Enrico Gori

CORDIC

1. Introduzione

CORDIC é l'acronimo di Cordinate Rotation Digital Computer; Opportune varianti dell'algoritmo possono essere usate per il calcolo delle funzioni trigonometriche, funzioni iperboliche ma anche per il calcolo di moltiplicazioni o divisioni.

Il progetto prevede l'implementazione e la sintesi in hardware dell'algoritmo CORDIC in modalità vectoring per la conversione da coordinate cartesiane a polari di un vettore nel piano XY.

L'algoritmo, descritto per la prima volta nel 1959 da Jack E. Volder, vide il suo primo impiego come calcolatore di funzioni trigonometriche nei computer ad uso bellico e nei sistemi radar. L'algoritmo è stato implementato anche nelle prime calcolatrici scientifiche ed è stato usato anche nei coprocessori matematici della famiglia 80x87. In definitiva l'algoritmo, oggi come allora, trova un impiego tutte le volte in cui si ha necessità di calcolare funzioni complesse ma si hanno scarse risorse hardware come ad esempio nei microcontrollori o nelle FPGA. Un altro possibile impiego è nel calcolo della trasformata di Fourier direttamente in hardware e più in generale in tutti gli ambiti di elaborazione dei segnali.

2. Descrizione dell'algoritmo

Una rotazione di un vettore può essere espressa mediante la matrice di rotazione

$$\begin{bmatrix} \cos\Theta & -\sin\Theta \\ \cos\Theta & \sin\Theta \end{bmatrix}$$

o in maniera del tutto equivalente:

$$\cos\Theta \cdot \begin{bmatrix} 1 & -\tan\Theta \\ 1 & \tan\Theta \end{bmatrix}$$

Inoltre una rotazione può essere ottenuta attraverso delle rotazioni di entità diverse, ad esempio quelle per cui:

$$\tan\Theta = \pm 2^{-i}$$

Quindi una rotazione può essere espressa in maniera ricorsiva:

$$x_{i+1} = K_i\left[x_i - y_i \cdot d_i \cdot 2^{-i}\right]$$
$$y_{i+1} = K_i\left[y_i + x_i \cdot d_i \cdot 2^{-i}\right]$$

Ovvero tramite sempre operazioni di somma, sottrazione e divisioni per 2 (shift verso destra). I risultati sono inoltre funzione del termine d_i il quale deve essere 1 se $y_i < 0$, -1 altrimenti. Il termine K_i è un fattore di scala che per la generica iterazione vale:

$$K_i = \cos\left(\tan^{-1} \cdot 2^{-i}\right) = \frac{1}{\sqrt{1+2^{-2i}}}$$

Nei vari passaggi i termini k_i si accumulano. In conclusione se si ignorano tali termini i risultati saranno maggiorati di una fattore pari a 1.647.
L'algoritmo opera nel seguente modo: si fanno compiere un certo numero di rotazioni al vettore fino a farlo adagiare sull'asse delle ascisse. Alla fine, tenuto conto della posizione del vettore, il termine y_i assumerà il valore 0 e il termine x_i assumerà un valore pari al modulo del vettore ma maggiorato di un fattore 1.647. Come già detto il termine d_i dipende dal segno dell'ordinata ed è il responsabile del senso della rotazione (antioraria od oraria) infatti se siamo nel semipiano delle ordinate positive la prossima rotazione dovrà essere in senso orario ma se siamo nel semipiano delle ordinate negative la rotazione dovrà essere in senso antiorario. Per tenere traccia dell'angolo di rotazione bisogna implementare un'ulteriore equazione.

$$z_{i+1} = z_0 - d_i \tan^{-1}\left(2^{-i}\right)$$

Dove il termine d_i é lo stesso di prima mentre i valori dell'arcotangente possono essere memorizzati in una LUT. Per quanto riguarda z_0, esso lo sfasamento iniziale e può essere pari a 0 se l'ascissa è positiva o pari a π se l'ascissa è negativa. L'algoritmo converge solo se il vettore si trova nel semipiano ad ascissa positiva ed è per questo motivo è necessario fissare z_0 al primo passo di iterazione (tenendo ovviamente conto del valore dell'ascissa) e, contestualmente, ridurre il vettore nel semipiano delle ascisse positive.

3. Possibili architetture

Per implementare l'algoritmo esistono varie possibilità architetturali. Una prima architettura è quella detta unrolled. Questa architettura prevede di avere più blocchi collegati in cascata ciascuno dei quali compie un passo di iterazione. Gli svantaggi principali di questa architettura sono che le operazioni di shift e i valori dell'arcotangente sono cablati direttamente in hardware, inoltre al crescere della precisione richiesta (e quindi al numero di passi dell'algoritmo) corrisponde un aumento dell'area del circuito.
Una seconda soluzione prevede di avere un solo blocco che, opportunamente pilotato da una macchina a stati finiti, adempia alle operazioni di caricamento dei dati, shift degli operandi e alle operazioni di somma e sottrazione necessarie al calcolo del modulo e dell'angolo. Questa architettura può avere due varianti che interessano l'implementazione delle operazioni di shift. Una prima variante prevede di usare dei barrel shifters ovvero una matrice di multiplexer che, opportunamente pilotati da un decoder, sono in grado di fare lo shift dell'operando che si trova al loro ingresso di un numero arbitrario di posizioni in un solo ciclo di clock. Una seconda variante invece prevede l'uso di registri a scorrimento PISO ovvero registri che permettono il caricamento del dato in maniera parallela (in un solo ciclo di clock) ma necessitano di un ulteriore ciclo di clock per ogni shift di una posizione che si vuole fare. Quest'ultima soluzione è sicuramente più lenta (in termini di numero di cicli di clock) rispetto alla prima e richiede una macchina a stati finiti più complessa ma presenterà un ingombro minore in termini di area.

Nel progetto sono state realizzate queste due ultime soluzioni. I file relativi al CORDIC implementato con barrel shifters si trovano nella sottocartella .\codici\VHDL\CORDIC barrel mentre i file relativi al CORDIC implementato con registri a scorrimento si trovano .\codici\VHDL\CORDIC piso.

4. Dimensionamento

Per entrambe le implementazioni i criteri di dimensionamento sono gli stessi. Le parole di ingresso, cioè l'ascissa e l'ordinata, sono su 14 bit e si è scelto la rappresentazione in complemento a 2 con valore dell'LSB pari a 2^{-7} quindi i valori possibili per gli ingressi possono variare da -64 a

63.9921875. Per quanto riguarda il modulo esso è rappresentato su 16 bit in codice binario con LSB pari a 2^{-8} e quindi può variare da 0 a 255,99609375 mentre la fase ha LSB pari a $2\pi/2^{16}$.

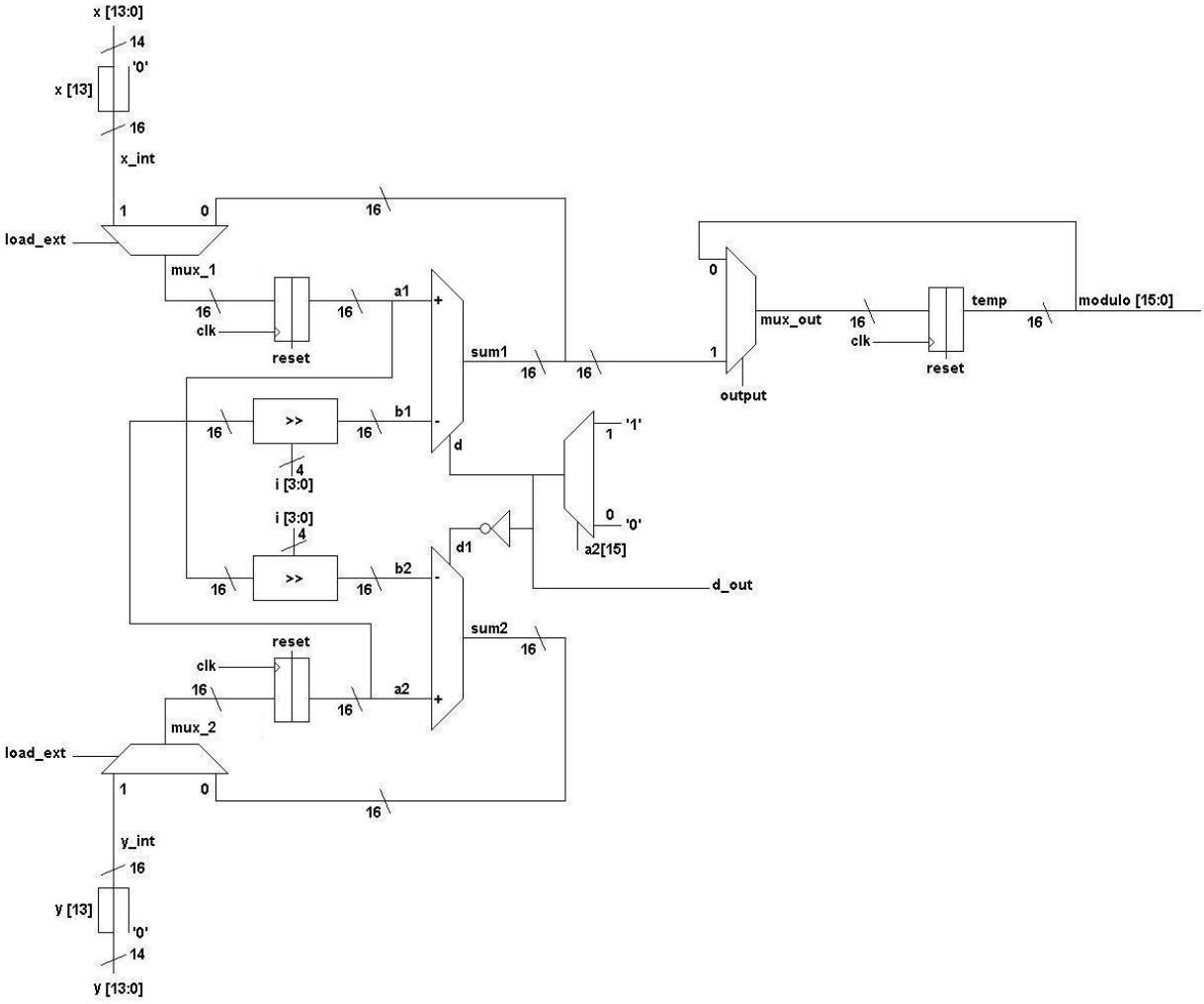
5. Implementazione in VHDL

Entrambe le implementazioni sono fatte nel seguente modo: vi sono due blocchi che implementano, rispettivamente, le prime due equazioni (per il calcolo del modulo) e la terza (per il calcolo della fase). Infine c'è un terzo blocco che è la macchina a stati finiti che coordina gli altri. Poiché una delle 2 implementazioni fa uso di 2 barrel shifters mentre l'altra fa uso di registri a scorrimento la macchina a stati finiti e i segnali di ingresso dei vari blocchi differiscono tra le due implementazioni. In realtà vi sono piccole differenze anche tra i 2 blocchi di ciascuna implementazione.

Vediamo prima la soluzione con barrel shifters.

5.1 Implementazione del calcolo del modulo

L'implementazione in VHDL del circuito è nel file CORDIC_A.vhd. Lo schema del circuito è il seguente:



Sia l'ascissa che l'ordinata vengono mappate su 16 bit a partire dai 14 iniziali. Dei 2 bit aggiuntivi uno è stato usato come LSB, così da aumentare la precisione mentre l'altro è stato usato come MSB così da aumentare la dinamica. Quando load_ext è pari a 1, al ciclo di clock successivo, si carica il dato in ciascuno dei 2 registri dopo di che si pone load_ext a 0 in maniera tale da riportare in

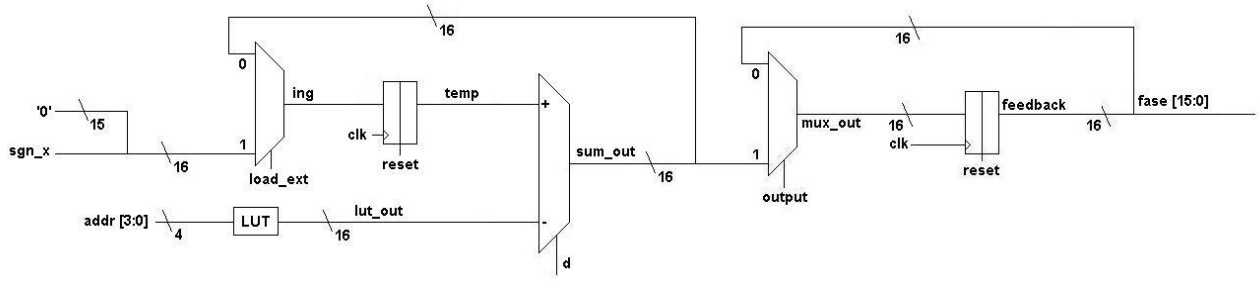
ingresso al registro temporaneo l'uscita del blocco sommatore (come richiesto dalle equazioni). Ad ogni ciclo di clock si modifica il valore del bus "i" che pilota i barrel shifter e si immagazzina nei registri temporanei i risultati.
Durante le operazioni di calcolo intermedie il registro di uscita è opportunamente retroazionato ovvero si riporta la sua uscita in ingresso, cosicché rimanga stabile al valore corrente; Quando invece si è arrivati a fine calcolo il multiplexer in ingresso al registro presenta, all'ingresso vero e proprio del registro stesso, il valore del modulo appena calcolato per poi ritornare (al ciclo di clock successivo) nella posizione di mantenimento. Il risultato di ciò è che il modulo rimane stabile all'ultimo valore corretto calcolato.
L'algoritmo così implementato prevede di fornire in ingresso i dati in complemento a 2 e fornirà in uscita i dati in base 2 senza alcuna codifica. Bisogna però tenere conto del fatto che il valore del modulo così ottenuto sono maggiorati di un fattore 1.647.

5.1.1 Implementazione del barrel shifter

Un barrel shifter è una logica puramente combinatoria formata da una matrice di multiplexer collegati tra di loro e opportunamente pilotati da un decoder. Questo dispositivo permette, con un solo ciclo di clock, di compiere operazioni di shift di un numero generico di posizioni del dato in ingresso. Tale componente è stato descritto secondo la metodologia behavioral nel file barrel_shifter.vhd

5.2 Implementazione del calcolo dell'angolo

L'implementazione in VHDL del circuito è nel file CORDIC_B.vhd. Lo schema del circuito è il seguente:



Il circuito calcola la fase mappando l'intervallo tra 0 e 2π su 16 bit; I valori di questi angoli sono memorizzati nella LUT e sono stati calcolati grazie allo script MATLAB LUT.m. Quando si caricano i dati dall'esterno viene fornito lo sfasamento iniziale che può essere 0 o π a seconda che l'ascissa sia positiva o negativa ma dal momento che π codificato su 16 bit corrisponde alla parola avente tutti bit a 0 tranne l'MSB (che è ad 1) per ricavare lo sfasamento iniziale è sufficiente che l'MSB della parola di ingresso sia pari all'MSB dell'ascissa.
Una volta caricato il dato iniziale, similmente a prima, il multiplexer del registro di ingresso presenta, all'ingresso del registro stesso, l'uscita del sommatore realizzando così un accumulatore di fase. Ad ogni ciclo di clock, quindi, l'accumulatore di fase prende in ingresso i valori codificati nella LUT la quale è pilotata dallo stesso segnale usato per comandare i barrel shifter e li somma o sottrae al valore precedentemente calcolato. I dati nella LUT altro non sono che la codifica su 16 bit dei valori di $\text{atan}(2^{-i})$, dove i è l'ingresso della LUT.
Similmente a quanto fatto per il modulo, l'uscita è pilotata da un registro opportunamente retroazionato: durante le operazioni di calcolo intermedie si riporta in ingresso an registro l'uscita del registro stesso cosicché rimanga stabile al valore corrente; Quando invece si è arrivati a fine calcolo il multiplexer in ingresso al registro fa acquisire il valore della fase appena calcolata per poi ritornare (al ciclo di clock successivo) nella posizione di mantenimento. Il risultato di ciò è che la fase rimane stabile all'ultimo valore corretto calcolato.

5.3 Implementazione della macchina a stati finiti

La macchina a stati finiti è descritta nel file MSF.vhd. E' una macchina di Moore ed è stata descritta in maniera behavioral, lasciando quindi il compito al sintetizzatore di implementare la circuiteria. Lo scopo del blocco è quello di tenere traccia del numero di iterazioni e produrre i segnali opportuni, cioè i segnali del bus che comanda i barrel shifters, il reset dei vari registri, il segnale per il caricamento dei dati dall'esterno e quello di presentazione dei dati in uscita.
Vi sono 3 processi: il primo è sensibile al segnale di clock e si occupa di far evolvere lo stato della macchina; Il secondo è attivato dal cambio dello stato corrente e si occupa di pilotare il bus dei barrel shifter e stabilisce quale sarà il prossimo stato che dovrà assumere la macchina. Infine vi è il terzo processo che, sulla base dello stato corrente, imposta il valore delle altre uscite.

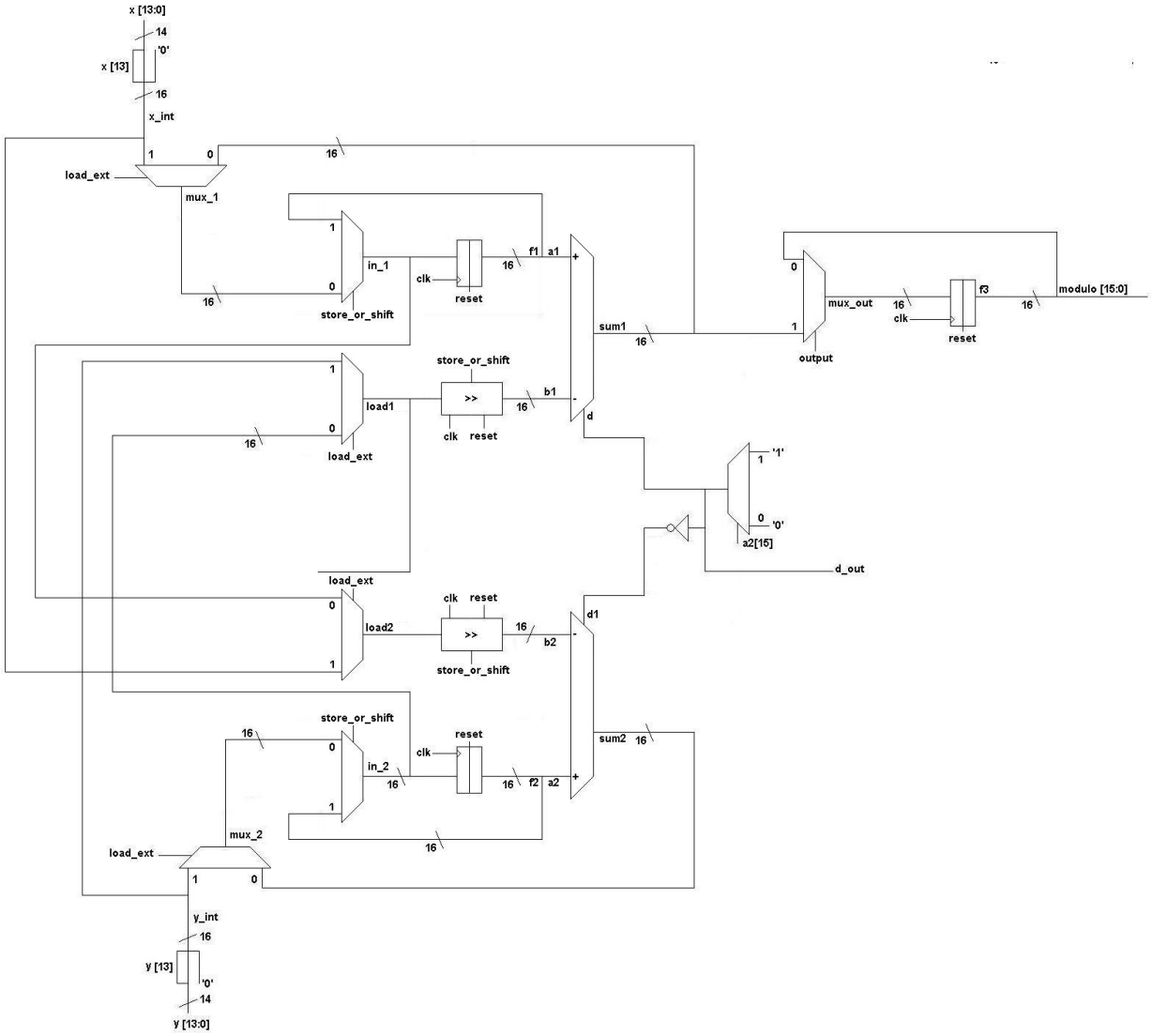
5.4 Wrapper

L'algoritmo così strutturato converge ai valori corretti di modulo ed angolo solo se il vettore appartiene al semipiano delle ascisse positive. Per estendere il funzionamento del dispositivo a tutti e quattro i quadranti si è creato un wrapper, implementato nel file CORDIC_wrapper.VHD cioè un'interfaccia di logica combinatoria tra i segnali di ingresso e gli ingressi veri e propri del CORDIC. Il wrapper è formato da 2 full adder\subtractor che sommano o sottraggono alla parola formata da tutti zeri l'ascissa e l'ordinata, così da avere sempre in uscita un valore positivo. Per stabilire se gli ingressi devono essere sommati o sottratti si usa l'MSB dell'ascissa (lo stesso usato per stabilire la fase iniziale nel calcolo dell'arcotangente). In uscita da questo blocco di logica combinatoria si avranno quindi 2 coordinate riferite sempre al semipiano positivo e che date in ingresso al resto della logica daranno in uscita il modulo e la fase corrispondenti al vettore originario.

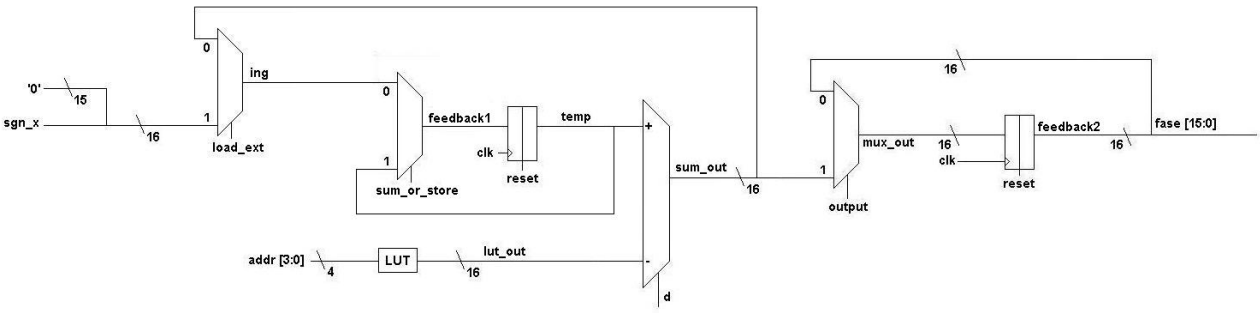
6. Implementazione del CORDIC con registri a scorrimento

Questa nuova architettura risulta più complessa. I barrel shifters sono stati sostituiti da 2 registri a scorrimento PISO che (in maniera sincrona con il clock) caricano i dati in maniera parallela o fanno lo shift verso destra di una posizione. In conseguenza di ciò tutti gli altri registri, che prima erano dei semplici flip flop D, sono stati sostituiti da dei registri di memorizzazione veri e propri, cioè dei registri in grado di mantenere il dato o di riceverne uno nuovo. La differenza più importante però è nella macchina a stati finiti che infatti è più complessa della precedente, pur essendo sempre una macchina di Moore.
Similmente a prima la macchina a stati finiti è stata implementata in maniera behavioral ma anziché cambiare di stato ad ogni ciclo di clock la macchina deve contare un certo numero di cicli corrispondenti al numero di shift verso destra che si vuole far fare agli operandi e contestualmente a ciò deve pilotare opportunamente i vari registri: i registri a scorrimento dovranno effettuare le operazioni di shift verso destra mentre i registri di memorizzazione dovranno conservare il dato (che altrimenti verrebbe modificato ad ogni ciclo di clock) per poi, solo alla fine riacquisire il risultato corretto.

Lo schema relativo alla nuova implementazione per il caloclo del modulo è il seguente:



Mentre per la fase lo schema si modifica nel seguente modo:

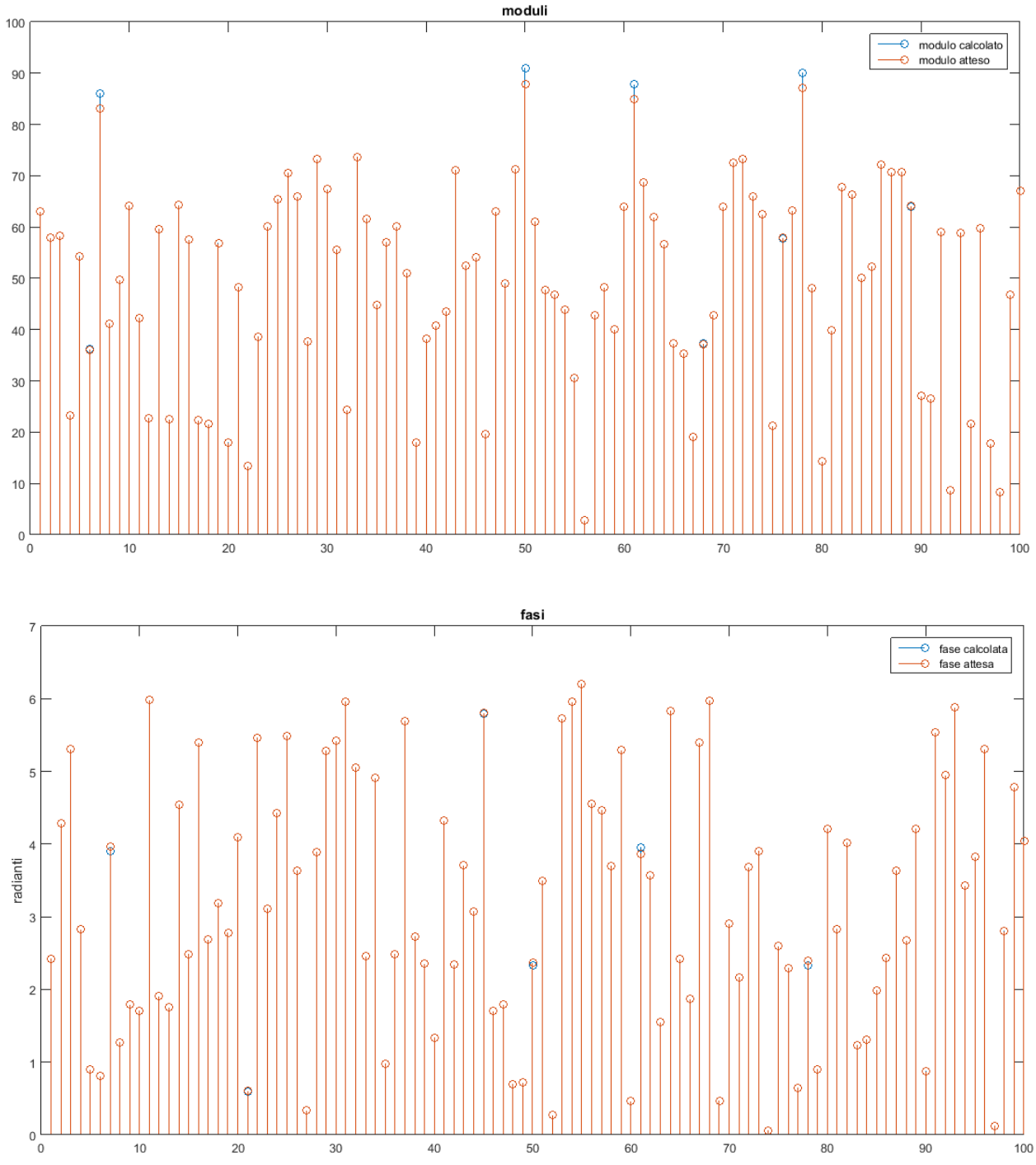


I nomi dei file della descrizione in VHDL dei due circuiti sono rimasti invariati (cioè rispettivamente CORDIC_A.vhd e CORDIC_B.vhd ma si trovano nella cartella CORDIC piso.

6. Testbench

Il testbench è stato implementato nel file td_cordic.vhd. Dalle simulazioni condotte si può già apprezzare la differenza tra i 2 dispositivi. Supponendo di avere un clock di periodo pari a 20ns si è ricavato che nel primo caso si ha un throughtput di 2490 valori al secondo e una latenza di 18 cicli

di clock nel secondo caso invece si ha un throughtput di 364 valori al secondo e una latenza di 138 cicli di clock.
Per entrambe i dispositivi si è usato lo stesso testbench che è implementato nel file td_cordic.vhd. Il file TB.m crea un file formato testo dove sono codificate in complemento a 2 le coordinate di 100 vettori generati casualmente. Per acquisire i dati da file e riportare i risultati su un nuovo file di testo si è usato la libreria IEEE.std_logic_textio. In seguito, sempre grazie all'ausilio di MATLAB si è fatto un confronto tra i moduli e fasi dei vettori effettivamente mandati in ingresso (pervia riconversione in base 10) e i valori di modulo e fase ricavati dal CORDIC. A seguito della simulazione si è osservato che l'errore assoluto medio per il modulo è 0.122779 mentre quello della fase è -0.000781.

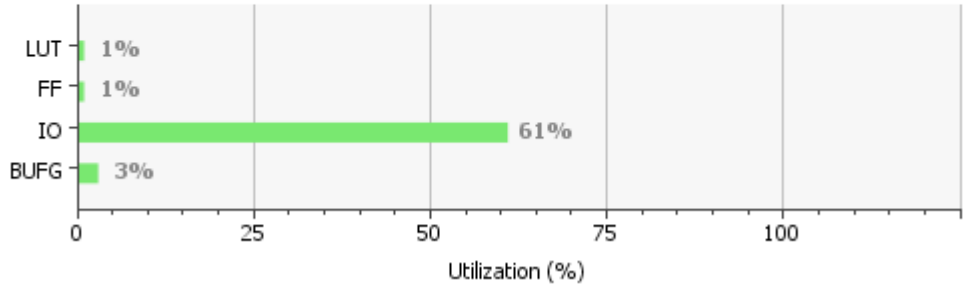


Istogrammi rappresentanti i valori veri (in blu) e quelli calcolati dal CORDIC (in arancione) per una simulazione.

7. Sintesi

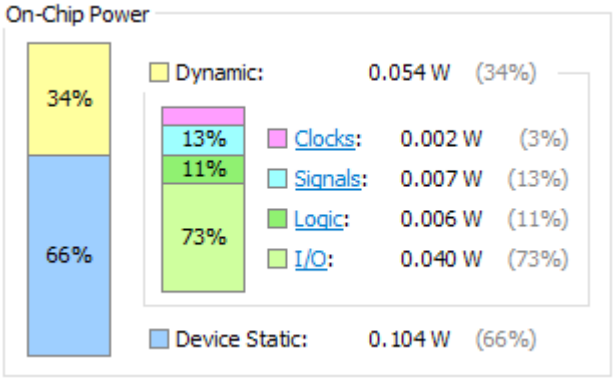
Infine si passa alla sintesi. Entrambi i CORDIC sono stati sintetizzati secondo la strategia standard di sintesi di VIVADO e con il clock avente duty cycle al 50%. Per il CORDIC con barrel shifter si hanno i seguenti dati:

Resource	Utilization	Available	Utilization %
LUT	221	17600	1.26
FF	85	35200	0.24
IO	61	100	61.00
BUFG	1	32	3.13



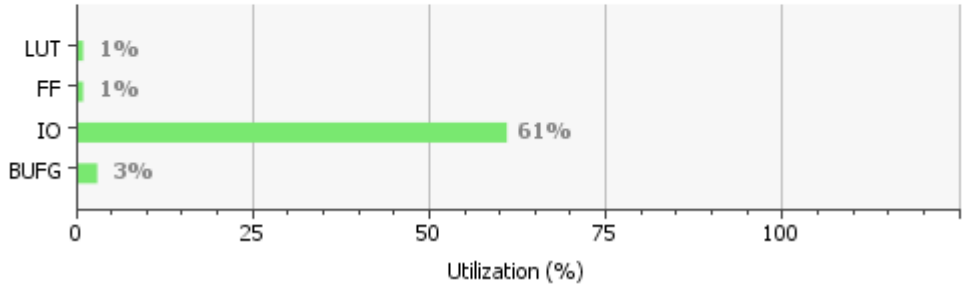
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.158 W**
Junction Temperature:	**26,8 ℃**
Thermal Margin:	58,2 ℃ (5,0 W)
Effective ϑJA:	11,5 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low



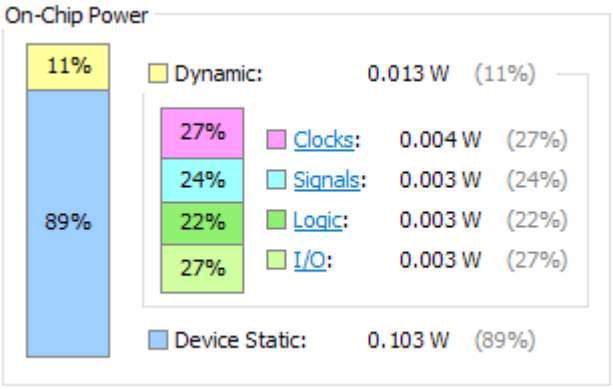
Mentre per quanto riguarda il CORDIC con registri a scorrimento

Resource	Utilization	Available	Utilization %
LUT	207	17600	1.18
FF	121	35200	0.34
IO	61	100	61.00
BUFG	1	32	3.13



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

Total On-Chip Power:	**0.116 W**
Junction Temperature:	**26,3 °C**
Thermal Margin:	58,7 °C (5,0 W)
Effective ϑJA:	11,5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low



La massima frequenza di clock per il CORDIC implementato con barrel shifter è 190.042 MHz mentre per quello implementato con registri PISO é 260.010 MHz:

Dai report di utilizzazione si evince che l'algoritmo di sintesi standard di vivado ha sintetizzato i due dispositivi in maniera molto simile anche se nel secondo caso si ha un maggiore uso di flip flop e un minor numero di LUT rispetto al primo.

I report di consumo di potenza sono stati fatti per le massime frequenze raggiunte. Come prevedibile nel primo caso si ha un consumo di potenza maggiore ed, in termini assoluti, il consumo è imputabile alla sola potenza dinamica dal momento che la potenza statica è pressoché la stessa in tutti e 2 i casi.