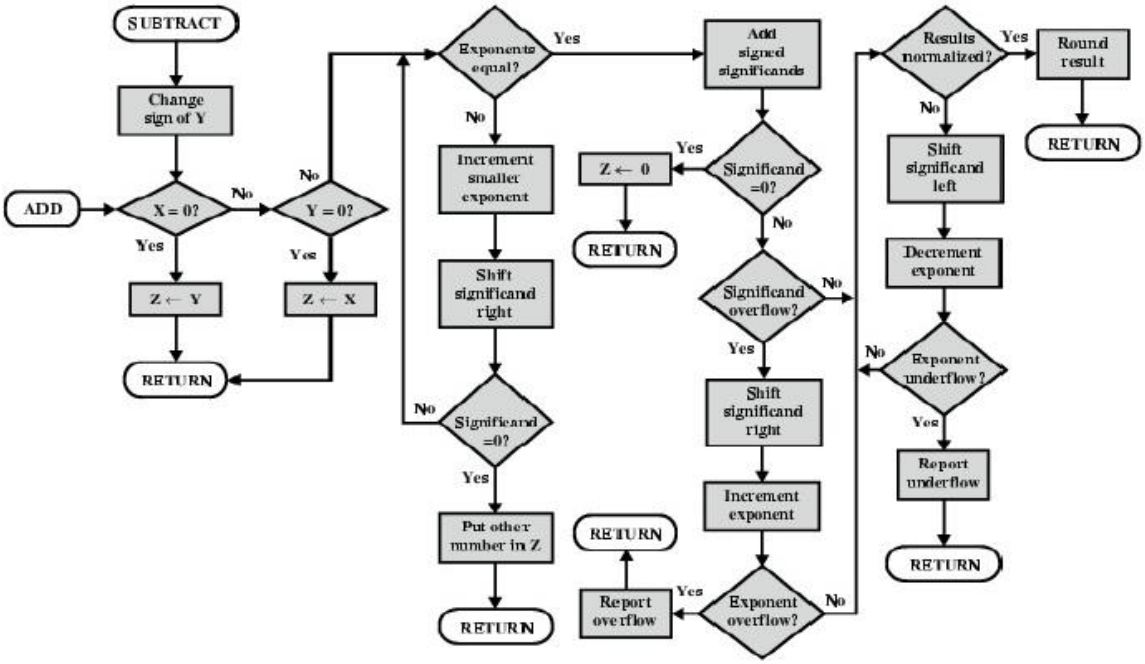
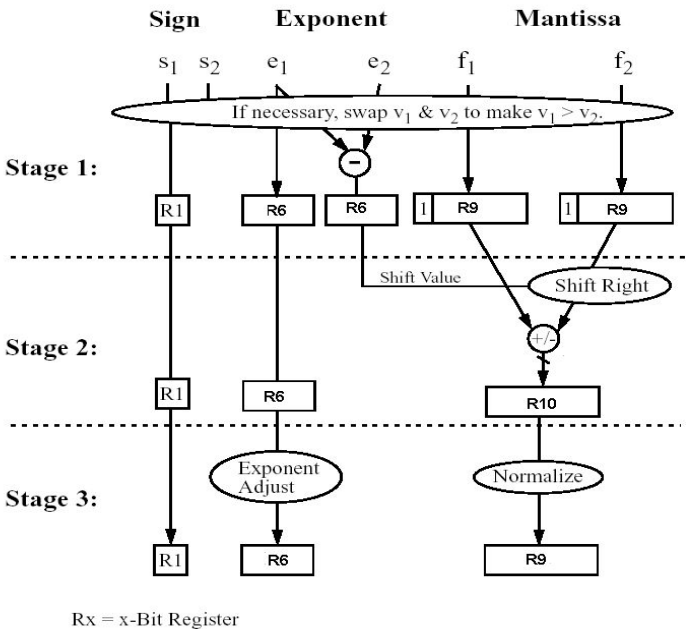
Progetto di Reti Logiche 2020-2021

Progetto 1: Sommatore floating-point IEEE754

Si realizzi un sommatore floating-point secondo lo standard IEEE754. Il sommatore deve trattare operandi normalizzati e non e, opzionalmente gli operandi speciali NaN, Infinity. Uno schema di massima del flusso di operazioni richieste è il seguente:



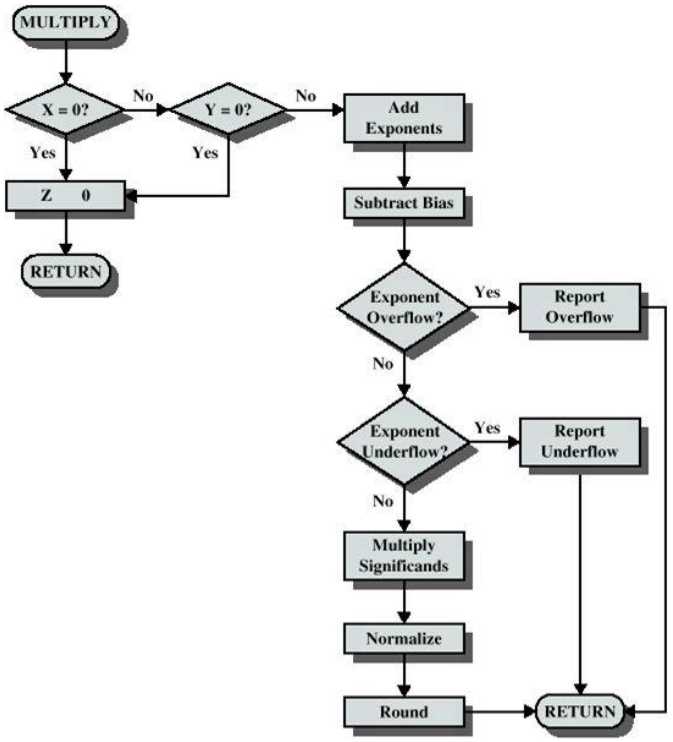
Il sommatore deve essere realizzato mediante una architettura pipelined. La figura seguente mostra una possibile realizzazione.



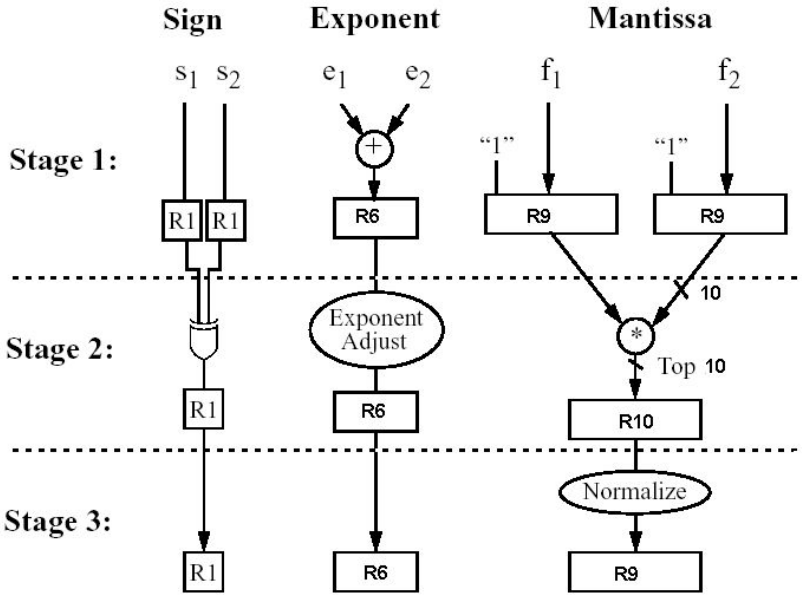
Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi.

Progetto 2: Moltiplicatore floating-point IEEE754

Si realizzi un sommatore floating-point secondo lo standard IEEE754. Il sommatore deve trattare operandi normalizzati e non e, opzionalmente gli operandi speciali NaN, Infinity. Uno schema di massima del flusso di operazioni richieste è il seguente:



Il sommatore deve essere realizzato mediante una architettura pipelined. La figura seguente mostra una possibile realizzazione.

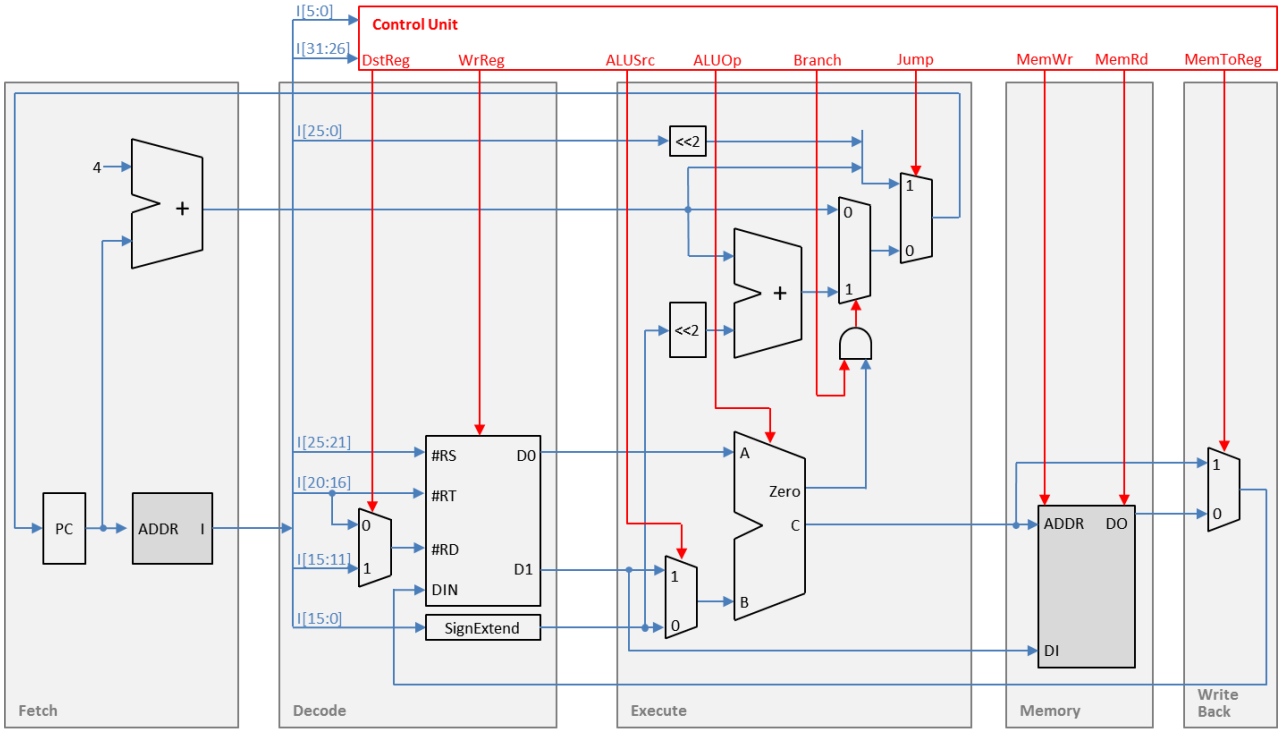


Rx = x-Bit Register

Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi.

Progetto 3: MIPS

Si progetti l'architettura MIPS senza pipeline e senza forwarding, così come mostrata dalla figura seguente.



Ai fini della progettazione, la memoria può essere semplicemente modellata a livello behavioral come array di parole (non è richiesta l'istanziazione delle memorie della FPGA).

Una volta realizzato il microprocessore, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento di alcune istruzioni rappresentative dell'instruction set.

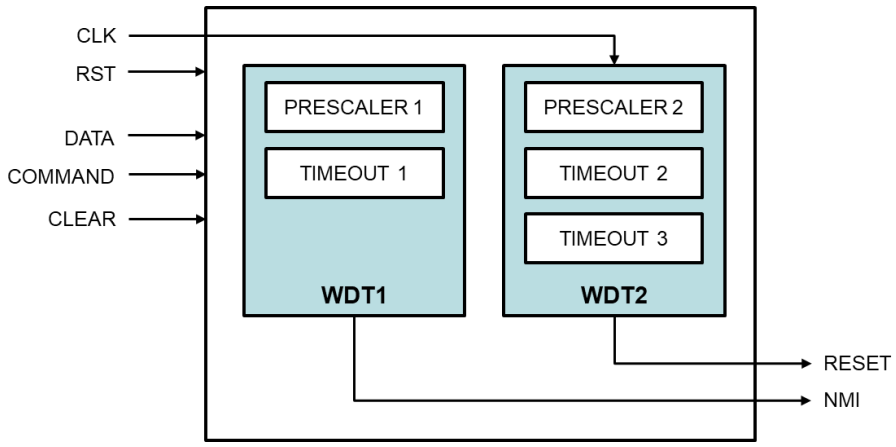
Progetto 4: Windowed watchdog counter programmabile multi-stadio

Si progetti un windowed watchdog counter multi-stadio, ovvero un windowed watchdog con tre differenti tempi di triggering:

- Il tempo TNMI è gestito da un primo watchdog counter WDT1, configurabile mediante un prescaler (PRESCALER 1) ed un timeout (TIMEOUT 1)

- I tempi TWMIN e TWMAX sono gestiti da un secondo watchdog WDT2 configurabile mediante un unico prescaler (PRESCALER 2) e due timeout distinti (TIMEOUT 2 e TIMEOUT 3)

Il primo watchdog è destinato a stimolare un'azione correttiva mediante un interrupt non mascherabile (NMI), il secondo è destinato al reset (RESET) del microcontrollore o del microprocessore. Una architettura di alto livello che indica gli ingressi e le uscite principali è la seguente:



Ognuno dei due stadi deve essere programmabile mediante i parametri:

- PRESCALER n: divide la frequenza del clock in ingresso di un fattore pari a 2^K, con K = [0..7]
- TMIOUT n: definisce il conteggio del timeout, espresso su 16 bit.

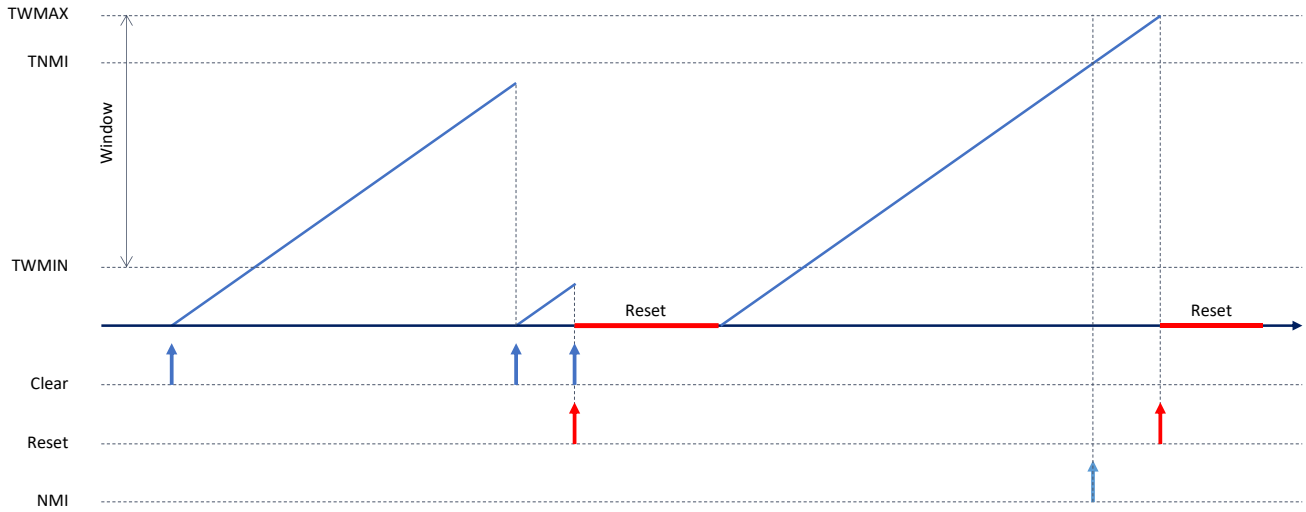
Una volta programmati, i due timer iniziano il normale conteggio, fino al timeout, a meno che non vengano resettati (titillati) dal segnale di CLEAR. Il sistema deve prevededre un comando per l'avvio del watchdog. Una volta avviato, il watchdog non può essere più arrestato né riprogrammato.

Il comportamento di un watchdog a finesta (windowed) stabilisce oltre ad un tempo massimo entro cui ricevere il clear ed un segnale di "preallarme" per la generazione del NMI, anche un tempo minimo. Qualora il segnale di clear giunga prima del valore minimo della finestra, il watchdog genera il segnale di reset. Il comportamento può essere così sintetizzato:

- TWMIN: se il segnale di clear arriva prima di TWMIN, scatta il reset
- TWMAX: se il segnale di clear arriva dopo TWMAX, scatta il reset
- TNMI: se il contatore raggiunge il conteggio TNMI, viene generato un interrupt.

Si tenga presente che deve sempre essere TWMIN < TNMI < TWMAX.

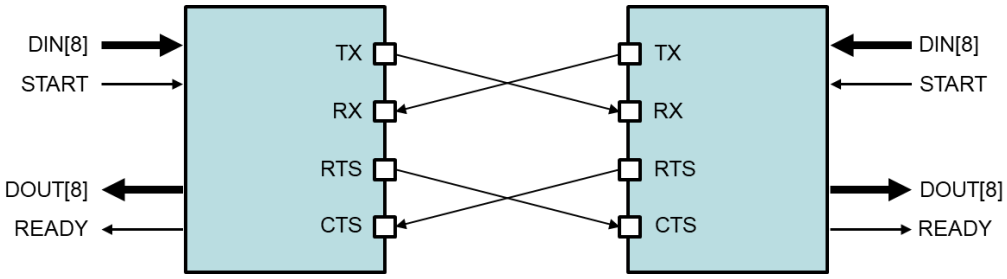
Lo schema seguente descrive tale comportamento.



Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi.

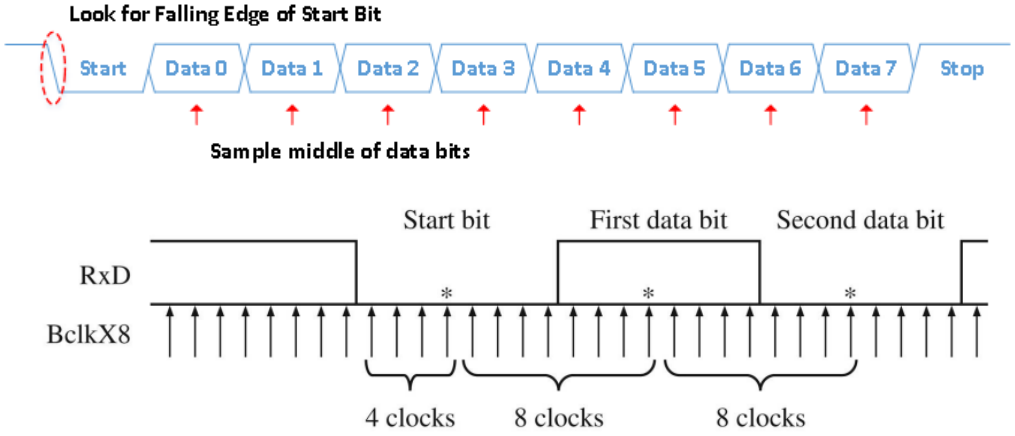
Progetto 5: Interfaccia seriale asincrona UART

Si progettino un trasmettitore ed un ricevitore UART dotati di segnalazione hardware del controllo del flusso. La trasmissione UART si basa su 4 linee digitali: TX (Trasmissione), RX (Ricezione), RTS (Request to send) e CTS(Clear to send). Queste linee sono connesse secondo lo schema mostrato di seguito.



La linea di trasmissione assume normalmente valore 1. Quando il trasmettitore intende iniziare un trasferimento dati abbassa la linea a 0 (start bit) quindi trasmette 8 bit, seguiti da un ulteriore 1 (stop bit). La trasmissione avviene sempre e solo in blocchi di 8 bit che, a seconda delle impostazioni possono essere 7 bit di dati più un bit di parità oppure 8 bit di dati, senza parità. La figura seguente mostra la trasmissione di un byte.

Trattandosi di una trasmissione asincrona, i fronti di transizione dei bit non sono allineati ad un clock condiviso fra trasmettitore e ricevitore ma avvengono ad una frequenza fissata, seppur spesso non precisa. Le frequenze tipiche di funzionamento (normalmente indicate come baudrate) sono 9600, 19200, 38400, 57600, 115200, 230400, 460800 e 921600. Dato che la frequenza di trasmissione/ricezione è spesso affetta da errori e disturbi, il segnale sulla linea viene sovracampionato con un clock 8 o 16 volte più veloce della frequenza effettiva di funzionamento. La figura seguente mostra tale situazione nel caso di sovracampionamento di un fattore 8.



I segnali RTS e CTS implementano il controllo di flusso a livello hardware. Quando un dispositivo (ricevente) è disponibile ed è in grado di accettare dati, mantiene il segnale RTS asserito: data la connessione tra RTS e CTS l'altro dispositivo (trasmittente) vede il segnale CTS asserito e trasmette. Poco prima che il ricevente diventi indisponibile (per esempio quando il suo buffer di ricezione è quasi pieno), questo deasserisce il segnale RTS, che viene rilevato sul CTS del trasmittente, il quale interrompe immediatamente la trasmissione.

Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi.

Progetto 6: Divisore intero con resto

Si progetti un divisore intero su 32 bit basato sul metodo detto di “divisione lunga”. Siano N il dividendo, D il divisore, Q il quoziente ed R il resto. Il metodo iterativo è descritto dal seguente pseudocodice:

```
if( D == 0  ) {
  error();
}

Q = 0
R = 0
for( n = 31; n >= 0; n-- ) {
  R = R << 1
  R[0] = N[n]
  if( R ≥ D ) {
    R = R – D
    Q[i] = 1
  } else {
    Q[i] = 0
  }
}
```

Si progetti una rete sequenziale che implementa il divisore basato su tale algoritmo. Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi.

Progetto 7: Filtro esponenziale fixed-point

Si progetti un circuito sequenziale per la realizzazione di un filtro esponenziale configurabile. Il sistema riceve una sequenza continua di campioni X_t rappresentati in virgola fissa su 32 bit in cui 16 bit rappresentano la parte frazionaria e produce in uscita il segnale filtrato Y_t rappresentato con la stessa notazione e calcolato come:

$$Y_t = \alpha X_t + (1 - \alpha) Y_{t-1}$$

Per semplicità implementativa si limita il coefficiente del filtro α alle potenze negative del 2. In questo modo si ha:

$$\alpha = \frac{1}{2^k} \qquad 1 - \alpha = \frac{2^k - 1}{2^k}$$

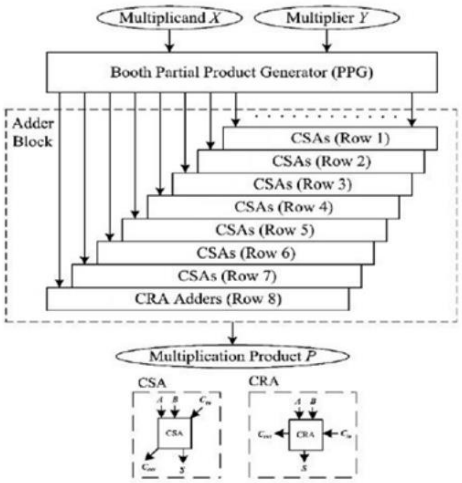
Il circuito deve pertanto essere configurabile rispetto al valore di k che determina il coefficiente del filtro.

Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi. In particolare è necessario verificare il comportamento del filtro rispetto alla risposta al gradino.

Progetto 8: Moltiplicatore di Booth

Si progetti un moltiplicatore intero - puramente combinatorio - in grado di calcolare il prodotto di operandi con segno mediante la codifica di Booth. Il moltiplicatore riceve in ingresso due operandi di 8 bit in codifica in complemento a due e produce in uscita un risultato su 16 bit, sempre in complemento a due. La codifica di Booth pertanto è utilizzato solo internamente e non è visibile all'interfaccia del componente.

Uno schema di massima dell'architettura è il seguente:



Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi.

Progetto 9: Moltiplicatore sequenziale

Si progetti un moltiplicatore sequenziale in grado di calcolare il prodotto di sue operandi A e B codificati in complemento a 2 su 32 bit. Si assuma inoltre che – nonostante la codifica – l'operando B sia sempre positivo. Il moltiplicatore riceve gli ingressi ed un segnale di start che avvia il prodotto, quindi procede a calcolare il risultato operando in modo sequenziale accumulando i prodotti parziali in un registro interno che, a prodotto ultimato, sarà fornito in uscita unitamente ad un segnale di ready.

Il comportamento del moltiplicatore può essere riassunto dalla seguente espressione:

$$Q_n = Q_{n-1} + |A| \cdot b_n$$

In cui Q_n indica il valore dell'accumulatore al passo n-esimo dell'operazione e b_n il bit di B in posizione n. Il risultato finale è ottenuto dopo 32 passi e pertanto coincide con Q_{32}. Si noti che l'operazione viene eseguita sul valore assoluto di A: per questa ragione il risultato finale deve poi essere corretto con il segno adeguato.

Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione e la verifica del corretto funzionamento nei diversi casi.