DIVISORE INTERO CON RESTO

Prova Finale di Reti Logiche

Realizzato da:
FRANCESCO MATTIOLI

POLITECNICO
MILANO 1863

Indice

Capitolo 1

Introduzione

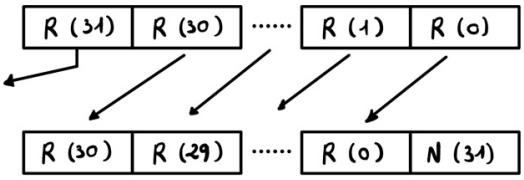
Lo scopo del progetto, descritto in questa dispensa, è l'implementazione di un componente hardware utilizzando linguaggio VHDL; tale componente deve essere in grado di effettuare la divisione intera tra due numeri in codifica binaria a 32 bit basandosi sul metodo di "divisione lunga".
Troverete all'interno del documento parti teoriche, grafici e brevi approfondimenti.

Divisione Binaria

Al fine di introdurre la divisione binaria, segue una breve descrizione dell'algoritmo di calcolo.

Supponendo di ricevere in ingresso due numeri binari interi N (Dividendo) e D (Divisore) a 32 bit, l'algoritmo che effettua il calcolo divisorio tra N e D fornisce in output il quoziente Q e il resto R (anch'essi a 32 bit) ed è composto dai seguenti passi:

1. Realizzazione di un controllo preventivo per segnalare un errore qualora il divisore D sia zero
2. Inizializzazione di Q ed R a zero.
3. Esecuzione di un ciclo[1] di 32 iterazioni caratterizzato dai seguenti passaggi:
    a. effettuare uno ***shift*** di un bit verso sinistra del resto R, dove in posizione R(0) viene inserito il bit N(n), dove n è la corrente iterazione del ciclo. Di seguito si veda un esempio;



    b. verificare se R "shiftato" sia maggiore di D:
        i. in caso affermativo, R assume il risultato ottenuto eseguendo la sottrazione binaria tra R e D; inoltre Q(n) assume il valore binario '1';
        ii. nel caso in cui D sia maggiore di R, il resto rimane inalterato, mentre Q(n) assume il valore binario '0'.

4. Al termine delle 32 iterazioni si ottiene il risultato della divisione

1. Il ciclo va da 31 a 0

Casi Particolari

In caso di divisione binaria, il caso particolare si ha quando il Divisore è pari zero. In questo caso, verrà opportunamente segnalato l'errore.

Capitolo 2

Struttura del Divisore

In questa sezione verrà descritto il divisore a Top Level, ovvero partendo da una descrizione del sistema come insieme di moduli, per poi procedere con l'approfondimento di ogni modulo.

Interfaccia del Divisore

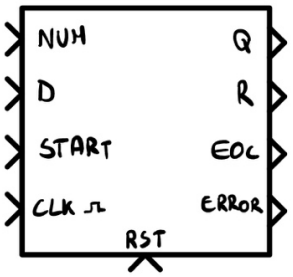


Figura 1

Il divisore a top level (Figura 1) prevede in ingresso:

- Un dividendo NUM a 32 bit
- Un divisore D a 32 bit
- Un segnale di START di un bit che permette di iniziare la divisione
- Un segnale di clock CLK di un bit in modo da poter sincronizzare il modulo
- Un segnale di reset RST di un bit che permette di portare l'uscita del modulo a zero

In uscita il divisore a top level prevede:

- Il quoziente Q di 32 bit
- Il resto R di 32 bit
- Il segnale EOC di *end of computation*, il quale segnala il termine del calcolo della divisione e la conseguente possibilità di leggere i risultati corretti
- Il segnale di errore ERROR, che assume valore '1' quando il divisore D in ingresso al sistema è pari a zero (questo caso viene gestito da un particolare componente analizzato in seguito (vedere il modulo CHECK_ERROR)

Il divisore impiega 32 cicli di clock per effettuare il calcolo; necessita inoltre di un ciclo di clock per salvare N e D nei rispettivi registri quando START = ‘1’ e infine un ciclo di clock per poter leggere i risultati corretti quando EOC = ’1’.

Il segnale EOC (end of computation) rimane alto per un solo ciclo di clock.

Il segnale ERROR viene asserito solo dopo aver memorizzato D e nel caso in cui il divisore sia nullo, rimane alto dal ciclo di clock successivo a START = ‘1’ fino a quando non viene portato a zero ***end of computation*** al termine della divisione.

Architettura del Divisore

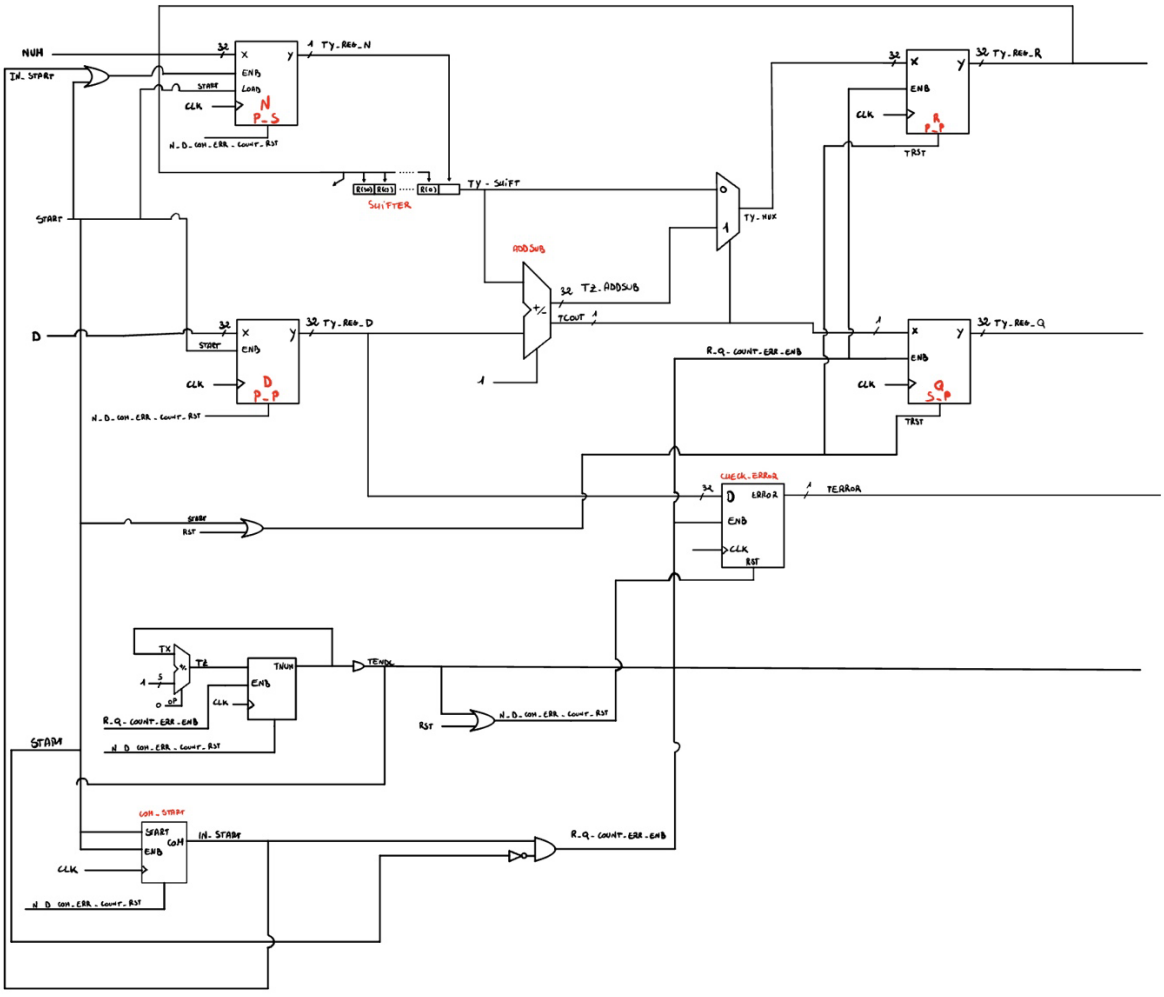


Figura 2 - In questo schema è possibile osservare i segnali in ingresso ai singoli moduli e come essi sono collegati tra loro.

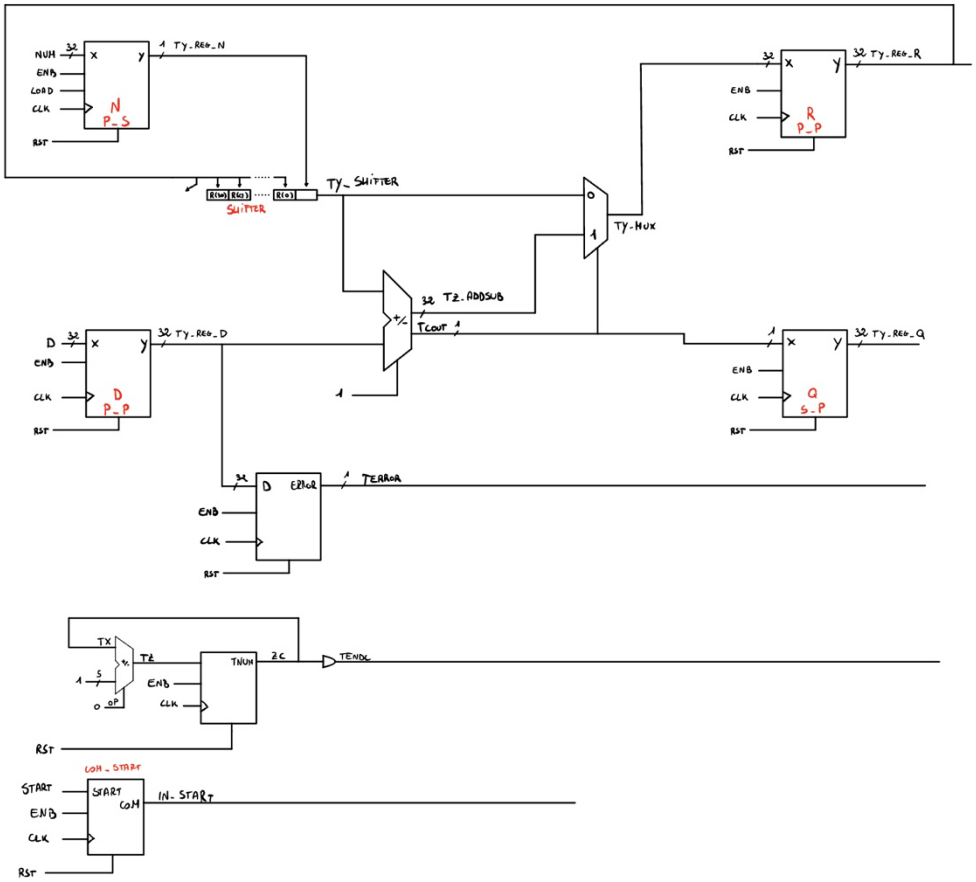


Figura 3 – Schema semplificato dell'architettura che evidenzia i collegamenti tra i vari moduli.

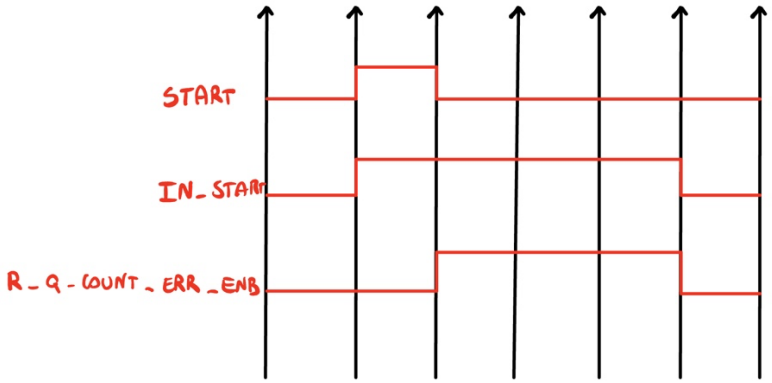
Il divisore è formato da quattro moduli che consentono di salvare N, D, Q e R, un modulo contatore che indica quando l'operazione è terminata, un modulo per effettuare il controllo del divisore nullo, un modulo *shifter* che permette di traslare verso sinistra di un bit il resto, un modulo per effettuare la sottrazione tra R "shiftato" e D; infine un modulo che permette di gestire l'***enable*** di determinati componenti.

Facendo riferimento alla Figura 2 e 3, il passo 3b dell'algoritmo viene svolto tramite un sottrattore e un multiplexer con un solo bit di selezione, in quanto la disuguaglianza può essere ricondotta a una sottrazione, il cui riporto permette di definire se $R_{shiftato} > D$. Il sottrattore, oltre a fornire il risultato dell'operazione, fornisce appunto il riporto COUT. Nel caso in cui COUT = '1' significa che $R_{shiftato} > D$, altrimenti se COUT = '0' significa che $R_{shiftato} < D$.
Conseguentemente COUT verrà utilizzato come bit di selezione del mux in Figura 2 per poter scegliere correttamente il nuovo valore di R; inoltre COUT verrà salvato in Q.

SEGNALI

Di seguito verranno introdotti alcuni segnali importanti che sono stati utilizzati all'interno del divisore come segnali di reset o enable di determinati moduli.

- $TRST = RST\ or\ START$ necessario per resettare alcuni componenti;
- $NENB = START\ or\ INSTART$ segnale di enable per il registro N;
- $N\_D\_COM\_ERR\_COUNT\_RST = RST\ or\ TEOC$ necessario per resettare alcuni componenti;
- $R\_Q\_COUNT\_ERR\_ENB = IN\_START\ and\ (not\ START)$ segnale di enable per alcuni componenti. Segue uno scema d'esempio;



Funzionamento del Divisore

In questa sezione viene spiegato, in breve, il funzionamento generale del componente divisore facendo riferimento all'algoritmo presentato sopra (Pagina 2).

1. Una volta che l'utente asserisce il segnale di START si verificano i seguenti eventi:

    a. I registri N e D memorizzano (sul fronte di salita del clock) rispettivamente il dividendo e il divisore e forniscono in uscita i primi dati validi per l'operazione;

    b. Il componente COM_START viene abilitato e conseguentemente salva il valore di START;

    c. Il contatore è disabilitato e la sua uscita è posta a zero tramite il proprio reset;

    d. Viene effettuato il primo shift di R e la prima sottrazione $R_{shiftato} - D$;

    e. In base al risultato ottenuto al punto precedente avviene la selezione, tramite un mux, del nuovo valore assunto da R.

2. Il ciclo di clock successivo si verificano i seguenti eventi:

    a. I registri Q e R salvano i valori calcolati al punto 1d (il nuovo valore di R e il bit COUT);

    b. Il contatore viene abilitato ed inizia il conteggio;

    c. Il componente CHECK_ERROR viene abilitato e controlla se il divisore D è nullo;

    d. Vengono ripetuti i passaggi 1d e 1e.

Il punto 2 viene ripetuto fino al termine del conteggio, ovvero fino a quando l'uscita del contatore è pari a 31.

3. Il ciclo di clock successivo al termine del conteggio viene asserito il segnale di end of computation, il quale indica il termine della computazione.

Moduli

In seguito, verranno analizzati singolarmente i moduli che compongono il divisore.

SOMMATORE/SOTTRATTORE

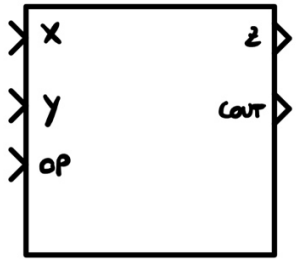


Figura 4

Il sommatore/sottrattore (Figura 4) è un modulo combinatorio che riceve in ingresso:

- Il primo operando X di n bit
- Il secondo operando Y di n bit
- Il segnale OP di un bit che determina l'operazione che verrà svolta dal modulo (se OP = '1' si avrà una sottrazione, mentre se OP = '0' si avrà una somma)

In uscita il sommatore/sottrattore prevede:

- Il valore dell'operazione Z di ***n*** bit
- Il riporto finale COUT

Nel caso del sottrattore, all'interno del divisore, il risultato Z viene fornito in ingresso a un MUX dotato di un solo bit di selezione; nel quale l'altro ingresso sarà il resto R "shiftato" (spiegato alla fine del capitolo Architettura del Divisore).

Visto che questo componente viene utilizzato in più contesti all'interno del divisore, si è deciso di non avere una dimensione fissa del componente ma di poterla assegnare al momento della creazione dell'istanza dello stesso. Per effettuare la sottrazione $R_{shiftato} - D$ serve un sottrattore a 32 bit mentre per effettuare la somma all'interno del contatore serve un sommatore a 5 bit.

Richiami Teorici

L'operazione di sottrazione può essere ricondotta a una somma tra il minuendo e il complemento a due del sottraendo. Il complemento a due di un numero binario si ottiene facendo:

- Il complemento a uno del numero negando ogni bit del numero fornito;
- Sommando il valore numerico 1 (codificato come 0…01 su n bit) al numero in complemento a uno.

Ricordando la funzione svolta dall'operatore XOR, si considerino l'ingresso OP e l'ingresso Y, si nota che $y_i \oplus OP$ vale y_i se OP = '0', mentre vale y_i' se OP = '1'. Grazie a questa proprietà, il complemento a uno può essere svolto mediante porte XOR programmabili mediante il segnale OP. Sfruttando questa proprietà il seguente modulo, si comporta da sottrattore nel caso in cui OP = '1', mentre si comporta da sommatore nel caso in cui OP = '0'.
Per la realizzazione di questo modulo è stato utilizzato un Ripple-Carry-Adder RCA su N bit, nel quale la somma si esegue a partire dalle cifre meno significative e, procedendo verso sinistra, si propaga l'eventuale riporto. Dati due valori X e Y segue lo schema della somma con propagazione del riporto;

$$\begin{array}{cccccc} c_{n+1} & c_n \cdots c_i \cdots c_1 & c_0 & & \\ & x_n \cdots x_i \cdots x_1 & x_0 & + \\ & y_n \cdots y_i \cdots y_1 & y_0 & = \\ \hline s_{n+1} & s_n \quad s_i \quad s_1 & s_0 & \end{array}$$

Figura 5

Nella generica posizione *i* si calcola $x_i + y_i + c_i$ ottenendo come risultato il bit della somma s_i ed il bit di riporto c_{i+1}. L'elemento di base necessario per la costruzione di un generico sommatore/sottrattore a n bit è chiamato full-adder FA, e prende in ingresso $x_i,\ y_i\ e\ c_i$ e produce come risultato s_i ed il bit di riporto per la propagazione c_{i+1}.
Dalla tabella di verità si ricavano le seguenti espressioni per s_i e c_{i+1}:

$$s_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

x_i	y_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

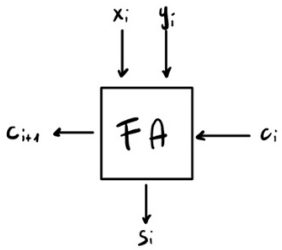


Figura 6

Per ottenere un sommatore/sottrattore a *n* bit si utilizzano *n* full-adder connessi in cascata, in modo tale che il riporto in uscita di uno sia connesso al riporto in ingresso del successivo. Nel caso del sommatore/sottrattore il riporto c_0 prende il valore di OP in quanto, nel caso di sottrazione, viene sommato '1' per effettuare il complemento a due del sottraendo. Facendo riferimento a quanto spiegato riguardo l'operatore XOR, si può notare dalla Figura 7 come ogni bit di Y venga portato in ingresso a una porta XOR con OP; questo permette, in caso di somma di mantenere il generico ingresso y_i invariato, mentre nel caso di sottrazione di negare il bit y_i per poi ottenere il complemento a uno di Y.
La struttura finale di questo componente è la seguente:

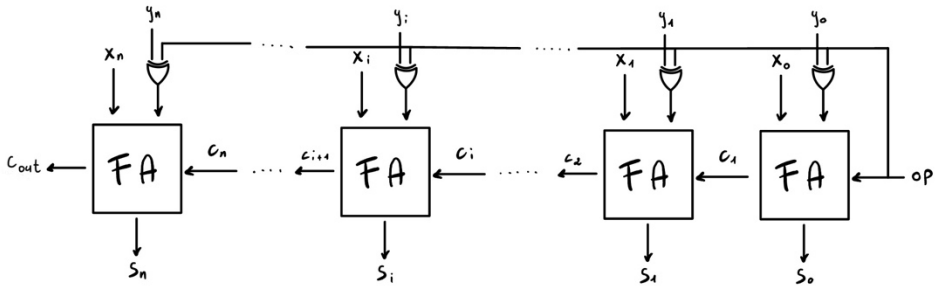


Figura 7

CONTATORE 5 BIT

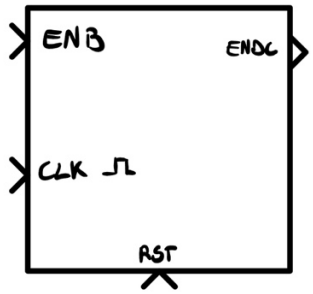


Figura 8

Il modulo contatore (Figura 8) riceve in ingresso:

- un segnale di ENB di un bit che permette di abilitare il modulo
- un segnale di CLK di un bit per la sincronizzazione del modulo
- un segnale di RST sincrono di un bit che permette di portare le uscite del modulo a zero.

Il contatore in uscita fornisce un segnale di ***end of computation*** ENDC che indica quando termina la computazione.

È stato realizzato un contatore modulo 5 dove il ciclo di conteggio corrisponde alla codifica binaria dei valori numerici da 0 a 31 ($= 2^5 - 1$). Per la progettazione del contatore si è tenuto conto della relazione tra il valore dell'uscita in istanti consecutivi, ovvero $Z_{(t+1)} = Z_t + 1$. A tale scopo il sommatore (vedere figura sottostante) provvede al calcolo del valore successivo del ciclo di conteggio a partire dal valore corrente memorizzato nel registro parallelo/parallelo che funge da registro di stato. La struttura del contatore utilizzato è la seguente.

La struttura del contatore utilizzato è la seguente

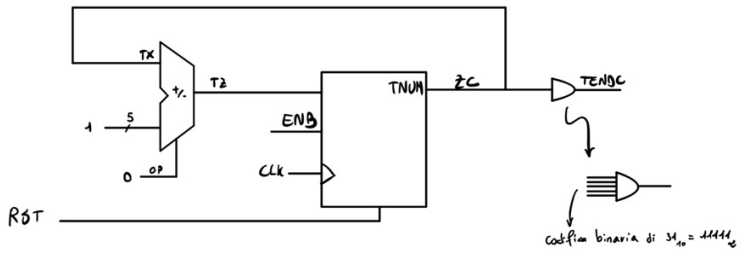


Figura 9

Nel caso del componente divisore, il segnale di RST sincrono è dato da *N_D_COM_ERR_COUNT_RST* (vedere sezione Segnali a pagina 6) così che il contatore venga resettato un ciclo di clock dopo che è stato asserito il segnale ENDC (così quest'ultimo rimane asserito per un solo ciclo di clock). Il segnale di ENB è dato da *R_Q_COUNT_ERR_ENB* (vedere sezione Segnali a pagina 6) così che, una volta che viene abbassato start, si abilita il contatore e una volta raggiunto il numero 31 durante il conteggio, il ciclo di clock successivo viene alzato il segnale di ***end of computation*** e il contatore viene disabilitato.

È stato scelto un reset sincrono in modo tale da poter tenere asserito il segnale di ***end of computation*** per un solo ciclo di clock; infatti, una volta asserito il segnale, il ciclo di clock successivo sul fronte utile viene resettato il componente.

REGISTRO PARALLELO/SERIE PER N

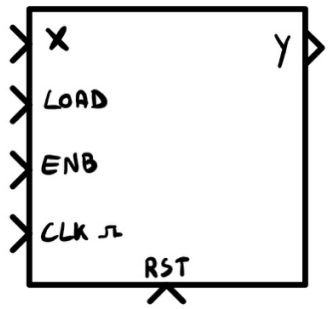


Figura 10

Il seguente modulo (Figura 10) riceve in ingresso:

- un segnale X su 32 bit da memorizzare
- un segnale di LOAD di un bit che permette di selezionare il valore in ingresso ai bistabili
- un segnale di reset RST di un bit che permette di portare l'uscita del modulo a zero
- un segnale di clock CLK di un bit per la sincronizzazione del modulo
- un segnale di enable ENB di un bit che permette di abilitare il modulo

In uscita viene fornito un solo bit Y.

Questo registro presenta un ingresso parallelo su 32 bit; che nel caso del componente divisore corrisponde al dividendo N, e un'uscita seriale Y di un solo bit. Grazie all'ingresso di selezione LOAD è possibile controllare il comportamento dei bistabili, in particolare è possibile selezionare come ingresso di ogni bistabile o il valore proveniente dall'uscita del bistabile immediatamente precedente z_{i-1} o un valore x_i proveniente dall'esterno. Un generico bistabile i appartenente a questo registro viene rappresentato come segue;

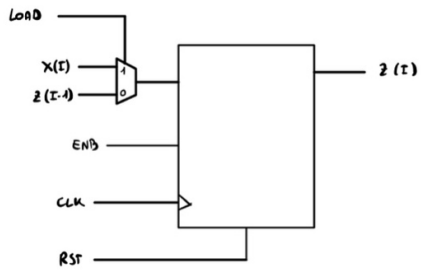


Figura 11

Nel caso del divisore il segnale di LOAD è dato dal segnale di $START$[1], in modo tale da poter memorizzare il valore del dividendo N durante il ciclo di clock in cui START = '1'. Il segnale di ENB è dato da *N_ENB* (vedere sezione Segnali a pagina 6) così da abilitare il registro quando START = '1' fino a quando viene terminata una divisione, ovvero quando viene asserito ***end of computation***. Il segnale di RST è dato da *N_D_COM_ERR_COUNT_RST* (vedere sezione Segnali a pagina 6), ovvero viene portato a '1' all'inizio oppure al termine del calcolo di ogni divisione. Il segnale Y di un bit in uscita corrisponde al *n-esimo* bit del numero N, dove n è la corrente iterazione per il calcolo della divisione. Ad es. durante la prima iterazione, si avrà in uscita N(31) e così via fino a N(0) (vedere passo 3a dell'algoritmo per svolgere la divisione).

Facendo riferimento all'algoritmo presentato nell'introduzione, per memorizzare N è stato scelto un registro parallelo/serie; la scelta è dettata dal fatto che durante ogni iterazione serve un bit di N per effettuare lo ***shift*** di R. Dovendo fornire in uscita un bit alla volta, questo registro risulta essere la scelta adatta.

1. START è il segnale di inizio fornito in input al divisore dall'utente.

REGISTRO PARALLELO/PARALLELO PER D E R

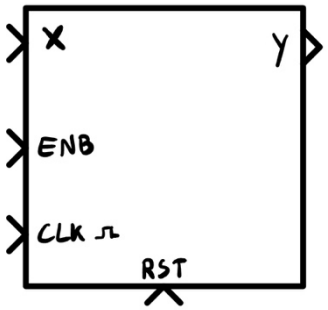


Figura 12

Questo modulo viene utilizzato per istanziare sia il registro per memorizzare D sia il registro per R in quanto per entrambi i segnali viene usato un registro parallelo/parallelo.

In ingresso vengono ricevuti:

- un segnale X a 32 bit da memorizzare
- un segnale di enable ENB di un bit che permette di abilitare il modulo
- un segnale di clock CLK di un bit per la sincronizzazione del modulo
- un segnale di reset RST di un bit che permette di portare l'uscita del modulo a zero.

In uscita viene fornito il valore memorizzato Y su 32 bit.

Un registro parallelo/parallelo riceve in ingresso tutti i bit contemporaneamente, un bit per ogni flip-flop. Tutti i flip-flop sono sincronizzati dallo stesso segnale di clock in modo tale da consentire la lettura in parallelo dei dati. Non appena il segnale di clock presenta un fronte di clock utile, tutti i flip-flop campionano gli ingressi x_i simultaneamente; inoltre, dal momento che anche i dati in uscita y_i sono prelevati dalle uscite Q dei bistabili in parallelo, i bit della parola di uscita sono tutti disponibili allo stesso istante.

Un generico bistabile i appartenente a questo registro viene rappresentato come segue

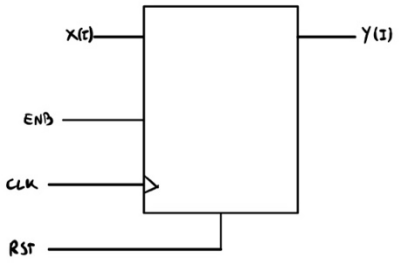


Figura 13

Registro D

Per memorizzare il divisore D è stato scelto un registro parallelo/parallelo in quanto, una volta memorizzato il valore opportuno di D, quest'ultimo deve essere fornito al sottrattore fino al termine del calcolo.

Nel caso del componente divisore, il segnale di ENB del registro D è dato dal segnale di $START^1$; questo permette di salvare il valore di D quando START = '1' e mantenerlo in uscita fino al termine del calcolo della divisione, quando verrà poi resettato. Il segnale di RST è dato da *N_D_COM_ERR_COUNT_RST* (vedere sezione Segnali a pagina 6), in modo tale che il registro venga "resettato" al termine di ogni divisione.

Registro R

Per il resto R è stato scelto un registro parallelo/parallelo per poter memorizzare il suo valore al termine di ogni iterazione e utilizzarlo poi all'interno dello shifter per effettuare successivamente la sottrazione R - D. Il resto R viene fornito anche in uscita al componente divisore; in questo modo, durante il calcolo, verranno forniti i valori che assume R durante ogni iterazione; al termine del calcolo viene fornito in uscita il valore corretto del resto della divisione.

Nel caso del componente divisore il segnale di ENB del registro D è dato dal segnale di *R_Q_COUNT_ERR_ENB* (vedere sezione Segnali a pagina 6); questo per abilitare il registro R soltanto durante il calcolo della divisione; viene quindi abilitato un ciclo di clock dopo $START^1 = '1'$ e viene disabilitato una volta che $EOC^2 = '1'$; in questo modo si ottiene il risultato corretto. Il segnale di RST è dato da $TRST$ (vedere sezione Segnali a pagina 6) in modo tale che il registro venga "resettato" quando l'utente pone $RST^3 = '1'$ oppure ogni volta che vengono salvati i nuovi valori di N e D, ovvero quando $START^1 = '1'$.

1. START è il segnale di inizio fornito in input al divisore dall'utente.

2. EOC è il segnale che indica la fine del calcolo della divisione ed è fornito dal contatore.

3. RST è il segnale di reset fornito in input al divisore dall'utente.

REGISTRO SERIE/PARALLELO PER Q

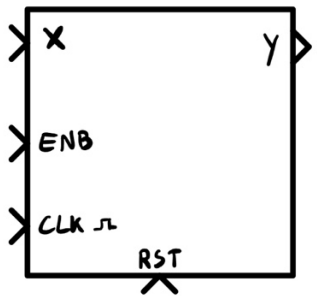


Figura 14

Questo modulo riceve in ingresso:

- un segnale X di un bit da memorizzare
- un segnale di enable ENB di un bit che permette di abilitare il modulo
- un segnale di clock CLK di un bit per la sincronizzazione del modulo
- un segnale di reset RST di un bit che permette di portare l'uscita del modulo a zero.

In uscita viene fornito in parallelo il segnale Y su 32 bit.

Per questo tipo di registro il caricamento dei bit della parola avviene in modo seriale grazie a un singolo ingresso x, mentre la lettura del valore memorizzato avviene in parallelo. Al fine di permettere il caricamento seriale, l'architettura deve essere in grado di immagazzinare un bit alla volta e far scorrere i bit già memorizzati per evitarne la sovrascrittura. La struttura generica di questo componente è la seguente

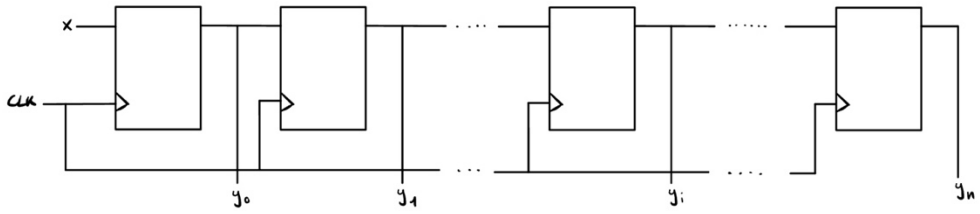


Figura 15

Facendo riferimento alla figura sopra si può notare come il salvataggio in serie è possibile grazie alla connessione in cascata dell'uscita di un bistabile con l'ingresso del successivo: in questo modo, a ogni fronte utile di clock, ogni bistabile campiona il bit 'ingresso e lo mantiene stabile per un ciclo realizzando di fatto uno scorrimento dei dati in ingresso.

Nel caso del componente divisore il bit in ingresso x corrisponde al riporto *COUT* del sottrattore durante la n-esima iterazione (vedere passo 3b dell'algoritmo). Il segnale di ENB del registro Q è dato dal segnale di $R\_Q\_COUNT\_ERR\_ENB$ (vedere sezione Segnali a pagina 6) in modo tale da abilitare il registro R soltanto durante il calcolo della divisione, abilitandolo un ciclo di clock dopo $START^1 = '1'$ e disabilitandolo una volta che $EOC^2 = '1'$; in questo modo durante ogni iterazione si avranno i risultati intermedi assunti da Q, poi una volta finita la divisione si avrà il risultato finale. Il segnale di RST è dato da $TRST$ (vedere sezione Segnali a pagina 6) in modo tale che il registro venga "resettato" quando l'utente pone $RST^3 = '1'$ oppure ogni volta che vengono salvati i nuovi valori di N e D, ovvero quando $START^1 = '1'$.

Facendo riferimento all'algoritmo presentato nell'introduzione, per memorizzare Q è stato scelto un registro serie/parallelo; la scelta è dettata dal fatto che durante ogni iterazione viene prodotto un bit di Q. Dovendo salvare un bit alla volta, questo registro risulta essere la scelta adatta.

1. START è il segnale di inizio fornito in input al divisore dall'utente.
2. EOC è il segnale che indica la fine del calcolo della divisione ed è fornito dal contatore.
3. RST è il segnale di reset fornito in input al divisore dall'utente.

CHECK_ERROR

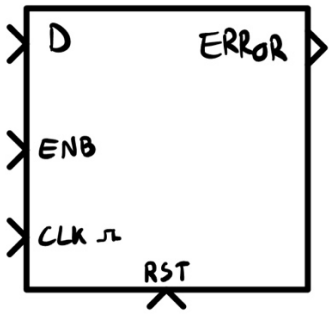


Figura 16

Questo modulo prende in ingresso:

- un segnale D a 32 bit
- un segnale di enable ENB di un bit che permette di abilitare il modulo
- un segnale di clock CLK di un bit per la sincronizzazione del modulo
- un segnale di reset RST sincrono di un bit che permette di portare l'uscita del modulo a zero.

In uscita viene fornito un segnale di ERROR che assume il valore '1' nel caso in cui D fosse uguale a 0, in caso contrario assume valore '0'.

È stato utilizzato per questo componente un flip flop di tipo D in modo tale da poter sincronizzare il controllo dell'errore ed abilitarlo solo dopo che il divisore D è stato salvato all'interno dell'apposito registro. A tale scopo il segnale di ENB è dato da *R_Q_COUNT_ERR_ENB* (vedere sezione Segnali a pagina 6), così da avere il registro abilitato durante il calcolo della divisione. Il segnale di RST è dato da *N_D_COM_ERR_COUNT_RST* (vedere sezione Segnali a pagina 6); in questo modo è possibile "resettare" il segnale un ciclo di clock dopo che è stato asserito ***end of computation***.

È stato scelto un reset sincrono in modo da poter mantenere in uscita il valore di ERROR quando ***end of computation*** è asserito; per poi essere resettato quando quest'ultimo viene abbassato (nel nostro caso dopo un ciclo di clock).

COM_START

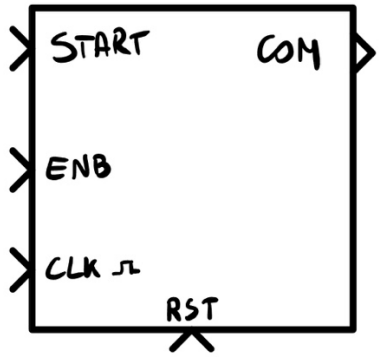


Figura 17

Questo modulo prende in ingresso:

- un segnale START di un bit
- un segnale di enable ENB di un bit che permette di abilitare il modulo
- un segnale di clock CLK di un bit per la sincronizzazione del modulo
- segnale di reset RST di un bit che permette di portare l'uscita del modulo a zero.

In uscita fornisce un segnale COM che vale 1 quando START = '1' e vale 0 quando START = '0'.

Il segnale COM verrà poi utilizzato all'interno del divisore come segnale INSTART, fondamentale per la gestione del segnale di enable di alcuni componenti. Il segnale di ENB di questo modulo è dato da $START$[1] in modo tale che, una volta che START assume il valore 1, il segnale COM mantiene il valore '1' fintantoché non viene poi portato a zero dal segnale di reset. Quest'ultimo è dato da $N\_D\_COM\_ERR\_COUNT\_RST$ (vedere sezione Segnali a pagina 6), così che COM rimanga uguale a 1 fino al termine del calcolo della divisione, ovvero fino a quando viene asserito ***end of computation***.

Questo modulo è stato fatto anche per gestire un caso particolare:

ad esempio, se l'utente inserisse il dividendo N e il divisore D e successivamente $RST^2 = '0'$, fino a che non viene asserito START, i registri R e D salverebbero valori privi di significato. Mentre nel caso in cui start non venisse asserito per 34 cicli di clock, verrebbe alzato il segnale di *end of computation*, nonostante i valori siano privi di significato. Quindi il segnale di INSTART, gestito in maniera opportuna tramite operatori logici, permette di ovviare a questo problema.

1. START è il segnale di inizio fornito in input al divisore dall'utente.
2. RST è il segnale di reset fornito in input al divisore dall'utente.

Capitolo 3

Verifica

Test bench

Il test bench ha lo scopo di verificare la funzionalità di un circuito. Per ottenere tale risultato bisogna generare ingressi significativi; come valori significativi e temporizzazione, e confrontare le corrispondenti uscite con i valori attesi.

Nel caso del divisore è stato utilizzato un clock con duty-cycle del 50% e durata totale di 48 ns in modo tale che $T_{clock} > T_D$, ovvero il clock deve essere maggiore del tempo di ritardo massimo del circuito.

Per quanto riguarda i segnali in ingresso sono state fatte alcune ipotesi:

- Il segnale di START può essere alzato per un solo ciclo di clock;
- Il segnale di RST può essere alzato una sola volta all'inizio.

Casi d'uso

I vari casi d'uso sono:

- Dividendo grande e divisore piccolo

  Input

  NUM = "01000100000011000001000000001000" → 1141641224
  D = "00000000000000000000000000010010" → 18

  Output atteso

  Q = "00000011110001111100100000000000" → 63424512
  R = "00000000000000000000000000001000" → 8
  ERROR = '0'

- Dividendo piccolo e divisore piccolo

  Input

  NUM = “00000000000000000000000000001000” → 8
  D = “00000000000000000000000000000010” → 2

  Output atteso

  Q = “00000000000000000000000000000100” → 4
  R = “00000000000000000000000000000000” → 0
  ERROR = ‘0’

- Dividendo piccolo e divisore nullo

  Input

  NUM = “00000000000000000000000000001000” → 8
  D = “00000000000000000000000000000000” → 0

  Output atteso

  Q = “11111111111111111111111111111111” → 4294967295
  R = “00000000000000000000000000001000” → 8
  ERROR = ‘1’

- Dividendo nullo e divisore piccolo

  Input

  NUM = “00000000000000000000000000000000” → 0
  D = “00000000000000000000000010001000” → 136

  Output atteso

  Q = “00000000000000000000000000000000” → 0
  R = “00000000000000000000000000000000” → 0
  ERROR = ‘0’

- Dividendo grande e divisore grande

  Input

  NUM = “10010000001000111000000000001000” → 2418245640
  D = “01010110000111000000000000000010” → 1444675586

  Output atteso

  Q = “00000000000000000000000000000001” → 1
  R = “00111010000001111000000000000110” → 973570054
  ERROR = ‘0’

- Dividendo minore del divisore (entrambi grandi)

  Input

  NUM = “00010000001000111000000000001000” → 270761992
  D = “11010110000111000000000000000010” → 3592159234

  Output atteso

  Q = “00000000000000000000000000000000” → 0
  R = “00010000001000111000000000001000” → 270761992
  ERROR = ‘0’

- Dividendo piccolo e divisore piccolo

  Input

  NUM = “00000000000000000000000000010001” → 17
  D = “00000000000000000000000000000010” → 2

  Output atteso

  Q = “00000000000000000000000000001000” → 8
  R = “00000000000000000000000000000001” → 1
  ERROR = ‘0’

- Dividendo piccolo e divisore grande

  Input

  NUM = “00000000000000000000000000010000” → 16
  D = “00100001000100100111010001100010” → 554857570

  Output atteso

  Q = “00000000000000000000000000000000” → 0
  R = “00000000000000000000000000010000” → 16
  ERROR = ‘0’

- Dividendo nullo e divisore nullo

  Input

  NUM = “00000000000000000000000000000000” → 0
  D = “00000000000000000000000000000000” → 0

  Output atteso

  Q = “11111111111111111111111111111111” → 4294967295
  R = “00000000000000000000000000000000” → 0
  ERROR = ‘1’

- Dividendo e divisore uguali (grandi)

  Input

  NUM = “01110000100000010001111000010000” → 1887510032
  D = “01110000100000010001111000010000” → 1887510032

  Output atteso

  Q = “00000000000000000000000000000001” → 1
  R = “00000000000000000000000000000000” → 0
  ERROR = ‘0