University of Pisa

MSc in Computer Engineering

Electronic System

2018/2019

Ricevitore CDMA

Report e implementazione VHDL

Matteo SUFFREDINI

Contents

1 Introduzione

Nell'ambito delle telecomunicazioni per permettere a più utenti di condividere contemporaneamente il medesimo canale trasmissivo esistono principalmente tre tecniche: divisione di tempo, divisione di frequenza e divisione di codice.

L'accesso multiplo a divisione di codice o CDMA (Code-Division Multiple Access) necessita, dal lato del trasmettitore, di un "Trasmettitore CDMA" in grado di codificare lo *Stream* di bit, decodificati successivamente dal lato del ricevitore mediante un "RicevitoreCDMA".

In pratica quello che viene effettuato è ridistribuire la potenza del segnale, occupando un maggior quantitativo di banda, questo comporta un aumento del periodo di trasmissione.

1.1 Funzionamento Ricevitore CDMA

Lo scenario è rappresentato nella sua interezza da un trasmettitore CDMA ed un ricevitore CDMA che vogliono scambiarsi dati attraverso un canale condiviso da diversi utenti. Il fatto di avere una codifica rende impossibile ad un utente non in possesso della parola in codice la decodifica del messaggio garantendone quindi la sicurezza. Inoltre con una opportuna scelta della parola in codice, è possibile eliminare il MAI (Multiple Access Interference).

Per codificare lo *Stream* viene utilizzata una parola in codice $CodeWord$ generata pseudo-randomicamente, la stessa parola è presente sia dal lato del trasmettitore che dal lato del ricevitore rendendo possibile codifica e decodifica.

La $CodeWord$ utilizzata per trasmettere un singolo bit di informazione dello *Stream* è composta da un certo numero di bit detti $CodeWordChip$, andando a moltiplicare ogni $CodeWordChip$ per il bit dello *Stream* da trasmettere, otteniamo un certo numero di $ChipStream$, che rappresentano la codifica del bit di *Stream*.

Ne segue che, per codificare e trasmettere un bit correttamente c'è bisogno di utilizzare un periodo minore rispetto a quello con cui viene inviato un bit di *Stream* al trasmettitore e

letto dal ricevitore.

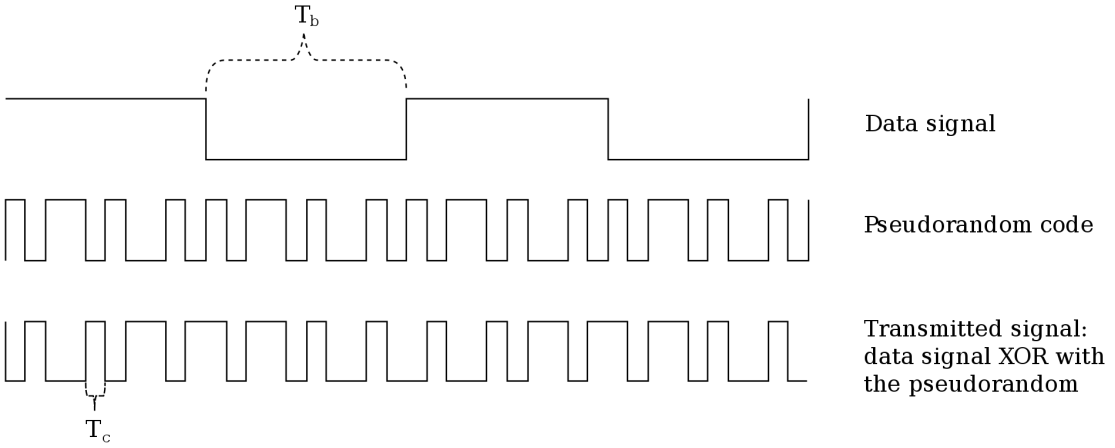

Figure 1: Generazione CDMA

Il numero di $CodeWordChip$ dipende dallo spreading factor SF, ovvero ci sono tanti $CodeWordChip$ pari allo SF. Ne segue che se la frequenza di trasmissione di un dato è pari a T_b, la trasmissione tra trasmettitore e ricevitore dovrà avvenire con periodo:

$$T_\mathrm{c} = \frac{T_\mathrm{b}}{SF}$$

1.2 Funzionamento decisore hard a soglia

Esistono due possibili implementazioni per il decisore: hard o soft.

Il **decisore hard** decide l'output in base al superamento di una predeterminata soglia, non è un'implementazione molto fine ma di sicuro permette di ridurre la complessità della componente.

Il **decisore soft** effettua una stima più fine ricevendo in ingresso una gamma di valori intesi come valori probabilistici di prossimità ai valori possibili garantendo migliori prestazioni in presenza di dati corrotti, tuttavia a discapito della complessità della componente.

Nel progetto viene fatto utilizzo di un decisore hard a soglia che prende in ingresso uno stream composto da un certo numero di bit. Sommando tutti i bit ricevuti, alla fine, deve decidere sulla base del valore raggiunto se mettere in uscita '0' oppure '1' a seconda del superamento di una determinata soglia.
Dato uno stream di N bit:

- **decido '1':** se $\sum_{k=1}^{N} bit[k] \geqslant \frac{N}{2}$
- **decido '0':** se $\sum_{k=1}^{N} bit[k] < \frac{N}{2}$

2 Architettura

Il Ricevitore CDMA contiene un Decisore hard a soglia, il quale è composto da alcune componenti di base come Contatori, D-Flip-Flop (con reset sincrono) e Ripple Carry Adder, l'architettura di questi ultimi non viene riportata essendo la medesima vista a lezione.
Di seguito verranno utilizzati variabili per rappresentare alcune entità precedentemente descritte:

- $CodeWordChip = X\_chip\_CDMAR$
- $ChipStream = Xs\_chip\_cdmar$
- un bit dello $Stream = S\_CDMAR$

2.1 Ricevitore CDMA

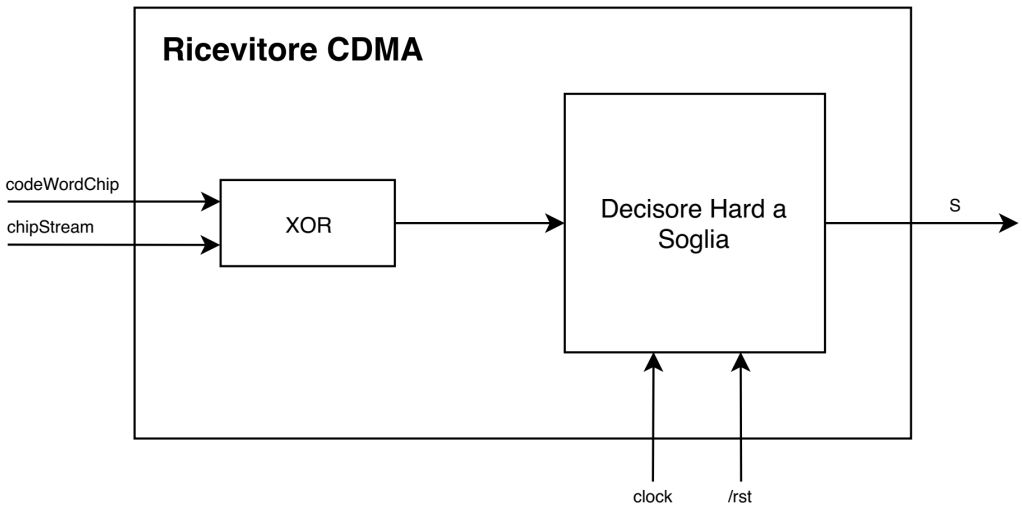

Figure 2: Architettura ricevitore CDMA

```
entity CDMAReceiver is generic (N: integer);
     port(
          Xs_chip_CDMAR : in std_logic;
          C_chip_CDMAR : in std_logic;
          reset_CDMAR : in std_logic;
```

```
          clock_CDMAR : in std_logic;
          S_CDMAR : out std_logic
     );
end CDMAReceiver;
```

Il despreading viene effettuato mediante una porta XOR:

```
S_chip_CDMAR <= Xs_chip_CDMAR xor C_chip_CDMAR;
```

2.2 Decisore hard a soglia

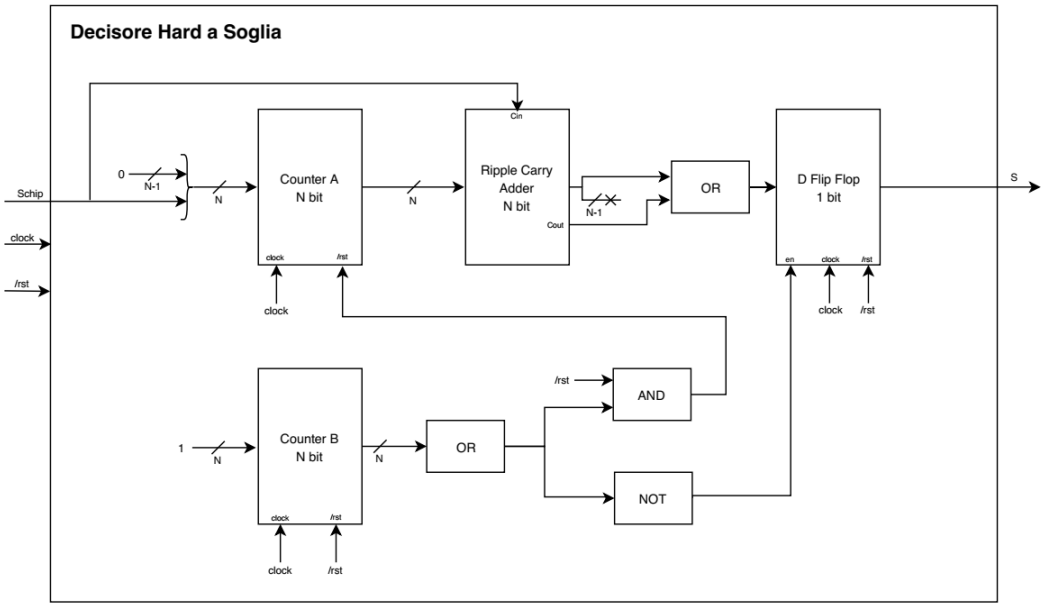

Figure 3: Architettura decisore hard a soglia

```
entity  decisoreHardASoglia is generic (N: integer);
     port(
          S_chip_DHS : in std_logic;
          clock_DHS : in std_logic;
          reset_DHS : in std_logic;
              S_DHS : out std_logic
     );
```

```
end decisoreHardASoglia;
```

Vengono ricevuti N chip durante N cicli di clock, i primi N-1 chip vengono sommati all'interno di un contatore, il chip N viene sommato all'output del contatore e salvato all'interno di un D-Flip-Flop ogni N cicli di clock. L'ultimo chip viene aggiunto mediante un sommatore per permettere al contatore di resettarsi e non perdere il primo chip della sequenza successiva.

3 Test

E' necessario testare principalmente se tutti i 16 chip del primo dato (al reset) e del successivo vengono letti ed interpretati correttamente, pertanto vengono effettuati due test:

- verifica se il primo chip di ogni sequenza viene letto correttamente
- verifica se l'ultimo chip di ogni sequenza viene letto correttamente

3.1 Lettura primo chip di un dato

Il test vuole valutare se il primo chip della prima e della seconda sequenza vengono rilevati correttamente.
Il test viene eseguito rendendo significativo il primo e il 17-esimo chip per ottenere un '1' in output al primo e al secondo dato dello *Stream*, nello specifico:

chip number	Xs chip	C chip	S chip
1-8	1	0	1
9-16	1	1	0
		SUM	8

chip number	Xs chip	C chip	S chip
17-24	1	0	1
25-32	1	1	0
		SUM	8

Ricevendo 8 chip a '1', il decisore decide per '1', se il primo chip non fosse ricevuto correttamente avremmo solo 7 chip a '1' e quindi verrebbe deciso per '0'.
Per valutare l'uscita del secondo dato, nell'esempio vengono inviati un totale di 50 chip, quelli relativi al terzo bit dello stream sono calcolati in modo da avere in uscita '0'.
Ci si aspetta quindi che, dopo i primi 16 chip, l'uscita transisca ad '1' e vi rimanga per i successivi 32 cicli di clock, dopodichè transisca nuovamente a '0':

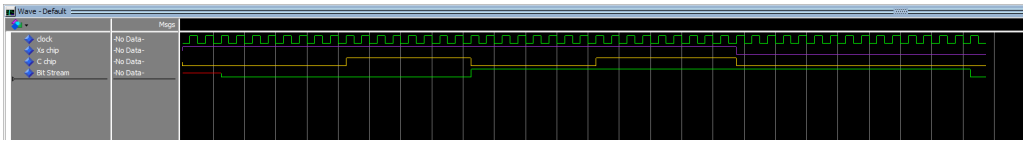

Figure 4: Test primo chip

3.2 Lettura ultimo chip di un dato

Il test vuole valutare se l'ultimo chip della prima e della seconda sequenza vengono rilevati correttamente.

Il test viene eseguito rendendo significativi il 16-esimo e il 32-esimo chip per ottenere un '1' in output al primo e al secondo dato dello *Stream*, nello specifico:

chip number	Xs chip	C chip	S chip
1-8	1	1	0
9-16	1	0	1
		SUM	8

chip number	Xs chip	C chip	S chip
17-24	1	1	0
25-32	1	0	1
		SUM	8

Ricevendo 8 chip a '1', il decisore decide per '1', se l'ultimo chip non fosse ricevuto correttamente avremmo solo 7 chip a '1' e quindi verrebbe deciso per '0'.

Per valutare l'uscita del secondo dato, nell'esempio vengono inviati un totale di 50 chip, quelli relativi al terzo bit dello stream sono calcolati in modo da avere in uscita '0'.

Ci si aspetta quindi che, dopo i primi 16 chip, l'uscita transisca ad '1' e vi rimanga per i successivi 32 cicli di clock, dopodichè transisca nuovamente a '0':

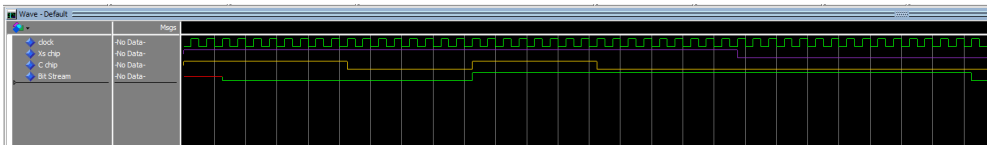

Figure 5: Test secondo chip

4 Sintesi e implementazione

4.1 Step 1: RTL Design

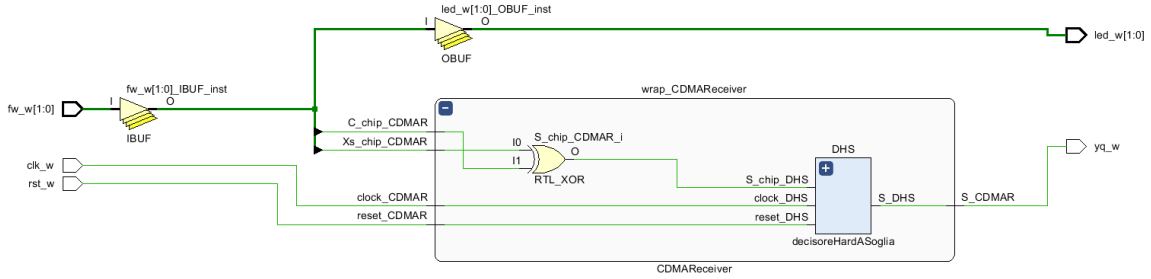

Figure 6: RTL Design Ricevitore CDMA da Vivado

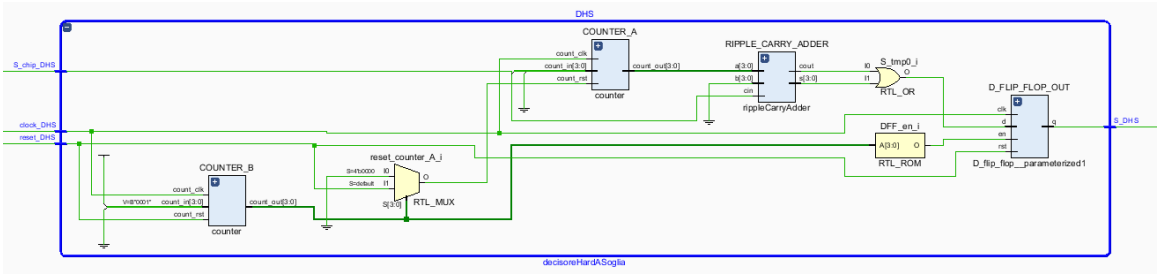

Figure 7: RTL Design Decisore Hard a Soglia da Vivado

4.2 Step 2: Sintesi

E' stato imposto un vincolo sul **clock di** $t_{clk} = 1Mhz$ ed è stata avviata la sintesi che è terminata con un warning.

Il warning in questione segnala l'assenza dei ritardi per i fili di ingresso e di uscita (escluso il clock), necessari per garantire una corretta stima dei tempi di ritardo. Non è richiesta la gestione di questo aspetto nel progetto pertanto il warning viene ignorato.

4.2.1 Timing report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6,974 ns	Worst Hold Slack (WHS):	0,142 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	14	Total Number of Endpoints:	14	Total Number of Endpoints:	10

All user specified timing constraints are met.

Figure 8: Timing report da Vivado

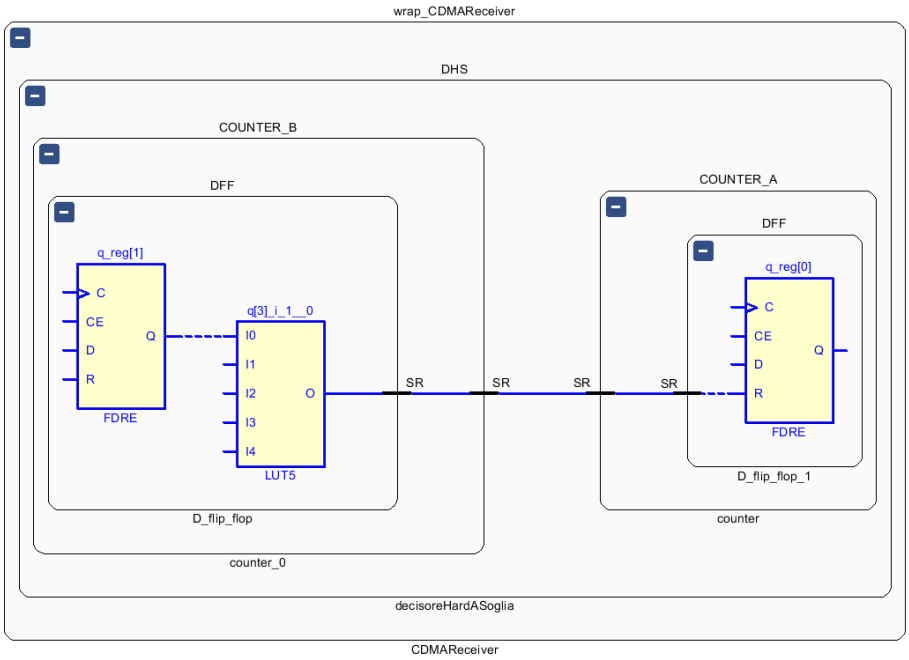

Figure 9: Worst path da Vivado

Partendo dallo slack è possibile valutare quanto veloce possiamo pilotare il clock:

$$t_{clk} = t_{setup} + t_{p-logic} + t_{c-q} + slack$$

la stessa espressione può essere scritta per il caso in cui abbiamo il minimo clock, considerando il caso limite con $slack = 0$:

$$t_{clkmin} = t_{setup} + t_{p-logic} + t_{c-q} + 0$$

Sottraendo tra loro le precedenti:

$$t_{clkmin} - t_{clk} = 0 - slack => t_{clkmin} = t_{clk} - slack$$

Dalla quale si ricava:

$$f_{max} = \frac{1}{t_{clkmin}} = \frac{1}{t_{clk} - slack} = \frac{1}{1000ns - 6,974ns} = 10MHz$$

4.2.2 Utilisation report

Dai dati ottenuti da Vivado si evince che non ci sono importanti utilizzi degli I/O pads della Zync700 Soc. Questo risultato era atteso in quanto il componente implementato non fa largo uso dell' I/O.

L'altro piccolo contributo BUFG è dovuto alla necessità di generare un clock.

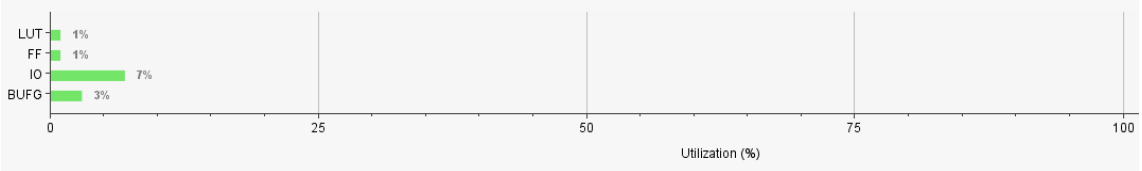

Figure 10: Utilisation report da Vivado

4.2.3 Power report

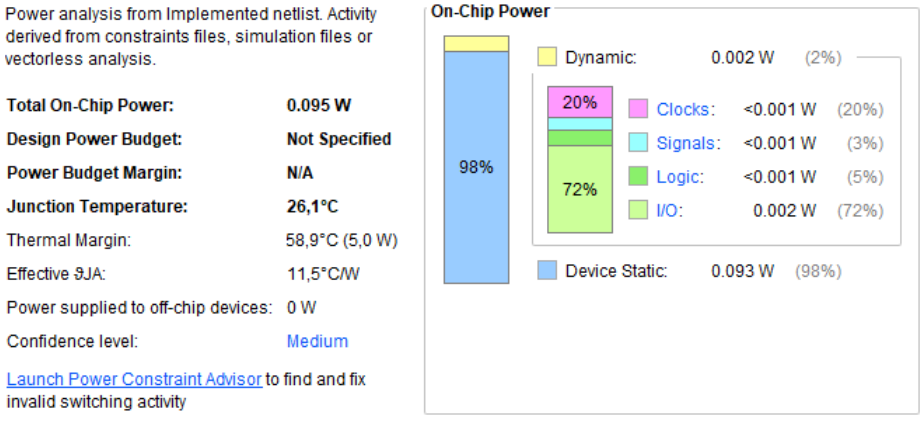

Figure 11: Power report da Vivado

Il report stima la potenza su chip a $0.095W$ di cui il 98% è relativo a dissipazioni statiche. Per quanto riguarda le dissipazioni dinamiche, la maggior parte sono dovute ad operazioni di I/O e alla generazione del clock, come ci si aspettava dal report sull'utilizzazione.

4.3 Step 3: Implementazione

La fase di implementazione viene eseguita senza produrre alcun warning, tuttavia viene segnalato il mancato utilizzo del "PS7 processor of the Zynq SoC", questa segnalazione può essere ignorata.
Il valore dello slack rimane praticamente immutato confermando:

$$f_{max} = 10MHz$$

5 Conclusioni

La componente realizzata trova largo utilizzo nell'ambiente delle telecomunicazioni e in particolare nei:

- sistemi GPS
- UMTS 3G
- sistemi satellitari per i trasporti logistici