infine, la libreria (contenuta in "libusb-win32-bin-1.2.2.0\lib\msvc\libusb.lib") deve essere inclusa nel progetto aggiungendola alla cartella "Source File" (click destro -> Add Existing Item).

La struttura del codice (o file sorgente) rimane inalterato dal punto di vista logico, cioè svolge gli stessi passi che venivano svolti dal codice nel test di linux, ma questa volta sono cambiate le chiamate alle funzioni con i variabili da passare ossia l'API, visto che libusb win32 è equivalente alla versione 0.1 della libusb per linux.

Questi sono i risultati ottenuti dall'analisi:

- valore massimo tra i massimi di trasferimento corrisponde a PACKET_BULK_LEN = 512*120; massimo = 39.8003311157 MB/s; minimo = 6.8637719154 MB/s; media = 37.6318931580 MB/s; delta=32.9365592003;
- valore massimo tra i minimi di trasferimento corrisponde a PACKET_BULK_LEN = 512*230; massimo = 37.9082756042 MB/s; minimo = 34.3418960571 MB/s; media = 37.3424758911 MB/s; delta=3.5663795471;
- valore massimo tra i media di trasferimento corrisponde a PACKET_BULK_LEN = 512*4470; massimo = 38.5313644409 MB/s; minimo = 29.2975978851 MB/s; media = 38.0610923767 MB/s; delta=9.2337665558;
- valore minimo tra i valori del delta corrisponde a PACKET_BULK_LEN = 512*230; massimo = 37.9082756042 MB/s; minimo = 34.3418960571 MB/s; media = 37.3424758911 MB/s; delta=3.5663795471.

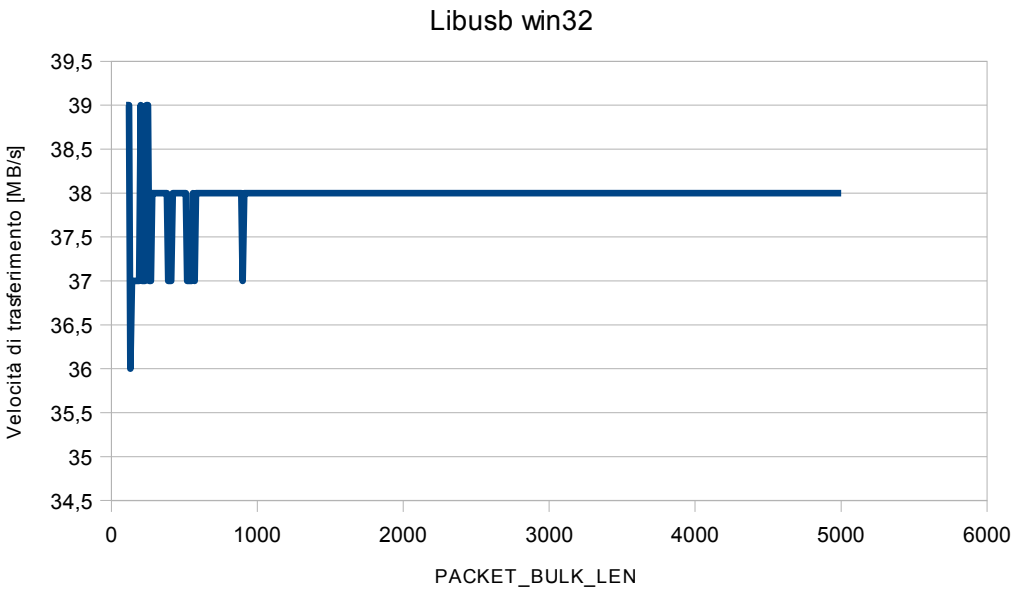


Figura 4.7: rappresentazione grafica dei dati salvati nel file

Ciò che si osserva nei risultati è che i trasferimenti per i valori piccoli di PACKET_BULK_LEN risultano più veloci in windows, mentre nel linux è il contrario. Questa differenza potrebbe essere ricondotta alle differenze delle versioni. Si nota che in windows il delta assume valori abbastanza grandi, mentre il valore medio è molto vicino al valore massimo. Ciò significa che la maggior parte dei valori di velocità di trasferimento è vicino al valore medio.

4.4.1.3 Test libusb windows_backend

Anche in questo caso per evitare errori di compilazione, è necessario includere nel progetto di Microsoft Visual Studio 2008, l'header file "libusb.h", disponibile all'interno della cartella "include\libusb-1.0" di libusb_1.0, aggiungendo in testa al file sorgente la seguente linea di codice:

```
#include "C:\libusb_1.0\include\libusb-1.0\libusb.h"
```

Infine, la libreria (contenuta in "libusb_1.0\MS32\static\libusb-1.0.lib") deve essere inclusa nel progetto aggiungendola alla cartella "Source File" (click destro -> Add Existing Item), oppure può essere aggiunta nel seguente modo: dal menù Project > %nome_del_progetto%Properties (oppure click destro sul nome del progetto -> Properties), selezionare Configuration Properties > Linker > Input ed aggiungere alla riga Additional Dependencies "C:\libusb_1.0\MS32\static\libusb-1.0.lib". Come si nota, i nomi dei file oltre ai percorsi è l'unica differenza rispetto al caso precedente.

I passi eseguiti dal codice o il file sorgente sono gli stessi descritti nei punti precedenti, cambiano soltanto le chiamate alle funzioni e variabili da passare, visto che è cambiata la versione rispetto al caso precedente.

Questi sono i risultati ottenuti dall'analisi:

- valore massimo tra i massimi di trasferimento corrisponde a PACKET_BULK_LEN = 512*4000; massimo = 41.4695663452 MB/s; minimo = 40.3984069824 MB/s; media = 40.7726135254 MB/s;

delta=1.0711593628;

- valore massimo tra i minimi di trasferimento corrisponde a PACKET_BULK_LEN = 512*4890; massimo = 41.1737785339 MB/s; minimo = 40.4695701599 MB/s; media = 40.8743057251 MB/s; delta=0.7042083740;
- valore massimo tra i media di trasferimento corrisponde a PACKET_BULK_LEN = 512*4890; massimo = 41.1737785339 MB/s; minimo = 40.4695701599 MB/s; media = 40.8743057251 MB/s; delta=0.7042083740;
- valore minimo tra i valori del delta corrisponde a PACKET_BULK_LEN = 512*3550; massimo = 40.8143653870 MB/s; minimo = 40.3487319946 MB/s; medio = 40.6609191895 MB/s; delta=0.4656333923.

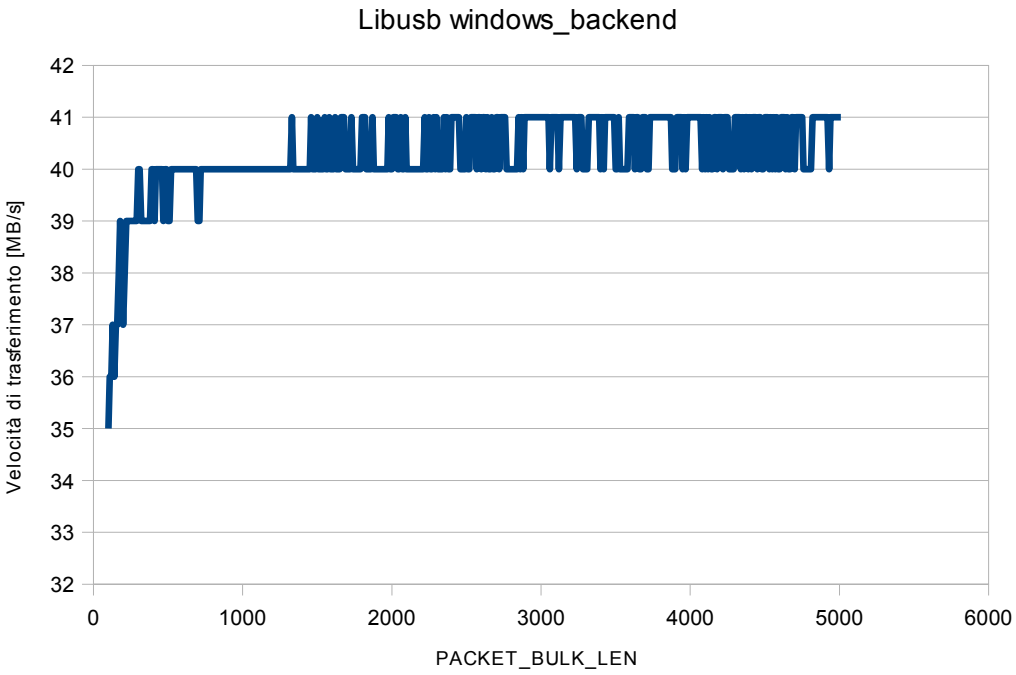


Figura 4.8: rappresentazione grafica della relazione tra PACKET_BULK_LEN e velocità di trasferimento dei dati salvati nel file

Ciò che si osserva nei risultati è che allineando le versioni si avvicinano anche i comportamenti dei due sistemi operativi. Osservando i risultati del test ci si accorge che assomigliano ai risultati ottenuti nel test su linux. Si nota che il valore di delta è diminuita, anche notevolmente, rispetto ai test svolti con la versione win32 della libreria libusb.

In questa versione si raggiunge la saturazione della velocità di trasferimento un po' prima rispetto alla stessa versione rilasciata per linux, il motivo è molto probabilmente nella gestione del sistema operativo visto che la stessa situazione si presenta anche nella versione win32. La velocità di trasferimento raggiunto dai test è soddisfacente, ossia ci permette di proseguire con il progetto.

4.4.1.4 Test libusb 1.0 con thread

Una volta superati i test della velocità di trasferimento, abbiamo pensato di suddividere le responsabilità, poiché sembrava di aver caricato troppo il codice. Così venne l'idea di creare dei thread, ognuno con una sola responsabilità, che è stato realizzato utilizzando la libreria pthread.

POSIX Thread, solitamente indicato come Pthread, è uno standard per i thread di POSIX. Lo standard POSIX.1c, estensioni dei thread (IEEE 1.3.1c-1995), definisce le API per creare e manipolare i thread. Le implementazioni delle API sono disponibili su molti sistemi UNIX, POSIX conforme, come sistemi operativi FREEBSD, NETBSD, OPENBSD, GNU/LINUX.

Pthread definisce un insieme di tipi, funzioni e costanti per il linguaggio di programmazione C. E' implementato con un header pthread.h ed una libreria thread.

Nel codice modificato sono stati creati tre thread, uno che si occupa soltanto del trasferimento, il secondo del salvataggio dei dati nella memoria permanente, mentre l'ultimo ha il compito del controllo degli errori nei dati ricevuti. I primi due thread dovevano condividere una zona di memoria perché il primo thread

dopo aver ricevuto i dati lo scriveva nella memoria volatile mentre il secondo lo trasferiva nella memoria permanente. Quindi è stata creata una struttura (buffer circolare con il semaforo) apposta poiché i due thread riescano a condividere i dati senza collisioni.

Per sincronizzare i diversi thread abbiamo dovuto utilizzare il semaforo. L'API POSIX semaforo funziona con i thread POSIX, ma non fa parte dei standard thread, essendo definito nel POSIX.1b estensione real-time (IEEE 1.3.1b-1993). Di conseguenza le procedure semaforo sono preceduti da “sem_” invece che da “pthread_”.

Il test è stato svolto soltanto nel sistema operativo linux visto che è conforme a POSIX. Questi sono i risultati ottenuti dall'analisi:

- PACKET_BULK_LEN = 512*500 [*200]

Thread\Dim buf. circ.	200	100	50
Riceve dati FPGA	1.3207175732 s	5.1679916382 s	7.8973984718 s
Scrive dati file	10.7986946106 s	10.7161245346 s	10.8839712143 s
Controlla dati file	10.6235141754 s	10.6013793945 s	10.6369600296 s

- PACKET_BULK_LEN = 512*1000 [*200]

Thread\Dim buf. circ.	200	100	50
Riceve dati FPGA	2.6148240566 s	10.8123283386 s	15.9707794189 s
Scrive dati file	22.0000190735 s	21.8669261932 s	21.7010707855 s
Controlla dati file	22.2480773926 s	21.2381362915 s	21.3757057190 s

- PACKET_BULK_LEN = 512*2000 [*200]

Thread\Dim buf. circ.	200	100	50
Riceve dati FPGA	5.2384867668 s	21.1344718933 s	31.8139171600 s
Scrive dati file	42.8528594971 s	42.9610710144 s	42.5680389404 s
Controlla dati file	42.4282417297 s	42.5139884949 s	42.5095176697 s

Ciò che si osserva nei risultati è che il thread che fa i trasferimenti termina la sua esecuzione in tempi rapidi, mentre il thread che deve scrivere questi dati sul disco richiede molto più tempo come il thread che deve controllore i dati. Il problema è dovuto alle differenze di velocità di scrittura/lettura su memoria volatile e memoria permanente. Diminuendo la dimensione del buffer circolare anche thread, che trasferisce i dati, impiega più tempo a fare i trasferimenti dovuto al blocco sul semaforo per la condivisione dei dati. Si è deciso di non adottare questa soluzione visto che globalmente presenta dei tempi di esecuzione maggiore rispetto alla soluzione con un solo processo che non ha almeno il blocco sul semaforo.

5. ASPETTI PRATICI E SVILUPPO

Una volta superati e soddisfatti dai risultati dei test che riguardano la velocità di trasmissione, si è pensato di sviluppare una libreria che pone come obiettivo quello di semplificare l'utilizzo della libreria libusb all'utente. Il suo compito principale è di incapsulare le chiamate alle API del libusb.

Per utilizzare la libreria sviluppata, l'utente deve installare nella sua macchina sia la libreria libusb windows_backend (ossia 1.0) che OpenCV (la libreria libusb non va installata nel modo tradizionale, ma solo scaricata e decompressa). Tra l'altro, l'utente deve assicurarsi che queste due librerie risiedono sotto la cartella C: del disco.

Perché funzioni correttamente la libreria, l'installazione del driver Cypress e winusb ed il caricamento del firmware devono essere a monte dell'utilizzo della stessa. In questo capitolo approfondiremo in modo dettagliato l'installazione dei driver, il caricamento del firmware e le particolarità della libreria.

5.1 Installazione driver Cypress

Quando avviene il primo collegamento tra il prototipo contenente il controller Cypress e l'host attraverso il cavo USB, tipicamente si vede un'icona di colore giallo con un punto interrogativo sotto la voce altri dispositivi nella finestra di gestione dei dispositivi (è possibile aprire questa finestra con un click destro sull'icona computer, scegliere gestione e nella finestra gestione computer scegliere la voce gestione dispositivi, oppure start → pannello di controllo → gestione dispositivi). Ciò vuol dire che il PC non è riuscito ad associare nessun driver al nuovo dispositivo collegato, ovvero nel computer non esiste un driver contenente lo stesso VID e PID che è stato inviato dal dispositivo. Per controllare il VID e PID inviato dal controller bisogna fare click destro sul dispositivo sconosciuto, scegliere proprietà e selezionare scheda dettagli tra le diverse schede.

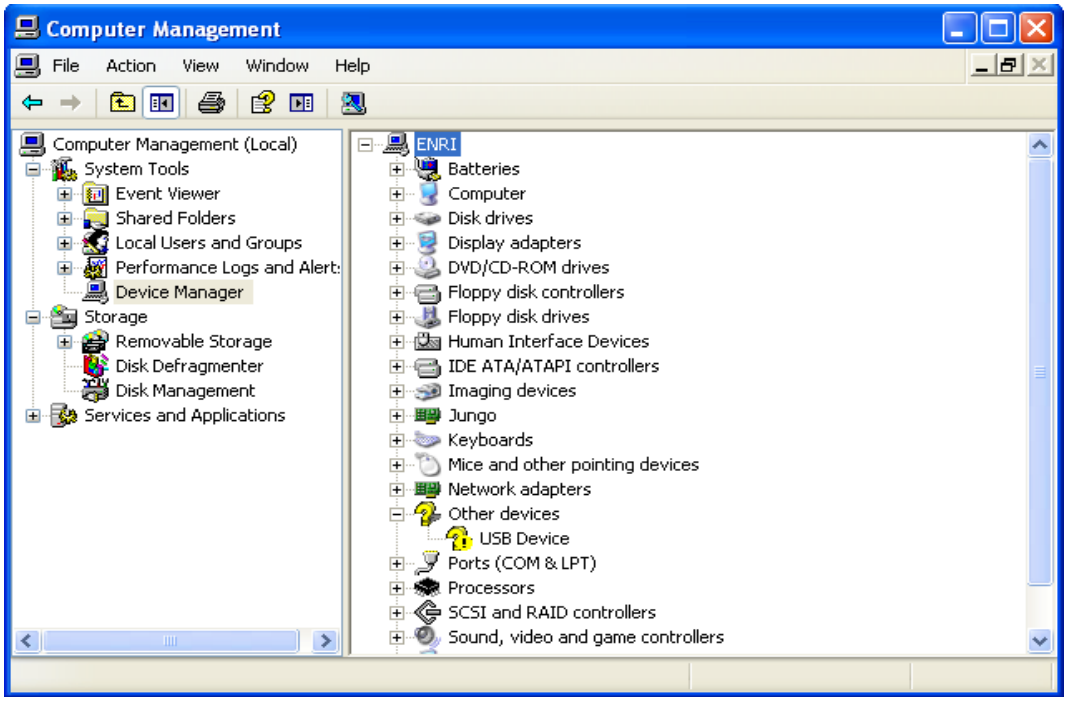


Figura 5.1: la vista della scheda gestione dispositivi

A questo punto bisogna prendere l'iniziativa dell'installazione del driver. Per fare ciò, bisogna aprire la proprietà del dispositivo (come descritto sopra):

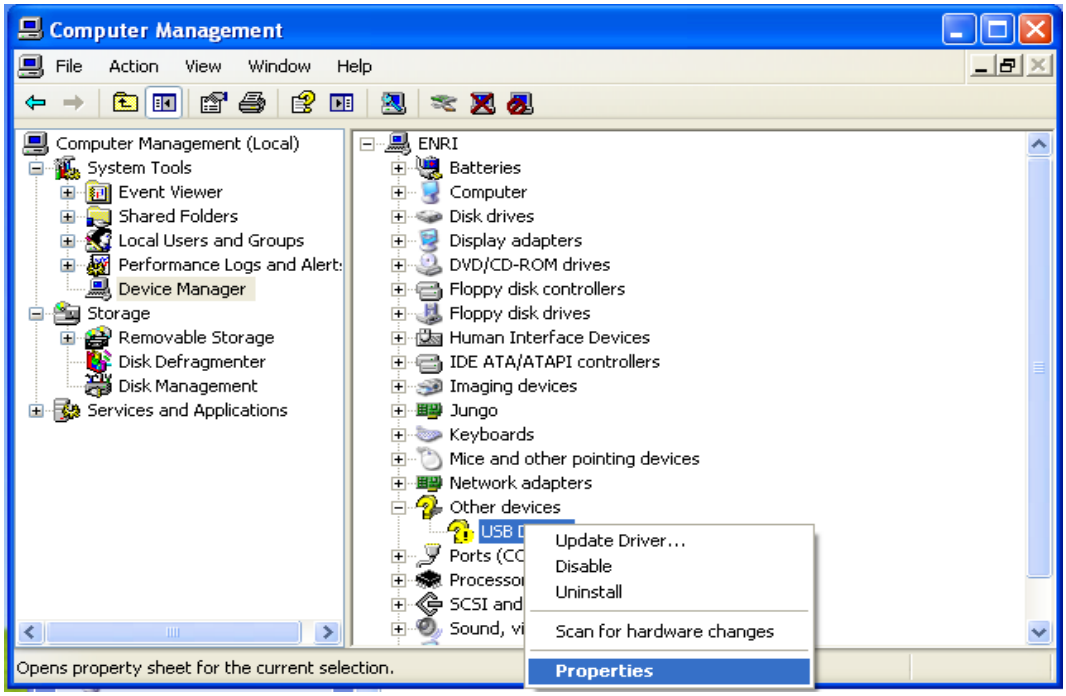


Figura 5.2: la vista della menù a tendina del dispositivo sconosciuto

Nella finestra della proprietà del controller scegliere la scheda driver. In questa scheda cliccare su aggiorna driver. A questo punto parte la procedura di installazione guidata di windows che varia in base alla versione (XP, Vista oppure windows 7).

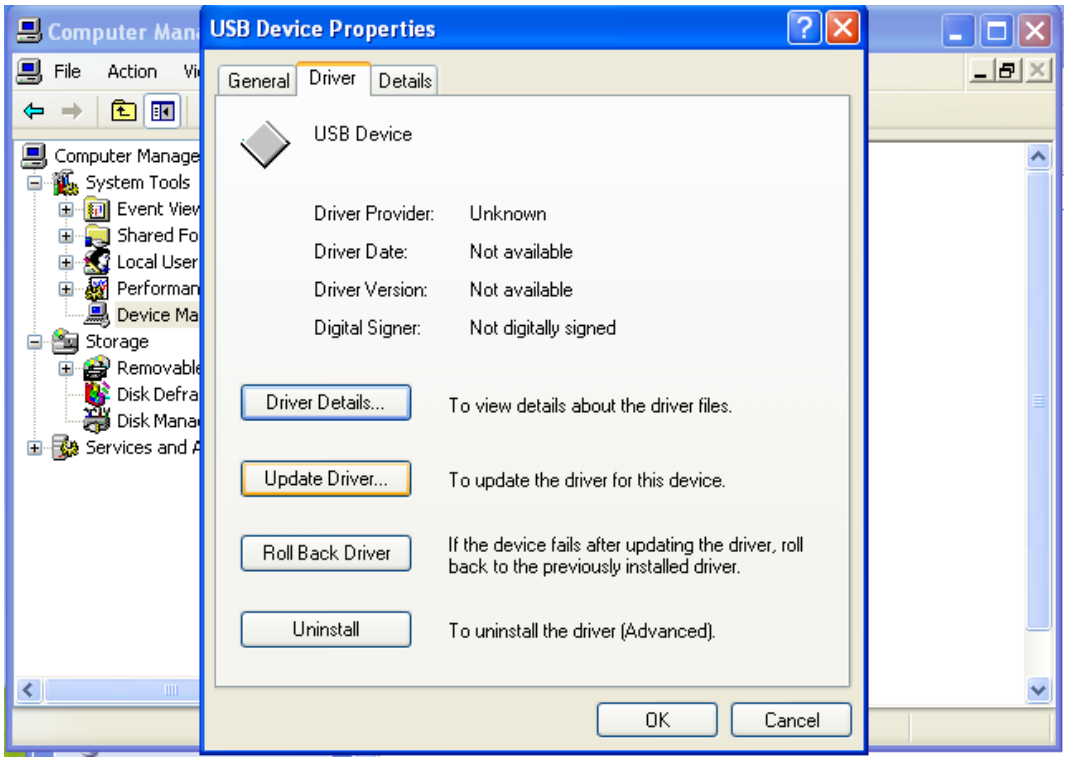


Figura 5.3: la vista della finestra proprietà del dispositivo sconosciuto

In questo caso mostriamo dei screenshot dell'installazione su una macchina XP, ma la procedura non varia di molto nelle altre versioni. Ricordiamo inoltre che il driver Cypress non ha la firma digitale, quindi va disattivata questa opzione (per come disattivare la firma digitale leggere il paragrafo 4.2.3 del capitolo precedente). All'inizio della procedura compare una finestra con due opzioni, il primo è “cerca automaticamente un driver aggiornato” ed il secondo è “cerca il software del driver nel computer”.

In questo caso bisogna scegliere la seconda opzione; ed una volta scelta, nel passo successivo della procedura verrà chiesto di inserire il percorso del driver.

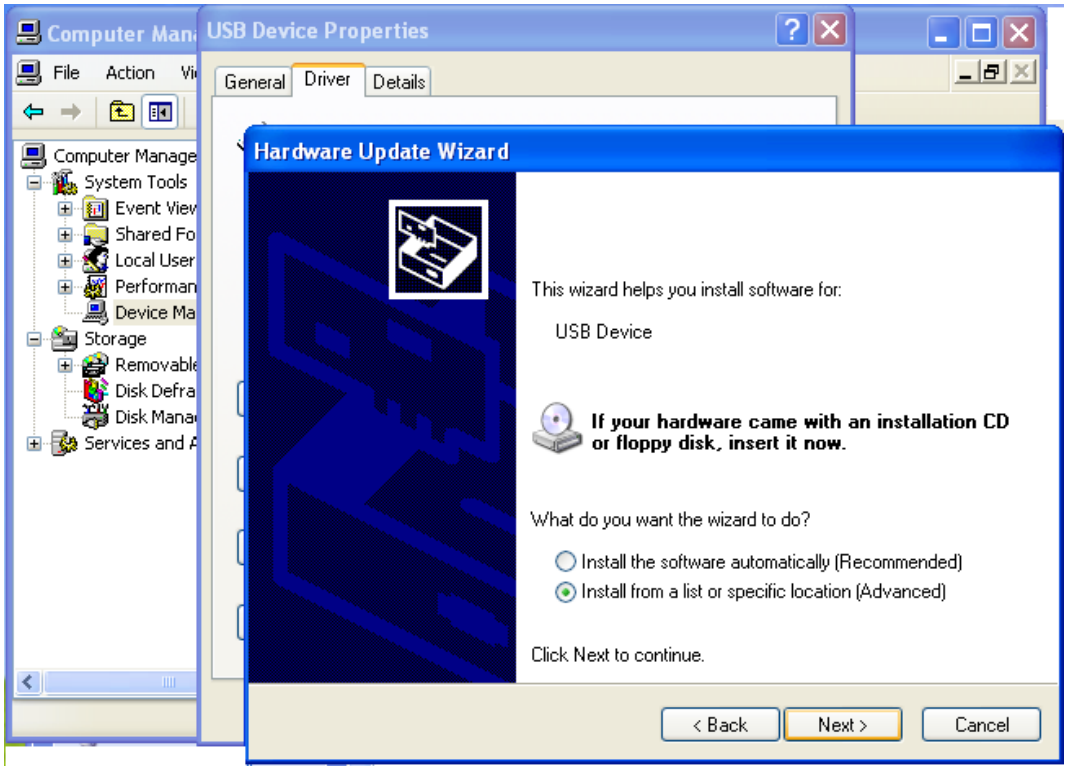


Figura 5.4: la vista dell'avvio della procedura dell'installazione guidata

Per la scelta del driver bisogna posizionarsi nel percorso "...\driver\bin" all'interno della cartella SuiteUSB e scegliere una cartella in base alla versione del windows installato sulla macchina e l'architettura del pc, che può essere a 32 oppure a 64 bit. Nel caso di Windows XP, scegliere la cartella wxp; e nel caso di windows vista, o windows 7, scegliere la cartella wlh. Dopo di che, selezionare la cartella x86 per l'architettura a 32 bit oppure x64 per l'architettura a 64 bit. Arrivato nell'ultimo livello della cartella (qualunque sia stato il percorso) si trova un file con estensione .inf. Bisogna modificare questo file in modo opportuno, ossia cambiare solo due righe: la prima è decommentare (cancellare ;) la riga in accordo all'architettura (32 o 64 bit) e al posto del VID_XXXX e PID_XXXX sostituire i valori dei VID e PID del controller; la seconda modifica riguarda l'aggiornamento del VID e PID che si trovano nella terza ultima riga del file prima della voce CYUSB.GUID. Comunque, per avere maggiori dettagli su come modificare questo file è possibile consultare il paragrafo 4.2.2 del capitolo

precedente.

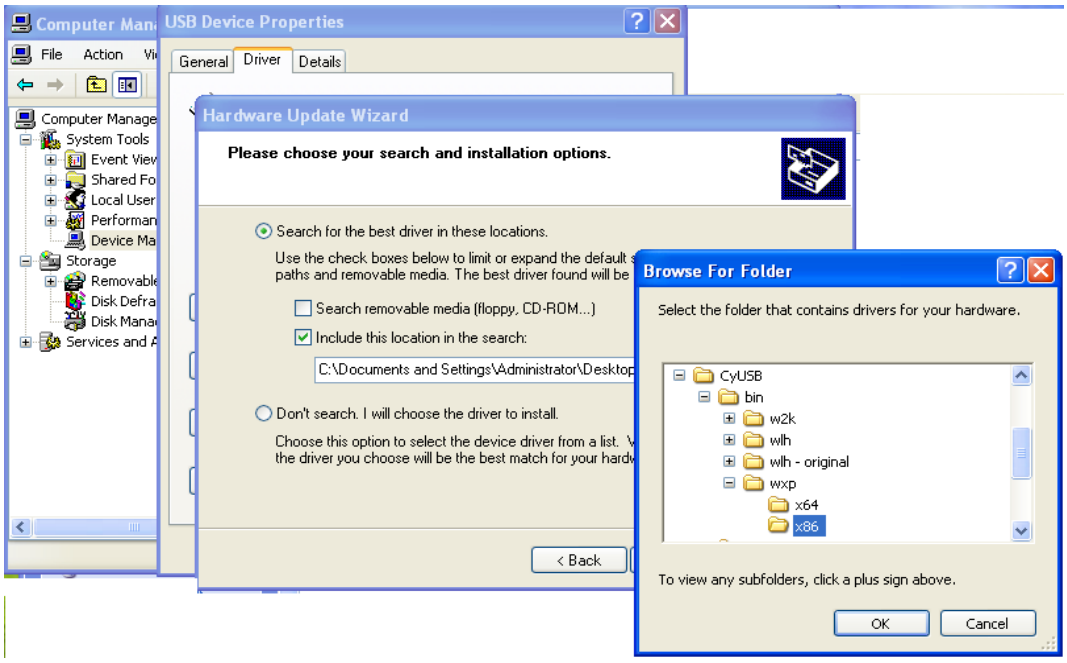


Figura 5.5: la vista della scelta del percorso di driver

Windows avvisa che il driver non ha la firma digitale e chiede se continuare. In questo caso bisogna andare avanti ed ignorare l'avviso.

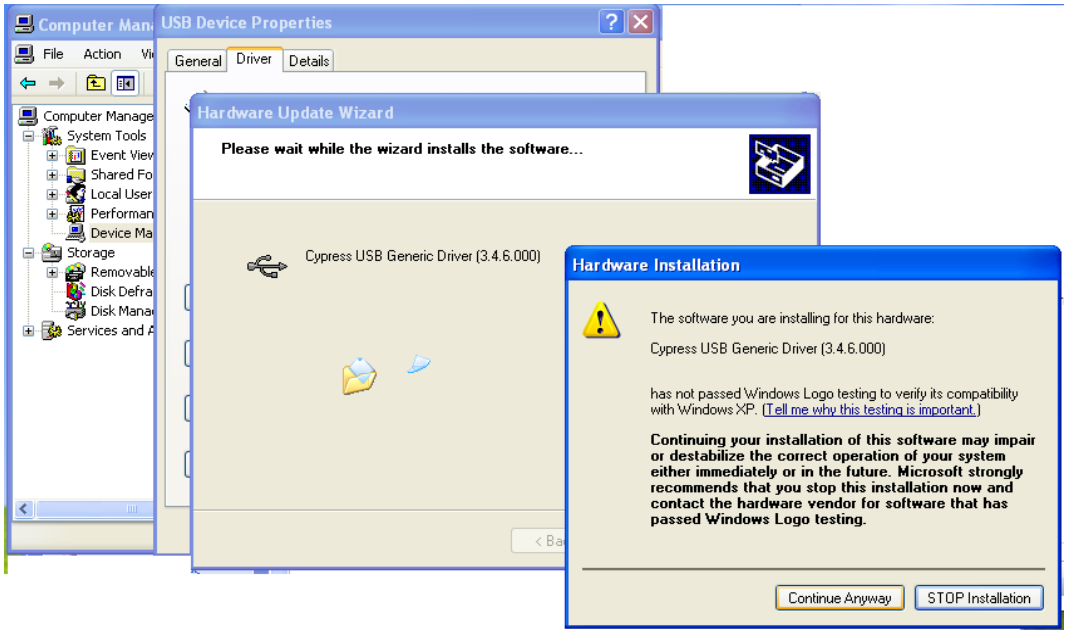


Figura 5.6: la vista della visualizzazione dell'avviso

A questo punto è stato installato il driver; la conferma viene data dalla finestra dei gestioni dei dispositivi.

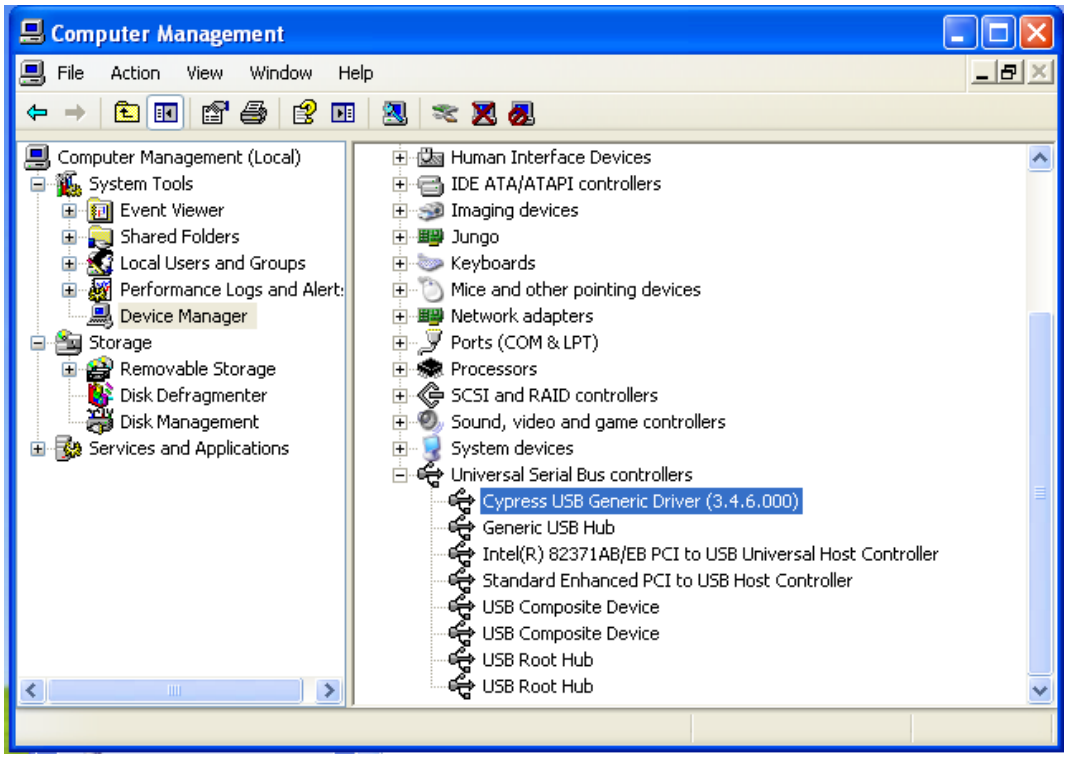


Figura 5.7: la vista della scheda gestione dispositivi

Infatti, scompare il punto interrogativo giallo con la voce altri dispositivi. Adesso il controller è stato identificato e viene riconosciuto come “Cypress USB Generic Driver (3.4.6.000)”.

5.2 Caricamento firmware

Una volta completata l'installazione, bisogna aprire il tool CyControl Center e si vede comparire il nome del controller alla sinistra della finestra. Se, il tool viene aperto prima dell'installazione, nella finestra non compare il nome del dispositivo. Tra l'altro, nella scheda “Descriptor Info” compaiono il VID e PID del controller che è identico a quello visto nella proprietà del dispositivo quando questo era sconosciuto all'host dovuto alla mancanza del driver.

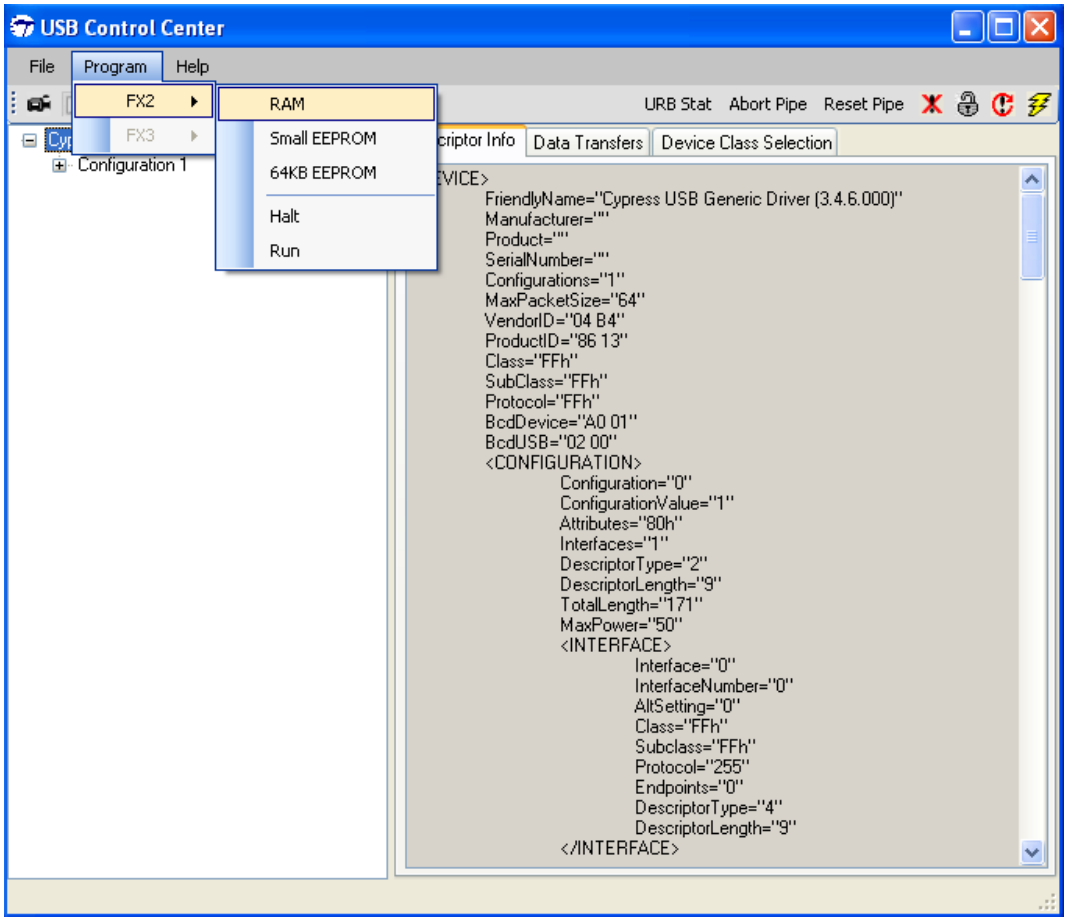


Figura 5.8: la vista del CyControl Center

Adesso si può caricare il firmware, scegliendo dal menù Program → FX2 → RAM ed infine selezionare il file con estensione .hex. Una volta caricato il firmware, il controller scompare dalla lista degli applicativi Cypress (in questo caso CyControl Center). Il motivo è il cambiamento del PID del dispositivo da parte del firmware, cioè il firmware impone al dispositivo un nuovo PID. Per maggiori dettagli su perché si procede così, è possibile prendere visione del paragrafo 4.3 del capitolo precedente.

5.3 Installazione driver winusb

Una volta caricato il firmware, il dispositivo non scompare solo dalla vista degli applicativi Cypress, ma anche il sistema operativo non lo riconosce più. Questo è

dovuto al fatto che l'host non riesce ad associare nessun driver contenente il VID e PID compatibili con il dispositivo (ricordiamo che il firmware ha cambiato il PID del controller).

Si presenta la stessa situazione affrontato nel paragrafo 5.1 di questo capitolo e si procede circa allo stesso modo. Bisogna aprire la finestra di gestione dispositivi se vuole assicurarsi che il controller non sia stato riconosciuto dal PC. Per controllare il nuovo VID e PID inviato dal controller si può procedere allo stesso modo di prima.

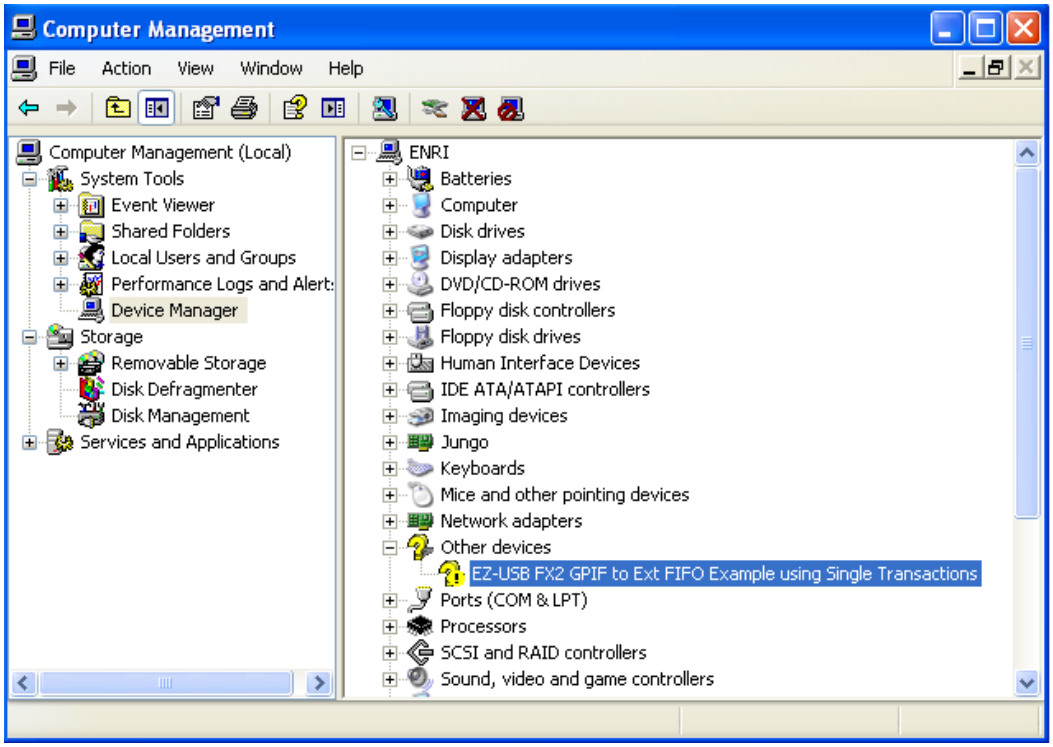


Figura 5.9: la vista della scheda gestione dispositivi

Quindi, bisogna accedere alla scheda driver della proprietà del dispositivo sconosciuto e cliccare su aggiorna driver; all'inizio della procedura di installazione bisogna scegliere l'opzione “cerca il software del driver nel computer”. Quando viene chiesto il percorso del driver, bisogna indicare quello del winusb e non quello di Cypress.

Nella cartella winusb è presente un unico file .inf che è indipendente sia dalla versione del sistema operativo che dall'architettura. In questo file l'unica modifica da apportare è l'aggiornamento del VID e PID presente nel file sotto la sezione "START USER CONFIGURABLE SECTION" con quello del dispositivo. Volendo si può cambiare il nome del dispositivo modificando la stringa DeviceName sempre sotto la stessa sezione.

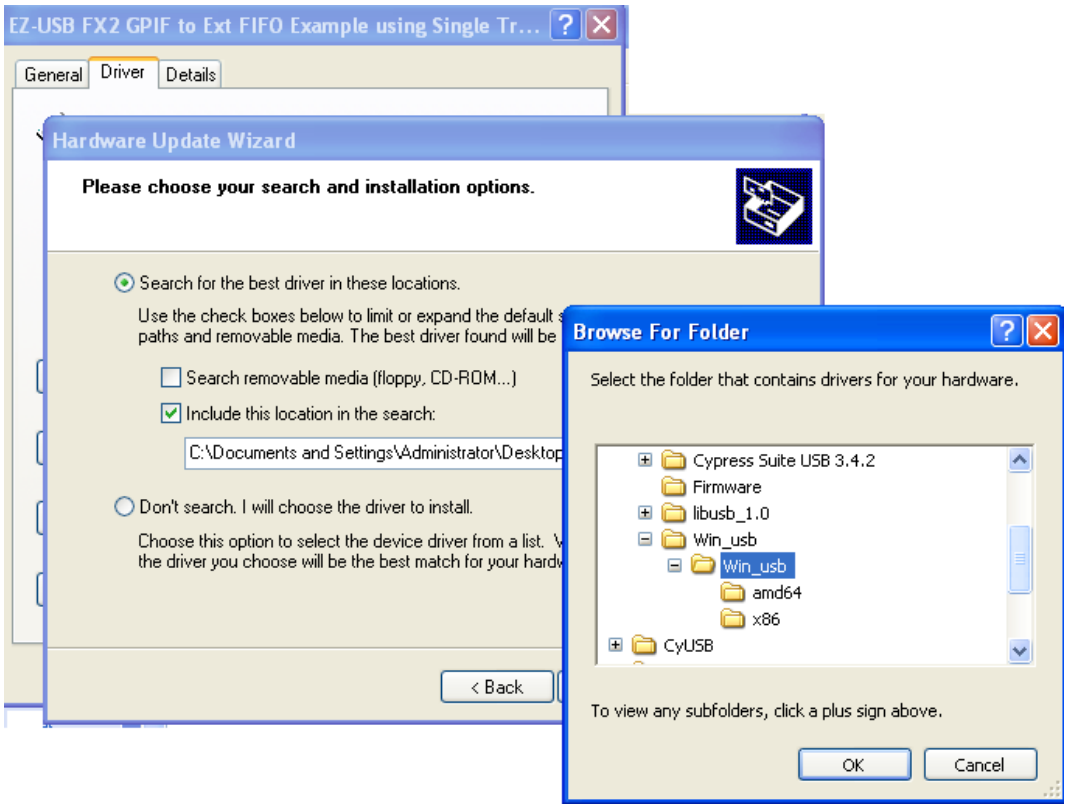


Figura 5.10: la vista della scelta del percorso di driver

Successivamente, viene completato l'installazione ed al controller è associato il driver winusb. Il controller adesso è pronto a scambiare i dati con il PC utilizzando la libreria libusb 1.0 (o windows_banckend) che è incapsulata nella libreria personale. La verifica potrebbe essere fatta controllando sempre la gestione dispositivi dove il controller non compare come un dispositivo sconosciuto.

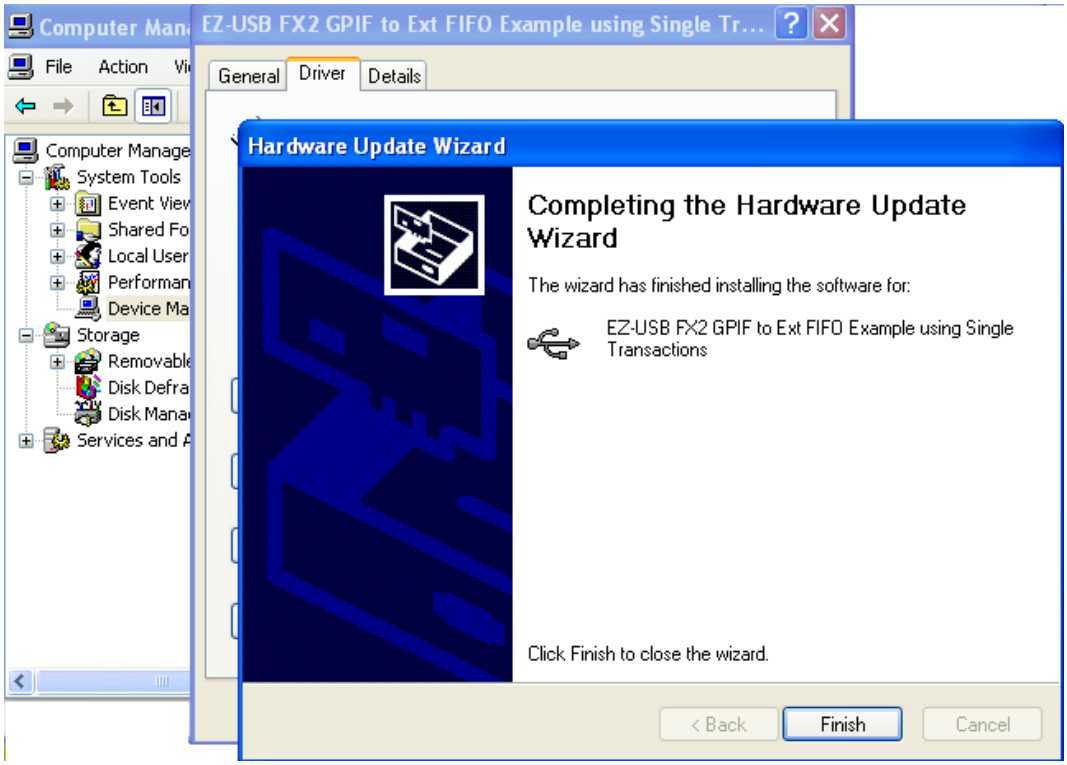


Figura 5.11: la vista del completamento dell'installazione del driver

Questo procedimento (installazione dei due driver) deve essere svolto solo la prima volta. In seguito, quando il controller viene collegato all'host, il sistema operativo all'inizio gli associa il driver Cypress dovuto alla compatibilità con il PID, invece, dopo il caricamento del firmware, windows gli associa automaticamente il driver winusb a causa del cambiamento avvenuto nel PID del controller.

5.4 Sviluppo libreria

La libreria ha come obiettivo principale quello di semplificare l'utilizzo della libreria libusb all'utente sviluppatore. Infatti, il suo compito è quello di incapsulare le chiamate alle API del libusb, così che l'utente non sa delle chiamate di basso livello delle API libusb. Le funzioni presenti nella libreria sono:

- void stereo_camera_usb_about(void);

- int stereo_camera_usb_init(void);
- int stereo_camera_usb_connect(void);
- int stereo_camera_usb_get_properties(unsigned char *buffer, int PACKET_BULK_LEN, StereoCameraProperty *Info);
- int stereo_camera_usb_set_properties(StereoCameraProperty *Info);
- int stereo_camera_usb_bulk_transfer(unsigned char *buffer, int PACKET_BULK_LEN, int timeout);
- int stereo_camera_usb_write(unsigned char *buffer, int PACKET_BULK_LEN, int timeout);
- int stereo_camera_usb_get_frame(unsigned char *buffer, int PACKET_BULK_LEN, IplImage *imgLM, IplImage *imgRS);
- int stereo_camera_usb_get_frame_and_save(unsigned char *buffer, int PACKET_BULK_LEN);
- int stereo_camera_usb_disconnect(void);

Tutte le funzioni hanno il prefisso stereo_camera_usb, che è il nome della libreria. Sono stati fatti due release della libreria: uno per windows e l'altro per il sistema operativo linux (le denominazioni ed i ruoli della funzioni contenute all'interno della libreria rimangono uguali per entrambe le versioni, quindi la portabilità del codice è garantita).

E' stato rilasciato anche una versione debug, che ha come obiettivo quello di aiutare gli sviluppatori. Infatti, nella versione debug, le funzioni e la mansione svolta da essi rimangono uguali alla versione standard, ma vengono fatte delle stampe all'interno delle funzioni in modo da far comprendere allo sviluppatore ciò che accade all'interno della funzione. Adesso vediamo in modo più dettagliato le funzioni.

5.4.1 Funzioni: about, init, connect e disconnect

La funzione about() non ha nessun parametro d'ingresso e nemmeno d'uscita. Questa funzione non fa nulla di particolare; il suo obiettivo è quello di stampare delle informazioni utili all'utente, come il nome e la versione della libreria.

La funzione init() non ha nessun parametro d'ingresso e ritorna un intero come valore d'uscita. Il valore d'uscita è positivo se la funzione è riuscito ad eseguire tutti i suoi passi mentre è negativo se per caso non è riuscito eseguire almeno un passo per bene. Il suo obiettivo è inizializzare la libreria libusb ed ottenere handle del dispositivo.

La funzione inizializza la libusb attraverso la chiamata alla funzione libusb_init(libusb_context **context), dove context è un output opzionale che punta al contesto che in questo caso è null. La funzione ritorna zero se è andato a buon fine la richiesta, mentre libusb_error in caso contrario. Questa deve essere la funzione ad essere chiamata per prima di ogni altra funzione di libusb dal codice, se si vuole utilizzare la libreria libusb.

Dopo aver inizializzato libusb, la funzione init ottiene un handle del dispositivo che ci permette di comunicare con esso. L'handle viene ottenuto attraverso la chiamata alla funzione libusb_open_device_with_vid_pid(libusb_context *ctx, uint16_t vendor_id, uint16_t product_id). Se l'handle ritornato è null allora vuol dire che non è andata a buon fine la richiesta. Nei parametri passati, il context è null; mentre si dà il VID e il PID del controller all'ingresso alla funzione come vendor_id e product_id. Si può utilizzare questa funzione per ottenere handle del dispositivo solo se si conosce il VID e PID.

In alternativa si può utilizzare la funzione libusb_open(). Per utilizzare quest'ultima funzione bisogna inizialmente prendere la lista dei dispositivi usb collegati all'host attraverso la funzione libusb_get_device_list(). Lo svantaggio di questa funzione è che una volta ottenuta la lista, bisogna cercare il dispositivo di nostro interesse e rilasciare gli altri dispositivi.

La funzione connect() non ha nessun parametro d'ingresso e ritorna un intero come valore d'uscita. Il valore è positivo se la funzione è riuscito ad eseguire tutti i suoi passi mentre è un valore negativo se per caso non è riuscito eseguire almeno un passo per bene. Essa configura e prepara il dispositivo per lo scambio dei dati.

All'inizio chiama la funzione libusb_set_configuration(libusb_device_handle *dev, int configuration) che imposta una configurazione attiva per il dispositivo. Come handle viene passato quello del controller ottenuto precedentemente e come valore di configuration uno, se viene passato un valore negativo il dispositivo non viene configurato. La funzione ritorna zero se tutto è andato a buon fine, mentre libusb_error in caso contrario. Il sistema operativo può aver oppure no impostato una configurazione attiva sul dispositivo. Visto che per le chiamate successive, ossia per la comunicazione con il dispositivo, serve prima impostare una configurazione attiva, per questo la funzione connect chiama questa funzione al primo passo.

Successivamente la funzione connect chiama la funzione libusb_claim_interface (libusb_device_handle *dev, int interface_number) che richiede un'interfaccia sul handle del dispositivo. L'handle passato è quello del controller e come valore di interface_number viene passato lo zero che è l'interfaccia richiesta. Anch'essa ritorna zero nel caso di successo e libusb_error nel caso opposto. Bisogna obbligatoriamente richiedere un'interfaccia prima di iniziare la comunicazione con un endpoint.

Infine, la funzione chiama libusb_set_interface_alt_setting (libusb_device_handle *dev, int interface_number, int alternate_setting) che attiva un ambiente alternativo per l'interfaccia. L'handle è quello del controller mentre vengono passati zero e zero come valori di interface number e alternate setting. Anch'essa ritorna zero nel caso di successo e libusb_error in caso contrario. Di seguito, il controller è pronto allo scambio dei dati con il pc.

La funzione disconnect non ha nessun parametro d'ingresso e ritorna un intero come valore d'uscita. Il valore è positivo se la funzione è riuscita a eseguire tutti i suoi passi mentre è un valore negativo se per caso non è riuscito eseguire almeno un passo per bene. Essa disconnette il dispositivo.

All'inizio rilascia l'interfaccia con la chiamata alla funzione libusb_release_interface(libusb_device_handle *dev, int interface_number), richiesta precedentemente. La funzione ritorna zero nel caso di successo e libusb_error in caso contrario. Successivamente rilascia l'handle del dispositivo, ottenuto precedentemente, con la chiamata alla funzione libusb_close(libusb_device_handle *dev). Ed infine, chiude il libusb con la chiamata alla funzione libusb_exit(libusb_context **context) che è il duale della funzione libusb_open(libusb_context **context) ed anche in questo caso il contesto è null.

5.4.2 Funzioni: bulk transfer, write, get e set properties

La funzione bulk_transfer(unsigned char *buffer, int PACKET_BULK_LEN, int timeout) esegue un trasferimento di massa dal controller all'host. La funzione ha come parametro d'ingresso un buffer di dati dove copiare i dati ricevuti dal controller, il PACKET_BULK_LEN che è la lunghezza del buffer ed il timeout, ossia il tempo dopo il quale la funzione rinuncia a causa della mancata risposta da parte del controller. La funzione ha come valore d'uscita un intero che è negativo nel caso non riesca ad eseguire la richiesta, ed un valore positivo che rappresenta un numero di byte effettivamente trasferiti.

All'inizio questa funzione cancella la condizione di stallo per l'endpoint attraverso la chiamata alla funzione libusb_clear_halt(libusb_device_handle *dev, unsigned char endpoint) perché gli endpoint, con lo stato di stallo, non sono in grado di ricevere o trasmettere dati fino a quando la condizione non viene tolta.

L'handle passato è quello del controller, mentre l'endpoint è ENDPOINT_BULK_IN. La funzione ritorna zero nel caso di successo e libusb_error in caso contrario.

Dopo di che, effettua il bulk transfer attraverso la chiamata alla funzione libusb_bulk_transfer(struct libusb_device_handle *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *transferred, unsigned int timeout). L'handle passato è quello del controller, endpoint è ENDPOINT_BULK_IN, data in questo caso è il buffer che l'utente ha dato in ingresso alla funzione, length è la lunghezza del buffer sempre data dall'utente denominato PACKET_BULK_LEN, transferred è parametro d'uscita che indica quanti byte effettivamente sono stati letti dall'endpoint ed infine il timeout rappresenta il tempo, dato in ingresso dall'utente, dopo il quale la funzione rinuncia, a causa della mancata risposta dal controller. La funzione ritorna zero nel caso di successo e libusb_error in caso opposto.

La funzione write() esegue un trasferimento di massa da lato host al controller, ma è lasciato per uno sviluppo futuro dato che non si è riuscito ancora a mettere in esecuzione i trasferimenti dall'host verso la periferica.

La funzione get_properties(unsigned char *buffer, int PACKET_BULK_LEN, StereoCameraProperty *Info) ha come parametro d'ingresso un buffer di dati, dove copiare i dati ricevuti dal controller, il PACKET_BULK_LEN, che è la lunghezza del buffer, e, come parametro d'uscita, StereoCameraProperty, che contiene le informazioni relative alle dimensioni delle immagini ricevute dalle telecamere. La funzione ritorna un valore positivo nel caso di successo ed un valore negativo in caso contrario.

All'inizio la funzione esegue un bulk transfer di una certa dimensione in modo da essere sicura che ci sia almeno un'immagine intero nei dati trasferiti. Di preciso si consiglia di fare un trasferimento di circa 2,9MB di dati (PACKET_BULK_LEN = 512*200*30). Un'immagine è 300KB

((640*480)/1024) e gli vengono aggiunti 225KB ((480*480)/1024) di dati dummy, ovvero fittizzi, per mantenere il ritmo nella catena da cui passano i dati. I dati fittizzi aggiunti sono 480 byte per ogni riga. Quindi un'immagine, complessivamente, occupa 525KB.

Sono due le telecamere che emanano dati insieme. Infatti, i dati ricevuti vengono interpretati come sequenze di word, dove nella parte bassa (byte 0-7) si trova il byte contenente il valore emanato dalla telecamera slave, mentre nella parte alta (byte 8-15) si trova il valore emanato dalla telecamera master. Quindi bisognerebbe fare almeno un trasferimento di circa 1MB (525 KB * 2) per trovare un'immagine intera da una telecamera.

Non è garantito che, nei dati trasferiti, i primi due byte rappresentino l'inizio di due immagini perché i trasferimenti non sono multipli delle dimensioni delle immagini. Quindi in un trasferimento si potrebbe trovare, nel caso peggiore, l'immagine intera a meno di una (prima) riga, che è stata trasferita nel precedente trasferimento scartato.

Per lo sviluppo della libreria abbiamo supposto di fare dei trasferimenti, elaborare i dati e cancellarli. Questa scelta è dovuta al fatto che sono molti i dati che vengono trasferiti, quindi la memoria volatile dopo un po' potrebbe non bastare, o saturare, e salvare i dati nella memoria fissa richiede troppo tempo e rischiamo di non mantenere il ritmo della catena.

Di fatto in un trasferimento le immagini validi, ovvero tenuto in considerazione, sono solo le prime due intere (una per ogni telecamera) ed il resto viene ignorato (l'effetto che si ha nel tralasciare il trattamento di questi dati non influisce significativamente sul resto). Per cui per essere certi di trovare almeno un'immagine (in verità sono due, visto che sono due le telecamere emittenti) bisogna fare un trasferimento circa di 2MB. Per sicurezza si consiglia di aumentare questa dimensione e portarla circa a 3MB.

La dimensione viene calcolata nel seguente modo: le telecamere aggiungono dei byte di controllo, oltre ai valori del pixel delle immagini. Ogni telecamera aggiunge un byte contenente il valore zero all'inizio dell'immagine e tre alla fine, ed aggiunge un uno ad ogni inizio della riga dell'immagine e due ad ogni fine riga.

Basta scorrere i dati ricevuti dal controller e prendere in considerazione un byte alla volta (byte bassi per la telecamera slave e byte alti per la telecamera master). Quando si trova uno zero vuol dire che sta iniziando un'immagine quindi bisogna inizializzare le variabili locali. Così ad ogni uno incontrato incremento il contatore della riga (lunghezza verticale) dell'immagine fino a che non incontro un tre nell'analizzare i dati. E per trovare la lunghezza orizzontale dell'immagine ho incrementato il contatore delle colonne tra un uno ed un due (solo nella prima riga).

Questa funzione analizza i dati soltanto su una delle due canali, visto che entrambe le telecamere emettono immagini delle stesse dimensioni, per migliorare l'efficienza. Il calcolo delle dimensioni delle immagini è molto sensibile ai byte di controllo aggiunti dalle telecamere. Infatti se qualche byte viene corrotto dalla catena ha un effetto molto grave sulla precisione del calcolo.

La funzione set_properties() è lasciato per uno sviluppo futuro, dove si dà la possibilità all'utente di settare le dimensioni delle immagini, visto che non siamo riusciti ancora a mettere in esecuzione i trasferimenti dall'host verso la periferica.

5.4.3 Funzioni: get_frame e get_frame_and_save

La funzione get_frame(unsigned char *buffer, int PACKET_BULK_LEN, IplImage *imgLM, IplImage *imgRS) ha come parametro d'ingresso un buffer di dati, dove risiedono i dati ricevuti dal controller, il PACKET_BELK_LEN, che è la lunghezza del buffer, e, come parametro di ingresso/uscita, due immagini, che devono essere allocati ed inizializzati dall'utente, la funzione caricherà queste immagini con i primi due immagini completi trovati nei dati. La funzione ritorna un valore positivo nel caso di successo e un valore negativo in caso opposto.

Per riuscire a catturare le due immagini, questa funzione utilizza i byte d'informazione aggiunte dalle telecamere come la funzione get_properties, quindi anch'essa è molto sensibile ai byte d'informazione.

La procedura scorre i dati ricevuti in ingresso e prende in considerazione un byte alla volta (byte bassi per la telecamera slave e byte alti per la telecamera master). Quando si trova uno zero vuol dire che sta iniziando un'immagine quindi inizializza le variabili locali. Così ad ogni uno incontrato aggiorno il contatore della riga (lunghezza verticale) dell'immagine fino a che non incontro un tre nell'analizzare i dati. E tra un uno ed un due incontrato aggiorno il contatore della colonna (lunghezza orizzontale) che inizializzo ad ogni inizio riga. Per i byte diversi dai byte di controllo, a meno che il byte non sia dummy, scrivo il valore del pixel posizionando al pixel (x,y) con i contatori della riga e della colonna.

La funzione stereo_camera_usb_get_frame_and_save(unsigned char *buffer, int PACKET_BULK_LEN) ha come parametro d'ingresso un buffer di dati, dove risiedono i dati ricevuti dal controller, il PACKET_BELK_LEN, che è la lunghezza del buffer. La funzione ritorna un valore positivo nel caso di successo e un valore negativo in caso contrario. La funzione salva i dati ricevuti dal controller in una memoria fissa.

Questa funzione serve all'utente per fare debugging. Infatti, se l'utente non è soddisfatto del comportamento di altre funzioni, o non trova riscontro dall'analisi dei dati fatti automaticamente dalle altre funzioni, attraverso questa funzione può salvare i dati ed analizzarli.

Questa funzione salva i dati in tre file diversi: nel primo file salva tutti i dati, mentre nel secondo e nel terzo file salva i dati in base all'appartenenza del canale (master o slave). Quindi il primo file è la somma degli altri due. I dati vengono salvati in questo modo per dare un'ampia scelta di analisi all'utente.

5.4.4 Sviluppo applicazione utente

Per testare la libreria è stata sviluppata una piccola applicazione utente che ha il compito di invocare in modo ordinato le funzioni della libreria. Il codice esegue le operazioni (acquisizione e visualizzazione delle immagini) dentro un loop infinito, così da sembrare un video.

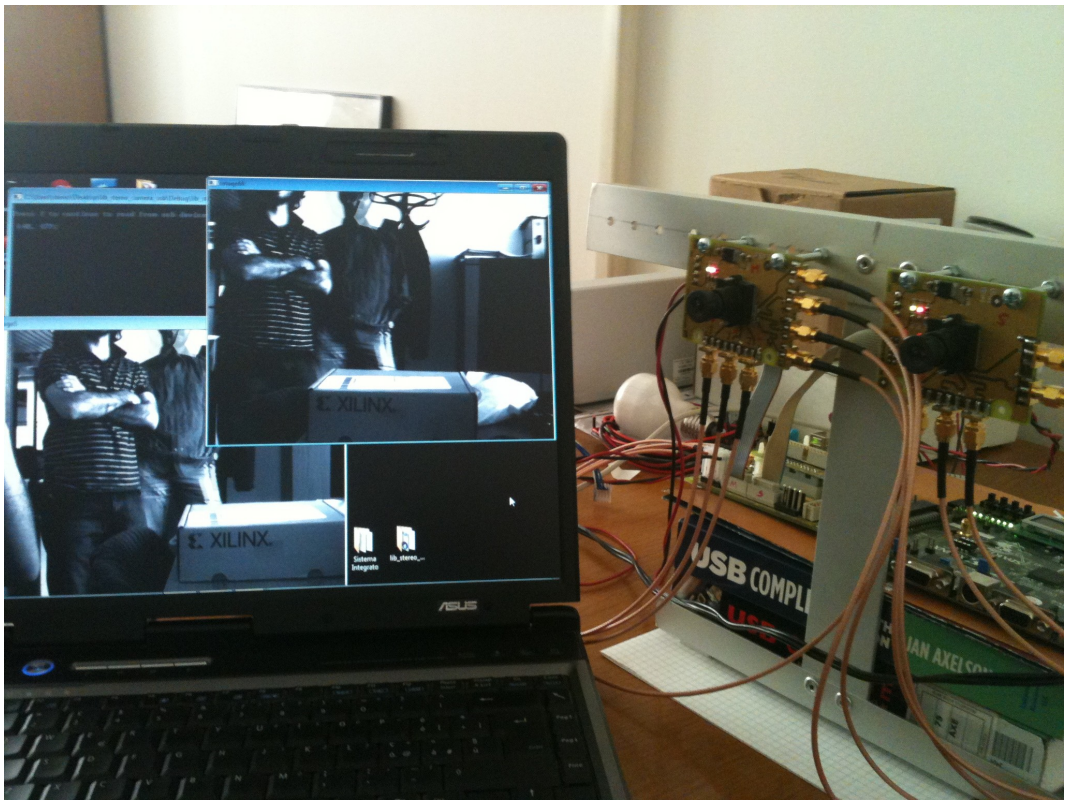


Figura 5.12: acquisizione delle immagini dalle telecamere

Si sta integrando il codice sviluppato con un'altra libreria che faccia anche la calibrazione software delle immagini.

L'utente può includere ed utilizzare la libreria sviluppata nel suo progetto software (per esempio in Microsoft Visual Studio) nel modo tradizionale, ossia come vengono inclusi ed utilizzati le librerie in generale.

CONCLUSIONI E SVILUPPI FUTURI

L’interfacciamento tra controller Cypress CY7C6013A ed il PC ha imposto un attento studio per la progettazione di un applicativo che gestisce correttamente i trasferimenti.

In particolare, apprendere l'uso e l'implementazione di una libreria è un processo a volte difficile, soprattutto per il sottoscritto che, prima di questa esperienza, praticamente non ha avuto mai a che fare con questo apprendimento.

I primi test sono stati svolti usando la libreria libusb, che fornisce un driver semplice ed efficace anche se con poche possibilità di personalizzazione, soprattutto nella versione 0.1. Decisamente più soddisfacente, d’altro canto, la versione 1.0 della libreria, che offre un’interfaccia molto più semplificata e di alto livello per l’accesso a dispositivi USB. Anche se non è stata testata l'ultima release della versione 1.0 di libusb, un possibile proseguo del progetto potrebbe prevederne l’implementazione per verificare se siano possibili ulteriori miglioramenti. Comunque, il funzionamento della versione 1.0 è decisamente più performante rispetto alla versione 0.1.

Nel mio caso di studio, oltre ai test, si chiedeva l'implementazione di una libreria che incapsulasse le chiamate alle funzioni della libreria libusb per la connessione, trasferimento ed elaborazione dati e la disconnessione con la periferica. Nonostante le difficoltà dovute al molto materiale a disposizione e all'iniziale disorientamento si è riuscito a comprendere le tecniche di analisi e di impiego dello strumento e utilizzarlo in modo efficace.

L'utilizzo della libreria in luogo di uno sviluppo tradizionale ha permesso di gestire in modo semplice ed efficace e di organizzare in modo comprensibile le attività e le responsabilità dei diversi componenti software coinvolti nel flusso di lavoro. Questo approccio è risultato molto più maneggevole, flessibile, chiaro, testabile, riutilizzabile ed elegante.

Durante l'implementazione della libreria sono emerse diverse difficoltà che hanno contribuito a portare dei rallentamenti nel progetto complessivo ma, nonostante ciò, la telecamera stereo funziona con una banda lorda di 36MB/s a cui corrispondono circa 30 frame/s (netti) con un singolo sensore, cioè 15 fps in modalità stereo.

Sono ancora attualmente in corso degli algoritmi implementabili su FPGA per eseguire una compressione no loss delle immagini catturate dai sensori e della mappa di disparità, che attualmente viene creata dall'FPGA stessa.

Attraverso la realizzazione di questo progetto ho potuto verificare e testare i vantaggi che offrono le librerie rispetto a soluzioni tradizionali per lo sviluppo di applicazioni, oltre alla conoscenza approfondita del protocollo USB.

Sono tante le modifiche, sia hardware che software, che si potrebbero portare al progetto; molte delle quali sono già in corso ed altre sono presenti nella pianificazione del progetto.

La prima modifica in corso è il trasferimento del progetto da Spartan3 a Spartan6. Durante il trasferimento si sono accorti che bisogna portare delle modifiche hardware, non semplici, visto che le caratteristiche delle due Spartan sono simili ma non uguali.

Ad oggi, si sta perfezionando lo sviluppo di un codice software per la calibrazione delle immagini a lato host, ma l'obiettivo è quello di riuscire a far svolgere questo compito direttamente sull'FPGA. Quindi, questo rimane ancora un traguardo da raggiungere.

Si potrebbe iniziare ad utilizzare l'ultima versione del libusb in modo da implementare l'interfaccia USB 3.0, anche quest'ultima modifica è in fase di test. Il vantaggio di rimanere agganciato all'ultima tecnologia è che normalmente le tecnologie mantengono la compatibilità verso il basso. Quindi, è possibile utilizzare sia l'hardware che il codice sviluppato per l'interfaccia USB 3.0 con la versione 2.0, mentre viceversa non è sempre possibile.

Un'altra opzione potrebbe essere quello di andare alla ricerca dell'esistenza delle altre libreria che abbiano delle prestazioni migliori rispetto alla libreria libusb.

Non è ancora stato implementato il flusso di dati bidirezionali; infatti ancora oggi si fanno dei trasferimenti soltanto dal controller ad host e non viceversa. Quindi, questo è un traguardo da raggiungere che è presente nel piano del progetto.

Tra l'altro, senza il passo precedente non si riuscirà nemmeno a completare l'implementazione della libreria, che potrebbe essere aggiornato utilizzando l'ultimo release di OpenCV.

Oltre all'interfaccia USB, si sta cercando di utilizzare anche l'interfaccia ethernet che ha due vantaggi rispetto all'USB. Il primo è che è possibile aumentare la distanza tra le telecamere ed Host, ed il secondo è che i trasferimenti sono molto più veloci rispetto ai trasferimenti attuali.

In futuro, si potrebbe pensare di creare una rete composta da più pc che ricevono i dati e fanno comunicare questi pc in modo da suddividere il carico globale.

BIBLIOGRAFIA

[1.1] Dipartimento di Fisica dell'Università degli studi di Palermo, “Interfaccia USB”, http://www.fisica.unipa.it/~agliolo/didattica_file/fisica /lab_fis_mat_1/Interfaccia%20USB.pdf, ottobre 2006.

[1.2] Enciclopedia libera, “Trasmissione (telecomunicazioni)”, http://it.wikipedia.org/wiki/Trasmissione_(telecomunicazioni),agosto 2011

[1.3] Enciclopedia libera, “Trasmissione analogica”, http://it.wikipedia.org/wiki/Trasmissione_analogica, marzo 2011.

[1.4] Enciclopedia libera, “Trasmissione digitale”, http://it.wikipedia.org/wiki/Trasmissione_digitale, luglio 2011.

[1.5] Enciclopedia libera, “Trasmissione parallela”, http://it.wikipedia.org/wiki/Trasmissione_parallela, settembre 2011.

[1.6] Enciclopedia libera, “Trasmissione seriale”, http://it.wikipedia.org/wiki/Trasmissione_seriale, settembre 2011.

[1.7] “Tecniche di comunicazione digitale”, http://www.pomante.net/sito_gg/SistemiEmbedded0809/ComDig.pdf

[1.9] Enciclopedia libera, “Universal Serial Bus”, http://it.wikipedia.org/wiki/Universal_Serial_Bus, settembre 2011.

[1.10] Tutorial USB, “USB: Universal Serial Bus”, http://www.bitportal.it/tutorial/usb_tutorial_1.html, obiettivi USB.

[1.11] Enciclopedia libera, “USB hub”, http://en.wikipedia.org/wiki/USB_hub, settembre 2011.

[1.11] TerraTec, “Guida alle soluzioni” http://www.terratec.it/tutorials/pci-usb-firewire.pdf, PCI-USB-FIREWIRE.

[1.12] Tecnoligichevolmente, “USB 3.0: ci sono vantaggi concreti rispetto ad USB 2.0?”, http://tecnologichevolmente.com/usb-3-0-ci-sono-vantaggi-concreti-rispetto-ad-usb-2-0/, agosto 2011.

[1.13] Enciclopedia libera, “USB OTG”, http://it.wikipedia.org/wiki/USB_OTG, giugno 2011.

[1.14] Enciclopedia libera, “IEEE 1394”, http://it.wikipedia.org/wiki/IEEE_1394, agosto 2011.

[1.15] Enciclopedia libera, “Ethernet”, http://en.wikipedia.org/wiki/Ethernet, settembre 2011.

[1.16] Tom's hardware, “USB 3.0, tutti i vantaggi del nuovo standard” http://www.tomshw.it/news.php?newsid=19500, settembre 2009.

[1.17] Enciclopedia libera, “Thunderbolt (interfaccia)”, http://it.wikipedia.org/wiki/Thunderbolt_(interfaccia), settembre 2011.

[1.18] “Protocollo USB”, http://whiteangel89.altervista.org/protocollo_USB.pdf

[2.1] “Cypress Fx2 Technical Reference Manual“, http://www.keil.com/dd/docs/datashts/cypress/fx2_trm.pdf

[2.2] Enciclopedia libera, “Phase-locked loop”,

http://it.wikipedia.org/wiki/Phase-locked_loop, settembre 2011.
[2.3] "Image sensor 1/3-inch Wide VGA CMOS image sensor data sheet", http://www.nevael.ru/d/33943/d/mt9v032_book.pdf
[2.4] "DS92LV16 16-bit bus LVDS Serializer/Deserializer – 25 – 80 Mhz" http://www.datasheetcatalog.org/datasheet/nationalsemiconductor/DS92LV16.pdf
[2.5] "Microchip PIC18F2585/2680/4585/4680 Data Sheet", http://ww1.microchip.com/downloads/en/DeviceDoc/DS-39963a.pdf
[2.6] Enciclopedia libera, "Field Programmable Gate Array", http://it.wikipedia.org/wiki/Field_Programmable_Gate_Array, settembre 2011.
[2.7] "Spartan-3 FPGA Family Data Sheet", http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[3.1] "EZ-USB FX2LP Bulk Transfer Application in C# Usnig SuiteUSB C# Library", http://www.cypress.com/?docID=30291
[3.2] Enciclopedia libera, "VHDL", http://en.wikipedia.org/wiki/VHDL, settembre 2011.
[3.3] Enciclopedia libera, "Xilinx ISE", http://en.wikipedia.org/wiki/Xilinx_ISE, settembre 2011.
[3.4] "Xilinx ISE 10 Tutorial", http://www.xess.com/appnotes/ise-10.pdf
[3.5] "ISE In-Depth Tutorial", http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_2/ise_tutorial_ug695.pdf
[3.6] Enciclopedia libera, "PicoBlaze", http://en.wikipedia.org/wiki/PicoBlaze, aprile 2011.
[3.7] "PicoBlaze 8-bit Embedded Microcontroller User Guide", http://etidweb.tamu.edu/classes/entc249/ug129.pdf
[3.8] Enciclopedia libera, "Libreria (software)", http://it.wikipedia.org/wiki/Libreria_(software), agosto 2011.
[3.9] libusb, http://www.libusb.org
[3.10] libusb-win32, http://www.libusb.org/wiki/libusb-win32
[3.11] libusb-1.0, http://www.libusb.org/wiki/libusb-1.0
[3.12] "OpenCV 2.1 C++ Reference" http://opencv.willowgarage.com/documentation/cpp/index.html
[3.13] Enciclopedia libera, "Microsoft Visual Studio", http://it.wikipedia.org/wiki/Microsoft_Visual_Studio, agosto 2011.