Politecnico di Torino
III Facoltà di Ingegneria

UWB baseband power optimization

Low-Power system design

III Facoltà di Ingegneria
Master degree in Electronic Systems Engineering

Authors: e01GSK3n

Alessandro Colonna, Massimo Cutrupi

5 giugno 2008

Indice

CAPITOLO 1

Introduzione al protocollo UWB

Quando si parla di UWB (*UltraWide Band*) ci si riferisce a un protocollo digitale di comunicazione wireless basato sulla trasmissione di impulsi. Questa tecnologia ha il vantaggio di essere a banda larga (si arriva a trasmissioni con *bit rate* dell'ordine dei Gbit/s) e a bassa energia. Grazie a queste proprietà i trasmettitori UWB tipicamente non interferiscono in maniera significativa con altri sistemi di comunicazione a banda più stretta.
Chiaramente in questa breve introduzione non si ha la pretesa di chiarire i dettagli teorici relativi alla tecnologia UWB, si introdurranno invece brevemente gli aspetti tecnico-implementativi che si rifletteranno sulle scelte architetturali esposte nei capitoli successivi.

1.1 Formato di trama

Come già accennato la modulazione UWB è basata sulla trasmissione di impulsi. Nella fattispecie l'informazione relativa ad ogni simbolo sarà codificata nella posizione in cui l'impulso ad esso associato è collocato all'interno della trama temporale, si parla quindi di *Pulse Position Modulation* (PPM).

Chiaramente una trasmissione di questo tipo è significativamente affetta da problemi relativi ai cammini multipli, che aggiungono al segnale trasmesso delle copie sfasate dello stesso. Da questo tipo di disturbi così come dal rumore del canale ci si difende ripetendo più volte la comunicazione e cioè aggiungendo informazione ridondante.

Una ulteriore caratteristica del protocollo UWB è la sua elasticità, esistono infatti dei gradi di libertà nella scelta del protocollo. Si parla quindi di *M-orthogonal Pulse Position Modulation*, in questo caso dunque si avranno a disposizione i seguenti due parametri di progetto:

- M: identifica il numero di posizioni possibili in cui l'impulso può essere collocato e quindi, in un certo senso, il quanto di informazione associato a un simbolo.

- N: detto *repetition code*, identifica il numero di volte in cui si ripete la trasmissione di un singolo simbolo.

In figura 1.1 è rappresentato un esempio di trasmissione dei simboli 0 e 3 con M=4 e N=2.

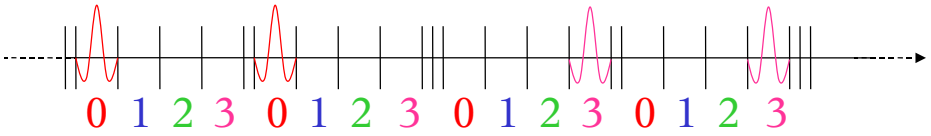

Figura 1.1: esempio di trasmissione UWB

CAPITOLO 2

Specifiche e obiettivi

Lo scopo di questa tesina è la realizzazione di un progetto della parte digitale di un ricevitore UWB. L'obiettivo che ci si è posto è essenzialmente quello di realizzare un progetto che sfrutti tutte le tecniche classiche relative alla riduzione del consumo di potenza e di unirle ad alcune soluzioni ad hoc al fine di ottenere un oggetto dalle elevate prestazioni, a bassa potenza e che sia soprattutto generico e cioè facilmente adattabile a tutte le possibili scelte applicative future.

2.1 Ricevitore UWB

Lo schema del ricevitore è rappresentato in figura 2.1, come si nota esso consta di un primo blocco di amplificazione, filtraggio e condizionamento analogico la cui uscita è l'integrale del valore assoluto del segnale ricevuto, quindi di un ADC e infine del *digital backend* che completa il processo di demodulazione originando il segnale digitale w.

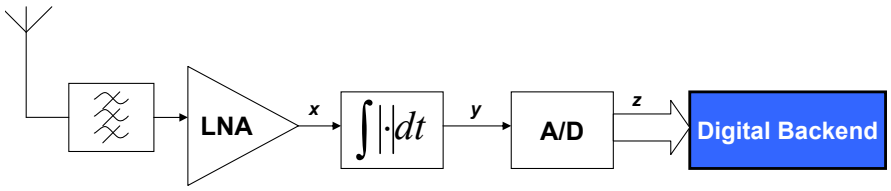

Figura 2.1: Schema a blocchi del ricevitore

In figura 2.2 è rappresentato un esempio di forme d'onda in assenza di rumore nel canale e di problemi di *multipath*.

Tuttavia il segnale ricevuto, in condizioni normali di funzionamento, non avrà l'aspetto di figura 2.2 (segnale x), ma sarà una sovrapposizione di impulsi ritardati più volte, sommersa nel rumore elettromagnetico. I blocchi analogici origineranno dunque un segnale digitale z+n come quello rappresentato in figura 2.3.

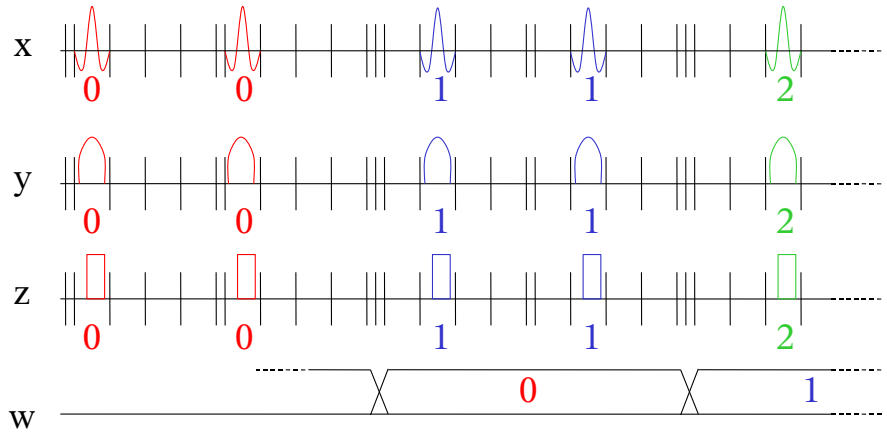

Figura 2.2: Forme d'onda relative ai blocchi del ricevitore

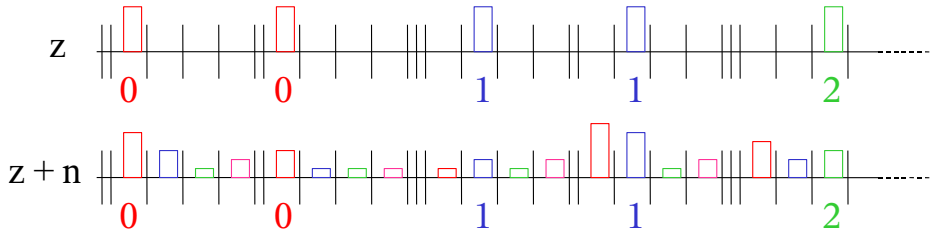

Figura 2.3: Ingresso del *digital backend* in presenza di rumore

Il *digital backend* quindi elabora i dati in uscita all'ADC proteggendo per quanto possibile la trasmissione da eventuali errori dovuti al rumore. In sostanza questo blocco svolge il ruolo di decisore individuando lo slot temporale in cui è stato trasmesso l'impulso con maggiore probabilita. Per fare questo semplicemente applica il seguente algoritmo.

$$w = \arg\left(\max_{m\in[0,m-1]} \sum_{n=0}^{N-1} z[m + n \cdot M]\right) \tag{2.1}$$

L'algoritmo è molto intuitivo e consiste nel trovare lo slot temporale in cui l'ingresso z raggiunge il valore massimo considerando le somme degli N valori assunti negli M slot per N ripetizioni.

2.2 Oggetto del *design*

Come già accennato, la porzione del sistema che è stata presa in esame è tutta digitale. Essa è rappresentata in figura 2.4 e consiste di più blocchi: il primo lo abbiamo già visto e non fa altro che applicare l'algoritmo di formula (2.1) fornendo così un'uscita w. Questo segnale è poi instradato su di un bus per mezzo di un blocco di codifica

che ha lo scopo di minimizzare la *switching activity* sul bus evitando così di caricare e scaricare frequentemente il carico.

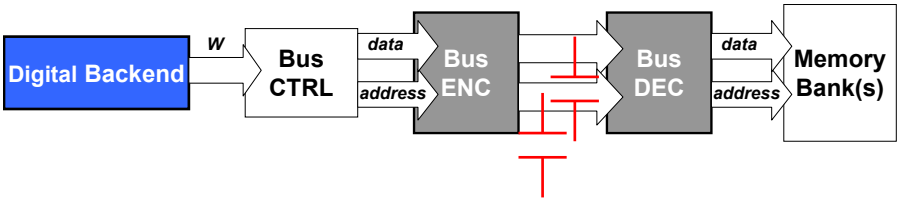

Figura 2.4: Schema a blocchi della struttura realizzata

I blocchi che seguono quindi decodificano il dato rimuovendo le eventuali ridondanze e lo instradano in maniera opportuna in una memoria che può essere partizionata in più banchi. Il nostro lavoro è consistito nel progettare i primi quattro blocchi, nell'ottimizzarli e nel fare alcune considerazioni su come gestire la memorizzazione dei dati. Quest'ultime considerazioni hanno portato infine anche alla realizzazione di una architettura non ottimizzata, ma generica di selezione dei banchi, di cui faremo menzione negli ultimi paragrafi. I parametri che si è deciso di mantenere generici, che cioè sono stati tradotti in `generic` del codice vhdl sono i seguenti:

- M : lo abbiamo già visto e rappresenta il numero di slot possibili che l'impulso UWB può avere per ciascun simbolo
- N : *repetition code*
- NQ : numero di bit su cui è rappresentato il dato di uscita dell'ADC che corrisonde chiaramente al numero di bit di quantizzazione dello stesso.
- NBANKS : numero di banchi
- NSYMBW : numero di simboli memorizzati in una word di memoria.

Esistono poi altri generic che insieme a questi sono definiti nei package `constants_def.vhd` e `array_def.vhd`, ma si tratta semplicemente di combinazioni algebriche dei precedenti.

CAPITOLO 3

Architettura del *digital backend*

L'algoritmo implementato dal *digital backend* è un algoritmo con memoria che pertanto intrinsicamente richiede un determinato dispendio di energia relativo alla memorizzazione dei dati parziali. Tuttavia tra le varie scelte implementative possibili ve ne sono sicuramente alcune che è possibile escludere sin dall'inizio ponendosi nell'ottica di un progetto a basso consumo. È senz'altro da escludere infatti la strategia di memorizzare inizialmente tutti i dati per poi effettuare le somme perché richiederebbe un uso estensivo di flip-flop nonché l'uso di più adder in parallelo con immaginabili effetti nefasti sul consumo di potenza.

3.1 Architettura di partenza

Proprio al fine di evitare i sopracitati problemi si è partiti da uno schema che minimizza la quantità di registri. Come si nota infatti in figura 3.1 i dati parziali delle somme, i minimi indispensabili, sono memorizzati in maniera ordinata nei registi interposti tra multiplexer e demultiplexer. Di lì sono di volta in volta prelevati per effettuare le somme relative all'ennesima ripetizione della trasmissione. Chiaramente all'inizio della trasmissione di ciascun simbolo il contenuto di questi registri deve essere resettato.

Lo stadio max_detector che segue preleva i dati di uscita dal mux a cui è collegato che saranno nella fattispecie i risultati finali delle somme contenuti nei registri al termine degli N cicli di ritrasmissione. Il blocco in questione quindi di volta in volta aggiornerà, se opportuno, il registro contenente il valore del massimo locale. L'unità funzionale adiacente associa al massimo locale il codice ad esso relativo in un modo che per il momento non approfondiamo.

Una prima osservazione che si può fare è che in questa soluzione ancora soltanto abbozzata e non ottimizzata si usano due demultiplexer quando l'ingresso dell'adder potrebbe essere prelevato direttamente a valle dell'altro demux. Si nota inoltre che l'algoritmo ciclico di comparazione potrebbe essere modificato a favore di un'altra soluzione basata su una struttura ad albero binario di comparatori. Questa tecnica

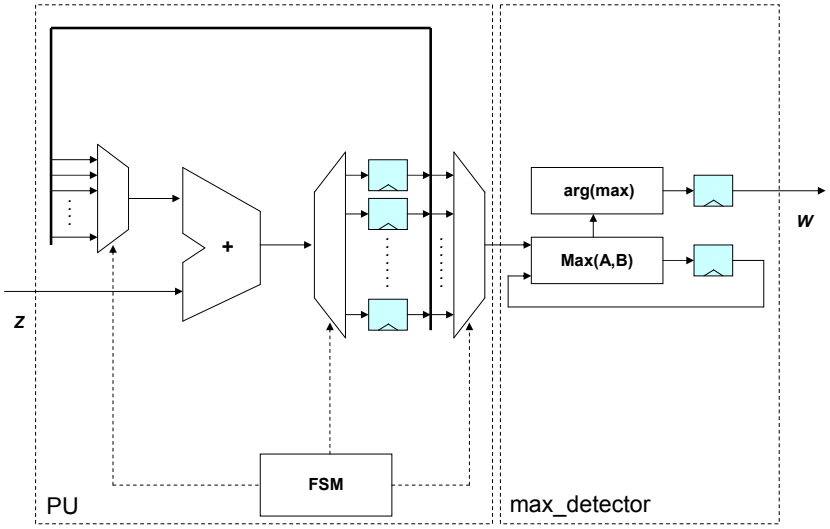

Figura 3.1: Schema a blocchi di partenza

alternativa avrebbe il vantaggio di ridurre il numero di comparazioni necessarie per ogni simbolo in uscita, ma ha lo svantaggio di essere strutturata su $\lceil \log_2(M) \rceil$ livelli. Ciò può comportare evidentemente un maggiore ritardo e conseguentemente, nel caso questo rappresenti il percorso combinatorio massimo, ostacolare il successivo abbassamento della tensione di alimentazione. A questo inconveniente si può rimediare pipelinando opportunamente la struttura.
Questa alternativa può essere valutata in vista del progetto di una specifica realizzazione, i suoi vantaggi valgono infatti per specifici valori di M, N e NQ e la comparazione in termini genterali non è stata affrontata poiché particolarmente onerosa in termini di tempo di simulazione soprattutto.

A partire dalla struttura di figura 3.1 sono state effettuate diverse modifiche architetturali e ottimizzazioni che hanno portato a degli schemi piuttosto complessi che analizzeremo nei prossimi paragrafi.

3.2 Parallelizzazione del *digital backend*

Una tecnica tipicamente utilizzata per ridurre il consumo di potenza è quella di parallelizzare unità logiche al fine di ottenere un vantaggio in termini di *timing* e sfruttare conseguentemente lo *slack* rispetto alle tempistiche che impongono le specifiche per ridurre la tensione di alimentazione. Questo *slack* è dovuto al fatto che le due unità logiche ottenute sono sincronizzate a parità di *throughput* con una frequenza di clock dimezzata rispetto a quella di partenza (Nel nostro caso parleremo del segnale **ck_slow**).

In questo modo ciascuna delle due unità ha il doppio del tempo per effettuare le stesse elaborazioni di prima se non talvolta meno.

Nel nostro caso, in sostanza, il blocco *positive edge triggered* analizzerà gli slot dispari, mentre il blocco *negative edge triggered* si occuperà di quelli pari. Entrambi i blocchi quindi indicheranno nel loro set di slot qual è il massimo locale.

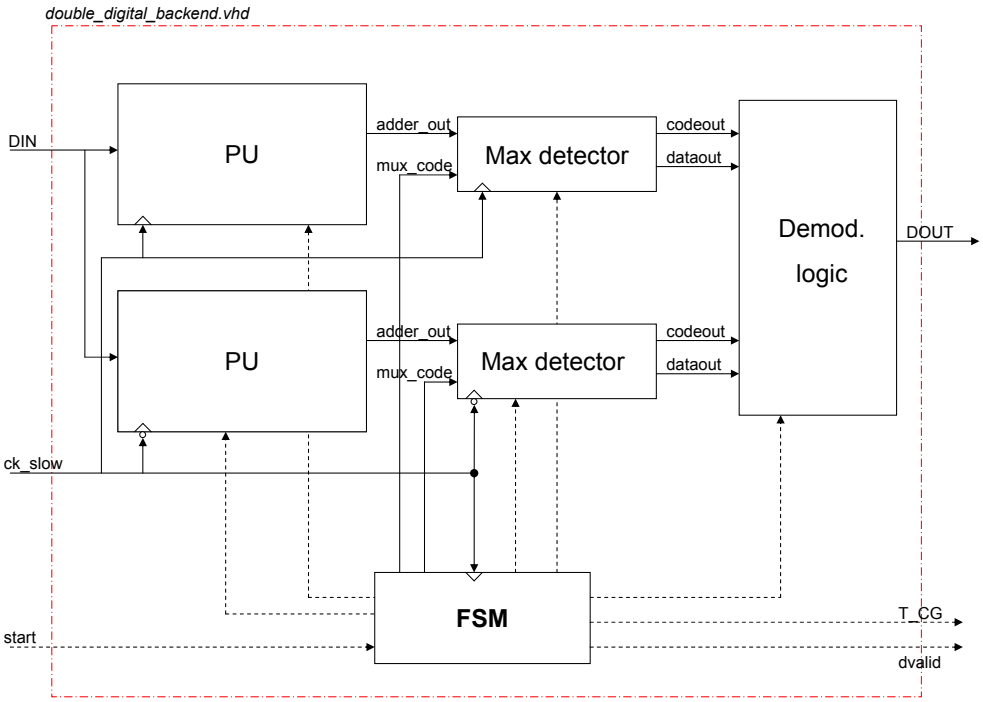

Figura 3.2: Schema a blocchi del circuito parallelizato

Ci si accorge subito, dunque, di come il nostro caso non sia dei più tipici, difatti, come si nota dallo schema a blocchi in figura 3.2, in uscita alla struttura parallelizzata non utilizziamo un multiplexer per ricongiungere il flusso di dati, ma c'è un ulteriore comparatore che determina tra i due massimi locali quale sia quello globale, fornendo così in uscita l'indice relativo ad esso come previsto dall'algoritmo. Quest'ultima unità funzionale riceve infatti due segnali denominati **codeout** che rappresentano gli indici relativi ai due massimi locali. Questi indici sono generati internamente dai due **max_detector** e quindi non appartengono a insiemi disgiunti. Per risolvere questo inconveniente basta associare al massimo assoluto il suo indice di partenza concatenato a un bit in posizione meno significativa a 0 o a 1 rispettivamente nel caso in cui il massimo assoluto sia associato a uno slot pari o dispari.

Si noti inoltre come l'*overhead* in termini di area sia minore del previsto poiché non è stato necessario raddoppiare l'FSM. I segnali di controllo relativi ai blocchi *negative edge triggered*, infatti, corrispondono esattamente ai segnali dei blocchi *positive edge triggered* ritardati di mezzo colpo di clock e cioè semplicemente campionati sul fronte di discesa.

Infine si consideri come la scelta di questa architettura ponga un vincolo sulla scelta di M. Difatti, applicando questa soluzione, non è più possibile avere un numero di slot dispari, necessità che praticamente non si presenta mai. Tuttavia, anche nel remoto caso in cui questa necessità si presenti, è possibile mantenere la struttura parallela al costo di qualche modifica all'architettura di controllo.

3.3 Architettura della *Processing Unit*

L'architettura della *Processing Unit* è rappresentata in figura 3.3, come già accennato presenta un multiplexer in meno rispetto all'architettura di partenza di figura 3.1 e questo anche perché si è scelto di prelevare i dati di uscita direttamente a valle dell'adder permettendo così di propagare inutilmente i risultati finali delle somme nel demux, di memorizzarli e di farli poi passare per il mux. Questa soluzione architetturale permette quindi di evitare transizioni in quest'ultime unità per 1/N colpi di clock grazie allo sbarramento del latch a monte del demux e a una congiunta stabilizzazione dei controlli del demux ad opera dell'FSM.

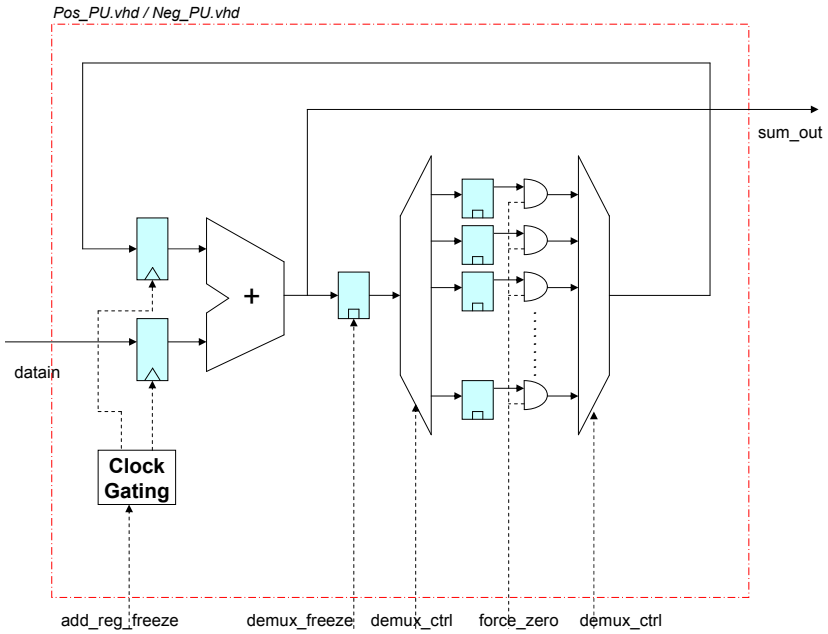

Figura 3.3: Architettura della *Processing Unit*

Un'ulteriore modifica è stata quella di utilizzare dei latch al posto di una schiera di FF-D tra demux e mux. In pratica ogni latch è trasparente quando si aggiorna la relativa uscita del demux ed è quindi stabile quando il mux ne preleva l'uscita, infatti il controllo del demux è sempre pari a quello del mux incrementato di uno.
Il vantaggio di questa scelta non risiede esclusivamente nel minore consumo dei latch rispetto ai normali flip flop D, ma anche nel minor *overhead* di controllo. I segnali di

enable di questi latch sono infatti "autogenerati" dal demux a monte, questo perché il blocco demux più latch è ottenuto molto semplicemente sintetizzando il seguente codice vhdl:

```
demux_p: process (ser_in, ctrl, reset)

begin  -- process demux_p

       if reset = '0' then
         par_out<=(others=>(others=>'0'));
       else
         par_out( conv_integer(ctrl) ) <= ser_in;
       end if;

end process demux_p;
```

Questo codice è tipicamente citato come esempio di codice vhdl errato, infatti rappresenta un errore tipico il fatto di dimenticare di assegnare delle uscite. Nel nostro caso tuttavia è stato scelto deliberatamente di tenere memoria dei dati in uscita al demux quando essi non sono in aggiornamento. È necessario però constatare come questa soluzione non abbia alcun carattere generale e sia molto comoda e sensata solo in casi analoghi a questo.

Si noti infine che all'ingresso dell'adder si applica il *clock gating* e come l'azzeramento di un operando dell'adder all'inizio della ricezione di ogni simbolo sia effettuato tramite una schiera di porte **AND**.

3.4 Architettura del *Max detector*

L'architettura del *Max detector* è rappresentata in figura 3.4. Questa unità funzionale ha la duplice funzione di valutare il massimo tra i risultati definitivi delle somme e di associarne il codice relativo. Come si nota infatti esso presenta due uscite: **dataout** e **codeout**. La prima rappresenta il massimo locale che volta per volta è identificato dal comparatore, la seconda invece identifica l'indice ad esso associato.

Si noti ora come la struttura preveda un meccanismo di *precomputation*, i comparatori infatti si prestano particolarmente all'uso di questa tecnica poiché basta valutare il bit più significativo di ciascun operando per evitare con buona probabilità di dover analizzare anche i bit meno significativi.
Nel nostro caso, in particolare, tutte le volte che i bit più significativi sono diversi tra loro (porta **XOR**) è inutile effettuare commutazioni su quelli meno significativi. La probabilità che sia sufficiente la sola precomputation dipende chiaramente dalla statistica degli ingressi, ma nel nostro caso possiamo affermare che essa è pari a 0,25 in condizioni di assenza di rumore e statistica dei simboli uniforme. Questo poiché in queste condizioni solo una delle somme finali avrà il bit più significativo a 1 e quindi

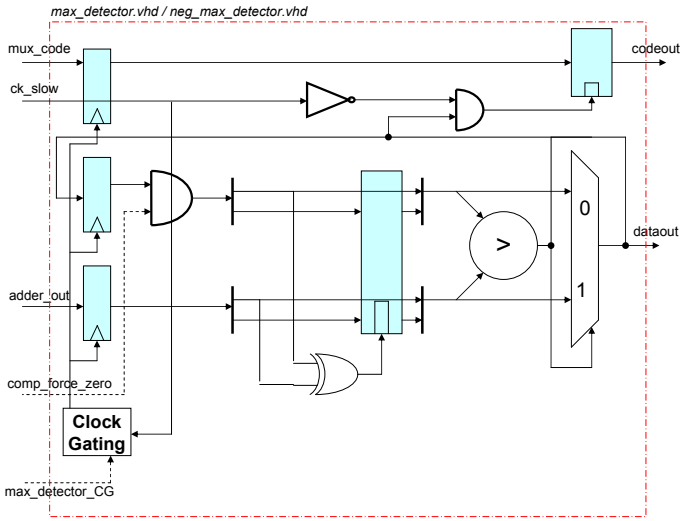

Figura 3.4: Architettura del *Max detector*

per ogni simbolo solo uno dei due *max detector* eseguirà precomputation in media il 50% delle volte.
Si nota quindi come la scelta di parallelizzare la struttura faccia perdere parte dei vantaggi derivanti dalla *precomputation*.

Per quanto riguarda poi il segnale **codeout**, esso è generato a partire del segnale **mux_code**. Esso corrisponde alla versione in codifica binaria semplice del segnale di controllo **mux_ctrl** della PU (figura 3.3) che invece è codificato gray come vedremo in seguito. Nella fattispecie il segnale **mux_code** viene aggiornato quando il massimo locale cambia e cioè tutte le volte che il controllo del mux va a '1'.

3.5 Macchina a stati di controllo

La logica di controllo consiste in una macchina a stati piuttosto complessa. Essa consta di due *counter*. Il più veloce (sincrono con **ck_slow**) conta fino ad M-1 in codifica binaria e contiene un convertitore gray in modo da ottenere dei segnali di controllo di mux e demux con una sola transizione per colpo di **ck_slow**. Il counter più lento invece tiene il conto delle ripetizioni della trasmissione e pertanto conta fino a N-1 con una cadenza M volte più lenta del precedente.
Chiaramente il primo counter ha delle transizioni piuttosto veloci che comportano un consumo di potenza non trascurabile.

La restante parte dei segnali di controllo è su un bit ed è ottenuta semplicemente con dei comparatori che segnalano ciascuno un preciso stato dei counter e quindi della FSM.

CAPITOLO 4

La codifica sul bus

4.1 Il sistema bus

Dopo aver definito l'architettura base del nostro ricevitore UWB ci si è concentrati sull'ottimizzazione dal punto di vista della potenza del bus. Come sappiamo i bus sono uno dei blocchi principali di un sistema di elaborazione delle informazioni, connettono molte volte blocchi funzionali che elaborano una grande quantità di dati a frequenze a volte piuttosto elevate (come nel nostro caso).
Nel sistema preso in esame il bus è utilizzato come interfaccia tra il digital backend e la memoria dati come mostrato nella figura sottostante:

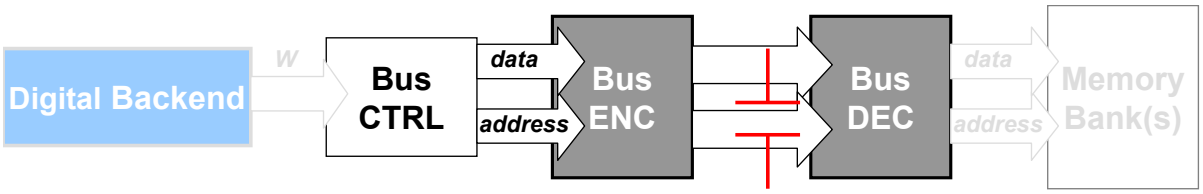

Figura 4.1: Schema generale dell'*UWB baseband processing*

Questo blocco è quindi molto critico sia dal punto di vista delle capacità in gioco (soprattutto nel caso la memoria sia localizzata *off-chip*) sia dal punto della switching activity delle linee che trasmettono dati ad una frequenza che a seconda dei valori specifici del protocollo (frequenza della portante, M ed N) può essere molto alta. Conoscendo la portante della modulazione (nel nostro caso ad esempio $f_{carrier} = 1GHz$) si può ricavare la frequenza di variazione dei dati attraverso la seguente formula:

$$f_{bus} = f_{SRAM} = \frac{f_{carrier}}{N \cdot M} \tag{4.1}$$

A seconda di varie specifiche (imposte da trasmettitore, canale di comunicazione ecc.) la frequenza del segnale sul bus potrà variare in un range abbastanza grande. Ad

esempio considerando M=2 ed N=2 (valori minimi utilizzati) si ha una frequenza di commutazione del dato sul *bus* di 250 MHz mentre per altri valori la frequenza può scendere anche sotto i 50 MHz.

Ci sono poi da considerare aspetti tecnologici legati alla effettiva localizzazione del bus all'interno del sistema di elaborazione. Tipicamente (ad esempio in un dispositivo portatile) la memoria sarà localizzata all'interno di un *SoC (System on Chip)* o *SoP (System on Package)* dove le capacità delle linee sono in generale più contenute rispetto ad una realizzazione su *board*. Nei primi due casi, però, si potrebbero avere frequenze di variazione del segnale più grandi rispetto al secondo quindi è necessario trovare (se possibile) una codifica il più possibile indipendente da vincoli tecnologici e di sistema.

Come prima cosa si è notato come in questo canale siano utilizzati sia indirizzi che dati: c'è bisogno quindi di una doppia ottimizzazione sotto questi due punti di vista. Si è scelto quindi come prima ottimizzazione di non trasmettere gli indirizzi sul bus ma di generarli direttamente sul circuito integrato di memoria per due motivi principali:

- tipicamente in sistemi elettronici di questo tipo la memoria posta a valle dei ricevitore digitale è utilizzata come un buffer di dati che verranno processati o filtrati da un microprocessore.

- La trasmissione avviene a *burst* poiché il ricevitore ha una velocità di scrittura superiore rispetto a quella di lettura del microprocessore. La memoria funge quindi sa semplice FIFO ed è un blocco di solito piuttosto piccolo (pochi kB) di pochi indirizzi (10 bit per 8kB).

Si noti tuttavia che si sarebbero potute adottare anche altre strategie per il bus indirizzi, ad esempio una codifica T0 proprio perché la scrittura avviene in modo sequenziale. In questa soluzione sulla linea di bus non ci sarebbero state commutazioni pur rimanendo l'*overhead* del circuito di ricezione(come nel nostro caso).

La parte critica è risultata essere ovviamente quella relativa alla trasmissione dei dati che, essendo scorrelati, non è possibile caratterizzare dal punto di vista probabilistico (come si fa ad esempio per i sistemi embedded dove si effettua un *profiling* del codice).
Riflettendo su una possibile realizzazione si è notato come il numero di linee minimo per trasmettere il dato demodulato in funzione del numero di simboli M sia:

$$N_{bit} = N_{linee} = logM \tag{4.2}$$

C'è però da considerare che in protcolli UWB (come in generale in sistemi di trasmissione ad alta velocità) difficilmente si ha a che fare con costellazioni con un numero grande di simboli.

Nel nostro caso quindi si è deciso di adottare semplicemente una codifica ridondante dove il dato trasmesso sul bus B(t) all'instante t è rappresentato su M bit e piuttosto che su $\log_2 M$ come nel caso del binario classico con una tecnica di tipo *one-hot* a

transizione, si tratta in pratica di codificare ogni simbolo in funzione della posizione in cui si genera una transizione.

Tra le soluzioni possibili sono state escluse il *BI* (*Bus Inverted*) e il *T0-BI* (*T0-Bus Inverted*) principalmente per l'elevato overhead di area del ricevitore e del trasmettitore e per le prestazioni minori in termini di numero di transizioni per dato.

L'unico inconveniente di questa soluzione può essere considerato l'elevata ridondanza per valori di M elevati (portare fuori da un circuito integrato più di 32/64 linee può far aumentare i costi del sistema e far diminuire la sua affidabiltà). Nel caso invece di una realizzazione con un SoC o un SoP tutto questo non costituisce un particolare problema. Tuttavia osserviamo che, nel caso ce ne fosse bisogno, si possono codificare i rimanenti bit in modo classico binario. Infine si consideri come, pur avendo questi problemi di natura tecnologica o economica, questo tipo di codifica presente altri vantaggi piuttosto rilevanti:

- questo circuito limita la potenza di picco assorbita dal trasmettitore del bus poichè garantisce un *upper bound* di transizioni per dato trasmesso (che nel nostro caso è uguale all'unità).In questo modo si aumenta la *reliability* dell'intero sistema poichè si evitano picchi elevati di assorbimento sulle linee di alimentazione con ricadute apesanti anche sul rapporto segnale rumore. Questo problema è particolarmente evidentente in bus *off chip* in cui si pilotano linee con capacità più grandi.

- Inoltre questo tipo di codifica non soffre (proprio perchè si garantisce una sola commutazione per dato) del problema del *dinamic delay* in cui commutazioni temporanee sulla linea provocano variazioni anche del $20 - 30\%$ dei ritardi di propagazione. In questo caso quindi la trasmissione potrà avere frequenze maggiori rispetto ad una codifica classica in cui sarà necessario disallineare i fronti per evitare questo problema.

4.2 Il trasmettitore

Lo schema a livello *RTL* del circuito di trasmissione è mostrato nella figura 4.2. Si è scelto di indicare il nome del file (in alto a sinistra di ogni blocco) e i nomi veri dei segnali utilizzati (con il loro parallelismo in numero di bit) nel nostro design per avere un riferimento preciso in futuro se mai fosse riutilizzato il nostro progetto. Per quanto riguarda i bus sono stati tracciati con linee più marcate mentre invece sono stati omessi i segnali di reset per non complicare troppo gli schemi.

Come notiamo dalla figura 4.2 il trasmettitore è composto semplicemente da un *Flip-Flop* di tipo T (il *Flip-Flop D* più l'inverter) comandato da un segnale di clock gating che è l'AND logico tra tre segnali ben distinti:

- **dvalid**: è il segnale, opportunamente ritardato, generato dalla FSM del ricevitore *UWB*. All'ingresso di quest'ultimo infatti è presente un segnale **start** che indica l'inizio della ricezione. Dopo $M \cdot N$ colpi di clock questo segnale è portato al livello alto per permettere al trsmettitore di iniziare la comunicazione sul bus poichè da questo momento in poi saranno presenti i dati ad una frequenza fissa.
- **T CG**: il segnale di strobe che abilita ogni $M \cdot N$ colpi di clock il flip flop T che commuterà o meno a seconda dell condizioni operative.
- **DATA**: il dato vero e proprio che, codificato in *one hot* semplice tramite un decoder, setta il *Flip-Flop T* in commutazione se a '1' o in memoria se a '0'.

Si noti infine che quando non si ha la necessità di instradare dati sul bus (**dvalid** a '0') non si genera alcuna transizione su nessuna linea. In questo modo si è evitato di trasmettere inutilmente il segnale **dvalid** sfruttando una delle tante configurazioni proibite che abbiamo grazie alla ridondanza.

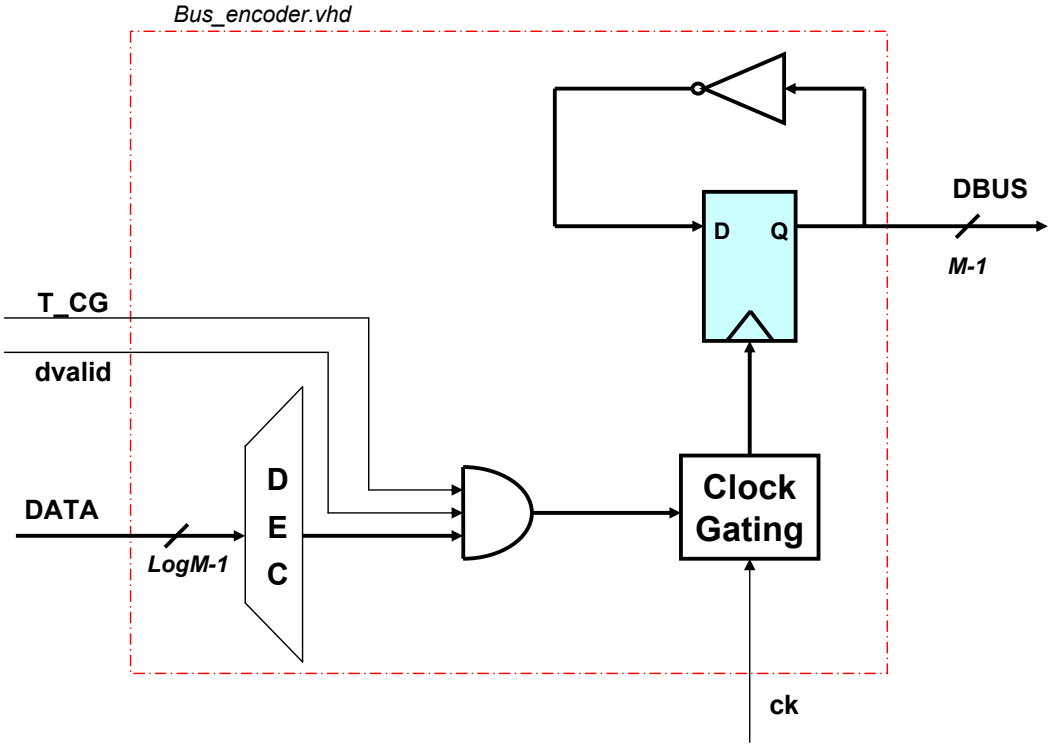

Figura 4.2: Schema a livello *RTL* del trsmettitore

4.3 Il ricevitore

Lo schema a livello *RTL* del circuito di ricezione è mostrato nella figura 4.3. Alla luce delle precedentio considerazioni per ricevere il dato corrttamente è necessario effettuare

una operazione di decorrelazione:

$$X(t) = B(t) \oplus B(t-1) \tag{4.3}$$

dove $B(t)$ e $B(t-1)$ sono rispettivamente il dato codificato al tempo t e t-1.

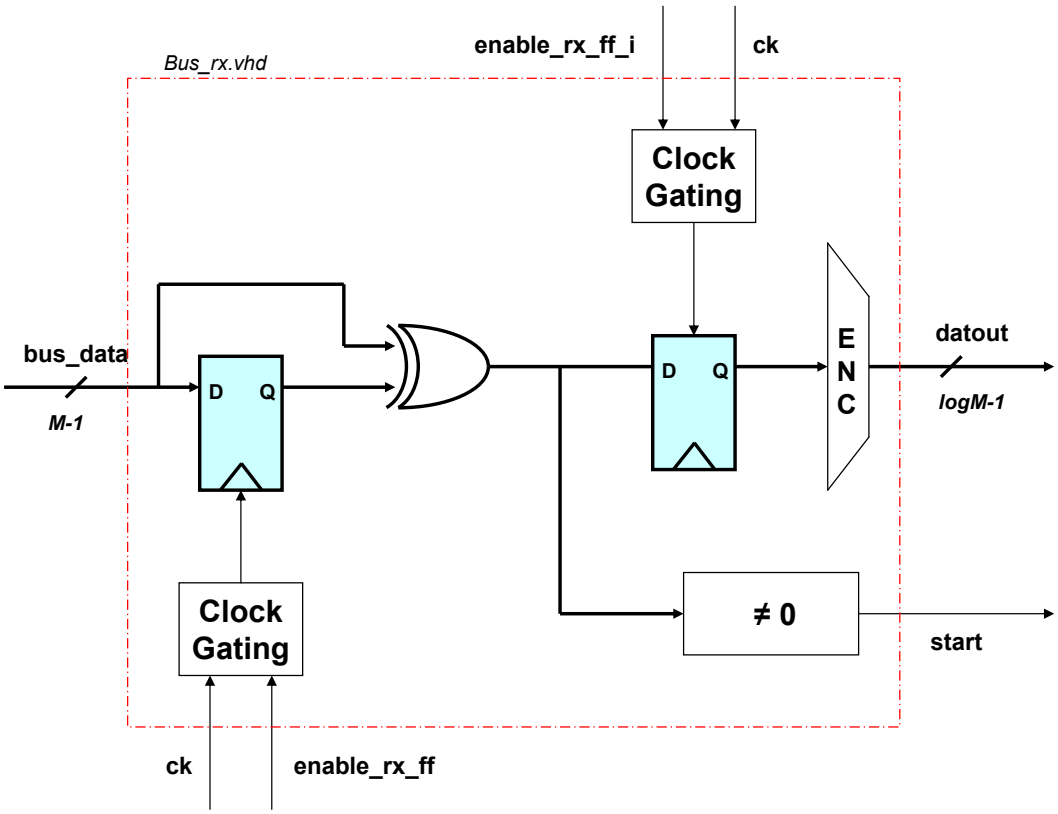

Figura 4.3: Schema a livello *RTL* del ricevitore

All'inizio della comunicazione, quando non c'è nessuna transizione sul bus, l'uscita dell' **EXOR** è settata a 0 e conseguentemente il *Flip-Flop* sarà disabilitato per mezzo del comparatore di disuguaglianza. Quest'ultimo, in queste condizioni, disabilita anche il *Flip-Flop* a valle del decoder (attraverso il segnale **enable_rx_ff_i**) e infine mediante l'uscita **start** disabiliterà tutta la circuiteria di controllo della memoria e di generazione degli indirizzi a valle che in queso modo conserverà l'ultimo dato ed indirizzo senza consumare potenza.

Alla prima trasmissione il segnale **start** salirà ad uno e l' FSM di controllo della memoria e del ricevitore si abiliterà per ricevere i dati e smistarli agli opportuni banchi di memorizzazione secondo opportune strategie di impacchettamento dei dati che spiegheremo in dettaglio nel prossimo capitolo.

4.4 Un esempio di trasmissione e ricezione sul bus

Per spiegare meglio il funzionamento del bus e della codifica utilizzata si propone un esempio tipico di trasmissione (con $f_{carrier} = 1GHz$). Si suppone per esempio di trasmettere la seguente sequenza di simboli (supponendo $M = 8$, $N = 2$, $N_Q = 4$):

$$S_{symbol} = \{7, 2, 2, 2, 2, 3, 6, 4\} \tag{4.4}$$

Le waveform dei principali segnali di ingresso e di uscita dei blocchi di trasmissione e ricezione sono mostarti nella figura 4.4. Il primo segnale in alto corrisponde all'ingresso del *digital backend*, notiamo come la modulazione M-PPM sia effettivamente applicata all'ingresso del nostro ricevitore UWB. Più in basso sono visualizzati i due clock del design (nel nostro caso $ck = 1GHz$ e $ck\_slow = 500MHz$).
Quando il segnale **start** è asserito il digital backend inizia a processare i dati mentre sul bus è ancora presente la codifica proibita. Dopo esattamente $M \cdot N$ colpi di clock il segnale **dvalid** abilita il trasmettitore che inzia a codificare i dati e a mandarli sul bus. Proprio come ci aspettavamo l'uscita del bus (**DBUS**) ha esattamente una commutazione per dato trasmesso e il ricevitore interpreta correttamente i dati (**data_out**).
Anche nel caso si voglia ritrasmettere lo stesso dato più volte di seguito ci sarà comunque una transizione sulla linea di indice pari al dato che si vuole trasmettere. Per esempio, nel caso del secondo e terzo dato trasmesso (2,2) si avrà

$$DBUS(T = 2 \cdot T_{CK}) = 10000100$$
$$DBUS(T = 3 \cdot T_{CK}) = 10000000$$

dove T_{CK} è l'inverso dell'equazione 4.1.

Notiamo inoltre che il segnale **start_i** è affetto da glitch. Per questo motivo è stato inserito un registro a monte dell'encoder che sincronizza il segnale. In questo modo non solo si evitano problemi di ricezione dei dati ma si bloccano questi impulsi evitandone la propagazione verso i blocchi combinatori che seguono diminuendo così la potenza dissipata.

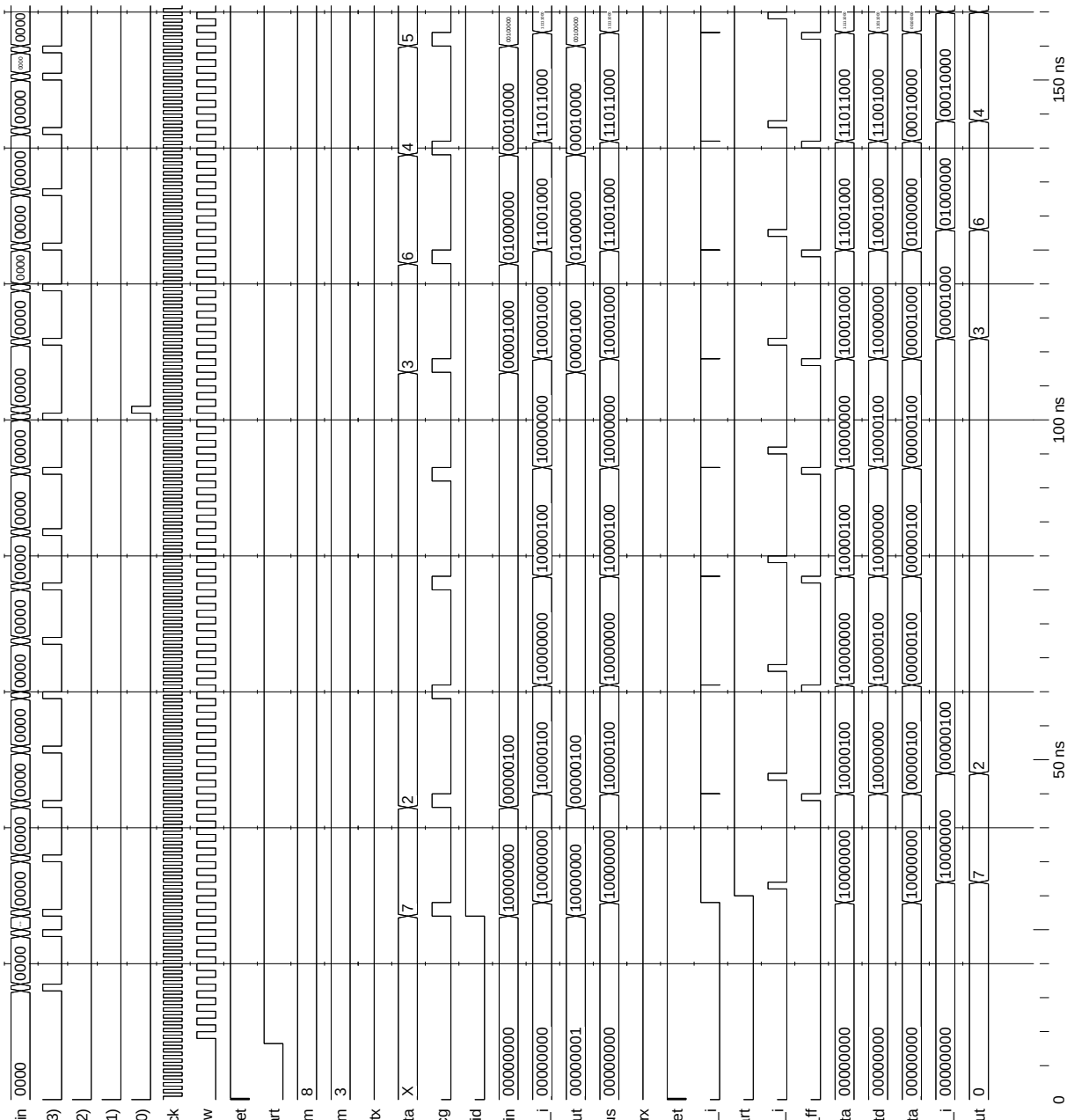

Figura 4.4: *Waveform* del sistema ricevitore-trasmettitore

CAPITOLO 5

La memoria

5.1 I modelli

Come ultima parte di questo progetto ci si è prefissi di ottimizzare dal punto di vista della potenza la gestione della memoria. Il modello fornitoci è composto di due blocchi principali:

- un file VHDL contenente l'implementazione behavioural di una memoria con dimensione di dati e indirizzi generici
- uno *spreadsheet* che descrive tramite modelli matematici approssimati il consumo di potenza dell'intero sistema di *baseband processing* digitale (ricevitore UWB, trasmettitore-ricevitore di bus e memoria).

Quest'ultimo è un modello non simulabile tipicamente utilizzato a livello di ottmizzazione di sistema. Pur essendo abbastanza grossolano si è rivelato molto utile per i nostri scopi poiché ci permette di effetturare una stima della potenza a livello astratto senza entrare in complicate considerazioni di tipo circuitale o tecnologico.

Poiché si è impostato l'intero progetto su una descrizione generica del problema indipendente dai valori specifici di M, N ed NQ si è voluto estendere questa filosofia anche al design della logica di selezione. Ciò ha comportato un ulteriore aumento della complessità del progetto e per questo è stato necessario modificare lo *spreadsheet* fornitoci ed adattarlo alle nostre esigenze. Attraverso quest'ultimo infatti si può effettuare una ottimizzazione *system-level* agendo su due parametri principali: il numero di banchi e la dimensione di una word.

5.1.1 Il modello matematico della memoria

Analizzando lo *spreadsheet* si può notare come il consumo di una generica SRAM possa essere espresso attraverso la seguente equazione:

$$P = V_{dd}^2 f_m K_m \tag{5.1}$$

dove V_{dd} è la tensione di alimentazione, f_m è la frequenza della SRAM e K_m è un parametro che dipende da molteplici fattori (switching activity di dati e indirizzi, dimensione della memoria, grandezza della word ecc.). Più precisamente questo fattore può essere scritto in questo modo:

$$K_m = CAVG \cdot AM + CCK \cdot ACK \tag{5.2}$$

dove $CAVG$ è la capacità equivalente dell'intera memoria e CCK è la capacità di ingresso equivalente sul clock. Questi due termini sono moltiplicati per due costanti: AM e ACK che rappresentano rispettivamente le *switching activity* dei dati e del clock. Da questa analisi risulta subito evidente una cosa: per diminuire il consumo di potenza del nostro blocco di memoria dovremmo agire su questi due parametri. E' quindi ovvio che nei momenti di non attività di ciascun banco di memoria dovremmo "congelare" il clock, i dati e gli indirizzi per limitare l'acitivity di quest'ultimi e quindi abbassare la potenza dissipata. Più precisamente le due capacità equivalenti $CAVG$ e CCK sono funzione dei parametri di sistema del banco di memoria (numero di *word* W, numero di bit per *word* N, numero di bit di *address* $M = log_2(W)$) in questo modo:

$$CAVG = 10 + 2.5 \cdot N + 0.0015 \cdot W + 0.00025 \cdot N \cdot M \tag{5.3}$$

$$CCK = 0.15 + 0.05 \cdot N + 0.025 \cdot M \tag{5.4}$$

Le equazioni (5.3) e (5.4) sono le equazioni che caratterizzano dal punto di vista della capacità una memoria. La potenza dissipata sarà dunque esprimibile come:

$$P_{average} = (CAVG \cdot AM + CCK \cdot ACK) \cdot V_{dd}^2 \cdot f_m \tag{5.5}$$

Per l'ottimizzazione di potenza della memoria dobbiamo quindi considerare molte variabili. Con un'attenta analisi, però, si può notare come la frequenza della memoria sia imposta dalla velocità del protocollo *N-M-PPM*. Sul bus infatti avviene una trasmissione alla frequenza di:

$$f_{bus} = f_{RAM} = \frac{f_{carrier}}{N \cdot M} \tag{5.6}$$

considerando poi che le specifiche del progetto richiedono un'unica tensione di alimentazione per tutto il blocco di *baseband processing* le variabili su cui giocare per abbassare il consumo di potenza della memoria rimangono poche. Non si può agire infatti nemmeno sul *critical path* per diminuire la tensione di alimentazione poichè quest'ultima è imposta dallo scaling della V_{dd} del digital backend. Infine le variabili su cui agire sono quindi solo due: il numero di banchi(parametro *NBANKS*) e il numero di simboli per word(parametro $NSYMBW$).

5.1.2 Lo spreadsheet

Lo *spreadsheet* fornitoci, come abbiamo più volte precisato, non è in grado di fornire una stima accurata del consumo di potenza. Per questo lo abbiamo moficato aggiungendo quattro parametri fondamentali:

- $f_{carrier}$ la frequenza della portante (nel nostro caso 1GHz e 100 MHz).
- M il numero di simboli.
- N il *repetition code number*.
- ed infine *DIMMEM* la dimensione della memoria. Infatti tipicamente questo è un parametro che rientra tra le specifiche e non ha quindi senso effettuare analisi comparative agendo su di esso.

A partire da questi parametri abbiamo stimato il consumo di potenza attraverso dei grafici come quello in figura 5.1(che sono generabili per qualsiasi parametro in ingresso).

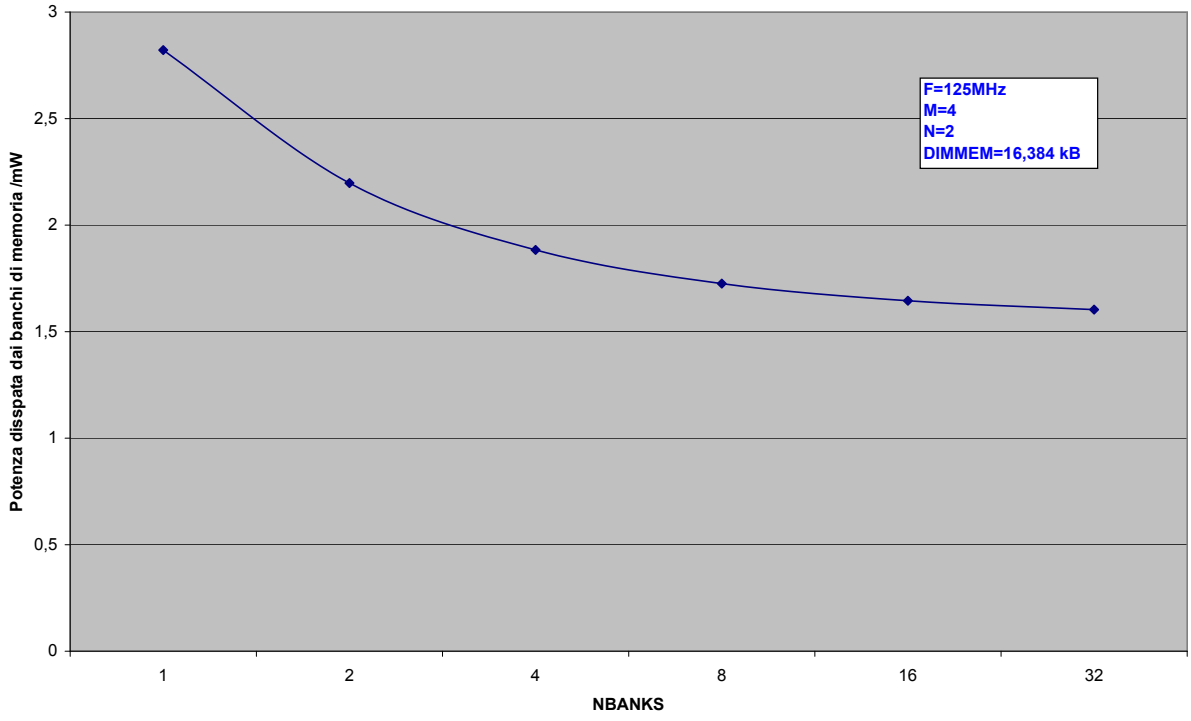

Figura 5.1: Grafico del consumo di potenza della memoria al variare del numero di banchi

Come si nota facilmente dalla figura aumentare il numero di banchi porta a risparmi che possono arrivare fino al 40 %. Ciò è dovuto al fatto che in presenza di banchi di memoria più piccoli si caricano *bitline* con capacità minore.

Dal grafico si evince anche come un partizionamento superiore ad 8 non porta ad abbassamenti di consumo considerevoli. Le memorie infatti (come anche indicato dalle equazioni (5.4) e (5.3)) hanno un consumo di potenza statico inpipendete dai parametri di sistema a disposizione del progettista.

Partizionare la memoria in più di 8 banchi quindi, non solo non porta benefici, ma aumenta anche l'overhead di consumo della circuiteria di selezione. Molte volte quest'ultima (come nel nostro caso) ha dei consumi di potenza molto grandi derivanti dal fatto di essere realizzata con della logica non *ad hoc*.

Il grafico che segue evidenzia invece come la tecnica di accorpare più simboli per word porti ad un ancora maggiore risparmio di potenza e soprattutto come non abbia alcun senso coniugarla alla strategia di dividere la memoria in più banchi. Infatti il grafico di figura 5.2 è tracciato per gli stessi valori di quello di figura 5.1 ($M = 4, N = 2, f_{SRAM} = 1GHz/(M \cdot N) = 125MHz$). In particolare, la curva più in alto corrisponde esattamente a quella di figura 5.1, le rimanenti curve sono tracciate ciascuna per un numero diverso di banchi e rappresentano la potenza al variare del numero di simboli per word. Si nota molto facilmente come queste curve convergano rapidamente e come quindi non abbia alcun senso combinare le due tecniche. Ciò è dovuto essenzialmente al fatto che la capacità delle bitline è poco influenzata dal numero di bit di dato, ma, al contrario, è notevolmente dipendente dalla profondità della memoria.

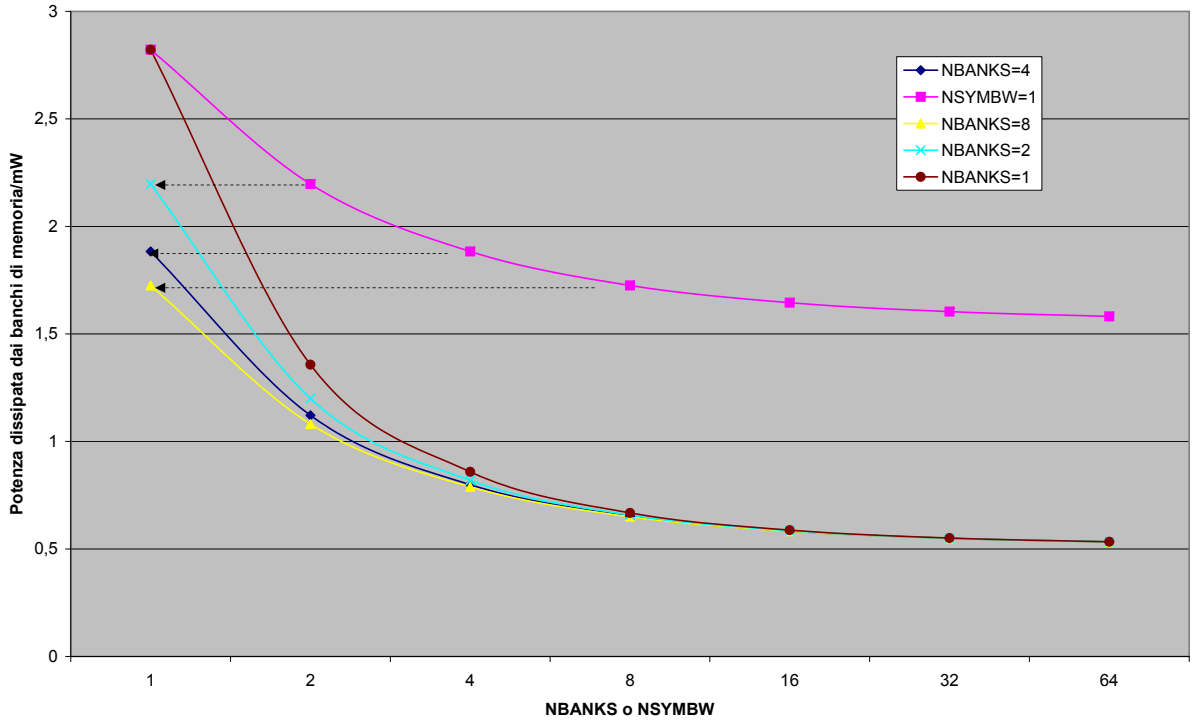

Figura 5.2: Grafico del consumo di potenza della memoria al variare del numero di banchi e del numero di simboli per word

È importante considerare infine che tutte queste considerazioni sono state effettuate tenendo conto di un modello puramente teorico che ha i suoi limiti e che non considera in nessun modo il consumo di potenza della circuiteria di selezione.

5.2 Il circuito di controllo della memoria

Alla luce delle precedenti considerazioni si è progettato un circuito generico che, dati alcuni parametri di ingresso settabili dal progettista (nel file *constants_def.vhd*), genera una opportuna logica di selezione della memoria.

Questo circuito dovrà impacchettare i simboli in macro-word più grandi (parametro *NSYMBW*) e generare i rispettivi segnali di indirizzo e di controllo (*clock gating*) per i banchi (parametro *NBANKS*).

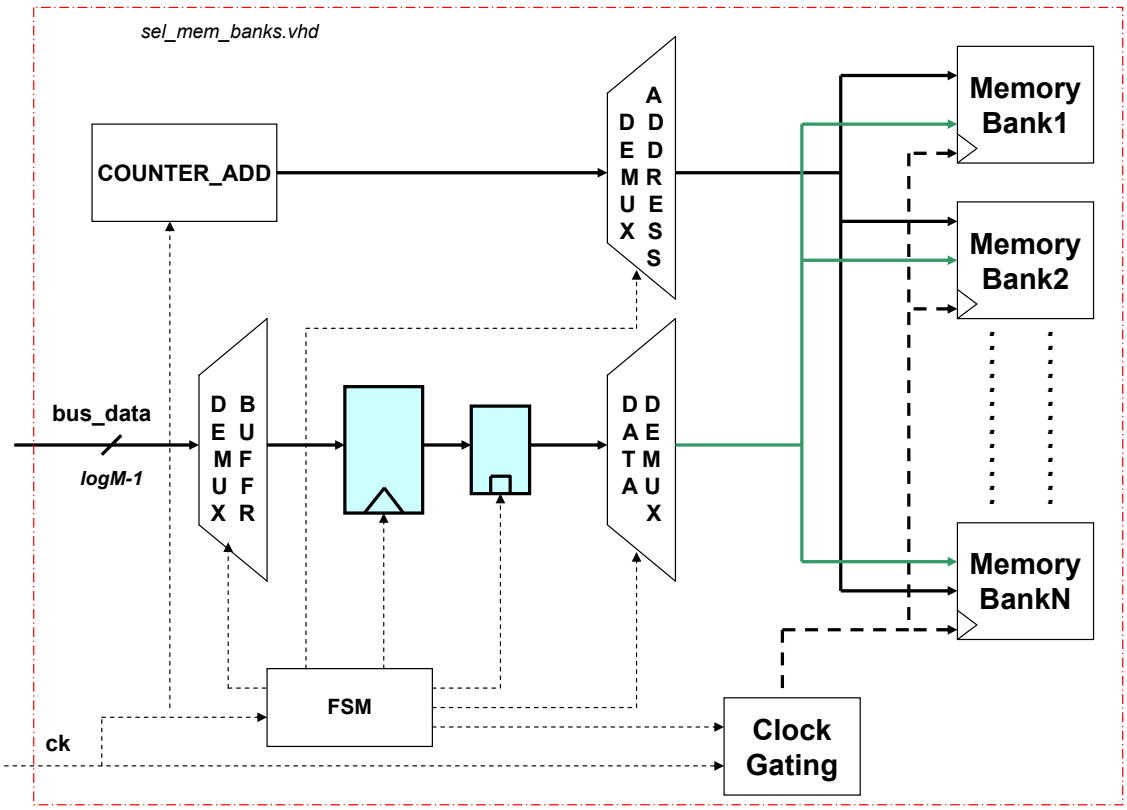

Figura 5.3: Schema di principio del circuito di selezione della memoria

Lo schema è basato principalmente su una doppia operazione di multiplexing il cui schema di principio è riportato nella figura 5.3. I dati arrivano nel primo multiplexer che li memorizza in un buffer di dimensione *NSYMBW*. A buffer pieno l'FSM genererà un impulso che rende trasparente il latch così da mandare i dati (raggruppati in una word multisimbolo) all'ingresso di uno dei banchi di memoria. I dati e gli indirizzi in ingresso

a ciascun banco resteranno quindi stabili fino alla scrittura successiva che viene eseguita abilitando il banco con un impulso di clock gating (nell'immagine *enable_banks*).

Tutti i segnali di abilitazione dei multiplexer così come i segnali di enable dei banchi sono implementati in codice *Gray* al fine di minimizzare la potenza sulle linee di controllo. La scrittura quindi supponendo $NBANKS = 4$ non avverrà in modo classico. Tutto ciò potrebbe sembrare un problema visto che la memoria viene usata di solito come buffer dati da un sistema a microprocessore. Se la lettura può avvenire esclusivamente in modo sequenziale si possono invertire (semplicemente in modo wired) le uscite dei multiplexer e collegarle al banco giusto. Ad esempio nel caso precedente si può collegare l'uscita quattro del multiplexer al terzo banco e viceversa eliminando definitivamente il problema senza rinunciare al risparmio di potnza sulla logica di controllo.

Contemporaneamente da un opportuno contatore con convertitore *Gray* saranno generati gli indirizzi da inviare al rispettivo banco di memoria in cui scrivere. In questo modo si può abbassare drasticamente la switching activity dei dati in ingresso, degli indirizzi e del clock aumentando il risparmio di potenza.

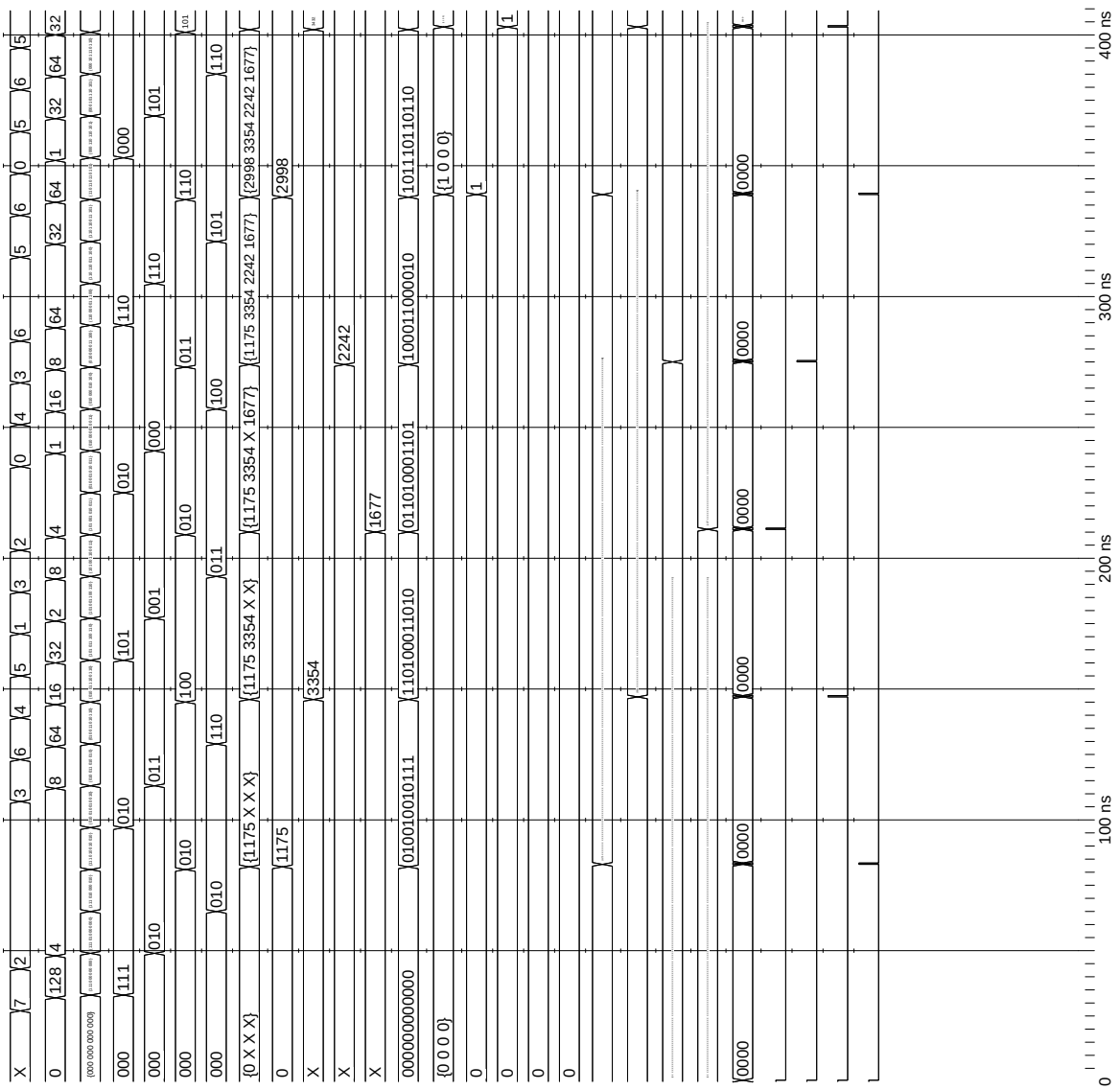

Figura 5.4: *Waveform* del sistema di controllo dei banchi di memoria

CAPITOLO 6

Simulazioni e conclusioni

6.1 Risultati delle simulazioni

Il consumo e le prestazioni del sistema sono stati valutati al variare di diversi parametri generici. Ci si è concentrati tuttavia soprattutto nel valutare il circuito nelle seguenti condizioni:

- M = 8
- N = 8
- NQ = 4
- NBANKS = 4
- NSYMBW = 4

In particolare si è simultato il sistema a due diverse frequenze di portante (1GHz e 100MHz) e per ciascuna frequenza si è valutato il consumo e il *timing* in presenza e in assenza di una capacità di bus di 10 pF.

I risultati dei report forniti da *synopsys* con *back annotation* della simulazione di *modelsim* sono qui di seguito esposti.

	$f_{carrier} = 100MHz$	$f_{carrier} = 1GHz$
$C_{bus} = 0$	$P = 310,1064\mu W$	$P = 2,4799mW$
$C_{bus} = 10pF$	$P = 318,8463\mu W$	P=2,5467 mW

Tabella 6.1: Risultati della sintesi

I risultati in tabella 6.1 fanno riferimento alla potenza dinamica e sono tutti relativi ad una tecnologia a 90nm con tensione di alimentazione di 1V. Essi comprendono la potenza del digital backend, bus encoder, bus decoder e logica di selezione della memoria.

Nel report che mostriamo di seguito invece si nota in che modo il consumo di potenza sia distribuito all'interno del circuito a 100 MHz (in assenza di bus). Si noti in particolare come l'unità Rx_and_store consumi ben il 28 % dell'intera potenza e questo fondamentalmente perché la struttura di selezione dei banchi non è ottimizzata al meglio (come quella che si potrebbe ottenere invece più facilmente utilizzando tool di *memory generation*) sia perché si spreca potenza nel gestire più banchi quando si adotta già la tecnica di accorpare più simboli in una word. Da qui deriva un ulteriore margine di miglioramento che è possibile sfruttare in futuro per diminuire ulteriormente il consumo. Infatti semplicemente diminuendo il numero di banchi e portandolo a 2 con $f_{carrier} = 1GHz$ si passa da un consumo di 2,4799 mW a un consumo di 2,2938 mW.

```
Power-specific unit information :
    Voltage Units = 1V
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW

--------------------------------------------------------------------------------
                                     Switch   Int      Leak     Total
Hierarchy                            Power    Power    Power    Power    %
--------------------------------------------------------------------------------
UWB_system_M8_N8_NQ4_logM3_logN3     3.36e-02    0.277 1.35e+07    0.324 100.0
  Rx_and_store_inst (Rx_and_store)   1.46e-03 8.56e-02 3.88e+06 9.10e-02  28.1
    rx_bus_inst (Bus_rx)             4.01e-05 2.03e-02 7.03e+05 2.11e-02   6.5
    sel_mem_banks_inst
                                     9.77e-04 4.06e-02 2.14e+06 4.37e-02  13.5
    mem_FSM_inst
                                     4.46e-04 2.47e-02 1.03e+06 2.62e-02   8.1
  bus_enc_inst (bus_encoder_M8_logM3) 3.55e-05 4.80e-03 3.11e+05 5.15e-03   1.6
  ddb_inst (double_digital_backend_N8_M8_NQ4_logM3_logN3_ABITS2)
                                     9.93e-03    0.149 8.42e+06    0.167  51.7
    demod_inst (Demod_logic_M4_N8_NQ4_logM2)
                                     2.76e-05 6.15e-05 6.05e+05 6.94e-04   0.2
    neg_backend (neg_digital_backend_N8_M4_NQ4_logM2)
                                     5.76e-03 8.39e-02 3.36e+06 9.30e-02  28.7
      max_det_inst (neg_max_detector_M4_N8_NQ4_logM2)
                                     2.52e-04 2.68e-02 1.23e+06 2.83e-02   8.8
      PU_inst (Neg_PU_M4_N8_NQ4_logM2) 5.50e-03 5.69e-02 1.92e+06 6.43e-02  19.9
    pos_backend (digital_backend_N8_M4_NQ4_logM2)
                                     2.82e-03 3.77e-02 2.07e+06 4.25e-02  13.1
      max_det_inst (max_detector_M4_N8_NQ4_logM2)
                                     1.59e-04 1.60e-02 8.35e+05 1.70e-02   5.3
      PU_inst (Pos_PU_M4_N8_NQ4_logM2) 2.66e-03 2.15e-02 9.98e+05 2.51e-02   7.8
    FSM_inst (ctrl_FSM_M4_N8_NQ4_logM2_logN3)
                                     1.17e-03 1.33e-02 8.72e+05 1.53e-02   4.7
      gray (gray_counter_M4_logM2)   4.99e-04 2.48e-03 8.00e+04 3.06e-03   0.9
      mod_N (mod_N_counter_M4_N8_logM2_logN3)
                                     1.97e-04 2.29e-03 1.12e+05 2.60e-03   0.8
```

6.2 Risultati dello spreadsheet

I risultati finora esposti non evidenziano i vantaggi relativi all'aver utilizzato una struttura parallela, infatti la scelta di parallelizzare non comporta nessun effetto sulla potenza dinamica se non quello di dare luogo a degli slack piuttosto grandi da sfruttare abbassando la tensione di alimentazione.

La simulazione di quello che è il consumo in funzione della tensione di alimentazione è effettuata applicando un modello matematico implementato nello *spreadsheet*.
Inoltre quest'ultimo deve essere necessariamente utilizzato per valutare il consumo dei banchi di memoria di cui abbiamo considerato finora solo un modello *behavioural* mai sintetizzato.

Le tabelle che seguono mostrano il consumo di potenza del sistema di ricezione, trasmissione sul bus e della memoria per due valori differenti di $T_S = 1/f_{carrier}$ (10ns e 1ns).
Le tre tabelle sono state ottenute per valori di tensione diversi. Nel primo caso è stata stimata la potenza considerando la tensione nominale prevista per la nostra libreria tecnologica (1V).

T_s	P_{UWB}	$P_{MEMORIA}$	P_{TOT}
10 ns	$310,1064\mu W$	$16,1555\mu W$	$326,262\mu W$
1 ns	2,4799 mW	$161,55\mu W$	2,6414 mW

Tabella 6.2: Risultati dello spreadsheet senza scaling

La tabella 6.3 mostra il consumo di potenza con lo scaling massimo ammesso dalla nostra libreria (0,9V). Notiamo come rispetto alla precedente ci siano risparmi di potenza intorno al 20%.

T_s	P_{UWB}	$P_{MEMORIA}$	P_{TOT}	V_{DD} scaled
10 ns	$251,186\mu W$	$13,086\mu W$	$264,272\mu W$	0,9 V
1 ns	2,0087 mW	$130,860\mu W$	2,13958 mW	0,9 V

Tabella 6.3: Risultati dello spreadsheet con scaling di libreria

Infine si è valutato il consumo di potenza che si riscontrerebbe applicando la tensione di alimentazione minima che rispetti i vincoli di timing imposti dalla memoria e dal *digital backend*. Nel nostro caso i vincoli di timing più stringenti sono relativi alla circuiteria di selezione della memoria. Ciò avviene proprio perché, come già più volte precisato, questa parte del circuito non è sufficientemente ottimizzata.

Per questo motivo non è stato possibile sfruttare a pieno lo *slack* del *digital backend* per diminuire il consumo. Questo è particolarmente evidente nel secondo caso ($T_s = 1ns$) dove gli *slack* sono notevolmente minori.

Per quanto riguarda le memorie, si noti come esse hanno un consumo trascurabile rispetto al resto del circuito.

T_s	P_{UWB}	$P_{MEMORIA}$	P_{TOT}	V_{DD} scaled
10 ns	$77,527\mu W$	$4,0389\mu W$	$81,5659\mu W$	0,5 V
1 ns	1,0477 mW	$68,26\mu W$	1,1160 mW	0,65 V

Tabella 6.4: Risultati dello spreadsheet con scaling massimo

Questo è dovuto a due motivi principali: il primo è rappresentato dalla ridotta frequenza di funzionamento (1,5MHz nel primo caso e 15,6MHz nel secondo). Il secondo, invece, è riconducibile all'ottimizzazione del numero di banchi e del numero di simboli per word.