UNIVERSITÁ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL' INFORMAZIONE

TESI DI LAUREA TRIENNALE IN
INGEGNERIA ELETTRONICA

Realizzazione di un Driver Mouse per FPGA su board NEXYS 3

RELATORE: **Vogrig Daniele**

TESI DI LAUREA DI:
Andreoli Stefano
Matr. 593800

ANNO ACCADEMICO 2012 / 2013

Indice

1) Introduzione

L'obiettivo di questo lavoro di tesi è quello di realizzare un driver per la gestione di una periferica connessa alla board FPGA tramite porta USB. In particolare la periferica presa in esame è il mouse.
Il progetto è implementato attraverso il linguaggio VHDL, specifico per la descrizione di architetture e per la progettazione hardware, ed il suo utilizzo è ampio in tutta Europa essendo uno standard nel settore.
La periferica connessa alla scheda, che sia essa un mouse o una tastiera, prevede l'invio delle informazioni utili per una corretta acquisizione attraverso il protocollo PS/2.
Nella pagina successiva è riportata una spiegazione generale di tale protocollo valido per un qualsiasi dispositivo che ne faccia utilizzo, mentre si rimanda nei paragrafi successivi per una più dettagliata esposizione riguardo la gestione del mouse.

Il presente lavoro di tesi è principalmente organizzato in quattro parti.
Il **primo capitolo**, dopo una breve introduzione, offre una descrizione generale del protocollo PS/2.
Nel **secondo capitolo** vengono forniti alcuni concetti fondamentali riguardo alle FPGA e al linguaggio e software di sviluppo, per poi passare ad una completa illustrazione dei componenti della board Nexys 3 e dell'attivazione e funzionamento del mouse.
Il **terzo capitolo** affronta le fasi di progettazione e realizzazione del progetto.
Nel **quarto capitolo** si spiegano le modalità di test e i risultati sperimentali della prova su scheda e si traggono le conclusioni sul lavoro svolto.

1.1 Protocollo PS/2

Il PS/2 [5] è un protocollo seriale, sincrono e bidirezionale sviluppato dall'IBM e ampiamente usato al giorno d'oggi per far comunicare PC con tastiere e mouse.
Esistono due tipologie di porte molto simili tra loro: 5–pin DIN e 6–pin DIN (*Figura 1.1*). I due connettori dal punto di vista elettrico sono perfettamente compatibili, l'unica differenza sta nella posizione dei pin, per cui nel caso si possedesse un dispositivo con connettore a 5–pin è sufficiente l'uso di un adattatore.

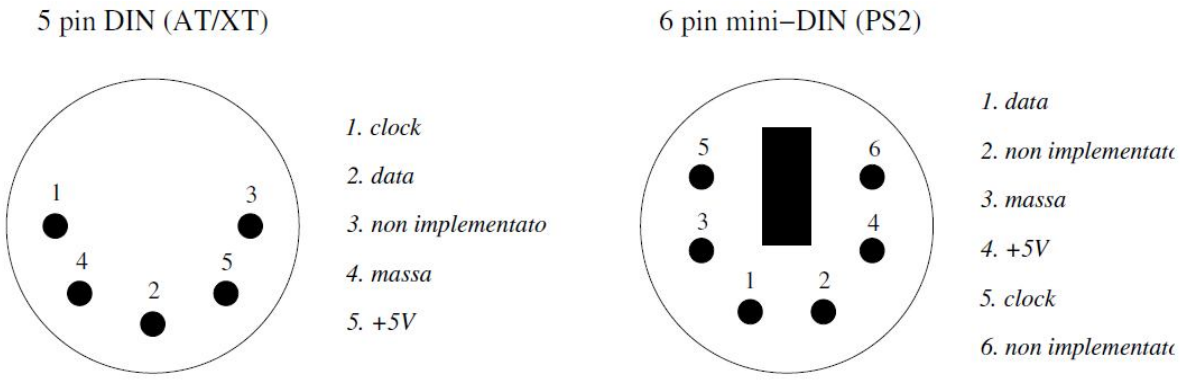


Figura 1.1: porta 5 – pin DIN e porta 6 – pin DIN

Tutta la comunicazione si basa su un bus formato da due linee bidirezionali: la linea di clock e la linea dati. Il bus è in uno stato di attesa quando le due linee di clock e di dati sono in uno stato logico alto: questo è l'unico stato nel quale è permesso al mouse e/o tastiera di trasmettere dati. L'host, cioè la scheda, ha sempre il controllo finale sul bus e può in ogni momento inibire la comunicazione mettendo a massa la linea di clock; alla periferica invece spetta il compito di generare il clock.

I dati spediti dalla periferica all'host sono letti sul fronte di discesa del clock; viceversa i dati trasmessi dall'host alla periferica vengono letti sul fronte di salita. La frequenza del clock deve stare all'interno del range 10,0 KHz – 16,7KHz, cioè il clock deve rimanere alto per almeno 30 µs. Questi parametri sono estremamente importanti per un corretto funzionamento del sistema.

Comunicazione periferica → host : quando la periferica vuole spedire un dato, come prima cosa controlla la linea di clock per assicurarsi che sia in uno stato logico alto. Se non lo è, significa che l'host sta inibendo la comunicazione e la periferica deve bufferizzare i dati da spedire finché la linea di clock non viene rilasciata per almeno 50 µs, quindi può spedire i suoi dati. Tutti i dati vengono trasmessi in modo seriale un bit alla volta, inviando frame di undici bit (*figura1.2*) composti da:

- 1 bit di start. Questo bit è sempre posto a 0;
- 8 bit di dati, ordinati dal meno significativo;
- 1 bit di parità;
- 1 bit di stop. Questo bit è sempre posto a 1.

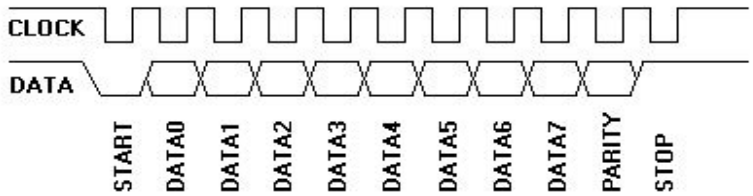


Figura 1.2: periferica → host

Comunicazione host → periferica : se l'host vuole trasmettere dei dati deve innanzitutto inibire la comunicazione mettendo a massa il clock per almeno 100 µs, quindi mettere a massa anche la linea dati e rilasciare il clock: questa sequenza fa capire alla periferica che sono in arrivo dei dati e quest'ultima inizia a generare il clock. Per trasmettere i dati la periferica usa lo stesso protocollo a undici bit visto nella sezione precedente (*figura 1.3*).

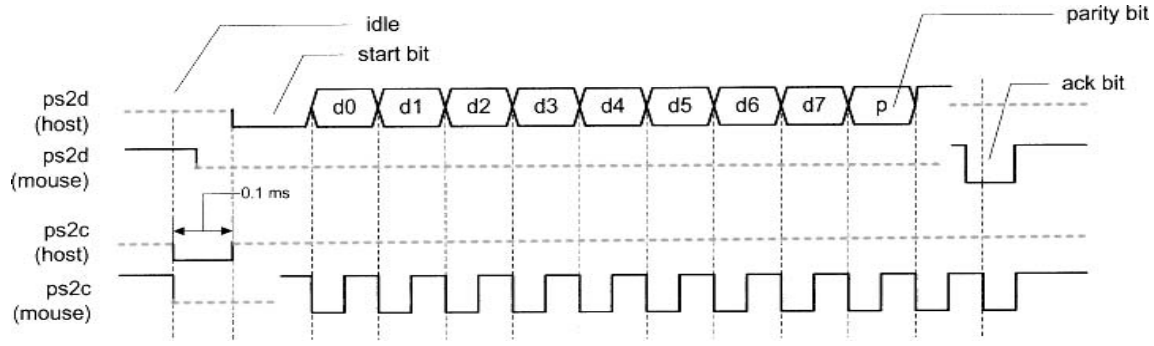


Figura 1.3: host → periferica

2) Gli strumenti e le specifiche tecniche

2.1 FPGA

L'architettura di riferimento impiegata, ovvero la FPGA, è un ritrovato tecnologico che ha già qualche decennio di vita, e la conoscenza della sua esistenza è circoscritta agli ambiti specializzati nei sistemi embedded. Nei paragrafi che seguono si cercherà di fare chiarezza su cosa sia, su come funzioni e sui suoi impieghi.

2.1.1 Cos'è una FPGA

Il dispositivo FPGA (*Field Programmable Gate Array*) [1][8] è un dispositivo digitale la cui funzionalità è programmabile via software. Non si tratta della classica programmabilità di un processore ma della possibilità di riconfigurare le risorse logiche disponibili per implementare un progetto complesso a livello di singole interconnessioni tra banchi di componenti logici.

I banchi e le connessioni variano da chip a chip, e per quanto riguarda quest'ultime, la FPGA ne dispone sia di brevi e veloci sia di più lente ma più lunghe che, attraversando tutto il dispositivo, possono quindi essere condivise da più parti dell'architettura e farle comunicare.
L'architettura tipica di una FPGA, *figura 2.1*, è caratterizzata da una matrice di blocchi logici programmabili interconnessi tra loro mediante connessioni, le quali vengono programmate utilizzando delle strutture chiamate switch – matrix, cioè delle matrici nelle quali sono realizzati tutti i collegamenti ammissibili. La struttura prevede, inoltre, una serie di blocchi Ingresso/Uscita che rappresentano la destinazione di tutti i segnali che comunicano con dispositivi posti al di fuori della FPGA, nonchè di tutti i segnali non instradati correttamente all'interno dell'architettura.
L'architettura generale può essere organizzata secondo array simmetrici, per riga oppure tramite PLD (*Programmable Logic Device*) gerarchici. Le FPGA della casa costruttrice *Xilinx* sono organizzate secondo array logici, quelle della *Actel* per riga,

mentre quelle della *Altera* secondo PLD gerarchici. In *figura 2.2* vengono riportati i 3 tipi di architettura appena nominati.

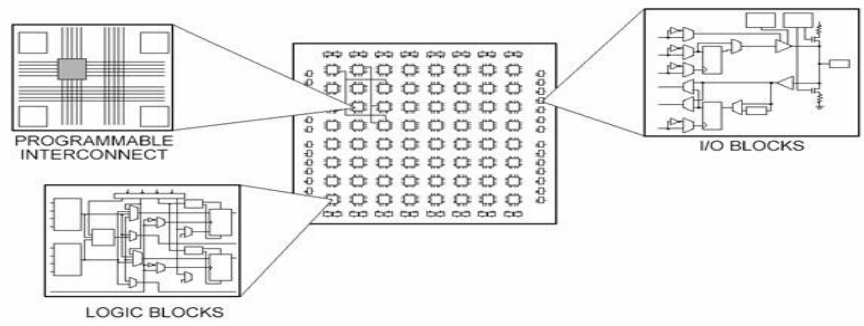
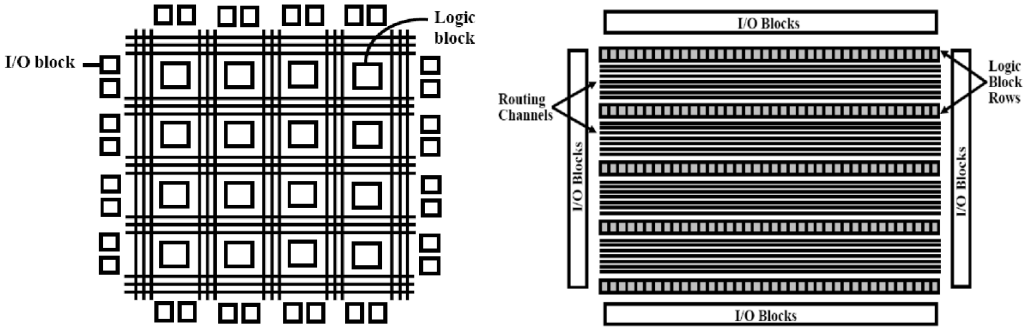
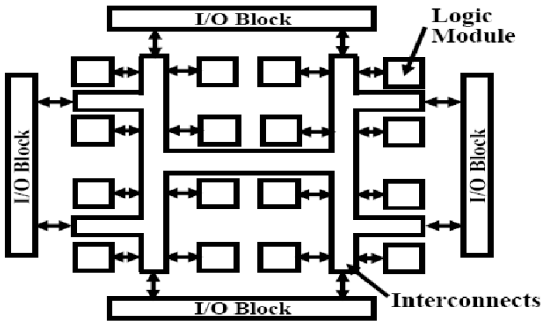


Figura 2.1: struttura tipica di una FPGA



(a) Architettura secondo array simmetrici (b) Architettura organizzata per righe



(c) Architettura a PLD gerarchici

Figura 2.2: architetture generali delle FPGA

Il risultato di un integrato così realizzato è un compromesso sotto molti aspetti. Bisogna ricordare infatti, che nel campo dei circuiti integrati vi è una vasta scelta: infatti si trovano ASIC, MPGA [9], PLD [10] ed infine FPGA che, date le sue caratteristiche, si crea uno spazio di mercato sempre più consistente.

Le *Application Specific Integrated Circuit*, ASIC, sono dei chip personalizzati in base alla necessità del costruttore dell'apparato su cui è montato e sono progettati per svolgere una determinata applicazione. Questa personalizzazione avviene interamente in fase di produzione ed ha conseguentemente alti costi e tempi per le fasi di progettazione e prototipazione. Inoltre una volta prodotti non sono più modificabili, perciò ne consegue degli strumenti decisamente poco flessibili e pratici per i piccoli produttori e specialmente per gli studenti.

Tra i dispositivi riprogrammabili si trovano MPGA, PLD e FPGA. La *Mask Programmable Array*, MPGA, è il dispositivo più grande e veloce. Essa consiste in una base di transistor già impostati in fase di produzione secondo le esigenze dell'utente a cui poi resta il compito di collegarli secondo le necessità pre-programmate che vincolano anche eventuali cambiamenti di struttura; il risultato è un chip piuttosto costoso e non molto flessibile sempre a causa del processo di produzione dal quale bisogna passare. Le *Programmable Logic Device*, PLD, sono invece i dispositivi più piccoli e meno costosi ma anche con meno potenzialità; di solito sono ROM, ad esempio E^2PROM, PAL, PLA e altre simili. La FPGA si colloca, per granularità di personalizzazione, quantità di risorse disponibili e costi, nel mezzo fra le due.

Quest'ultima effettivamente costa poco per produzioni in piccola scala e la fase di prototipazione è più semplice e veloce.

2.1.2 Vantaggi e svantaggi

Le FPGA sono elementi che presentano caratteristiche intermedie rispetto ai dispositivi ASIC (*Application Specific Integrated Circuit*) da un lato e a quelli con architettura PAL (*Programmable Array Logic*) dall'altro. L'uso di tali componenti comporta alcuni vantaggi rispetto agli ASIC: sono, infatti, dei dispositivi standard la cui funzionalità da implementare non viene impostata dal produttore che quindi può produrre su larga scala a basso prezzo. La loro genericità li rende adatti a un gran

numero di applicazioni come elettronica di consumo, comunicazioni, progettazione di automobili, aerospaziale, militare e biomedicale.
Essi sono programmati direttamente dall'utente finale, consentendo la diminuzione dei tempi di progettazione, di verifica mediante simulazioni e di prova sul campo dell'applicazione.
Il grande vantaggio rispetto agli ASIC è che permettono di apportare eventuali modifiche o correggere errori semplicemente riprogrammando il dispositivo in qualsiasi momento.
Di contro hanno prestazioni inferiori, una minore integrazione e sono antieconomici per applicazioni con elevati volumi di produzione (milioni di pezzi), poichè il prezzo unitario del dispositivo è superiore a quello degli ASIC, che permettono in questi casi di ammortizzare gli elevati costi di progettazione.

2.1.3 Metodologie di programmazione delle FPGA

Generalmente per descrivere il sistema e poterlo trasformare, tramite i tool di sintesi, nello stream di bit che andrà a programmare i chip delle FPGA, si utilizzano linguaggi di alto livello come il Verilog o il VHDL. Con questi linguaggi ci si può permettere di usare librerie di componenti predefiniti e lavorare con blocchi e macro – blocchi di alto livello. Inoltre esiste la possibilità di usare la modalità schematic – entry che consente un approccio veloce e semplificato a tale tecnologia collegando tra loro blocchi già forniti dal costruttore.
Molte case costruttrici (le principali sono: *Xilinx*, *Altera* e *Actel*) forniscono gratuitamente sistemi di sviluppo che supportano quasi tutta la loro gamma di prodotti.

2.2 Linguaggio VHDL

Per la descrizione hardware esistono due diversi tipi di linguaggi di programmazione:

1. Linguaggi di programmazione già esistenti estesi nelle funzionalità e sintassi per permettere di descrivere anche oggetti hardware, questo è il caso del linguaggio SystemC.

2. Linguaggi costruiti ex – novo, specifici per la descrizione di architetture ma comunque con strutture simili a quelle dei linguaggi di programmazione comuni, allo scopo di semplificare l'uso per chi programma in altri ambiti. Questo è il caso dei linguaggi di descrizione hardware, HD, come il Very Hight Speed Integrated Circuits, VHSIC, Hardware Description Language, VHDL, e Verilog.

Il linguaggio scelto per descrivere l'architettura implementata è il VHDL-93 per l'ampio utilizzo che se ne fa praticamente in tutta Europa per la progettazione hardware che lo porta ad essere uno standard di fatto in materia.

2.2.1 Cos'è il VHDL

Il VHDL [2][7] è un linguaggio per la descrizione dell'hardware (un *Hardware Description Language*), che può essere utilizzato per la documentazione, la simulazione e la sintesi di sistemi digitali. Inizialmente, nei primi anni '80, lo sviluppo del VHDL è stato supportato dal dipartimento della difesa statunitense, nell'ambito di un progetto denominato VHSIC (*Very High Speed Integrated Circuits*). VHDL è infatti un acronimo che sta per VHSIC Hardware Description Language. Nel 1987 il VHDL è stato adottato come standard dalla IEEE (*Institution of Electrical and Electronics Engineers*); questa prima distribuzione ufficiale del linguaggio è nota come VHDL-87. Nel 1993 lo standard è stato revisionato dalla IEEE e si è giunti così alla versione attuale del linguaggio, nota come VHDL-93, che differisce solo in pochi aspetti dal precedente.

Il VHDL è stato infatti introdotto come linguaggio standard per la documentazione di

sistemi digitali complessi. Il linguaggio è nato quindi con lo scopo di fornire una descrizione non ambigua di un sistema digitale, che potesse essere interpretata univocamente dai vari progettisti impegnati nello sviluppo del sistema stesso. Una delle caratteristiche richieste al VHDL è la possibilità di simulare il sistema descritto, sia a livello funzionale sia tenendo in considerazione tutte le non idealità che si hanno in un circuito reale.
Negli anni seguenti, oltre che per la documentazione e la simulazione, esso ha assunto un ruolo importante nella fase di sintesi dei sistemi digitali, cioè a partire da una descrizione comportamentale di un sistema permette di ottenere automaticamente una descrizione del circuito a basso livello attraverso una *netlist* in cui il sistema viene descritto come interconnessione di porte logiche elementari appartenenti ad un'opportuna libreria.

2.2.2 Caratteristiche distintive

La caratteristica principale di questo linguaggio, che permette di affrontare problemi con logiche diverse, è la possibilità di scegliere il livello di diversificazione dell'hardware. È infatti possibile descrivere i componenti in due maniere diverse: una, la più ostica forse e più impegnativa in termini di codice da scrivere, è la cosiddetta *architetturale*, l'altra, più semplice per un programmatore, è la *comportamentale*, *behavioural*. La prima consiste nel descrivere il componente nella sua struttura logica, descrivendo quindi le connessioni fra le porte logiche, gli ingressi e le uscite; il flusso di lavoro fra le varie parti avviene in parallelo. La seconda, invece, permette una descrizione molto simile alla programmazione imperativa, quindi si hanno i costrutti tipici di questo stile come i cicli *for* e *while*, l'*if then else* e molti altri; il flusso di esecuzione, ovviamente, all'interno di queste strutture è di tipo sequenziale.

La potenza del VHDL sta in costrutti di poche righe che descrivono circuiti di decine di migliaia di porte, consentendo al progettista di concentrarsi sul comportamento del sistema e non sui dettagli di implementazione. Questo comporta, però, una profonda conoscenza del linguaggio, perchè se il codice di un determinato blocco, anche se formalmente corretto, è scritto in maniera ridondante, la sintesi potrebbe non essere efficiente poichè è soltanto il sintetizzatore che decide l'implementazione a livello di

gate. La differenza sostanziale rispetto ai comuni linguaggi di programmazione è che le istruzioni vengono eseguite non in successione ma in parallelo, richiedendo un approccio mentale al linguaggio totalmente diverso da quello tradizionale.

Le principali fasi di descrizione sono due:

- nella prima l'oggetto viene descritto come *ENTITY*, ovvero come il componente viene visto dall'esterno: questa sezione comprende generalmente le porte di comunicazione, i tempi di ritardo, la larghezza dei bus e i parametri di configurazione;

- nella seconda si progetta la *ARCHITECTURE*, ovvero l'architettura interna dove si descrive il funzionamento del dispositivo: è questa la parte più importante perchè ne costituisce la descrizione funzionale vera e propria.

Gli elementi distintivi di un modello VHDL sono i *segnali* e i *processi*. Un processo è un blocco di codice che descrive il funzionamento di un modulo di logica sequenziale o combinatoriale, e più processi diversi comunicano tra loro attraverso i segnali che spostano i dati da una parte all'altra del sistema.
Inoltre i processi sono delle strutture concorrenziali, cioè hanno tutti la stessa priorità e il loro ordine di scrittura all'interno dell'architettura non influenza il risultato finale. Un processo viene attivato ed esegue la sua funzione in risposta ad un evento, cioè ad un cambiamento di valore su uno dei segnali a cui è sensibile, i quali sono indicati in una apposita lista denominata *sensivity list*.

Successivamente per testare il codice prodotto esiste la possibiltà di creare, sempre in VHDL, dei file denominati *testbench*, i quali definiscono gli stimoli da dare agli ingressi e quindi permettere di valutare l'andamento delle uscite. Questi file istanziano il componente da testare e contengono dei processi che definiscono i vari segnali. Essendo una rappresentazione del mondo esterno, i testbench normalmente non presentano porte di ingresso e uscita nella loro dichiarazione entity.

2.2.3 Software di sviluppo

Per sviluppare il progetto del driver mouse e curarne il codice, compilarlo, sintetizzarlo ed infine usarlo per poter programmare la board si sono utilizzati tre software: *ISE Design Suite* con *Project Navigator*, *Isim Simulator* e *ISE Impact*.

L'ISE (*Integrated Software Environment*) Design Suite è il primo strumento che è stato utilizzato , ed è un software sviluppato da Xilinx con la collaborazione della Mentor Graphics per l'interfaccia grafica Project Navigator. ISE è un ambiente di sviluppo per progetti di architetture per FPGA; esso contiene un editor di testi e una serie di strumenti utili per lo sviluppo del progetto in tutte le sue fasi, a partire dal codice VHDL per finire con il codice sintetizzato e adattato per essere portato su Ise Impact. Le principali funzionalità messe a disposizione sono nel riquadro *Processes* in *figura 2.3* e sono organizzate in cinque gruppi: Design Utilities, User Constraints, Synthesize – XST, Implement Design e Configure Target Device. Tramite *User Constraints* si ha la possibiltà di creare i vincoli temporali riguardo il corretto campionamento degli eventuali Flip – Flop al successivo fronte di clock e di impostare i pin fisici della scheda per eventuali segnali di uscita come ad esempio i led o i display a sette – segmenti. Il gruppo *Synthesize – XST* contiene il compilatore per controllare la sintassi ed il comando per sintetizzare il codice, operazione indispensabile per il trasferimento del lavoro su un dispositivo FPGA. Sotto *Implement Design* si trovano tutti gli strumenti dediti alla finale traduzione del codice sintetizzato e mappatura su scheda. Infine *Configure Target Device* è l'ultima fase di lavoro dove, dopo aver generato il bitstream tramite Generate Programming File, si effettua il download del bitstream su scheda.
In alto a sinistra di *figura 2.3* si trovano i comandi per iniziare un progetto tramite File→New Project, dove si scelgono locazione del progetto, file sorgenti e tipologia di FPGA da utilizzare. I risultati di tali scelte verranno visualizzate nella finestra *Hierarchy*, sempre in alto a sinistra, dove si potranno creare nuovi file di codice che verranno visualizzati per l'implementazione ed eventuali modiche e/o correzioni nel riquadro più ampio.
La segnalazione di errori di sintassi, di sintetizzazione e di implementazione vengono visualizzati nel riquadro *Console* sempre di *figura 2.3*.

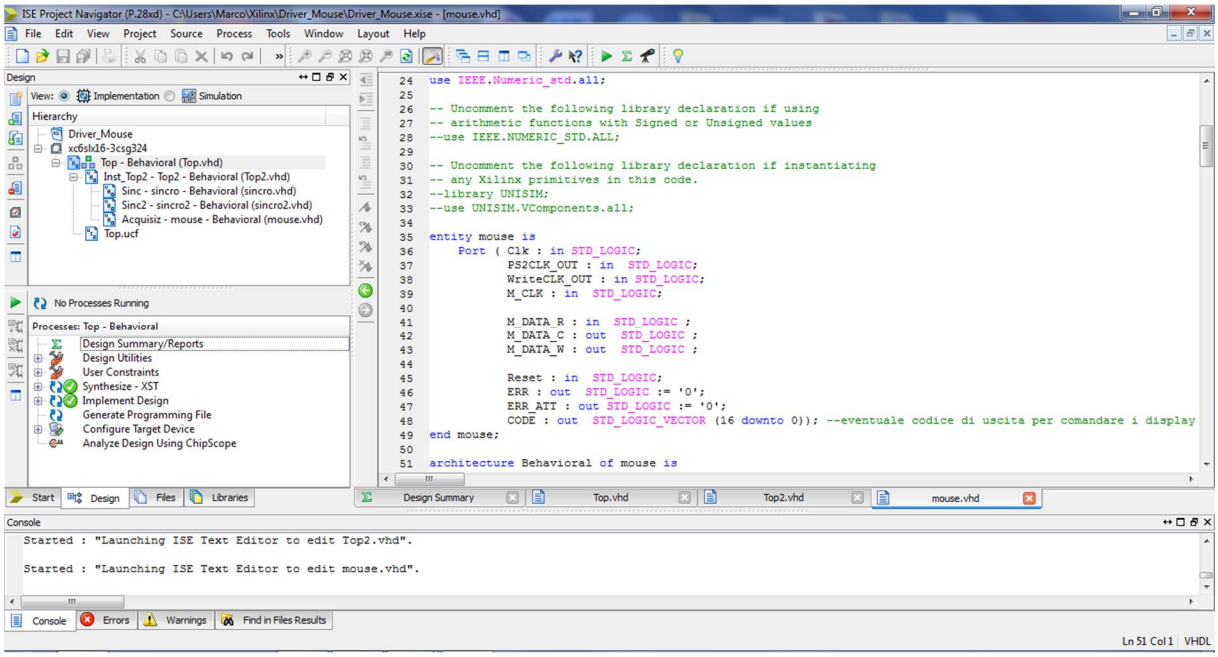


Figura 2.3: interfaccia grafica di ISE Project Navigator

Isim Simulator e ISE Impact sono due strumenti complementari ad ISE Design Suite. Il primo offre la possibiltà di testare il codice appena prodotto attraverso la visualizzazione di una simulazione che dipende da un'oppurtuna compilazione di ISE Design Suite secondo le scelte dell'utente, e può essere comportamentale o temporale. La prima verifica il comportamento descritto dal codice ma non tiene conto delle porte logiche e dei ritardi che caratterizzano il sistema reale; la seconda, invece, tiene conto di tutti i ritardi della logica e delle interconnessioni.

Il secondo lo si utilizza, una volta finito di verificare la correttezza del progetto tramite sintetizzazione, implementazione e simulazione, quando si vuole inviare il codice bitstream, prodotto da Generate Programming File, alla board per la visualizzazione finale.

2.3 FPGA – Nexys 3

La FPGA – Nexys 3 [3][4] è una scheda digitale pronta all'uso, sviluppata su una piattaforma basata sul componente Xilinx Spartan – 6 LX16. Come si può vedere dallo schema a blocchi, *figura 2.4*, i componenti principali sono:

- *Xilinx Spartan 6 XC6SLX16 CSG324C FPGA* contenente 14500 blocchi logici, 18000 Filp – Flops, 32 blocchi di RAM da 576 Kbits, 32 unità DSP48A1, 4 DCM, fino a 232 I/O;
- *16 M – byte di memoria non volatili Micron PCM (Phase Charge Memory)* che permette di memorizzare la programmazione della FPGA, può essere utilizzata per la memorizzazione di dati non volatili e si ha accesso sia parallelo che seriale tramite porta SPI;
- *16 M – byte di Cellular RAM* che può operare come memoria SRAM asincrona con tempi di lettura e scrittura di 70ns, oppure come memoria sincrona con velocità di bus fino a 80 MHz;
- *Porta VGA a 8 bit (256 colori);*
- *Porta USB HID Host* che permette di collegare una tastiera o un mouse USB;
- *Bridge USB – UART (Porta seriale)* che utilizza la porta USB per comunicazioni di tipo seriale tramite lo standard RS – 232;
- *Oscillatore CMOS a 100 MHz;*
- *GPIO (General Purpose Imput/Output)* che è un'interfaccia che include 8 led, 6 pulsanti, 8 switch e 4 display a sette – segmenti.

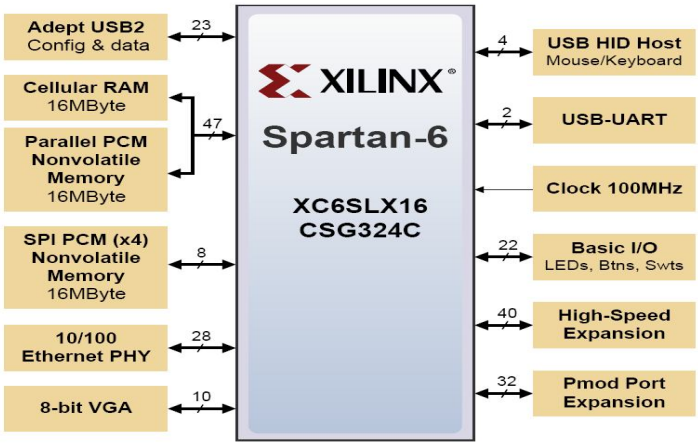


Figura 2.4: schema a blocchi Xilinx Spartan – 6

Una vista più completa della scheda si ha in *figura 2.5*, dove si può vedere il posizionamento dei vari componenti che la formano.

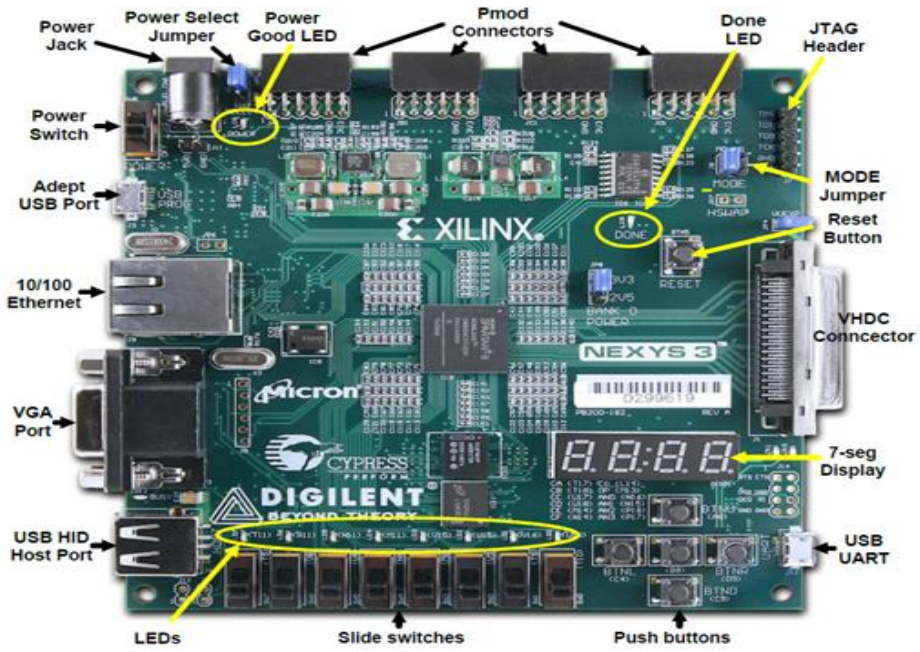


Figura 2.5: vista board Nexys 3

Dall'immagine qui sopra visualizzata sono ben visibili tutti i componenti e le porte di Ingresso/Uscita precedentemente elencate, in particolare i pulsanti, gli switch, i led, il sistema di alimentazione (Power Switch), i quattro display a sette – segmenti e la porta USB HID Host.
Di seguito verrà riportata una descrizione dei più importanti componenti della board soffermandosi maggiormente su quelli utilizzati nell'implementazione e nella realizzazione del progetto.

I pulsanti e gli switch sono connessi alla FPGA attraverso una serie di resistori, *figura 2.6*, in modo da prevenire eventuali guasti dovuti, per esempio, da una loro non corretta assegnazione ai pin della scheda come output. I pulsanti solo quando vengono premuti generano un'uscita a livello logico alto mentre gli switch generano un'uscita a livello logico alto o basso in base al loro posizionamento.

Gli otto led sono connessi alla board mediante resistenze di 390 Ω, *figura 2.6*, e si accendono quando i pin di I/O sono a livello logico alto per via di una tensione applicata. Ulteriori led sono quelli non accessibili che segnalano l'accensione, la programmazione e lo stato della porta USB ed Ethernet della scheda.

Ognuno dei quattro display a sette – segmenti è composto, come suggerisce il nome, da sette segmenti con altrettanti led all'interno per la loro illuminazione combinati come in *figura 2.7*. Ci sono otto catodi comuni tra i quattro display dove i rispettivi led si accendono mediante un segnale a livello logico basso, e la selezione del dispositivo da illuminare viene fatta sempre portando a livello logico basso il segnale che pilota l'anodo comune.

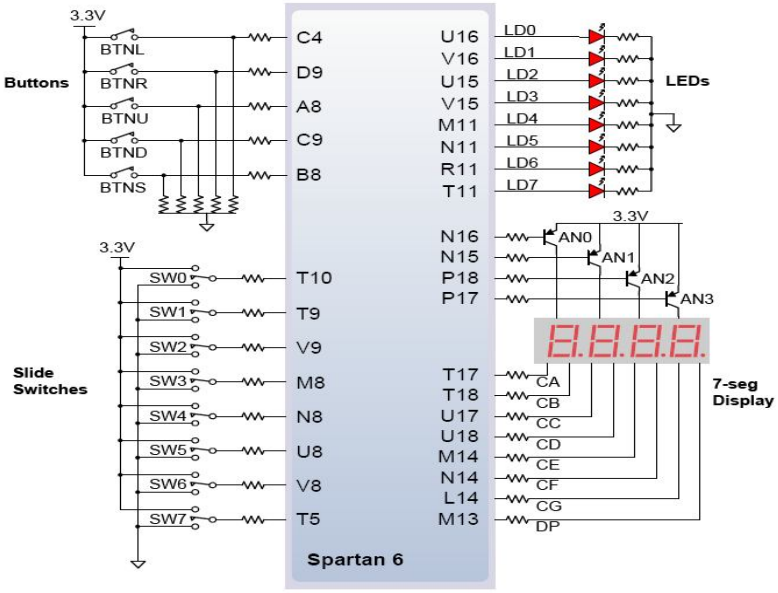


Figura 2.6: schema pulsanti – deviatori – led – display

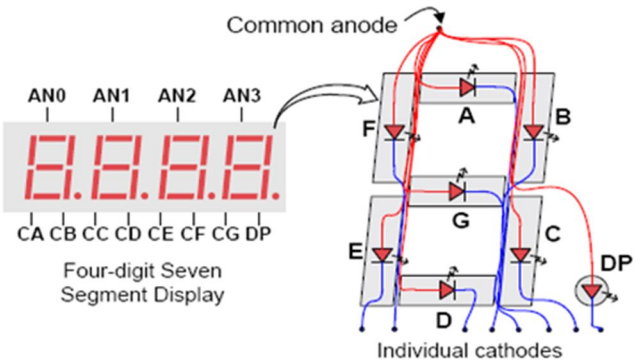


Figura 2.7: display a sette – segmenti

Per una corretta visualizzazione di dati diversi nei display, si devono accenderli alternativamente in modo da utilizzare la persistenza dell'occhio umano. Due accortezze molto importanti sul loro tempo di accensione sono:

- non può essere troppo breve perchè i led richiedono un certo tempo per accendersi;
- non può essere troppo veloce perchè all'occhio umano risulterebbe un fastidioso lampeggio.

Il sistema di alimentazione (Power Switch) della board, *figura 2.8*, può avvenire attraverso una tensione continua da 3.6V a 15V o normalmente si utilizzano i 5V della porta USB, ma ciò non vieta che si possono usare sia alimentatori esterni che batterie.

La scheda presenta anche un oscillatore a 100 MHz, *figura 2.9*, che le fornisce il segnale di clock per le applicazione dell'utente.

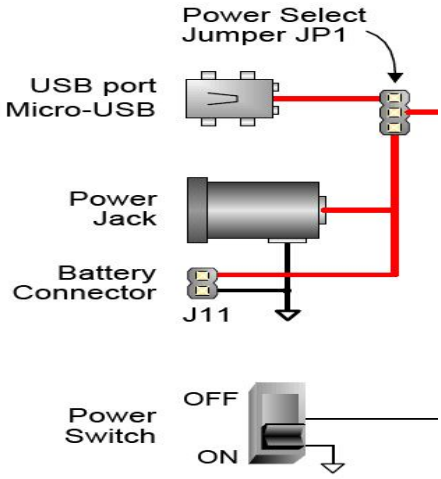


Figura 2.8: power switch

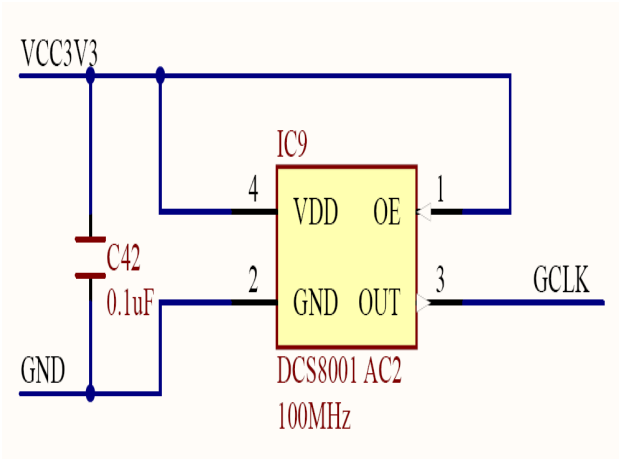


Figura 2.9: oscillatore 100 MHz

Sulla board è presente la porta VGA, la quale viene comandata da 8 segnali che creano le otto gradazioni dei tre colori principali e altri due segnali standard denominati HS (Horizontal Sync) e VS (Vertical Sync), *figura 2.10*. Le uscite della FPGA sono digitali e assumono solamente i valori 0V e 3.3V, e quindi grazie alle resistenze e ai 75 Ω di terminazione del monitor si ottengono valori intermedi contenuti nello standard VGA: 0 – 0.7V. Inoltre i due segnali HS e VS utilizzano lo

standard TTL, la frequenza di clock della VGA è pari a 25 MHz e tutti questi parametri permettono di realizzare tale porta con una risoluzione di 640x480 a 60 Hz adoperando 256 colori.

Per quanto riguarda la temporizzazione, essa è gestita dal controller VGA, integrato nel monitor, che genera due rampe di comando, verticale ed orizzontale, partendo dai segnali HS e VS che sono forniti dalla porta come detto in precedenza. Per la visualizzazione si defiscono i pixel uno ad uno partendo dall'angolo in alto a sinistra (0,0) fino a quello in basso a destra (479, 639), *figura 2.11*, e quando si arriva fuori schermo i segnali RGB bisogna che siano a '0'. Anche qui i dati si susseguono alla frequenza di clock pari a 25 MHz.

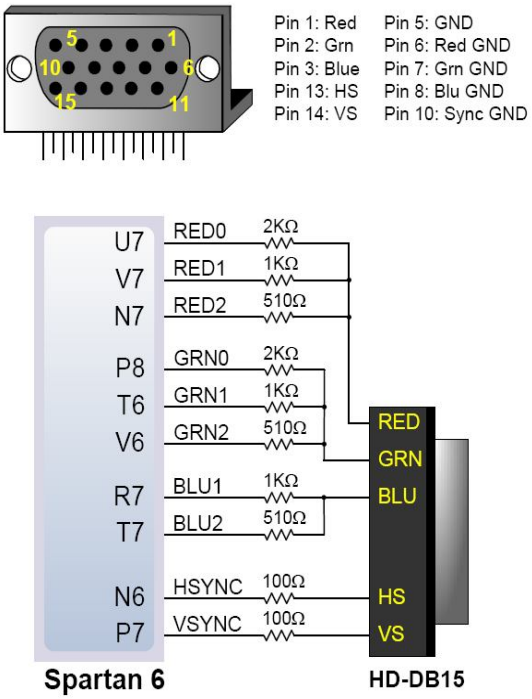


Figura 2.10: segnali porta VGA

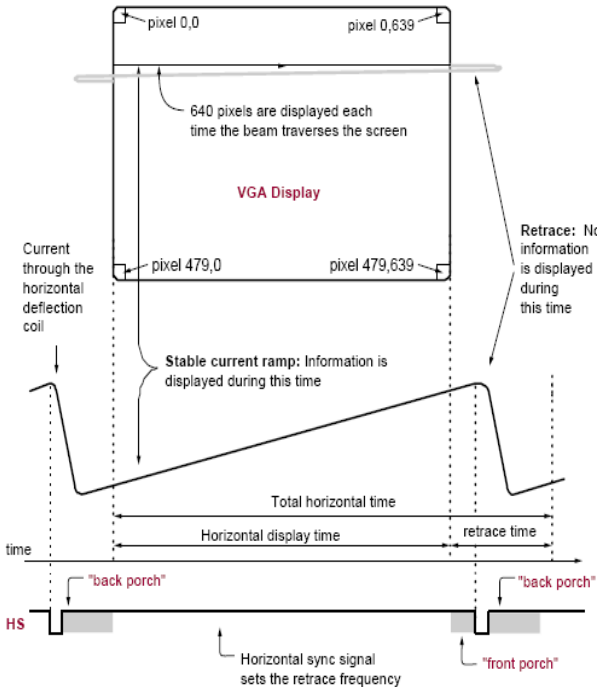


Figura 2.11: temporizzazione VGA

La scheda predispone anche una porta USB – UART bridge, che emula la comunicazione di tipo seriale che veniva eseguita con la porta RS232 presente nei vecchi computer. La porta si interfaccia con la scheda tramite un chip FT232 che usa due segnali TXD/RXD per comunicare con il microprocessore Spartan 6 come è illustrato in *figura 2.12*.

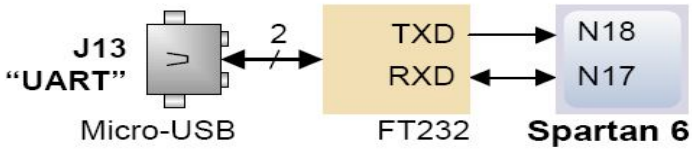


Figura 2.12: porta USB – UART bridge

È presente pure un layer fisico Ethernet che permette la comunicazione a 10/100 Mbps. Il controllo è piuttosto complesso e utilizza dei blocchi hardware per FPGA pre–progettati ed associati ad opportune istruzioni software che necessitano di utilizzare uno sviluppo a livello di sistema. Per tali applicazioni il Design Suite della Xilinx mette a disposizione il tool EDK. La gestione della comunicazione avviene in questo caso attraverso il componente SMSC LAN8710A, il quale è da tramite tra il connettore RJ – 45 e il microprocessore Spartan 6 come si può vedere in *figura 2.13*.

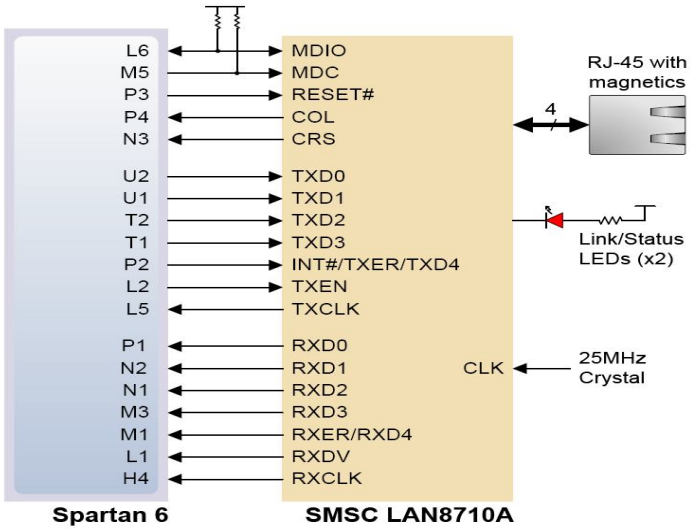


Figura 2.13: porta Ethernet PHY

La board una volta accesa può essere programmata in più modi. Come si vede in *figura 2.14* un modo è quello di usare il PC attraverso l'uso del connettore Adept USB Prog Port ogni volta che la scheda è accesa; un'altra soluzione è usare le due memorie non volatili, una parallela e una seriale, dove il codice del programma è stato memorizzato e verrà traferito mediante l'uso della BPI Port o della SPI Port;

un'ultima possibiltà è attraverso l'uso della porta USB HID Port dove il programma è scaricabile da una chiavetta USB.

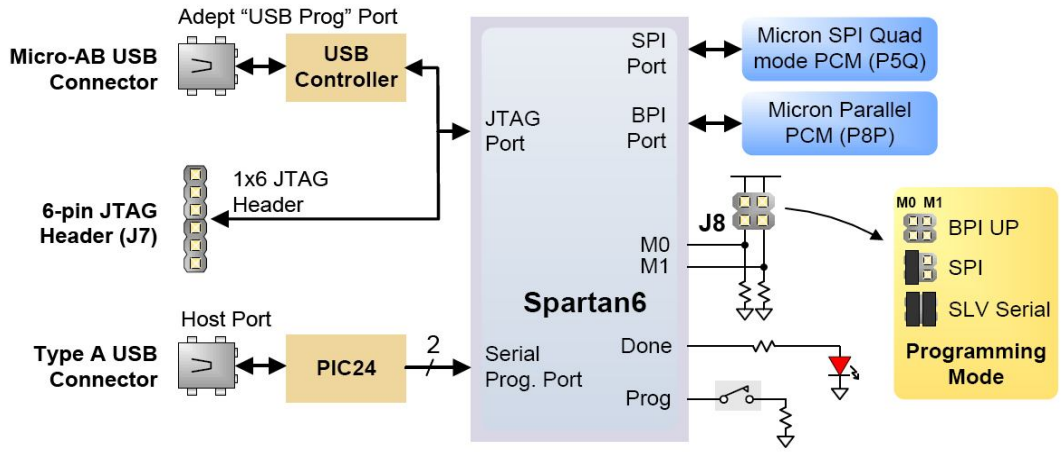


Figura 2.14: modalità di programmazione

La porta USB HID Host, oltre che per la possibiltà di programmare la scheda, permette di collegare al connettore J4, *figura 2.15*, una tastiera o un mouse.
Il controllo di tali perifiche avviene attraverso un microcontrollore PIC24FJ192 della Microchip ed una precisa gestione del protocollo PS/2 che è stato analizzato nel paragrafo introduttivo e la parte relativa al mouse verrà spiegata dettagliatamente nel paragrafo successivo. Nella figura sottostante, presa dal manuale della scheda, presenta però degli errori in quanto anche le porte K_CLK e M_CLK del microcontrollore relative alla tastiera e al mouse sono anch'esse di tipo ingresso/uscita e non solo di ingresso.

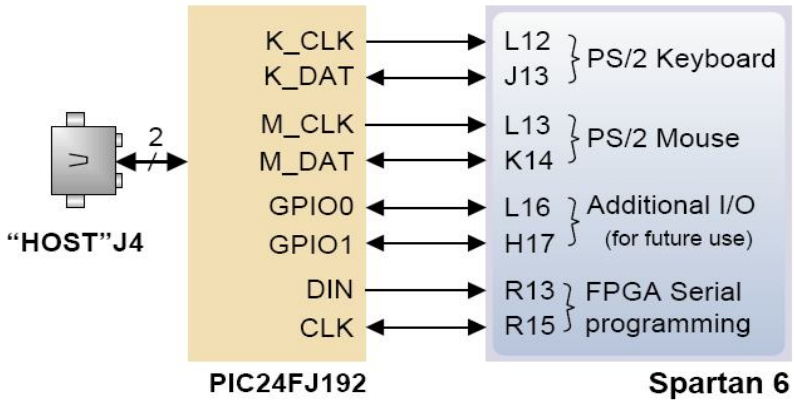


Figura 2.15: porta USB HID Host

Infine la board presenta dei connettori di espansione per permettere di aumentare le funzionalità della scheda. Essi sono i 68 pin del connettore VHDC per un’alta velocità di comunicazione di ingresso/uscita e gli 8 pin dei connettori Pmod per una bassa velocità di comunicazione.

2.4 Interfaccia mouse del protocollo PS/2

L'interfaccia mouse PS/2 [6] utilizza un protocollo seriale bidirezionale per trasmettere alla FPGA il movimento e gli stati dei pulsanti. La scheda può inviare una serie di comandi alla periferica, ad esempio il comando reset, quello per cambiare la risoluzione, il report rate o di disabilitazione della stessa. La board alimenta il mouse con un voltaggio pari a 5V e con una corrente di circa 100 mA.

L'interfaccia standard del protocollo prevede l'invio di tre parole da undici bit ciascuna ogni volta che esso viene mosso o gli viene premuto un tasto. Ogni parola inviata inizia sempre con un bit di start che è lo zero, successivamente otto bit dove viene comunicato lo stato dei bottoni e/o del movimento, il decimo bit è di parità dispari, cioè di controllo di un corretto invio, ed infine l'ultimo bit di stop che è rappresentato dall'uno.
Nella prima parola che il mouse invia alla board ci sono i bit relativi allo stato dei pulsanti e l'indicazione del movimento se è alto, basso, destra, sinistra.
La seconda e la terza rappresentano il valore della direzione intrapresa dal mouse, il quale è espresso in complemento a due ed è dettato dai contatori che lo incrementano/decrementano, da -255 a +255, in base al movimento. Inoltre sono associati due bit di overflow nella parola precedente che indicano se il mouse raggiunge la velocità massima.
Se continuamente mosso, il mouse invia ripetutamente i dati ad ogni 50 ms circa, tranne quando lo si vuole resettare, comando 0xFF, perchè in quello stato disabilita la comunicazione per eseguire un check – up interno, e successivamente per riabilitarla bisogna eseguire il comando 0xF4.
Si possono modificare ulteriori parametri come la risoluzione, comando 0xE8, dove il contatore di default è impostato su 4 conteggi/mm, e la scala, comando 0xE7, che ha effetto non sul conteggio del movimento ma sul valore che poi esso rappresenta, e di default è impostata su 1:1.

Una visione più chiara e descrittiva si ha di seguito in *tabella 2.1*.

	Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Byte 1	Yv	Xv	Ys	Xs	1	M	R	L
Byte 2	movimento X							
Byte 3	movimento Y							

- **L**, **R** e **M** : rappresentano lo stato dei pulsanti: '1' se vengono premuti;
- **Xs** : '0' per movimento verso DX, '1' verso SX;
- **Ys** : '0' per movimento verso alto, '1' verso basso;
- **Xv** e **Yv** : sono ad '1' alla velocità massima;
- **movimento X** e **movimento Y**: valore maggiore, direzione maggiore;

Tabella 2.1: schema e descrizione dell'estensione Standard

Il mouse può operare in quattro modi diversi:

- **Reset**: è il modo iniziale in cui il mouse si inizializza e fa un'autodiagnosi;
  si entra in esso all'accensione della board o in risposta al comando 0xFF, il self – test è chiamato BAT (*Basic Assurance Test*) e i valori di default sono:
  - sample rate = 100 campionamenti/sec;
  - risoluzione = 4 conteggi/mm;
  - scala = 1:1;
  - data reporting = disabilitato.

  Il mouse una volta ricevuto il comando 0xFF, risponde con 0xFA indicando che la ricezione della parola è avvenuta correttamente e una volta finita l'autodiagnosi invia prima 0xAA e poi 0x00.

- **Stream**: è il modo operativo di default nel quale invia i dati quando c'è un movimento e/o un tasto premuto;
  il mouse, però, finchè non riceve il comando di abilitazione del data reporting,

comando 0xF4, non invia nessun pacchetto alla scheda. Il massimo campionamento dipende dal sample rate che può essere modificato, tramite il comando 0xF3, fino a 200 campionamenti/sec. Inoltre con il comando 0xEA si entra in questo stato se precedentemente si era in un altro.

- **Remote**: è il modo in cui la scheda chiede un invio di un pacchetto dati al mouse tramite il comando 0xEB. Si accede a questo stato con il comando 0xF0.

- **Wrap**: è il modo puramente di diagnostica, nel quale il mouse ripete ogni pacchetto ricevuto indietro dalla scheda, anche se questo rappresenta un comando valido. Ci sono solo due comandi, reset 0xFF o reset wrap mode 0xEC, che sono delle eccezioni in cui la parola non viene ripetuta. Questo modo come il precedente è raramente usato.

Una popolare estensione dell'interfaccia standard è la cosiddetta Microsoft Intellimouse. Essa include la gestione di due bottoni aggiuntivi e di un ulteriore asse di movimento. Queste informazioni vengono trasmesse in una quarta parola sempre da undici bit, nei quali i quattro bit meno sgnificativi indicano lo stato della rotellina, dove il numero è anch'esso in complemento a due ed il range va da -8 a +7. I successivi due bit indicano lo stato dei due pulsanti aggiuntivi e gli ultimi due bit sono sempre a zero, come si può vedere in *tabella 2.2*.
Di norma questa interfaccia è disabilitata, e per attivarla bisogna compiere determinate operazioni:

- per abilitare solo l'invio dei dati relativi alla rotellina si deve impostare in sequenza il sample rate a 200, 100 ed infine a 80 rispettando per tutti e tre le parole le fasi di voler cambiare il sample rate, 0xF3, di ricezione avvenuta, 0xFA, di impostazione del nuovo valore, di ricezione e di attivazione avvenuta tramite la parola 0x03;
- per l'invio anche dei dati relativi ai due pulsanti aggiuntivi i valori del sample rate sono 200, 200 e 80 mentre la sequenza delle fasi rimane la stessa tranne che ora la parola di attivazione è 0x04.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Yv	Xv	Ys	Xs	1	M	R	L
Byte 2	movimento X							
Byte 3	movimento Y							
Byte 4	0	0	5° P	4° P	movimento Z			

- **movimento Z** : valore in complemento a due della rotellina, range da -8 a +7, positivo movimento in avanti, negativo indietro;
- **4° P** e **5° P**: stato dei due pulsanti aggiuntivi: ‘1’ se vengono premuti.

Tabella 2.2: schema e descrizione dell’estensione Intellimouse

3) Implementazione

In questo capitolo si esporrà tutto il procedimento di implementazione del progetto, evidenziandone i passi principali, le difficoltà incontrate e le scelte prese.
Nell'esporre questo processo si seguirà l'ordine temporale col quale sono stati affrontati. Si inizierà dunque con la fase di studio di una progettazione per la gestione e visualizzazione dei dati ricevuti dal mouse, per poi passare a quella di verifica di un corretto funzionamento.

3.1 Progettazione

In questo paragrafo verrà trattata la fase di progettazione del driver mouse, rispettando le caratteristiche fondamentali descritte nei paragrafi precedenti.
Il progetto è stato diviso in tre blocchi principali: sincronizzazione, acquisizione e visualizzazione, *figura 3.1*. Il primo genera una corretta sincronizzazione tra il clock della board e quello più lento del mouse, il secondo compie le fasi di reset e di gestione dei dati ricevuti dalla periferica, ed il terzo permette la loro visualizzazione sui display e sui led. Le fasi di acquisizione e visualizzazione ciclano aggiornando continuamente i display e i led.

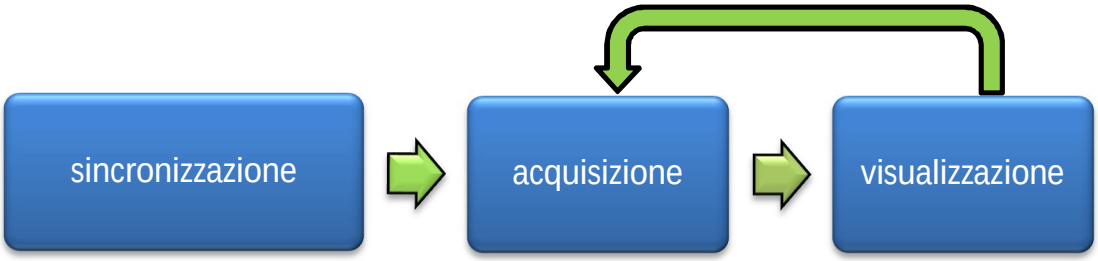


Figura 3.1: schema a blocchi del progetto

3.1.1 Buffer Tri – State

Riguardo alla gestione delle porte M_CLK e M_DATA che sono di tipo INOUT, si è dovuto procedere con la realizzazione di un buffer Tri – State per entrambe. Questo dispostivo fa si che quando si vuole scrivere sulla linea si porti ad ‘1’ il segnale di controllo, mentre quando si vuole solo leggere il dato inviato dal mouse lo si porti a ‘0’. Una rappresentazione grafica del componente nel caso di M_DATA si ha in *figura 3.2* preceduta dalle righe di codice che servono per tale implementazione:

```
M_DATA <= M_DATA_W when M_DATA_C = '1' else 'Z';
M_DATA_R <= M_DATA;
```

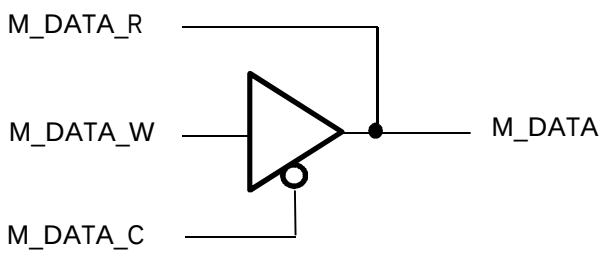


Figura 3.2: buffer Tri – State nel caso di M_DATA

3.1.2 Sincronizzazione

Per una corretta fase di invio/ricezione dati si è dovuto progettare un sistema di sincronismo mediante il quale ad ogni fronte di discesa di M_CLK_R si ha l’elevazione ad ‘1’ del segnale in uscita, PS2CLK_OUT, che andrà a scandire le fasi degli stati di Attivation, Ack, Stream, Ack2 e Shifting. L’implementazione consiste nella serie di due Flip – Flop, i quali vengono comandati dal segnale di Clk che funge da abilitazione, e la loro uscita viene portata all’ingresso di una porta AND non prima di aver negato quella corrispondente al primo.
Dallo schema e dalla simulazione, eseguita tramite un testbench del file *sincro.vhd* , riportati in *figura 3.3* e in *figura 3.4*, risulterà tutto molto più chiaro.

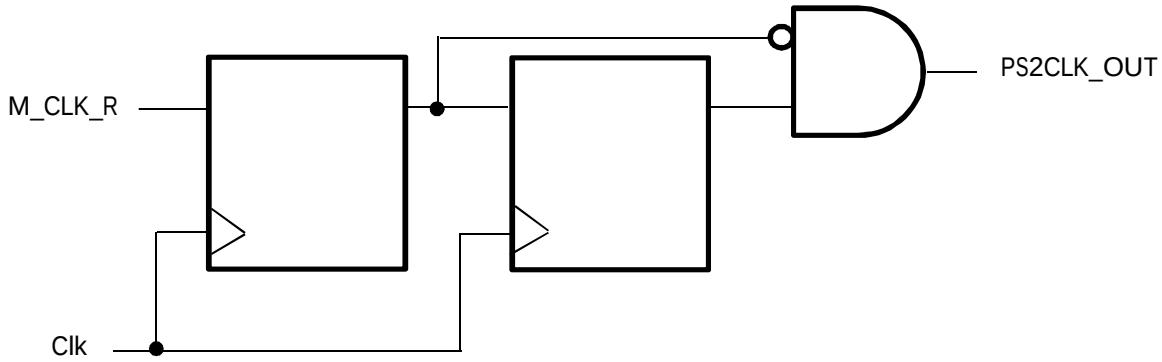


Figura 3.3: schema di sincronismo

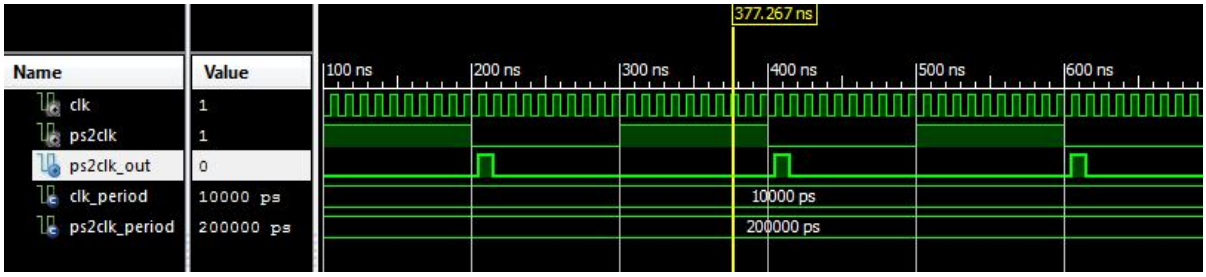


Figura 3.4: simulazione del sincronismo

3.1.3 Diagramma di flusso

La struttura del codice è stata realizzata in diversi stati, i quali hanno compiti da svolgere ben precisi l'uno dall'altro. Il primo è lo stato ***Res*** al quale si arriva ogni volta che si preme il tasto reset (pulsante centrale della board), e la sua funzione è quella di inibire la comunicazione tra mouse e board per far si di riuscire ad inviare correttamente una parola alla periferica, per esempio in questo caso 0xFF che significa reset e successivamente 0xF4 il quale la abilita all'invio dati. Nel primo caso quando il segnale *Controll* è uguale a *'0'* si passa allo stato ***Attivation*** dove si ha l'effettivo invio della parola 0xFF al mouse, il quale dopo la fase di inibizione genera il clock per la ricezione ed il bit viene letto volta per volta sul rispettivo fronte di salita. Successivamente c'è quello chiamato ***Ack*** e non fa altro che ricevere e verificare i byte di risposta da parte del mouse. Poi si torna nuovamente nello stato Res per

inibire la comunicazione per l'invio di 0xF4 che avviene nello stato **Stream** perchè il segnale *Controll* è stato portato a '1'. Anche qui dopo l'invio si procede alla ricezione e verifica dei bit di risposta della periferica nello stato **Ack2**. Ora che la fase completa di reset e attivazione è conclusa, si passa allo stato denominato ***Idle*** che rappresenta la periferica nella sua condizione di non utilizzo e ci si arriva ogni volta dopo un processo di reset – attivazione o di acquisizione dati. Successivamente c'è quello detto ***Shifting*** e avviene ogni volta che intercorre uno spostamento o una pressione di un tasto del mouse. Il compito di questo ultimo stato è quello di immagazzinare in appositi registri i bit inviati dal mouse, di verificare che non ci siano stati errori nella trasmissione dati e predisporre un codice di uscita che servirà come controllo per la successiva fase di visualizzazione sui display e led. Infine si ritorna allo stato ***Idle*** dove si è pronti per una nuova acquisizione dati o per una fase di reset se tale tasto verrà premuto. Il diagramma di flusso è illustrato di seguito in *figura 3.5*.

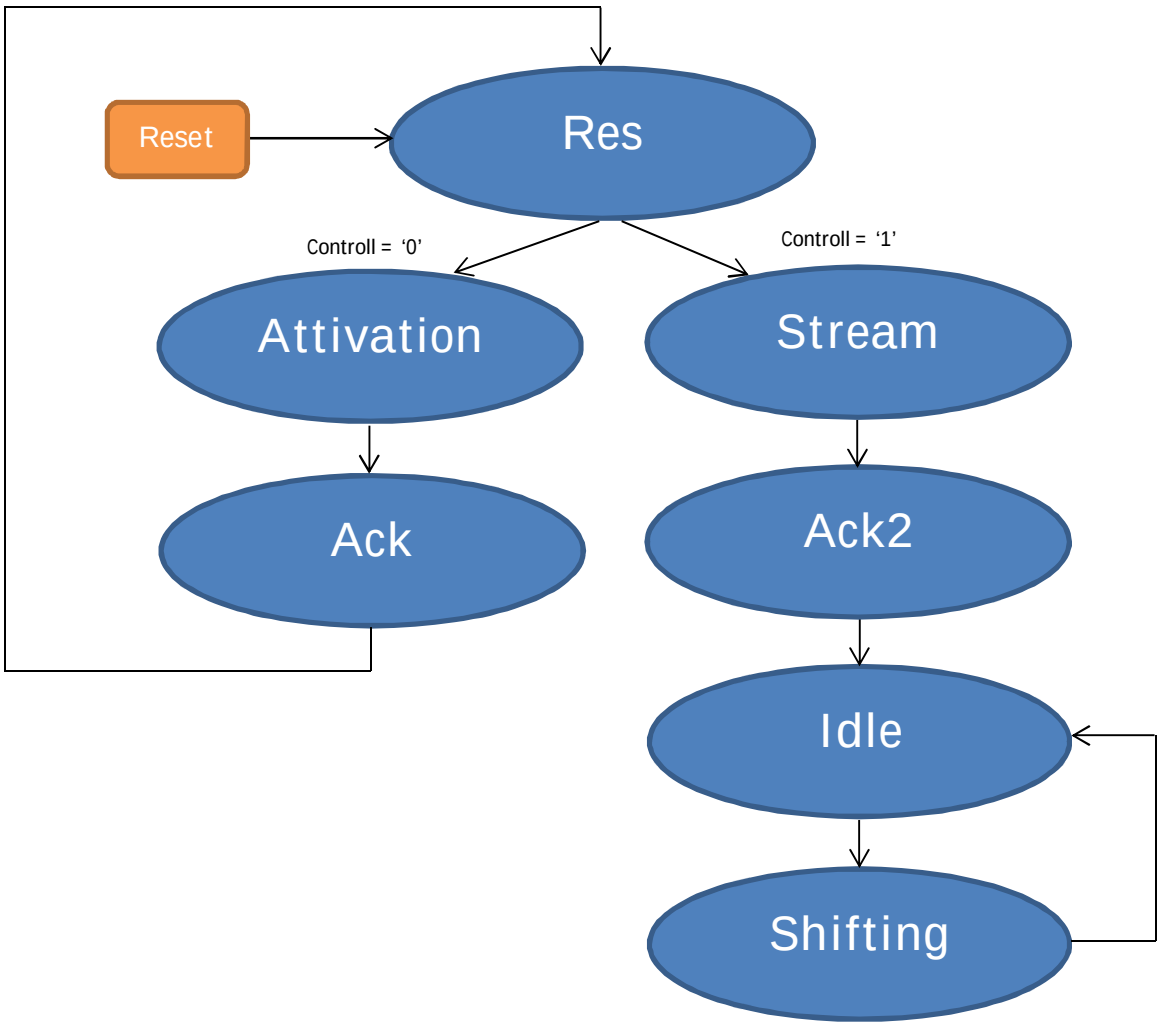


Figura 3.5: diagramma di flusso

Questi blocchi sono stati implementati nel file *mouse.vhd* che verrà riportato completamente in appendice, mentre ora saranno sottolineate e descritte le scelte più rilevanti adottate nella realizzazione di questo codice.

Quando il tasto Reset viene premuto, dopo un azzeramento dei segnali e dei registri utilizzati come memorie, si porta a zero il segnale *controll* e si passa allo stato **Res** dove per la prima fase di inibizione della comunicazione tra mouse e scheda si porta il segnale *M_CLK_C = '1'* e si forza il segnale *M_CLK_W = '0'* per circa 110us, mediante un contatore che conteggia 11000 cicli di clock alla frequenza di 100 MHz l'uno. Una seconda fase è quella dove vengono portati il segnale *M_DATA_C = '1'* e *M_DATA_W = '0'* e qui il conteggio è tra gli 11000 cicli e i 13000 cicli, cioè tra 110us e 130us. La terza fase è di rilasciare il segnale *M_CLK_W = '1'* cosicchè il mouse percepisca che da quel momento debba generare lui il clock per la ricezione della parola inviata dalla scheda. Infine l'ultima fase, una volta usciti dal conteggio, è quella di passare allo stato Attivation o Stream in base al valore del segnale *Controll*. Di seguito sono riportate le righe di codice appena descritte.

```
when Res =>-- inibizione della comunicazione per la richiesta di invio della parola
        Count <= Count + 1;
        if(Count < 11000) then
                M_CLK_C <= '1';
                M_CLK_W <= '0';
        elsif (Count < 13000) then
                M_CLK_C <= '1';
                M_CLK_W <= '0';
                M_DATA_C <= '1';
                M_DATA_W <= '0';  - - bit di start
        elsif(Count = 13000) then
                M_CLK_W <= '1';
        else
                CODE <= (OTHERS => '0');
                Count <= (OTHERS => '0');
                M_CLK_C <= '0';
                if(Controll = '0') then
                        State <= Attivation;
                else
                        State <= Stream;
                end if;
        end if;
```

Il compito dello stato **Attivation** e dello stato **Stream** è uguale, cioè di inviare una parola al mouse sincronizzandosi correttamente per cambiare il dato quando il segnale *PS2CLK_OUT* è uguale a *‘1’*. In un caso la parola è 0xFF e nell’altro è 0xF4, ed anche in questa situazione un registro svolge la funzione di contatore come si può vedere nel codice sottostante nel caso dello stato Stream.

```
when Stream => --invio di F4 per mandare il mouse in strem mode
        if (PS2CLK_OUT = '1') then
                Bit_Cnt <= Bit_Cnt + 1;
                if (Bit_Cnt >= 0 and Bit_Cnt < 2) then
                        M_DATA_W <= '0';
                elsif (Bit_Cnt = 2) then
                        M_DATA_W <= '1';
                elsif (Bit_Cnt = 3) then
                        M_DATA_W <= '0';
                elsif (Bit_Cnt >= 4 and Bit_Cnt < 8) then
                        M_DATA_W <= '1';
                elsif (Bit_Cnt = 8) then
                        M_DATA_W <= '0';
                else
                        State <= Ack2;
                        Bit_Cnt <= "000000";
                        M_DATA_C <= '0';
                end if;
        end if;
```

Identica cosa per quanto riguarda gli stati **Ack** e **Ack2** che hanno il compito di ricevere in appositi registri, attraverso un continuo spostamento verso destra (*shift right*), e di verificare le parole che il mouse invia alla scheda durante le fasi di attivazione. Di seguito è riportato il codice del caso Ack, il quale è più completo perchè riceve sia un bit di ricezione e 0xFA, parola di attivazione come nel caso Ack2 del comando appena eseguito, sia 0xAA e 0x00, le quali indicano lo stato in cui si trova il mouse.

```
when Ack =>
        if (PS2CLK_OUT = '1')   then
                Bit_Cnt <= Bit_Cnt + 1;
                if (Bit_Cnt < 33) then
                        if (Bit_Cnt < 10) then
                                Data_Reg3 <= M_DATA_R & Data_Reg3(7 downto 1);
                        end if;
                        if (Bit_Cnt >= 13 and Bit_Cnt < 21) then
                                Data_Reg4 <= M_DATA_R & Data_Reg4(7 downto 1);
                        end if;
                        if (Bit_Cnt >= 24 and Bit_Cnt < 32) then
                                Data_Reg5 <= M_DATA_R & Data_Reg5(7 downto 1);
                        end if;
                else
                        Bit_Cnt <= (OTHERS => '0');
                        State <= Res;  -- ricezione di ack, FA, AA, 00 (34 bit)
                        controll <= '1';
                        if(Data_Reg3 = "11111010" and Data_Reg4 = "10101010" and
                        Data_Reg5 = "00000000") then  --controllo ricezione
                                ERR_ATT <= '0';
                        else
                                ERR_ATT <= '1';
                        end if;
                end if;
        end if;
```

Dopo queste fasi di attivazione si arriva nello stato **Idle** dove anche qui c'è un azzeramento del registro contatore e dei segnali di controllo (Parity, M_DATA_C e M_CLK_C) e si attende il bit M_DATA = '0' che indica l'inizio della comunicazione del mouse riguardo al movimento o allo stato dei pulsanti, così da passare nello stato **Shifting** dove avviene la gestione di tali dati immagazzinandoli in opportuni registri, controllando il loro bit di parità dispari e impostando il segnale *CODE* che andrà in uscita verso il blocco di visualizzazione con i dati importanti per l'avanzamento del progetto. All'inizio della pagina seguente ne sarà riportato un pezzo di quest'ultimo stato evidenziando le fasi che si ripeteranno per tutte e tre le parole.

```
when Shifting =>
        if (PS2CLK_OUT = '1')   then
                if Bit_Cnt < 9 then
                        Bit_Cnt <= Bit_Cnt + 1;
                        Data_Reg0 <= M_DATA_R & Data_Reg0(8 downto 1);
                        Parity <= Parity xor M_DATA_R;
                end if;
                if (Bit_Cnt >= 9 and Bit_Cnt < 11) then
                        if (Bit_Cnt = 9) then
                                if(not Parity = M_DATA_R) then
                                        ERR <= '0';
                                else
                                        ERR <= '1';
                                end if;
                        Parity <= '0';--dopo il controllo azzero il bit di Parità
                        end if;
                        Bit_Cnt <= Bit_Cnt + 1;
                end if;
. . . . . stesso procedimento per la seconda e la terza parola . . . . . .
                if (Bit_Cnt = 32) then
                        CODE <= '1' & Data_Reg0(8 downto 0) &
                                Data_Reg1(8 downto 0) & Data_reg2(8 downto 0);
                        State <= Idle;
                        Bit_Cnt <= (OTHERS => '0');
                end if;
        end if;
```

Adesso che tutti gli stati sono implementati, in *figura 3.6* si può vedere lo schema a blocchi del file *Top2.vhd* che prende al suo interno sia la fase di sincronizzazione sia la fase di reset/acquisizione.

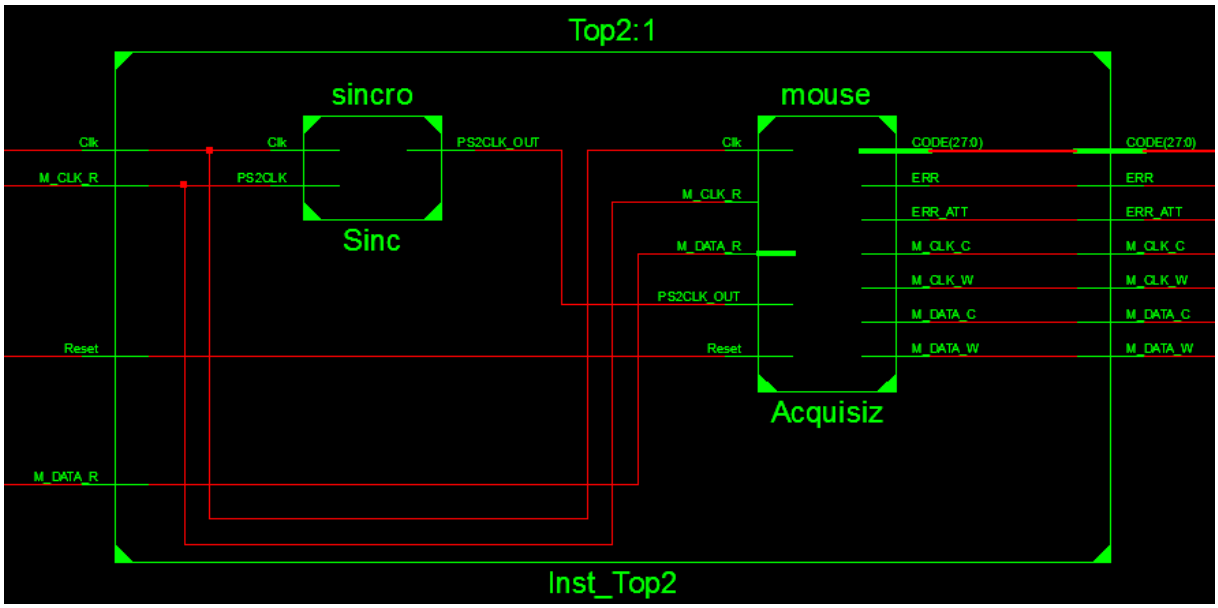


Figura 3.6: schema a blocchi del file Top2.vhd

3.1.4 Visualizzazione

L'ultimo blocco è quello che riguarda la visualizzazione sui dispaly e sui led del movimento e dello stato dei pulsanti quando vengono premuti.
In primo luogo si considerano e si mettono in segnali di appoggio le informazioni ricevute dal segnale CODE, proveniente dalla gestione dell'acquisizione con i bit relativi allo stato dei pulsanti e al movimento del mouse. In secondo luogo si elabora una strategia per far si che ci sia una distinzione tra tutte le otto possibili direzioni: alto, basso, destra, sinistra, alto–destra, alto–sinistra, basso–destra e basso–sinistra. Bisogna ricordare e prestare molta attenzione che il valore del movimento sulle coordinate X e Y è espresso in complemento a 2, quindi nei casi in cui un numero sia positivo e l'altro negativo, bisogna invertire tale valore negativo bit per bit affinchè sia possibile eseguire un confronto paritario su entrambe le coordinate per una corretta visualizzazione. Si è fatto ciò tramite un processo, come riportato di seguito, dove, appunto, si può notare che nelle direzioni con bit discordi si compia l'inversione dei bit del messaggio utile su una di esse, ed ora tale valore andrebbe sommato di un'unita, ma non lo si è fatto per non complicare ulteriormente il codice in quanto l'errore che si compie è davvero trascurabile.

```
process(Move, X, Y)
begin
if(Move = "01") then  --alto sinistra
       X_direction <= not X(7) & not X(6) & not X(5) & not X(4) & not X(3) &
                      not X(2) & not X(1) & not X(0);
       Y_direction <= Y(7 downto 0);
elsif(Move = "10") then  --alto sinistra
       Y_direction <= not Y(7) & not Y(6) & not Y(5) & not Y(4) & not Y(3) &
                      not Y(2) & not Y(1) & not Y(0);
       X_direction <= X(7 downto 0);
else
       X_direction <= X(7 downto 0);
       Y_direction <= Y(7 downto 0);
end if;
end process;
```

A questo punto si è implementato un altro processo dove si valuta se i bit ricevuti sono relativi allo stato di reset o ad una acquisizione, e si fa in modo che i display e i led vengano azzerati e inizializzati oppure illuminati secondo delle soglie di visualizzazione impostate. Queste soglie sono caratterizzate in due confronti per ognuno dei quattro stati che il mouse invia, e all'interno di ogni confronto avviene un

ulteriore riscontro sulle due coordinate per decidere se il movimento si trova in una situazione intermedia tra le due direzioni o il contrario. Tali confronti e soglie si possono osservare nel codice riportato di seguito dove è citato solo un caso, mentre il codice completo si trova anch'esso in appendice nel file *Visual.vhd*.

```
process(Stato,Reg_0,Speed_max,Move,X_direction, Y_direction)
begin
if (Stato = '0') then
        Anodi <= "0000";  -- attivazione display 0-1-2-3
        Catodi <= "11111101";  -- display solo con il led centrale acceso
        Led <= "0000";
else
        Led <= Reg_0(2 downto 0) & Speed_max;
        Anodi <= "0000";  -- attivazione display 0-1-2-3
        if(Move = "00") then  -- alto-destra
                if (X_direction = 0 and Y_direction = 0) then
                        Catodi <= "11111111";
                elsif(X_direction > Y_direction) then
                        if(X_direction < 60) then
                                if(X_direction - Y_direction < 10) then
                                        Catodi <= "00111111";  --a-b attivi
                                else    -- destra
                                        Catodi <= "10011111";  -- b-c attivi
                                end if;
                        else
                                if(X_direction - Y_direction < 30) then
                                        Catodi <= "00111111";  --a-b attivi
                                else    -- destra
                                        Catodi <= "10011111";  -- b-c attivi
                                end if;
                        end if;
                else
                        if(Y_direction < 60) then
                                if(Y_direction - X_direction < 10) then
                                        Catodi <= "00111111";  --a-b attivi
                                else    -- alto
                                        Catodi <= "01111111";  -- a attivo
                                end if;
                        else
                                if(Y_direction - X_direction < 30) then
                                        Catodi <= "00111111";  --a-b attivi
                                else    -- alto
                                        Catodi <= "01111111";  -- a attivo
                                end if;
                        end if;
                end if;
  . . . . . il codice continua con gli altri casi . . . . . .
```

4) Test e conclusioni

4.1 Verifica del codice VHDL

Uno strumento molto utile per verificare se il codice che si sta scrivendo compie effettivamente le operazioni volute, sono i testbench, cioè dei file che simulano il comportamento dell'oggetto in questione attraverso il cambiamento di certi segnali a cura del programmatore. In precedenza si è riportato il grafico relativo al blocco di sincronizzazione (*figura 3.4*). Lo stesso procedimento si è intrapreso per la simulazione del file *Top2.vhd* e del file *Top.vhd* , e grazie ad alcune segnalazioni di errori si è corretto e migliorato il loro codice. Queste simulazioni sono di tipo *Behavioral*, cioè comportamentali e quindi non tengono conto di alcuni fattori importanti come l'implementazione su scheda ed i relativi ritardi reali dei segnali.
Se, invece, si vuole una simulazione più precisa esiste la cosiddetta *Post – Route* , la quale contiene tutti i blocchetti logici che verranno utilizzati nella FPGA, oltre che tutti i ritardi. Questa operazione si può eseguire solo sul progetto che contiene tutti i file, quindi in questo caso è stata eseguita sul blocco Top.
Altri strumenti di controllo sono il sintetizzatore, che indica se ci sono errori logici o se qualche parte di registri e/o segnali non vengono utilizzati nella realtà, ed il blocco di implementazione dove viene verificato se il codice potrà essere implementato correttamente sulla scheda.
Una volta concluse tutte le operazioni di verifica si può passare alla fase successiva, cioè quella di generare il *bitstream* per poi programmare la scheda attraverso il tool grafico *ISE Impact*. Solo durante quest'ultima prova ci si può rendere conto se effettivamente tutte le scelte decise in fase di progettazione e sviluppo si sono rivelate corrette, altrimenti, se il comportamento non è quello voluto, si dovrà modificare il codice o parte del progetto fino ad arrivare al risultato aspirato.

4.2 Conclusioni e possibili sviluppi futuri

Per quanto riguarda le risorse occupate, il progetto ne occupa una minima parte: 111 registri/Flip – Flops a disposizione (1%), 219 LUTs (Look – Up – Table) totali (2%), 32 Multiplexer MUXCys (1%) e 2 BUFTs (6%).

Durante questo progetto di tesi si sono approfondite la conoscenza del linguaggio VHDL e le abilità con gli strumenti di sviluppo per riuscire a completare con successo l'obiettivo prefissato. In particolare la realizzazione di un progetto diviso in più blocchi uniti tra loro, la creazione di testbench opportuni per una verifica del codice prodotto e si è preso maggiore coscienza delle enormi potenzialità che offrono le FPGA, con tutte le possibili periferiche che riescono a gestire.

Tramite questa esperienza, si ha avuto l'occasione di studiare approfonditamente il metodo in cui una periferica con porta USB, soprattutto riguardo al mouse, si interfaccia con un microprocessore e comprendere i possibili modi di funzionamento del dispositivo ausiliario per far si che le fasi di attivazione e gestione dati avvengano correttamente.

Nello svolgimento del lavoro ci si è imbattuti in difficoltà di carattere prima progettuale e poi implementativo riguardo le scelte da compiere per la realizzazione di un lavoro corretto e completo, ma sono state sempre superate, anche grazie all'aiuto del docente per quelle più complicate.

Un ulteriore sviluppo di questo progetto potrebbe essere quello di riuscire ad interfacciarsi con un monitor esterno tramite l'uscita VGA e quindi visualizzare il movimento del mouse attraverso sia i display e i led della scheda sia ad un puntatore presente sul monitor.

Appendice A

Sincro.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sincro is
    Port ( Clk : in  STD_LOGIC;
           PS2CLK : in  STD_LOGIC;
           PS2CLK_OUT : out  STD_LOGIC);
end sincro;

architecture Behavioral of sincro is
signal A, B : std_logic := '0'; --segnali interni
begin
 process(Clk)
begin
 if (rising_edge(Clk)) then
  A <= PS2CLK;
  B <= A;
 end if;
 end process;
 PS2CLK_OUT <= (not A) and B;
end Behavioral;
```

Mouse.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mouse is
    Port ( Clk : in STD_LOGIC;
               PS2CLK_OUT : in  STD_LOGIC;
               M_CLK_R : IN std_logic;
               M_CLK_C : OUT std_logic;
               M_CLK_W : OUT std_logic;
               M_DATA_R : in  STD_LOGIC ;
               M_DATA_C : out  STD_LOGIC ;
               M_DATA_W : out  STD_LOGIC ;
               Reset : in  STD_LOGIC;
               ERR : out  STD_LOGIC := '0';
```

```
              ERR_ATT : out STD_LOGIC := '0';
              CODE : out  STD_LOGIC_VECTOR (27 downto 0)); --codice di uscita per comandare i
                                                --  display e i led con le informazioni acquisite
end mouse;

architecture Behavioral of mouse is
signal Parity : std_logic := '0';  --bit di Parità
signal Bit_Cnt : std_logic_vector(5 downto 0):= "000000";
signal Count : std_logic_vector(13 downto 0) := "00000000000000";
signal Data_Reg0 : std_logic_vector(8 downto 0) := "000000000";
signal Data_Reg1 : std_logic_vector(8 downto 0) := "000000000";
signal Data_Reg2 : std_logic_vector(8 downto 0) := "000000000";
signal Data_Reg3 : std_logic_vector(7 downto 0) := "00000000";
signal Data_Reg4 : std_logic_vector(7 downto 0) := "00000000";
signal Data_Reg5 : std_logic_vector(7 downto 0) := "00000000";
signal Controll : std_logic := '0';
Type State_t is (Idle, Shifting, Attivation, Res, Ack, Stream, Ack2);
signal State : State_t;
begin
process(Clk, PS2CLK_OUT, Reset, State)
 begin
  if rising_edge(clk) then
    if Reset = '1' then    -- azzeramento dei vari registri e segnali
      Parity <= '0';
       Bit_Cnt <= (OTHERS => '0');
       Count <= (OTHERS => '0');
       Data_Reg0 <= (OTHERS => '0');
       Data_Reg1 <= (OTHERS => '0');
       Data_Reg2 <= (OTHERS => '0');
       Data_Reg3 <= (OTHERS => '0');
       ERR_ATT <= '0';
       State <= Res;
       Controll <= '0';
    else

     case State is
        when Res =>   --  inibizione della comunicazione per la richiesta di invio della parola
         Count <= Count + 1;
          if(Count < 11000) then
            M_CLK_C <= '1';
            M_CLK_W <= '0';
          elsif (Count < 13000) then
            M_CLK_C <= '1';
            M_CLK_W <= '0';
            M_DATA_C <= '1';
            M_DATA_W <= '0';  -- bit di start
          elsif(Count = 13000) then
            M_CLK_W <= '1';
          else
            CODE <= (OTHERS => '0');
            Count <= (OTHERS => '0');
            M_CLK_C <= '0';
```

```
      if(Controll = '0') then
        State <= Attivation;
      else
        State <= Stream;
      end if;
    end if;

when Attivation =>   --invio di FF per check-up del mouse
  if (PS2CLK_OUT = '1') then
        Bit_Cnt <= Bit_Cnt + 1;
        if (Bit_Cnt >= 0 and Bit_Cnt < 9) then
             M_DATA_W <= '1';
        else
             State <= Ack;
             Bit_Cnt <= "000000";
             M_DATA_C <= '0';
       end if;
  end if;

 when Ack =>   -- inizio ricezione delle parole di controllo e attivazione del mouse
  if (PS2CLK_OUT = '1')   then
      Bit_Cnt <= Bit_Cnt + 1;
         if (Bit_Cnt < 33) then
           if (Bit_Cnt < 10) then
              Data_Reg3 <= M_DATA_R & Data_Reg3(7 downto 1);  --shift right
           end if;
           if (Bit_Cnt >= 13 and Bit_Cnt < 21) then
              Data_Reg4 <= M_DATA_R & Data_Reg4(7 downto 1);  --shift right
           end if;
           if (Bit_Cnt >= 24 and Bit_Cnt < 32) then
              Data_Reg5 <= M_DATA_R & Data_Reg5(7 downto 1);  --shift right
           end if;
         else
           Bit_Cnt <= (OTHERS => '0');
           State <= Res;                -- ricezione di ack, FA, AA, 00 (34 bit)
           controll <= '1';
           if(Data_Reg3 = "11111010" and Data_Reg4 = "10101010" and
               Data_Reg5 = "00000000") then  --controllo ricezione
                ERR_ATT <= '0';
           else
                ERR_ATT <= '1';
           end if;
         end if;
  end if;

when Stream =>   --invio di F4 per mandare il mouse in stream mode
   if (PS2CLK_OUT = '1') then
      Bit_Cnt <= Bit_Cnt + 1;
       if (Bit_Cnt >= 0 and Bit_Cnt < 2) then
         M_DATA_W <= '0';
       elsif (Bit_Cnt = 2) then
         M_DATA_W <= '1';
```

```
      elsif (Bit_Cnt = 3) then
        M_DATA_W <= '0';
      elsif (Bit_Cnt >= 4 and Bit_Cnt < 8) then
         M_DATA_W <= '1';
     elsif (Bit_Cnt = 8) then
         M_DATA_W <= '0';
     else
        State <= Ack2;
        Bit_Cnt <= "000000";
        M_DATA_C <= '0';
     end if;
  end if;

when Ack2 =>     -- inizio ricezione della parola di controllo
  if (PS2CLK_OUT = '1')   then
       if (Bit_Cnt < 11) then
        if (Bit_Cnt < 10) then
            Data_Reg3 <= M_DATA_R & Data_Reg3(7 downto 1);  --shift right
        end if;
        Bit_Cnt <= Bit_Cnt + 1;
       else
        Bit_Cnt <= (OTHERS => '0');
        State <= Idle;     -- ricezione di FA
        if (Data_Reg3 = "11111010") then  --controllo ricezione FA
            ERR_ATT <= '0';
        else
            ERR_ATT <= '1';
        end if;
       end if;
  end if;

 when Idle => -- stato intermedio dove si aspetta l'inizio della comunicazione del mouse
    Parity <= '0';
   Bit_Cnt <= (OTHERS => '0');
   M_DATA_C <= '0';
   M_CLK_C <= '0';
   if (M_DATA_R = '0') then  --bit di Start
     ERR <= '0';
     State <= Shifting;
   end if;

 when Shifting =>   -- inizio acquisizione dati
   if (PS2CLK_OUT = '1')   then
     if Bit_Cnt < 9 then
       Bit_Cnt <= Bit_Cnt + 1;
       Data_Reg0 <= M_DATA_R & Data_Reg0(8 downto 1);  --shift right
       Parity <= Parity xor M_DATA_R;
     end if;
    if (Bit_Cnt >= 9 and Bit_Cnt < 11) then
       if (Bit_Cnt = 9) then
           if(not Parity = M_DATA_R) then
                ERR <= '0';
```

```
                else
                    ERR <= '1';
                end if;
                Parity <= '0';  --dopo il controllo azzero il bit di Parità
             end if;
             Bit_Cnt <= Bit_Cnt + 1;
         end if;
        if (Bit_Cnt >= 11 and Bit_Cnt < 20) then
             Bit_Cnt <= Bit_Cnt + 1;
             Data_Reg1 <= M_DATA_R & Data_Reg1(8 downto 1);  --shift right
                       Parity <= Parity xor M_DATA_R;
        end if;
        if (Bit_Cnt >= 20 and Bit_Cnt < 22) then
             if (Bit_Cnt = 20) then
                  if(not Parity = M_DATA_R) then
                      ERR <= '0';
                  else
                      ERR <= '1';
                  end if;
                     Parity <= '0';  --dopo il controllo azzero il bit di Parità
              end if;
              Bit_Cnt <= Bit_Cnt + 1;
        end if;
        if (Bit_Cnt >= 22 and Bit_Cnt < 31) then
             Bit_Cnt <= Bit_Cnt + 1;
             Data_Reg2 <= M_DATA_R & Data_Reg2(8 downto 1);  --shift right
             Parity <= Parity xor M_DATA_R;
        end if;
        if (Bit_Cnt >= 31 and Bit_Cnt < 32) then
             if (Bit_Cnt = 31) then
                  if(not Parity = M_DATA_R) then
                      ERR <= '0';
                  else
                      ERR <= '1';
                  end if;
                  Parity <= '0';  --dopo il controllo azzero il bit di Parità
              end if;
              Bit_Cnt <= Bit_Cnt + 1;
        end if;
        if (Bit_Cnt = 32) then
             CODE <= '1' & Data_Reg0(8 downto 0) & Data_Reg1(8 downto 0) &
                      Data_reg2(8 downto 0);  -- codice in uscita con i bit utili alla
                                              -- visualizzazione
             State <= Idle;
             Bit_Cnt <= (OTHERS => '0');
         end if;
     end if;

   when others =>
     State <= Idle;
   end case;
end if;
```

```
 end if;
 end process;
end Behavioral;
```

Visual.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Visual is
    Port ( CODE : in  STD_LOGIC_VECTOR (27 downto 0);
           Anodi : out  STD_LOGIC_VECTOR (3 downto 0);
           Catodi : out  STD_LOGIC_VECTOR (7 downto 0);
           Led : out  STD_LOGIC_VECTOR (3 downto 0));
end Visual;

architecture Behavioral of Visual is
signal Stato : std_logic := '0';
signal Speed_max : std_logic := '0';
signal Move : std_logic_vector(1 downto 0) := "00";
signal Reg_0 : std_logic_vector(7 downto 0) := "00000000";
signal X_direction : std_logic_vector(7 downto 0) := "00000000";
signal Y_direction : std_logic_vector(7 downto 0) := "00000000";
signal X : std_logic_vector(7 downto 0) := "00000000";
signal Y : std_logic_vector(7 downto 0) := "00000000";

begin
 Speed_max <= CODE(26) or CODE(25) ;
 Stato <= CODE(27);
 Move <= CODE(24 downto 23);
 Reg_0 <= CODE(26 downto 19);
 X <= CODE(17 downto 10);
 Y <= CODE(8 downto 1);
 process(Move, X, Y)
 begin
 if(Move = "01") then  --alto sinistra
   X_direction <= not X(7) & not X(6) & not X(5) & not X(4) & not X(3) & not X(2) &
                  not X(1) & not X(0);
   Y_direction <= Y(7 downto 0);
 elsif(Move = "10") then  --alto sinistra
   Y_direction <= not Y(7) & not Y(6) & not Y(5) & not Y(4) & not Y(3) & not Y(2) &
                  not Y(1) & not Y(0);
   X_direction <= X(7 downto 0);
 else
   X_direction <= X(7 downto 0);
   Y_direction <= Y(7 downto 0);
 end if;
```

```
end process;


process(Stato,Reg_0,Speed_max,Move,X_direction, Y_direction)
begin
if (Stato = '0') then
  Anodi <= "0000";  -- attivazione display 0-1-2-3
  Catodi <= "11111101";  -- display solo con il led centrale acceso
  Led <= "0000";
else
  Led <= Reg_0(2 downto 0) & Speed_max;
  Anodi <= "0000";  -- attivazione display 0-1-2-3
  if(Move = "00") then       -- alto-destra
      if (X_direction = 0 and Y_direction = 0) then
          Catodi <= "11111111";
      elsif(X_direction > Y_direction) then
          if(X_direction < 60) then
             if(X_direction - Y_direction < 10) then
                 Catodi <= "00111111";  --a-b attivi
             else    -- dstra
                 Catodi <= "10011111";  -- b-c attivi
             end if;
           else
              if(X_direction - Y_direction < 30) then
                  Catodi <= "00111111";  --a-b attivi
              else    -- dstra
                  Catodi <= "10011111";  -- b-c attivi
              end if;
           end if;
      else
          if(Y_direction < 60) then
             if(Y_direction - X_direction < 10) then
                Catodi <= "00111111";  --a-b attivi
             else    -- alto
                Catodi <= "01111111";  -- a attivo
             end if;
           else
             if(Y_direction - X_direction < 30) then
                Catodi <= "00111111";  --a-b attivi
             else    -- alto
                Catodi <= "01111111";  -- a attivo
             end if;
          end if;
      end if;

  elsif(Move = "01") then  -- alto-sinistra
         if(X_direction > Y_direction) then
              if(X_direction < 60) then
                 if(X_direction - Y_direction < 10) then
                    Catodi <= "01111011";  --a-f attivi
                 else    -- sinistra
                    Catodi <= "11110011";  -- e-f attivi
```

```
            end if;
         else
            if(X_direction - Y_direction < 30) then
               Catodi <= "01111011";  --a-f attivi
            else    -- sinistra
               Catodi <= "11110011";  -- e-f attivi
            end if;
         end if;
      else
         if(Y_direction < 60) then
               if(Y_direction - X_direction < 10) then
                  Catodi <= "01111011";  --a-f attivi
               else    -- alto
                  Catodi <= "01111111";  -- a attivo
               end if;
         else
               if(Y_direction - X_direction < 30) then
                  Catodi <= "01111011";  --a-f attivi
               else    -- alto
                  Catodi <= "01111111";  -- a attivo
               end if;
         end if;
      end if;

elsif(Move = "10") then  -- basso-destra
      if(X_direction > Y_direction) then
          if(X_direction < 60) then
             if(X_direction - Y_direction < 10) then
                 Catodi <= "11001111";  --c-d attivi
             else    -- destra
                 Catodi <= "10011111";  -- b-c attivi
             end if;
          else
             if(X_direction - Y_direction < 30) then
                Catodi <= "11001111";  --c-d attivi
             else    -- destra
                Catodi <= "10011111";  -- b-c attivi
             end if;
          end if;
      else
          if(Y_direction < 60) then
             if(Y_direction - X_direction < 10) then
                Catodi <= "11001111";  --c-d attivi
             else    -- basso
                Catodi <= "11101111";  -- d attivo
             end if;
          else
             if(Y_direction - X_direction < 30) then
                 Catodi <= "11001111";  --c-d attivi
             else    -- basso
                Catodi <= "11101111";  -- d attivo
             end if;
```

```
                end if;
             end if;

        else                    -- basso-sinistra
            if(X_direction > Y_direction) then
               if(X_direction < 60) then
                   if(X_direction - Y_direction < 10) then
                       Catodi <= "11100111";  --d-e attivi
                    else    -- basso
                       Catodi <= "11101111";  -- d attivi
                    end if;
               else
                   if(X_direction - Y_direction < 30) then
                       Catodi <= "11100111";  --d-e attivi
                   else    -- basso
                       Catodi <= "11101111";  -- d attivi
                    end if;
                end if;
             else
               if(Y_direction < 60) then
                   if(Y_direction - X_direction < 10) then
                      Catodi <= "11100111";  --d-e attivi
                   else    -- sinistra
                      Catodi <= "11110011";  -- e-f attivo
                   end if;
                else
                   if(Y_direction - X_direction < 30) then
                      Catodi <= "11100111";  --d-e attivi
                   else    -- sinistra
                      Catodi <= "11110011";  -- e-f attivo
                   end if;
                end if;
             end if;
          end if;
       end if;
   end process;
end Behavioral;
```

Top2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Top2 is
   Port ( Clk : in  STD_LOGIC;
          Reset : in  STD_LOGIC;
          M_CLK_R : IN std_logic;
          M_CLK_C : OUT std_logic;
```

```
          M_CLK_W : OUT std_logic;
          M_DATA_R : in  STD_LOGIC ;
          M_DATA_C : out  STD_LOGIC ;
          M_DATA_W : out  STD_LOGIC ;
          ERR : out  STD_LOGIC;
          ERR_ATT : out STD_LOGIC;
          CODE : out  STD_LOGIC_VECTOR (27 downto 0));
end Top2;

architecture Behavioral of Top2 is

COMPONENT sincro
   Port ( PS2CLK : in  STD_LOGIC;
          Clk : in  STD_LOGIC;
          PS2CLK_OUT : out  STD_LOGIC);
       END COMPONENT;
COMPONENT mouse
   Port ( Clk : in STD_LOGIC;
          M_CLK_R : IN std_logic;
          M_CLK_C : OUT std_logic;
          M_CLK_W : OUT std_logic;
          M_DATA_R : in  STD_LOGIC ;
          M_DATA_C : out  STD_LOGIC ;
          M_DATA_W : out  STD_LOGIC ;
          Reset : in  STD_LOGIC;   --System Reset
          PS2CLK_OUT : in STD_LOGIC;   --Clock sincronizzato
          ERR : out  STD_LOGIC;
          ERR_ATT : out STD_LOGIC;
          CODE : out  STD_LOGIC_VECTOR (27 downto 0));
 END COMPONENT;

signal PS2CLK_OUT : std_logic;
begin
Sinc : sincro
  PORT MAP( PS2CLK => M_CLK_R,
            Clk => Clk,
            PS2CLK_OUT => PS2CLK_OUT);

Acquisiz : mouse
  PORT MAP( Clk => Clk,
            M_CLK_R => M_CLK_R,
            M_CLK_C => M_CLK_C,
            M_CLK_W => M_CLK_W,
            M_DATA_R => M_DATA_R,
            M_DATA_C => M_DATA_C,
            M_DATA_W => M_DATA_W,
            Reset => Reset,
            PS2CLK_OUT => PS2CLK_OUT,
            ERR => ERR,
            ERR_ATT => ERR_ATT,
            CODE => CODE);
end Behavioral;
```

Top.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Top is
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           M_CLK : inout  STD_LOGIC;
           M_DATA : inout  STD_LOGIC;
           ERR : out  STD_LOGIC;
           ERR_ATT : out STD_LOGIC;
           Anodi : out  STD_LOGIC_VECTOR (3 downto 0);
           Catodi : out STD_LOGIC_VECTOR (7 downto 0);
           Led : out STD_LOGIC_VECTOR (3 downto 0));
end Top;

architecture Behavioral of Top is
COMPONENT Top2
    PORT(
           Clk : IN std_logic;
           Reset : IN std_logic;
           M_CLK_R : IN std_logic;
           M_CLK_C : OUT std_logic;
           M_CLK_W : OUT std_logic;
           M_DATA_R : IN std_logic;
           M_DATA_C : OUT std_logic;
           M_DATA_W : OUT std_logic;
           ERR : OUT std_logic;
           ERR_ATT : OUT std_logic;
           CODE : OUT std_logic_vector(27 downto 0));
END COMPONENT;

signal M_DATA_W : std_logic := '0';
signal M_DATA_R : std_logic := '0';
signal M_DATA_C : std_logic := '0';
signal M_CLK_W : std_logic := '0';
signal M_CLK_R : std_logic := '0';
signal M_CLK_C : std_logic := '0';
signal CODE : std_logic_vector (27 downto 0);

COMPONENT Visual
    PORT(
           CODE : IN std_logic_vector(27 downto 0);
           Anodi : OUT std_logic_vector(3 downto 0);
           Catodi : OUT std_logic_vector(7 downto 0);
           Led : OUT std_logic_vector(3 downto 0));
END COMPONENT;
```

```
begin
Inst_Top2: Top2 PORT MAP(
        Clk => Clk,
        Reset => Reset,
        M_CLK_R => M_CLK_R,
        M_CLK_C => M_CLK_C,
        M_CLK_W => M_CLK_W,
        M_DATA_R => M_DATA_R,
        M_DATA_C => M_DATA_C,
        M_DATA_W => M_DATA_W,
        ERR => ERR,
        ERR_ATT => ERR_ATT,
        CODE => CODE);

M_DATA <= M_DATA_W when M_DATA_C = '1' else 'Z';
M_DATA_R <= M_DATA;
M_CLK <= M_CLK_W when M_CLK_C = '1' else 'Z';
M_CLK_R <= M_CLK;

Visualization: Visual PORT MAP(
        CODE => CODE,
        Anodi => Anodi,
        Catodi => Catodi,
        Led => Led );
end Behavioral;
```

Bibliografia e siti Web consultati

1. M.M. Manoe C.R. Kime, “Reti Logiche”, 4a Ed., Prentice Hall, 2008

2. Dispense del corso “Elettronica dei Sistemi Digitali”, Corso di Laurea di Ingegneria Elettronica, Università di Padova, A.A.2012/2013

3. http://www.digilentinc.com/nexys3/

4. http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf

5. http://www.computer-engineering.org/ps2protocol/

6. http://www.computer-engineering.org/ps2mouse/

7. http://it.wikipedia.org/wiki/VHDL

8. http://it.wikipedia.org/wiki/Field_Programmable_Gate_Array

9. http://www.fpgacentral.com/pld-types/mpga-mask-programmable-gate-array

10. http://it.wikipedia.org/wiki/Programmable_Logic_Device