PROGETTO DI RETI LOGICHE

Funzione Seno in Virgola Fissa

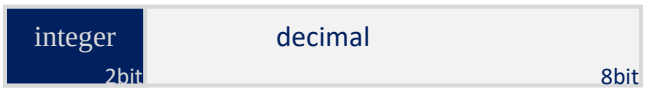
Gabriele Vianello, Vincenzo Zappia

Sommario

Introduzione

La traccia scelta prevede la progettazione e l'implementazione di un modulo che calcoli il seno di un angolo. L'angolo fornito al modulo deve essere compreso tra 0 e 359 (è espresso in gradi), codificato in binario naturale. Il valore del seno, che varia da -1 a 1, deve essere rappresentato su 10 bit in virgola fissa in cui si hanno 2 bit per la parte intera e 8 bit per la parte frazionaria.

Il modulo deve poter calcolare il seno di un angolo tra 0 e 90 gradi, quindi utilizzare le identità trigonometriche per gli angoli compresi tra 91 e 359. Il calcolo del seno tra 0 e 90 gradi deve essere basato su una lookup-table che riporta il seno degli angoli tra 0 e 88 ogni 8 gradi ed il seno per gli angoli 89 e 90 gradi.
Il seno dell'angolo di ingresso, se diverso da uno di quelli nella lookup-table, viene poi calcolato mediante interpolazione lineare nell'intervallo in cui cade nella lookup table.



Per passare da binario a decimale nella parte frazionaria il procedimento di calcolo può essere riassunto come di seguito.

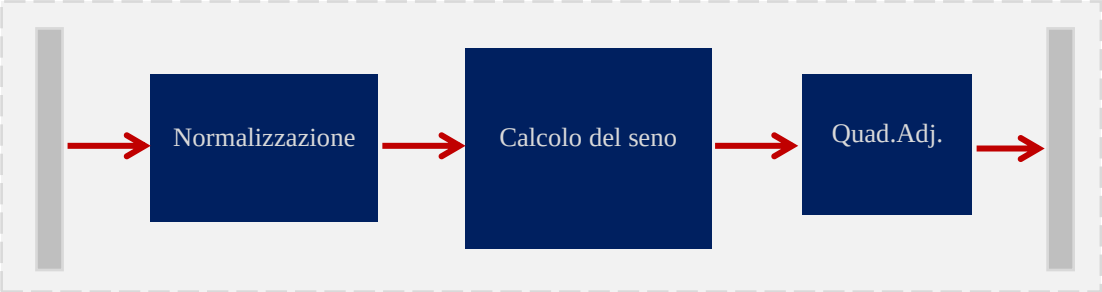
$$2^{exp}, \exp = 1 \qquad 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8$$

Esempio: $0 + 0 + 0 + 0 + 1 + 0 + 0 + 0 + 1 + 1$

$$00.00100011_2 = 0.13671875_{10}$$

Specifica

L'architettura implementata è stata virtualmente suddivisa in tre differenti stadi di funzionamento: *normalizzazione dell'angolo*, *calcolo del seno* e *correzione del quadrante*.



Normalizzazione dell'angolo: fase dove l'angolo del quale si vuole sapere il seno viene "normalizzato", cioè, ridotto all'intervallo 0° ↔ 90°. Per utilizzare il segno corretto viene propagato alla fase finale un apposito segnale a 2 bit che identifica il quadrante di appartenenza.
In questo punto dell'esecuzione si ottiene un altro risultato, *fraction* rappresenta i 3 bit meno significativi dell'angolo normalizzato ed è utilizzato per l'interpolazione. Inoltre, tutti gli input con valori maggiori di 359° vengono gestiti mandando in uscita uno '0.0'.

Calcolo del seno: vengono svolte simultaneamente tre operazioni, cioè si prelevano i valori degli angoli multipli di 8 (inferiore e superiore) anche se l'angolo è già multiplo di 8 oppure corrisponde ad un angolo di 89° o 90°, che vengono "mandati" nella fase successiva che si occupa dell'interpolazione. L'ultima delle tre operazioni, prevede che venga mandato un segnale di controllo per utilizzare direttamente il valore del seno ottenuto dalla LUT "bassa". Quindi viene mandato in uscita il segnale con il valore finale del seno (privo di segno ma già nella codifica finale richiesta dalla specifica).

Correzione del quadrante: fase finale dove viene modificato il valore del seno ottenuto in precedenza con il segno corretto. Il quadrante di appartenenza determinato nella normalizzazione dell'angolo viene utilizzato per decidere se negare o meno il risultato; è bene indicare che in questo caso, il sistema sfrutta un componente per il complemento a 2.

Inoltre, è possibile distinguere quattro casistiche nell'utilizzo del sistema:

1. **Caso semplice**, dove il valore dell'angolo normalizzato è un multiplo di 8 oppure corrisponde ad un angolo di 89° o 90°, il sistema preleva direttamente il corrispettivo valore dalla LUT e salta alla fase di calcolo del quadrante.

2. **Caso base**, situazione in cui il valore normalizzato non è multiplo di 8, si procede per l'interpolazione lineare.

3. **Errore**, nell'architettura proposta si può ricadere nella casistica di errore quando viene messo in input un valore maggiore di 359°. Non sono gestiti eventuali angoli negativi poiché non ottenibili con la codifica adottata per gli ingressi (binaria naturale).

4. **Reset**, situazione dove viene asserito il segnale di reset per il sistema (azzera i registri).

Si sottolineare che a causa del limite imposto dalla codifica del risultato in uscita dal sistema, la precisione della parte frazionaria è da considerarsi affidabile fino alla seconda cifra.

Algoritmo di Base

L'operazione alla base del funzionamento del sistema è l'interpolazione lineare, dove per calcolare -in questo caso- il seno di un angolo dato per cui si ha disponibile solo il valore del seno dell'angolo precedente e del successivo (a distanza di 8°): in primo luogo si calcola la "distanza" tra l'angolo in ingresso e le due estremità, poi si effettua una "media pesata" per determinare con sufficiente precisione il seno cercato.

```
function calculate_sine(angle):
    // Angle Normalizer
    if angle < 90:
        quadrant = 0
        normalized_angle = angle
    else if angle < 180:
        quadrant = 1
        normalized_angle = 180 - angle
    else if angle < 270:
        quadrant = 2
        normalized_angle = angle - 180
    else:
        quadrant = 3
        normalized_angle = 360 - angle

    // Lookup Table
    lower_index = normalized_angle / 8
    upper_index = lower_index + 1
    fraction = normalized_angle % 8

    lower_sine = sine_table[lower_index]
    upper_sine = sine_table[upper_index]

    // Linear Interpolator
    diff = upper_sine - lower_sine
    interpolated_sine = lower_sine + (diff * fraction / 8)

    //Quadrant Adjuster
    if quadrant == 0 or quadrant == 1:
        adjusted_sine = interpolated_sine
    else:
        adjusted_sine = -interpolated_sine

return adjusted_sine
```

Per comprendere meglio il funzionamento, a sinistra, lo pseudocodice dell'interpolazione lineare realizzata dall'architettura.[1]

In equazioni, l'operazione di interpolazione può essere semplificata come di seguito:

$$\Delta sine = sine_{upper} - sine_{lower}$$

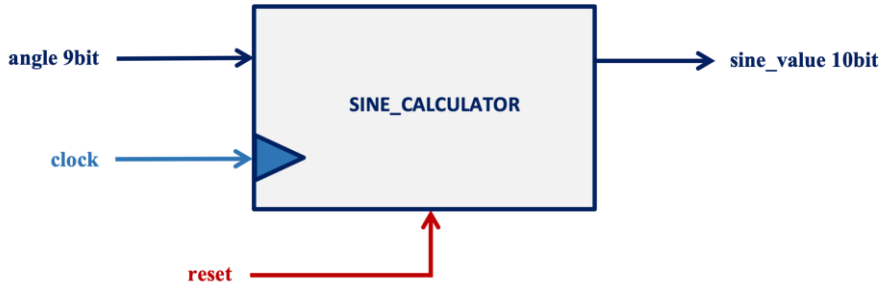
$$fraction = angle_{norm} \,\%\, 8$$

$$sine_{interpolated} = sine_{lower} + \left(\frac{diff \cdot fraction}{8}\right)$$

[1] Lo pseudocodice si riferisce al componente top-level senza scendere nello specifico, come nei casi in cui il valore dell'angolo sia già presente in LUT (senza quindi affrontare l'operazione di interpolazione) o > 359°.

Interfaccia Generale

Il componente *top-level* può essere rappresentato come di seguito, sarà denominato ***sine_calculator***[2] ; per l'architettura proposta è stata adottata la codifica binaria naturale.



I segnali in ingresso al sistema sono:

- *angle*, a 9bit rappresenta l'angolo dato, con valori compresi tra i 0° e 359°.
- *clock*, segnale di clock del sistema.
- *reset*, segnale che azzera il contenuto dei registri.

I segnali in uscita dal sistema:

- *sine_value*, a 10 bit rappresenta il risultato del calcolo del seno dell'angolo dato.

Temporizzazione:

- Per ottenere le uscite il più possibile allineate con i corrispettivi ingressi è stato scelto di rendere sensibile gli ingressi al fronte di salita del clock, mentre le uscite al fronte di discesa del clock.

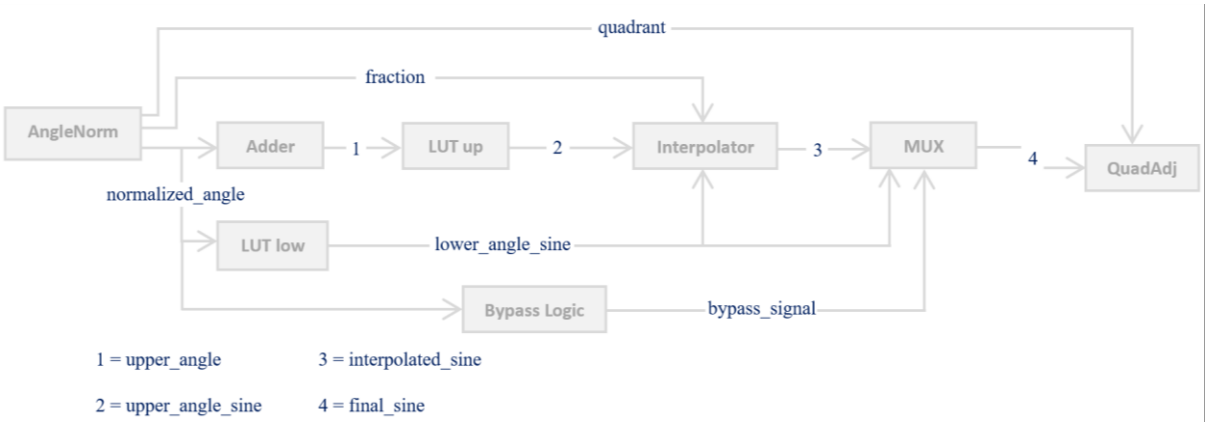
- Grazie a test che stimolassero il più possibile il percorso critico, è stato scelto di garantire al sistema un periodo di clock di almeno 40 ns, cioè una frequenza operativa di 25 MHz.

[2] In evidenza gli input e output attesi, più ulteriori segnali previsti dall'architettura.

Architettura

Nota la specifica e l'interfaccia di alto livello del sistema si può allora rappresentare quest'ultimo con uno schema a blocchi che lo scomponga in macro-componenti. Tali parti saranno poi analizzate nello specifico per osservare sia la loro struttura che il loro comportamento. I nomi assegnati, ai vari moduli, indicano la funzione svolta (non tutti corrispondono ai nomi utilizzati nel codice VHDL).

L'architettura proposta avrà due registri, uno per gli ingressi posto prima del modulo di normalizzazione, mentre l'altro per le uscite posto dopo il componente di correzione del quadrante.



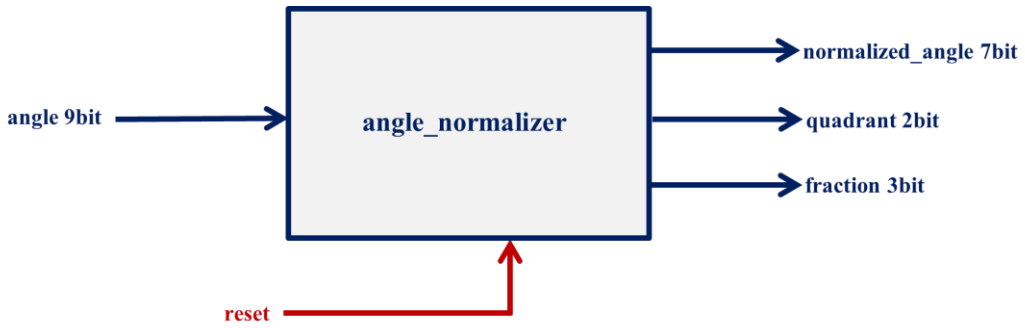
I segnali interni al componente principale, indicati nello schema, sono:

- *normalized_angle*, a 7 bit rappresenta il valore dell'angolo normalizzato da 0° a 90°.
- *fraction*, a 3 bit, contiene i 3-bit meno significativi di normalized_angle.
- *quadrant*, a 2 bit indica il quadrante di appartenenza.
- *bypass_signal*, a 1 bit usato come segnale di controllo per usare direttamente l'uscita della LUT (oppure no).
- *lower_angle_sine*, valore del seno a 10 bit.
- *upper_angle*, a 7 bit valore dell'angolo successivo a quello normalizzato (somma 8).
- *upper_angle_sine*, a 10 bit valore del seno dell'angolo successivo a quello normalizzato.
- *interpolated_sine*, a 10 bit rappresenta il valore del seno ottenuto con l'interpolazione lineare.
- *final_sine*, rappresenta il seno con dell'angolo con il rispettivo segno.

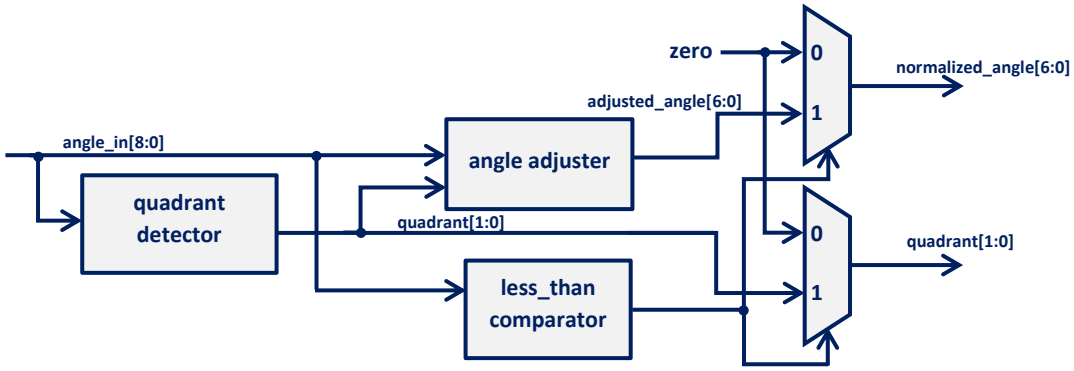
Utilizzando i *generic* ed il costrutto *generate* è stato possibile realizzare i moduli riutilizzabili per le operazioni aritmetiche necessarie, così come i multiplexer (sia a 2 che 4 vie).

ANGLE_NORMALIZER

Primo modulo ad operare sui dati a livello temporale, è fondamentale per la gestione degli errori e per garantire il corretto formato dei dati ai successivi componenti. Per dare al sistema una maggiore flessibilità è stato suddiviso in ulteriori sotto moduli che saranno descritti di seguito.



La validazione dell'input è affidata ad un comparatore la cui uscita è usata come segnale di controllo per decidere se far passare o meno l'angolo in ingresso. Nel caso in cui non fosse valido, viene restituito uno 0.0_{10}; nello schema sottostante la struttura del modulo con i suoi sottocomponenti.

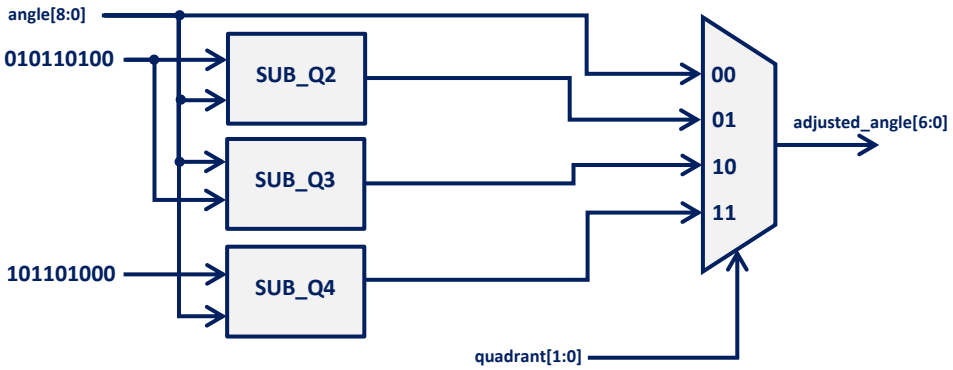


Quadrant_Detector: componente che sfrutta anch'esso dei comparatori per il calcolo del segnale di controllo utilizzato per decidere quale valore far passare in uscita con un MUX a quattro vie; i quadranti sono codificati in due bit,

- '00' per il primo, tra 0° e 90°
- '01' per il secondo, tra 91° e 180°
- '10' per il terzo, tra 181° e 270°
- '11' per il quarto, tra 271° e 359°

In particolare, i comparatori "<" determinano se l'angolo in ingresso è minore di 90, 180 e 270; i valori in uscita da questi moduli viene utilizzato da una rete combinatoria per mandare in uscita il valore del quadrante corretto.

Angle_Adjuster: componente deputato all'implementazione effettiva delle identità trigonometriche. È costituito da una combinazione di MUX e sottrattori[3], come di seguito.



Se l'angolo appartiene già all'intervallo 0-90 non verrà modificato, si può vedere come il segnale *angle* sia connesso direttamente all'*input0* del MUX4; si possono notare i valori costanti in codifica binaria naturale in ingresso ai sottrattori: $010110100_2 = 180_{10}$ e $101101000_2 = 360_{10}$.

BYPASS_LOGIC

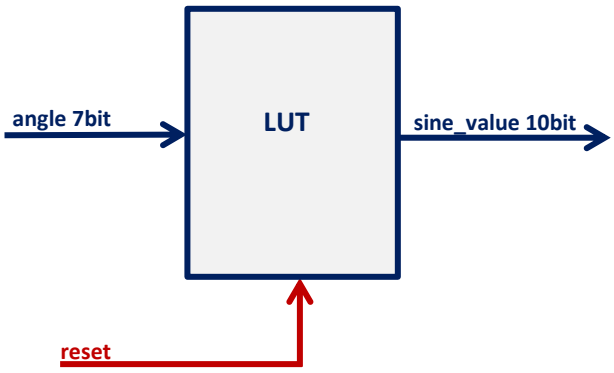
È il componente che determina se utilizzare direttamente il valore in uscita dalla LUT (dell'angolo "inferiore") oppure il risultato dell'interpolazione lineare, basandosi sul valore dell'angolo normalizzato in ingresso. Il bypass viene attivato (cioè, l'uscita 'bypass' viene impostata a '1') quando l'angolo normalizzato soddisfa una delle tre condizioni specificate nelle righe successive. In tutti gli altri casi, il bypass rimane disattivato (l'uscita 'bypass' è '0').

- **Input:**
  - *normalized_angle*, vettore di 7 bit che rappresenta l'angolo normalizzato.
- **Output:**
  - *bypass*, un singolo bit che indica se il bypass è attivo o meno.
- **Segnali interni:**
  - Le costanti binarie che corrispondono agli angoli di 89° e 90°.
- **Funzionamento:**
  - Sfrutta una rete combinatoria e comparatori di uguaglianza per asserire il segnale di bypass nei casi in cui i 3LSB siano pari a '0' oppure quando l'angolo normalizzato è di 89 o 90 gradi.

[3] Un'analisi più approfondita del modulo che si occupa di effettuare la sottrazione (e anche di tutte le altre operazioni aritmetiche).

LOOK-UP TABLE

Una LUT è un componente che permette di ottenere in uscita un valore a seconda dell'ingresso. Per l'implementazione sono stati inseriti i valori pre-calcolati del seno degli angoli tra 0 e 90° ad intervalli di 8° (eccetto gli ultimi due). La logica di scelta del valore della LUT prevede l'utilizzo dei primi quattro bit dell'angolo normalizzato, per "contare" di 8 in 8 senza ricorrere a sistemi eccessivamente complessi. La tabella che contiene i valori del seno evidenzia il limite della precisione dato dalla codifica di uscita dell'architettura: gli angoli di 88 e 89 gradi riportano apparentemente lo stesso valore.



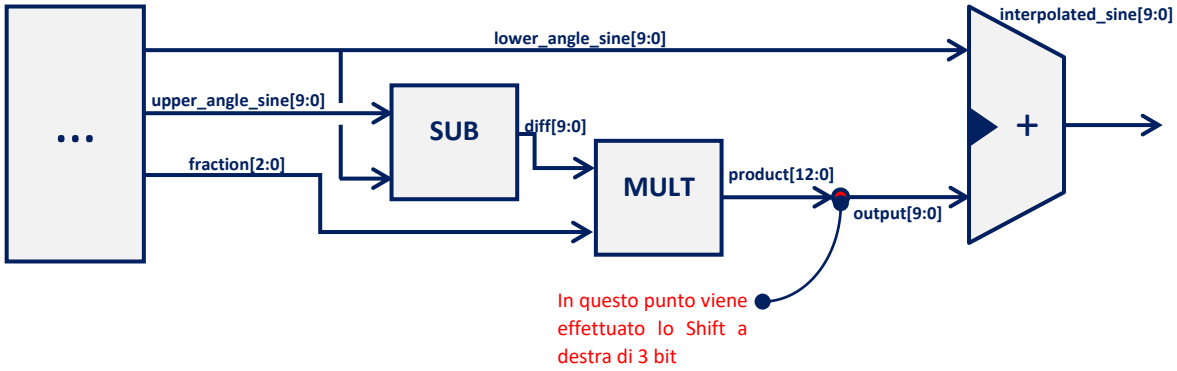
LUT values

00.00000000	sin(0) = 0.00
00.00100011	sin(8) ≈ 0.13
00.01000101	sin(16) ≈ 0.27
00.01101000	sin(24) ≈ 0.40
00.10000111	sin(32) ≈ 0.52
00.10100100	sin(40) ≈ 0.64
00.10111110	sin(48) ≈ 0.74
00.11010100	sin(56) ≈ 0.82
00.11100110	sin(64) ≈ 0.89
00.11110011	sin(72) ≈ 0.95
00.11111100	sin(80) ≈ 0.98
00.11111111	sin(88) ≈ 0.99
00.11111111	sin(89) ≈ 0.99
01.00000000	sin(90) = 1.00

- **Input:**
  - *angle*, angolo di ingresso normalizzato (7 bit, da 0 a 90 gradi)
- **Output**:
  - *sine_value*, valore del seno (10 bit, rappresentazione in virgola fissa)
- **Segnali interni:**
  - *angle_index/angle_index_temp*, indice dell'angolo decodificato (4 bit)
  - *selector*, segnale di selezione del MUX
- **Costanti:**
  - *sine_table*, tabella di lookup con 14 valori del seno precalcolati (array di 14 elementi, ciascuno di 10 bit)
- **Funzionamento:**
  - Una rete combinatoria calcola il segnale di selezione del MUX, per scegliere che valore usare come indice della LUT. In questo punto si sceglie se utilizzare il valore in uscita dallo shifter (4MSB del segnale *angle*), oppure gli indici assegnati ai due casi speciali.

INTERPOLATOR

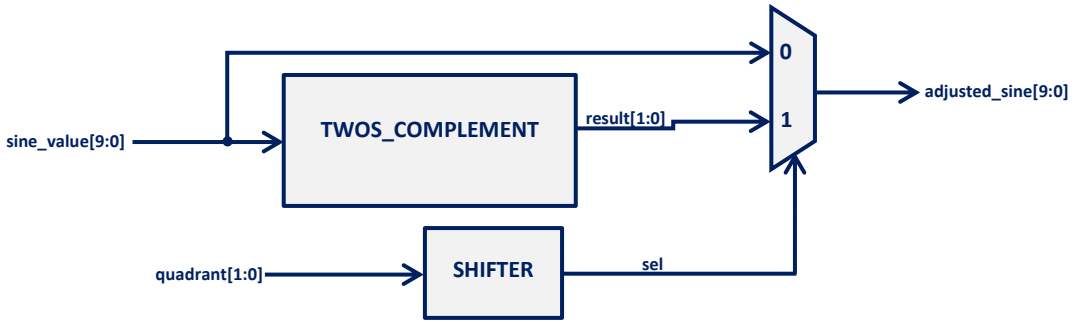
È il componente realizza effettivamente l'operazione di interpolazione lineare secondo l'equazione riportata nella sezione dedicata all'algoritmo implementato; di seguito uno schema complessivo del componente. È importante evidenziare che tanta più precisione è richiesta nel calcolo dell'interpolazione (angoli con variazioni di seno molto piccole) tanto meno sarà preciso il risultato, fino ad un massimo di $2^{-8} = 0.00390625$.



- **Input:**
  - *lower_angle*, seno dell'angolo inferiore 10 bit in codifica a virgola fissa.
  - *upper_angle*, seno dell'angolo superiore 10 bit in codifica a virgola fissa.
  - *fraction*, i 3 LSB dell'angolo normalizzato.
- **Output:**
  - *interpolated_value*, valore interpolato risultante 10 bit.
- **Segnali interni principali:**
  - *diff*, differenza tra *upper_angle* e *lower_angle.*
  - *product*, prodotto tra *diff* e *fraction.*
  - *scaled_diff*, prodotto scalato (diviso per 8).
- **sottocomponenti generici:**
  - *subtractor_generic*, calcola la differenza tra i seni degli angoli.
  - *multiplier_generic*, moltiplica la differenza per la frazione.
  - *adder_generic*, somma il risultato scalato all'angolo inferiore.
  - *right_shifter*, utilizzato per la divisione per 8 (shift di 3 bit).
- **Funzionamento:**
  - Calcolo della differenza = $upper\_angle - lower\_angle$.
  - Moltiplicazione =$diff \cdot fraction$.
  - Right shift, il risultato viene diviso per 8.
  - Somma finale = $lower\_angle + scaled\_diff$.

QUADRANT_ADJUSTER

È il componente che aggiunge il segno al seno dell'angolo ottenuto nelle fasi precedenti; è costituito da 3 moduli interni: right shifter, complemento a 2 ed un multiplexer a due vie. Il segnale di selezione viene utilizzato per scegliere se mandare in uscita il valore positivo o negativo del seno. Il valore negativo viene calcolato utilizzando il *twos_complement*; tutte le operazioni sono eseguite utilizzando la codifica in virgola fissa prevista dalle specifiche.



- **Input:**
  - *sine_value*, a 10 bit rappresenta il valore del seno in ingresso, codificato in virgola fissa.
  - *quadrant*, a 2 bit indica il quadrante.

- **Output:**
  - *adjusted_sine*, a 10 bit rappresenta il valore del seno modificato in base al quadrante d'appartenenza.

- **Segnali interni:**
  - *negated_sine*, a 10 bit rappresenta il valore negato (complemento a due) del seno in ingresso.
  - *selector*: ad un bit è il segnale di selezione per il multiplexer, ottenuto effettuando lo shift a destra di una posizione.

- **Funzionamento:**
  - Una volta ricevuto l'angolo in ingresso ed il corrispettivo quadrante, vengono calcolati simultaneamente sia lo shift che il CA2.
  - Il segnale ottenuto mediante lo shift va ad impostare l'uscita del valore del seno corretto.
  - L'uscita del multiplexer di questo modulo sarà collegata dal componente top-level al registro di uscita.

TWOS_COMPLEMENT

Questo componente calcola il complemento a due di un numero binario formato da 10bit *WIDTH*.

La logica impiegata per eseguire il calcolo consiste nel determinare inizialmente il complemento a uno del numero, e successivamente, incrementarlo di uno.

Il componente è formato da un ingresso *a* che corrisponde al numero binario del quale si vuole effettuare il complemento e un'uscita *result* che corrisponde al risultato dell'operazione. Sono presenti 2 segali interni *not_a* e *one*, il primo è un vettore di segnali che contiene l'inverso del vettore *a*, calcolato mediante l'operazione *not_a <= not a*, mente il secondo contiene il binario del numero 1 di lunghezza *WIDTH*.

Infine, il componente *ripple_carry_adder*, un sommatore a propagazione di riporto, si occuperà di eseguire l'operazione di addizione tra *not_a* e *one*.

ADDER

Questo componente è un sommatore a propagazione di riporto, formato da una catena di full adders il quale somma due numeri binari di lunghezza (*WIDTH*) fissata a 10bit. Il riporto viene propagato bit per bit lungo l'intera larghezza del vettore, partendo dal bit meno significativo e propagandosi fino al bit più significativo.

Il componente riceve in ingresso due vettori binari, *a* e *b*, che rappresentano i valori da sommare, e fornisce in uscita un vettore *sum*, il quale rappresenta la somma dei due vettori di ingresso.

MULTIPLIER

Il componente multiplier è un moltiplicatore binario che utilizza la tecnica dei prodotti parziali per moltiplicare due numeri binari *a* e *b* di lunghezza 10bit e 3bit. I prodotti parziali vengono successivamente sommati utilizzando una catena di *ripple_carry_adder*, sommatori a propagazione di riporto. Il componente presenta un'uscita (product) che rappresenta il prodotto tra i due numeri di lunghezza pari a (*A_WIDTH + B_WIDTH*).

Il componente presenta due segnali interni: *partial_product_array* e *sum_array* che memorizzano uno il vettore di prodotti parziali e l'altro i risultati intermedi delle somme dei prodotti parziali.

SUBTRACTOR

Il componente è un sottrattore binario a 10bit che utilizza la tecnica del complemento a due. In particolare, presenta due ingressi *a* e *b*, entrambi vettori binari interi positivi di lunghezza pari a 10bit. Il componente effettua il complemento a due del vettore binario *b* in due fasi, inizialmente complementando tutti i bit del vettore e successivamente imposta il valore *1* al primo bit di riporto in modo da eseguire il complemento.

Infine, restituisce il risultato della differenza tramite *diff*.

Verifica

Per la verifica della correttezza dell'architettura è stato seguito un approccio modulare, dove ciascun componente è stato testato separatamente per verificarne il comportamento. Una volta terminati i moduli più importanti è stato simulato il funzionamento del componente top-level, andando a modificare opportunamente il periodo di clock in funzione dei risultati ottenuti.

Test-bench

Il percorso critico è quello che comprende l'interpolazione, per questo motivo una buona parte degli angoli in ingresso sono quelli che necessitano dell'interpolazione lineare ed il cambiamento di segno. Sono state effettuate e verificate le seguenti simulazioni: *Behavioural*, *post-implementation functional* e *timing*. Nella pagina successiva i risultati delle simulazioni (angoli inseriti in ordine sparso).

Casi d'uso

Di seguito sono elencati i casi d'uso verificati dal test-bench, con ingressi e risultati attesi.

- Reset, qualunque sia l'input, finché il segnale reset è attivo l'uscita è $0000000000_2 = 0.0_{10}$.

- Angoli il cui valore del seno precalcolato è presente in LUT, nel test-bench si possono osservare in particolare i seguenti valori:

INPUT	OUTPUT
$0_{10} = 000000000_2$	$+0.00_{10} = 00.00000000_2$
$8_{10} = 000001000_2$	$+0.13_{10} = 00.00100011_2$
$56_{10} = 000111000_2$	$+0.82_{10} = 00.11010100_2$
$88_{10} = 001011000_2$	$+0.99_{10} = 00.11111111_2$
$89_{10} = 001011001_2$	$+0.99_{10} = 00.11111111_2$
$90_{10} = 001011010_2$	$+1.00_{10} = 01.00000000_2$
$180_{10} = 010110100_2$	$+0.00_{10} = 00.00000000_2$
$270_{10} = 100001110_2$	$-1.00_{10} = 11.00000000_2$

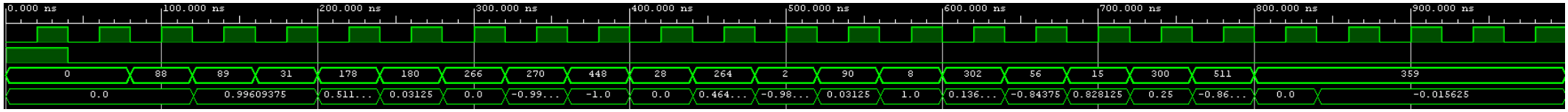
- Angoli che necessitano di interpolazione, alcuni con cambiamento di segno (viene stimolato il percorso critico):

INPUT	OUTPUT
$2_{10} = 000000010_2$	$+0.03_{10} = 00.00001000_2$
$15_{10} = 000001111_2$	$+0.25_{10} = 00.01000000_2$
$28_{10} = 000011100_2$	$+0.46_{10} = 00.01110111_2$
$31_{10} = 000011111_2$	$+0.51_{10} = 00.10000011_2$
$178_{10} = 001011001_2$	$+0.03_{10} = 00.00001000_2$
$264_{10} = 001011010_2$	$-0.98_{10} = 11.00000011_2$
$266_{10} = 010110100_2$	$-0.99_{10} = 1100000010_2$
$300_{10} = 100001110_2$	$-0.86_{10} = 11.00100011_2$
$302_{10} = 101100111_2$	$-0.84_{10} = 11.00101000_2$
$359_{10} = 101100111_2$	$-0.01_{10} = 11.11111100_2$

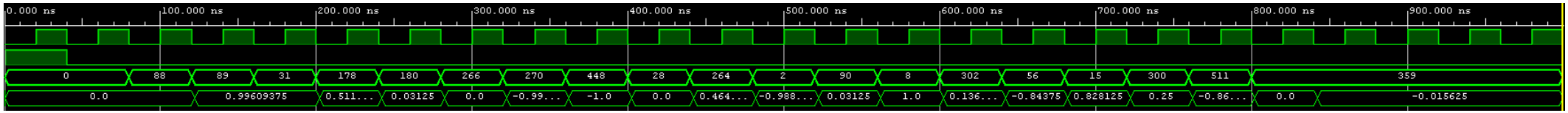
- Angoli maggiori di 359°, si ricorda che il sistema li gestisce mandando in output 0.0_{10}:

INPUT	OUTPUT
$448_{10} = 111000000_2$	$0.0_{10} = 00.00000000_2$
$511_{10} = 111111111_2$	$0.0_{10} = 00.00000000_2$

Post-Implementation Functional Simulation



Post-Implementation Timing Simulation



Behavioural Simulation