ANNO ACCADEMICO 2010/2011

SINCRONIZZATORE UWB-IR

Matteo Molino 579104

29/11/2011

Sommario

Questo documento descrive l'implementazione su FPGA della Xilinx di un circuito per la sincronizzazione del ricevitore di una trasmissione UWB-IR.

Si è scelto l'utilizzo di un FPGA per l'implementazione del prototipo del sincronizzatore per la facilità di riprogrammazione e la versatilità dell' FPGA.

Indice

1 Introduzione

Il circuito descritto nelle pagine di questa tesi è stato sviluppato da dei ricercatori dell'università di Padova per aderire al progetto WISE-WAY terminato qualche mese fa. L'acronimo WISE WAY significa "Wireless SEnsor networks for city-Wide Ambient Intelligence", ovvero "Sensori Wireless per Reti Cittadine Intelligenti di Controllo Ambientale". Lo scopo è quello di creare reti di sensori di controllo a basso consumo energetico.

Il basso consumo energetico di un sensore in una rete urbana è fondamentale perché generalmente tale sensore è indipendente dalla rete elettrica in quanto utilizza micro batterie con una risorsa energetica limitata. Ridurre il consumo di un nodo sensore, inoltre, migliora il dispositivo che risulta avere dimensioni inferiori ed un'autonomia maggiore .

Il sensore, al suo interno, si compone di varie sezioni con ruoli diversi, tra queste la più critica dal punto di vista energetico, ovvero con il consumo maggiore, è il front-end di comunicazione per questo motivo il presente lavoro si incentra sulla costruzione di un circuito di ricezione a basso consumo energetico adatto all'implementazione su un nodo sensore.

1.1 Struttura della tesi

Questo elaborato si compone di sette sezioni organizzate nel seguente ordine:

1. Nel primo capitolo si forniscono le basi riguardo al tipo di comunicazione scelta per il sensore ovvero la comunicazione UWB-IR.
2. Nel secondo capitolo si presenta il supporto su cui viene realizzato il circuito ovvero l'FPGA Xilinx e le tecniche di progettazione.
3. Nel terzo capitolo si introduce il sistema descrivendone la struttura generale e la struttura dello *stream* di bit utilizzato.
4. Nel quarto capitolo si descrive nello specifico l'interfaccia del circuito con l'ingresso.
5. Nel quinto capitolo si descrive il sistema di controllo.
6. Nel sesto capitolo si analizzano le prestazioni del circuito.
7. Nel settimo capitolo si conclude la trattazione descrivendone i risultati.

2 Conoscenze preliminari.

2.1 Ultra Wide Band.

In telecomunicazioni con il termine *ultra-wide-band* (banda ultra larga) o UWB si intende una tecnica di trasmissione di segnali mediante impulsi in radiofrequenza.

La caratteristica principale di una trasmissione UWB è che la durata dell'impulso di trasmissione è molto ridotta, ciò comporta che la banda occupata dal segnale sia molto grande, da questo deriva l'acronimo UWB.

Come si può vedere in figura 1 la differenza principale tra le trasmissioni a banda continua ed impulsive sta nell'occupazione della banda infatti possiamo vedere la differenza della banda occupata in una trasmissione impulsiva (figura 1 a) e in una trasmissione FSK(figura 1b).

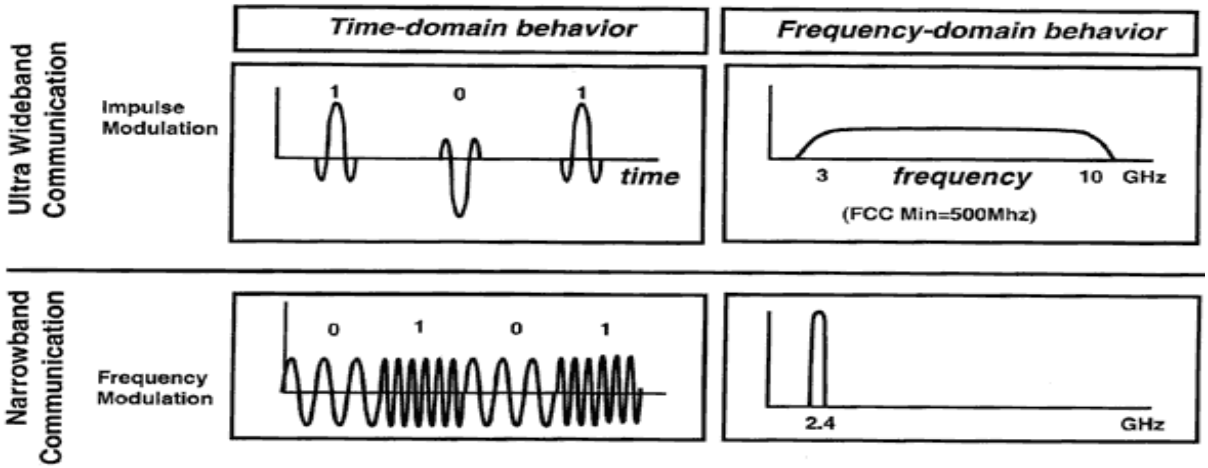

Fig. 1 a) Impulsive UWB (sopra); b) Narrowband (sotto) (1)

Un'altra differenza significativa tra le trasmissioni classiche a banda continua e le trasmissioni UWB è che i sistemi tradizionali trasmettono informazione variando ampiezza, frequenza o fase di una portante, solitamente sinusoidale, mentre le trasmissioni UWB trasmettono informazione generando energia attraverso la trasmissione della portante in istanti specifici.

Il vantaggio di utilizzare questo ultimo tipo di trasmissione consiste nel fatto che, utilizzare un impulso molto potente per un tempo molto breve, rende il segnale trasmesso quasi immune ai disturbi. La brevità del segnale comporta inoltre una diminuzione di eventuali azioni di

disturbo verso altre apparecchiature che trasmettono sulla stessa banda nonché la realizzazione di dispositivi dal basso consumo energetico.

Le trasmissioni UWB, generalmente, sono utilizzate per trasmissioni brevi e poco veloci, si adattano quindi particolarmente o in modo specifico all'implementazione di sensori per la raccolta dati.

D'altra parte il segnale trasmesso, essendo distribuito su una gamma di frequenze molto ampia, fa si che la *power spectral density* (densità spettrale di potenza) sia molto bassa al punto da poterla confondere col rumore del canale rendendo così difficile sincronizzare le trasmissioni.

L'UWB è nata dalle cosiddette trasmissioni impulsive la FCC (2) (*Federal Communications Commission*) la definisce infatti una trasmissione il cui impulso abbia una banda di almeno 500MHz oppure una banda larga almeno il 20% della frequenza centrale della portante.

La FCC ha autorizzato inoltre l'uso senza licenza del *range* di frequenze che va da 3,1 a 10,6 GHz che risulta libero per trasmissioni UWB a patto che la *power spectral density* del segnale trasmesso sia inferiore a -41,3 dBm/MHz (Fig. 2) all'interno di questa banda e -75 dBm/MHz al di fuori in modo da non interferire tra le varie trasmissioni.

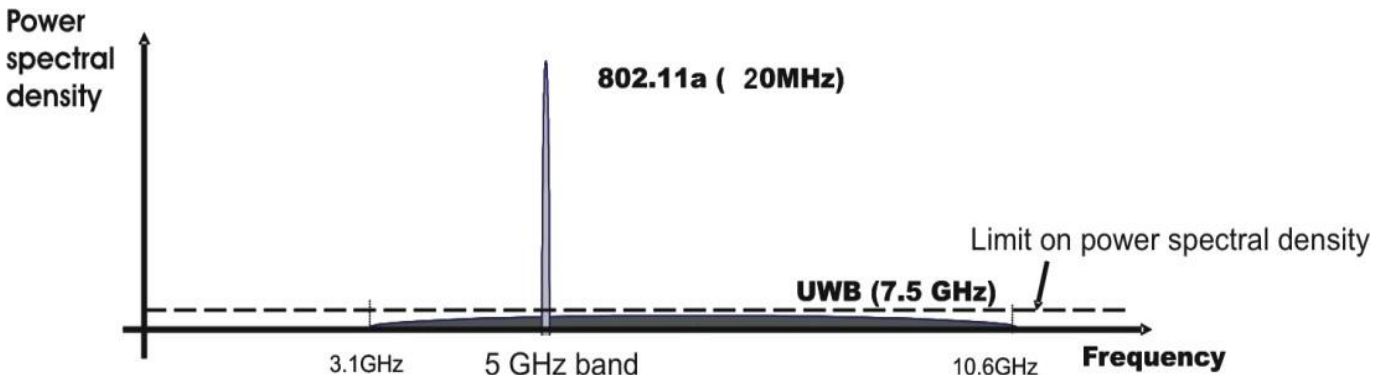

Fig. 2 limite alla densità spettrale di potenza (3)

3 FPGA

3.1 Introduzione agli FPGA.

FPGA è un acronimo usato per *Field Programmable Gate Array* (4), ovvero campo di matrici di porte programmabili, si tratta di dispositivi elettronici la cui struttura e di conseguenza la relativa funzione è programmabile attraverso un software.

Tali dispositivi rappresentano, proprio per questa loro caratteristica, un buon compromesso tra i dispositivi ASIC (*application specific integrated circuit*) e i componenti con architettura PAL (*programmable array logic).*

In particolare gli FPGA, utilizzati per implementare circuiti ad applicazione specifica presentano dei vantaggi sugli ASIC in quanto la funzionalità del componente viene implementata direttamente dall'utilizzatore tramite dei linguaggi di descrizione hardware come *VHDL* o *Verilog*, quindi il produttore può produrre il circuito integrato su più larga scala e a basso prezzo.

La programmazione diretta dell'FPGA da parte dell'utente finale consente la riduzione sia dei tempi di progettazione di un circuito per applicazione specifica che dei tempi di verifica mediante simulazione. L'impiego dell' FPGA permette infine all'utente di apportare delle correzioni al circuito vero e proprio, o delle modifiche allo stesso, semplicemente sovrascrivendo la programmazione del dispositivo, operazione che nella maggior parte dei casi, può essere fatta velocemente ed in qualsiasi momento.

Produrre dispositivi su larga scala tuttavia può rendere sconveniente l'uso del FPGA in quanto il prezzo *unitario* di tale dispositivo è molto superiore a quello di un dispositivo ASIC che a sua volta presenta lo svantaggio di avere maggiori tempi e costi di progettazione.

3.2 Storia degli FPGA.

Gli FPGA nascono da un'evoluzione delle PROM, circuiti basati su una ROM programmabile con una struttura di porte logiche AND e OR che, opportunamente programmate forniscono in uscita tutti i tipi di funzione desiderata, e dallo sviluppo dei PLD ovvero circuiti che permettono di realizzare semplici funzioni combinatorie.

I primi circuiti programmabili potevano essere scritti solo una volta e la loro programmazione poteva avvenire in due modi: tramite fusibile, annullando fisicamente il collegamento a bassa resistenza applicando un impulso elevato di corrente elettrica oppure tramite antifusibile

trasformando i collegamenti di silicio amorfo (isolante) in silicio policristallino (conduttore) applicando un potente impulso di tensione.

Tali sistemi vennero quindi sostituiti da dispositivi EPROM (a cancellazione a raggi UV) e EEPROM (a cancellazione elettrica con l'aggiunta quindi di un transistor di controllo) che presentavano la caratteristica di essere riprogrammabili più volte.

Nella figura 3 possiamo vedere la struttura base della tecnologia EPROM: il Transistor a Gate flottante" . Il funzionamento di questo dispositivo avviene attraverso un accumulo di elettroni nel Gate flottante che funge da controllo modificando la soglia del transistor in modo che rimanga spento qualunque siano le tensioni presenti nel Gate. Per programmare tale dispositivo vengono pertanto comandati i *gate* flottanti dei vari transistor. Rimuovendo le cariche tramite raggi UV(EPROM) o per via elettrica(EEPROM) è possibile quindi cancellare la programmazione del dispositivo.

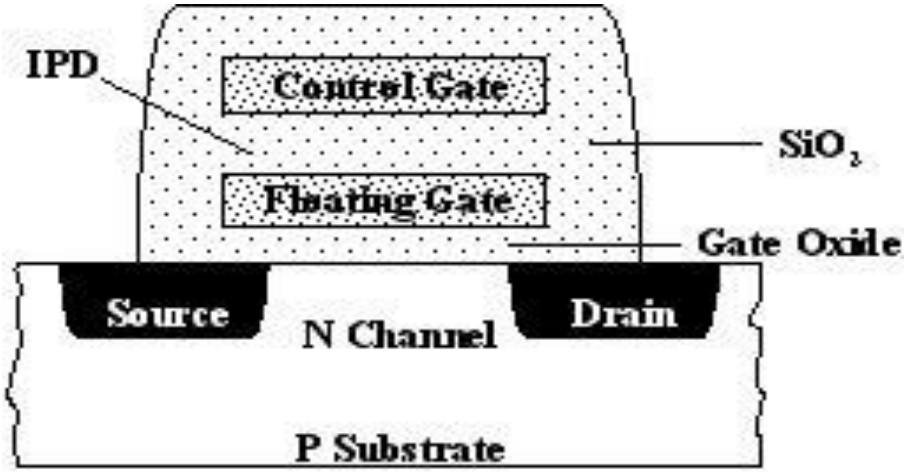

Fig. 3 (5)

Un altro metodo per realizzare dispositivi programmabili è attraverso una struttura con SRAM. A seconda del bit scritto nella SRAM il MOSFET è acceso o spento.

Un dispositivo di questo tipo va supportato con una memoria flash che lo riprogrammi ad ogni accensione della scheda in quanto, essendo la SRAM una memoria volatile, il dato vi permane solo fino a quando persiste l'alimentazione.

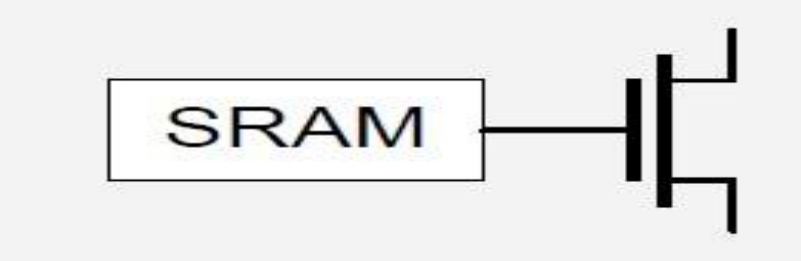

Fig. 4 (5)

Normalmente gli FPGA sono realizzati con quest'ultimo metodo oppure con sistema ad antifusibile.

3.3 Struttura degli FPGA

Gli FPGA sono componenti realizzati con tecnologia CMOS solitamente con memoria di configurazione SRAM. Sono strutturati in blocchi regolari. Ciascuno di questi blocchi si collega ad un altro attraverso delle connessioni programmabili (figura 5).

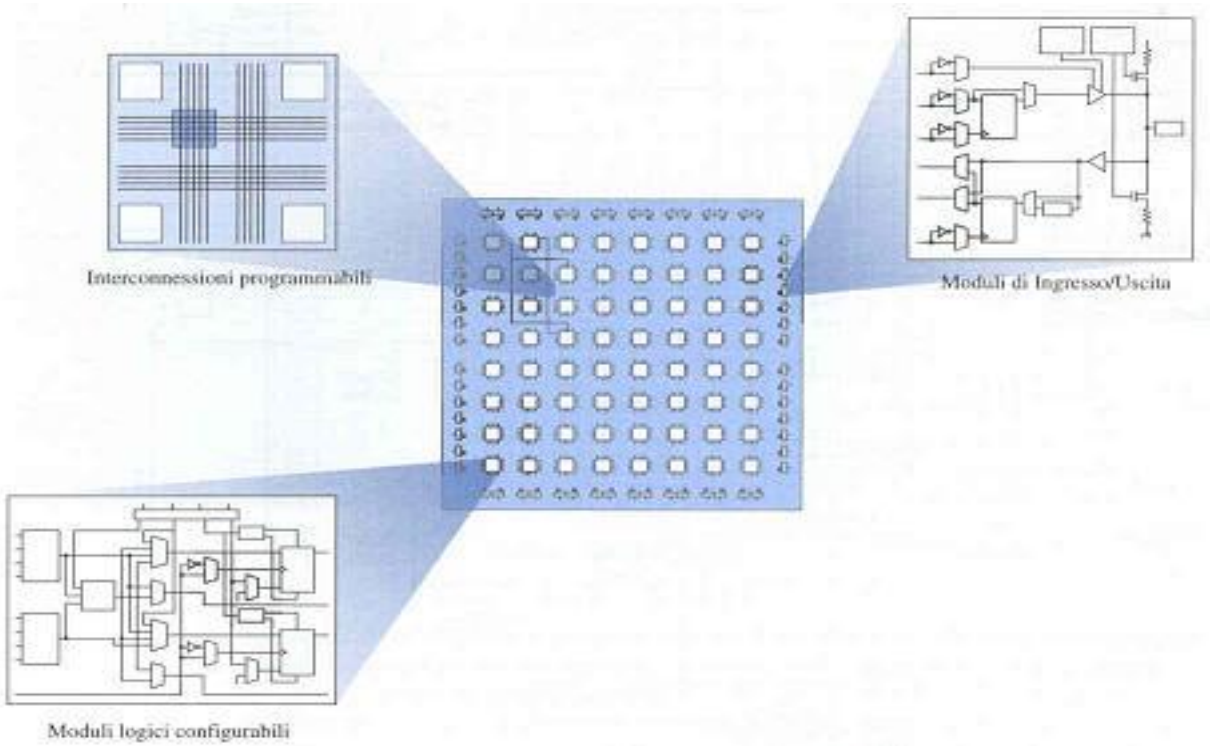

Fig.5 (6)

Le connessioni programmabili però non permettono il collegamento totale del dispositivo cioè

non permettono il collegamento di tutti i blocchi tra di loro questo permette di avere una minor complessità circuitale, vengono utilizzate delle matrici di commutazione.

Ogni modulo logico configurabile è chiamato CLB (*Complex Logic Block*), ogni CLB contiene dei blocchi detti *Slices* e che a loro volta contengono dei blocchi logici più piccoli. Questi blocchi logici all'interno delle *Slices* possono essere porte logiche, LUT e multiplexer oltre a componenti sequenziali. Per interfacciarsi al mondo esterno l'FPGA utilizza dei moduli di I/O.

Le LUT sono contenute in ogni cella logica base di un FPGA, sono dispositivi programmabili che possono realizzare qualsiasi funzione a tre o più ingressi. In figura 6 possiamo vederne la struttura, si tratta semplicemente di un multiplexer collegato con una SRAM programmabile.

Nella RAM viene scritta la tabella di verità della funzione da implementare e il multiplexer seleziona la cella dell'ingresso corrispondente al minterm selezionato dai tre ingressi di selezione.

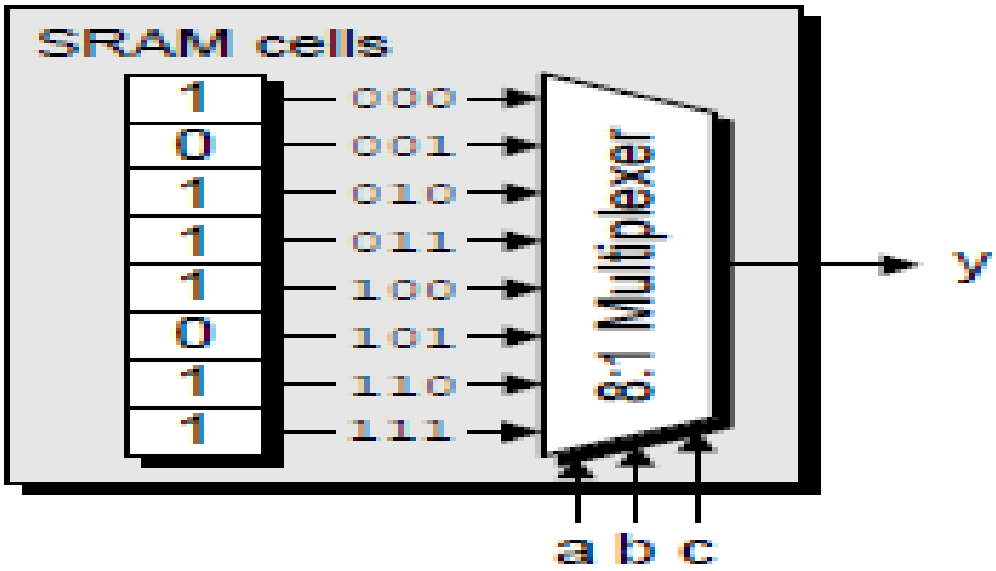

Fig. 6 (5)

Oltre alle LUT la cella logica contiene normalmente anche un *flip flop* che può essere collegato direttamente ad un ingresso del blocco logico oppure all'uscita della LUT in modo da renderla sequenziale (Fig. 7).

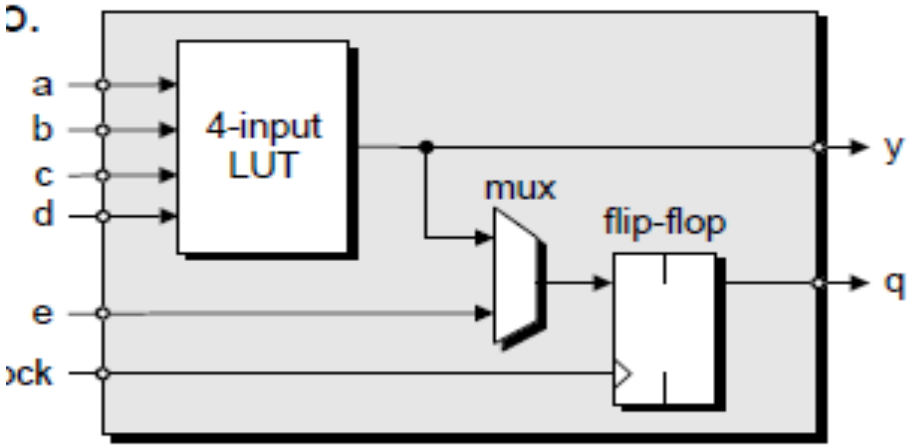

Fig. 7 (5)

Ogni cella logica è collegata con un'altra attraverso un multiplexer in modo da poter realizzare con due celle logiche una funzione a N+1 ingressi (dove N è il numero di ingressi della LUT). A sua volta le due *slices* possono essere collegate per espandere ancora le funzioni, così si possono implementare funzioni con qualsiasi numero di variabili. Unendo i blocchi logici è possibile implementare anche delle memorie infatti ogni LUT può essere configurata come *shift Register* o come RAM distribuita, anche in questo caso le LUT possono essere collegate in cascata per aumentare la dimensione della memoria distribuita.

Con l'aumento della complessità dei circuiti implementati su FPGA sono stati inseriti vari blocchi predefiniti ottimizzati per strutture specifiche. Ad esempio possono essere inseriti dei blocchi sommatori o essere implementati dei percorsi dedicati per il *carry* in modo da ottimizzare la costruzione di sommatori oppure possono essere inseriti dei moltiplicatori. Altri circuiti che normalmente si possono trovare all'interno di FPGA sono DCM (*Digital Clock Manager*) oppure addirittura microprocessori.

Nei CLB le interconnessioni programmabili permettono il collegamento totale di ciascun blocco del dispositivo quindi è possibile determinare a priori i ritardi tra i blocchi, al contrario negli FPGA la disposizione e i percorsi dei collegamenti vengono decisi durante il *routing* del circuito, solamente dopo questa operazione quindi è possibile determinare i ritardi tra i blocchi.

3.4 VHDL

VHDL è l'acronimo di *Very high speed integrated circuits Hardware Description Language* (5) ed è un linguaggio per la progettazione di circuiti digitali nato da un progetto del dipartimento della difesa Americano. Ora è utilizzato per implementare applicazioni che spaziano dai microprocessori (DSP, acceleratori grafici), alle comunicazioni (Cellulari, TV satellitare) all'automobile (Navigatori, controllo stabilità) e molte altre.

Il VHDL si presenta come un qualsiasi linguaggio di programmazione come possono essere java oppure C infatti presenta i classici costrutti *if then, for, while*. La differenza principale però è che il VHDL descrive un circuito Hardware quindi si deve immaginare che più parti del codice VHDL vengono eseguite contemporaneamente al contrario dei normali linguaggi di programmazione che eseguono solo istruzioni sequenziali, si dice quindi che il VHDL è un linguaggio concorrenziale. In VHDL quindi si utilizzano i construtti tipici dei linguaggi di programmazione per descrivere la relazione ingresso – uscita di un circuito.

Nella progettazione elettronica in VHDL si rispettano due passaggi fondamentali, viene per prima cosa inizializzata una *entity* che descrive come il componente è visto dall'esterno cioè i vari ingressi e uscite. Successivamente si progetta l'*architecture* cioè il comportamento del dispositivo.

Si può descrivere l'architettura di un componente in tre modi diversi, si può utilizzare la descrizione *behavoral* (comportamentale) che descrive il comportamento del dispositivo attraverso un algoritmo. Il circuito può inoltre essere descritto come *structural* (struttura) che descrive la struttura del dispositivo collegando tra loro componenti di basso livello cioè si realizzano dei componenti e li si collega tra loro. Normalmente l'ultimo passaggio prima dell'implementazione del circuito è scrivere una *netlist* che viene però sintetizzata automaticamente dal *tool* partendo dalle descrizioni *structural* o *behavoral* . Per verificare la corretta implementazione del dispositivo c'è la possibilità di eseguire delle simulazioni attraverso un file di *test-bench* che racchiude una descrizione dei segnali di ingresso da fornire al componente in modo da calcolarne poi le uscite.

3.5 Tecniche di progettazione

I passaggi tra l'inizio della fase di progettazione e il prodotto finito su un FPGA sono abbastanza standard e si possono riassumere in figura 8. Rispetto ad un ASIC si ha il vantaggio di avere a disposizione degli strumenti di simulazione e, inoltre, la sintesi a livello fisico del circuito consiste nel piazzamento dei vari componenti che lo compongono sfruttando le risorse disponibili nell'FPGA a disposizione. La prima fase di progettazione detta fase di sintesi, in comune con la progettazione ASIC, consiste nel dividere in blocchi il progetto e implementare i vari blocchi descrivendoli tramite un linguaggio ad alto livello.

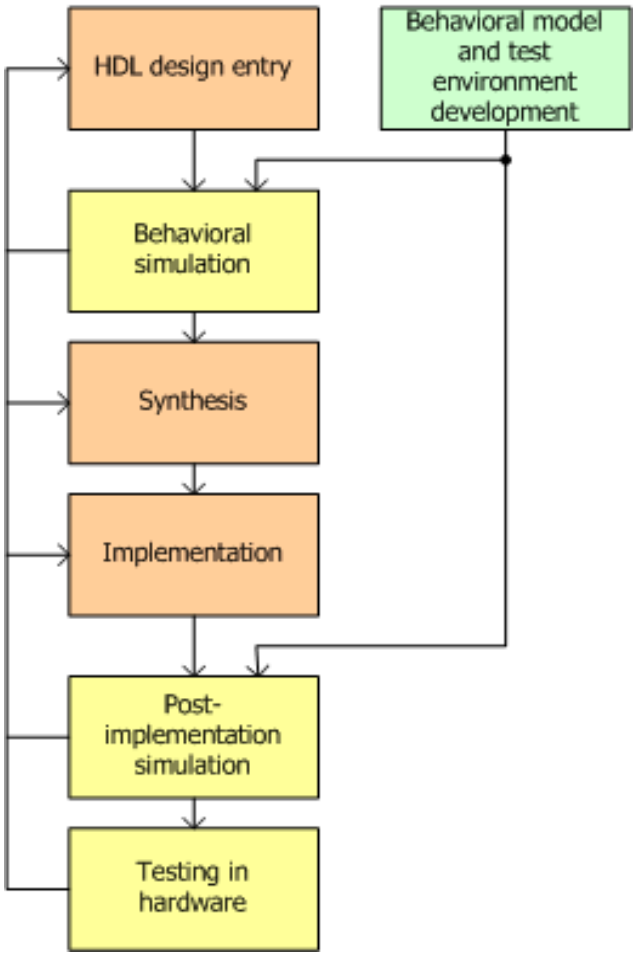

Fig.8 (7)

A questo punto si può verificare tramite una simulazione comportamentale se il nostro codice descrive in modo corretto il comportamento del circuito, per far questo vengono forniti degli ingressi attraverso un file di *test-bench*. La fase successiva è detta fase di implementazione, tramite l'apposito strumento software il codice viene tradotto in componenti virtuali con le medesime caratteristiche dei componenti fisici reali presenti nell'FPGA che possono essere simulati, questa simulazione è detta *post-routed* e simula perfettamente il comportamento del circuito. A questa fase segue quella di programmazione che configura il chip implementando il circuito realizzato.

3.6 Verifica, simulazione e analisi di un progetto su FPGA.

L'ambiente di sviluppo utilizzato per l'implementazione di un progetto su FPGA è ISE, che è stato realizzato dalla Xilinx. Questo software permette non solo di sintetizzare e implementare il codice VHDL scritto ma mette a disposizione vari *tool* molto utili che permettono di semplificare la realizzazione del codice e la valutazione delle prestazioni. Tra i *tool* a disposizione ne citeremo in particolare due: *iSim* e *XPower Analyzer* (8) ma ne esistono

altri altrettanto utili che permettono ad esempio di creare dei vincoli temporali oppure di visualizzare il circuito implementato all'interno dell'FPGA.

XPower Analyzer è un *tool* dedicato all'analisi del consumo di potenza di progetti dopo l'implementazione *post-route*. Fornisce inoltre dati riguardo alla temperatura interna in condizioni di lavoro e a riposo del dispositivo in modo da permettere al progettista di decidere quale tipo di raffreddamento sia necessario al chip programmato.

ISim è il tool più importante tra quelli presenti all'interno di ISE perché è il software che permette di simulare il codice VHDL. Il software esegue quattro tipi di simulazione: *behavoral* che è quella comportamentale che simula il codice VHDL, *post-translate* che introduce un primo livello di ottimizzazione, *post-Map* che utilizza i componenti specifici dell'FPGA su cui deve essere implementato il circuito e *post-route* che è la simulazione definitiva che tiene conto dei componenti dell'FPGA scelto e dei collegamenti tra di essi, ovviamente quest'ultimo tipo di simulazione risulta particolarmente complessa e quindi è molto lenta, a seconda dell'estensione del circuito implementato la simulazione può durare anche diversi giorni.

3.7 Applicazioni degli FPGA.

Le applicazioni degli FPGA includono l'elaborazione di segnali digitali, la protopizzazione di ASIC, l'emulazione di hardware e un crescente range di aree.

A questo scopo spesso vengono fornite dal produttore dell'FPGA delle *board* multifunzionali (Figura 9) che sono destinate al test di una serie di applicazioni base. Sono sempre presenti componenti come display, led, pulsanti, uscite standard come seriale usb ethernet e altre, e memorie di cui una flash solitamente dedicata alla riprogrammazione dell'FPGA.

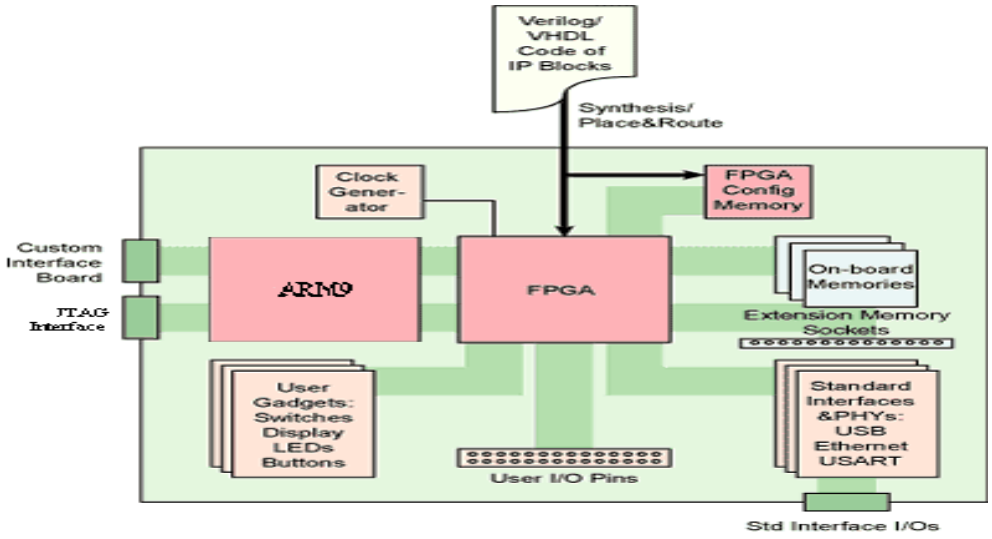

Fig. 9 (9)

L'utilizzo principale dell'FPGA è l'elaborazione di segnali digitali, questo perché gli FPGA hanno un architettura strutturata per lavorare in parallelo, il che rende questo tipo di operazioni più performanti rispetto alle performance che si hanno utilizzando un microprocessore. Nello specifico alcune applicazioni sono la crittografia e operazioni come FFT (*Fast Fourier Trasform*) o convoluzione.

Tutte le applicazioni in cui vengono usati gli FPGA sono destinate a progetti con basso volume di produzione dove il costo unitario del chip è un costo più affrontabile dei costi di progettazione di un ASIC a celle standard.

3.8 Gli FPGA Xilinx.

Xilinx è uno dei maggiori produttori al mondo di dispositivi logici, famosa per essere l'azienda che ha inventato il primo FPGA messo in commercio.

Le categorie di prodotti che Xilinx produce per quanto riguarda gli FPGA sono due: Virtex e Spartan. La famiglia Virtex racchiude i prodotti che hanno le migliori caratteristiche in termini di performance che quindi sono più costosi, invece la famiglia Spartan privilegia il minor costo alle prestazioni elevate.

3.9 Xilinx XC3S200

I parametri principali nella scelta di un FPGA generalmente sono il numero di CLB, i bit di memoria SRAM, il numero di moltiplicatori (infatti normalmente gli FPGA vengono utilizzati in circuiti di elaborazione in cui le operazioni di moltiplicazione devono essere ottimizzate) e il numero di I/O disponibili.

Qui di seguito in tabella 1 sono inseriti questi dati per l'FPGA utilizzato.

Dispositivo	**System Gates**	**Celle logiche**	**Righe**	**Colonne**	**CLD totali**	**Ram distribuita totale**	**BLock RAM**	**Moltiplicatori**	**DCM**	**I/O**
XC3S200	200K	4320	24	20	480	30K	216K	12	4	173

Tab 1 (10) (11)

4 Sincronizzatore UWB-IR.

4.1 Descrizione generale del progetto.

In riferimento alla comunicazione UWB che, come precedentemente precisato, è un tipo di comunicazione basata sulla presenza o meno di impulsi di energia in istanti temporali specifici, la sincronizzazione temporale è fondamentale. Infatti nel caso in cui il segnale risulti sfasato non si riceve il dato corretto.

Il circuito implementato nel presente lavoro è di conseguenza finalizzato alla perfetta sincronizzazione della trasmissione.

Il sincronizzatore rimane in stato di riposo fino all'arrivo di un segnale generato a livello software più alto che attiva l'elaborazione del dato in ingresso. Al segnale ricevuto viene applicata un'operazione di convoluzione binaria con una specifica parola binaria chiamata "Gold Code" per verificare la corretta destinazione del segnale e la sua corrispondenza con la fase del clock interno. Nel caso ci sia uno sfasamento, sia nell'inizio del Gold Code che del segnale con il segnale di clock, esso viene corretto e si rimane in attesa del Gold Code negato per dare il via alla comunicazione.

4.2 Struttura della comunicazione

La comunicazione UWB di cui ci occupiamo è una comunicazione via radio che trasmette il segnale con una codifica posizionale nel tempo, come si può vedere in figura 8 ogni bit dura 10 impulsi di clock. Durante i primi due slot temporali viene inviato un impulso dal trasmettitore in corrispondenza del primo o del secondo slot a seconda che si voglia codificare un segnale corrispondente al valore binario uno o al valore binario zero.

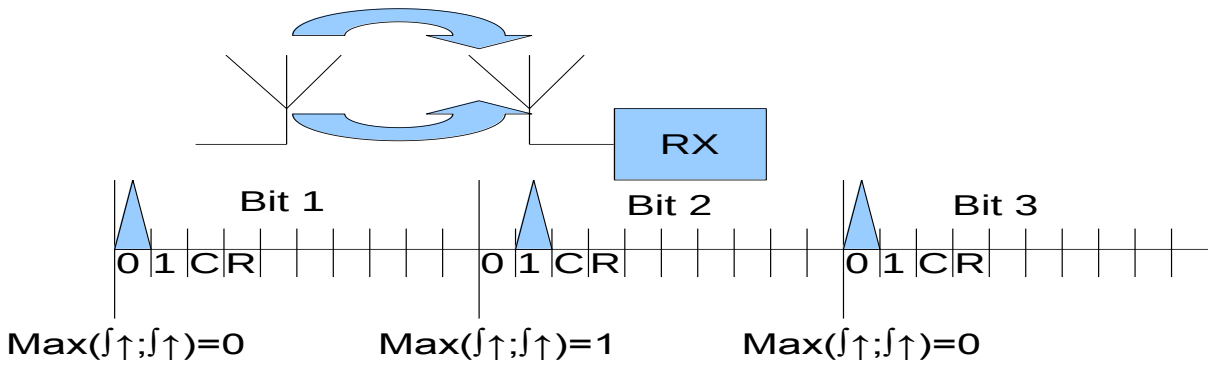

Fig. 8

Nel blocco ricevitore, già implementato e quindi qui considerato *black-box*, viene calcolato l'integrale nel primo slot e nel secondo ricavandone l'energia, di seguito nel terzo slot temporale (slot di calcolo) viene calcolato il massimo tra i due integrali. Dopo questa ultima operazione il dato è pronto all'uscita del ricevitore. Il quarto slot temporale agisce da reset per ricevitore, il bit in uscita rimane costante fino al successivo calcolo del segnale ricevuto. I successivi sei slot temporali non sono rilevanti, vengono lasciati non utilizzati per rispettare le limitazioni di potenza media dello standard di trasmissione UWB.

La trasmissione UWB si esegue su una banda di frequenze senza licenza ovvero libere che possono essere utilizzate contemporaneamente da più utenti, per evitare di ricevere trasmissioni non pertinenti si usa il sistema dei Gold Code.

Un Gold Code è una parola binaria a 31 bit che ha delle proprietà particolari. Due *gold code* sono tra di loro ortogonali quindi la loro convoluzione è nulla, sfruttando questo è possibile identificare se una trasmissione è destinata a noi (stesso *gold code),* oppure no, (*gold code* diverso), tramite un'operazione di convoluzione.

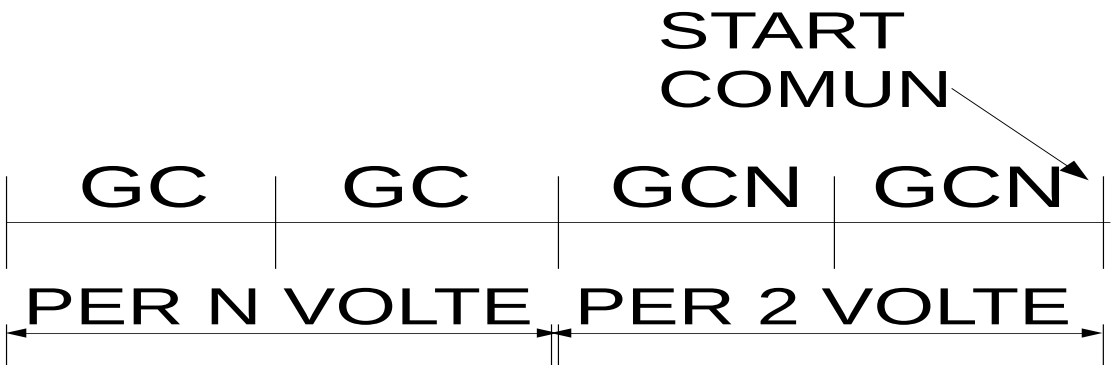

Fig. 9

Per poter sincronizzare la comunicazione vengono inviati prima della trasmissione un numero definito di Gold Code (nel nostro caso 20) che ci permettono di capire se la trasmissione è destinata a noi e di sincronizzarci. Per capire quando la comunicazione sta per iniziare vengono inseriti altri due *Gold Code* negati al termine dei quali ha inizio la comunicazione.

4.3 Ingressi e uscite del sincronizzatore.

Il circuito sincronizzatore esternamente si presenta con cinque ingressi e due uscite (figura 10). Uno degli ingressi è il clock che è generato esternamente, il circuito è stato progettato per funzionare a 50MHz. Ovviamente in ingresso si ha anche il gold code di riferimento per la nostra comunicazione e l'uscita del blocco ricevitore collegato all'antenna.

Il circuito dispone inoltre di un ingresso di reset che inizializza il componente e di un ingresso di start che gli indica quando iniziare ad elaborare l'ingresso. In uscita abbiamo due segnali: un segnale sincro che segnala quando viene ricevuto il gold code negato con un fronte ed un contatore che indica quanto manca al raggiungimento del segnale di sincronizzazione.

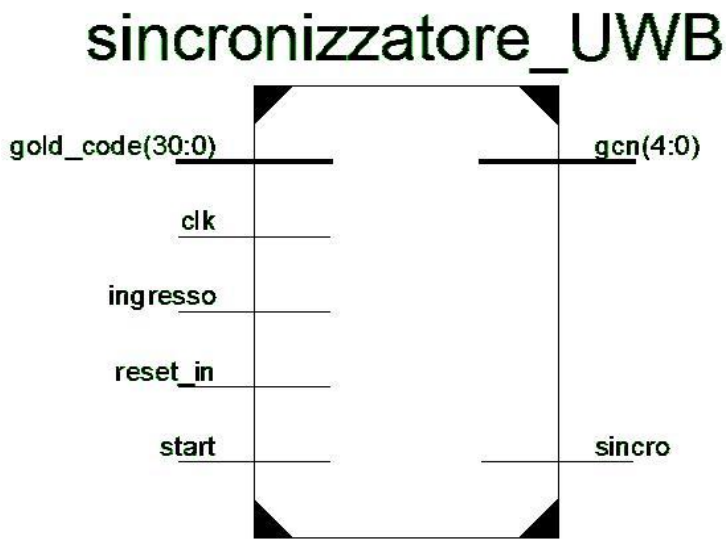

Fig.10

Si è diviso il circuito sincronizzatore in più blocchi comandati da una macchina a stati che li coordina, nel proseguo di questo capitolo si troverà la descrizione di ciascuno di questi blocchi.

Il circuito si compone di:

- 31 circuiti comparatore instanziati con il *gold code* in tutte le sue traslazioni
- 1 circuito di selezione
- 1 macchina a stati finiti di controllo
- 1 circuito comparatore instanziato con *gold code* negato con reset automatico

In figura 11 si può vedere uno schema a blocchi generale del progetto nel suo insieme.

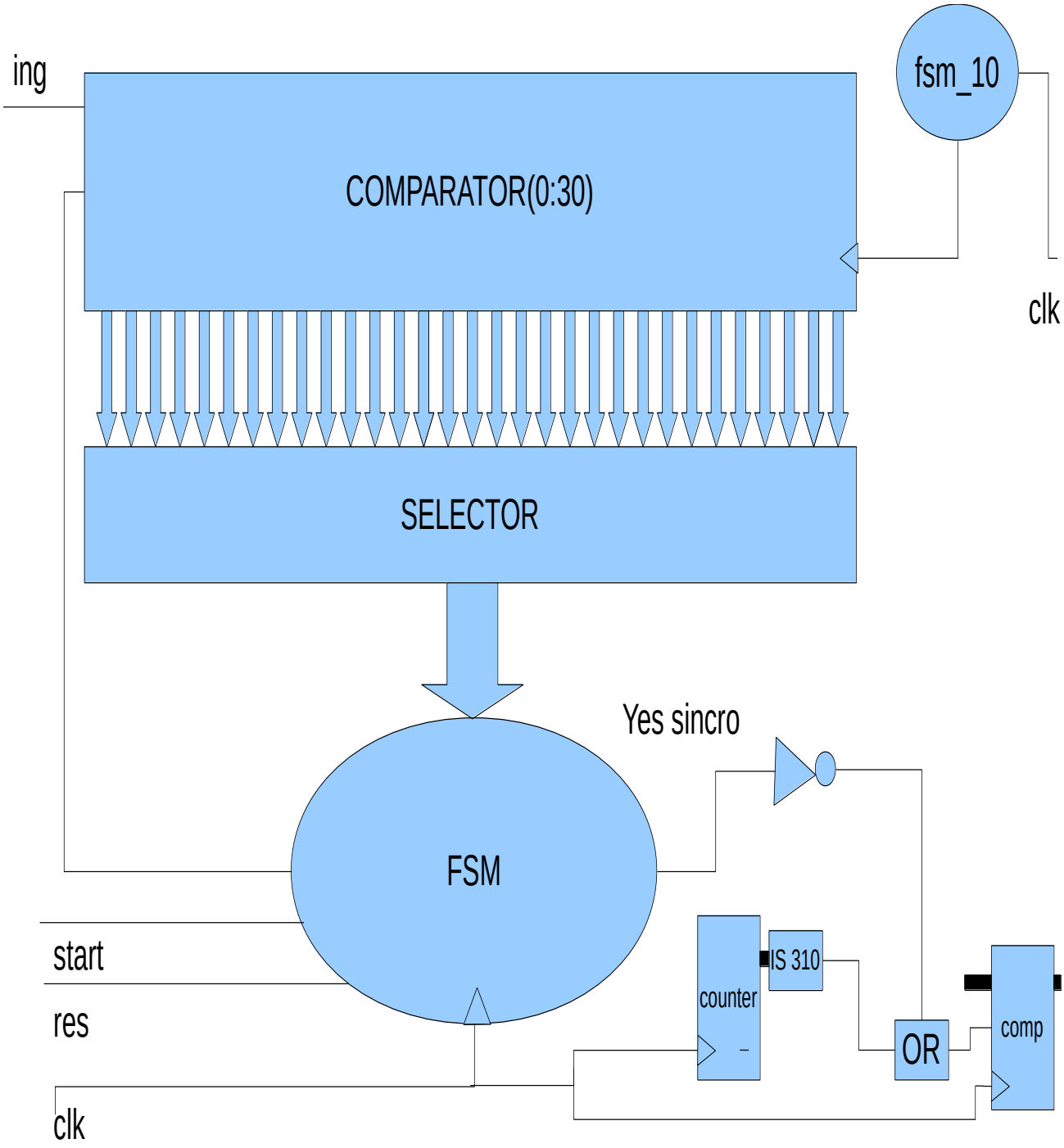
ing
fsm_10
COMPARATOR(0:30)
clk
SELECTOR
Yes sincro
FSM
start
res
clk
counter
IS 310
OR
comp

Fig. 11

5 Interfaccia con l'ingresso.

5.1 Gestione del clock interno.

Esistono due categorie di componenti in cui possiamo dividere i blocchi che compongono il sincronizzatore, i blocchi di interfaccia con l'ingresso ed i blocchi di controllo. La differenza principale tra le due categorie è la frequenza di lavoro infatti, come precisato nella sezione "Struttura della Comunicazione", ogni bit del segnale in ingresso corrisponde a 10Tclk, questo significa che i componenti di interfaccia con l'ingresso devono lavorare con frequenza dieci volte inferiore al clock di sistema.

Inizialmente si era optato per implementare un divisore di frequenza, ne risultava tuttavia un clock generato da logica combinatoria e conseguentemente un segnale non pulito e non adatto quindi il problema è stato superato inserendo in tutti i componenti che necessitavano di un clock di frequenza inferiore un *chip enable* che, "congelando" il componente, lo rendeva insensibile al clock. Il segnale di chip enable ha quindi il compito di far funzionare il componente per un ciclo di clock e tenerlo congelato per i restanti nove.

Il segnale che pilota i vari chip enable è stato implementato con una macchina a stati finiti chiamata fsm_10 (figura 12a) . Questa macchina ha due ingressi, un reset e il segnale di clock, inoltre ha un segnale di uscita il quale pilota i vari *chip enable*. Per generare il segnale desiderato si sono implementati dieci stati di cui nove danno in uscita uno 0 logico e uno da in uscita un 1 logico(figura 12b).

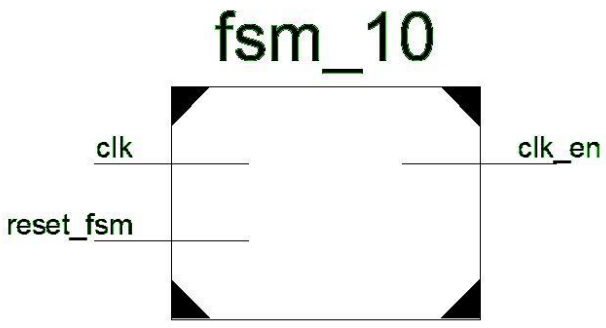

Fig 12a

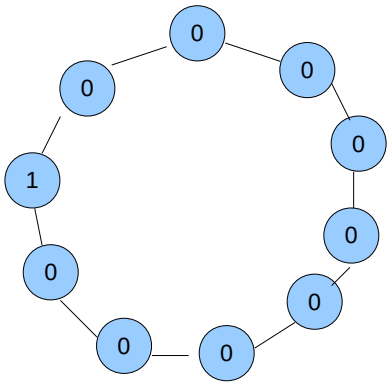

Fig 12b

In figura 13 si può vedere la simulazione della macchina a stati e il segnale di controllo dei chip enable.

Fig. 13

5.2 Comparatore.

Il comparatore è il componente base per eseguire la convoluzione tra il segnale binario in ingresso ed il *Gold Code.* Come si può vedere in figura 14 il comparatore ha sei ingressi ed un'uscita. Uno degli ingressi è ovviamente il `reset` che blocca le commutazioni dell'uscita inizializzandola a 0.

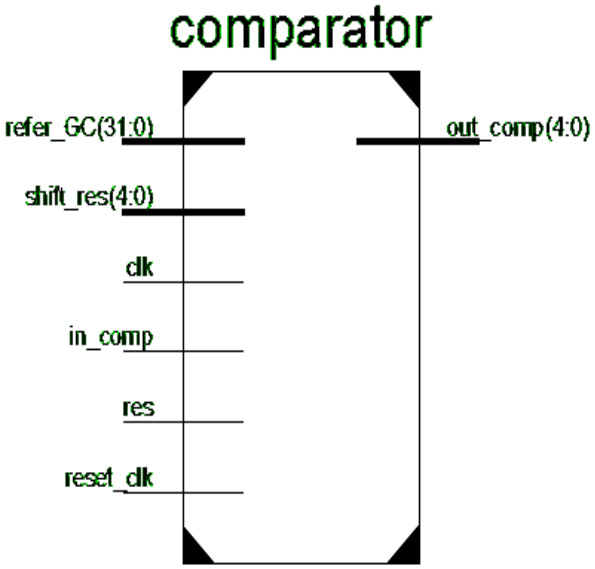

Fig. 14

Il segnale di clock è quello di sistema e quindi funziona a velocità 10 volte superiore alla frequenza di lavoro che dovrebbe avere il comparatore essendo un'interfaccia con l'ingresso, è stato quindi inserito l'ingresso `reset_clk` che viene direttamente pilotato dalla macchina a stati fsm_10 rendendo sensibile il comparatore solo ad un impulso di clock ogni dieci campionando così correttamente l'ingresso. L'ingresso `refer_GC` fornisce il *gold code* al comparatore che lo scorre. Il bit di inizio da cui il contatore inizia a scorrere il *gold code* è determinato dall'ingresso `shift_res`; questo ingresso è utile perché permette di utilizzare

un solo registro letto da ciascun comparatore in modo indipendente al posto di 31 registri contenenti il *gold code* in tutte le sue traslazioni necessarie per la convoluzione, creando comunque l'effetto di shifting. Il comparatore quindi confronta il segnale di ingresso con il *gold code,* perciò ad ogni impulso del clock interno viene confrontato l'ingresso con il bit del *gold code* corrispondente al valore del contatore interno (cont1 figura 15) inizializzato al valore di `shift_res`. Se i due bit sono uguali viene incrementato un altro contatore che conta quanti bit sono uguali. Se il valore di uscita del contatore arriva a 31 significa che la trasmissione è destinata a noi e abbiamo ricevuto esattamente il *gold code* assegnato alla nostra trasmissione o una sua traslazione.

In figura 15 possiamo vedere uno schema a blocchi di principio della struttura del comparatore.

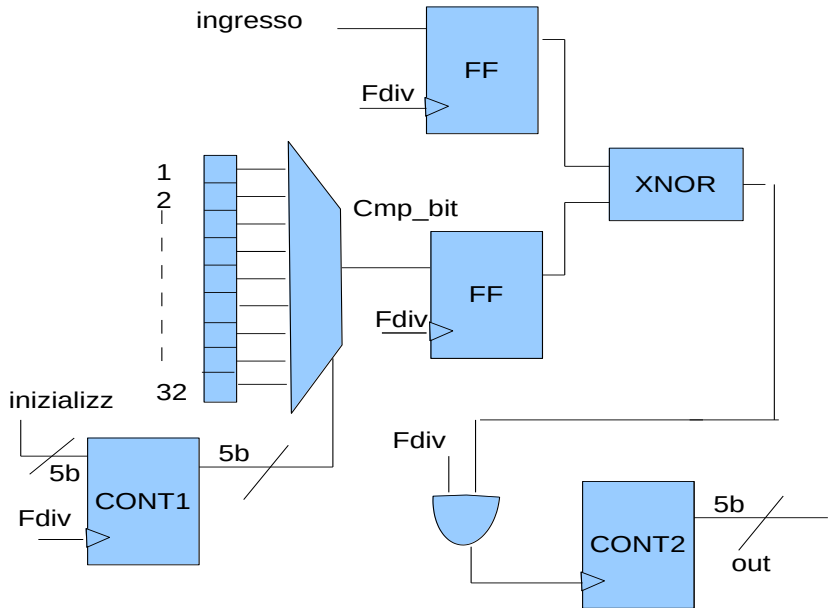

Fig.15

5.3 Convolutore.

Per identificare la provenienza della comunicazione si usa il sistema dei *gold code*, la loro caratteristica principale, come già precisato, è che la loro convoluzione è nulla.

La convoluzione è un'operazione matematica tra due funzioni f(.), g(.) e precisamente si definisce convoluzione:

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Caso continuo (12)

$$(f * g)(m) = \sum_{n} f(n)g(m-n)$$

Caso discreto

Si tratta quindi dell'area del prodotto della funzione f(.) per tutte le traslazioni della funzione g(.), nel caso binario l'area della funzione f (ovvero l'ingresso) per le traslazioni della funzione g (ovvero delle finite traslazioni del *gold code*).

Per costruire un convolutore binario si calcola l'area di un segnale in ingresso moltiplicato per le varie traslazioni del segnale di riferimento. Nel nostro caso il segnale di riferimento è il *gold code* che quindi va sfasato in tutte le sue varianti. L'ingresso `shift_res` del comparatore a questo scopo permette al componente di iniziare a leggere il *gold code* da un bit diverso producendo l'effetto dello sfasamento.

Come possiamo vedere in figura 16 vengono piazzati 31 comparatori che calcolano l'area di ogni traslazione, se la comunicazione è pertinente e stiamo leggendo correttamente i bit allora uno e uno solo dei comparatori ha l'uscita a 31.

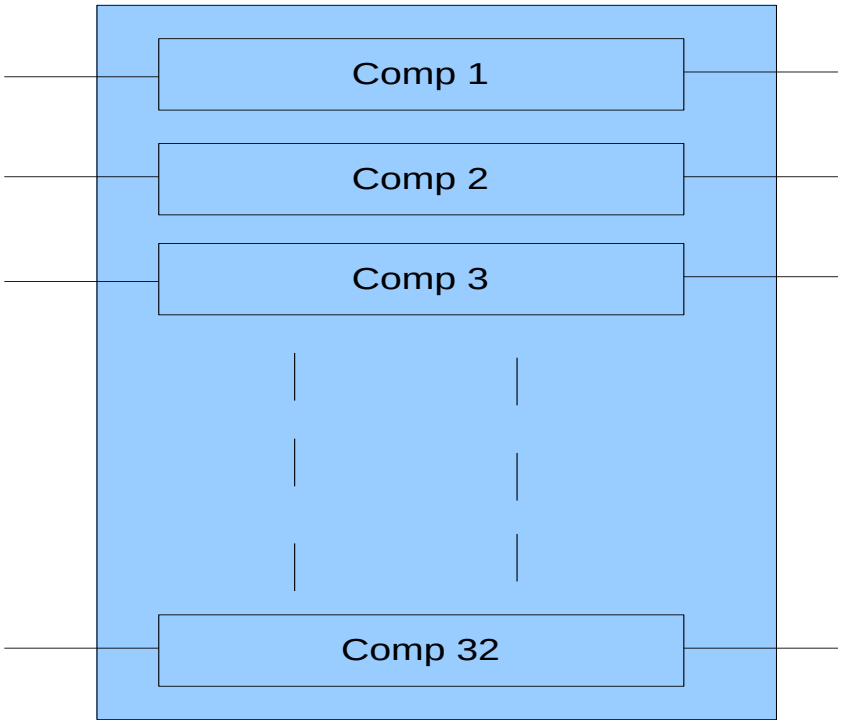

Fig. 16

In questo modo viene eseguita la convoluzione tra segnale di ingresso e *gold code*.

5.4 Selettore.

Dopo la conferma che la comunicazione è destinata al nostro ricevitore, in quanto uno dei comparatori è arrivato a 31, è necessario sincronizzarsi per allinearsi all'inizio del sucessivo *gold code* che verrà ricevuto dal circuito. Se il comparatore numero "b" è diventato 31 per essere perfettamente in fase con il segnale di ingresso è necessario attendere 32-b bit ovvero *(32-b)*M* impulsi di clock in quanto ogni bit dura 10Tclk. Serve quindi un circuito che individui quale comparatore ha raggiunto il valore 31, questa operazione la esegue il selettore.

Come si vede in figura 17 il selettore ha 31 ingressi, uno per ogni comparatore, e un'uscita che assume valori tra 0-31. L'uscita assume valore 0 è nel caso nessuno dei comparatori abbia raggiunto il valore 31, questo avviene nel caso di clock interno fuori fase o nel caso che la comunicazione non sia destinata al nostro circuito. I valori tra 1 e 31 rappresentano il numero del comparatore che ha raggiunto il valore 31. Il dato in uscita del selettore viene elaborato dalla macchina a stati finiti che provvederà a portarsi in fase con l'ingresso.

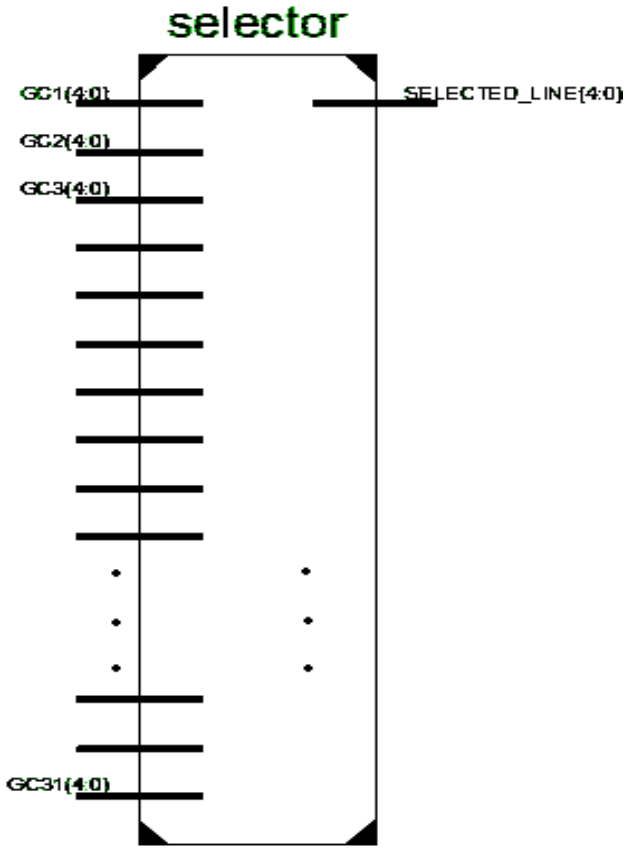

Fig. 17

5.5 Prova Comparatore e Selettore.

Per provare il funzionamento dei due circuiti appena descritti, comparatore e selettore, si è realizzato un piccolo circuito di test che possiamo vedere in figura 18.

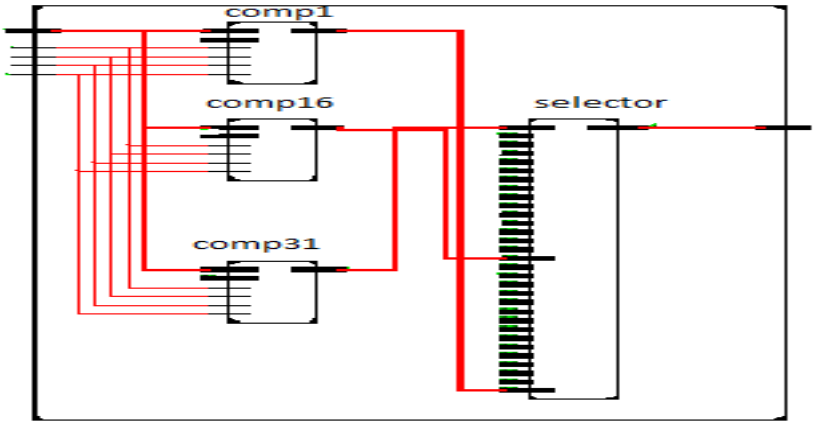

Fig.18

Sono stati piazzati tre comparatori uno col *gold code* in fase, uno traslato di 15 bit e uno traslato di 31 bit che sono stati poi collegati ai corrispondenti ingressi del selettore. Si è reso necessario simulare questo circuito con solo tre comparatori e il selettore invece del circuito completo perché la complessità della simulazione post route di un circuito completo avrebbe richiesto troppo tempo. Nella figura 19 possiamo vedere la simulazione del circuito di prova, l'ingresso è stato settato in modo da corrispondere al gold code sfasato di 31, infatti l'uscita assume valore 11111, cioè 31, quando il comp31 raggiunge 31.

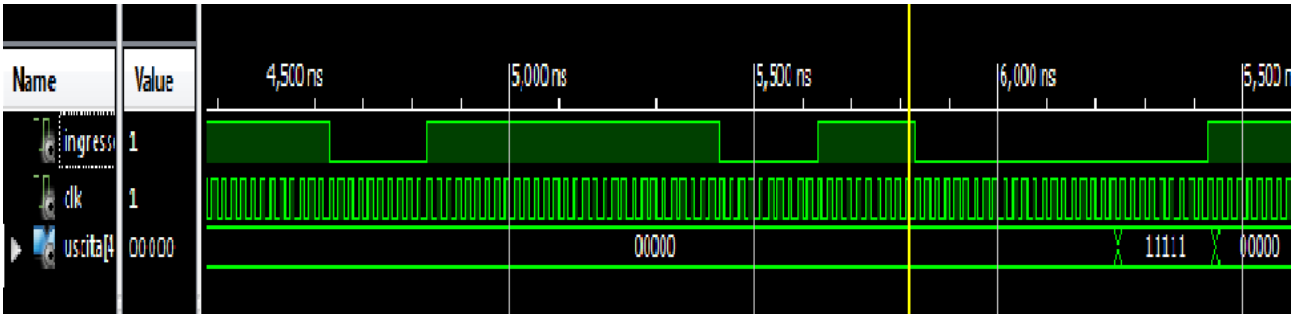

Fig. 19

6 Gestione del controllo.

6.1 Macchina a stati finiti di controllo.

La macchina a stati finiti è il dispositivo che gestisce e regola le operazioni di tutti i componenti del sincronizzatore fornendo i segnali di reset, elaborando i dati e agendo di conseguenza in modo da portarsi in fase col segnale di ingresso.

Come possiamo vedere in figura 20 la macchina a stati ha quattro ingressi e tre uscite. È progettata per lavorare con il clock a 50 MHrz della *board* multifunzionale della Xilinx su cui è inserito l'FPGA. In ingresso ha un reset asincrono (`reset_fsm`) e un segnale di START che è comandato da un livello logico più alto e che indica alla macchina a stati quando iniziare ad analizzare il dato in ingresso. Riceve inoltre il segnale di uscita del selettore (`selected_GC`) e lo elabora per calcolare i tempi di attesa. Quindi la macchina a stati verifica e autentica il segnale di ingresso allineandosi e attivando l'attesa del *gold code* negato con l'ingresso `yes_sincro`.

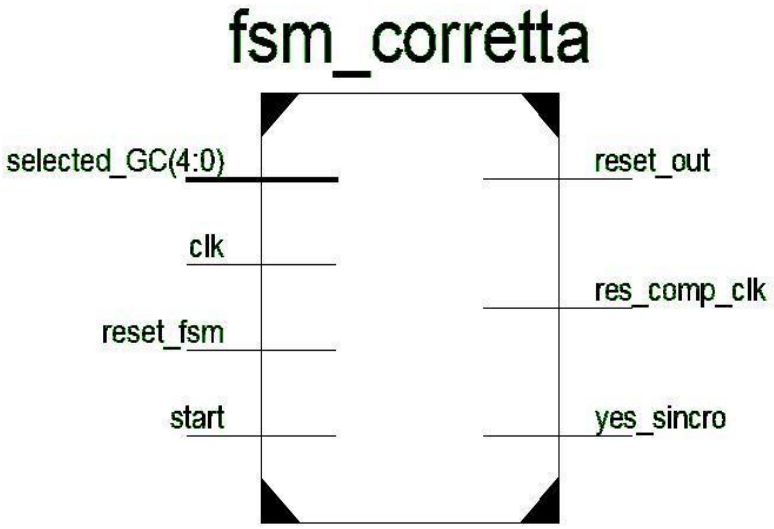

Fig. 20

In figura 21 possiamo vedere lo schema a blocchi della macchina a stati; in fase di progettazione sono stati definiti due tipi di stati: lo stato tipo LOAD (presentato in colore blu) e lo stato tipo WAIT (in colore verde). All'interno della macchina a stati è presente un contatore che viene decrementato per creare le attese necessarie al funzionamento corretto del circuito. Questo contatore viene inizializzato al valore di un segnale interno chiamato

how_mutch_wait negli stati di tipo LOAD e viene decrementato fino a zero negli stati di tipo WAIT.

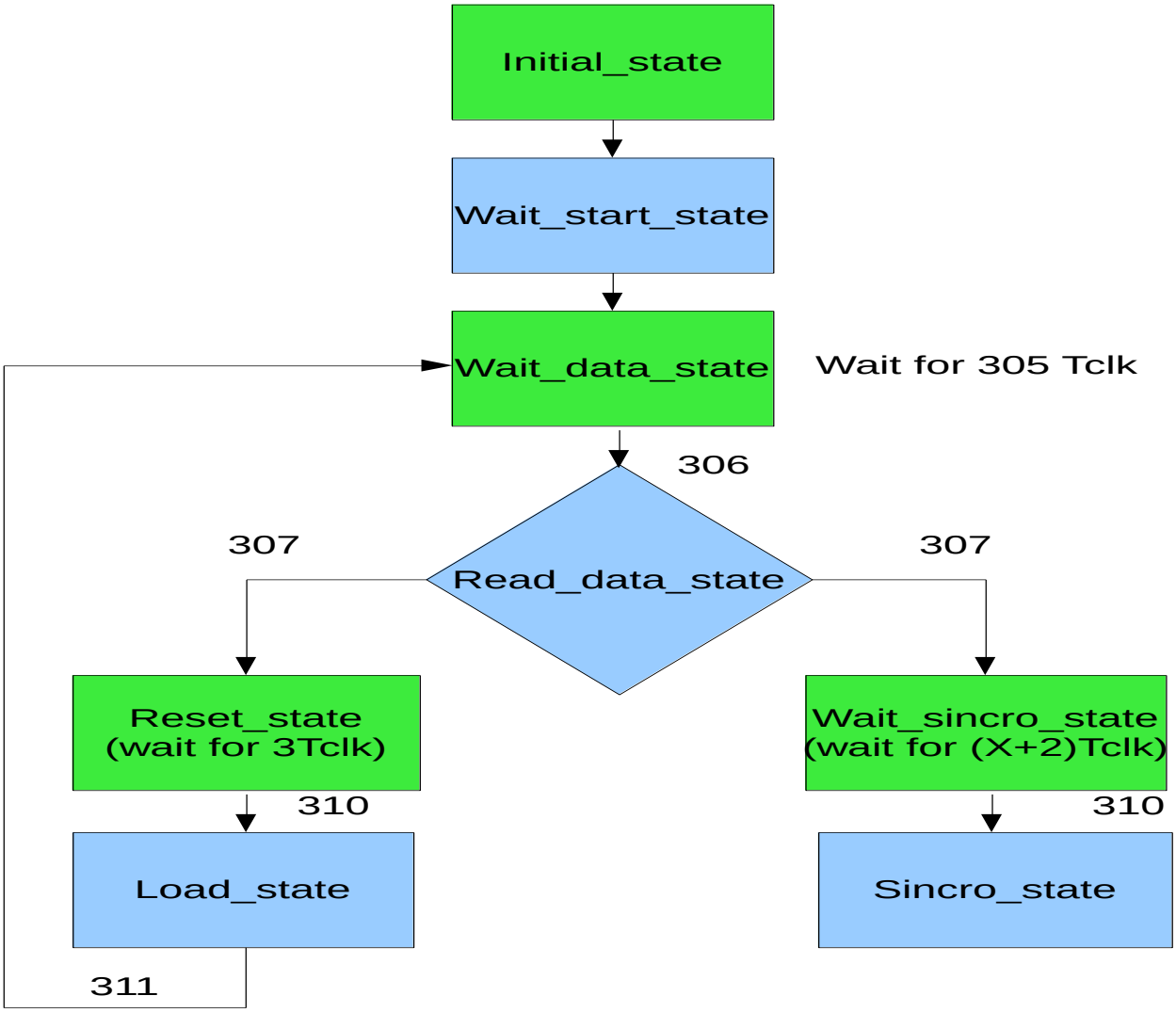

Fig 21

Dopo lo stato read_data_state il diagramma a stati ha una ramificazione. Viene percorso il ramo di destra quando la trasmissione è destinata al nostro dispositivo e dobbiamo semplicemente portare la lettura dell'ingresso all'inizio del *gold code* successivo. Nello stato wait_data_state si attendono 305 impulsi di clock, in totale un gold code dura 310 impulsi di clock e l'elaborazione eseguita dalla macchina a stati deve avvenire all'interno di questo periodo di tempo, da read_data_state a sincro_state vengono quindi recuperati i 5 bit guadagnati riducendo il tempo di attesa durante wait_data_stat. Siccome uno dei comparatori ha raggiunto il valore 31, e quindi abbiamo ricevuto un dato corretto, il campionamento dell'ingresso avviene nella posizione corretta. Di conseguenza la

somma totale, senza contare lo sfasamento per portarsi all'inizio del gold code che dipende da quale comparatore ha raggiunto il valore 31, deve essere 310.

La macchina a stati percorre invece il ramo di sinistra dello schema a blocchi quando la trasmissione ricevuta è destinata ad un altro dispositivo oppure quando la lettura dell'ingresso non avviene nella posizione corretta. Nel caso la trasmissione sia destinata ad un altro dispositivo la macchina a stati continuerà a percorrere il ramo di sinistra in un *loop* infinito. Nel caso che la trasmissione sia destinata al nostro dispositivo invece continuando a percorrere il ramo di sinistra lo sfasamento nella lettura viene corretto, infatti la somma totale degli impulsi di clock nel ramo di sinistra è 311, in questo modo, come si può vedere in figura 22, il ramo di sinistra sfasa la ricezione dell'ingresso di uno slot temporale così il campionamento dell'ingresso viene provato in tutte le dieci posizioni possibili. Quando il campionamento sarà in fase con l'ingresso uno dei comparatori arriverà così al valore 31 e di conseguenza la macchina percorrerà il ramo di destra.

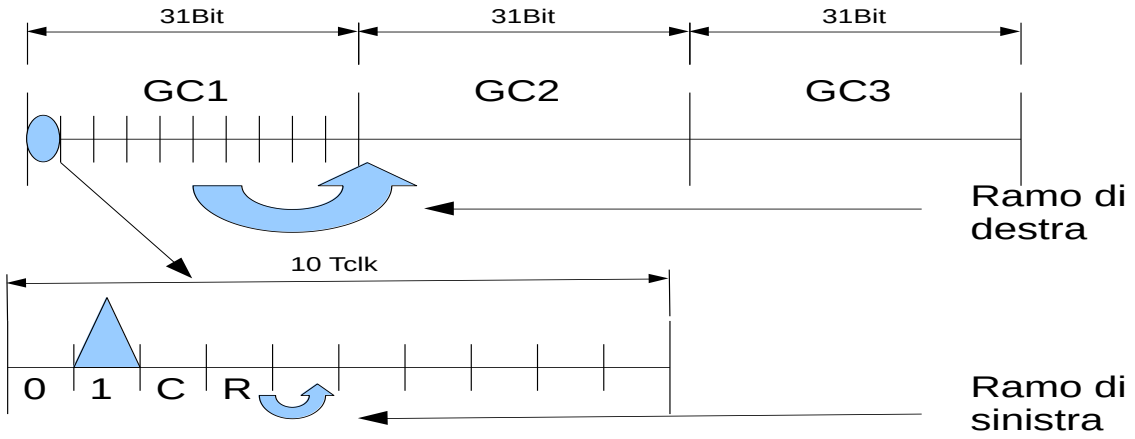

Fig 22

Di seguito verranno descritti i singoli stati della macchina realizzata.

6.1.1 Initial state

Lo stato `initial_state` (figura 23) è definito uno stato tipo wait, infatti `how_mutch_wait` è indicato come "don't care", anche se in realtà in questo caso il registro non viene decrementato e il passaggio allo stato successivo non è condizionato al valore del contatore. `Initial_state` resetta completamente il circuito infatti le uscite `reset_out` e `reset_comp_clk` sono a livello logico alto. Questo stato è necessario per il tempo di set del divisore di clock .

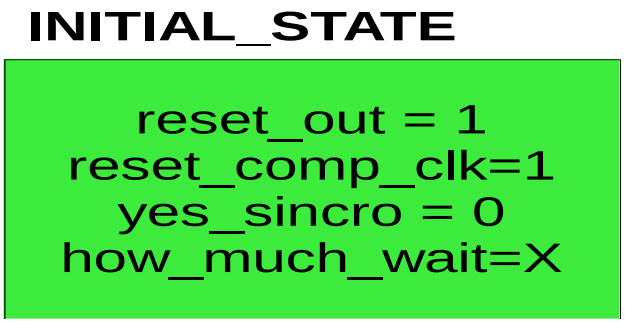

Fig 23

6.1.2 Wait start state

Lo stato `wait_start_state` (figura 24) è uno stato tipo load infatti il segnale `how_much_wait` è definito, in questo caso vale 305. Tale valore rappresenta i cicli di clock ovvero gli impulsi di clock necessari ai 31 comparatori per confrontare l'ingresso e il *gold code* traslato in tutte le sue varianti. Qui il contatore viene solamente inizializzato, l'attesa avviene nello stato successivo. Il passaggio allo stato `wait_data_state` avviene alla ricezione del segnale di `start`.

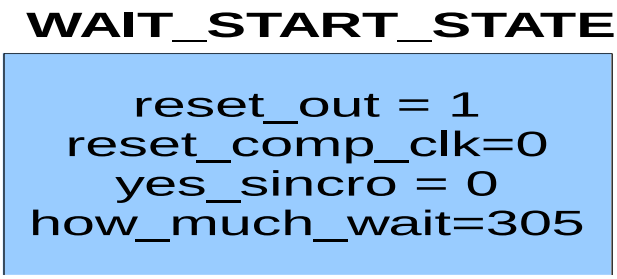

Fig 24

Affinché il sistema funzioni correttamente è necessario che il segnale di `start` arrivi sincrono con il clock di frequenza ridotta che comanda i comparatori, a questo scopo viene utilizzato il circuito in figura 25a.

Per portare in fase il segnale di start con il clock a frequenza ridotta, il segnale di `start` viene campionato con un *flip flop* e viene inviato alla macchina a stati finiti. Questa soluzione presenta l'inconveniente di far arrivare il segnale di start alla macchina a stati in ritardo di un impulso di clock rispetto al segnale inviato ai comparatori. Per risolvere il problema è stato aggiunto il secondo *flip flop* che ritarda anche quest'ultimo segnale in modo da portare in fase il segnale destinato alla macchina e quello destinato ai comparatori. In figura 25b si possono vedere le forme d'onda del circuito.

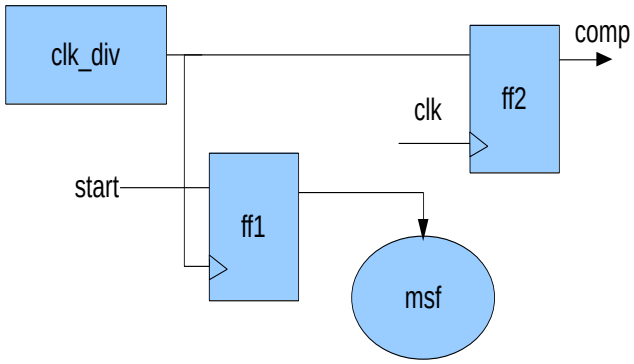

Fig 25a

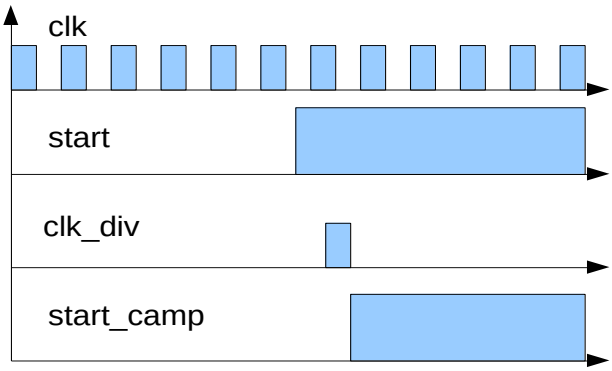

Fig 25b

6.1.3 Wait data state

Lo stato `wait_data_state` (figura 26) è di tipo wait. Il contatore interno è stato inizializzato allo stato precedente al valore 305, il passaggio allo stato successivo è quindi condizionato al valore del contatore che viene decrementato progressivamente fino a 0. Il valore 305 a cui è inizializzato il contatore è stato scelto considerando la durata di un *gold code*. Ogni *gold code* è lungo 31 bit ognuno di durata 10 Tclk per un totale di 310 Tclk. Il dato però è pronto già dopo i primi 4 impulsi di clock quindi è stato scelto il valore 305 cosicché i restanti 5 impulsi che sono stati così recuperati vengono utilizzati dalla macchina a stati per elaborare il dato e operare le decisioni.

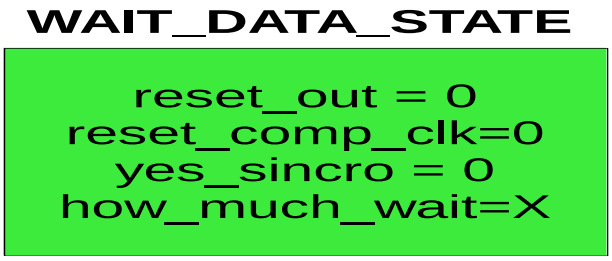

Fig 26

6.1.4 Read data state

Lo stato read_data_state (figura 27) è di tipo load infatti il suo scopo principale è leggere `selected_GC`, ovvero l'uscita del selettore, e assegnare il valore corrispondente al registro `how_much_wait` determinando poi quale sarà lo stato successivo.

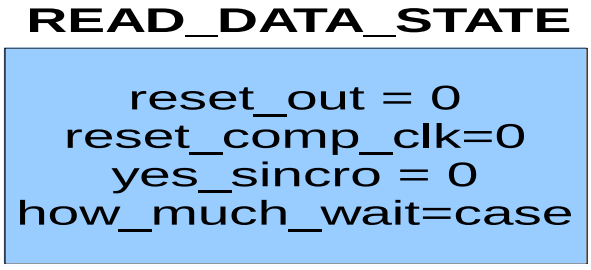

Fig 27

Nella tabella 2 possiamo vedere i vari valori assegnati al registro `how_much_wait` che si conseguenti al valore di `selected_GC`. Gli stati successivi possibili sono due: `reset_state` e `wait_sincro_state`.

Selected_GC	(32-sel_GC)*M+2	Ramo selezionato	Selected_GC	(32sel_GC)*M+2	Ramo selezionato
00000	3	Sx	10000	162	Dx
00001	312	Dx	10001	152	Dx
00010	302	Dx	10010	142	Dx
00011	292	Dx	10011	132	Dx
00100	282	Dx	10100	122	Dx
00101	272	Dx	10101	112	Dx
00110	262	Dx	10110	102	Dx
00111	252	Dx	10111	92	Dx
01000	242	Dx	11000	82	Dx
01001	232	Dx	11001	72	Dx
01010	222	Dx	11010	62	Dx
01011	212	Dx	11011	52	Dx
01100	202	Dx	11100	42	Dx
01101	192	Dx	11101	32	Dx
01110	182	Dx	11110	22	Dx
01111	172	Dx	11111	12	Dx

Tab 2

Lo stato `reset_state` che nello schema a blocchi si trova nel ramo di sinistra viene assunto nel caso `selected_GC` sia rimasto a 0, invece `wait_sincro state` che si trova nel ramo di destra viene assunto nel caso `selected_GC` abbia un valore diverso da 0.

6.1.5 Reset state

Lo stato `reset_state` (figura 28) è di tipo wait viene infatti atteso il valore caricato nel contatore durante lo stato precedente, ovvero in questo caso 3. Il sistema viene quindi mantenuto in condizione di reset in attesa di ricominciare ad elaborare l'ingresso del sistema, infatti l'uscita `reset_out` è a livello logico alto. Questo stato e il successivo hanno lo scopo di correggere lo sfasamento del segnale di clock con l'ingresso. Per correggere lo sfasamento la somma totale nel ramo sinistro deve essere 311 in modo da scalare di un ciclo di clock la lettura dell'ingresso. Dopo 3 cicli di clock si passa quindi allo stato successivo.

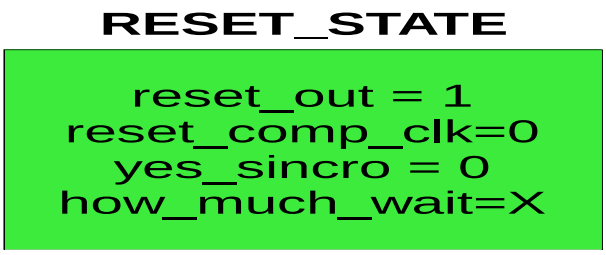

Fig 28

6.1.6 Load state

Lo stato `load_state` (figura 29) è uno stato di tipo load, inizialmente non era stato implementato ma si è reso necessario per mantenere l'alternanza di stato load e stato wait, infatti unico scopo di questo stato è permettere di caricare di nuovo il valore 305 nel registro del contatore in modo che nello stato successivo, ovvero `wait_data_state`, possa essere decrementato fino a 0 in attesa del calcolo dei comparatori.

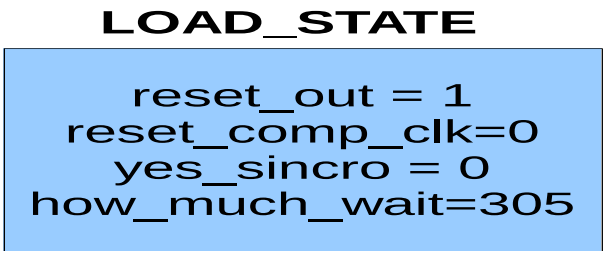

Fig 29

6.1.7 Wait sincro state

Lo stato `wait_sincro_state` (figura 30) è di tipo wait. Il registro `how_much_wait`, che è stato inizializzato secondo la tabella 2, viene decrementato fino a 0 in modo da portare la lettura dell'ingresso all'inizio del *gold code*. Il sistema durante questo periodo viene mantenuto in reset quindi l'ingresso non viene elaborato. Al termine del conteggio del contatore lo stato della macchina a stati finiti viene aggiornato e si passa allo stato `sincro_state`.

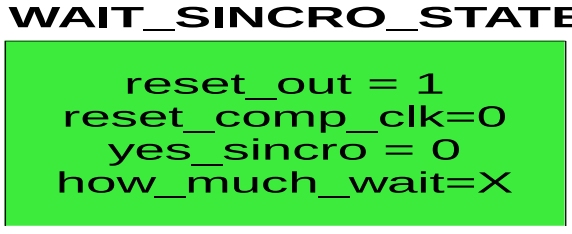

Fig 30

6.1.8 Sincro state

Lo stato `sincro_state` (figura 31) è lo stato finale della macchina a stati finiti che continua quindi a ciclare in questo stato fino all'arrivo di un segnale di reset che la riporta allo stato

iniziale. In questo stato il segnale di uscita `yes_sincro` viene portato a livello logico alto in modo da segnalare al sistema l'inizio del *gold code*. Questo segnale funge da reset negato per il comparatore instanziato con il *gold code* negato che da quel momento inizia ad elaborare l'ingresso. L'uscita `reset_out` è messa a livello logico alto in modo da impedire ai 31 comparatori di elaborare inutilmente il segnale , riducendo di conseguenza le commutazioni e quindi il consumo dinamico.

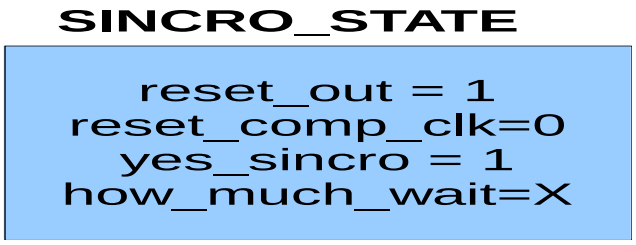

Fig 31

6.2 Prova della macchina a stati finiti di controllo.

Per capire il corretto funzionamento della macchina a stati finiti qui di seguito verranno inserite le simulazioni eseguite.

In figura 32 si può vedere il funzionamento dei primi stati della macchina a stati finiti, mentre l'ingresso `reset_in` è a livello logico alto la macchina viene inizializzata allo stato `initial_state`. Quando il segnale di `reset` torna a livello logico basso lo stato commuta passando a `wait_start_state`, in questo stato viene inizializzato il contatore a 305; successivamente, quando l'ingresso start passa a livello logico alto, la macchina attende di essere in fase col clock interno a frequenza ridotta passando così allo stato `wait_data_state` che inizia a decrementare il contatore attendendo l'elaborazione dell'ingresso da parte dei comparatori.

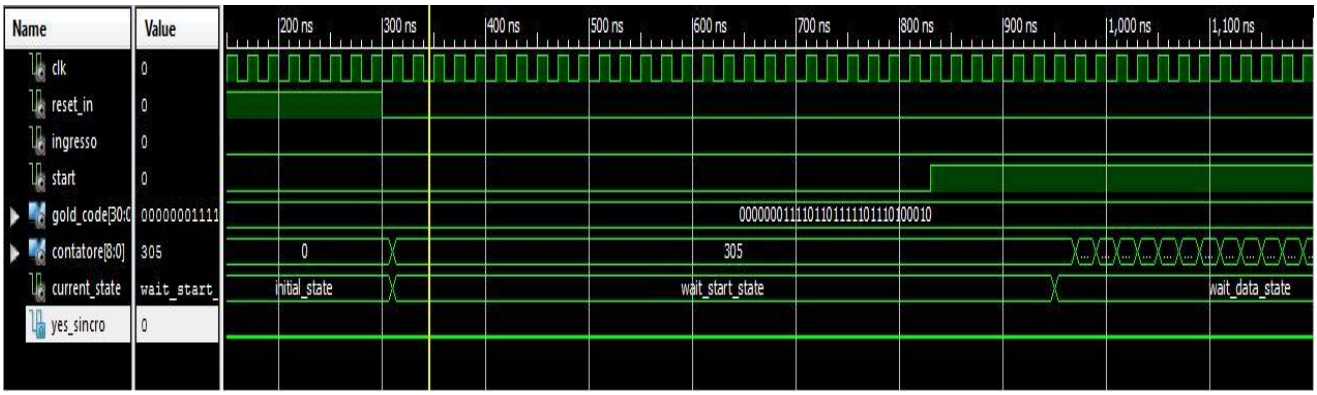

Fig 32

In figura 33 si può vedere la conclusione del conto alla rovescia del contatore che, al raggiungimento del valore 0, passa allo stato `read_data_state` che assegna il valore

corrispondente al contatore a seconda del valore dell'uscita del selettore. In questa simulazione l'ingresso era stato settato a 0 per tutto il tempo quindi nessun comparatore ha raggiunto il valore 31. Gli stati successivi sono quindi prima `reset_state` e, dopo il tempo necessario, `load_state`. Durante questo ultimo stato il contatore viene inizializzato al valore 305 e viene completata l'attesa. Lo stato successivo è `wait_data_state` da cui ricomincia il ciclo.

Fig 33

La figura 34a&b mostra cosa succede quando un comparatore raggiunge 31 e quindi quando l'uscita del selettore è diversa da 0. In figura 34a si vede che terminato il conteggio del tempo necessario ai comparatori per elaborare l'ingresso si passa allo stato `read_data` che acquisisce l'uscita del selettore e assegna di conseguenza il valore di attesa per sincronizzarci con l'inizio del *gold code* successivo. Lo stato `wait_sincro_state` attende di conseguenza il tempo necessario. Terminato il conteggio del contatore, come si può vedere nella figura 34b si passa allo stato `sincro_state` che porta a livello logico alto l'uscita `yes_sincro` che a sua volta attiva il comparatore con il *gold code* negato che attenderà le condizioni necessarie per comunicare a livello software più alto l'inizio della comunicazione.

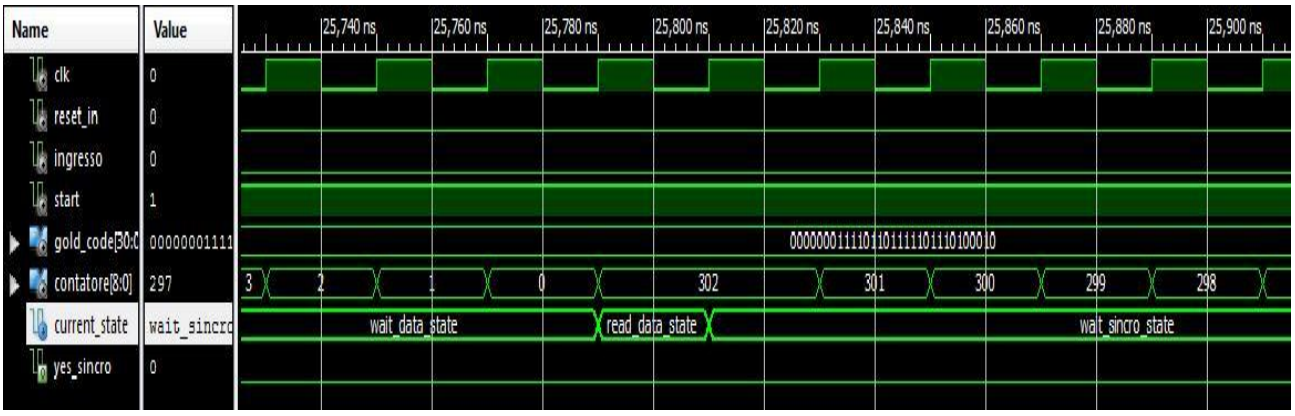

Fig 34a

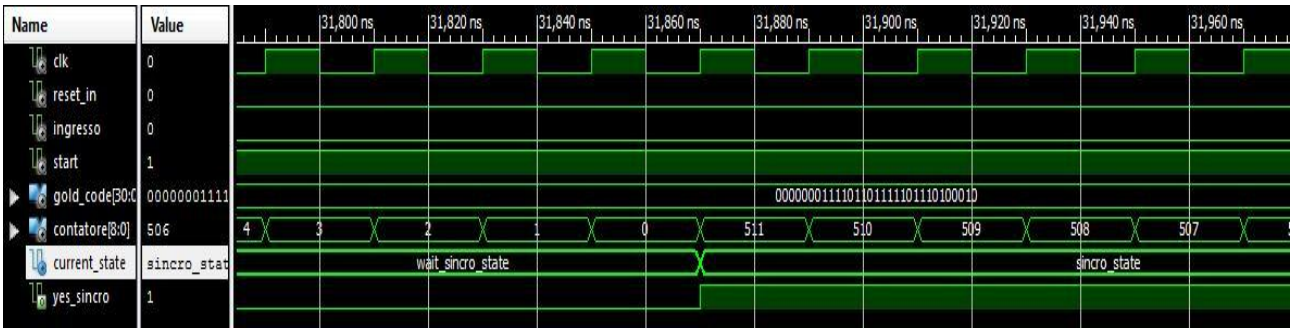

Fig 34b

6.3 Attesa del gold code negato.

All'avvenuta sincronizzazione, ovvero quando la macchina a stati finiti raggiunge lo stato `sincro_state`, il sistema si mette in attesa del *gold code* negato in ingresso. Per realizzare questa attesa si è utilizzato un circuito del tipo in figura 35.

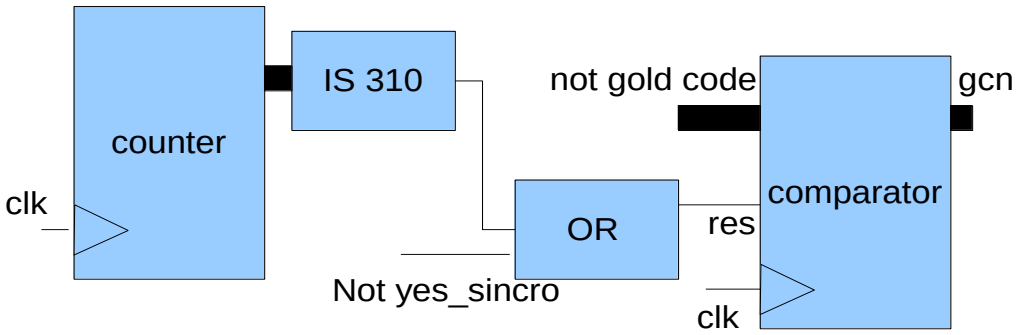

Fig 35

Si tratta di un altro circuito comparatore istanziato con il *gold code* negato, ovviamente non sfasato, che rimane resettato finché l'uscita `yes_sincro` è a valore logico basso e inoltre viene resettato ogni 310 Tclk ovvero ogni parola binaria a 31 bit. Quando si inizia a ricevere il *gold code* negato il segnale `gcn` viene incrementato fino al valore 31. L'uscita gcn viene poi portata in uscita del sistema con un'altra uscita ausiliaria formata dall'AND dei 5 bit dell'uscita `gcn` (vedi fig 36).

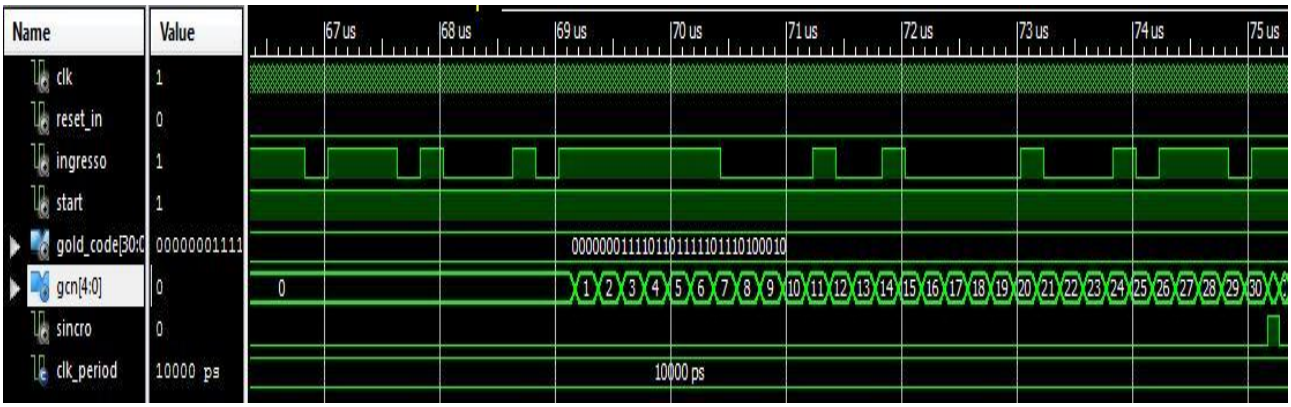

Fig.36

7 Sincronizzatore UWB-IR

7.1 Sintesi del sincronizzatore UWB.

Viene fornito in questa sezione, nella tabella 3, il prospetto riassuntivo fornito da ISE dopo l'implementazione completa del sincronizzatore UWB.

Sommario sull'utilizzo del dispositivo			
Utilizzazione della logica	***Usati***	***Disponibili***	***Utilizzazione***
Numero di flip flop delle slices	358	3840	9%
Numero di LUT a 4 ingressi	1115	3840	29%
Numero di slices occupate	604	1920	31%
• ***Numero di Slice che contengono solo logica correlata***	604	604	100%
• ***Numero di Slice che contengono logica non correlata***	0	604	0%
Numero totale di LUT a 4 ingressi	1123	3840	29%
• ***Numero di LUT usate come logica***	1114		
• ***Numero di LUT usate come pass-thru***	8		
• ***Numero di LUT usate come Shift Register***	1		
• ***Numero di IOB legate***	41	173	23%
Flip Flop IOB	1		
• ***Numero di BUFGMUX***	1	8	12%
Media del Fan Out delle reti non dedicate al clock	4,68		

Tab 3

Di particolare interesse sono i dati percentuali riguardanti l'occupazione del circuito realizzato, infatti la voce "number of occupated Slices" indica che all'interno del dispositivo vengono utilizzate solo il 31% delle slices disponibili.

La disponibilità di ulteriori risorse all'interno del dispositivo implica che, in caso di necessità, è possibile implementare eventuali ulteriori circuiti necessari al sistema anche indipendenti dal sincronizzatore stesso.

7.2 Consumo energetico.

Tramite il tool *Xilinx XPower Analyzer* è possibile valutare alcune caratteristiche del circuito come il consumo di potenza totale dall'alimentazione e le proprietà termiche del dispositivo. Vengono riportate di seguito le tabelle generate dal tool assumendo una temperatura ambientale di 25°C.

Temperatura di giunzione e temperatura massima ambientale (tabella 4):

Proprietà termiche	Temperatura di giunzione effettiva	Massima temperatura ambientale	Temperatura di giunzione
	30,9 °C/W	83,5 °C	26,5 °C

Tab 4

Consumo di potenza in watt del dispositivo totale e le componenti di consumo dinamico e statico (tabella 5).

Consumo di potenza	Consumo totale	Consumo dinamico	Consumo statico
	47mW	6mW	41mW

Tab 5

7.3 Frequenza massima di lavoro.

Nella realizzazione del sincronizzatore in fase di progettazione sono state inserite delle *time costraint* ovvero delle limitazioni temporali che sono state correttamente implementate dal tool di sintesi che nel report segnala il rispetto di tutte le limitazioni. Si è stabilito che il tempo di propagazione massimo di un *flip flop* e della logica combinatoria connessa alla sua uscita sia inferiore a 40ns in modo che il *flip flop* in cascata possa campionare il dato successivo. Si è stabilito ulteriormente che il tempo di propagazione massimo tra un pin di ingresso e il primo *flip flop* sia inferiore a 10ns. Avendo inserito dei vincoli temporali il programma ISE fornisce dei dati interessanti raccolti in tabella 6.

Limitazioni temporali	Caso peggiore	Caso migliore realizzabile
Tempo di propagazione tra pin di ingresso e il primo flip flop	SETUP 0,279ns HOLD 10,412ns	9,721ns
Tempo di propagazione tra due flip flop consecutivi	SETUP 5,598ns HOLD 0,806ns	14,402ns

Tab 6

La tabella indica i valori peggiori ottenuti durante l'implementazione, in particolare il tempo di propagazione tra dai pin di ingresso al primo *flip flop* risulta essere di 0,279 nanosecondi nel caso peggiore che soddisfa le condizioni imposte. Dal secondo vincolo otteniamo inoltre una informazione molto utile, infatti dice che nel peggiore dei casi il tempo di propagazione tra due *flip flop* è 14,402ns che corrisponde al minimo periodo di clock a cui il sincronizzatore

può lavorare. Da questo dato possiamo calcolare quindi la massima frequenza di lavoro del sincronizzatore che è l'inverso del periodo di clock minimo che risulta essere circa 69MHz.

8 Conclusioni.

L'obbiettivo prefissato da questo elaborato consisteva nella realizzazione di un circuito sincronizzatore funzionante che potesse lavorare con una frequenza minima di 50MHz su un FPGA xc3s200-4ft256. Tutte le specifiche sono state rispettate e si è ottenuta un'implementazione con delle performance molto buone per quanto riguarda consumi e frequenza di lavoro.

8.1 Problemi riscontrati.

La difficoltà principale riscontrata durante la realizzazione del sincronizzatore UWB è stata quella di interfacciare le due distinte sezioni del sincronizzatore ovvero il blocco di interfaccia con l'ingresso con il blocco di controllo, che lavoravano a frequenze diverse.

Inizialmente era stato inserito all'interno di ciascun comparatore un divisore di frequenza che metteva a disposizione un segnale di clock alternativo di frequenza inferiore sintetizzato con della logica combinatoria. Tale soluzione si è rivelata non adatta perché il software ISE non riesce a trattare il segnale generato da logica combinatoria come clock non riuscendo a risalire all'effettivo *duty cicle* e all'effettiva durata del segnale. Tale metodo non permetteva di rispettare i tempi di *HOLD* e *SETUP* dei flip flop. Inoltre le linee dedicate al clock del dispositivo non erano sufficienti e quindi non venivano utilizzate.

Per risolvere il problema è stato inserito nei comparatori il segnale di *chip enable* che abilita e disabilita il componente, questa è una soluzione circuitale ottimizzata infatti in questo modo tutte le strutture sono controllate dal clock di sistema con linee dedicate a basso tempo di propagazione ma i componenti che necessitano di un clock di frequenza inferiore hanno il segnale di *chip enable* che permette alla sezione di interfaccia con l'ingresso di lavorare ogni 10 cicli di clock come richiesto.

8.2 Possibili miglioramenti

Nelle specifiche del progetto si richiedeva che la macchina a stati finiti includesse un'altra funzione che non è stata implementata ma è stata affidata ad un livello logico più alto. La funzione richiesta era un controllo sul numero di cicli eseguiti dalla macchina a stati. Durante l'elaborazione del segnale ricevuto, nel caso nessuno dei comparatori raggiunga il valore 31, la *fsm* opera uno sfasamento di un ciclo di clock tentando di portarsi in fase con il segnale. Se la trasmissione non è riservata a noi e quindi non viene ricevuto il *gold code* corrispondente

questa operazione viene ripetuta continuamente in un ciclo infinito. Sarebbe possibile migliorare il sistema implementando una funzione aggiuntiva nella macchina a stati che esegua questa operazione al massimo 10 volte provando tutte le posizioni possibili.

Per realizzare questa funzione deve essere aggiunta un'uscita alla macchina a stati che viene portata a valore logico alto solo nello stato `load_state`, a questa uscita deve essere interfacciato un contatore che si incrementa ad ogni fronte di salita di questa uscita ovvero ad ogni passaggio per `load_state`. Quando il contatore arriva quindi a 10 viene generato un segnale di reset per la macchina a stati, in tal modo il sistema smette l'elaborazione del segnale di ingresso.

Bibliografia

1. Following the IDF: Ultra Wide Band Wireless Data Transfer Technology. *iXBT Labs - Computer Hardware In Detail.* [Online] 30 Ottobre 2002. http://ixbtlabs.com/articles2/uwb/index.html.

2. **Commission, Federal Communications.** FCC Electronic filing & Pubblic Access System. *http://hraunfoss.fcc.gov/.* [Online] aprile 2002. http://hraunfoss.fcc.gov/edocs_public/attachmatch/FCC-02-48A1.pdf.

3. **Engineering, USC Viterbi school of.** http://viterbi.usc.edu/. *WIDES - Wireless Devices and System Grops.* [Online] http://wides.usc.edu/research/ultrawideband-communications-and-location/.

4. **Maxfield, Clive.** *The Design Warrior's Guide to FPGAs.* s.l. : Elsevier, 2004.

5. **Daniele, Vogrig.** *Dispense del corso di laboratorio di elettronica digitale.* 2009.

6. FPGA. *D.E.I.S Università di Bologna.* [Online] http://www-micrel.deis.unibo.it/SMLS/corso/fpga/fpga.htm.

7. **Technologies, CORE.** FPGA design quick start guide. *http://www.1-core.com.* [Online] http://www.1-core.com/library/digital/fpga-design-tutorial/quick-start-guide.shtml.

8. **Xilinx.** *ISE Web Pack.*

9. **Emulation Lab.** emulationlab.com. [Online] http://emulationlab.com/Products.htm.

10. **Xilinx.** Spartan-3 FPGA Family Data Sheet. *www.xilinx.com.* [Online] http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf.

11. —. Spartan-3 Generation FPGA User Guide. *www.xilinx.com.* [Online] http://www.xilinx.com/support/documentation/user_guides/ug331.pdf.

12. **N.Benvenuto, R.Corvaja, T.Erseghe, and N.Laurenti.** *Communication System Foundamentals and Design Methods.* s.l. : Wiley, 2007.