Tesina di Sistemi Embedded
Gruppo IV

Colella Gianni - Mat. M63/670 Guida Ciro - Mat. M63/592
Lombardi Daniele - Mat. M63/576

18 luglio 2017

Indice

Capitolo 1

Progetto finale Task 4

1.1 Premessa

Il Task 4 rappresenta il progetto finale da realizzare per l'esame di Sistemi Embedded del corso di Laurea Magistrale in Ingegneria Informatica, a. a. 2016/2017. Esso è inserito all'interno di un progetto più ampio, sviluppato in collaborazione con l'azienda Aster, che prevede l'implementazione su FPGA di un Radar Bistabile Passivo. A differenza dei radar attualmente utilizzati, detti attivi, questo tipo di radar non fa uso dell'apparato trasmissivo. Per adempiere alle proprie funzioni, questo dispositivo, non trasmettendo segnali, fa uso di segnali già presenti nell'etere. In particolare, sono stati scelti in fase progettuale i segnali utilizzati dal sistema **G**lobal **N**avigation **S**atellite **S**ystem **G**alileo. Il sistema si compone di 3 moduli principali:

1) Fase di Aquisizione;
2) Fase di Tracking;
3) Fase di Compressione.

Il Task in esame, incapsulato nella fase di Tracking, utilizzando i segnali **EaryGate** e **LateGate**, rappresentazioni del segnale primario anticipato e ritardato, fig. 1.1, fornisce una stima del valore **r**. Analizzando quest'ultimo e confrontandolo con una opportuna soglia, si è in grado di stabilire se l'oggetto, di cui si vuole conoscere la posizione, si sta allontanando o avvicinando dalla fonte trasmissiva del segnale. Viene, per questo motivo, utilizzato per correggere i valori di Delay, Frequenza Doppler e Fase.

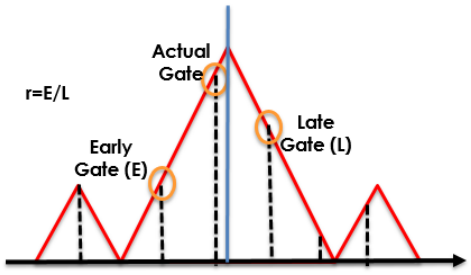


Figura 1.1: Early Gate e Late Gate

1.2 Traccia

Si realizzi un IP core che implementi il raffinamento del calcolo del delay che avviene durante la fase di Tracking, relativamente alla seconda parte durante la quale bisogna effettuare i moduli delle sommatorie, ottenute durante lo step precedente, calcolarne il rapporto e ricavarne la radice quadrata.

Per la realizzazione del task si richiede l'implementazione di:

1. moltiplicatore;
2. sommatore;
3. divisore;
4. radice quadrata.

Il Task, dunque, può essere rappresentato seguendo lo schema di principio di fig.1.2

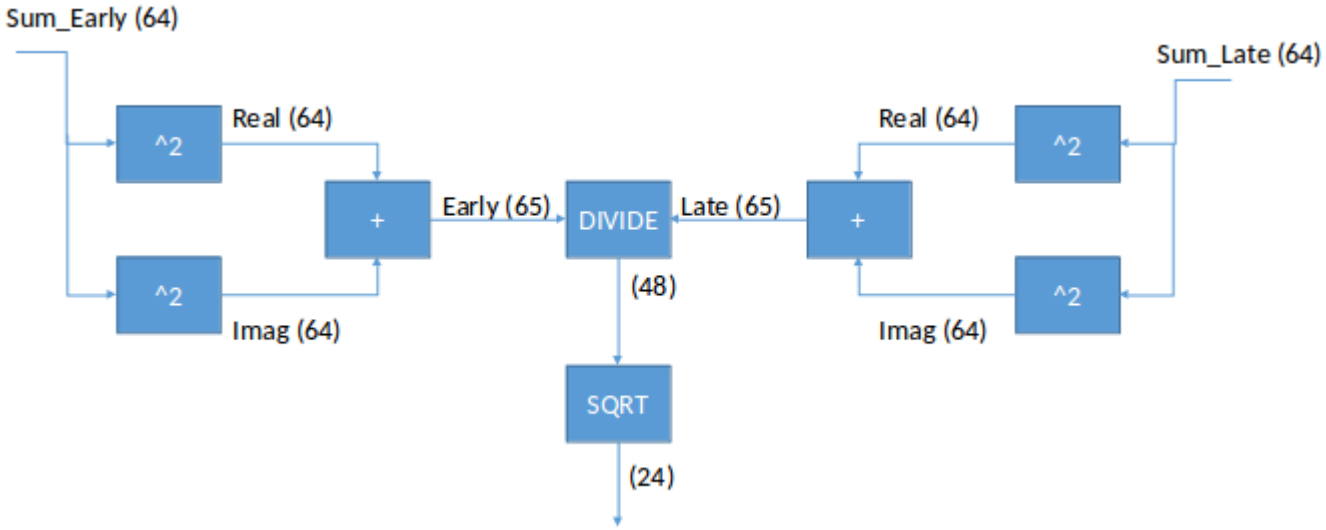


Figura 1.2: Schema di principio

L'obiettivo è quello di dispiegare tale componente su board FPGA Zybo.

1.3 Rappresentazione segnali e codice MATLAB

Il Task in esame accetta in ingresso due numeri complessi signed espressi su 64 bit che rappresentano i segnali **Sum_Early** e **Sum_Late** provenienti dal Task precedente. Essi devono subire una serie di manipolazioni per restituire in uscita un segnale signed fixed point espresso su 24 bit. Con riferimento allo schema di principio visto in precedenza, fig.1.2, il primo stadio è rappresentato da due componenti **Modulo Quadro**, fig. 1.4, che effettuano parallelamente il calcolo per i segnali Sum_Early e Sum_Late. Più precisamente, presi gli ingressi, ogni componente divide il segnale in parte reale e parte immaginaria, fig. 1.3. Ogni parte è trattata separatamente da un moltiplicatore.

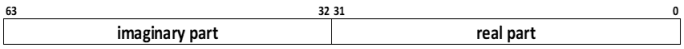


Figura 1.3: Rappresentazione dei segnali in ingesso

In uscita ogni moltiplicatore restituisce un segnale signed espresso su 64 bit. Le uscite dei due moltiplicatori sono gli ingressi di un sommatore che restituisce un segnale signed, espresso su 65 bit. Esso rappresenta il valore del modulo quadro del numero complesso. Tale segnale è signed, ma, considerato il fatto che la somma di due valori positivi è sicuramente positiva, si può elidere il bit più significativo del risultato che rappresenta il segno, e, dunque, viene considerato un intero unsigned espresso su 64 bit.

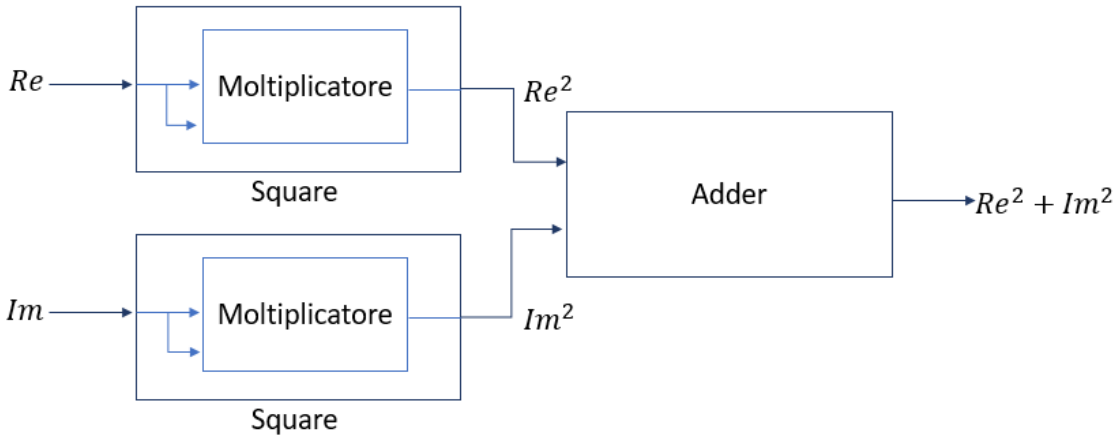


Figura 1.4: Componente Modulo Quadro

Succesivamente, i due segnali in uscita al primo stadio vengono processati da un divisore, fig. 1.5, il quale, secondo le specifiche fornite, restituisce un segnale rappresentato come fixed point a 48 bit, di cui gli 8 più significativi rappresentano la parte intera, i restanti 40 la parte decimale.

Figura 1.5: Componente Divisore

Quest'ultimo, dunque, diviene l'input da fornire all'ultimo stadio della catena, rappresentato da un componente che effettua il calcolo di **r**, 1.6. Il componente Radice quadrata fornisce in uscita un segnale unsigned rappresentabile su 24 bit dove i 4 bit più significativi rappresentano la parte intera, gli ultimi 20 quella decimale, fig. 1.3. Tale rappresentazione del segnale in uscita è stata presa in considerazione nella prima versione del Task proposta. Nelle successive versioni, a causa di un adeguamento delle specifiche di progetto, viene presa in considerazione una diversa rappresentazione del segnale di uscita basata su 24 bit signed, di cui 13 parte intera e 11 parte decimale, fig. 1.8.

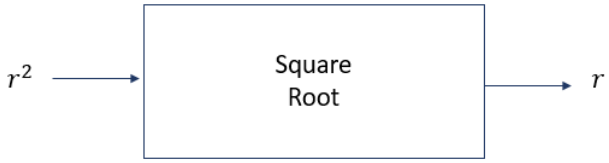


Figura 1.6: Componente Radice Quadrata

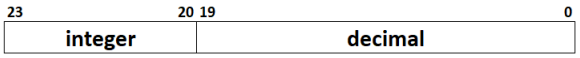


Figura 1.7: Rappresentazione di r <24,20>

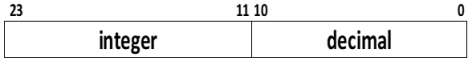


Figura 1.8: Rappresentazione di r <24,11>

Di seguito, si riportano due frammenti di codice MATLAB in cui sono inglobate le operazioni appena descritte.

```
64 %% Delay Deviation Estimation
65 for bbb = 1:nr_block
66 index_P=(1+(bbb-1)*sample_in_P:bbb*sample_in_P);% indeces of the samples in
       the primary code
67 Data_preConditioned(index_P)= Data_plus_Noise_Block(index_P).*DRR_in_P.*
       DRR_in_B(bbb)*SS_a_p(bbb); % secondary code stripping and doppler
       removal
68 r(bbb) = abs(sum(Data_preConditioned(index_P).*S_P_Early_1))/ ...
69         abs(sum(Data_preConditioned(index_P).*S_P_Late_1));  % gating
70 end
71 r_avg(bb) = mean(r);
```

Codice 1.1: Test2DelayDeviationAndAlignment.m

```
62 % Compute operations
63 reE2=real(sigEarly).^2;
64 imE2=imag(sigEarly).^2;
65 reL2=real(sigLate).^2;
66 imL2=imag(sigLate).^2;
67 sE=reE2+imE2;
68 sL=reL2+imL2;
69 d1=sE./sL;
70 R=sqrt(d1);
```

Codice 1.2: "T4dataGenerator.m"

1.4 Task 4 v1.0

Tale soluzione, che è anche quella più immediata da realizzare, consiste nell'utilizzare solamente IP core realizzati da terze parti: essi vengono forniti direttamente da Xilinx, gratuitamente e presenti nella suite di sviluppo Vivado.

1.4.1 Design

La soluzione prevede di istanziare i seguenti IP core :

- 4 **Multiplier** e 2 **Adder/Subtractor**, per realizzare i primi due componenti paralleli;
- 1 **Divider Generator**, per realizzare l'operatore di divisione;
- 1 **Cordic**, per realizzare l'operatore di radice quadrata.

La fig. 1.9 mostra le istanze di tali componenti e relativi collegamenti tra essi.

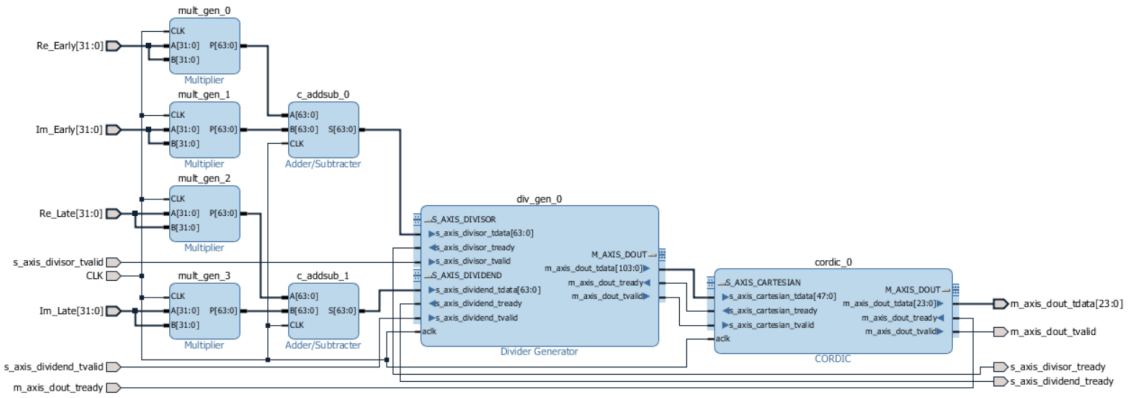


Figura 1.9: Block Diagram relativo ad una prima soluzione

1.4.2 Codice

Di seguito si riportano unicamente i codici VHDL scritti manualmente.

```
1 ----------------------------------------------------------------------------
2 --! @file     Task4_v1_0/src/Task4_m.vhd
3 --! @authors
4 --!              Colella Gianni      <gian.colella@studenti.unina.it>
      <br>
5 --!              Guida Ciro          <ciro.guida4@studenti.unina.it>
      <br>
6 --!              Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
      <br>
7 --! @version  V2.0
8 --! @date     17-July-2017
9 --! @copyright
```

```
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  Task4 is free software: you can redistribute it and/or modify
18 --!  it under the terms of the GNU Affero General Public License as
       published by
19 --!  the Free Software Foundation, either version 3 of the License, or
20 --!  (at your option) any later version.
21 --!
22 --!  Task4 is distributed in the hope that it will be useful,
23 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
24 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
25 --!  GNU Affero General Public License for more details.
26 --!
27 --!  You should have received a copy of the GNU Affero General Public
       License
28 --!  along with Linux Driver: Examples. If not, see
29 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
30 --! @brief Questo componente include tutte le funzionalità che deve eseguire
        il
31 --!        Task 4. In particolare, in ingresso al componente vengono forniti
32 --!        2 segnali complessi espressi su 64 bit (32 Im, 32 Re); in uscita,
33 --!        invece, è reso disponibile un segnale contenente la radice del
34 --!        rapporto del modulo quadro dei due segnali di ingresso. Tale
       segnale
35 --!    è espresso su 24 bit, di cui 13 sono la parte intera, 11 quella
36 --!    decimale. Tutto il componente è realizzato in modo tale da essere
37 --!        compatibile con interfaccia AXI4 Stream.
38 ----------------------------------------------------------------------------
39 library IEEE;
40 use IEEE.STD_LOGIC_1164.ALL;
41
42 entity Task4_m is
43     Port ( aclk : in STD_LOGIC;                                          --!
           Segnale di temporizzazione
44            aresetn : in STD_LOGIC;                                       --!
                 Reset sincrono, attivo basso
45            -- Interfaccia Slave del componente
46            s_axis_sum_early_tvalid :  in STD_LOGIC;                      --!
                 Se alto, il dato sum_early è valido
47            s_axis_sum_early_tready :  out STD_LOGIC;                     --!
                 Se alto il componente è pronto a ricevere sum_early
48            s_axis_sum_early_tdata : in STD_LOGIC_VECTOR (63 downto 0);   --!
```

```
                Segnale di input rappresentante sum_early
49              s_axis_sum_late_tvalid :  in STD_LOGIC;                         --!
                Se alto, il dato sum_late è valido
50              s_axis_sum_late_tready :  out STD_LOGIC;                        --!
                Se alto il componente è pronto a ricevere sum_late
51              s_axis_sum_late_tdata : in STD_LOGIC_VECTOR (63 downto 0);      --!
                Segnale di input rappresentante sum_late
52              -- Interfaccia Master del componente
53              m_axis_r_tvalid: out std_logic;                                 --!
                Se alto, il dato r in output è valido
54              m_axis_r_tready: in std_logic;                                  --!
                Se alto, il componente a valle è pronto a ricevere il dato r
55              m_axis_r_tdata : out STD_LOGIC_VECTOR (23 downto 0)             --!
                Segnale di output rappresentante r
56              );
57  end Task4_m;
58
59  architecture Structural of Task4_m is
60  ------------------------------------------------------------------------
61  ----------------------Absolute Square Component-------------------------
62  ------------------------------------------------------------------------
63      component AXI4_Stream_Absolute_Square_m is
64          Port ( aresetn : in STD_LOGIC;
65                 aclk : in STD_LOGIC;
66                 -- Interfaccia Slave del componente
67                 s_axis_value_tdata : in STD_LOGIC_VECTOR (63 downto 0);
68                 s_axis_value_tvalid : in STD_LOGIC;
69                 s_axis_value_tready : out STD_LOGIC;
70                 -- Interfaccia Master del componente
71                 m_axis_abssqr_tdata : out STD_LOGIC_VECTOR (63 downto 0);
72                 m_axis_abssqr_tvalid : out STD_LOGIC;
73                 m_axis_abssqr_tready : in STD_LOGIC);
74      end component;
75
76  ------------------------------------------------------------------------
77  --------------------------Divisor Component-----------------------------
78  ------------------------------------------------------------------------
79     component AXI4_Stream_Divider_m IS
80        PORT (
81          aclk : IN STD_LOGIC;
82          aresetn : IN STD_LOGIC;
83          -- Interfaccia Slave del componente
84          s_axis_divisor_tvalid : IN STD_LOGIC;
85          s_axis_divisor_tready : OUT STD_LOGIC;
86          s_axis_divisor_tdata : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
87          s_axis_dividend_tvalid : IN STD_LOGIC;
88          s_axis_dividend_tready : OUT STD_LOGIC;
89          s_axis_dividend_tdata : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
90          -- Interfaccia Master del componente
```

```
91          m_axis_dout_tvalid : OUT STD_LOGIC;
92          m_axis_dout_tready : IN STD_LOGIC;
93          m_axis_dout_tdata : OUT STD_LOGIC_VECTOR(103 DOWNTO 0)
94        );
95      END component;
96
97  ------------------------------------------------------------------------
98  -------------------------Square Root Component--------------------------
99  ------------------------------------------------------------------------
100     component AXI4_Stream_Square_Root_m is
101         Port ( aclk    : in   STD_LOGIC;
102                aresetn : in   STD_LOGIC;
103                -- Slave signal interface
104                s_axis_value_tvalid  : in   STD_LOGIC;
105                s_axis_value_tready   : out  STD_LOGIC;
106                s_axis_value_tdata  : in   STD_LOGIC_VECTOR (47 downto 0);
107                -- Master signal interface
108                m_axis_result_tvalid : out STD_LOGIC;
109                m_axis_result_tready : in STD_LOGIC;
110                m_axis_result_tdata : out  STD_LOGIC_VECTOR (23 downto 0));
111     end component;
112
113 --! Segnali ausiliari per i due componenti che realizzano il modulo quadro
        di Sum_Early e Sum_Late
114 signal late2buffer: std_logic_vector(63 downto 0);
115 signal late2tvalid : std_logic;
116 signal late2tready : std_logic;
117 signal early2buffer: std_logic_vector(63 downto 0);
118 signal early2tvalid : std_logic;
119 signal early2tready : std_logic;
120
121 --! Segnali ausiliari per il componente che realizza la divisione tra il
        modulo di Sum_Early e il modulo di Sum_Late
122 signal quozient_tdata : std_logic_vector(103 downto 0);
123 signal quozient_tvalid : std_logic;
124 signal quozient_tready : std_logic;
125
126 --! Segnale ausiliare per gestire il dato in uscita da rappresentare su 24
        bit,
127 --! di cui 13 costituiscono la parte intera e 11 quella decimale.
128 signal root_value : std_logic_vector (23 downto 0);
129
130 begin
131
132 ABS_SQR_EARLY:  AXI4_Stream_Absolute_Square_m
133         PORT MAP (
134                aresetn => aresetn,
135                aclk => aclk,
136                s_axis_value_tdata => s_axis_sum_early_tdata,
```

```
                s_axis_value_tvalid => s_axis_sum_early_tvalid,
                s_axis_value_tready => s_axis_sum_early_tready,
                m_axis_abssqr_tdata => early2buffer,
                m_axis_abssqr_tvalid => early2tvalid,
                m_axis_abssqr_tready => early2tready);

ABS_SQR_LATE:  AXI4_Stream_Absolute_Square_m
       PORT MAP (
               aresetn => aresetn,
               aclk => aclk,
               s_axis_value_tdata => s_axis_sum_late_tdata,
               s_axis_value_tvalid => s_axis_sum_late_tvalid,
               s_axis_value_tready => s_axis_sum_late_tready,
               m_axis_abssqr_tdata => late2buffer,
               m_axis_abssqr_tvalid => late2tvalid,
               m_axis_abssqr_tready => late2tready);

DIVIDER: AXI4_Stream_Divider_m
          PORT MAP(
                aclk => aclk,
                aresetn => aresetn,
                s_axis_divisor_tvalid => late2tvalid,
                s_axis_divisor_tready => late2tready,
                s_axis_divisor_tdata => late2buffer,
                s_axis_dividend_tvalid => early2tvalid,
                s_axis_dividend_tready => early2tready,
                s_axis_dividend_tdata => early2buffer,
                m_axis_dout_tvalid => quozient_tvalid,
                m_axis_dout_tready => quozient_tready,
                m_axis_dout_tdata => quozient_tdata);

SQUARE_ROOT: AXI4_Stream_Square_Root_m
        PORT MAP (
               aclk => aclk,
               aresetn => aresetn,
               s_axis_value_tvalid => quozient_tvalid,
               s_axis_value_tready => quozient_tready,
               s_axis_value_tdata => quozient_tdata(47 downto 0),
               m_axis_result_tvalid => m_axis_r_tvalid,
               m_axis_result_tready => m_axis_r_tready,
               m_axis_result_tdata => root_value);

m_axis_r_tdata <= "000000000" & root_value(23 downto 9);

end Structural;
```

Codice 1.3: "Task4v1.0"

```
--------------------------------------------------------------------------------
```

```
2  --! @file     Task4_v1_0/src/modulo_quadro.vhd
3  --! @authors
4  --!             Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5  --!             Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
6  --!             Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7  --! @version  V1.0
8  --! @date     17-July-2017
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  This file is part of Task4.
18 --!
19 --!  Task4 is free software: you can redistribute it and/or modify
20 --!  it under the terms of the GNU Affero General Public License as
       published by
21 --!  the Free Software Foundation, either version 3 of the License, or
22 --!  (at your option) any later version.
23 --!
24 --!  Task4 is distributed in the hope that it will be useful,
25 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
26 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
27 --!  GNU Affero General Public License for more details.
28 --!
29 --!  You should have received a copy of the GNU Affero General Public
       License
30 --!  along with Linux Driver: Examples. If not, see
31 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
32 --! @brief Il componente realizza il modulo quadro di un segnale complesso.
33 --!        Esso è realizzato utilizzando unicamente IP-Core messi a
       disposizione
34 --!        dalla Xilinx. Pertanto, la filosofia di progetto utilizzata è
       quella
35 --!        Strutturale.
36 ----------------------------------------------------------------------------
37 library IEEE;
38 use IEEE.STD_LOGIC_1164.ALL;
39
40 entity modulo_quadro is
41     Port ( Re_Early : in STD_LOGIC_VECTOR (31 downto 0);    --! Parte reale
```

```
        del segnale Sum_Early
42          Im_Early : in STD_LOGIC_VECTOR (31 downto 0);     --! Parte
               immaginaria del segnale Sum_Early
43          Re_Late : in STD_LOGIC_VECTOR (31 downto 0);      --! Parte reale
               del segnale Sum_Late
44          Im_Late : in STD_LOGIC_VECTOR (31 downto 0);      --! Parte
               immaginaria del segnale Sum_Late
45          Early2 : out STD_LOGIC_VECTOR (63 downto 0);      --! Modulo
               quadro del segnale Sum_Early
46          Late2 : out STD_LOGIC_VECTOR (63 downto 0);       --! Modulo
               quadro del segnale Sum_Late
47          reset: in std_logic;                              --! Reset
               sincrono, attivo alto
48          clk : in STD_LOGIC);                              --! Segnale di
               temporizzazione
49  end modulo_quadro;
50
51  architecture Behavioral of modulo_quadro is
52
53  ----------------------------------------------------------------------
54  ----------------------Moltiplicator Component-----------------------
55  ----------------------------------------------------------------------
56      component Square IS
57        PORT (
58          CLK : IN STD_LOGIC;
59          SCLR : IN STD_LOGIC;
60          A : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
61          B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
62          P : OUT STD_LOGIC_VECTOR(63 DOWNTO 0)
63        );
64      END component;
65
66  ----------------------------------------------------------------------
67  ---------------------------Adder Component----------------------------
68  ----------------------------------------------------------------------
69      COMPONENT c_addsub_0 IS
70        PORT (
71          A : IN STD_LOGIC_VECTOR(62 DOWNTO 0);
72          B : IN STD_LOGIC_VECTOR(62 DOWNTO 0);
73          CLK : IN STD_LOGIC;
74          SCLR : IN STD_LOGIC;
75          S : OUT STD_LOGIC_VECTOR(63 DOWNTO 0)
76        );
77      END COMPONENT;
78
79      --! Segnali ausiliari utilizzati per collegare i 2 moltiplicatori con l'
           addizionatore, per realizzare il singolo modulo quadro
80      signal ReL2_buffer: std_logic_vector(63 downto 0);  --! Quadrato della
           parte reale di Sum_Late
```

```
81      signal ImL2_buffer: std_logic_vector(63 downto 0);  --! Quadrato della
           parte immaginaria di Sum_Late
82      signal ReE2_buffer: std_logic_vector(63 downto 0);  --! Quadrato della
           parte reale di Sum_Early
83      signal ImE2_buffer: std_logic_vector(63 downto 0);  --! Quadrato della
           parte immaginaria di Sum_Early
84
85  begin
86
87  --! Moltiplicatore che realizza il quadrato della parte reale di Sum_Early
88  RE2E: Square
89    PORT map(
90      CLK=>clk,
91      A=>Re_Early,
92      B=>Re_Early,
93      SCLR=>reset,
94      P=>ReE2_buffer
95    );
96
97  --! Moltiplicatore che realizza il quadrato della parte immaginaria di
       Sum_Early
98  IM2E: Square
99    PORT map(
100     CLK=>clk,
101     A=>Im_Early,
102     B=>Im_Early,
103     SCLR=>reset,
104     P=>ImE2_buffer
105   );
106
107 --! Addizionatore che, sommando il quadrato della parte reale e immaginaria
       di Sum_Early, realizza il modulo quadro del segnale in oggetto
108 E2: c_addsub_0
109   PORT map(
110     CLK=>clk,
111     SCLR=>reset,
112     A=>ReE2_buffer(62 downto 0),
113     B=>ImE2_buffer(62 downto 0),
114     S=>Early2
115   );
116
117 --! Moltiplicatore che realizza il quadrato della parte reale di Sum_Late
118 RE2L: Square
119   PORT map(
120     CLK=>clk,
121     SCLR=>reset,
122     A=>Re_Late,
123     B=>Re_Late,
124     P=>ReL2_buffer
```

```
125   );
126
127 --! Moltiplicatore che realizza il quadrato della parte immaginaria di
      Sum_Late
128 IM2L: Square
129   PORT map(
130     CLK=>clk,
131     SCLR=>reset,
132     A=>Im_Late,
133     B=>Im_Late,
134     P=>ImL2_buffer
135   );
136
137 --! Addizionatore che, sommando il quadrato della parte reale e immaginaria
      di Sum_Late, realizza il modulo quadro del segnale in oggetto
138 L2: c_addsub_0
139   PORT map(
140     CLK=>clk,
141     SCLR=>reset,
142     A=>ReL2_buffer(62 downto 0),
143     B=>ImL2_buffer(62 downto 0),
144     S=>Late2
145   );
146
147 end Behavioral;
```

Codice 1.4: "modulo quadro"

1.4.3 Testbench

Per verificare il funzionamento, viene eseguito un semplice testbench, fig. 1.10, fornendo in input al Task una coppia di dati generati in MATLAB. Si può notare come il risultato finale viene restituito in uscita dopo 136 cicli di clock, impostando idealmente un clock in ingresso a 100 Mhz. In questa implementazione, l'uscita r è espressa in una forma fixed point unsigned <24,20>.

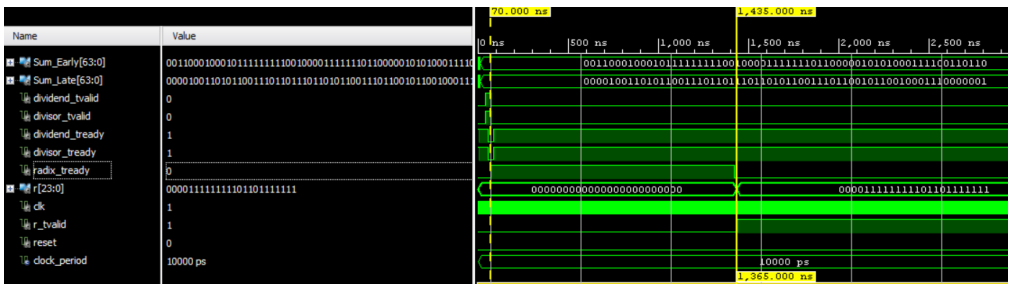


Figura 1.10: Testbench

1.4.4 Analisi soluzione

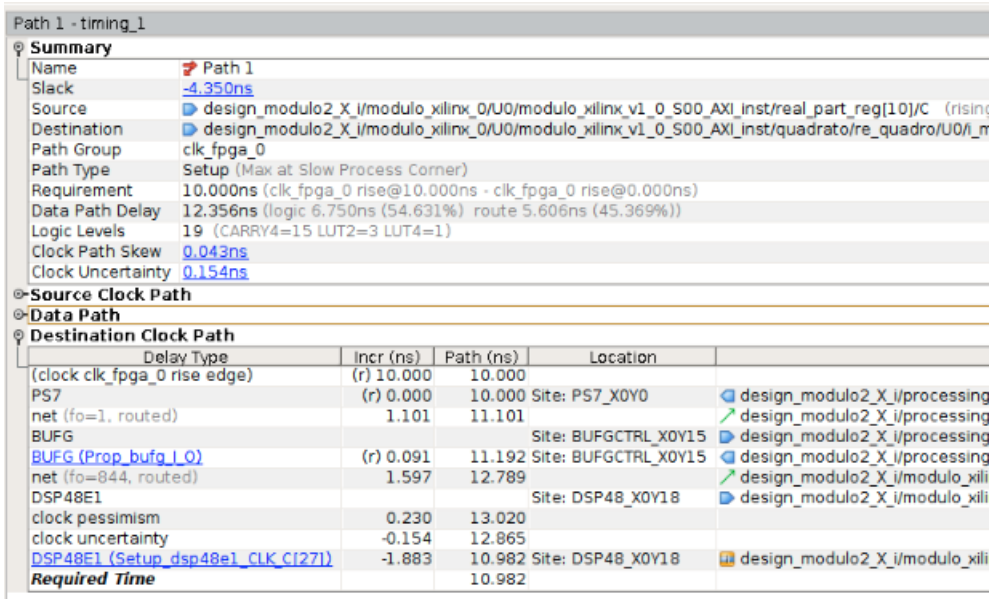
La soluzione adottata può essere analizzata considerando 2 parametri fondamentali: area occupata, frequenza massima di lavoro del circuito.

Per quanto riguarda l'area occupata dal componente complessivo, viene fornita la seguente tabella che riassume l'occupazione dovuta ai singoli IP core dispiegati, nonché quella relativa all'intero Task.

Componente	LUT	Slice Register	DSP48
Multiplier	1103	64	0
Adder/Subtractor	63	64	0
Divider generator	2036	4474	0
Cordic	673	892	0
Totale	7248(41%)	5750(16%)	0(0%)

Tabella 1.1: Occupazione d'area post-sintesi

Effettuando un'analisi post-implementation, fornendo ad ogni componente un constraint fisico temporale (clock dell'FPGA) si scopre che il modulo che lavora a più basse frequenze, è quello che implementa l'operazione di modulo quadro, che accetta una frequenza massima $69,686Mhz$. Come si può infatti notare dalla fig. 1.11, il path con il più alto delay presenta una slack negativa di $-4.350ns$ da dover aggiungere in modulo al periodo di clock di riferimento.


Path 1 - timing_1

Summary

Name	Path 1
Slack	-4.350ns
Source	design_modulo2_X_i/modulo_xilinx_0/U0/modulo_xilinx_v1_0_S00_AXI_inst/real_part_reg[10]/C (risin
Destination	design_modulo2_X_i/modulo_xilinx_0/U0/modulo_xilinx_v1_0_S00_AXI_inst/quadrato/re_quadro/U0/i_m
Path Group	clk_fpga_0
Path Type	Setup (Max at Slow Process Corner)
Requirement	10.000ns (clk_fpga_0 rise@10.000ns - clk_fpga_0 rise@0.000ns)
Data Path Delay	12.356ns (logic 6.750ns (54.631%) route 5.606ns (45.369%))
Logic Levels	19 (CARRY4=15 LUT2=3 LUT4=1)
Clock Path Skew	0.043ns
Clock Uncertainty	0.154ns

Source Clock Path

Data Path

Destination Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	
(clock clk_fpga_0 rise edge)	(r) 10.000	10.000		
PS7	(r) 0.000	10.000	Site: PS7_X0Y0	design_modulo2_X_i/processing
net (fo=1, routed)	1.101	11.101		design_modulo2_X_i/processing
BUFG			Site: BUFGCTRL_X0Y15	design_modulo2_X_i/processing
BUFG (Prop_bufg_I_O)	(r) 0.091	11.192	Site: BUFGCTRL_X0Y15	design_modulo2_X_i/processing
net (fo=844, routed)	1.597	12.789		design_modulo2_X_i/modulo_xili
DSP48E1			Site: DSP48_X0Y18	design_modulo2_X_i/modulo_xili
clock pessimism	0.230	13.020		
clock uncertainty	-0.154	12.865		
DSP48E1 (Setup_dsp48e1_CLK_C[27])	-1.883	10.982	Site: DSP48_X0Y18	design_modulo2_X_i/modulo_xili
Required Time		10.982		

Figura 1.11: Most negative slack

Tale frequenza, chiaramente, sarà la stessa a cui viene garantito il corretto funzionamento dell'intero circuito.

1.4.5 Vantaggi e svantaggi

Essendo la soluzione totalmente composta da IP core Xilinx, si ottengono una serie di vantaggi come la velocità di dispiegamento della soluzione, l'affidabilità di ogni componente utiizzato, infine, trattandosi appunto di proprietà intellettuali è significativa la facilità nel riuso dei componenti.

Di contro, poiché il primo stadio della catena, a differenza dei successivi, non è dotato di interfaccia AXI Stream, è necessario gestire esternamente i segnali di tvalid e tready sull'interfaccia slave del Divider Generator. Inoltre, dalla tab.1.1, è possibile notare come la soluzione proposta occupa molte risorse.

1.5 Analisi singola su ogni componente

In questa sezione si propone una panoramica su ogni componente della pipeline, prendendo in considerazione per ognuno di essi una serie di alternative e cercando di trovare una combinazione "vincente" in termini di area e spazio. Tutto questo non implica che la prima soluzione proposta è da scartare a priori.

La seguente tabella mostra per ogni componente le soluzioni prese in considerazione.

Per ogni singola proposta, viene effettuato uno studio in termini di occupazione d'area, frequenza massima di lavoro e numero di cicli di clock necessari per avere il risultato dell'operazione.

Figura 1.12: Componenti analizzati

1.5.1 Modulo quadro

Così come visto in tab.**??**, per tale componente vengono prese in considerazioni quattro possibili soluzioni. Si ricorda che tale componente, nella catena di elaborazione del task, è quello istanziato nel primo stadio della catena, dove parallelamente vengono calcolati i moduli quadri dei segnali Sum_Early e Sum_Late.

Realizzazione	LUT	Slice Register	DSP48	F.max	Cicli di clock
2 Multiplier Xil + Add/Sub Xil	1283	106	0	89.896 MHz	2
* e +	158	0	8	80.901 MHz	2
2 Booth + Ripple Carry Adder	425	264	0	82.967 MHz	128
2 MAC + Ripple Carry Adder	5468	128	0	20.927 MHz	2

Tabella 1.2: Occupazione d'area post-implementation

Dalla tabella si evince come il componente migliore in termini di occupazione sia quello relativo alla sua descrizione dataflow, in quanto il sintetizzatore **UG901** di Vivado, accorgendosi di operatori matematici, li va ad inferire sui DSP.

Per quanto riguarda invece le prestazioni, la scelta del migliore ricade su quello composto da IP core Xilinx con una frequenza massima di lavoro poco inferiore ai 90 MHz. Poiché 3 soluzioni su 4 sono puramente combinatorie, bufferizzando opportunamente ingressi e uscite, il numero di cicli di clock necessari ad avere il risultato pronto è pari a 2. Tranne per la soluzione con moltiplicatore a celle MAC che risulta essere la scelta nettamente peggiore in termini di area e prestazioni, le restanti 3 potrebbero essere prese tranquillamente in considerazione per la realizzazione del Task. Trovando un buon compromesso tra area e frequenza di lavoro, il componente migliore risulta essere quello relativo alla sua descrizione dataflow.

1.5.1.1 Aggiunta macchina a stati

Avendo scelto un componente combinatorio per il calcolo del modulo quadro nel primo stadio, per ovviare ad uno dei problemi visti nella prima proposta di soluzione, è necessario aggiungere una parte di controllo in modo tale che il componente abbia un'interfaccia del tutto compatibile con un bus AXI Stream, almeno per quanto riguarda i segnali tdata, tvalid e tready. Dunque si wrappa il componente e si aggiungono degli ulteriori segnali, oltre a quelli già presenti. Idealmente, seguendo la filosofia di un generico componente AXI Stream, l'interfaccia black box viene differenziata in Slave e Master. Sull'interfaccia slave vengono posti 3 segnali di cui 2 in ingresso ed 1 in uscita, rispettivamente di *tdata*, di *tvalid* e di *tready*. Il segnale di input *tvalid* indica il fatto che se esso è asserito, il dato in ingresso è valido e la macchina può iniziare ad elaborarlo; viceversa, il segnale di output *tready*, se pari ad 1, indica che la macchina è pronta ad accettare un nuovo dato, affinché possa essere processato. Sull'interfaccia master invece vi sono 2 segnali in output e 1 di input, in particolare oltre al dato in uscita vi è lo stesso segnale *tvalid* come sullo slave che indica un dato pronto in uscita ed uno *tready*, il quale indica che un'eventuale dato in uscita è pronto ad essere accettato dal componente posto a valle della catena.

Entrando nei dettagli, la parte di controllo è interpretata come FSM composta da 4 stadi e può essere descritta come segue: a partire dallo stato inziale di **RESET**, essa vi permane finché l'eventuale segnale di reset non viene posto a 1, facendo transitare la macchina nello stato di **IDLE**. In questa fase si attende che il segnale di input *tvalid* divenga 1 in modo tale che la macchina combinatoria processi un dato valido, memorizzato appositamente in un buffer di input. Nel transire la macchina passa per lo stato di **RESULT_CALCULATION**, dove viene abilitato il buffer in uscita per la memorizzazione del risultato. Successivamente, si entra nello stato di **WAIT_M_TREADY** in cui si attende che il componente a valle segnali che esso è pronto ad accettare un nuovo valore, quindi, se il segnale di ready è pari a 1, allora la macchina torna nello stato di IDLE in attesa di un nuovo dato da elaborare. Si fa notare come da qualsiasi stato è possibile tornare nello stato di RESET all'attivazione dell'omonimo segnale.

In fig. si propone il diagramma a bolle di quanto descritto sopra.

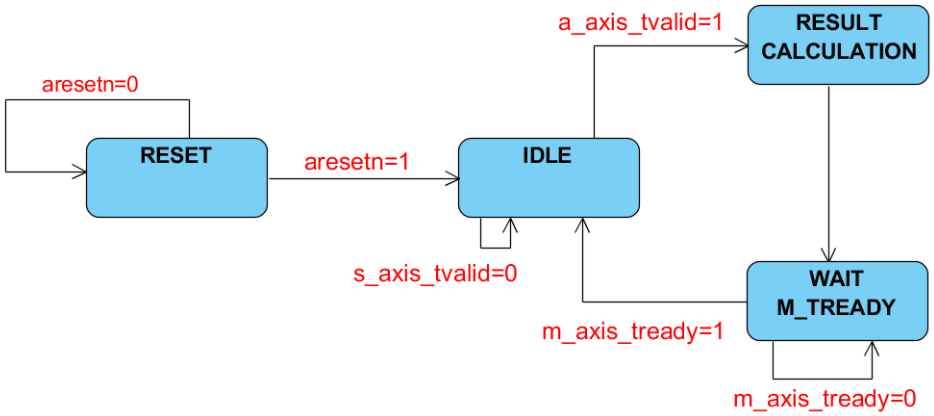


Figura 1.13: Macchina a stati

1.5.2 Divisore

Il componente preposto all'operazione di divisione nella catena di elaborazione si trova nel secondo stadio della pipe ed effettua il rapporto tra il modulo quadro di Sum_Early e Sum_Late. Per una possibile realizzazione del componente sono prese in considerazione 3 possibili soluzioni mostrate di seguito. Da sottolineare come l'IP core Divider generator offre la possibilità tramite il settaggio del parametro *Clocks per division* di ottenere una macchina diversa in termini di area e prestazioni. In via del tutto sperimentale, viene settato tale parametro con 3 valori differenti e ricavati i valori suddetti.

Divider generator	LUT	Slice Register	DSP48	F.max	Cicli di clock
Clocks per division = 1	7168	17165	0	163.514 MHz	91
Clocks per division = 2	7089	9144	0	102.776 MHz	91
Clocks per division = 8	1819	2635	0	104.493 MHz	91

Tabella 1.3: Varianti Divider generator

La scelta tra i tre ricade sulla terza opzione in quanto, al costo di una maggiore latenza, si ottiene un componente ottimizzato nell'occupazione d'area.

Infine, viene confrontato il suddetto IP core con due soluzioni alternative totalmente custom, come mostrato in tab.1.4.

Realizzazione	LUT	Slice Register	DSP48	F.max	Cicli di clock
IP core Divider generator	1819	2635	0	104.493	91
Divisore Non Restoring fixed	5996	428	0	83,198 MHz	104
Descrizione dataflow con operatori /	6850	171	0	2,407 MHz	2

Tabella 1.4: Occupazione d'area divisori

Analizzando la tabella si evince come il Divider generator risulta essere la scelta migliore rispetto alle altre in termini di area e frequenza di lavoro, pur pagandone il prezzo rispetto alle altre nel numero di slice register occupate.

1.5.3 Radice quadrata

Infine, per l'ultimo componente del Task vengono confrontati 3 realizzazioni diverse dell'operatore di radice quadrata. In particolare si confronta l'IP core Xilinx che sfrutta l'algoritmo Cordic con 2 soluzioni che implementano l'algoritmo digit-by-digit per il calcolo del valore di radice.

Realizzazione	LUT	Slice Register	DSP48	F.max	Cicli di clock
Cordic	741	403	0	130.056 MHz	24
Digit-by-digit combinatorio	1917	74	0	13.818 MHz	2
Digit-by-digit sequenziale	202	121	0	120.642 MHz	25

Tabella 1.5: Occupazione d'area radice quadrata

Con un buon compromesso tra area occupata e prestazioni, la scelta migliore ricade sul componente che calcola in maniera in maniera sequenziale il risultato sfruttando l'algoritmo digit-by-digit (per ulteriori informazioni si rimanda a `https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Digit-by-digit_calculation`)

1.5.3.1 Implementazione come FSM dell' Algoritmo digit-by-digit

Come da titolo, il componente che realizza la radice quadrata è implementato secondo la logica di una macchina a stati finiti. Per renderlo compatibile con bus AXI Stream ed altri componenti che vi si interfacciano con esso, vengono aggiunti gli stessi segnali di input/output come descritto in cap.1.5.1.1

Di seguito, in fig.1.14, si propone il diagramma che descrive la FSM.

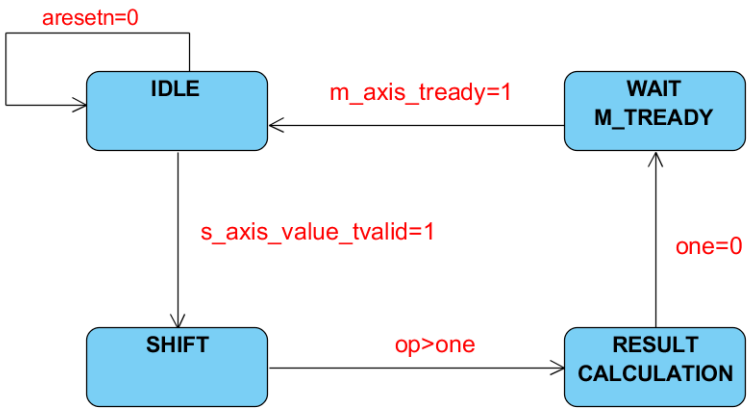


Figura 1.14: Macchina a stati digit-by-digit

A partire dallo stato inziale di **IDLE**, la macchina vi permane fintantoché il segnale di *tvalid* sull'interfaccia slave è pari a 0, oppure viene asserito il segnale di reset. Una volta che il *tvalid* diviene pari a 1, la macchina transita nello stato di **SHIFT** dove viene effettuata la seguente operazione: a partire da un valore che indica la precisione del calcolo, *one*, su di esso viene effettuato uno shift a destra di 2 posizioni fino a quando non si ottiene la più grande potenza di 4 più piccola del radicando. Una volta trovato tale valore, la macchina passa nello stato di **RESULT_CALCULATION** in cui viene effettuato il calcolo della radice secondo la logica

dell'algoritmo. Una volta determinata la radice (il segnale *one* è pari a 0), si passa nello stato di **WAIT_M_TREADY** in cui in uscita viene settato ad 1 il valore di *tvalid* e si attende che il componente a valle asserisca il segnale di *tready*.

1.6 Task 4 v2.0

Tale soluzione deriva direttamente dallo studio effettuato sui singoli moduli del Task 4, presentato accuratamente nella sezione precedente. Inoltre, si ricorda, che il segnale di uscita r è rappresentato come signed su 24 bit, di cui 13 parte intera, 11 decimale.

1.6.1 Design

La soluzione prevede di istanziare i seguenti IP core :

- 2 **Absolute Square**, che utilizzano gli operatori + e * inferiti su DSP;
- 1 **Divider Generator**, per realizzare l'operatore di divisione;
- 1 **Square Root**, che, facendo uso dell'algoritmo digit-by-digit, calcola la radice quadrata.

La fig. 1.15 mostra le istanze di tali componenti e relativi collegamenti tra essi.

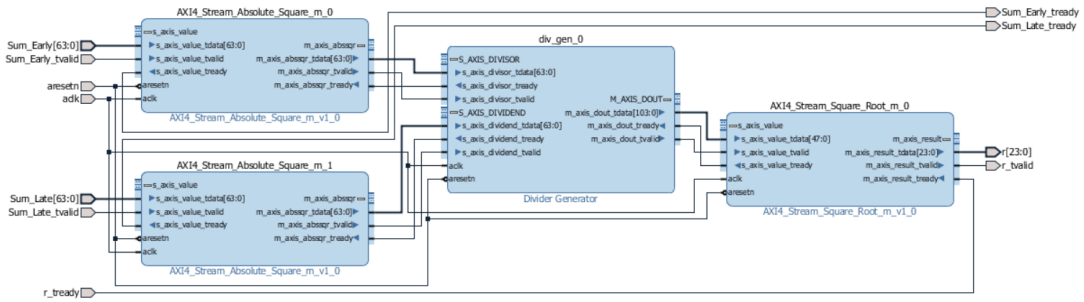


Figura 1.15: Block Diagram relativo ad una seconda soluzione

1.6.2 Codice

Di seguito si riportano unicamente i codici VHDL scritti manualmente.

1.6.2.1 Top Module

```
1  ----------------------------------------------------------------------------
2  --! @file      Task4_v2_0/src/Task4_m.vhd
3  --! @authors
4  --!              Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5  --!              Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
```

```
6  --!            Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7  --! @version  V2.0
8  --! @date     17-July-2017
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  Task4 is free software: you can redistribute it and/or modify
18 --!  it under the terms of the GNU Affero General Public License as
       published by
19 --!  the Free Software Foundation, either version 3 of the License, or
20 --!  (at your option) any later version.
21 --!
22 --!  Task4 is distributed in the hope that it will be useful,
23 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
24 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
25 --!  GNU Affero General Public License for more details.
26 --!
27 --!  You should have received a copy of the GNU Affero General Public
       License
28 --!  along with Linux Driver: Examples. If not, see
29 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
30 --! @brief Questo componente include tutte le funzionalità che deve eseguire
        il
31 --!        Task 4. In particolare, in ingresso al componente vengono forniti
32 --!        2 segnali complessi espressi su 64 bit (32 Im, 32 Re); in uscita,
33 --!        invece, è reso disponibile un segnale contenente la radice del
34 --!        rapporto del modulo quadro dei due segnali di ingresso. Tale
       segnale
35 --!    è espresso su 24 bit, di cui 13 sono la parte intera, 11 quella
36 --!    decimale. Tutto il componente è realizzato in modo tale da essere
37 --!        compatibile con interfaccia AXI4 Stream.
38 ----------------------------------------------------------------------------
39 library IEEE;
40 use IEEE.STD_LOGIC_1164.ALL;
41
42 entity Task4_m is
43     Port ( aclk : in STD_LOGIC;                                          --!
           Segnale di temporizzazione
44            aresetn : in STD_LOGIC;                                       --!
                 Reset sincrono, attivo basso
45            -- Interfaccia Slave del componente
```

```
46            s_axis_sum_early_tvalid :  in STD_LOGIC;                        --!
                 Se alto, il dato sum_early è valido
47            s_axis_sum_early_tready :  out STD_LOGIC;                       --!
                 Se alto il componente è pronto a ricevere sum_early
48            s_axis_sum_early_tdata : in STD_LOGIC_VECTOR (63 downto 0);     --!
                 Segnale di input rappresentante sum_early
49            s_axis_sum_late_tvalid :  in STD_LOGIC;                         --!
                 Se alto, il dato sum_late è valido
50            s_axis_sum_late_tready :  out STD_LOGIC;                        --!
                 Se alto il componente è pronto a ricevere sum_late
51            s_axis_sum_late_tdata : in STD_LOGIC_VECTOR (63 downto 0);      --!
                 Segnale di input rappresentante sum_late
52            -- Interfaccia Master del componente
53            m_axis_r_tvalid: out std_logic;                                 --!
                 Se alto, il dato r in output è valido
54            m_axis_r_tready: in std_logic;                                  --!
                 Se alto, il componente a valle è pronto a ricevere il dato r
55            m_axis_r_tdata : out STD_LOGIC_VECTOR (23 downto 0)             --!
                 Segnale di output rappresentante r
56            );
57  end Task4_m;
58
59  architecture Structural of Task4_m is
60  ------------------------------------------------------------------------
61  ----------------------Absolute Square Component-------------------------
62  ------------------------------------------------------------------------
63      component AXI4_Stream_Absolute_Square_m is
64          Port ( aresetn : in STD_LOGIC;
65                 aclk : in STD_LOGIC;
66                 -- Interfaccia Slave del componente
67                 s_axis_value_tdata : in STD_LOGIC_VECTOR (63 downto 0);
68                 s_axis_value_tvalid : in STD_LOGIC;
69                 s_axis_value_tready : out STD_LOGIC;
70                 -- Interfaccia Master del componente
71                 m_axis_abssqr_tdata : out STD_LOGIC_VECTOR (63 downto 0);
72                 m_axis_abssqr_tvalid : out STD_LOGIC;
73                 m_axis_abssqr_tready : in STD_LOGIC);
74      end component;
75
76  ------------------------------------------------------------------------
77  --------------------------Divisor Component-----------------------------
78  ------------------------------------------------------------------------
79     component AXI4_Stream_Divider_m IS
80        PORT (
81          aclk : IN STD_LOGIC;
82          aresetn : IN STD_LOGIC;
83          -- Interfaccia Slave del componente
84          s_axis_divisor_tvalid : IN STD_LOGIC;
85          s_axis_divisor_tready : OUT STD_LOGIC;
```

```
86          s_axis_divisor_tdata : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
87          s_axis_dividend_tvalid : IN STD_LOGIC;
88          s_axis_dividend_tready : OUT STD_LOGIC;
89          s_axis_dividend_tdata : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
90          -- Interfaccia Master del componente
91          m_axis_dout_tvalid : OUT STD_LOGIC;
92          m_axis_dout_tready : IN STD_LOGIC;
93          m_axis_dout_tdata : OUT STD_LOGIC_VECTOR(103 DOWNTO 0)
94        );
95      END component;
96
97  ------------------------------------------------------------------------
98  -------------------------Square Root Component--------------------------
99  ------------------------------------------------------------------------
100     component AXI4_Stream_Square_Root_m is
101         Port ( aclk    : in   STD_LOGIC;
102                aresetn : in   STD_LOGIC;
103                -- Slave signal interface
104                s_axis_value_tvalid  : in   STD_LOGIC;
105                s_axis_value_tready   : out  STD_LOGIC;
106                s_axis_value_tdata  : in   STD_LOGIC_VECTOR (47 downto 0);
107                -- Master signal interface
108                m_axis_result_tvalid : out STD_LOGIC;
109                m_axis_result_tready : in STD_LOGIC;
110                m_axis_result_tdata : out  STD_LOGIC_VECTOR (23 downto 0));
111     end component;
112
113 --! Segnali ausiliari per i due componenti che realizzano il modulo quadro
        di Sum_Early e Sum_Late
114 signal late2buffer: std_logic_vector(63 downto 0);
115 signal late2tvalid : std_logic;
116 signal late2tready : std_logic;
117 signal early2buffer: std_logic_vector(63 downto 0);
118 signal early2tvalid : std_logic;
119 signal early2tready : std_logic;
120
121 --! Segnali ausiliari per il componente che realizza la divisione tra il
        modulo di Sum_Early e il modulo di Sum_Late
122 signal quozient_tdata : std_logic_vector(103 downto 0);
123 signal quozient_tvalid : std_logic;
124 signal quozient_tready : std_logic;
125
126 --! Segnale ausiliare per gestire il dato in uscita da rappresentare su 24
        bit,
127 --! di cui 13 costituiscono la parte intera e 11 quella decimale.
128 signal root_value : std_logic_vector (23 downto 0);
129
130 begin
131
```

```
132 ABS_SQR_EARLY:  AXI4_Stream_Absolute_Square_m
133         PORT MAP (
134                 aresetn => aresetn,
135                 aclk => aclk,
136                 s_axis_value_tdata => s_axis_sum_early_tdata,
137                 s_axis_value_tvalid => s_axis_sum_early_tvalid,
138                 s_axis_value_tready => s_axis_sum_early_tready,
139                 m_axis_abssqr_tdata => early2buffer,
140                 m_axis_abssqr_tvalid => early2tvalid,
141                 m_axis_abssqr_tready => early2tready);
142
143 ABS_SQR_LATE:  AXI4_Stream_Absolute_Square_m
144        PORT MAP (
145                aresetn => aresetn,
146                aclk => aclk,
147                s_axis_value_tdata => s_axis_sum_late_tdata,
148                s_axis_value_tvalid => s_axis_sum_late_tvalid,
149                s_axis_value_tready => s_axis_sum_late_tready,
150                m_axis_abssqr_tdata => late2buffer,
151                m_axis_abssqr_tvalid => late2tvalid,
152                m_axis_abssqr_tready => late2tready);
153
154 DIVIDER: AXI4_Stream_Divider_m
155           PORT MAP(
156                  aclk => aclk,
157                  aresetn => aresetn,
158                  s_axis_divisor_tvalid => late2tvalid,
159                  s_axis_divisor_tready => late2tready,
160                  s_axis_divisor_tdata => late2buffer,
161                  s_axis_dividend_tvalid => early2tvalid,
162                  s_axis_dividend_tready => early2tready,
163                  s_axis_dividend_tdata => early2buffer,
164                  m_axis_dout_tvalid => quozient_tvalid,
165                  m_axis_dout_tready => quozient_tready,
166                  m_axis_dout_tdata => quozient_tdata);
167
168 SQUARE_ROOT: AXI4_Stream_Square_Root_m
169         PORT MAP (
170                aclk => aclk,
171                aresetn => aresetn,
172                s_axis_value_tvalid => quozient_tvalid,
173                s_axis_value_tready => quozient_tready,
174                s_axis_value_tdata => quozient_tdata(47 downto 0),
175                m_axis_result_tvalid => m_axis_r_tvalid,
176                m_axis_result_tready => m_axis_r_tready,
177                m_axis_result_tdata => root_value);
178
179 m_axis_r_tdata <= "000000000" & root_value(23 downto 9);
180
```

```
181 end Structural;
```

Codice 1.5: "Task4v2.0"

1.6.2.2 AXI4 Stream Absolute Square

```
1  ----------------------------------------------------------------------------
2  --! @file     Task4_v2_0/src/AXI4_Stream_Absolute_Square_m.vhd
3  --! @authors
4  --!             Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5  --!             Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
6  --!             Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7  --! @version  V2.0
8  --! @date     17-July-2017
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  This file is part of Task4.
18 --!
19 --!  Task4 is free software: you can redistribute it and/or modify
20 --!  it under the terms of the GNU Affero General Public License as
       published by
21 --!  the Free Software Foundation, either version 3 of the License, or
22 --!  (at your option) any later version.
23 --!
24 --!  Task4 is distributed in the hope that it will be useful,
25 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
26 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
27 --!  GNU Affero General Public License for more details.
28 --!
29 --!  You should have received a copy of the GNU Affero General Public
       License
30 --!  along with Linux Driver: Examples. If not, see
31 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
32 --! @brief Il componente riceve in ingresso un segnale complesso, avente
       parte reale
33 --!        e parte immaginaria su 32 bit, e ne calcola il modulo quadro.
       Esso,
```

```
34 --!        attraverso l'utilizzo dei segnali tvalid, tready  e tdata, è
       compatibile
35 --!        con interfaccia AXI4 Stream.
36 ----------------------------------------------------------------------------
37 library IEEE;
38 use IEEE.STD_LOGIC_1164.ALL;
39
40 entity AXI4_Stream_Absolute_Square_m is
41     Port ( aresetn : in STD_LOGIC;                                    --!
           Reset del componente, sincrono e attivo basso
42            aclk : in STD_LOGIC;                                       --!
               Segnale di temporizzazione
43            -- Interfaccia AXI4 Stream Slave
44            s_axis_value_tdata : in STD_LOGIC_VECTOR (63 downto 0);  --! Dato
                in ingresso espresso su 64 bit (32 Im, 32 Re)
45            s_axis_value_tvalid : in STD_LOGIC;                       --! Se
               alto, il dato in input è valido
46            s_axis_value_tready : out STD_LOGIC;                      --! Se
               alto, si è pronti ad accettare il tdata in input
47            -- Interfaccia AXI4 Stream Master
48            m_axis_abssqr_tdata : out STD_LOGIC_VECTOR (63 downto 0);--!
               Valore del modulo quadro calcolato
49            m_axis_abssqr_tvalid : out STD_LOGIC;                     --! Se
               alto, il segnale tdata in output è valido
50            m_axis_abssqr_tready : in STD_LOGIC);                     --! Se
               alto il componente a valle è pronto ad accettare il tdata in
               output
51 end AXI4_Stream_Absolute_Square_m;
52
53 architecture Structural of AXI4_Stream_Absolute_Square_m is
54
55 ----------------------------------------------------------------------------
56 -----------------------Control Unit Component-----------------------------
57 ----------------------------------------------------------------------------
58     component abs_sqr_control_unit_m is
59     Port ( aclk : in STD_LOGIC;
60            aresetn : in STD_LOGIC;
61            s_axis_value_tvalid : in STD_LOGIC;
62            m_axis_abssqr_tready : in STD_LOGIC;
63            enable_buffer_in : out STD_LOGIC;
64            enable_buffer_out : out STD_LOGIC;
65            reset_buffer_in_n : out STD_LOGIC;
66            reset_buffer_out_n : out STD_LOGIC;
67            s_axis_value_tready : out STD_LOGIC;
68            m_axis_abssqr_tvalid : out STD_LOGIC);
69     end component;
70
71 ----------------------------------------------------------------------------
72 ---------------------Operative Part Component-----------------------------
```

```
73  ----------------------------------------------------------------------------
74      component abs_sqr_operative_part_m is
75          Port ( s_axis_value_tdata : in STD_LOGIC_VECTOR (63 downto 0);
76                 aresetn : in STD_LOGIC;
77                 aclk : in STD_LOGIC;
78                 enable_buffer_in : in STD_LOGIC;
79                 enable_buffer_out : in STD_LOGIC;
80                 reset_buffer_in_n : in STD_LOGIC;
81                 reset_buffer_out_n : in STD_LOGIC;
82                 m_axis_abssqr_tdata : out STD_LOGIC_VECTOR(63 downto 0));
83      end component;
84
85
86      --! Segnali ausiliari per collegare la Parte Operativa alla parte di
            controllo
87      signal enable_buffer_in : std_logic :='0';
88      signal enable_buffer_out : std_logic :='0';
89      signal reset_buffer_in_n : STD_LOGIC:='1';
90      signal reset_buffer_out_n : STD_LOGIC:='1';
91
92  begin
93
94      CONTROL_UNIT_INST : abs_sqr_control_unit_m port map(
95          aclk => aclk,
96          aresetn => aresetn,
97          s_axis_value_tvalid => s_axis_value_tvalid,
98          m_axis_abssqr_tready => m_axis_abssqr_tready,
99          enable_buffer_in => enable_buffer_in,
100         enable_buffer_out => enable_buffer_out,
101         s_axis_value_tready => s_axis_value_tready,
102         reset_buffer_out_n => reset_buffer_out_n,
103         reset_buffer_in_n => reset_buffer_in_n,
104         m_axis_abssqr_tvalid => m_axis_abssqr_tvalid);
105
106     OPERATIVE_UNIT_INST : abs_sqr_operative_part_m port map(
107         s_axis_value_tdata => s_axis_value_tdata,
108         aresetn => aresetn,
109         aclk => aclk,
110         enable_buffer_in => enable_buffer_in,
111         enable_buffer_out => enable_buffer_out,
112         reset_buffer_out_n => reset_buffer_out_n,
113         reset_buffer_in_n => reset_buffer_in_n,
114         m_axis_abssqr_tdata => m_axis_abssqr_tdata);
115
116 end Structural;
```

Codice 1.6: "AXI4 Stream Absolute Square"

Control Part

```
1  ----------------------------------------------------------------------------
2  --! @file     Task4_v2_0/src/abs_sqr_control_unit_m.vhd
3  --! @authors
4  --!             Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5  --!             Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
6  --!             Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7  --! @version  V2.0
8  --! @date     17-July-2017
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  Task4 is free software: you can redistribute it and/or modify
18 --!  it under the terms of the GNU Affero General Public License as
       published by
19 --!  the Free Software Foundation, either version 3 of the License, or
20 --!  (at your option) any later version.
21 --!
22 --!  This file is part of Task4.
23 --!
24 --!  Task4 is distributed in the hope that it will be useful,
25 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
26 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
27 --!  GNU Affero General Public License for more details.
28 --!
29 --!  You should have received a copy of the GNU Affero General Public
       License
30 --!  along with Linux Driver: Examples. If not, see
31 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
32 --! @brief Questo componente realizza una FSM per il controllo del
       componente
33 --!        che si occupa di calcolare il modulo quadro di un numero. Tale
       macchina
34 --!        a stati si è resa necessaria, per interfacciare il componente con
        AXI4 Stream.
35 ----------------------------------------------------------------------------
36 library IEEE;
37 use IEEE.STD_LOGIC_1164.ALL;
38
39 entity abs_sqr_control_unit_m is
```

```
40      Port ( aclk : in STD_LOGIC;                          --! Segnale di
            temporizzazione della FSM
41              aresetn : in STD_LOGIC;                      --! Reset della FSM,
                sincrono e attivo bass
42              s_axis_value_tvalid : in STD_LOGIC;          --! Se alto, vuol dire
                che il dato di input all'interfaccia slave è valido
43              m_axis_abssqr_tready : in STD_LOGIC;         --! Se alto, vuol dire
                che il componente che si trova a valle è pronto a ricevere il
                 dato di output
44              enable_buffer_in : out STD_LOGIC;            --! Se alto, abilita a
                scrivere nel buffer di input della macchina
45              enable_buffer_out : out STD_LOGIC;           --! Se alto, abilita a
                scrivere nel buffer di output della macchina
46              reset_buffer_in_n : out STD_LOGIC;           --! Reset sincrono per
                il buffer di input della macchina
47              reset_buffer_out_n : out STD_LOGIC;          --! Reset sincrono per
                il buffer di output della macchina
48              s_axis_value_tready : out STD_LOGIC;         --! Se alto, vuol dire
                che l'interfaccia slave è pronta a ricevere un dato
49              m_axis_abssqr_tvalid : out STD_LOGIC);       --! Se alto, vuol dire
                che il dato in uscita dall'interfaccia master è valido
50  end abs_sqr_control_unit_m;
51
52  architecture Behavioral of abs_sqr_control_unit_m is
53
54      type state_type is(RESET,IDLE,RESULT_CALCULATION,WAIT_M_TREADY);
55      signal current_state, next_state : state_type := RESET;
56
57  begin
58
59  ------------------------------------------------------------------------
60  ----------------------------Sync Process--------------------------------
61  ------------------------------------------------------------------------
62  --! Questo process realizza la sincronizzazione degli stati con il segnale
63  --! di temporizzazione. Ogni volta che si verifica un rising_edge la FSM
64  --! entra nello stato definito in next_state
65      SYNC_PROCESS : process(aclk)
66          begin
67              if(rising_edge(aclk))then
68                  if(aresetn='0')then                      --! Se il segnale di
                    reset è attivo, la FSM entra nello stato di RESET
69                      current_state <= RESET;
70                  else
71                      current_state <= next_state;         --! altrimenti entra
                        nello stato definito in next_state
72                  end if;
73              end if;
74      end process SYNC_PROCESS;
75
```

```
76  ------------------------------------------------------------------------
77  ----------------------Output Decode Process-----------------------------
78  ------------------------------------------------------------------------
79  --! In questo process vengono definiti i valori dei segnali di output
80  --! della FSM
81      OUTPUT_DECODE : process(current_state)
82          begin
83              case(current_state) is
84                  when RESET =>
85                      enable_buffer_in <= '0';
86                      enable_buffer_out <= '0';
87                      s_axis_value_tready <= '0';
88                      m_axis_abssqr_tvalid <= '0';
89                      reset_buffer_in_n <= '1';
90                      reset_buffer_out_n <= '1';
91                  when IDLE =>
92                       enable_buffer_in <= '1';
93                       enable_buffer_out <= '0';
94                       s_axis_value_tready <= '1';
95                       m_axis_abssqr_tvalid <= '0';
96                       reset_buffer_in_n <= '1';
97                       reset_buffer_out_n <= '0';
98                   when RESULT_CALCULATION =>
99                       enable_buffer_in <= '0';
100                      enable_buffer_out <= '1';
101                      s_axis_value_tready <= '0';
102                      m_axis_abssqr_tvalid <= '0';
103                      reset_buffer_in_n <= '1';
104                      reset_buffer_out_n <= '1';
105                  when WAIT_M_TREADY =>
106                      enable_buffer_in <= '0';
107                      enable_buffer_out <= '0';
108                      s_axis_value_tready <= '0';
109                      m_axis_abssqr_tvalid <= '1';
110                      reset_buffer_in_n <= '0';
111                      reset_buffer_out_n <= '1';
112             end case;
113     end process OUTPUT_DECODE;
114
115
116 ------------------------------------------------------------------------
117 -------------------------Next State Process-----------------------------
118 ------------------------------------------------------------------------
119 --! In questo process viene definito chi deve essere lo stato successivo
120 --! in cui la FSM deve entrare, in base ai segnali di input
121     NEXT_STATE_DECODE : process(current_state,s_axis_value_tvalid,
        m_axis_abssqr_tready,aresetn)
122          begin
123              case(current_state)is
```

```
124                 when RESET =>
125                     if(aresetn='0')then
126                         next_state <= RESET;
127                     else
128                         next_state <= IDLE;
129                     end if;
130                 when IDLE =>
131                     if(s_axis_value_tvalid='0')then
132                         next_state <= IDLE;
133                     else
134                         next_state <= RESULT_CALCULATION;
135                     end if;
136                 when RESULT_CALCULATION =>
137                     next_state <= WAIT_M_TREADY;
138                 when WAIT_M_TREADY =>
139                     if(m_axis_abssqr_tready='0')then
140                         next_state <= WAIT_M_TREADY;
141                      else
142                         next_state <= IDLE;
143                     end if;
144             end case;
145     end process;
146 
147 end Behavioral;
```

Codice 1.7: "Control Part"

Operative Part

```
1  ----------------------------------------------------------------------------
2  --! @file     Task4_v2_0/src/abs_sqr_operative_part_m.vhd
3  --! @authors
4  --!             Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5  --!             Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
6  --!             Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7  --! @version  V2.0
8  --! @date     17-July-2017
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
```

```
17 --!  This file is part of Task4.
18 --!
19 --!  Task4 is free software: you can redistribute it and/or modify
20 --!  it under the terms of the GNU Affero General Public License as
    published by
21 --!  the Free Software Foundation, either version 3 of the License, or
22 --!  (at your option) any later version.
23 --!
24 --!  Task4 is distributed in the hope that it will be useful,
25 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
26 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
27 --!  GNU Affero General Public License for more details.
28 --!
29 --!  You should have received a copy of the GNU Affero General Public
    License
30 --!  along with Linux Driver: Examples. If not, see
31 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
32 --! @brief Questo componente ingloba in sè tutta la parte operativa del
    componente
33 --!        AXI4_Stream_Absolute_Square_m. Al suo interno sono presenti 2
    registri,
34 --!        che funzionano da buffer di ingresso e uscita; la logica
    comportamentale
35 --!        che realizza l'operazione di modulo quadro.
36 ----------------------------------------------------------------------------
37
38 library IEEE;
39 use IEEE.STD_LOGIC_1164.ALL;
40
41 entity abs_sqr_operative_part_m is
42     Port ( s_axis_value_tdata : in STD_LOGIC_VECTOR (63 downto 0);      --!
        Valore di ingresso di cui si vuole calcolare il modulo quadro
43            aresetn : in STD_LOGIC;                                      --!
                Reset sincrono esterno, attivo basso
44            aclk : in STD_LOGIC;                                         --!
                Segnale di temporizzazione della macchina
45            enable_buffer_in : in STD_LOGIC;                             --!
                Segnale di enable, che abilita il funzionamento del buffer di
                 ingresso
46            enable_buffer_out : in STD_LOGIC;                            --!
                Segnale di enable, che abilita il funzionamento del buffer di
                 uscita
47            reset_buffer_in_n : in STD_LOGIC;                            --!
                Reset al buffer di ingresso, sincrono, proveniente dalla FSM,
                 attivo basso
48            reset_buffer_out_n : in STD_LOGIC;                           --!
                Reset al buffer di uscita, sincrono, proveniente dalla FSM,
                attivo basso
49            m_axis_abssqr_tdata : out STD_LOGIC_VECTOR(63 downto 0));    --!
```

```
                Valore di uscita rappresentante il modulo quadro calcolato
end abs_sqr_operative_part_m;

architecture Structural of abs_sqr_operative_part_m is

------------------------------------------------------------------------
----------------------Modulo Quadro Component---------------------------
------------------------------------------------------------------------
    component mod_quad_oper is
    generic(n: natural:=32);
    Port ( Im : in STD_LOGIC_VECTOR (n-1 downto 0);
           Re : in STD_LOGIC_VECTOR (n-1 downto 0);
           modulo2 : out STD_LOGIC_VECTOR (2*n -1 downto 0));
    end component;

------------------------------------------------------------------------
-------------------------Register Component-----------------------------
------------------------------------------------------------------------
    component register_m is
        generic(N : natural:=64);
        Port ( data_in : in STD_LOGIC_VECTOR (N-1 downto 0);
               enable : in STD_LOGIC;
               reset_n : in STD_LOGIC;
               data_out : out STD_LOGIC_VECTOR (N-1 downto 0);
               clock : in STD_LOGIC);
    end component;

--! Segnali ausiliari per la gestione del circuito e i corretti collegamenti
--! tra i vari componenti
    signal real_part : std_logic_vector(31 downto 0):=(others=>'0');
    signal imag_part : std_logic_vector(31 downto 0):=(others=>'0');
    signal result : std_logic_vector(63 downto 0):=(others=>'0');
    signal operand : std_logic_vector(63 downto 0):=(others=>'0');
    signal reset_in_n : std_logic := '1';
    signal reset_out_n : std_logic := '1';

begin
    reset_in_n <= aresetn and reset_buffer_in_n;
    BUFFER_IN_INST : register_m port map(
        data_in => s_axis_value_tdata,
        enable => enable_buffer_in,
        reset_n => reset_in_n,
        data_out => operand,
        clock => aclk);

    imag_part <= operand(63 downto 32);
    real_part <= operand(31 downto 0);
    SQUARE_INST :mod_quad_oper port map(
        Im => imag_part,
```

```
98          Re => real_part,
99          modulo2 => result);
100
101     reset_out_n <= aresetn and reset_buffer_out_n;
102     BUFFER_OUT_INST : register_m port map(
103          data_in => result,
104          enable => enable_buffer_out,
105          reset_n => reset_out_n,
106          data_out => m_axis_abssqr_tdata,
107          clock => aclk);
108
109 end Structural;
```

Codice 1.8: "Operative Part"

Modulo Quadro

```
1  --! @file     Task4_v2_0/src/mod_quad_oper.vhd
2  --! @authors
3  --!              Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
4  --!              Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
5  --!              Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
6  --! @version  V2.0
7  --! @date     17-July-2017
8  --! @copyright
9  --!  Copyright (C) 2017
10 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
11 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
12 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
13 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
14 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
15 --!
16 --!  Task4 is free software: you can redistribute it and/or modify
17 --!  it under the terms of the GNU Affero General Public License as
       published by
18 --!  the Free Software Foundation, either version 3 of the License, or
19 --!  (at your option) any later version.
20 --!
21 --!  This file is part of Task4.
22 --!
23 --!  Task4 is distributed in the hope that it will be useful,
24 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
25 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
26 --!  GNU Affero General Public License for more details.
27 --!
```

```
28 --!  You should have received a copy of the GNU Affero General Public
       License
29 --!  along with Linux Driver: Examples. If not, see
30 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
31 --! @brief Questo componente realizza il modulo quadro di un numero
       compmlesso.
32 --!        Si presuppone che sia la parte reale sia quella immaginaria del
       numero
33 --!        siano rappresentate entrambe sullo stesso numero di bit, in
       notazione
34 --!        signed integer. Si precisa, a tal proposito, che ai fini del
       progetto,
35 --!        essendo il modulo un numero positivo e nell'ottica del risparmio
       di spazio,
36 --!        questo componente tronca l'ultimo bit di uscita, che si sa
       apriori
37 --!        essere nullo.
38 --!        Per la sua realizzazione, è stata utilizzata una filosofia di
       progetto
39 --!        dataflow, in questo modo il sintetizzatore UG901 di Vivado cerca
       di
40 --!        inferire, lì dove possibile, le DSP48E presenti sulla Zynq Zybo.
41 ----------------------------------------------------------------------------
42 library IEEE;
43 use IEEE.STD_LOGIC_1164.ALL;
44 use IEEE.NUMERIC_STD.ALL;
45
46 entity mod_quad_oper is
47     generic(n: natural:=32);                                        --!
           Specifica la dimensione, n, su cui deve essere rappresentata la
           parte reale e immaginaria del numero
48     Port ( Im : in STD_LOGIC_VECTOR (n-1 downto 0);                 --!
           Specifica la parte immaginaria del numero, espressa su n bit
49            Re : in STD_LOGIC_VECTOR (n-1 downto 0);                 --!
                  Specifica la parte reale del numero, espressa su n bit
50            modulo2 : out STD_LOGIC_VECTOR (2*n -1 downto 0));       --!
                  Specifica il modulo quadro calcolato
51 end mod_quad_oper;
52
53 architecture Dataflow of mod_quad_oper is
54
55    --! Affinchè le stringhe di bit in ingresso vengano trattate come numeri
          signed,
56    --! vengono utilizzati i seguenti segnali ausiliari definiti signed.
57         signal data_im:  signed(n-1 downto 0);
58         signal data_re:  signed(n-1 downto 0);
59
60     --! Per garantire il corretto calcolo dell'operazione di addizione di
          numeri signed,
```

```
61      --! sono definiti i segnèali seguenti.
62          signal data_im2:  signed(2*n-1 downto 0);   --! Im^2
63          signal data_re2:  signed(2*n-1 downto 0);   --! Re^2
64
65      --! Il segnale data_mod, invecec, conterrà il risultato dell'addizione
           signed
66      --! tra data_im2 e data_re2
67      signal data_mod:  signed(2*n-1 downto 0);         --! Im^2 + Re^2
68
69  begin
70
71      data_im<=signed(Im);    --! Casting da std_logic_vector a signed
72      data_re<=signed(Re);
73
74      --! Calcolo del quadrato della parte immaginaria
75      data_im2<=data_im*data_im;                  --32 bit_signed * 32
           bit_signed= 64 bit_signed
76      --! Calcolo del quadrato della parte reale
77      data_re2<=data_re*data_re;                  --32 bit_signed * 32
           bit_signed= 64 bit_signed
78
79      --! Somma dei due quadrati per ottenere il modulo del numero.
80      data_mod<=data_im2+data_re2;                 --64 bit_signed + 64 bit
           _signed = 65 bit_signed = 64 bit unsigned (poichè il valore è senz'
           altro positivo)
81
82      --! Il valore calcolato viene portato in uscita a meno dell'ultimo bit
83      modulo2<=std_logic_vector(data_mod(2*n-1 downto 0));
84
85  end Dataflow;
```

Codice 1.9: "modulo quadro"

1.6.2.3 Square Root

```
1   --
       ----------------------------------------------------------------------
2   --! @file     Task4_v2_0/src/AXI4_Stream_Square_Root_m.vhd
3   --! @authors
4   --!             Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5   --!             Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
6   --!             Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7   --! @version  V2.0
8   --! @date     17-July-2017
```

```
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  This file is part of Task4.
18 --!
19 --!  Task4 is free software: you can redistribute it and/or modify
20 --!  it under the terms of the GNU Affero General Public License as
       published by
21 --!  the Free Software Foundation, either version 3 of the License, or
22 --!  (at your option) any later version.
23 --!
24 --!  Task4 is distributed in the hope that it will be useful,
25 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
26 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
27 --!  GNU Affero General Public License for more details.
28 --!
29 --!  You should have received a copy of the GNU Affero General Public
       License
30 --!  along with Linux Driver: Examples. If not, see
31 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
32 --! @brief Tale componente, attraverso una FSM, realizza il calcolo della
       radice
33 --!        quadrata di un numero.
34 --
       ----------------------------------------------------------------------------

35 library IEEE;
36 use IEEE.STD_LOGIC_1164.ALL;
37 use IEEE.NUMERIC_STD.ALL;
38
39 entity AXI4_Stream_Square_Root_m is
40     Port ( aclk    : in   STD_LOGIC;                                    --!
           Segnale di temporizzazione
41            aresetn : in   STD_LOGIC;                                    --!
                 Reset sincrono, attivo basso
42            -- Slave signal interface
43            s_axis_value_tvalid  : in   STD_LOGIC;                       --!
                 Se alto, il tdata in input è valido
44            s_axis_value_tready   : out  STD_LOGIC;                      --!
                 Se alto, il componente è pronto a ricevere il tdata
45            s_axis_value_tdata  : in   STD_LOGIC_VECTOR (47 downto 0);   --!
                 Valore di cui si vuole calcolare la radice
```

```
46
47          -- Master signal interface
48          m_axis_result_tvalid : out STD_LOGIC;                          --!
               Se alto, il tdata in output è valido
49          m_axis_result_tready : in STD_LOGIC;                           --!
               Se alto, il componente a valle è pronto a ricevere il dato
50          m_axis_result_tdata : out  STD_LOGIC_VECTOR (23 downto 0));  --!
               Valore della radice calcolato
51  end AXI4_Stream_Square_Root_m;
52
53  architecture Behavioral of AXI4_Stream_Square_Root_m is
54
55      --! Segnali ausiliari utili al funzionamento dell'algoritmo
56      signal op  : unsigned(47 downto 0); --! In questo segnale viene
           memorizzato inizialmente il valore di cui si vuole calcolare la
           radice
57      signal res : unsigned(47 downto 0); --! Qui viene memorizzato il
           risultato temporaneo dell'operazione
58      signal one : unsigned(47 downto 0); --! Valore iniziale di confronto
59
60      --! Deinizione degli stati della FSM
61      type state is (idle, shift, calc, wait_tready); --! Il tipo stato può
           assumere i valori idle, shift, calc, wait_tready
62      signal next_state : state;                        --! Creazione di un
           segnale next_state di tipo state
63
64  begin
65
66      ROOT_ALG : process
67
68           begin
69
70            wait until rising_edge(aclk);
71
72                case next_state is
73
74                   when idle =>
75                       if (aresetn = '0') then            --! Se il reset è
                           attivo
76                          s_axis_value_tready <= '0';     --! il tready deve
                              essere portato a zero
77                       else
78                           s_axis_value_tready <= '1';    --! altrimenti è
                              settato a 1
79                       end if;
80
81                       m_axis_result_tvalid <= '0';       --! Il tvalid del
                          master è posto a zero
82
```

```
83                  --! Inizializzazione segnali ausiliari all'algoritmo
84                  one <= x"400000000000";
85                  op  <= unsigned(s_axis_value_tdata);
86                  res <= (others=>'0');
87
88                  if (s_axis_value_tvalid='1') then   --! se il t valid
                        dello slave è pari ad 1 inizia un nuovo calcolo
89                     next_state <= shift;             --! e la FSM passa
                           nello stato di SHIFT
90                  end if;
91
92              when shift =>
93                  if (aresetn = '0') then             --! Se il reset è
                        attivo
94                     next_state <= idle;              --! si torna nello
                           stato di idle
95                  end if;
96
97                  s_axis_value_tready <= '0';
98
99                  if (one > op) then
100                    one <= one/4;
101                 else
102                    next_state   <= calc;
103                 end if;
104
105             when calc =>
106                 if (aresetn = '0') then             --! Se il reset è
                        attivo
107                     next_state <= idle;             --! si torna nello
                           stato di idle
108                  end if;
109
110                 if (one /= 0) then
111                    if (op >= res+one) then
112                       op   <= op - (res+one);
113                       res  <= res/2 + one;
114                    else
115                       res  <= res/2;
116                    end if;
117                    one <= one/4;
118                 else
119                    next_state  <= wait_tready;
120                 end if;
121
122             when wait_tready =>
123                 if (aresetn = '0') then             --! Se il reset è
                        attivo
```

```
124                         next_state <= idle;              --! si torna nello
                               stato di idle
125                     end if;
126
127                     m_axis_result_tvalid <= '1';
128
129                     --! Conserva il dato calcolato finchè non arriva un
                           tready sul master
130                     if (m_axis_result_tready='1') then
131                        next_state <= idle;
132                     end if;
133             end case;
134         end process;
135
136     --! Salvataggio del risultato sui segnali di uscita
137     m_axis_result_tdata <= std_logic_vector(res(m_axis_result_tdata'range));
138
139 end Behavioral;
```

Codice 1.10: "Square Root"

1.6.3 Testbench

Per verificare il funzionamento, viene eseguito un semplice testbench, fig. 1.16, fornendo in input al Task una coppia di dati generati in MATLAB. Si può notare come il risultato finale viene restituito in uscita dopo 144 cicli di clock, impostando idealmente un clock in ingresso a 100 Mhz. In questa implementazione, l'uscita r è espressa in una forma fixed point unsigned <24,11>.

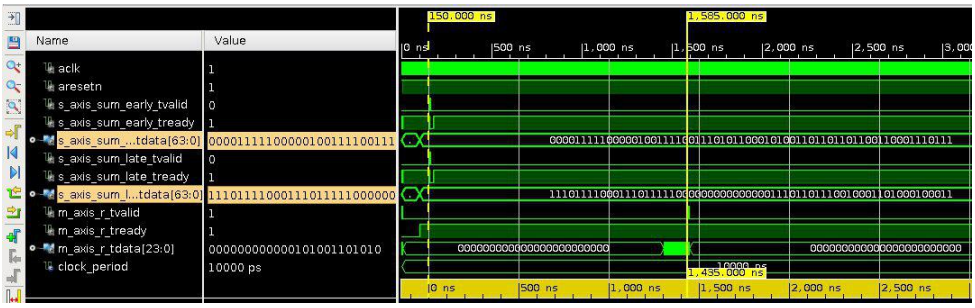


Figura 1.16: Testbench

1.6.4 Analisi soluzione

Per quanto riguarda l'area occupata dal componente complessivo, viene fornita la seguente tabella che riassume l'occupazione dovuta ai singoli IP core dispiegati, nonché quella relativa all'intero Task.

Componente	LUT	Slice Register	DSP48
Absolute Square	165	66	8
Divider generator	2036	4474	0
Square Root	233	124	0
Totale	2600(15%)	4730(13%)	16(20%)

Tabella 1.6: Occupazione d'area post-sintesi

Si può notare che, rispetto alla versione precedente, c'è un netto risparmio delle risorse disponibili sulla board.

Per quanto riguarda, invece, la frequenza di lavoro del circuito, utilizzando lo stesso metodo descritto per la versione precedente, fig. , si trova che, in questo caso, il componente può lavorare ad una frequenza di $80,901 Mhz$, dovuta al componente Absolute Square.

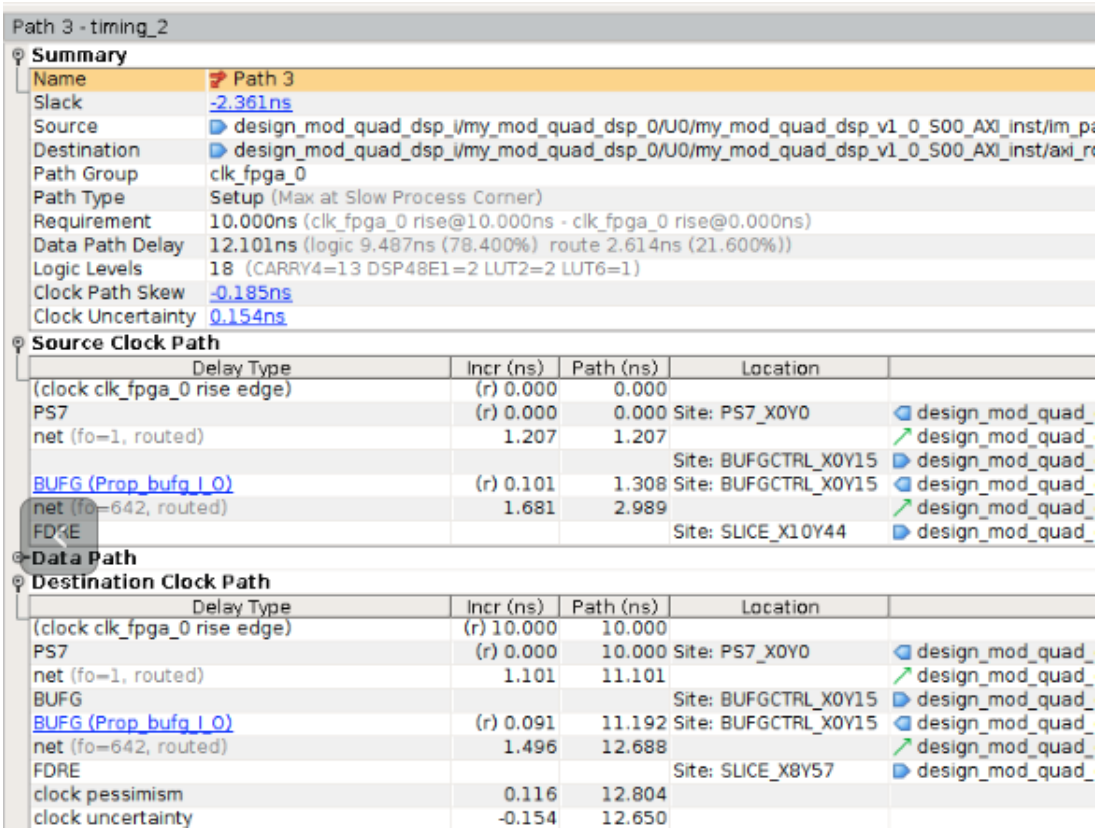


Figura 1.17: Most negative slack

1.6.5 Vantaggi e svantaggi

In questa versione è facile notare come vengono superate le criticità incontrate nella precedente soluzione. In particolare, viene migliorato il consumo di risorse disponibili sulla board, l'intero componente è compatibile con AXI4 Stream, il tutto al prezzo di calcolare il risultato finale in 144 cicli di clock, rispetto ai 137 della versione precedente.

Un ulteriore svantaggio riscontrato nella soluzione proposta riguarda l'adattamento del segnale di uscita. Far restituire al divisore un numero di cifre decimali pari a 40, risulta essere uno spreco

significativo di risorse, in quanto successivamente, dopo l'operazione di radice i 20 bit decimali vengono comunque troncati, per diventare 11. A questo punto, nella soluzione successiva, si pensa un modo per superare tale criticità.

1.7 Task 4 v2.1

Tale soluzione, come precisato sopra, si propone di ridurre sia il consumo di area sia il numero di cicli di clock necessari ad ottenere il risultato finale. In particolare, si modifica la dimensione della parte frazionaria in uscita al divisore. In questo modo, il divisore occupa meno area e impiega meno cicli di clock per calcolare il quoziente.

1.7.1 Design

La fig. 1.18 mostra il block design.

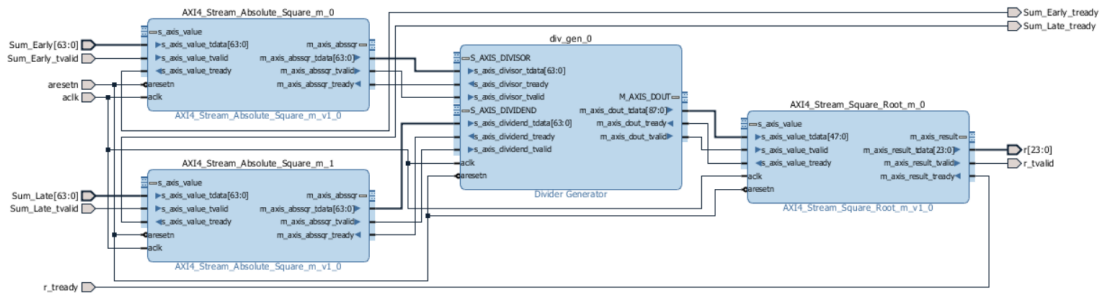


Figura 1.18: Block Diagram relativo ad una terza soluzione

1.7.2 Codice

Di seguito si riporta l'unico codice variato rispetto alla soluzione precedente.

```
1  ----------------------------------------------------------------------------
2  --! @file     Task4_v2_1/src/Task4_m.vhd
3  --! @authors
4  --!             Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5  --!             Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
6  --!             Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7  --! @version  V2.1
8  --! @date     18-July-2017
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
```

```
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  Task4 is free software: you can redistribute it and/or modify
18 --!  it under the terms of the GNU Affero General Public License as
       published by
19 --!  the Free Software Foundation, either version 3 of the License, or
20 --!  (at your option) any later version.
21 --!
22 --!  Task4 is distributed in the hope that it will be useful,
23 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
24 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
25 --!  GNU Affero General Public License for more details.
26 --!
27 --!  You should have received a copy of the GNU Affero General Public
       License
28 --!  along with Linux Driver: Examples. If not, see
29 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
30 --! @brief Questo componente include tutte le funzionalità che deve eseguire
        il
31 --!        Task 4. In particolare, in ingresso al componente vengono forniti
32 --!        2 segnali complessi espressi su 64 bit (32 Im, 32 Re); in uscita,
33 --!        invece, è reso disponibile un segnale contenente la radice del
34 --!        rapporto del modulo quadro dei due segnali di ingresso. Tale
       segnale
35 --!        è espresso su 24 bit, di cui 13 sono la parte intera, 11 quella
36 --!        decimale. Tutto il componente è realizzato in modo tale da essere
37 --!        compatibile con interfaccia AXI4 Stream.
38 ----------------------------------------------------------------------------
39 library IEEE;
40 use IEEE.STD_LOGIC_1164.ALL;
41
42 entity Task4_m is
43     Port ( aclk : in STD_LOGIC;                                          --!
           Segnale di temporizzazione
44            aresetn : in STD_LOGIC;                                       --!
                  Reset sincrono, attivo basso
45            -- Interfaccia Slave del componente
46            s_axis_sum_early_tvalid :  in STD_LOGIC;                      --!
                  Se alto, il dato sum_early è valido
47            s_axis_sum_early_tready :  out STD_LOGIC;                     --!
                  Se alto il componente è pronto a ricevere sum_early
48            s_axis_sum_early_tdata : in STD_LOGIC_VECTOR (63 downto 0);   --!
                  Segnale di input rappresentante sum_early
49            s_axis_sum_late_tvalid :  in STD_LOGIC;                       --!
                  Se alto, il dato sum_late è valido
50            s_axis_sum_late_tready :  out STD_LOGIC;                      --!
                  Se alto il componente è pronto a ricevere sum_late
51            s_axis_sum_late_tdata : in STD_LOGIC_VECTOR (63 downto 0);    --!
```

```
                Segnale di input rappresentante sum_late
52              -- Interfaccia Master del componente
53              m_axis_r_tvalid: out std_logic;                                 --!
                    Se alto, il dato r in output è valido
54              m_axis_r_tready: in std_logic;                                  --!
                    Se alto, il componente a valle è pronto a ricevere il dato r
55              m_axis_r_tdata : out STD_LOGIC_VECTOR (23 downto 0)             --!
                    Segnale di output rappresentante r
56              );
57  end Task4_m;
58
59  architecture Structural of Task4_m is
60  ------------------------------------------------------------------------
61  ----------------------Absolute Square Component-------------------------
62  ------------------------------------------------------------------------
63      component AXI4_Stream_Absolute_Square_m is
64          Port ( aresetn : in STD_LOGIC;
65                 aclk : in STD_LOGIC;
66                 -- Interfaccia Slave del componente
67                 s_axis_value_tdata : in STD_LOGIC_VECTOR (63 downto 0);
68                 s_axis_value_tvalid : in STD_LOGIC;
69                 s_axis_value_tready : out STD_LOGIC;
70                 -- Interfaccia Master del componente
71                 m_axis_abssqr_tdata : out STD_LOGIC_VECTOR (63 downto 0);
72                 m_axis_abssqr_tvalid : out STD_LOGIC;
73                 m_axis_abssqr_tready : in STD_LOGIC);
74      end component;
75
76  ------------------------------------------------------------------------
77  --------------------------Divisor Component-----------------------------
78  ------------------------------------------------------------------------
79      component AXI4_Stream_Divider_m IS
80        Port (
81          aclk : IN STD_LOGIC;
82          aresetn : IN STD_LOGIC;
83          s_axis_divisor_tvalid : IN STD_LOGIC;
84          s_axis_divisor_tready : OUT STD_LOGIC;
85          s_axis_divisor_tdata : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
86          s_axis_dividend_tvalid : IN STD_LOGIC;
87          s_axis_dividend_tready : OUT STD_LOGIC;
88          s_axis_dividend_tdata : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
89          m_axis_dout_tvalid : OUT STD_LOGIC;
90          m_axis_dout_tready : IN STD_LOGIC;
91          m_axis_dout_tdata : OUT STD_LOGIC_VECTOR(87 DOWNTO 0)    --! rispetto
                alla v2.0 viene richiesta una parte frazionaria di 22 bit, al
               posto di 40
92        );
93      end component;
94
```

```
95  ------------------------------------------------------------------------
96  -------------------------Square Root Component--------------------------
97  ------------------------------------------------------------------------
98      component AXI4_Stream_Square_Root_m is
99          Port ( aclk    : in   STD_LOGIC;
100                aresetn : in   STD_LOGIC;
101                s_axis_value_tvalid  : in   STD_LOGIC;
102                s_axis_value_tready   : out  STD_LOGIC;
103                s_axis_value_tdata  : in   STD_LOGIC_VECTOR (47 downto 0);
104                m_axis_result_tvalid : out STD_LOGIC;
105                m_axis_result_tready : in STD_LOGIC;
106                m_axis_result_tdata : out  STD_LOGIC_VECTOR (23 downto 0));
107     end component;
108 
109 --! Segnali ausiliari per i due componenti che realizzano il modulo quadro
       di Sum_Early e Sum_Late
110 signal late2buffer: std_logic_vector(63 downto 0);
111 signal late2tvalid : std_logic;
112 signal late2tready : std_logic;
113 signal early2buffer: std_logic_vector(63 downto 0);
114 signal early2tvalid : std_logic;
115 signal early2tready : std_logic;
116 
117 --! Segnali ausiliari per il componente che realizza la divisione tra il
       modulo di Sum_Early e il modulo di Sum_Late
118 signal quozient_tdata : std_logic_vector(87 downto 0);
119 signal quozient_tvalid : std_logic;
120 signal quozient_tready : std_logic;
121 
122 --! Segnale ausiliare per gestire il dato in uscita da rappresentare su 24
       bit,
123 --! di cui 13 costituiscono la parte intera e 11 quella decimale.
124 --! Il valore è rapppresentato come signed.
125 signal root_value : std_logic_vector (23 downto 0);
126 
127 begin
128 
129 ABS_SQR_EARLY:  AXI4_Stream_Absolute_Square_m
130         PORT MAP (
131                aresetn => aresetn,
132                aclk => aclk,
133                s_axis_value_tdata => s_axis_sum_early_tdata,
134                s_axis_value_tvalid => s_axis_sum_early_tvalid,
135                s_axis_value_tready => s_axis_sum_early_tready,
136                m_axis_abssqr_tdata => early2buffer,
137                m_axis_abssqr_tvalid => early2tvalid,
138                m_axis_abssqr_tready => early2tready);
139 
140 ABS_SQR_LATE:  AXI4_Stream_Absolute_Square_m
```

```
141         PORT MAP (
142                 aresetn => aresetn,
143                 aclk => aclk,
144                 s_axis_value_tdata => s_axis_sum_late_tdata,
145                 s_axis_value_tvalid => s_axis_sum_late_tvalid,
146                 s_axis_value_tready => s_axis_sum_late_tready,
147                 m_axis_abssqr_tdata => late2buffer,
148                 m_axis_abssqr_tvalid => late2tvalid,
149                 m_axis_abssqr_tready => late2tready);
150 
151 DIVIDER: AXI4_Stream_Divider_m
152             PORT MAP(
153                   aclk => aclk,
154                   aresetn => aresetn,
155                   s_axis_divisor_tvalid => late2tvalid,
156                   s_axis_divisor_tready => late2tready,
157                   s_axis_divisor_tdata => late2buffer,
158                   s_axis_dividend_tvalid => early2tvalid,
159                   s_axis_dividend_tready => early2tready,
160                   s_axis_dividend_tdata => early2buffer,
161                   m_axis_dout_tvalid => quozient_tvalid,
162                   m_axis_dout_tready => quozient_tready,
163                   m_axis_dout_tdata => quozient_tdata);
164 
165 
166 SQUARE_ROOT: AXI4_Stream_Square_Root_m
167           PORT MAP (
168                 aclk => aclk,
169                 aresetn => aresetn,
170                 s_axis_value_tvalid => quozient_tvalid,
171                 s_axis_value_tready => quozient_tready,
172                 s_axis_value_tdata => quozient_tdata(47 downto 0),
173                 m_axis_result_tvalid => m_axis_r_tvalid,
174                 m_axis_result_tready => m_axis_r_tready,
175                 m_axis_result_tdata => root_value);
176 
177 m_axis_r_tdata <= '0' & root_value(22 downto 0);
178 
179 end Structural;
```

Codice 1.11: "Task4 v2.1"

1.7.3 Testing

Per verificare il funzionamento, viene eseguito, in un primo momento, un semplice testbench, fig. 1.19, fornendo in input al Task una coppia di dati generati in MATLAB. Si può notare come il risultato finale viene restituito in uscita dopo 128 cicli di clock, impostando idealmente un clock in ingresso a 100 Mhz.

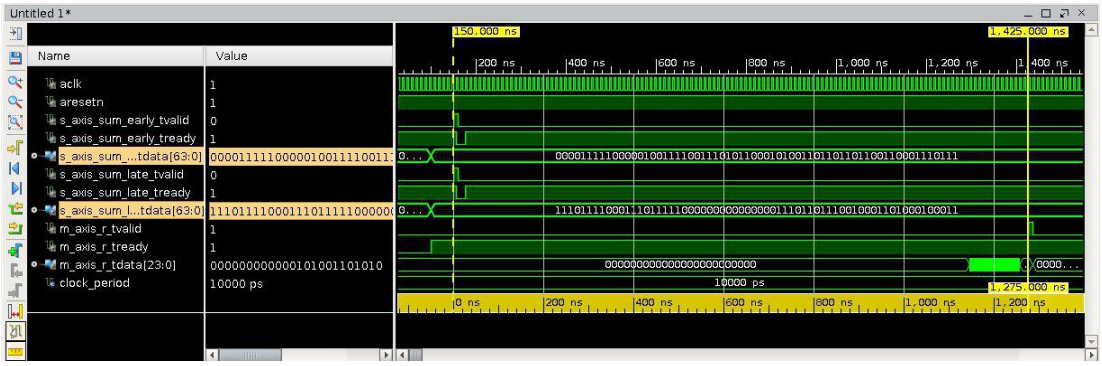


Figura 1.19: Testbench

Successivamente, viene eseguito un testbench che, presi in ingresso dei dati generati dallo script Matlab T4_data_generator.m, fornito dall'ing. Ricci, restituisce i risultati calcolati dal componente e li stampa nel file outputR_Vivado.txt.

```
1  ----------------------------------------------------------------------------------
2  --! @file     Task4_v2_0/tb/Task4_automatic_tb.vhd
3  --! @authors
4  --!             Colella Gianni      <gian.colella@studenti.unina.it>
       <br>
5  --!             Guida Ciro          <ciro.guida4@studenti.unina.it>
       <br>
6  --!             Lombardi Daniele    <daniele.lombardi@studenti.unina.it>
       <br>
7  --! @version  V1.0
8  --! @date     17-July-2017
9  --! @copyright
10 --!  Copyright (C) 2017
11 --!  Colella Gianni      <gian.colella@studenti.unina.it>        <br>
12 --!  Guida Ciro          <ciro.guida4@studenti.unina.it>         <br>
13 --!  Lombardi Daniele    <daniele.lombardi@studenti.unina.it>    <br>
14 --!  This file is part of Task4. It is realized from Group IV of Embedded
       System
15 --!  Class, University of Naples "Federico II", in the academic year
       2016/17.
16 --!
17 --!  This file is part of Task4.
18 --!
19 --!  Task4 is free software: you can redistribute it and/or modify
20 --!  it under the terms of the GNU Affero General Public License as
       published by
21 --!  the Free Software Foundation, either version 3 of the License, or
22 --!  (at your option) any later version.
23 --!
24 --!  Task4 is distributed in the hope that it will be useful,
25 --!  but WITHOUT ANY WARRANTY; without even the implied warranty of
26 --!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
27 --!  GNU Affero General Public License for more details.
28 --!
29 --!  You should have received a copy of the GNU Affero General Public
   License
30 --!  along with Linux Driver: Examples. If not, see
31 --!  <https://www.gnu.org/licenses/agpl-3.0.html>.
32 --! @brief Con questo file si effettua un test bench su un numero
   significativo di
33 --!        elementi, pari a 1000. I dati in input al componente Task 4
   vengono letti
34 --!        da 2 file generati da uno script Matlab (inputEarly.txt e
   inputLate.txt)
35 ----------------------------------------------------------------------------
36 library IEEE;
37 use IEEE.STD_LOGIC_1164.ALL;
38 use std.textio.all;             --! Package necessario per la lettura e
   scrittura di file
39 use ieee.std_logic_textio.all;
40
41 entity Task4_automatic_tb is
42 --  Port ( );
43 end Task4_automatic_tb;
44
45 architecture Behavioral of Task4_automatic_tb is
46
47 ----------------------------------------------------------------------------
48 ---------------------------Task 4 Component---------------------------------
49 ----------------------------------------------------------------------------
50     component Task4_m is
51         Port ( aclk : in STD_LOGIC;
52                aresetn : in STD_LOGIC;
53                s_axis_sum_early_tvalid :  in STD_LOGIC;
54                s_axis_sum_early_tready :  out STD_LOGIC;
55                s_axis_sum_early_tdata : in STD_LOGIC_VECTOR (63 downto 0);
56                s_axis_sum_late_tvalid :  in STD_LOGIC;
57                s_axis_sum_late_tready :  out STD_LOGIC;
58                s_axis_sum_late_tdata : in STD_LOGIC_VECTOR (63 downto 0);
59                m_axis_r_tvalid : out std_logic;
60                m_axis_r_tready : in std_logic;
61                m_axis_r_tdata : out STD_LOGIC_VECTOR (23 downto 0)
62                );
63     end component;
64
65     signal aclk : STD_LOGIC;
66     signal aresetn : STD_LOGIC := '1';
67     signal s_axis_sum_early_tvalid : STD_LOGIC:='0';
68     signal s_axis_sum_early_tready : STD_LOGIC;
69     signal s_axis_sum_early_tdata : STD_LOGIC_VECTOR (63 downto 0) := (
       others => '0');
```

```
70      signal s_axis_sum_late_tvalid : STD_LOGIC:='0';
71      signal s_axis_sum_late_tready : STD_LOGIC;
72      signal s_axis_sum_late_tdata : STD_LOGIC_VECTOR (63 downto 0)  := (
            others => '0');
73      signal m_axis_r_tvalid : std_logic;
74      signal m_axis_r_tready : std_logic := '1';
75      signal m_axis_r_tdata : STD_LOGIC_VECTOR (23 downto 0);
76
77      --! Vengono definiti i nomi dei file da utilizzare durante il test bench
78    file data_early : text;         --! Questo è il file in cui sono presenti i
          dati di Sum_Early
79    file data_late : text;          --! Questo è il file in cui sono presenti i
          dati di Sum_Late
80    file data_r : text;             --! Questo è il file in cui saranno salvati
          i risultati di R
81
82  constant clock_period: time := 10 ns;
83  begin
84
85  uut: Task4_m
86      Port map(
87           aclk => aclk,
88           aresetn => aresetn,
89           s_axis_sum_early_tvalid => s_axis_sum_early_tvalid,
90           s_axis_sum_early_tready  => s_axis_sum_early_tready,
91           s_axis_sum_early_tdata  => s_axis_sum_early_tdata,
92           s_axis_sum_late_tvalid  => s_axis_sum_late_tvalid,
93           s_axis_sum_late_tready => s_axis_sum_late_tready,
94           s_axis_sum_late_tdata  => s_axis_sum_late_tdata,
95           m_axis_r_tvalid  => m_axis_r_tvalid,
96           m_axis_r_tready  => m_axis_r_tready,
97           m_axis_r_tdata  => m_axis_r_tdata);
98
99  --! Clock process definitions
100 clock_process :process
101     begin
102      aclk <= '0';
103      wait for clock_period/2;
104      aclk <= '1';
105      wait for clock_period/2;
106 end process;
107
108
109 --! Stimulus process
110   stim_proc: process
111
112     --! Variabili utilizzate per la lettura e scrittura dei file
113     variable early_file_line : line;      --! Variabile associata alla linea
          letta dal file data_early
```

```
114     variable late_file_line : line;       --! Variabile associata alla linea
            letta dal file data_late
115     variable r_file_line : line;          --! Variabile associata alla linea
            scritta nel file data_r
116
117     variable read_sum_early : std_logic_vector (63 downto 0) := (others
           =>'0');   --!Variabile associata al valore letto di sum_early
118     variable read_sum_late : std_logic_vector (63 downto 0) := (others=>'0')
           ;    --!Variabile associata al valore letto di sum_late
119
120     variable write_r : std_logic_vector (23 downto 0) := (others => '0');
                 --!Variabile associata al valore scritto di r
121
122
123     begin
124        -- hold reset state for 100 ns.
125        wait for clock_period*10;
126
127
128       file_open(data_early, "/home/daniele/Scrivania/Task4/testing2.0/
             inputEarly.txt",  read_mode);   --! apertura del file inputEarly
129       file_open(data_late, "/home/daniele/Scrivania/Task4/testing2.0/
             inputLate.txt", read_mode);      --! apertura del file inputLate
130       file_open(data_r, "/home/daniele/Scrivania/Task4/testing2.0/
             outputR_Vivado.txt", write_mode);   --! apertura del file outputR
131
132
133       while not endfile(data_early) loop       --! Il ciclo serve per
             leggere tutti i dati contenuti nei file di input
134                                                --! poichè la loro dimensione
                                                       è la stessa, la
                                                       condizione di
                                                       terminazione
135                                                --! del ciclo può essere
                                                       fatta indifferentemente
                                                       usando la funzione
136                                                --! endfile(   ) su uno dei
                                                       due file.
137         readline(data_early,early_file_line);
138         read(early_file_line,read_sum_early);
139         s_axis_sum_early_tdata<=read_sum_early;
140
141         readline(data_late,late_file_line);
142         read(late_file_line,read_sum_late);
143         s_axis_sum_late_tdata<=read_sum_late;
144
145         s_axis_sum_early_tvalid <='1';       --! Senza perdere di generalità
                si suppone che i segnali tvalid dei dati
146           s_axis_sum_late_tvalid <='1';        --! in input arrivino
```

```
            contemporaneamente
147
148         wait for clock_period;                --! Per simulare il
                comportamento del protocollo AXI4 Stream
149             s_axis_sum_early_tvalid <='0';    --! dopo un colpo di clock i
                    segnali tvalid sono messi a zero
150             s_axis_sum_late_tvalid <='0';
151
152         wait until m_axis_r_tvalid = '1';     --! Qui il test deve arrestarsi
                in attesa che il segnale tvalid sull'interfaccia
153                                                 --! AXI4 Stream sia alto. Ciò
                                                        sta a significare che il
                                                        componente ha finito di
154                                                 --! processare i segnali che ha
                                                        ricevuto in ingresso
155         write(r_file_line, m_axis_r_tdata, right, 24);
156         writeline(data_r, r_file_line);
157
158           end loop;
159
160         --! Chiusura dei file precedentemente aperti
161       file_close(data_early);
162       file_close(data_late);
163       file_close(data_r);
164        wait;
165     end process;
166
167 end Behavioral;
```

Codice 1.12: "Task4 automatic tb.vhd"

Ottenuto il file *outputR_ Vivado.txt*, viene eseguito il seguente script in Matlab per poter calcolare errore relativo e assoluto tra i valori generati in Matlab e quelli calcolati dal componente realizzato.

```
1  %% CALCOLO ERRORI
2  % Con questo piccolo script in codice MATLAB si vuole calcolare l'errore
3  % relativo e l'errore assoluto dei valori di output del TASK 4.
4  % Tali valutazioni sono effettuate confrontando i
5  % valori di R generati dal codice MATLAB e forniti al gruppo IV da chi ha
6  % commissionato questo progetto, con i valori ottenuti dall'IP%core custom,
7  % sotto ambiente Vivado, realizzato ad hoc per rispettare le specifiche di
8  % progetto.
9
10 q=quantizer([24 13]);
11 [bin_r_mat,dec_r_mat]=range(q);
12 [bin_r_viv,dec_r_viv]=range(q);
13
14 format longE;
15 r_file_matlab = 'outputR_Matlab.txt';
```

```
r_file_vivado = 'outputR_Vivado.txt';
error_abs_file = 'error_abs.txt';
error_rel_file = 'error_rel.txt';

%leggi e carica i dati binari dai file
%trasformo i dati in decimale cosi' da poter fare un confronto e valutare l'
    errore relativo e assoluto
r_mat=textread(r_file_matlab,'%s');
bin_r_mat= r_mat(1:1:end);
dec_r_mat=bin2num(q,bin_r_mat);%r in decimale signed su 13 bit intero e 11
    parte decimale


r_viv=textread(r_file_matlab,'%s');
bin_r_viv= r_viv(1:1:end);
dec_r_viv=bin2num(q,bin_r_viv);%r in decimale signed su 13 bit intero e 11
    parte decimale

dec_r_viv=cell2mat(dec_r_viv);
dec_r_mat=cell2mat(dec_r_mat);

error_abs_dec= abs(dec_r_viv % dec_r_mat);
dlmwrite(error_abs_file,error_abs_dec,' ');
error_rel_dec= abs(dec_r_viv%dec_r_mat)./dec_r_mat;
dlmwrite(error_rel_file,error_abs_dec,' ');
```

Codice 1.13: "errors evaluation.m"

Il testing effettuato su un numero di 1000 campioni ha restituito errore relativo e assoluto pari a 0 per tutti i valori di r. Dunque, avendo una rappresentazione di r su 24 bit, di cui 11 decimali, si può commettere un errore minore di 2^{-12}.

1.7.4 Analisi soluzione

Di seguito si riportano le occupazioni d'area di ogni singolo modulo e del componente totale.

Componente	LUT	Slice Register	DSP48
Absolute Square	165	66	8
Divider generator	1736	3806	0
Square Root	233	124	0
Totale	2300(13%)	4062(12%)	16(20%)

Tabella 1.7: Occupazione d'area post-sintesi

Si può notare che, rispetto alla versione precedente, c'è un leggero risparmio delle risorse. La frequenza di lavoro è confermata a $80,901 Mhz$.

1.8 Conclusioni

In conclusione, dai dati proposti in precedenza emerge che la soluzione migliore realizzata è l'ultima analizzata. Infatti, si ha il massimo risparmio di area tra quelle proposte, come anche il minor numero di cicli di clock necessari al calcolo del risultato.

I test effettuati dimostrano che il componente restituisce il risultato corretto nel 100% dei casi.

Per garantire il facile riutilizzo del componente, ne è stato creato un IP-core con licenza **GNU Affero General Pubblic License 3.0**.