University of Pisa

MSc in Computer Engineering

Electronic System

2018/2019

Ricevitore CDMA

Report e implementazione VHDL

Matteo SUFFREDINI

Contents

1 Introduzione

1.1 Ambienti di utilizzo del Ricevitore CDMA

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 o divisione di codice.

L'accesso multiplo a divisione di codice o CDMA (Code-Division Multiple Access) viene utilizzato in diversi ambiti:

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

1.2 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 decodificazione 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.

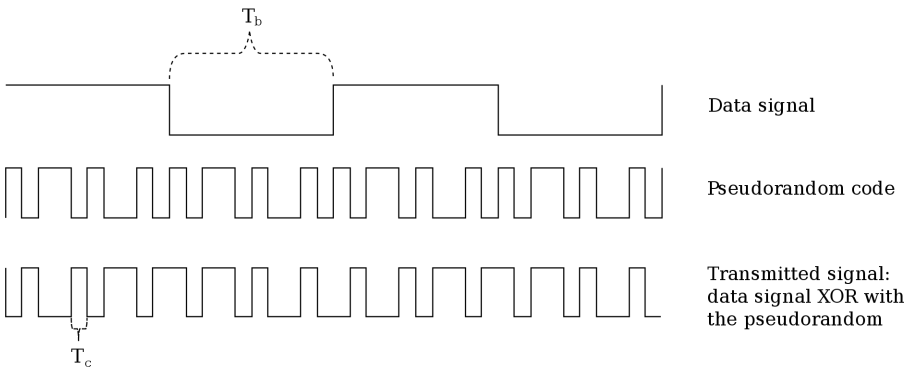

Figure 1: Generazione CDMA

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:

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

1.3 Funzionamento decisore hard a soglia

Il decisore hard a soglia prende in ingresso uno stream composto da un certo numero di bit, alla fine deve decidere sulla base dello stream ricevuto se mettere

in uscita 0 oppure 1 valutando se i bit ricevuti superano 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 e Ripple Carry Adder, l'architettura di questi ultimi non viene riportata essendo la medesima vista a lezione.

2.1 Ricevitore CDMA

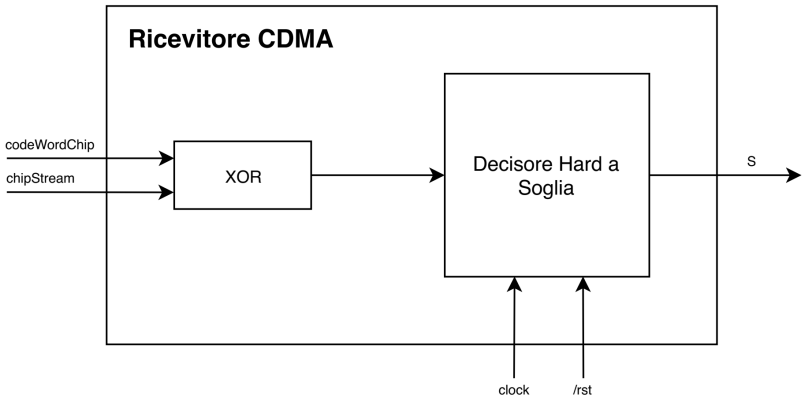

Figure 2: Architettura ricevitore CDMA

```
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

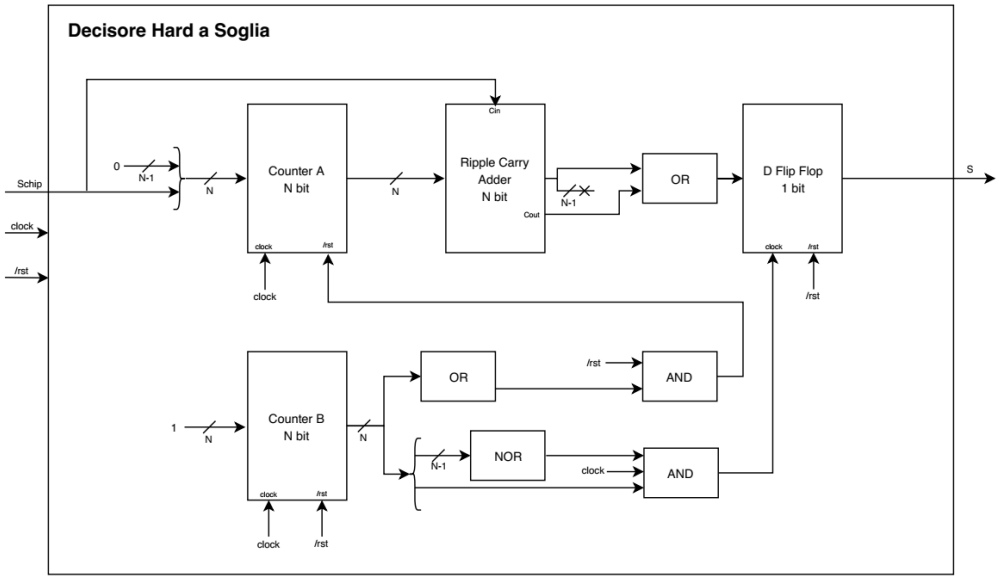

Figure 3: Architettura decisore hard a soglia

```
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.
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'.

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':

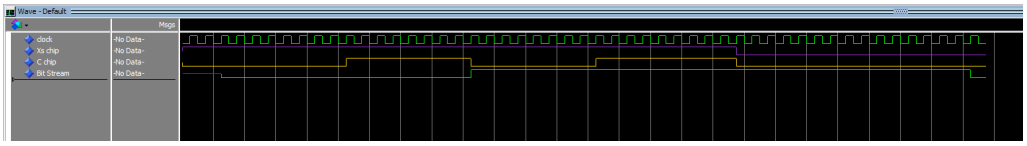

Figure 4: Test primo chip

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'.

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':

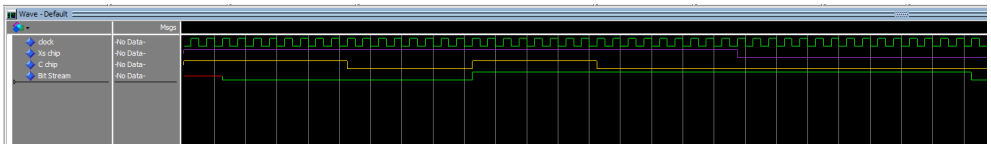

Figure 5: Test secondo chip

4 Sintesi e implementazione

5 Conclusioni