



UNIVERSITÀ DI PISA

Dipartimento di Ingegneria Dell'Informazione

CDMA RECEIVER

Progetto di Electronics and Communications Systems

Amedeo POCHIERO

A.Y. 2019-2020

Indice

1	Introduzione	1
1.1	Descrizione del Problema	1
1.2	CDMA	1
1.3	Applicazioni	2
1.4	Esempio utilizzato	2
1.4.1	Ricostruzione del segnale	3
2	Architettura	3
2.1	Diagramma a Blocchi	3
2.2	Implementazione	4
2.3	Test-plan	5
2.3.1	Testbench 1	6
2.3.2	Testbench 2	7
3	Sintesi	8
3.1	Utilizzo	8
3.2	Massima frequenza di utilizzo	8
3.3	Cammino Critico	9
3.4	Consumo di Potenza	9
4	Conclusioni	10

1 Introduzione

1.1 Descrizione del Problema

Il collegamento broadcast può avere più nodi trasmettenti e riceventi connessi allo stesso canale broadcast condiviso. In uno scenario del genere si pone il problema di come coordinare l'accesso di più nodi trasmettenti e riceventi in un stesso canale, ossia il problema dell'accesso multiplo. Dato che tutti i nodi sono in grado di trasmettere frame, è possibile che due o più lo facciano nello stesso istante, per cui tutti i nodi riceveranno contemporaneamente più frame. Tra questi si genera una collisione a causa della quale nessuno dei nodi riceventi riuscirà a interpretare i frame. È necessario utilizzare dei protocolli al fine di regolare gli accessi al mezzo condiviso.

1.2 CDMA

Il protocollo **CDMA** (*code division multiple access*) è un protocollo a suddivisione del canale in cui i vari utenti possono trasmettere contemporaneamente, causando quindi collisioni e interferenze tra loro, tuttavia il ricevente è in grado comunque di ricostruire il segnale trasmesso. A tale scopo, ogni utente modula il proprio segnale di periodo T_b (*symbol period*) con un codice, unico per ogni utente, di periodo T_c (*chip period*) dove $T_c \ll T_b$ come si può vedere in Figura 1.

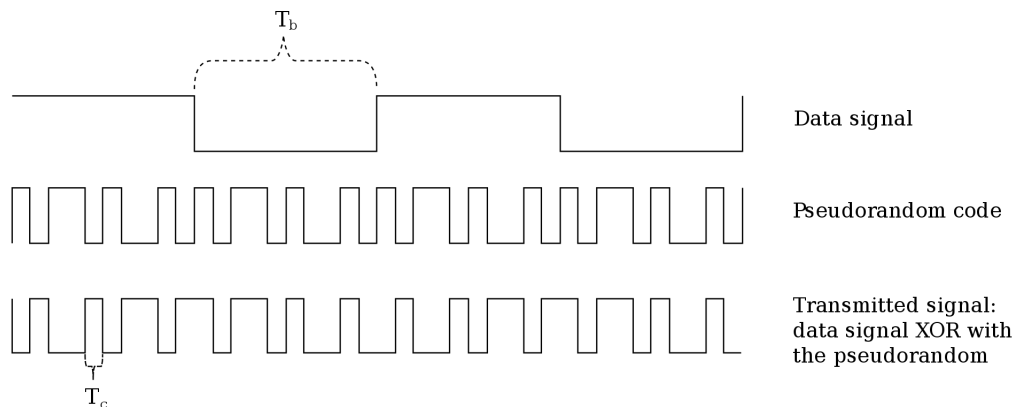


Figura 1: Generazione del segnale CDMA trasmesso

Il rapporto tra i due periodi è definito come *Spreading Factor* e come requisito è stato posto a 16:

$$\frac{T_b}{T_c} = 16$$

I codici devono essere scelti in modo tale che la correlazione tra i vari segnali dei diversi utenti sia il più vicino possibile a zero. Nel synchronous CDMA si può sfruttare la proprietà matematica di *ortogonalità* tra vettori che rappresentano stringhe di dati. Due vettori a e b si dicono ortogonali se vale la seguente relazione:

$$a \cdot b = 0$$

Ogni utente deve usare un codice ortogonale a quello di tutti gli altri. Nella tabella di seguito viene riportato un esempio di vettori $cw_1, cw_2 \in \mathbb{Z}^{16}$ ortogonali tra di loro, i bit 0, 1 vengono rappresentati rispettivamente dai simboli $-1, 1$:

vector	bit																Prodotto Scalare
cw_1	1	1	-1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1	-
cw_2	1	-1	1	-1	1	-1	-1	-1	1	1	-1	-1	-1	1	-1	-1	-
$cw_{1,i} * cw_{2,i}$	1	-1	1	-1	1	-1	-1	-1	1	1	-1	-1	-1	1	-1	-1	0

Tabella 1: Vettori Ortogonali

1.3 Applicazioni

Il **CDMA** è il protocollo di accesso a canale condiviso più diffuso nelle reti wireless e nelle tecnologie cellulari. Deriva da una tecnologia usata per implementare il **GPS** (*Global Position System*) ed è stato usato in :

- *Globastar network*, con il nome di CDMA2000, ed altre compagnie telefoniche
- *UMTS 3G* come protocollo di accesso multiplo standard
- *OmniTRACS satellite system*, per trasporti logistici.

1.4 Esempio utilizzato

Al fine di verificare il corretto funzionamento del ricevitore, è stato seguito un caso reale di ricezione di due bit in un ricevitore CDMA. Facendo riferimento ai codici (*code words*) della tabella 1, si è considerato uno scenario in cui il primo utente trasmette il simbolo 1, mentre un secondo utente trasmette il simbolo -1 . Per le proprietà fisiche dell'interferenza, se 2 segnali interferenti sono in fase, essi si sommano e si crea un segnale la cui ampiezza è la somma delle ampiezze, altrimenti si sottraggono e creano un segnale con un'ampiezza pari alla differenza delle ampiezze. Nel mondo digitale, questo comportamento può essere rappresentato dalla somma componente per componente dei vettori trasmessi.

Nella seguente tabella 2, il simbolo trasmesso è modulato con la rispettiva *code word*, ottenendo un *chip stream* per ogni utente. Ogni chip stream trasmesso interferisce con gli altri trasmessi nello stesso istante, formando l'*interference pattern*, cioè il segnale realmente ricevuto dai ricevitori.

data	value															
d_1	1															
d_2	-1															
$cw_{1,i} * d_1$	1	1	-1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
$cw_{2,i} * d_2$	-1	1	-1	1	-1	1	1	1	-1	-1	1	1	1	-1	1	1
<i>interference pattern</i>	0	2	-2	2	-2	0	0	2	0	-2	0	0	2	0	0	2

Tabella 2: Interferenza tra trasmettitori

1.4.1 Ricostruzione del segnale

Per ricostruire il segnale, un ricevitore moltiplica ogni componente dell'*interference pattern* con la propria *code word*. Il vettore r ottenuto è dato in input ad un *Decisore Hard A Soglia* il quale decide il bit da porre in uscita. La decisione è presa nel seguente modo:

- se $\sum_{i=1}^{16} r_i \geq 0$ allora $bitstream = 1$
- se $\sum_{i=1}^{16} r_i < 0$ allora $bitstream = 0$

Nell'esempio considerato i ricevitori effettueranno le seguenti operazioni:

data	value															Somma	Bit deciso	
$r_i * cw_{1,i}$	0	2	2	2	2	0	0	2	0	2	0	0	2	0	0	2	16	1
$r_i * cw_{2,i}$	0	-2	-2	-2	-2	0	0	-2	0	-2	0	0	-2	0	0	-2	-16	0

Come si nota dalla tabella, i ricevitori sono in grado di ricostruire il bit trasmesso originariamente (primo utente 1, secondo utente 0) come specificato in 1.4 .

2 Architettura

2.1 Diagramma a Blocchi

Nella Figura 2 è rappresentato il diagramma a blocchi per il ricevitore CDMA.

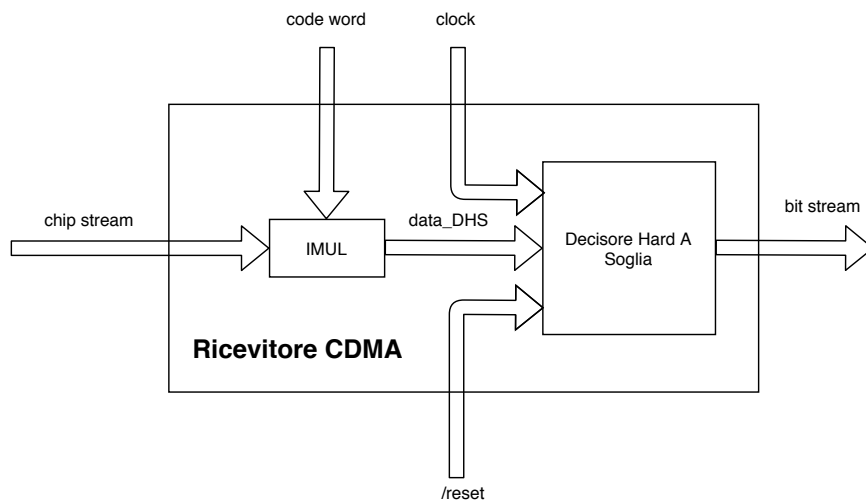


Figura 2: Diagramma a blocchi dell'architettura usata

In particolare vengono effettuate le operazioni descritte in 1.4.1 in modo tale da risalire al bit trasmesso. Per quanto riguarda il *clock* questo ha una frequenza pari al *chip rate*, cioè la frequenza $\frac{1}{T_c}$ con cui variano i bit della *code word*.

2.2 Implementazione

Per quanto riguarda l'implementazione, il Ricevitore CDMA è descritto tramite un misto tra l'architettura strutturale e dataflow. Infatti, questo si compone di un *Decisore Hard A Soglia* il cui ingresso segue sempre il prodotto tra la propria *code word* e il *chip stream* ricevuto o *interference pattern* secondo quanto detto in 2. Questi ultimi sono stati rappresentati tramite degli interi che di default sono su 32bit, ma per rendere più efficiente il sistema, è stato specificato il range di valori, in modo tale che il sintetizzatore faccia le dovute ottimizzazioni.

```
1  library IEEE;
2  use ieee.std_logic_1164.all;
3
4  entity CDMA_Receiver is
5      -- K active users
6      generic ( K : positive := 2;
7              SF : positive := 16 );
8      port(
9          clk_R : in std_ulogic;
10         reset_R : in std_ulogic;
11         code_word : in integer range -1 to 1 ;
12         chip_stream : in integer range -K to K;
13         bitstream_R : out std_ulogic
14     );
15 end CDMA_Receiver;
16
17 architecture data_flow of CDMA_Receiver is
18     -- Add component
19     component DecisoreHardASoglia is
20         generic ( K : positive := 2;
21                 SF : positive := 16 );
22         port(
23             clk_DHS : in std_ulogic;
24             reset_DHS : in std_ulogic;
25             data_DHS : in integer range -K to K;
26             bitstream_DHS : out std_ulogic
27         );
28     end component;
29     -- signal used as input for the DecisoreHardASoglia
30     signal data_s : integer range -K to K;
31
32     begin
33         DHS: DecisoreHardASoglia
34             -- Mapping
35             generic map ( K => K, SF => SF )
36             port map (
37                 clk_DHS => clk_R,
38                 reset_DHS => reset_R,
39                 data_DHS => data_s,
40                 bitstream_DHS => bitstream_R
41             );
42
43         -- Multiply the received signal ( chip_stream ) with own code word
44         data_s <= code_word * chip_stream;
45     end data_flow;
```

Figura 3: Descrizione VHDL CDMA Receiver

Il *Decisore Hard A Soglia* è stato descritto tramite l'architettura di tipo Behavioral in modo tale da specificare cosa accade quando arriva il fronte di salita del clock. Superata la fase iniziale di reset, il Decisore effettua la somma dei dati ricevuti ingresso ad ogni ciclo di clock, tenendo

nel segnale *waitData* quanti dati ha già contato. Un volta sommati SF dati si può procedere con la decisione seguendo quanto descritto in 1.4.1.

```

1  library IEEE;
2  use ieee.std_logic_1164.all;
3  use IEEE.NUMERIC_STD.ALL;
4  use ieee.std_logic_unsigned.all;
5  -- This component has to take a decision on the output bit
6  entity DecisoreHardASoglia is
7      generic ( K: positive := 2;
8                SF : positive := 16 );
9      port(
10         clk_DHS : in std_ulogic;
11         reset_DHS : in std_ulogic;
12         data_DHS : in integer range -K to K;
13         bitstream_DHS : out std_ulogic
14     );
15 end DecisoreHardASoglia;
16
17 architecture behavior of DecisoreHardASoglia is
18     -- Count how many data have been received
19     signal waitData : integer range 0 to SF := 0;
20     -- Sum data at every clock cycle
21     signal sum : integer range -SF to SF := 0;
22     begin
23         proc : process(clk_DHS)
24         begin
25             if ( rising_edge(clk_DHS)) then
26                 -- synchronous reset
27                 if (reset_DHS = '0' ) then
28                     bitstream_DHS <= 'U';
29                 else
30                     -- Update sum and waitData
31                     if ( waitData < 16 ) then
32                         sum <= sum + data_DHS;
33                         waitData <= waitData + 1;
34                     else
35                         -- otherwise make a decision on the bit
36                         if ( sum >= 0 ) then
37                             bitstream_DHS <= '1';
38                         else
39                             bitstream_DHS <= '0';
40                         end if;
41                         -- reset signals
42                         waitData <= 0;
43                         sum <= 0;
44                     end if;
45                 end if;
46             end if;
47         end process proc;
48     end behavior;

```

Figura 4: Descrizione VHDL Decisore Hard A Soglia

2.3 Test-plan

Per testare il ricevitore sono stati creati 2 testbench che seguono l'esempio descritto in 1.4. In particolare, è stato ricreato lo scenario in cui 2 utenti trasmettono nello stesso istante rispettivamente il bit 1 e il bit 0 mentre 2 ricevitori con le stesse code word dei trasmettitori stanno in ascolto. I test devono verificare che i ricevitori riescano a ricevere i bit giusti nonostante l'interferenza. Quindi:

- Ricevitore 1 deve decidere il bit 1
- Ricevitore 2 deve decidere il bit 0

2.3.1 Testbench 1

Entrambi i test sono stati effettuati con un periodo di clock pari a 10 ns, quindi una frequenza di 100 MHz.

```
1  library IEEE;
2  use ieee.std_logic_1164.all;
3
4  entity TestBench_CDMA_Receiver is
5  end TestBench_CDMA_Receiver;
6
7  architecture behavior of TestBench_CDMA_Receiver is
8
9      -- Constants
10     constant resetTime : time := 5 ns;
11     constant clockPeriod: time := 10 ns;
12     -- Spreading Factor, that is SymbolPeriod/ChipPeriod, following requirements this is equal to 16
13     constant SF : positive := 16;
14     constant K : positive := 2;
15
16     -- type of array to store codeWord, chipStream and interference pattern
17     type integerVector is array ( 0 to SF-1 ) of integer range -K to K;
18
19     component CDMA_Receiver
20     generic ( K : positive := 2;
21             SF : positive := 16 );
22     port(
23         clk_R : in std_ulogic;
24         reset_R : in std_ulogic;
25         code_word : in integer range -1 to 1 ;
26         chip_stream : in integer range -K to K;
27         bitstream_R : out std_ulogic
28     );
29 end component;
30
31 -- Inputs
32 signal clk_tb :std_logic := '0'; -- Clock
33 signal reset_tb: std_logic := '0'; -- Reset
34 signal code_word_tb: integer range -1 to 1 := 0;
35 signal chip_stream_tb: integer range -K to K := 0;
36
37 -- Outputs
38 signal bitstream_tb : std_ulogic; -- Received Bit
39
40 -- Others
41 signal stopSimulation : std_logic := '1';
42 begin
43     -- Clock variation
44     clk_tb <= (not(clk_tb) and stopSimulation) after clockPeriod/2;
45
46     --Reset
47     reset_tb <= '1' after resetTime;
48
49     -- CDMA Instantiation
50     test_CDMA_Receiver: CDMA_Receiver
51     generic map ( K => K, SF => SF )
52     port map ( clk_tb, reset_tb, code_word_tb, chip_stream_tb, bitstream_tb );
53
54     -- Stimulus Process
55     stim_proc: process
56     -- Code Word of User 1
57     constant code_word_1 : integerVector := ( 1,1,-1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,1);
58     -- Received Signal, simulating 2 signal interfering each other
59     constant interference_pattern: integerVector := (0,2,-2,2,-2,0,0,2,0,-2,0,0,2,0,0,2);
60
61     begin
62         wait until reset_tb = '1';
63         -- every clock period update inputs
64         for i in interference_pattern' range loop
65             code_word_tb <= code_word_1(i);
66             chip_stream_tb <= interference_pattern(i);
67             wait for clockPeriod;
68         end loop;
69         -- after ending inputs, simulation can be stopped
70         stopSimulation <= '0';
71     end process stim_proc;
72 end behavior;
```

Figura 5: Testbench 1: il trasmettitore invia il bit '1'

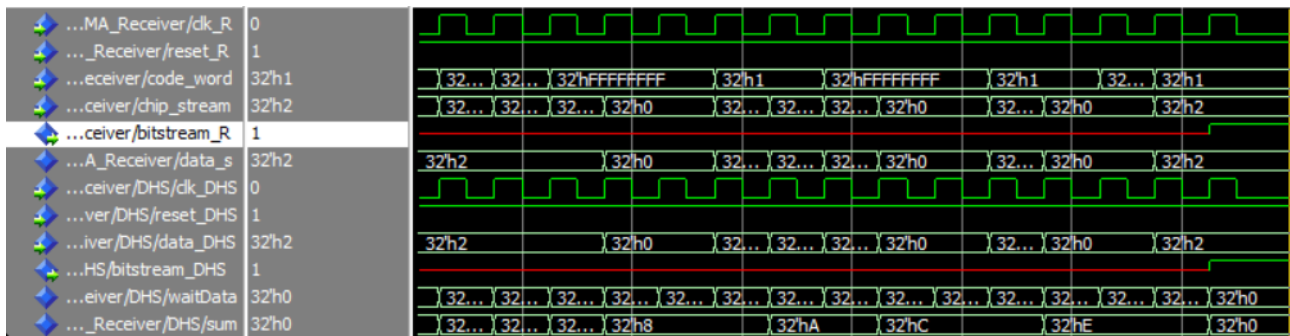


Figura 6: Simulazione ricezione 1

Come si può vedere in 6, l'uscita *bitstream* del ricevitore viene settata ad 1 una volta che tutto il chip stream è stato ricevuto.

2.3.2 Testbench 2

Di seguito è riportata solo la parte relativo allo Stimulus process del secondo test dato che il resto è uguale al primo.

```

49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

-- CDMA Instantiation
test_CDMA_Receiver2: CDMA_Receiver
generic map ( K => K, SF => SF )
port map ( clk_tb, reset_tb, code_word_tb, chip_stream_tb, bitstream_tb );

-- Stimulus Process
stim_proc: process
-- Code Word of User 2
constant code_word_2 : integerVector := (1,-1,1,-1,1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,-1);
-- Received Signal, simulating 2 signal interfering each other
constant interference_pattern: integerVector := (0,2,-2,2,-2,0,0,2,0,-2,0,0,2,0,0,2);
begin
wait until reset_tb = '1';
-- every clock period update inputs
for i in interference_pattern' range loop
code_word_tb <= code_word_2(i);
chip_stream_tb <= interference_pattern(i);
wait for clockPeriod;
end loop;
-- after ending inputs, simulation can be stopped
stopSimulation <= '0';
end process stim_proc;
end behavior;

```

Figura 7: Testbench 2: il trasmettitore invia il bit '0'

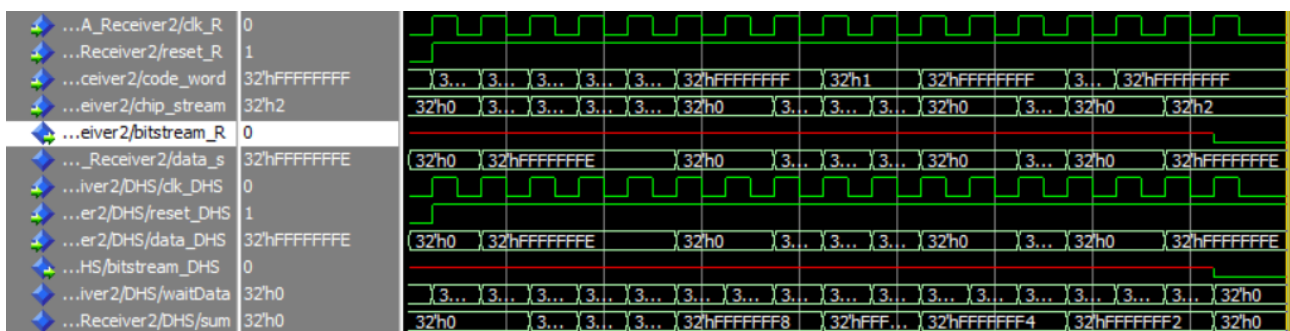


Figura 8: Simulazione ricezione 0

Il test conferma che il ricevitore 2 riesce decidere il bit 0 a partire dallo stesso chip stream ricevuto dal primo ricevitore.

3 Sintesi

La sintesi è stata effettuata tramite il tool Vivado applicando come unico constraint:

- Clock: 10ns

3.1 Utilizzo

Summary

Resource	Utilization	Available	Utilization %
LUT	12	17600	0.07
FF	12	35200	0.03
IO	8	100	8.00

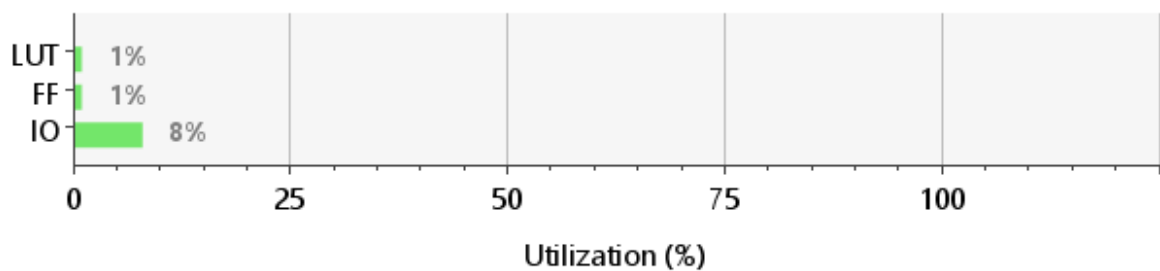


Figura 9: Percentuale di utilizzo

Le risorse utilizzate sono poche rispetto a quelle disponibili. Il maggior utilizzo si ha sull'I/O in quanto vengono utilizzati 8 pin su 100 disponibili (3 chipstream, 2 code word, 1 clock, 1 /reset, 1 bitstream).

3.2 Massima frequenza di utilizzo

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 8,116 ns	Worst Hold Slack (WHS): 0,064 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: 19	Total Number of Endpoints: 19	Total Number of Endpoints: 13

All user specified timing constraints are met.

Figura 10: Timing

Il ricevitore così descritto è in grado di rispettare il vincolo sul clock imposto ottenendo un **WNS** di 8.116 ns, quindi è possibile tollerare una frequenza massima pari a:

$$f_{max} = \frac{1}{T'_{clock}} = \frac{1}{T_{clock} - T_{slack}} = 530.78 \text{ MHz}$$

3.3 Cammino Critico

Il cammino critico che produce il T_{slack} mostrato in 10 è il seguente:

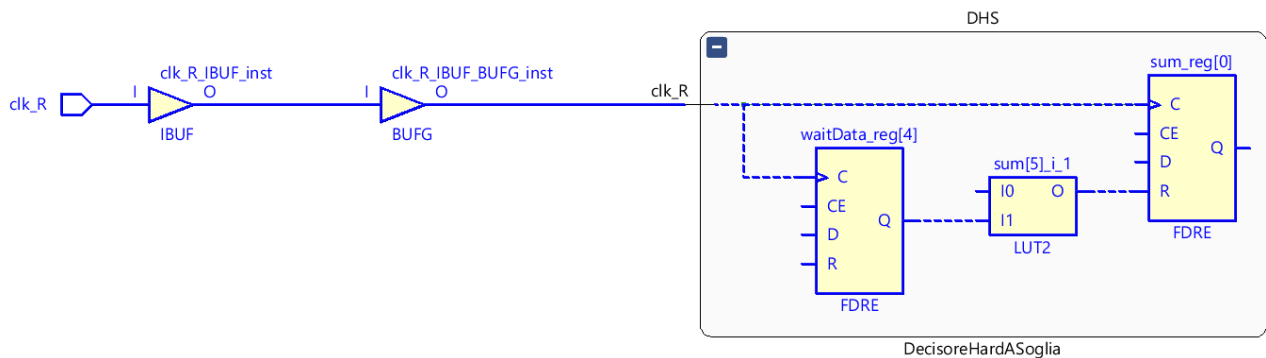


Figura 11: Cammino Critico

3.4 Consumo di Potenza

Dal consumo di potenza si può notare quanto l'algoritmo implementato ha necessità computazionali basse, dato che la maggior parte della potenza è richiesta a mantenere attivo il dispositivo più che ad effettuare le dovute operazioni.

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	0.093 W
Design Power Budget:	Not Specified
Power Budget Margin:	N/A
Junction Temperature:	26,1°C
Thermal Margin:	73,9°C (6,2 W)
Effective θ_{JA} :	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

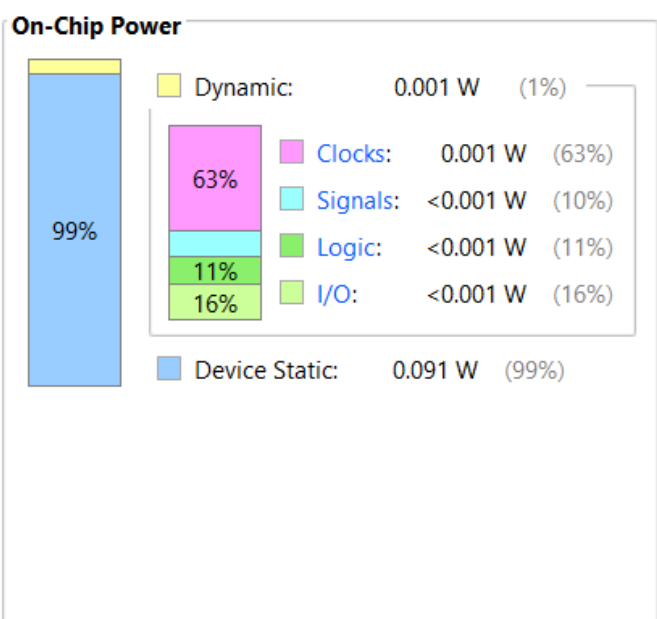


Figura 12: Percentuale Consumo di Potenza

4 Conclusioni

In questo progetto è stato descritto, implementato e sintetizzato un Ricevitore CDMA che riceve un segnale, somma dei segnali trasmessi in quell'istante, ma è in grado comunque di ricostruire il segnale trasmesso dal suo interlocutore nonostante le interferenze. I test hanno verificato il corretto funzionamento dei ricevitori, andando a valutare il comportamento in uno scenario realistico. Inoltre, la sintesi ha mostrato che il dispositivo è in grado di sopportare frequenze fino a 530.78 MHz quindi a tale frequenza e considerando uno *Spreading Factor* pari a 16, l'intera sequenza di *chip stream* è ricevuta in $T'_{clock} * 16 = 30.11$ ns.