

# Progetto Finale Task 4

Colella Gianni – M63/670

Guida Ciro – M63/592

Lombardi Daniele – M63/576

UNIVERSITA' DEGLI STUDI DI  
NAPOLI FEDERICO II

Corso: Sistemi Embedded

Anno Accademico: 2016-2017

Prof. Antonino Mazzeo  
Ing. Mario Barbareschi

# Traccia

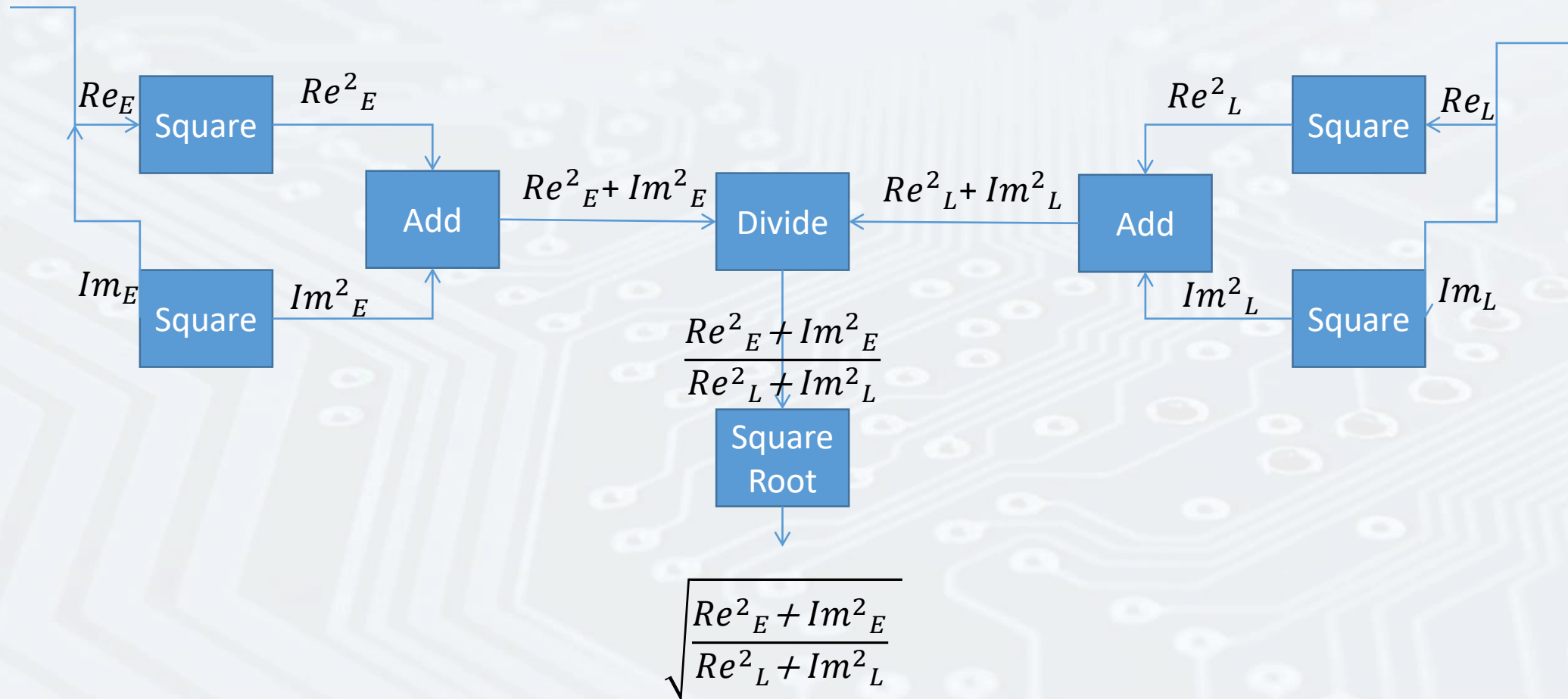
Si realizzi un IP Core che implementi il raffinamento del calcolo del delay che avviene durante la fase di Tracking, relativamente alla seconda parte durante la quale bisogna effettuare i moduli delle sommatorie, ottenute durante lo step precedente, calcolarne il rapporto e ricavarne la radice quadrata

- Per la realizzazione del task si richiede l'implementazione di
  1. Moltiplicatore
  2. Sommatore
  3. Divisore
  4. Radice quadrata

# Schema di Principio

$$SumEarly = Re_E + j Im_E$$

$$SumLate = Re_L + j Im_L$$



# MATLAB

```
64 %% Delay Deviation Estimation
65 for bbb = 1:nr_block
66     index_P = (1+(bbb-1)*sample_in_P:bbb*sample_in_P);
67     Data_preConditioned(index_P) = Data_plus_Noise_Block(index_P).*DRR_in_P.*DRR_in_B(bbb)*SS_a_p(bbb);
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);
```

Da Test2\_DelayDeviationAndAlignment.m

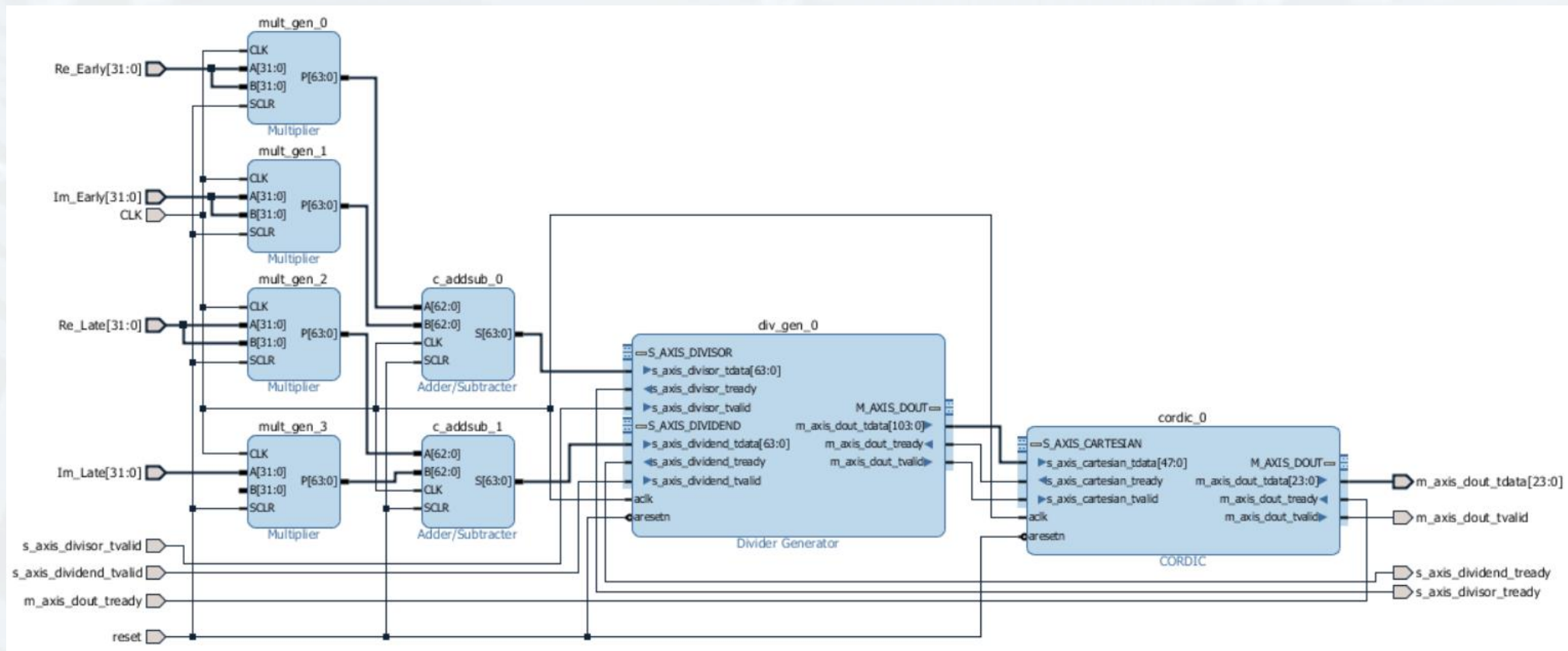
```
62 % Compute operations
63 reE2=real(sigEarly).^2;
64 imE2=imag(sigEarly).^2;
65 reL2=real(sigLate).^2;
66 imL2=imag(sigLate).^2;
67 sE=reE2+imE2;
68 sL=reL2+imL2;
69 d1=sE./sL;
70 R=sqrt(d1);
```

Da T4\_data\_generator.m

# Realizzazione Task 4 v.1.0

## Impiego di soli IP Core Xilinx

Multiplier - Adder/Subtractor – Divider Generator - Cordic



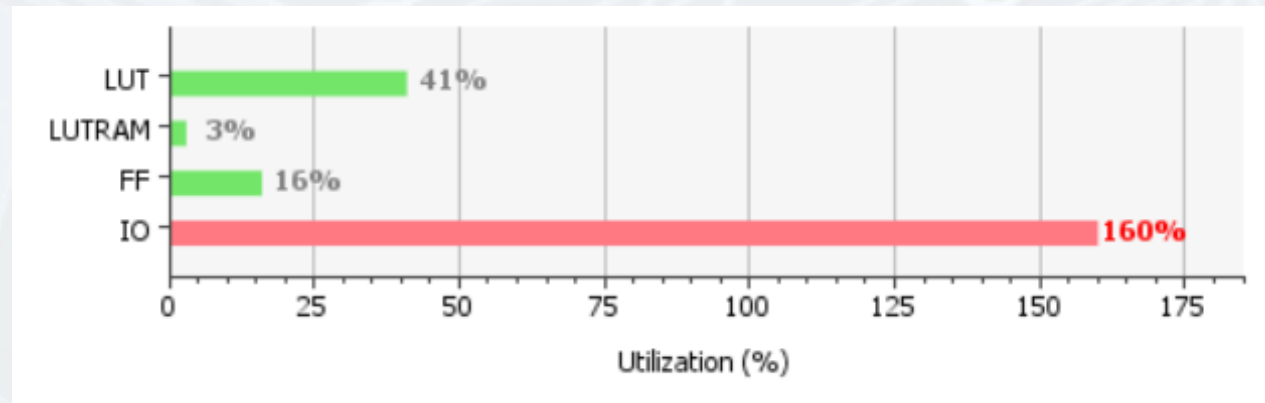
# Analisi dell'area occupata

Componente		LUT	Slice Reg.	DSP48
Multiplier	x4	1103	64	0
Adder	x2	63	64	0
Divider Generator	x1	2036	4474	0
Cordic	x1	673	892	0
TOT:		(41%) 7248	(16%) 5750	(0%) 0

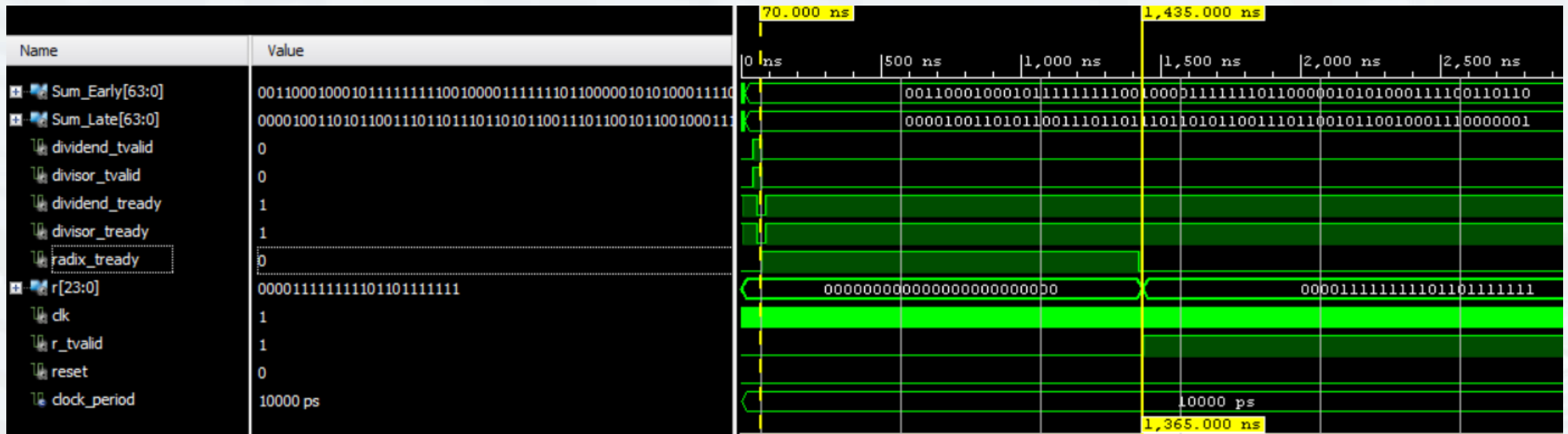
Tali dati sono valutati post - synthesis



Name	^1	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	Bonded IOB (100)
Task4		7248	5750	8	160
DIVISOR (DIVISORE)		2036	4474	8	0
U0 (DIVISORE_div_gen_...)		2036	4474	8	0
SQUARE (modulo_quadro)		4538	384	0	0
E2 (c_addsub_0)		63	64	0	0
IM2E (Square)		1103	64	0	0
IM2L (Square)		1103	64	0	0
L2 (c_addsub_0)		63	64	0	0
RE2E (Square)		1103	64	0	0
RE2L (Square)		1103	64	0	0
SQUARE_ROOT (SQRT)		673	892	0	0
U0 (SQRT_cordic_v6_0_...)		673	892	0	0



# Analisi dei Tempi



Si riporta un esempio di simulazione behavioral in cui il clock di ingresso è a 100 MHz.  
Il risultato è restituito dopo 136 Cicli di Clock.



# Analisi soluzione Task 4 v. 1.0

## Vantaggi:

- Velocità e semplicità di implementazione;
- Riutilizzo dei componenti;

## Svantaggi:

- Gestione dei segnali tvalid di ingresso;
- Non interoperabilità con AXI Stream;
- Area occupata;

# Componenti in Analisi

## Modulo Quadro

1. Due IpCore Multiplier e un IP Core Adder/Subtractor
2. Operatori VHDL \* e +
3. Due Moltiplicatori di Booth e un Ripple Carry Adder
4. Due Moltiplicatori MAC e un Ripple Carry Adder

## Divisore

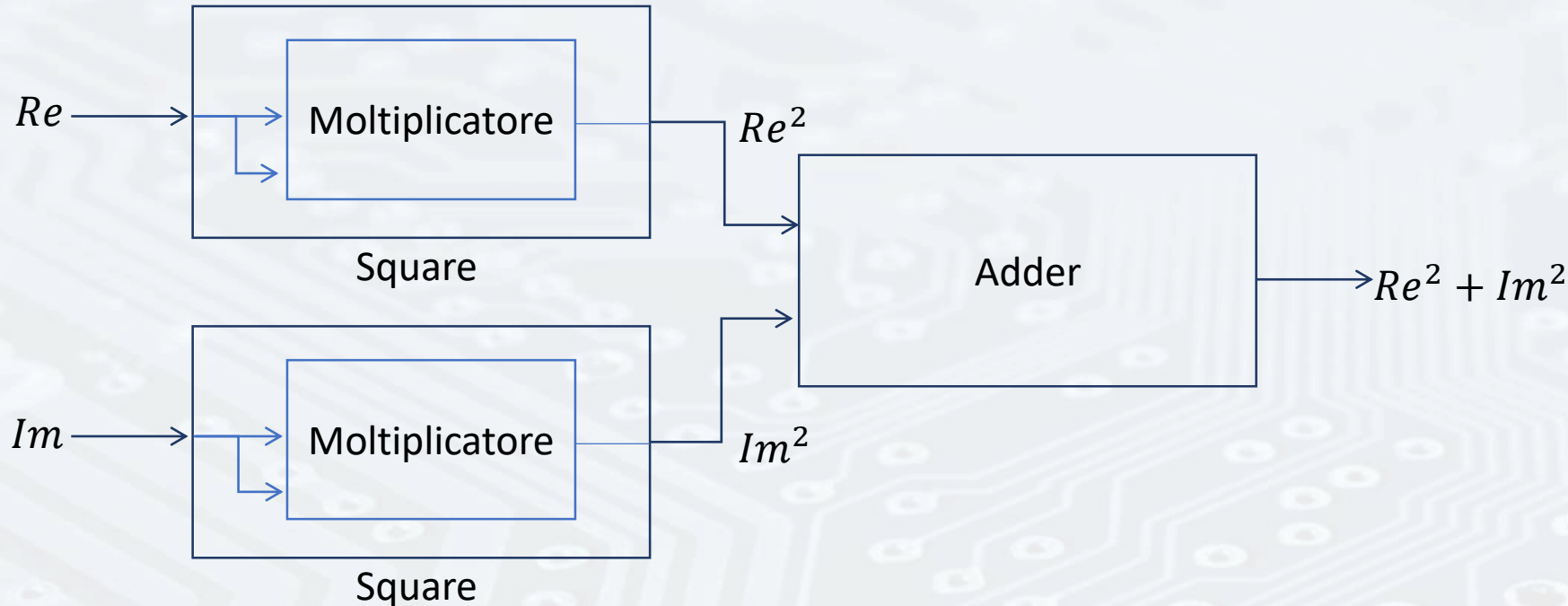
1. IPCore Divider Generator
2. Divisore Non Restoring
3. Operatore VHDL /

## Radice Quadrata

1. IPCore CORDIC
2. Algoritmo Digit-by-Digit Custom Combinatorio
3. Algoritmo Digit-by-Digit Custom Sequenziale

# Componente n.1 : Modulo Quadro

Il Modulo quadro è suddiviso a sua volta in 3 sotto-componenti: due Square e un Adder.



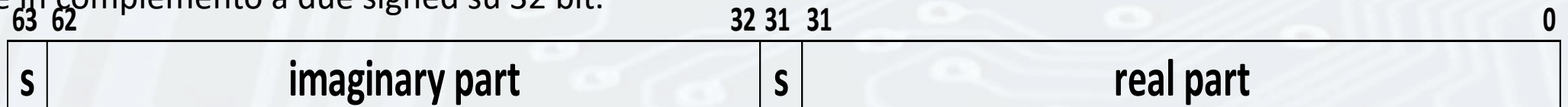
Square:

- Moltiplicatore di Booth
- Moltiplicatore MAC
- IPCore Multiplier
- Operatore VHDL \*

Adder:

- RCA
- IPCore Adder/Subtractor
- Operatore VHDL +

Il componente ha ingresso un numero complesso espresso su 64 bit . La parte immaginaria e la parte reale sono rappresentate in complemento a due signed su 32 bit.



Il componente Square è un moltiplicare che ha in ingresso un signed a 32 bit e in uscita un signed su 64 bit.

Il componente Adder è un addizionatore con 64 bit signed in ingresso e 65 bit in uscita, ma di cui viene troncato il MSB in quanto il modulo quadro è sempre positivo.

# Modulo Quadro pt.2

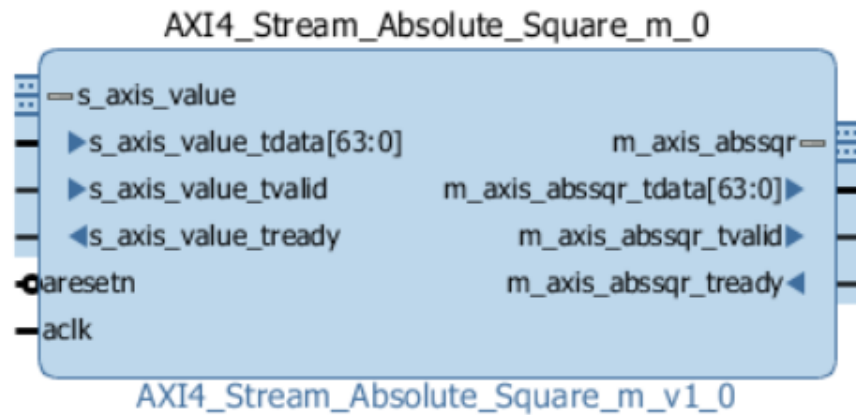
## Implementazioni analizzate

1. Due IpCore Multiplier e un IP Core Adder/Subtractor;
2. Operatori VHDL \* e +;
3. Due Moltiplicatori di Booth e un Ripple Carry Adder;
4. Due Moltiplicatori MAC e un Ripple Carry Adder;

Implementazione numero	Area			Tempi	
	LUT Register	Slice Register	DSP48	Frequenza	Cicli di clock
1	1283	106	0	69.686 MHz	2
2	158	0	8	80.901 MHz	2
3	425	264	0	82.967 MHz	[correggi]
4	5468	128	0	20.927MHz	2

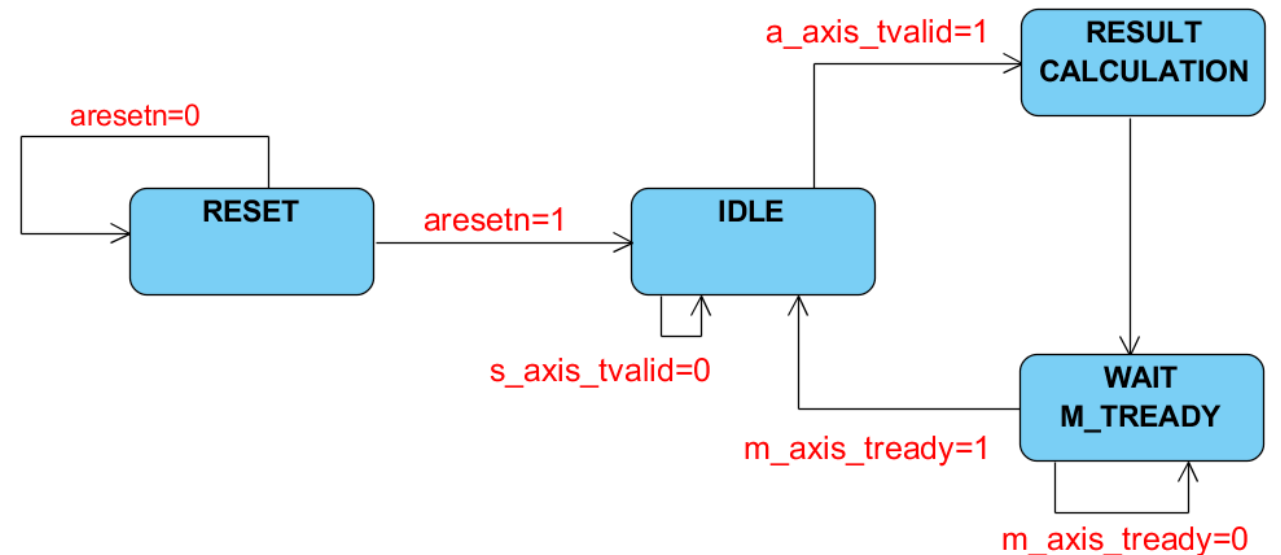
Tali dati sono valutati post - implementation

# Absolut Square



Si aggiunge compatibilità con AXI Stream

```
data_im2<=data_im*data_im;  
data_re2<=data_re*data_re;  
data_mod<=data_im2+data_re2;
```



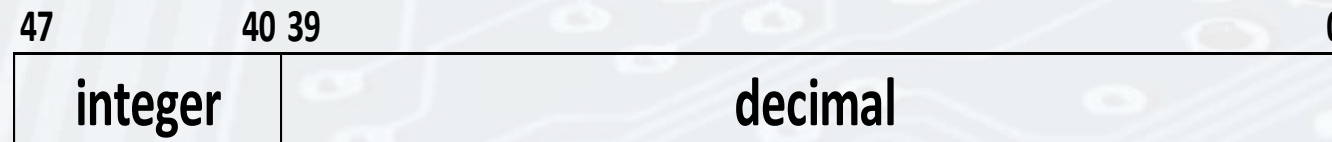
## Componente n.2 : Divisore

Il Divisore deve eseguire l'operazione di divisione tra il modulo quadro di Early e il modulo quadro di Late



Il componente ha ingresso due valori unsigned integer espressi su 64 bit .

In uscita il divisore restituisce una numero unsigned espresso in fixed point su 48 bit, di cui 40 bit usati per la parte decimale e 8 bit per la parte intera.





# Divisore pt.2

## Implementazioni analizzate

1. IpCore Divider Generator
  1. Clocks per division: 1
  2. Clocks per division: 2
  3. Clocks per division: 8
2. Operatore VHDL /;
3. Divisore non restoring;

Implementazione numero	Area			Tempi	
	LUT Register	Slice Register	DSP48	Frequenza	Cicli di clock
1.1	7168	17165	0	163,514 MHz	91
1.2	7089	9144	0	102,776 MHz	91
1.3	1819	2635	0	104,493 MHz	91
2	6850	171	0	2,407 MHz	2
3	5996	428	0	83,198 MHz	104

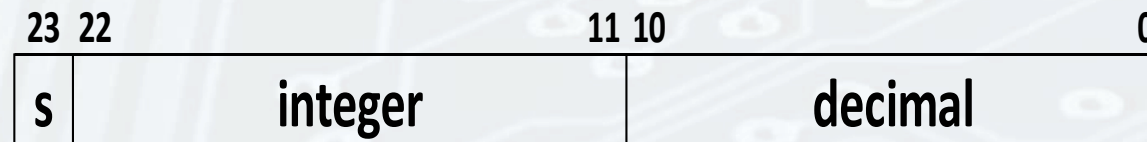
Tali dati sono valutati post - implementation

## Componente n.3 : Radice Quadrata



Il componente ha ingresso un valore fixed point espresso su 48 bit: <48,40>.

In uscita restituisce un numero signed espresso in fixed point signed su 24 bit: <24,11>



# Radice Quadrata pt.2

Implementazioni analizzate

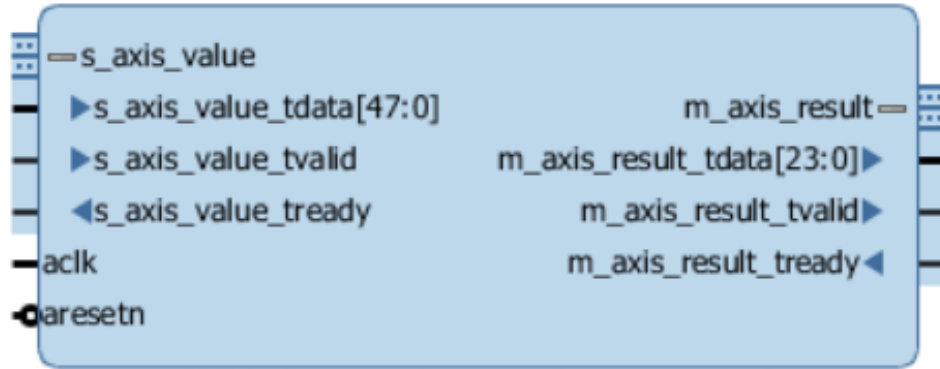
1. IpCore Cordic;
2. Algoritmo Digit-by-Digit Custom Combinatorio;
3. Algoritmo Digit-by-Digit Custom Sequenziale;

Implementazione numero	Area			Tempi	
	LUT Register	Slice Register	DSP48	Frequenza	Cicli di clock
1	741	403	0	130,056 MHz	24
2	1917	74	0	13,818 MHz	2
3	202	121	0	120,642 MHz	25

Tali dati sono valutati post - implementation

# Square\_Root

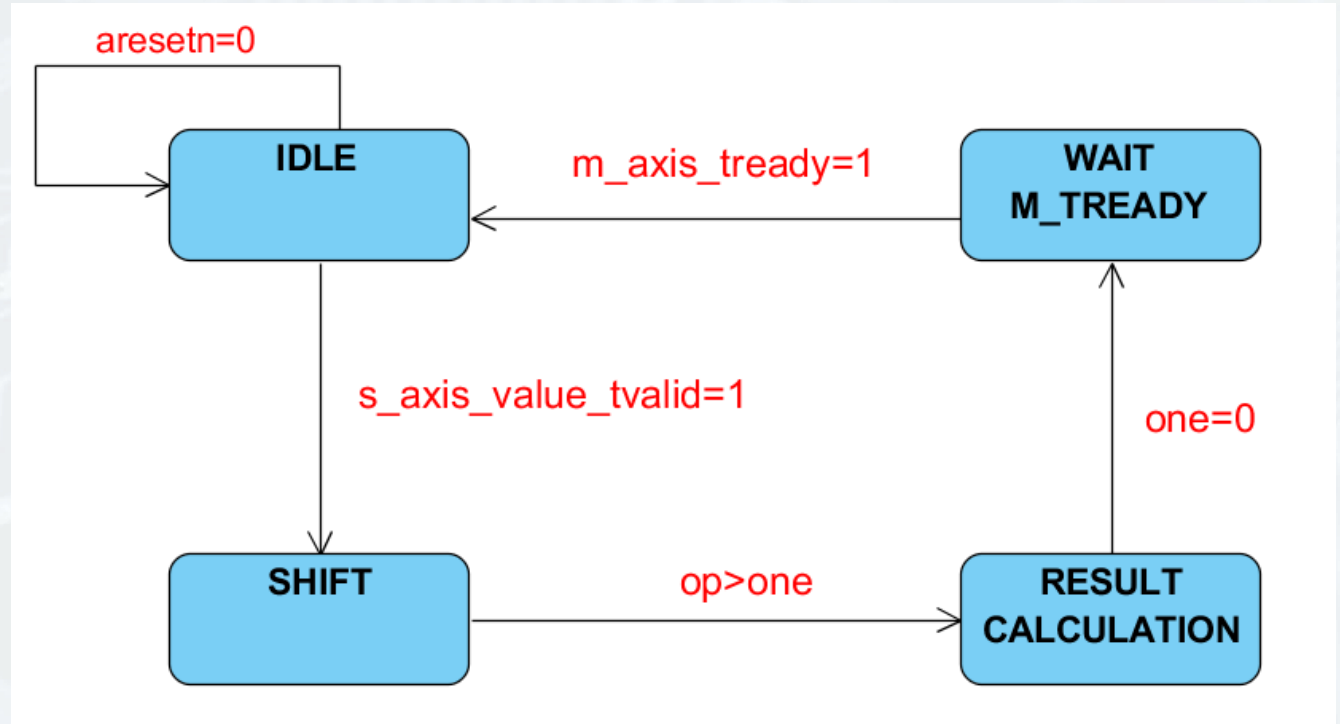
AXI4\_Stream\_Square\_Root\_m\_0



AXI4\_Stream\_Square\_Root\_m\_v1\_0

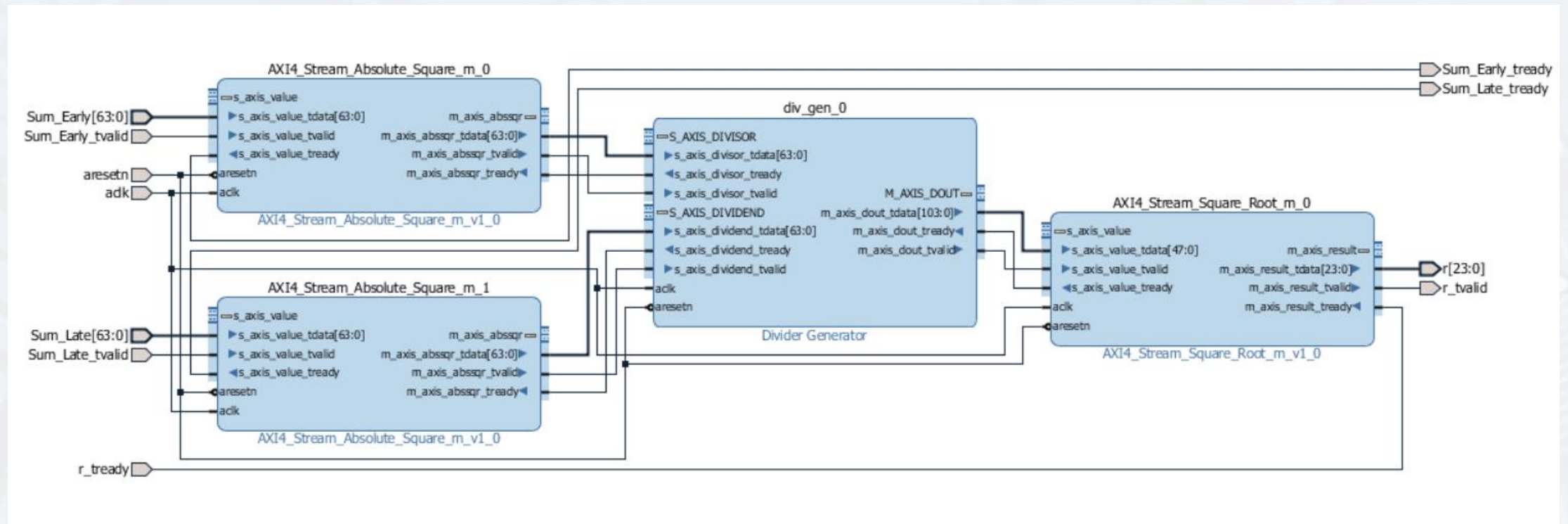
```
int sqrt(int num, int precision){  
    int op = num;  
    int res = 0;  
    int one = 1 << precision;  
    while (one > op)  
        one >>= 2;  
    while (one != 0){  
        if (op >= res + one) {  
            op -= res + one;  
            res += 2 * one;  
        }  
        res >>= 1;  
        one >>= 2;  
    }  
    return res; }  
}
```

Si aggiunge compatibilità con AXI Stream



# Realizzazione Task 4 v.2.0

Modulo Quadro con Operatori VHDL – Divider Generator – Algoritmo Custom Sequenziale



# Analisi dell'area occupata

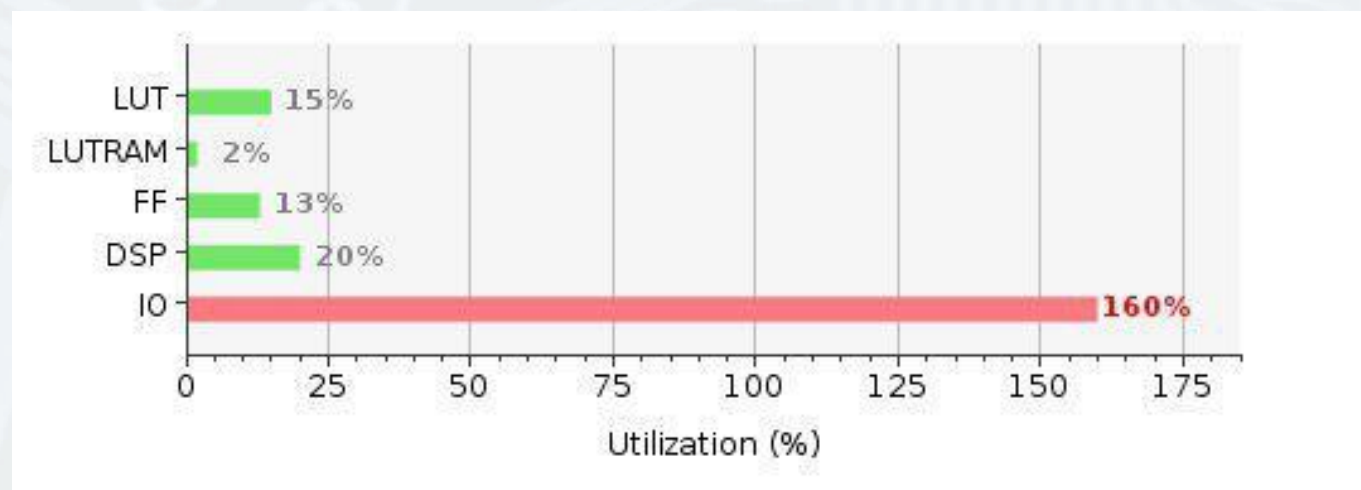
Component		LUT	Slice Reg.	DSP48
Absolute_Square	x2	165	66	8
Divider Generator	x1	2036	4474	0
Square_Root	x1	223	124	0

TOT:	(15%) 2600	(14%) 4730	(20%) 16
------	------------	------------	----------

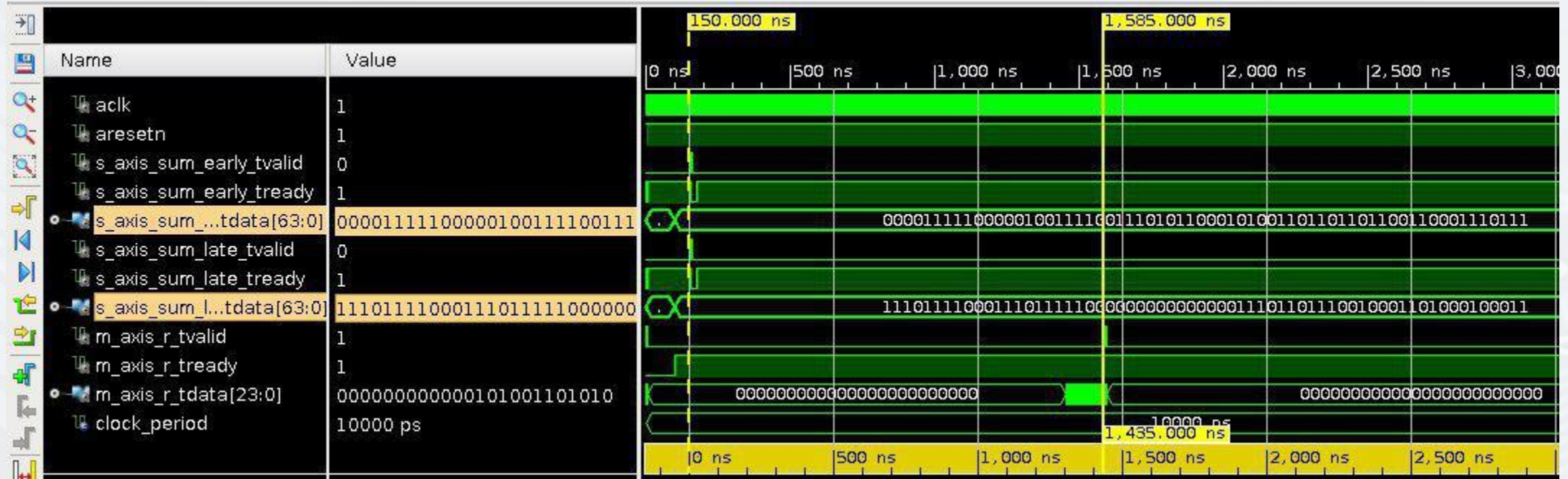
Tali dati sono valutati post - synthesis



Hierarchy							
	Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	DSPs (80)	Bonded IOB (100)	BUFGCTRL (32)
	Task4_m	2600	4730	8	16	160	1
	ABS_SQR_EARLY (AXI4...	165	66	0	8	0	0
	ABS_SQR_LATE (AXI4...	166	66	0	8	0	0
	DIVIDER (AXI4_Stream...	2036	4474	8	0	0	0
	SQUARE_ROOT (AXI4...	233	124	0	0	0	0



# Tempi



Si riporta un esempio di simulazione behavioral in cui il clock di ingresso è a 100 MHz.  
Il risultato è restituito dopo 144 Cicli di Clock.

# Criticità superate rispetto alla versione 1.0:

Tutti i componenti sono interfacciati con AXI Stream;

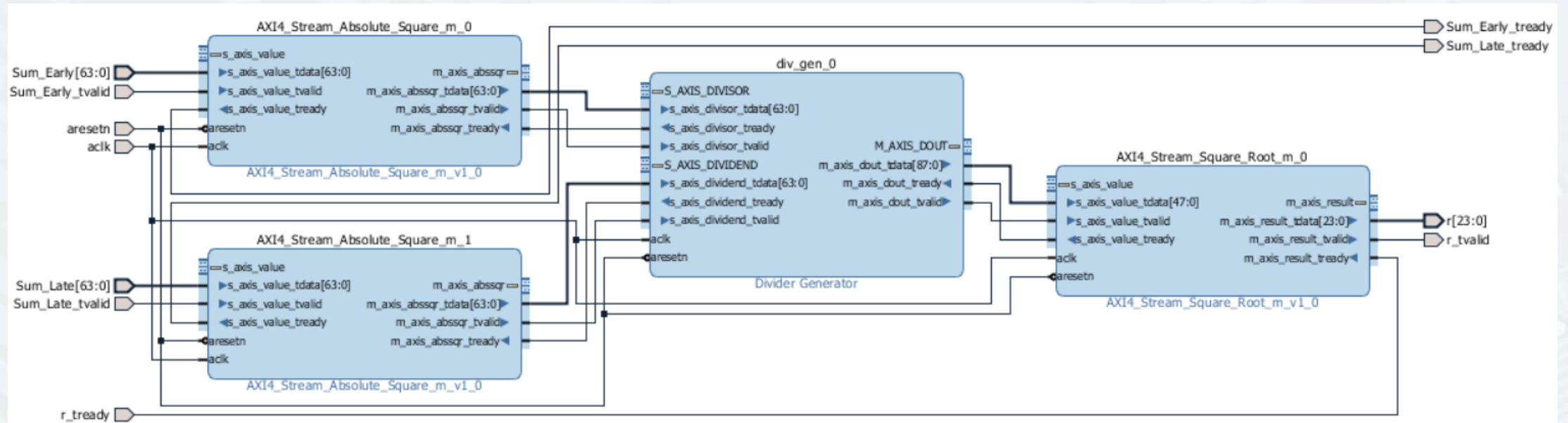
Ottimizzazione dell'area occupata;

	v1.0	v2.0	
LUT	7248	2600	-179%
Slice Register	5750	4730	-22%

Valutazione Tempi tra le due soluzioni:

	v1.0	v2.0	
Cicli di Clock	137	144	4,86%

# Realizzazione Task 4 v.2.1



Si adatta la cardinalità della parte frazionaria del valore di uscita del Divider Generator

# Analisi dell'area occupata

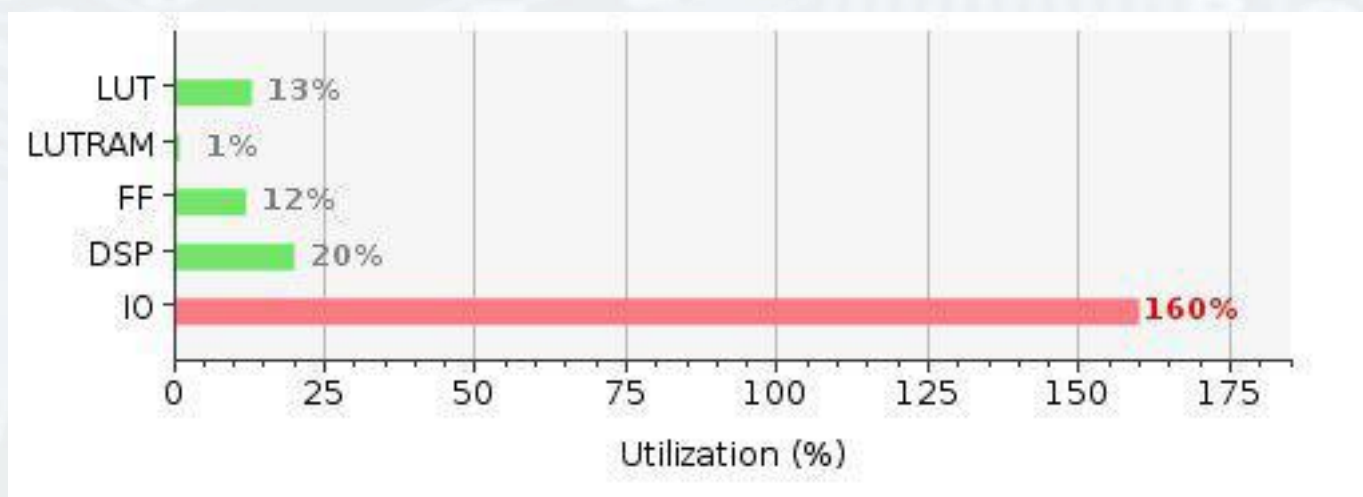
Component		LUT	Slice Reg.	DSP48
Absolute_Square	x2	165	66	8
Divider Generator	x1	1735	3806	0
Square_Root	x1	223	124	0

TOT:	(13%) 2300	(12%) 4062	(20%) 16
------	------------	------------	----------

Tali dati sono valutati post - synthesis

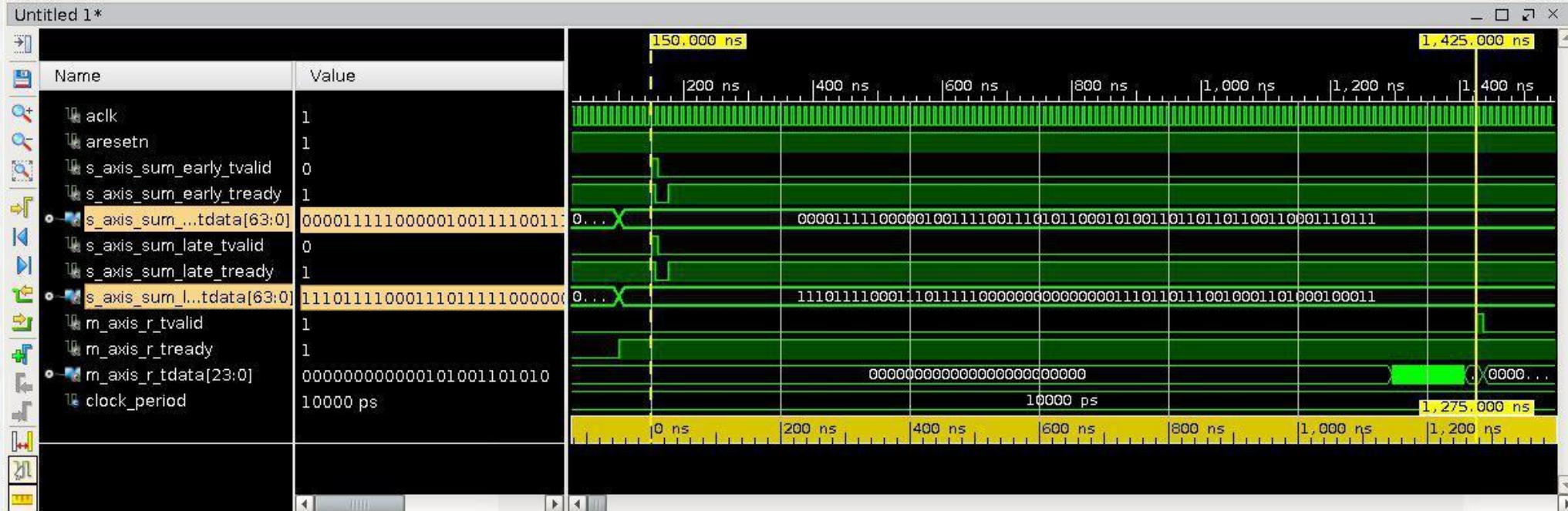


Hierarchy							
	Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	DSPs (80)	Bonded IOB (100)	BUFGCTRL (32)
	Task4_m	2300	4062	8	16	160	1
	ABS_SQR_EARLY (AXI4...	165	66	0	8	0	0
	ABS_SQR_LATE (AXI4...	166	66	0	8	0	0
	DMIDER (AXI4_Stream...	1736	3806	8	0	0	0
	SQUARE_ROOT (AXI4_...	233	124	0	0	0	0





# Tempi



Si riporta un esempio di simulazione behavioral in cui il clock di ingresso è a 100 MHz.  
Il risultato è restituito dopo 128 Cicli di Clock.

# Miglioramenti rispetto alla versione 2.0:

Ottimizzazione dell'area occupata;

	v2.0	v2.1	
LUT	2600	2300	-13,04%
Slice Register	4730	4062	-16,45%

Valutazione Tempi tra le due soluzioni:

	v2.0	v2.1	
Cicli di Clock	144	128	-12,50%

# Testing

Si è generato un testbench che prende in ingresso i file da inputEarly.txt e inputLate.txt generati su MATLAB R2017.a© a partire dallo script 'T4\_data\_generator.m'.

Tutti i valori di r calcolati da Vivado 2016.3 sono stati successivamente salvati in un file testuale così da poter fare delle valutazioni in termini di errore relativo e di errore assoluto rispetto alla soluzione calcolata da MATLAB.

	r
Errore Relativo*	0
Errore Assoluto*	0

$$*\epsilon = 2^{-12}$$

Numero di campioni generati: 1000

Link repository

<https://github.com/F3CDG/Task-4>