UNIVERSITÀ DI PISA

Progettazione di un ricevitore CDMA attraverso il linguaggio di descrizione dell' hardware VHDL

Studente:
Antonio
Rasulo

INTRODUZIONE:

L' architettura descrive un ricevitore CDMA. Il protocollo CDMA è un protocollo ad accesso multiplo di canale, cioè consente a più utenti di accedere al canale condiviso tra gli utenti stessi. In trasmissione, vengono create un numero di repliche del simbolo che l' utente vuole trasmettere pari allo spreading factor; successivamente le repliche vengono moltiplicate per un' opportuna parola di codice, detta chip. Ciascun utente ha un codice diverso ed i codici sono ortogonali fra di loro. Ciascun bit del codice può valere 1 o -1. In ricezione il segnale ricevuto, chiamato codeword, è costituito dalla somma vettoriale dei segnali trasmessi da ciascun utente. L' estrazione del simbolo trasmesso avviene moltiplicando la codeword con il codice associato, sommando i bit ottenuti dalla moltiplicazione e successivamente dividendo per lo spreading factor.

UTILIZZO:

Nell' ambito delle telecomunicazioni il protocollo CDMA è uno dei più diffusi nelle reti wireless.

ARCHITETTURA:

L' architettura ha 4 ingressi ed un' uscita.

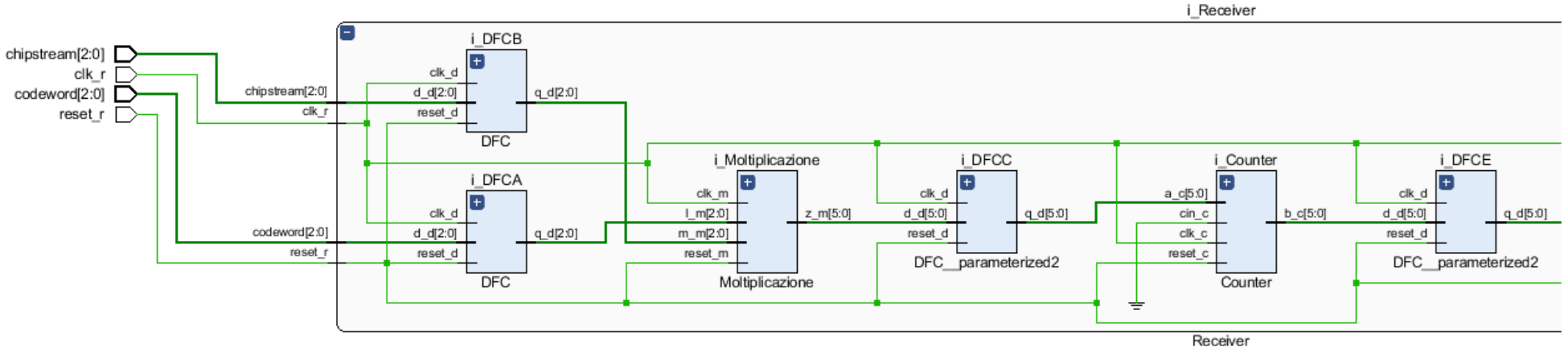
PORTE:

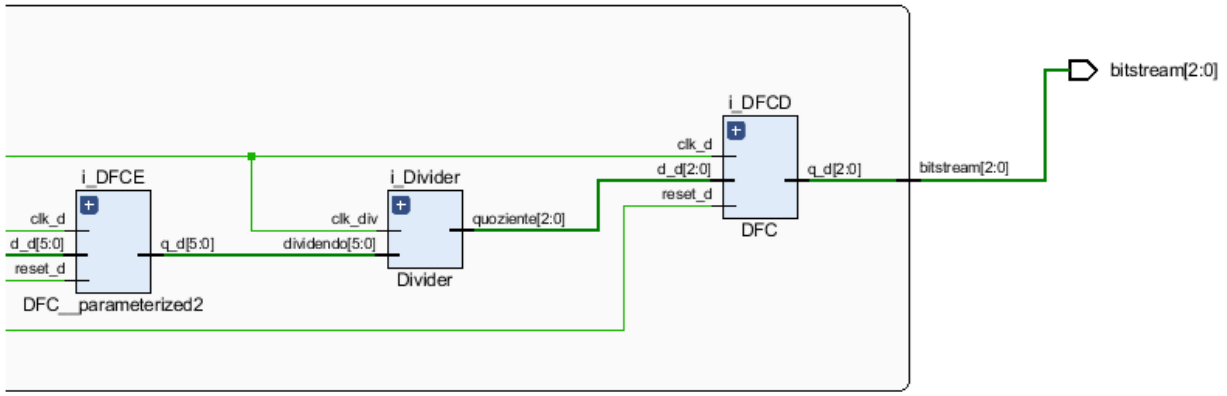
-Chipstream (I): è la parola di codice che va moltiplicata per il segnale informativo ricevuto;
-Codeword (I): è il segnale informativo che viene ricevuto;
-Clk_r (I): è il segnale di clock;
-Reset_r (I): è il segnale di reset;
-Bitstream (O): mi dice quale simbolo è stato trasmesso.

BLOCCHI:

L' architettura è principalmente formata dai seguenti blocchi:
-Moltiplicazione: lo scopo di questo blocco è quello di andare a moltiplicare il chipstream con la codeword;
-Counter: questo blocco somma i vettori ottenuti dalla moltiplicazione della codeword con il chipstream. Questo blocco è a sua volta formato dai blocchi RippleCarryAdder e DFC;
-Divider: divide per lo spreading factor il valore del segnale ricevuto in ingresso;
-DFC: registri, portano l' ingresso in uscita dopo un periodo di clock. Nell' architettura ci sono 6 registri, di cui uno all' interno del Counter.
Nel caso in cui il reset valga uno tutti i blocchi hanno come segnale di uscita zero.

Schematico dell'analisi register transfer level (RTL)





Codici VHDL

Ricevitore

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Receiver is                      --Descrizione dell'architettura del ricevitore CDMA
    generic (
            Nbit : positive:= 3;
            Nbit_c: positive:=6
    );
    port (
        codeword : in std_logic_vector(Nbit-1 downto 0);        -- Porta di ingresso che riceve il segnale informativo
        bitstream : out std_logic_vector(Nbit-1 downto 0);      -- Porta di uscita che dice quale simbolo è stato ricevuto
        chipstream : in std_logic_vector(Nbit-1 downto 0);      -- È la parola di codice
        reset_r : in std_logic;         -- Porta di ingresso per il reset
        clk_r : in std_logic            -- Porta di ingresso per il clock
    );
end Receiver;

architecture Struct of Receiver is

    signal s_cin: std_logic:='0';                       -- Segnale usato per mettere il carry di ingresso del counter a massa
    signal s_cout: std_logic:='0';                      -- Segnale usato per mettere il carry di uscita del counter a massa
    signal s_qa: std_logic_vector (Nbit-1 downto 0);    -- Segnale usato per collegare l'uscita del registro DFCA all'ingresso del moltiplicatore
    signal s_qb: std_logic_vector (Nbit-1 downto 0);    -- Segnale usato per collegare l'uscita del registro DFCB all'ingresso del moltiplicatore
    signal s_dc: std_logic_vector (Nbit_c-1 downto 0);  -- Segnale usato per collegare l'uscita del moltiplicatore all'ingresso del registro DFCC
    signal s_qc: std_logic_vector (Nbit_c-1 downto 0);  -- Segnale usato per collegare l'uscita del registro DFCC all'ingresso del contatore
    signal s_de: std_logic_vector (Nbit_c-1 downto 0);  -- Segnale usato per collegare l'uscita del contatore all'ingresso del registro DFCE
    signal s_qe: std_logic_vector(Nbit_c-1 downto 0);   -- Segnale usato per collegare l'uscita del registro DFCE all'ingresso del divisore
    signal s_dd: std_logic_vector (Nbit-1 downto 0);    -- Segnale usato per collegare l'uscita del divisore all'ingresso del registro DFCD
    signal s_qd: std_logic_vector (Nbit-1 downto 0);    -- Segnale in uscita al DFCD
```

```
component Moltiplicazione                                   -- Richiamo del componente moltiplicazione
    generic (Nbit : positive:= 3);
    port (
        clk_m       : in std_logic;                         -- Segnale di clock
        l_m     : in std_logic_vector (Nbit-1 downto 0);-- Operando della moltiplicazione
        m_m     : in std_logic_vector (Nbit-1 downto 0);-- Operando della moltiplicazione
        z_m     : out std_logic_vector((Nbit-1)*2+1 downto 0);-- Risultato della moltiplicazione
        reset_m : in std_logic                              -- Porta di ingresso reset
        );
end component;

component DFC                                               -- Richiamo del registro
    generic (Nbit : positive:=3);
    port (
        clk_d : in std_logic ;                              -- Segnale di clock
        reset_d : in std_logic ;                            -- Porta di ingresso reset
        d_d : in std_logic_vector(Nbit-1 downto 0) ;        -- Ingresso
        q_d : out std_logic_vector(Nbit-1 downto 0)         -- Uscita
    );
end component;

component Divider                                           -- Richiamo del componente Divider
    generic (
        Nbit : positive:=3;
        Nbit_c : positive:=6
    );
    port (
        dividendo: in std_logic_vector (Nbit_c-1 downto 0); -- Operando della divisione
        quoziente: out std_logic_vector (Nbit-1 downto 0);  -- Risultato della divisione
        clk_div: in std_logic                               -- Segnale di clock
    );
end component;
```

```vhdl
component Counter                                       -- Richiamo del componente Counter
    generic (Nbit_c :positive:=6);
    port (
        a_c : in std_logic_vector (Nbit_c-1 downto 0) ; -- Ingresso del contatore
        b_c : out std_logic_vector(Nbit_c-1 downto 0) ; -- Uscita del contatore
        cin_c : in std_logic;                           -- Carry in del contatore
        reset_c : in std_logic;                         -- Porta di ingresso reset
        clk_c : in std_logic;                           -- Porta di ingresso per il clock
        cout_c : out std_logic                          -- Carry out del contatore
    );
end component;
begin

            i_Moltiplicazione: Moltiplicazione          -- Definizione del blocco usato per la moltiplicazine
                    generic map(Nbit=>Nbit)
                    port map(                           -- Definizione dei collegamenti
                        clk_m => clk_r,
                        l_m => s_qa,
                        m_m => s_qb,
                        z_m => s_dc,
                        reset_m => reset_r
                    );

            i_DFCA: DFC                                 -- Definizione di uno dei due registri in ingresso al moltiplicatore
                generic map (Nbit=>Nbit)
                port map(                               -- Definizione dei collegamenti
                    d_d => codeword,
                    clk_d => clk_r,
                    reset_d => reset_r,
                    q_d => s_qa
                );
```

```
i_DFCB: DFC                             -- Definizione di uno dei due registri in ingresso al moltiplicatore
    generic map (Nbit=>Nbit)
    port map(                           -- Definizione dei collegamenti
        d_d => chipstream,
        clk_d => clk_r,
        reset_d => reset_r,
        q_d => s_qb
    );

i_DFCC: DFC                             -- Definizione del registro fra l'uscita del moltiplicatore e l'ingresso del contatore
    generic map (Nbit=>Nbit_c)
    port map(                           -- Definizione dei collegamenti
        d_d => s_dc,
        clk_d => clk_r,
        reset_d => reset_r,
        q_d => s_qc
    );

i_DFCD: DFC                             -- Definizione del registro fra l'uscita del blocco divisore e l'uscita del ricevitore
    generic map (Nbit=>Nbit)
    port map(                           -- Definizione dei collegamenti
        d_d => s_dd,
        clk_d => clk_r,
        reset_d => reset_r,
        q_d => s_qd
    );
```

```
        i_DFCE: DFC                                 -- Definizione del registro posto fra l'uscita del contatore e l'ingresso del divisore
            generic map (Nbit=>Nbit_c)
            port map(                               -- Definizione dei collegamenti
                d_d => s_de,
                clk_d => clk_r,
                reset_d => reset_r,
                q_d => s_qe
            );


        i_Counter: Counter                          -- Definizione del contatore
            generic map (Nbit_c=>Nbit_c)
            port map(                               -- Definizione dei collegamenti
                a_c => s_qc,
                b_c => s_de,
                cin_c => s_cin,
                reset_c => reset_r,
                clk_c => clk_r,
                cout_c => s_cout
            );
        i_Divider: Divider                          -- Definizione del divisore
            generic map (
                Nbit => Nbit,
                Nbit_c => Nbit_c
            )
            port map(                               -- Definizione dei collegamenti
                dividendo => s_qe,
                quoziente => s_dd,
                clk_div => clk_r
            );
        bitstream <= s_qd;                      -- Uscita del ricevitore
end Struct;
```

Moltiplicatore

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;           -- Questa libreria mi consente di fare il prodotto fra vettori
use ieee.std_logic_unsigned.all;

entity Moltiplicazione is
    generic (Nbit : positive:= 3);
    port (
        clk_m       : in std_logic;
        l_m     : in std_logic_vector (Nbit-1 downto 0);
        m_m     : in std_logic_vector (Nbit-1 downto 0);
        z_m     : out std_logic_vector((Nbit-1)*2+1 downto 0);
        reset_m : in std_logic
        );
end Moltiplicazione;

architecture rtl of Moltiplicazione is
begin

    Moltiplicazione_p: process(clk_m,reset_m)                           -- Se clk_m e/o reset_m cambiano allora il process viene attivato
    --variable product  : std_logic_vector((Nbit-1)*2+1 downto 0);
    begin
        if (reset_m= '1') then                                          -- Se reset_m è pari a '1' l'uscita del moltiplicatore è '0'

            z_m<=(others=>'0');

        elsif (clk_m'event and clk_m='1') then      -- altrimenti se c'è un fronte di salita del clock


                z_m<=l_m*m_m;                       -- l' uscita è il prodotto dei vettori in ingresso
        end if;
    end process Moltiplicazione_p;
```

Contatore

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Counter is                                       --Descrizione dell'architettura del contatore
    generic (Nbit_c : positive:= 6);
    port (
        a_c : in std_logic_vector (Nbit_c-1 downto 0) ;     -- Vettore in ingresso al contatore, si vuole ottenere la somma dei suoi elementi
        b_c : out std_logic_vector(Nbit_c-1 downto 0) ;     -- Vettore in uscita al contatore che contiene i risultati delle somme
        cin_c : in std_logic;                               -- Carry in in ingresso al contatore
        reset_c : in std_logic;                             -- Porta per il reset del contatore
        clk_c : in std_logic;                               -- Porta per il segnale di clock
        cout_c : out std_logic                              -- Carry out in uscita al contatore
    );
end Counter;

architecture Struct of Counter is
    signal s_s :std_logic_vector(Nbit_c-1 downto 0):=(others => '0');   -- Segnale usato per collegare l'uscita del ripple carry adder all'ingresso del registro
    signal q_s :std_logic_vector(Nbit_c-1 downto 0):=(others => '0');   -- Segnale usato per collegare l'uscita del registro ad uno dei due ingressi del ripple

    component RippleCarryAdder                                  -- Richiamo del componente ripple carry adder
        generic (Nbit : positive:= 6) ;
        port (
            a_rca : in std_logic_vector(Nbit-1 downto 0) ;          -- Porta di ingresso per uno dei due operandi della somma
            b_rca : in std_logic_vector(Nbit-1 downto 0) ;          -- Porta di ingresso per uno dei due operandi della somma
            cin_rca : in std_logic ;                                -- Porta per il carry in del ripple carry adder
            s_rca : out std_logic_vector (Nbit-1 downto 0) ;        -- Porta di uscita per il risultato della somma
            cout_rca : out std_logic;                               -- Porta per il carry out del ripple carry adder
            reset_rca: in std_logic                                 -- Porta di ingresso per il reset
        );
    end component;
```

```
component DFC
    generic (Nbit : positive:=6);
    port (
        clk_d : in std_logic ;
        reset_d : in std_logic ;
        d_d : in std_logic_vector(Nbit-1 downto 0) ;
        q_d : out std_logic_vector(Nbit-1 downto 0)
    );
end component;
begin
        i_RCA: RippleCarryAdder
            generic map(Nbit=>Nbit_c)
            port map(
                a_rca => a_c,
                b_rca => q_s,
                cin_rca => cin_c,
                s_rca => s_s,
                cout_rca => cout_c,
                reset_rca => reset_c
            );
        i_DFC: DFC
            generic map(Nbit=>Nbit_c)
            port map(
                clk_d => clk_c,
                reset_d => reset_c,
                d_d => s_s,
                q_d => q_s
            );
        b_c <= q_s;
end Struct;
```

Ripple Carry Adder

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity RippleCarryAdder is                                  -- Descrizione dell'architettura del RippleCarryAdder
    generic (Nbit : positive:= 6);
    port (
        a_rca   : in std_logic_vector (Nbit-1 downto 0) ;
        b_rca   : in std_logic_vector (Nbit-1 downto 0) ;
        cin_rca : in std_logic ;
        s_rca   : out std_logic_vector(Nbit-1 downto 0);
        cout_rca: out std_logic;
        reset_rca: in std_logic
        );
end RippleCarryAdder;
architecture beh of RippleCarryAdder is
begin
    combinational_p: process(a_rca,b_rca,cin_rca,reset_rca)      -- Il process viene attivato se uno o più degli elementi nella sensitivity list cambia
    variable c : std_logic_vector (Nbit+1 downto 0);            -- Definizione della variabile di supporto c
begin
    if (reset_rca='1') then                                  -- Se il reset vale '1' l'uscita del ripple carry adder viene impostata a '0'

        s_rca<=(others=>'0');

    else
            c(0) := cin_rca;                                 -- il bit meno significativo di c assume il valore del carry in
            for i in 0 to Nbit-1 loop                        -- vengono usate le formule del ripple carry adder secondo cui
                s_rca(i) <= a_rca(i) xor b_rca(i) xor c(i);  -- S(i)= P(i) xor C(i) dove P(i) = A(i) xor B(i)
                c(i+1) := (a_rca(i) and b_rca(i)) or (a_rca(i)and c(i)) or (b_rca(i) and c(i));     -- C(i+1)= A(i)and B(i) or A(i)and C(i) or B(i) and C(i)
            end loop;
            cout_rca <= c(Nbit);                             -- il bit più significativo di c è il carry out del ripple carry adder

    end if;
    end process combinational_p;
end beh;
```

DFC

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;

entity DFC is                                                   -- Descrizione dell'architettura del registro
    generic (Nbit : positive := 3) ;
    port (
        clk_d       : in std_logic;                             -- Porta di ingresso per il clock
        reset_d : in std_logic;                                 -- Porta di ingresso per il reset
        d_d     : in std_logic_vector (Nbit-1 downto 0) ;       -- Porta di ingresso per il dato informativo
        q_d     : out std_logic_vector (Nbit-1 downto 0):= (others => '0')  -- Porta di uscita per il dato informativo
        );
end DFC;

architecture rtl of DFC is

begin

    dfc_p: process(reset_d, clk_d)                              -- Se clk_d e/o reset_d cambia il processo viene attivato
    begin
        if reset_d='1' then                                     -- Se il reset_d è pari a '1' l'uscita del registro è il vettore nullo
            q_d <= ( others => '0' );
        elsif (clk_d'event and clk_d='1') then              -- altrimenti se c'è stato un fronte di salita del clock
                                                                -- porta l'ingresso in uscita
                q_d <= d_d;

        end if;
    end process dfc_p;

end rtl;
```

Divisore

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
use ieee.std_logic_unsigned.all;
entity Divider is
    generic (
            Nbit : positive:= 10;
            Nbit_c: positive:=10
    );
    port (
        dividendo: in std_logic_vector (Nbit_c-1 downto 0);
        quoziente: out std_logic_vector (Nbit-1 downto 0);
        clk_div: in std_logic
    );
end Divider;
architecture beh of Divider is
    constant divisore: integer :=16;                                    -- Spreading factor
begin
    Divider_p: process(clk_div, dividendo)                              -- Se clk_div e/o dividendo cambiano allora il process viene attivato
    variable input_div: integer;                                            -- Variabili usate nella divisione
    variable output_quo: integer;
    variable cont:integer:=0;                                           -- Contatore per il numero di cicli di clock
    begin
        if (clk_div'event and clk_div='1') then                         -- In presenza di un fronte di clock positivo
            input_div := to_integer(unsigned(dividendo));               -- Dividendo viene convertito in un intero senza segno
            output_quo := input_div / divisore;                         -- Si divide il valore del segnale in ingresso per lo spreading factor
            cont:=cont+1;                                               -- Il contatore incrementa
            if (cont=24) then               --Se ci sono stati 24 cicli di clock viene fornito il risultato della divisione in uscita
                quoziente <= std_logic_vector(to_unsigned(output_quo, quoziente'length));
            else
                quoziente <= ( others=> '0');
            end if;
        end if;
    end process Divider_p;
end beh;
```

Testbench e risultati delle simulazioni

La parte iniziale dei testbench è uguale per tutte le simulazioni, naturalmente ciò che cambiano sono gli ingressi che vengono dati al ricevitore CDMA. Quindi di seguito verrà riportata solo una volta la parte iniziale del testbench. Nelle simulazioni si è supposto che il numero di utenti sia due: all'utente 0 è associato il codice c0=1111111111111111 mentre all'utente 1 è associato il codice c1=1-11-111-1-1111-1-1-1-11.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.all;

entity Receiver_tb is
    end Receiver_tb;

architecture beh of Receiver_tb is

    constant clk_period : time  := 100 ns;
    constant Nbit   :positive   := 3;
    constant Nbit_c :positive   := 6;

    component Receiver
    generic (
            Nbit : positive:= 3;
            Nbit_c: positive:=6
        );
    port (
        codeword : in std_logic_vector(Nbit-1 downto 0);
        bitstream : out std_logic_vector(Nbit-1 downto 0);
        chipstream : in std_logic_vector(Nbit-1 downto 0);
        reset_r : in std_logic;
        clk_r : in std_logic
        );
    end component;
```

Simulazione #1

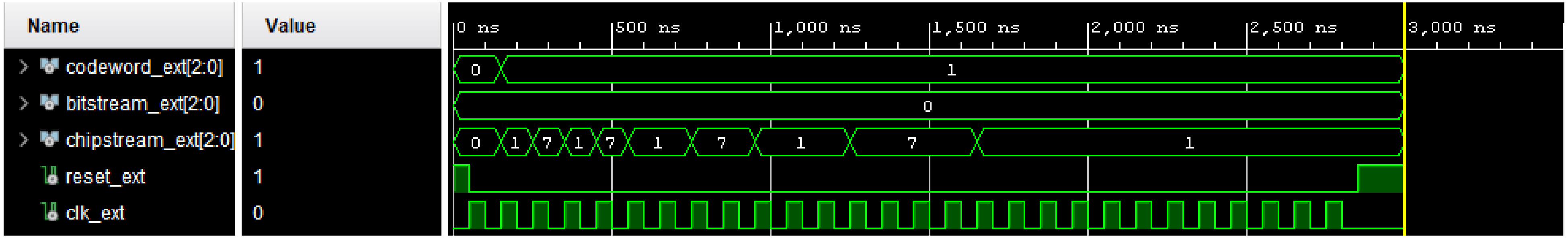
```
begin
    clk_ext <= not clk_ext after clk_period/2 when testing else '0';

    dut: Receiver
    generic map (
        Nbit_c => Nbit_c,
        Nbit => Nbit
        )
    port map(
        codeword => codeword_ext,
        bitstream => bitstream_ext,
        chipstream => chipstream_ext,
        reset_r => reset_ext,
        clk_r => clk_ext
        );
                stimulus : process
                    begin

                        reset_ext <= '1';
                        wait until rising_edge(clk_ext);
                        reset_ext <= '0';
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "001";
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "111";
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "001";
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "111";
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "001";
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "001";
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "111";
                        wait until rising_edge(clk_ext);
                        codeword_ext <= "001";
                        chipstream_ext <= "111";
```

```
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "111";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "111";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "111";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "111";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait for 1200 ns;
            testing <= false;
        end process;
    end beh;
```

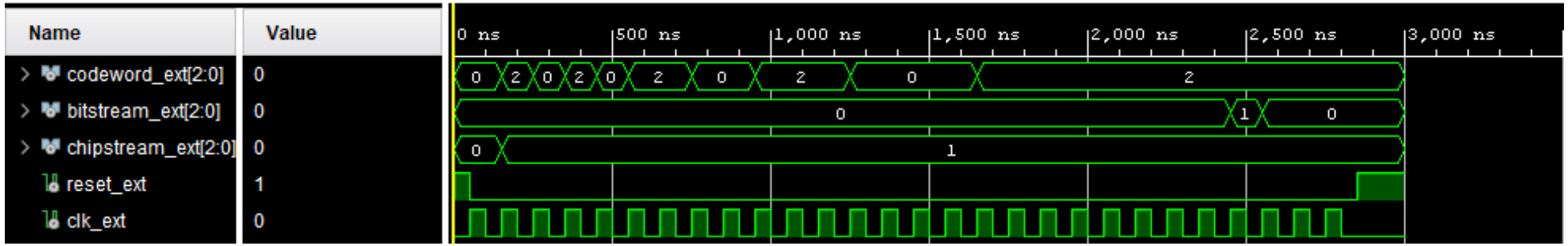
In questa simulazione viene ricevuto il segnale (cioè la codeword) a1*c0+a0*c1 dove a0 = 000 e a1 = 001 sono i simboli. Si è andato a moltiplicare il segnale per c1, così da ottenere in uscita il simbolo a0. Il ricevitore fornisce il simbolo dopo 24 periodi di clock.



Simulazione #2

```
stimulus : process
    begin

        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait for 1200 ns;
        testing <= false;
    end process;
end beh;
```



In questa simulazione viene ricevuto il segnale (cioè la codeword) a1*c0+a0*c1 dove a0 = 000 e a1 = 001 sono i simboli. Si è andato a moltiplicare il segnale per c0, così da ottenere in uscita il simbolo a1. Il ricevitore fornisce il simbolo dopo 24 periodi di clock.

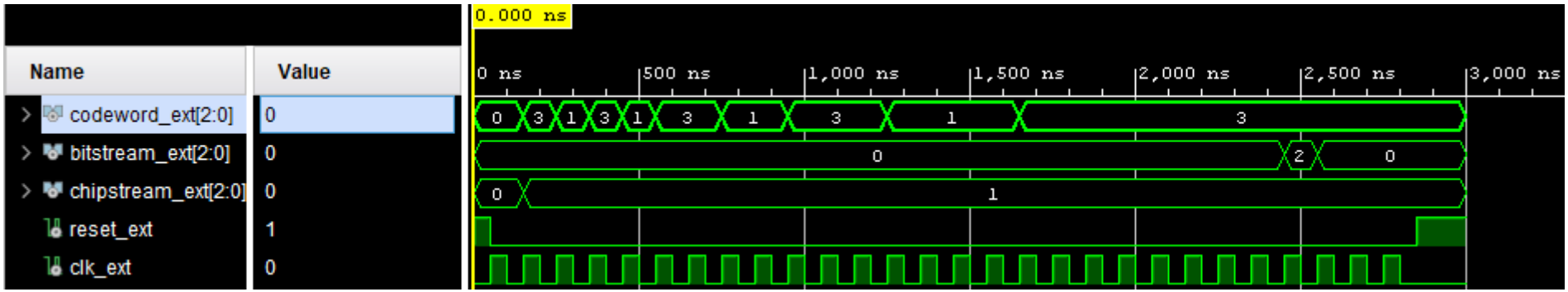
Simulazione #3

```
stimulus : process
    begin

        reset_ext <= '1';
        wait until rising_edge(clk_ext);
        reset_ext <= '0';
        wait until rising_edge(clk_ext);
        codeword_ext <= "011";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "011";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "011";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "011";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
        wait until rising_edge(clk_ext);
        codeword_ext <= "001";
        chipstream_ext <= "001";
```

```
            wait until rising_edge(clk_ext);
            codeword_ext <= "011";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "011";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "011";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "001";
            chipstream_ext <= "001";
            wait until rising_edge(clk_ext);
            codeword_ext <= "011";
            chipstream_ext <= "001";
            wait for 1200 ns;
            testing <= false;
    end process;
end beh;
```

In questa simulazione viene ricevuto il segnale (cioè la codeword) a0*c0+a1*c1 dove a0 = 010 e a1 = 001 sono i simboli. Si è andato a moltiplicare il segnale per c0, così da ottenere in uscita il simbolo a0. Il ricevitore fornisce il simbolo dopo 24 periodi di clock.



Fase di Sintesi

Timing report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,079 ns	Worst Hold Slack (WHS):	0,139 ns	Worst Pulse Width Slack (WPWS):	3,500 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:	56	Total Number of Endpoints:	56	Total Number of Endpoints:	63

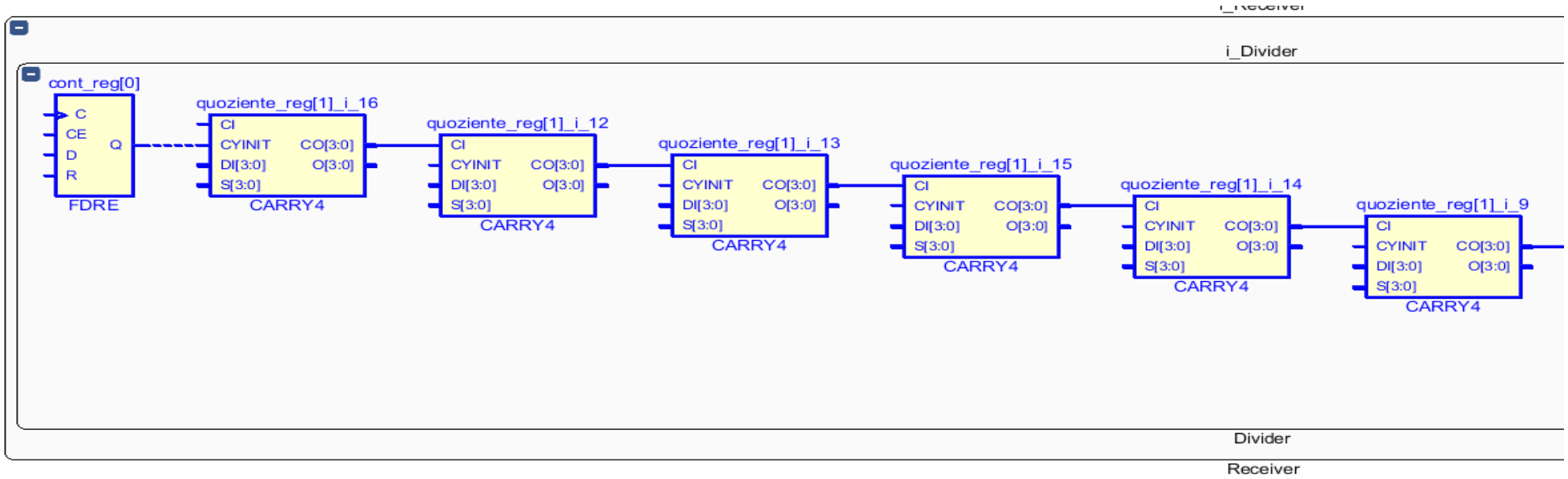
All user specified timing constraints are met.

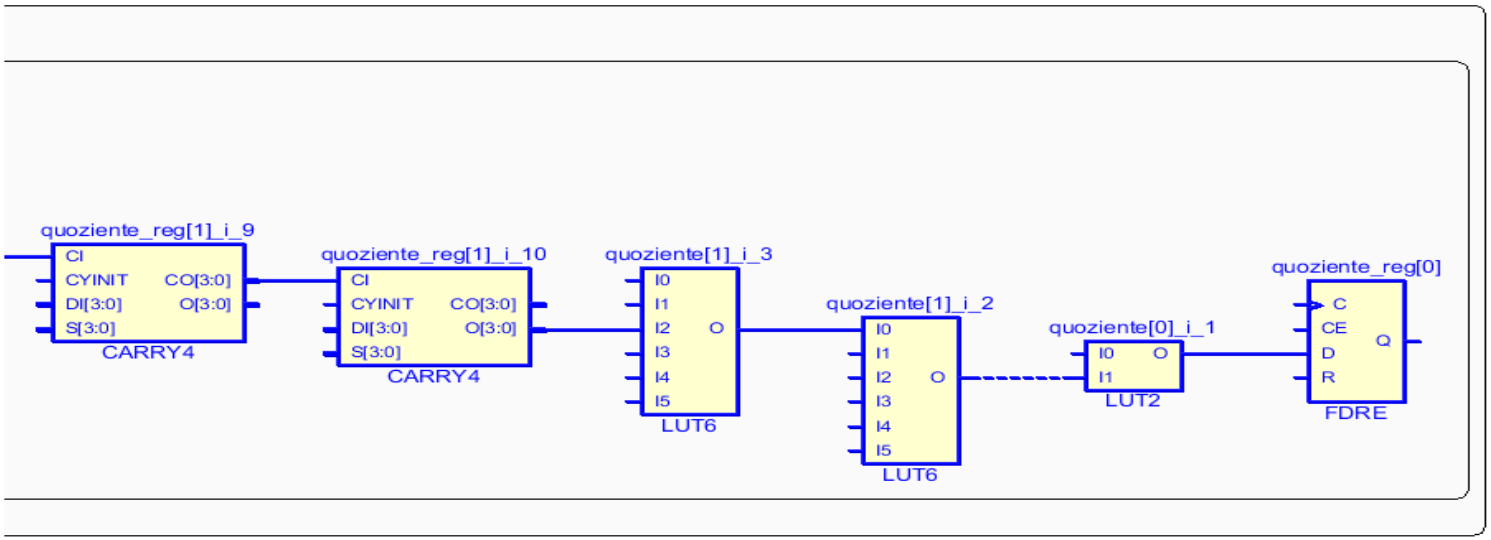
Fase di Implementazione

Risorse utilizzate

Name	Slice LUTs (17600)	Bonded IOB (100)	BUFGCTRL (32)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)
CDMA	22	11	1	62	28	22
i_Receiver (Receiver)	22	0	0		28	22
i_Counter (Counter)	1	0	0		3	1
i_DFCA (DFC)	2	0	0		3	2
i_DFCB (DFC_0)	1	0	0		2	1
i_DFCC (DFC__para...	6	0	0		6	6
i_DFCD (DFC_1)	0	0	0		1	0
i_DFCE (DFC__para...	0	0	0		1	0
i_Divider (Divider)	9	0	0		22	9
i_Moltiplicazione (Mo...	3	0	0		4	3

Critical Path





Design Timing Summary

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,624 ns	Worst Hold Slack (WHS):	0,131 ns	Worst Pulse Width Slack (WPWS):	3,500 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:	56	Total Number of Endpoints:	56	Total Number of Endpoints:	63

All user specified timing constraints are met.

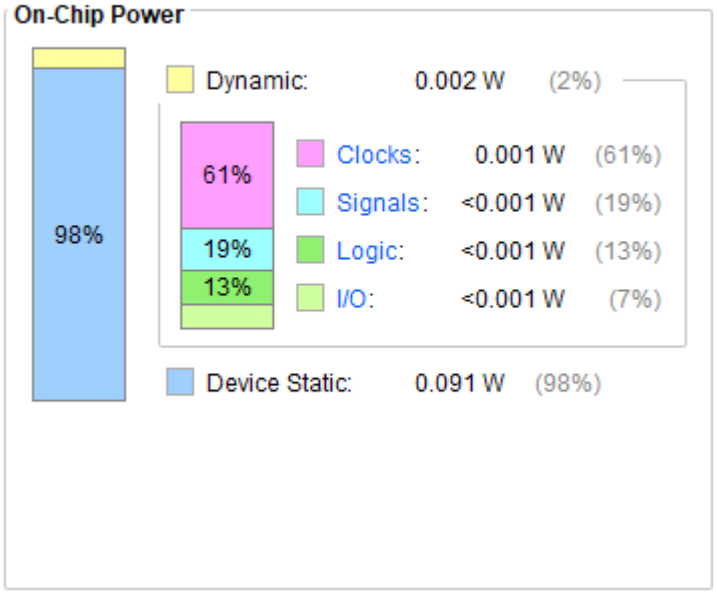
Report sulla potenza dissipata

Summary

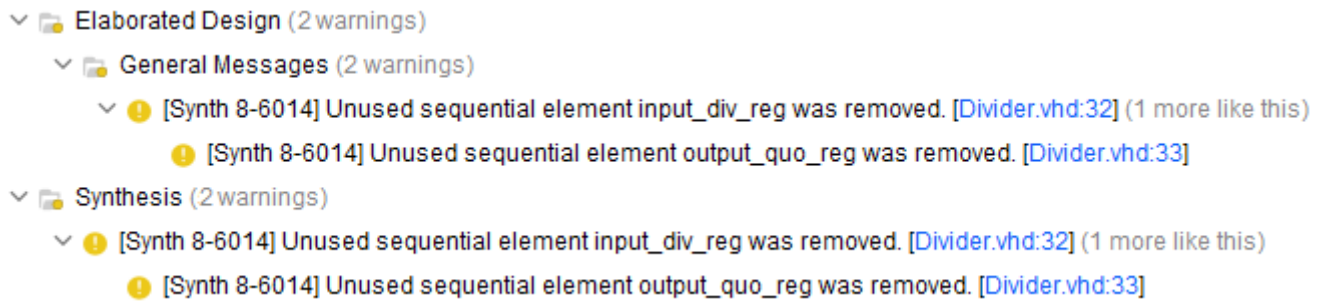
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.093 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,1°C**
Thermal Margin:	58,9°C (5,0 W)
Effective ϑJA:	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



Messaggi di warning



Questi messaggi di warning sono dovuti al fatto che durante la sintesi sono stati rimosse le variabili input_div e output_quo scritte nel codice del divisore. Queste variabili sono state usate per scrivere il codice in maniera più chiara per il progettista. Come è scritto nel codice del divisore:

```
if (clk_div'event and clk_div='1') then
        input_div := to_integer(unsigned(dividendo));
        output_quo := input_div / divisore;
        cont:=cont+1;
        if (cont=24) then
            quoziente <= std_logic_vector(to_unsigned(output_quo, quoziente'length));
        else
            quoziente <= ( others=> '0');
        end if;
```

Come poteva essere scritto il codice in modo da non far apparire i messaggi di warning:

```
if (clk_div'event and clk_div='1') then
        cont:=cont+1;
        if (cont=24) then
            quoziente <= std_logic_vector(to_unsigned(to_integer(unsigned(dividendo)) /
divisore, quoziente'length));
        else
            quoziente <= ( others=> '0');
        end if;
```

Si nota che il codice è più compatto ma di più difficile comprensione per il progettista.

Massima frequenza di lavoro

Il circuito non sta lavorando alla massima frequenza possibile perché il Worst Negative Slack, pari a 2.624 ns è positivo. La massima frequenza di lavoro è data da:

fmax = 1/(tclk – slack) = (10^9)/(100-2.624) ≈ 10.27 MHz