UNIVERSITÀ DI PISA

Dipartimento di Ingegneria Dell'Informazione

CDMA Receiver

Progetto di Electronics and Comminications Systems

Amedeo POCHIERO

A.Y. 2019-2020

Indice

1 Introduzione

1.1 Descrizione del Problema

Il collegamento broadcast può avere più nodi trasmittenti 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 trasmittenti e riceventi in un canale broadcast condiviso, 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.

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.

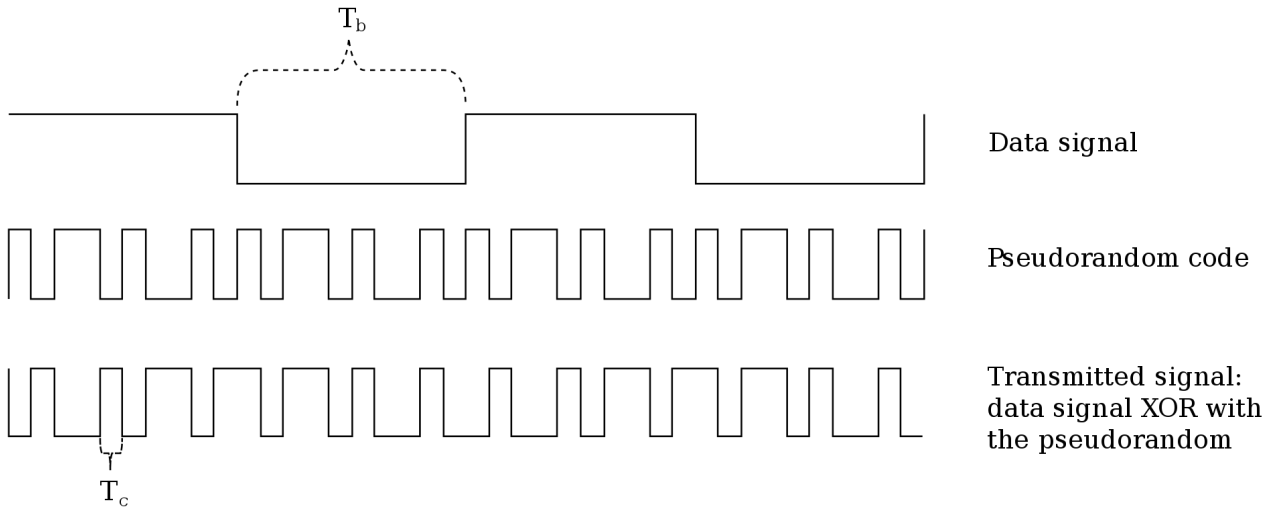

Figura 1: Generazione del segnale CDMA trasmesso

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 di ampiezza doppia, 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 realemente 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
interference pattern	0	2	-2	2	-2	0	0	2	0	-2	0	0	2	0	0	2

Tabella 2: Interferenza tra trasmettitori

Per ricostruire il segnale, 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 effetturanno 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

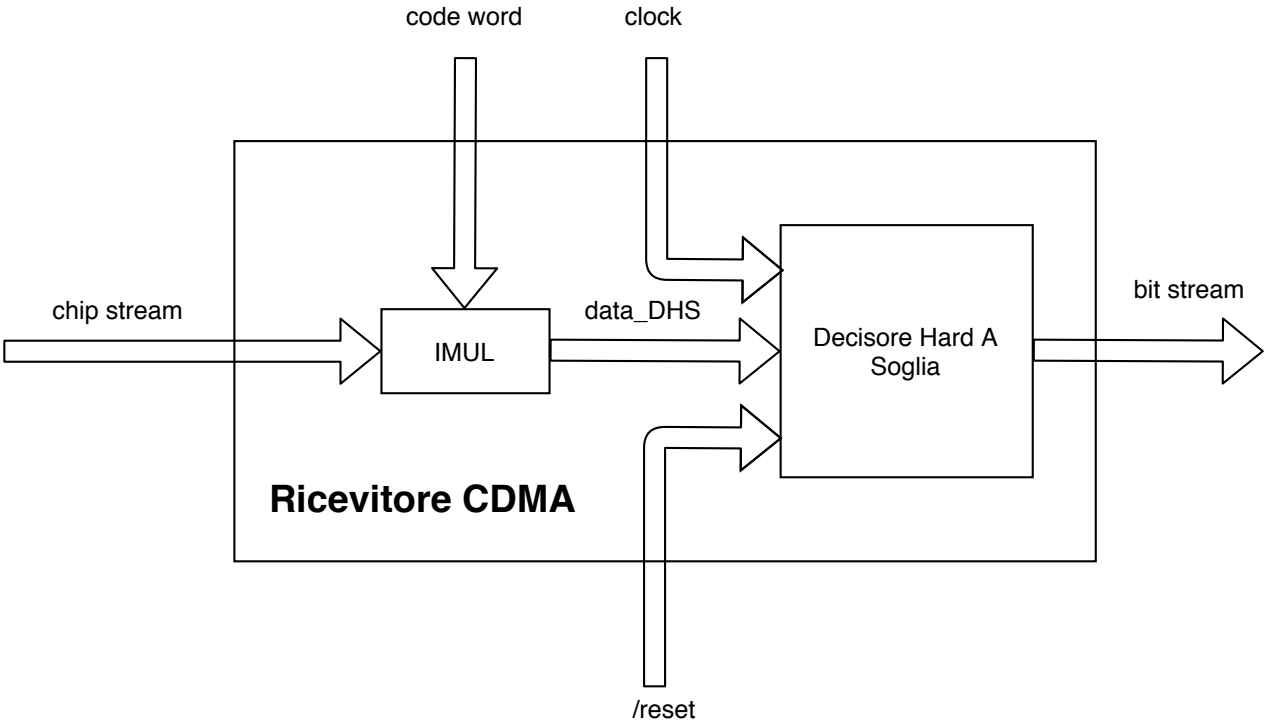

Figura 2: Diagramma a blocchi dell'architettura usata

2.2 Implementazione

```
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 DesisoreHardASoglia
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

```vhdl
library IEEE;
use ieee.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_unsigned.all;
-- This component has to take a decision on the output bit
entity DecisoreHardASoglia is
        generic ( K: positive := 2;
                  SF : positive := 16 );
        port(
                clk_DHS : in std_ulogic;
                reset_DHS : in std_ulogic;
                data_DHS : in integer range -K to K;
                bitstream_DHS : out std_ulogic
        );
end DecisoreHardASoglia;

architecture behavior of DecisoreHardASoglia is
        -- Count how many data have been received
        signal waitData : integer range 0 to SF := 0;
        -- Sum data at every clock cycle
        signal sum : integer range -SF to SF := 0;
        begin
                proc : process(clk_DHS)
                begin
                        if ( rising_edge(clk_DHS)) then
                                -- synchronous reset
                                if (reset_DHS = '0' ) then
                                        bitstream_DHS <= 'U';
                                else
                                        -- Update sum and waitData
                                        if ( waitData < 16 ) then
                                                sum <= sum + data_DHS;
                                                waitData <= waitData + 1;
                                        else
                                                -- otherwise make a decision on the bit
                                                if ( sum >= 0 ) then
                                                        bitstream_DHS <= '1';
                                                else
                                                        bitstream_DHS <= '0';
                                                end if;
                                                -- reset signals
                                                waitData <= 0;
                                                sum <= 0;
                                        end if;
                                end if;
                        end if;
                end process proc;
        end behavior;
```

Figura 4: Descrizione VHDL Decisore Hard A Soglia

2.3 Test-plan

```
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;
```

```
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              begin
61                      wait until reset_tb = '1';
62                      -- every clock period update inputs
63                      for i in interference_pattern' range loop
64                              code_word_tb <= code_word_1(i);
65                              chip_stream_tb <= interference_pattern(i);
66                              wait for clockPeriod;
67                      end loop;
68                      -- after ending inputs, simulation can be stopped
69                      stopSimulation <= '0';
70              end process stim_proc;
71      end behavior;
72
```

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

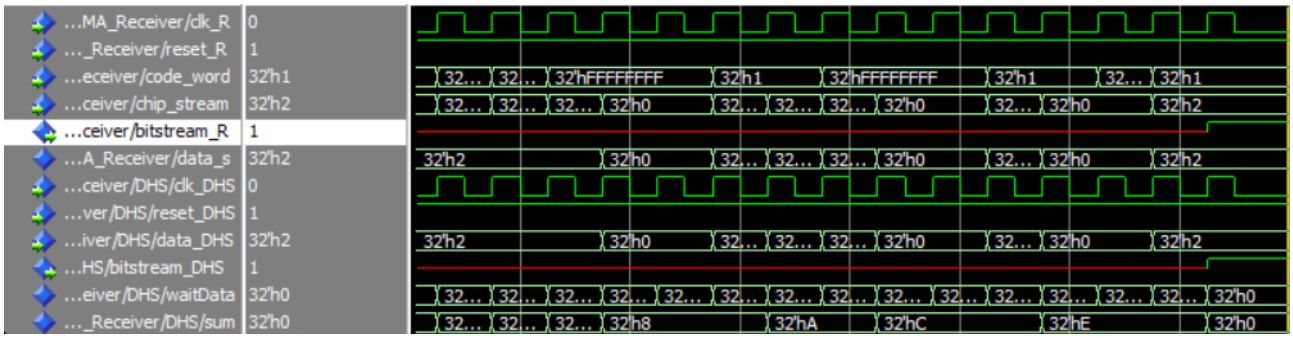

Figura 6: Simulazione ricezione 1

```
49              -- CDMA Instantiation
50              test_CDMA_Receiver2: 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 2
57                      constant code_word_2 : integerVector := (1,-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              begin
61                      wait until reset_tb = '1';
62                      -- every clock period update inputs
63                      for i in interference_pattern' range loop
64                              code_word_tb <= code_word_2(i);
65                              chip_stream_tb <= interference_pattern(i);
66                              wait for clockPeriod;
67                      end loop;
68                      -- after ending inputs, simulation can be stopped
69                      stopSimulation <= '0';
70              end process stim_proc;
71      end behavior;
```

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

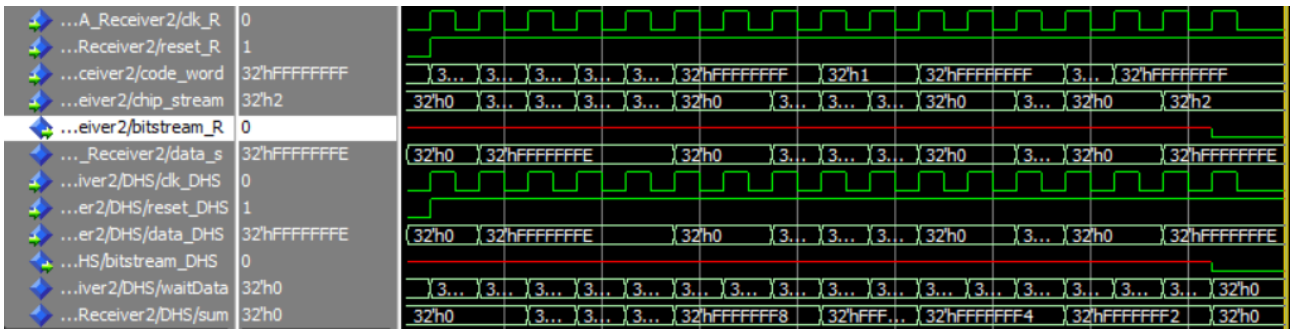

Figura 8: Simulazione ricezione 0

3 Sintesi

3.1 Utilizzo

Summary

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

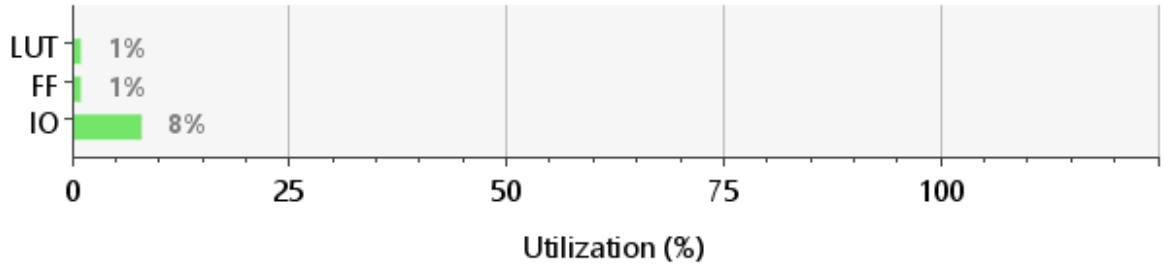

Figura 9: Percentuale di utilizzo

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

3.3 Cammino Critico

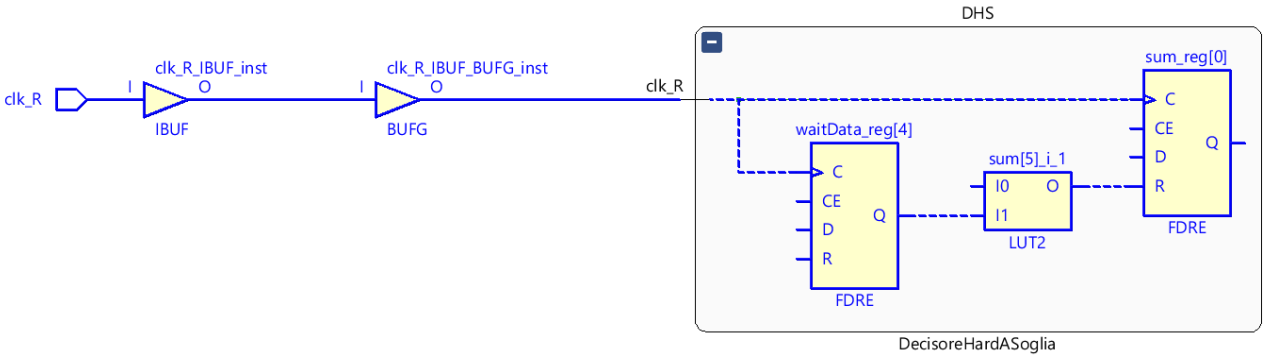

Figura 11: Cammino Critico

3.4 Consumo di Potenza

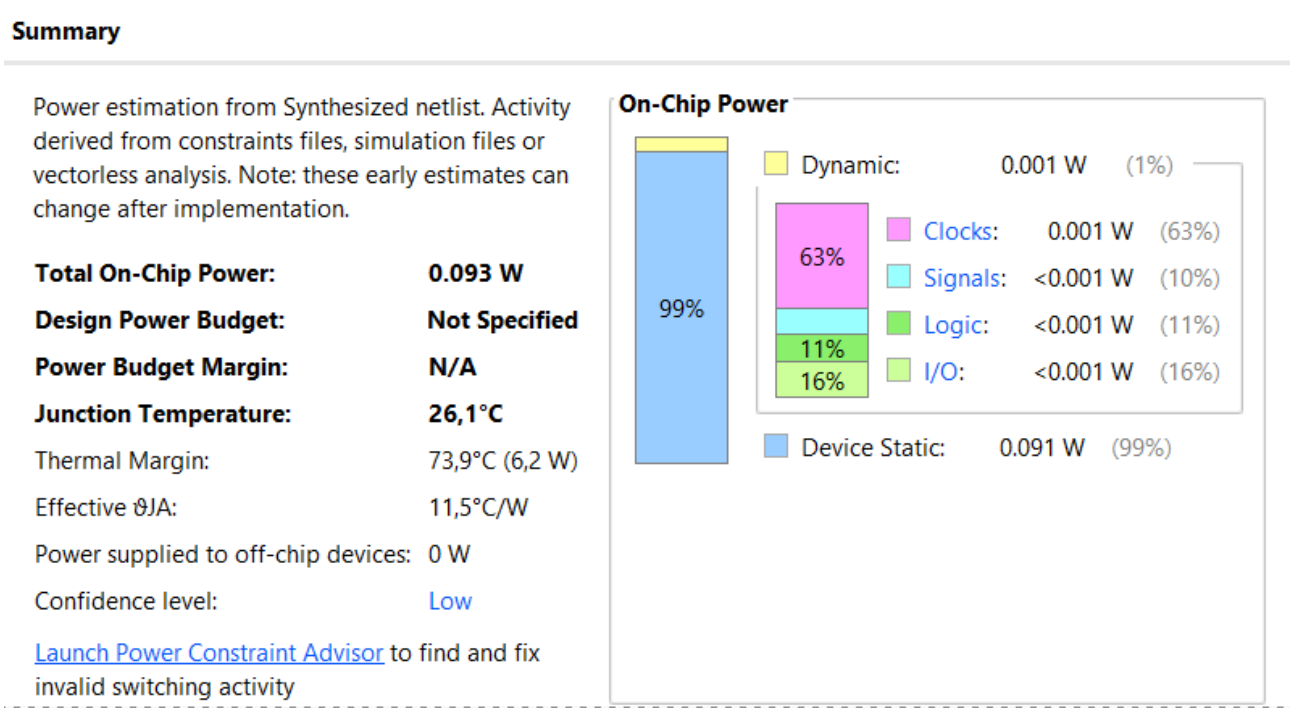

Figura 12: Percentuale Consumo di Potenza

4 Conclusioni