CORDIC

1. Introduzione

CORDIC è stato concepito nel 1956 da Jack E. Volder presso il dipartimento di Aeroelectronics di Convair, per necessità di sostituire il risolutore analogico nel computer di navigazione del bombardiere B-58 con una soluzione digitale in tempo reale più accurata e performante.
CORDIC (**CO**ordinate **R**otation **DI**gital **C**omputer) è un semplice ed efficiente algoritmo iterativo per il calcolo di funzioni trigonometriche. Questo algoritmo appartiene alla classe di algoritmi nota come shift-and-add, visto che vengono impiegati in contesti nei quali non sono disponibili moltiplicatori, riuscendo ad ottenere il risultato sfruttando solamente operazioni di somma, sottrazione, shift e table lookup.

Generalmente l'algoritmo CORDIC permette il calcolo di funzioni trigonometriche, quali seno e coseno, operando nella cosiddetta Rotation mode. Un particolare caso applicativo della Rotation mode è la Vectoring mode, nella quale è possibile calcolare modulo e fase di un vettore note le sue coordinate cartesiane a patto che si trovi nel primo o quarto quadrante.
L'algoritmo si presenta come iterativo, garantendo la convergenza ai valori desiderati secondo le seguenti equazioni:

$$\begin{cases} X_{i+1} = X_i + Y_i \cdot 2^{-i} \cdot d_i \\ Y_{i+1} = Y_i - X_i \cdot 2^{-i} \cdot d_i \\ Z_{i+1} = Z_i + \arctan(2^{-i}) \cdot d_i \\ d_i = sign(Y_i) \end{cases}$$

Dopo un numero n di passi, le equazioni convergono a:

$$X_n = A_n \cdot \sqrt{{X_0}^2 + {Y_0}^2}$$

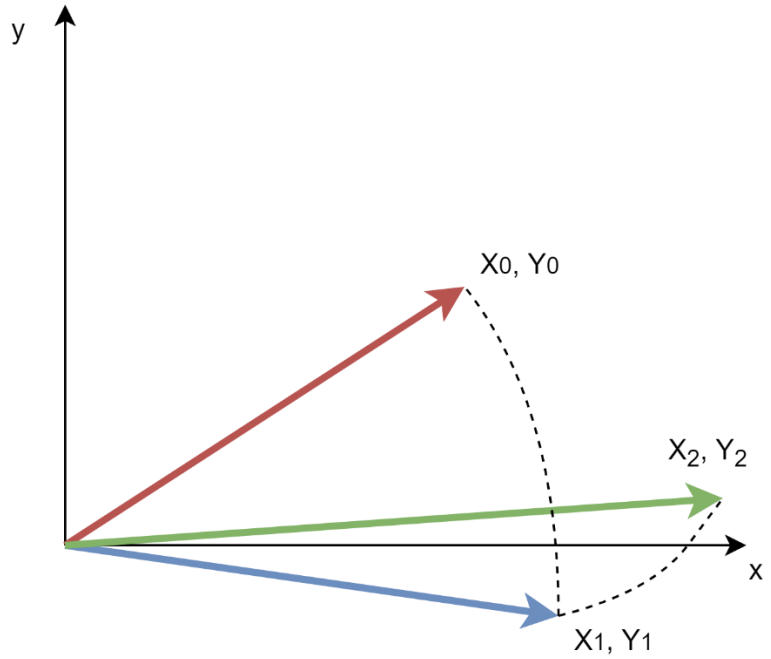
$$A_n = \prod_1^n \sqrt{1 + 2^{-i}}$$

$$Y_n = 0$$

$$Z_n = Z_0 + \arctan\left(\frac{Y_0}{X_0}\right)$$

Una semplice spiegazione dell'algoritmo è la seguente: il vettore iniziale, descritto dalle coordinate cartesiane (X_0, Y_0), viene fatto ruotare ad ogni iterazione, fino a "stenderlo" lungo l'asse x. Queste rotazioni sono descritte da $\arctan(2^{-i})$ e sono sempre più piccole ad ogni passo. La direzione della rotazione è data da d_i. Quindi ad ogni iterazione si ha la diminuzione del modulo di Y_i mentre il valore di X_i aumenta. Questo incremento porterebbe ad un errata valutazione del modulo, ragione per cui il valore verrà corretto con la costante A_n, che rappresenta l'incremento totale, una volta terminato l'algoritmo. Questa possibilità, permette di correggere il valore del modulo alla fine dell'algoritmo anziché ad ogni passo, permettendo di effettuare una sola moltiplicazione invece che n. Il valore della fase θ del vettore corrisponde a Z_n a patto che l'algoritmo inizi con $Z_0 = 0$, sommando algebricamente tutte le rotazioni

effettuate nelle varie iterazioni, calcolando così l'effettiva rotazione da compiere per adagiare il vettore sull'asse delle x rispetto alla posizione di partenza.



Rappresentazione del progressivo aumento di X_i e della distensione sull'asse X

L'algoritmo per come è strutturato permette di evitare l'utilizzo di moltiplicatori, se non per la correzione finale del valore del modulo. Vengono quindi usati shift register per implementare le moltiplicazioni per 2^{-i}. La moltiplicazione per il valore d_i, cioè un eventuale cambio di segno, è semplice grazie alla rappresentazione dei numeri utilizzata. Inoltre, i valori di $\arctan(2^{-i})$ vengono ottenuti tramite operazione di lookup da una tabella, senza la necessità di essere calcolati a runtime, visto che sono sempre gli stessi per le n iterazioni.

Si presenta subito la necessità di individuare il numero ottimale di iterazioni n da compiere in questa specifica implementazione dell'algoritmo, essendo impossibile compiere un numero infinito di iterazione su un dispositivo reale. Il numero di iterazioni deve essere individuato tenendo conto della convergenza delle equazioni, cercando un valore tale da ottenere una precisione soddisfacente. Di seguito si riporta una semplice analisi dei parametri e della loro convergenza eseguita con Excel:

Iteration	A_n	K_n
0	1,4142135624	0,7071067812
1	1,5811388301	0,6324555320
2	1,6298006013	0,6135719911
3	1,6424840658	0,6088339125
4	1,6456889158	0,6076482563
5	1,6464922787	0,6073517701
6	1,6466932543	0,6072776441
7	1,6467435066	0,6072591123
8	1,6467560702	0,6072544793
9	1,6467592111	0,6072533211
10	1,6467599964	0,6072530315
11	1,6467601927	0,6072529591

Per quanto riguarda il fattore A_n si ha una convergenza piuttosto rapida. È possibile notare che già dall'ottava iterazione in poi le variazioni sono dell'ordine di grandezza di 10^{-5}.
Si è scelto di considerare le prime 8 iterazioni per poter utilizzare 3 bit per il contatore che gestirà il numero di iterazioni all'interno del circuito.

In conclusione, per ottenere il modulo del vettore ρ, basterà quindi calcolare $\rho = \frac{X_n}{A_n} = K_n \cdot X_n$

12	1,6467602418	0,6072529410

I valori che verranno utilizzati all'interno del dispositivo, sono da interpretarsi nel mondo esterno come numeri reali, tuttavia l'aritmetica finita impone alcune limitazioni. Per questa implementazione si è deciso di gestire internamente i numeri come degli interi rappresentati in complemento a due, tuttavia l'utilizzatore esterno può associare un significato differente ai valori forniti in ingresso. È stato scelto di utilizzare la rappresentazione con notazione a virgola fissa (fixed point) su 16 bit. In particolare:

- $N = 16$ Numero di bit su cui rappresentare i valori
- $S = 8$ Fattore di scala

Il valore di S è scelto in modo tale che i primi n valori di $\arctan(2^{-i})$ possano essere rappresentati con una precisione sufficiente.

Scelti questi due valori è possibile identificare la risoluzione $R = \frac{1}{2^S} = \frac{1}{2^8} = 0.00390625$. Questo significa che la variazione di un bit rappresenta una variazione di ± 0.00390625 del valore.

Considerando i valori interi rappresentabili su $N = 16$ bit, cioè $[-32768, +32767]$, presa come risoluzione $R = 0.00390625$ si ha che il nuovo intervallo rappresentabile diventa $[-128,\ 127{,}99609375]$.

i.e.

Se l'utente ha intenzione di utilizzare $X_0 = 6,\ Y_0 = 9$ dovrà fornire in ingresso al CORDIC i corrispondenti valori in fixed point cioè $\left\lfloor X_0 \cdot \frac{1}{R} \right\rceil, \left\lfloor Y_0 \cdot \frac{1}{R} \right\rceil$. Dove l'operatore $\lfloor\ \ \rceil$ corrisponde alla parte intera più vicina.

Tuttavia, non tutti i valori rappresentabili $[-128,\ 127{,}99609375]$ possono essere forniti in ingresso, in quanto alcuni potrebbero generare un overflow durante il processo di calcolo iterativo. La soglia massima individuata suggerisce di utilizzare valori $-90 < 0 < 90$. Tuttavia, per aggirare il problema si ricorda che vale la seguente relazione:

$$|kX_0, kY_0| = k \cdot |X_0, Y_0|, \forall k \in R$$

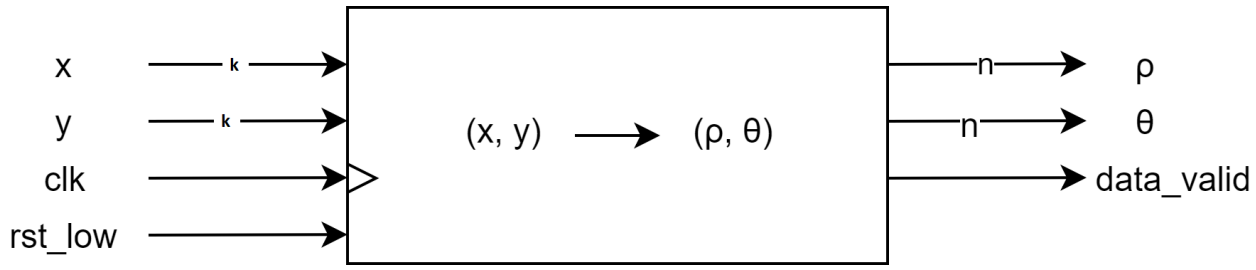
Inoltre, per i valori di ingresso non possono essere forniti $X_0 < 0$ in quanto i valori di $arctan(\ \ )$, caricati nella LUT, non sarebbero consistenti in quanto l'angolo sarebbe fuori dal range $-\frac{\pi}{2} < 0 < \frac{\pi}{2}$ in cui è definita la funzione $arctan(\ \ )$.

Per quanto riguarda le uscite:

- Per ottenere il vero valore del modulo è necessario moltiplicare ρ per la risoluzione al quadrato: $Modulo = \rho R^2$
- Per ottenere la vera fase è necessario moltiplicare θ per la risoluzione: $Fase = \theta R$

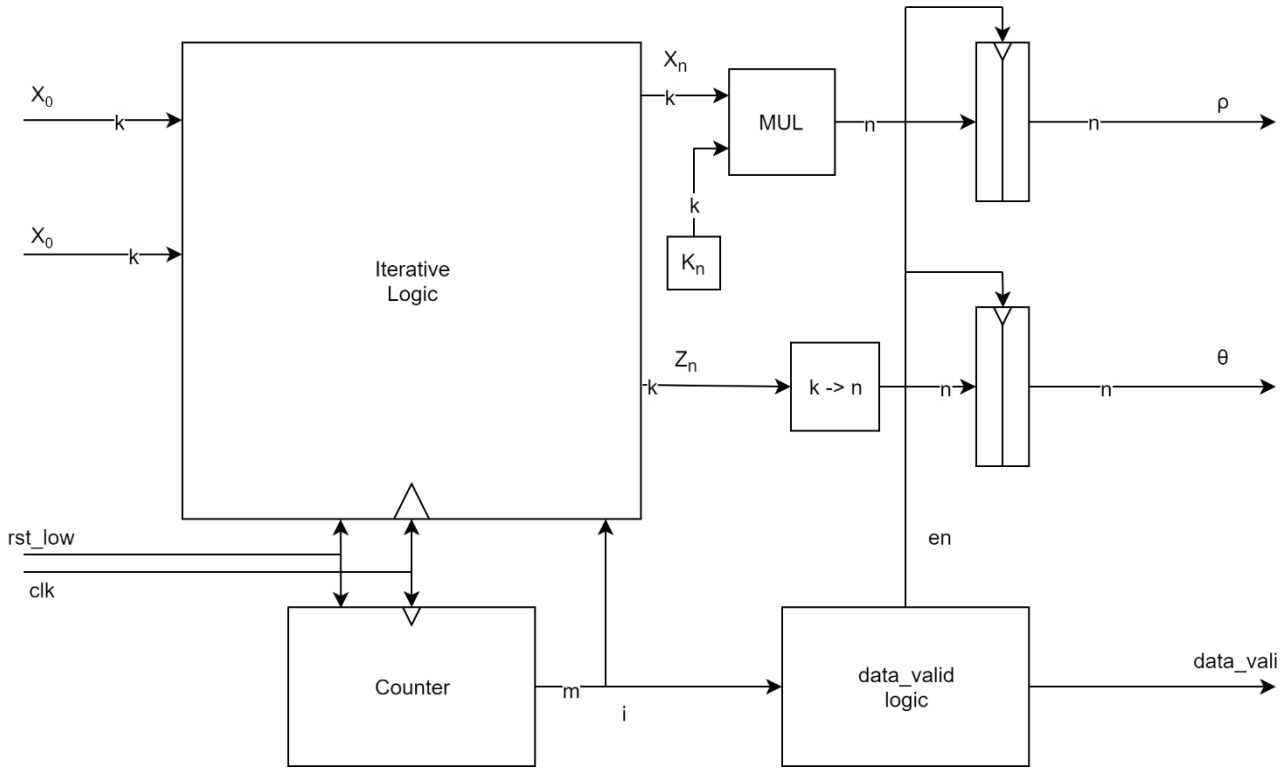
2. Implementazione

Il circuito si propone di calcolare modulo e fase di un vettore, partendo dalle relative coordinate cartesiane. Lo schema semplificato del circuito è il seguente:



Schema a blocchi CORDIC

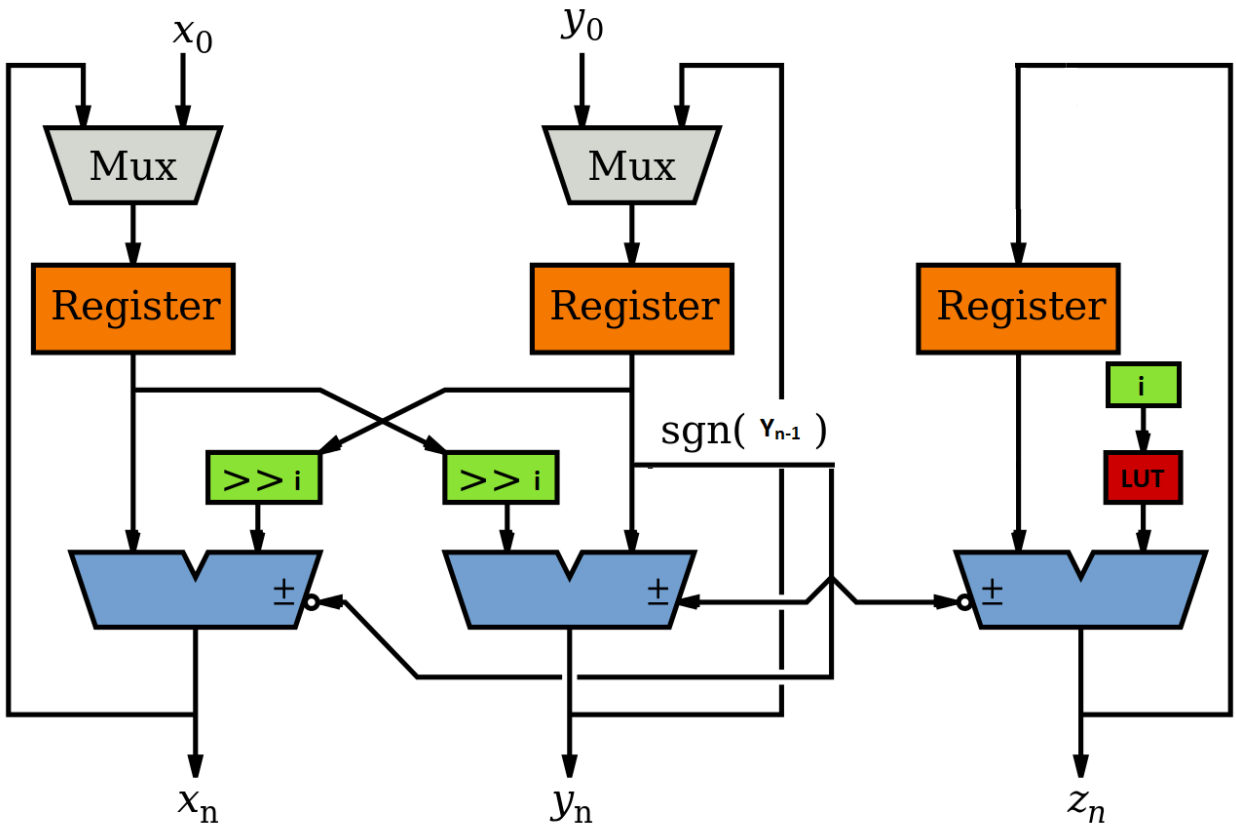
Il circuito viene utilizzato dando in ingresso i valori X_0, Y_0 su 16 bit ed il comando di reset, una volta rilasciato quest'ultimo il calcolo iterativo avrà inizio, terminando con la transizione 0 → 1 dell'uscita $data\_valid$. Il risultato fornito in uscita è da ritenersi corretto dal momento della transizione.



Schema a blocchi interno del CORDIC

Nello specifico la gestione interna è la seguente: un circuito contatore tiene conto dell'iterazione corrente, e informa la logica che effettua le operazioni iterative. L'iterazione corrente è indicata dal segnale i su 3 bit,

in quanto abbiamo deciso di fermarci all'ottava iterazione (quindi n = 7). Il valore di $\arctan(2^{-i})$ e il numero di bit da shiftare dipendono proprio da i, cioè dall'iterazione corrente. Quando il valore di i coincide con n, cioè quando l'algoritmo deve terminare dopo l'ottava iterazione, il pin $data\_valid$ viene settato, i registri di uscita verranno abilitati per un ciclo di clock e memorizzeranno il risultato ottenuto. Per la parte riguardante il calcolo del modulo, è necessario ridurre il valore di X_n della costante K_n, operazione effettuata grazie al moltiplicatore, la cui uscita sarà su 32 bit, dato che X_n e K_n sono entrambi su 16 bit. Per quanto riguarda la fase è sufficiente una semplice estensione di campo, in modo da avere entrambe le uscite del CORDIC su 32 bit. Modulo e fase rimangono sempre validi in uscita, dopo il segnale di $data\_valid$ (quindi dopo l'ottava iterazione), in quanto i registri posti prima delle uscite non saranno più abilitati, a meno che non si effettui un reset per iniziare un nuovo calcolo.



Schema di funzionamento per il calcolo iterativo del CORDIC

Per quanto riguarda la logica iterativa, il nostro circuito contiene tre blocchi distinti che si occupano di calcolare i valori di $X_{i+1}, Y_{i+1}, Z_{i+1}$ necessari ad ogni iterazione. In ingresso vengono forniti i valori X_0, Y_0, Z_0, nel nostro caso sarà $Z_0 = 0$.

Nello specifico i calcoli sono effettuati come segue:

- X_{i+1}: Multiplexer, pilotato da rst_low, che fa passare X_0 quando viene dato il comando di reset, altrimenti lascia passare X_i. Questo valore viene memorizzato da un registro che ha la particolarità di memorizzare il dato in ingresso anche al segnale di reset. Il valore del registro, che corrisponde a X_i, viene mandato ad un circuito ADD/SUB che ha il compito di sommarlo come indicato nella formula del CORDIC. Inoltre, X_i viene mandato in uscita per fornirlo al blocco relativo al calcolo di

Y_{i+1}. L'altro componente fornito in ingresso al blocco attuale è il valore di Y_i dal quale viene prelevato il segno e successivamente shiftato, ottenendo così la variabile di comando e il secondo addendo per il circuito ADD/SUB. L'uscita di questo circuito, fornita dall'ADD/SUB, è X_{i+1}.

- Y_{i+1}: Multiplexer, pilotato da rst_low, che fa passare Y_0 quando viene dato il comando di reset, altrimenti lascia passare Y_i. Questo valore viene memorizzato da un registro che ha la particolarità di memorizzare il dato in ingresso anche al segnale di reset. Il valore del registro, che corrisponde a Y_i, viene mandato ad un circuito ADD/SUB che ha il compito di sommarlo come indicato nella formula del CORDIC. Inoltre, Y_i viene mandato in uscita per fornirlo al blocco relativo al calcolo di X_{i+1}, mentre il solo segno di Y_i viene fornito al blocco relativo al calcolo di Z_{i+1}. Il segno di Y_i rappresenta inoltre la variabile di comando del circuito ADD/SUB. L'altro componente fornito in ingresso al blocco attuale è il valore di X_i il quale viene shiftato, ottenendo così il secondo addendo del circuito ADD/SUB. L'uscita di questo circuito, fornita dall'ADD/SUB, è Y_{i+1}.

- Z_{i+1}: Un registro mantiene il valore di Z_i. In particolare, all'iterazione 0 quando si deve fornire $Z_0 = 0$, non si fa altro che fornire il comando di rst_low al registro, il quale imposterà il suo contenuto a 0. L'uscita di tale registro costituisce Z_i ed è il primo ingresso del circuito ADD/SUB. Il secondo ingresso è la $i - esima$ entrata della **LUT** (*look-up table)* ovvero $\arctan(2^{-i})$, dove il valore i rappresenta l'iterazione corrente ed è ricevuto in ingresso dal contatore. La LUT contiene n valori, ognuno pari ad $\arctan(2^{-i})$, con $i \in (0, n-1)$. Il segno di Y_i viene ricevuto in ingresso e costituisce la variabile di comando per lo ADD/SUB. L'uscita di questo circuito, fornita dall'ADD/SUB, è Z_{i+1}.

Dopo l'ottava iterazione otterremo in uscita da questa logica iterativa i valori i X_7, Y_7, Z_7. Di questi ci interessano solo X_7 e Z_7 (ovvero X_n e Z_n) i quali saranno utilizzati per ottenere il risultato finale di modulo e fase, come spiegato precedentemente.

3. Test plan

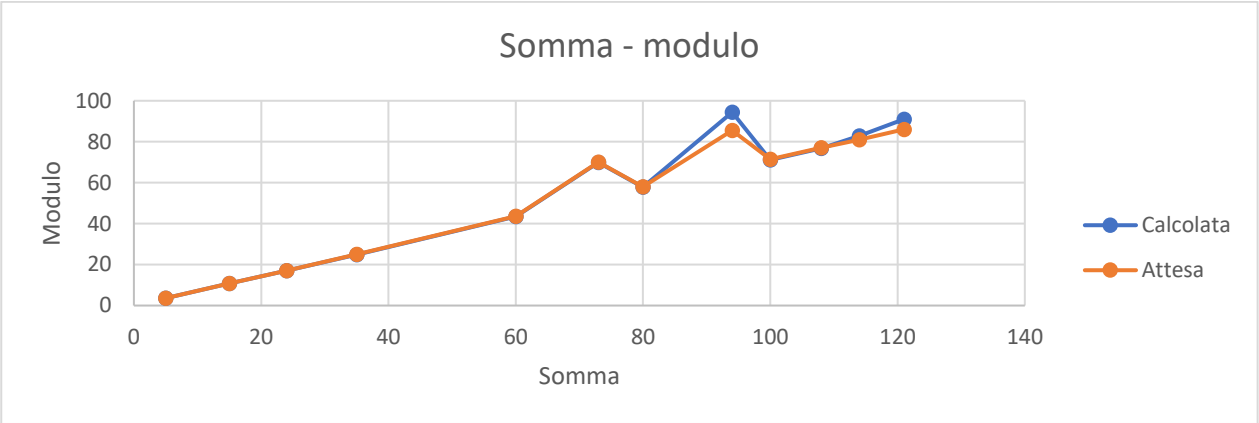
Per il test plan abbiamo seguito un modello tipico, caratterizzato da una prima fase di *unit testing* in cui abbiamo testato il funzionamento dei singoli componenti, cioè i blocchi per il calcolo di $X_{i+1}, Y_{i+1}, Z_{i+1}$. Dopo averne verificato la correttezza abbiamo fatto il cosiddetto *integration test* nel quale si verifica il funzionamento dell'intero sistema con tutti i suoi moduli.

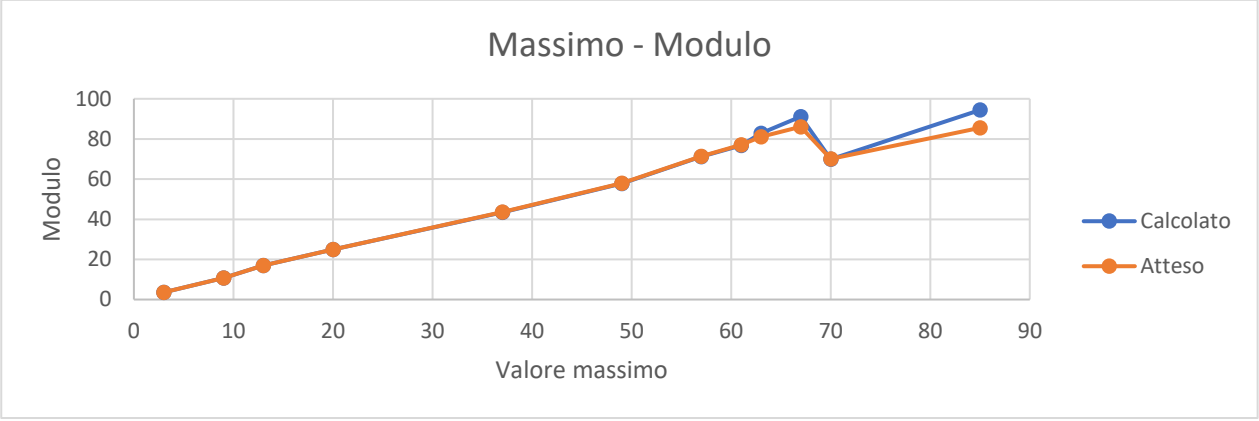
Come precedentemente accennato questa fase di test ha permesso di scoprire la limitazione da imporre agli ingressi $(-90 < 0 < 90)$ per evitare un overflow nel calcolo del modulo. Di seguito riportiamo una porzione di test effettuati per stabilire la precisione del dispositivo:

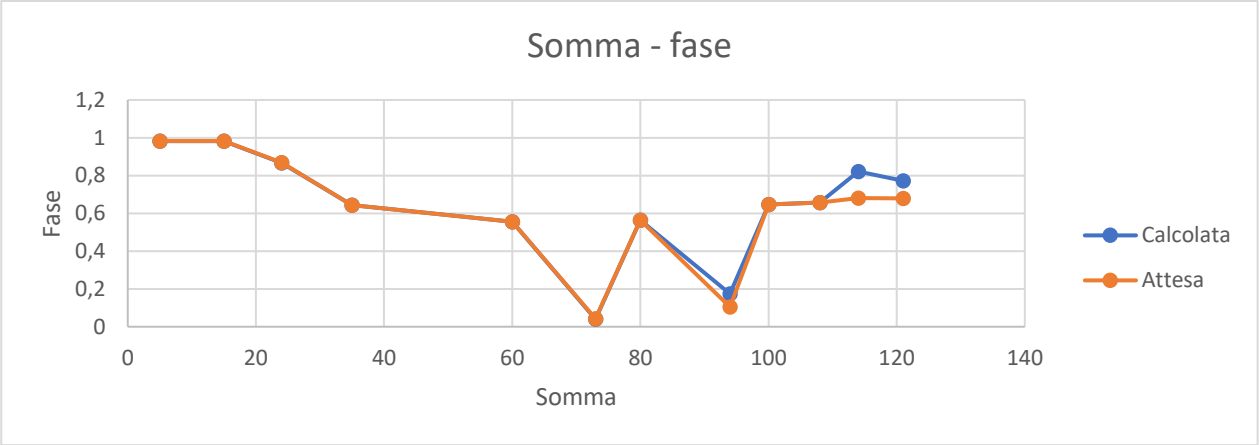
				Errore				Errore		
Somma	**singoloMax**	**X0**	**Y0**	**Modulo**	**Atteso**	**Err**		**Fase**	**Atteso**	**Err**
5	3	2	3	3,590987	3,605551	-0,01456		0,98208	0,982794	-0,00071
15	9	6	9	10,77768	10,81665	-0,03897		0,98208	0,982794	-0,00071
24	13	11	13	16,96918	17,02939	-0,0602		0,867673	0,868539	-0,00087
35	20	20	15	24,91761	25	-0,08239		0,643751	0,643501	0,00025

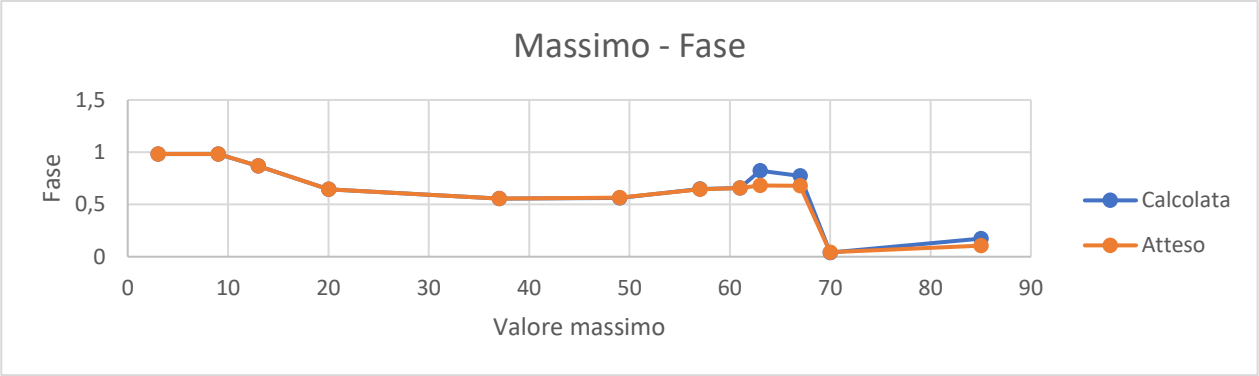
60	37	37	23	43,41348	43,56604	-0,15256		0,555292	0,556166	-0,00087
73	70	70	3	69,82285	70,06426	-0,24141		0,039723	0,042831	-0,00311
80	49	49	31	57,78251	57,98276	-0,20025		0,563209	0,564084	-0,00088
94	85	85	9	94,39232	85,47514	8,917177		0,173801	0,105489	0,068312
100	57	57	43	71,15646	71,40028	-0,24382		0,646552	0,646302	0,00025
108	61	61	47	76,7446	77,00649	-0,26189		0,656737	0,656487	0,00025
114	63	63	51	82,86411	81,05554	1,808569		0,821399	0,680521	0,140878
121	67	67	54	91,0438	86,05231	4,991492		0,772513	0,678371	0,094142

Per diverse configurazioni di input X_0, Y_0 abbiamo ottenuto i seguenti andamenti di modulo e fase rispetto alla somma di X_0, Y_0 (per vedere se il fatto di averli entrambi troppo alti fosse un problema) e rispetto al $max\{X_0, Y_0\}$ (per vedere se il fatto di averne anche uno solo dei due troppo alto fosse un problema)









Da questi grafici abbiamo intuito che viene introdotto un errore non trascurabile quando X_0, Y_0 diventano troppo grandi, in particolare quando la loro somma o il massimo dei due è troppo alto. Per questo bisogna cercare di non inserire valori troppo alti, in particolare:

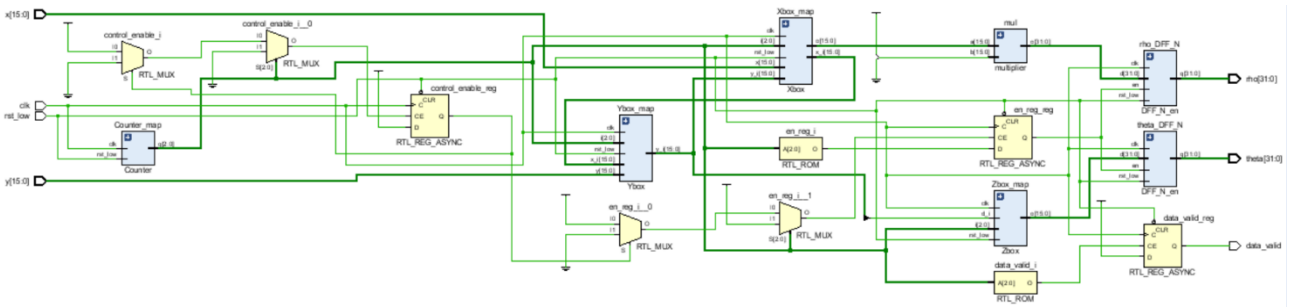
- si osserva che $X_0 + Y_0 < R$ con R che dipende dal $max(X_0, Y_0)$. Nel senso che se la somma è alta può dipendere semplicemente dal fatto che uno tra X_0 e Y_0 sia molto alto
- si osserva che $max(X_0, Y_0) < S$

Indicativamente abbiamo notato che se $max(X_0, Y_0)$ e la somma di X_0 e Y_0 restano sotto a 50, il dispositivo fornirà risultati molto precisi sia per il modulo che la fase.

Si ricorda che è comunque valido il fatto che $|kX_0, kY_0| = k \cdot |X_0, Y_0|, \forall k \in R$

4. VHDL Logic synthesis

Si ottiene lo RTL schematic



Si esegue una sintesi con un clock di periodo $14ns \approx 71.42MHz$.(Massima frequenza)

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1,612 ns	Worst Hold Slack (WHS):	0,142 ns	Worst Pulse Width Slack (WPWS):	6,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	136	Total Number of Endpoints:	136	Total Number of Endpoints:	105

All user specified timing constraints are met.

L'implementazione su ZyBo:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1,280 ns	Worst Hold Slack (WHS):	0,264 ns	Worst Pulse Width Slack (WPWS):	6,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	168	Total Number of Endpoints:	168	Total Number of Endpoints:	121

All user specified timing constraints are met.

Di seguito si riporta l'utilizzazione:

Name 1	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT Flip Flop Pairs (17600)	DSPs (80)	Bonded IOB (100)	BUFGCTRL (32)
N finalBox	260	151	119	260	36	1	99	1
Counter_map (Counter)	19	3	9	19	2	0	0	0
mul (multiplier)	12	0	6	12	0	1	0	0
theta_DFF_N (DFF_N_...	1	32	13	1	0	0	0	0
Xbox_map (Xbox)	92	48	51	92	7	0	0	0
Ybox_map (Ybox)	132	48	63	132	12	0	0	0
Zbox_map (Zbox)	4	16	7	4	2	0	0	0

5. Conclusioni

Il dispositivo è stato realizzato ed è funzionante, calcola con discreta precisione i risultati richiesti. È sicuramente possibile incrementare la precisione del CORDIC aumentando il numero di bit su cui vengono rappresentati gli ingressi e di conseguenza le uscite. Per un ulteriore incremento delle prestazioni il CORDIC può rimuovere il moltiplicatore, infatti il risultato è corretto anche se in ingresso vengono forniti direttamente $X'_0 = \frac{X_0}{A_n}$ e $Y'_0 = \frac{Y_0}{A_n}$. In questo caso sta all'utilizzatore esterno effettuare la divisione/moltiplicazione, di conseguenza però il CORDIC non deve effettuare questa operazione.