**Progetto di reti Logiche**

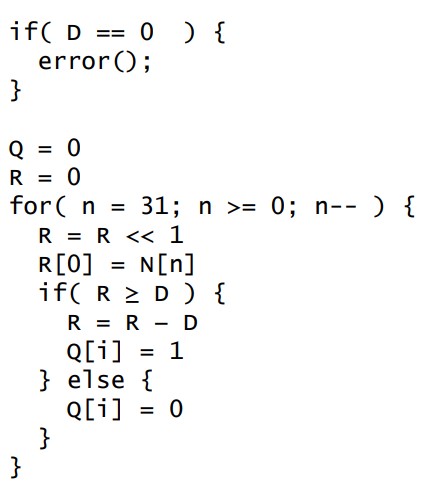
[Divisore intero con resto]

[Pedroni Simone]

# Introduzione

Si vuole progettare una rete sequenziale che implementi un **divisore intero con resto** su 32 bit, basato sul metodo detto di “divisone lunga”.

Definendo con *N* il dividendo, *D* il divisore, *Q* il quoziente ed *R* il resto, l’algoritmo iterativo usato per il calcolo è descritto dal seguente pseudocodice:



IPOTESI. Per quanto riguarda il calcolo effettuato dal componente, si ipotizza l’utilizzo delle seguenti definizioni del quoziente *Q* e del resto *R*:

Siano *x* un numero reale, mentre *n* e *d* due numeri interi. Si definisce, allora, la funzione *TRUNC* nel seguente modo:

Indicando come *a/b* la divisione fra numeri reali, è possibile definire nel seguente modo l’operazione **div** (*divisione intera*) come:

Esempi esplicativi:

e l’operazione **rem** (*resto*) come:

Esempi esplicativi:

# Specifica

## Interfaccia del sistema

L’interfaccia del divisore si presenta nel seguente modo:

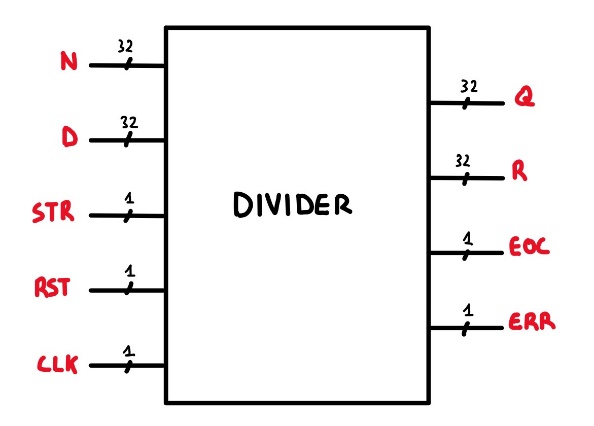


Figura 1. *Divisore*

**Lista dei segnali primari**:

* N: segnale d’ingresso che rappresenta il dividendo inserito dall’utente; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit interpretato tramite codifica a complemento a 2;
* D: segnale d’ingresso che rappresenta il divisore inserito dall’utente; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit interpretato tramite codifica a complemento a 2;
* STR (*Start*): segnale d’ingresso che, quando attivo alto, permette all’utente di effettuare una nuova computazione;
* RST (*Reset*): segnale d’ingresso che, quando attivo alto, permette di resettare l’architettura;
* CLK: segnale d’ingresso che definisce l’andamento del clock
* Q: segnale d’uscita che rappresenta il quoziente tra il dividendo *N* ed il divisore *D*; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit interpretato tramite codifica a complemento a 2 che segue la convezione di cui sopra;
* R: segnale d’uscita che rappresenta il resto tra il dividendo *N* ed il divisore *D*; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit interpretato tramite codifica a complemento a 2 che segue la convenzione di cui sopra;
* EOC (*End Of Computation*): segnale d’uscita che, quando alto, notifica l’utente della fine della computazione precedente e la possibilità di avviarne una successiva;
* ERR (*Error*): segnale d’uscita che, quando alto, notifica l’utente della condizione di errore sul divisore *D*, ovvero quando questo viene posto in ingresso pari a 0.

**Modalità di utilizzo del sistema**

1. **Utilizzo del segnale di reset RST**:

Il segnale di *Reset* RST è un segnale d’ingresso attivo alto che viene utilizzato dall’utente nel momento in cui questo vuole resettare l’architettura del componente. In particolare, si adotta la convenzione riportata in *figura 2*:

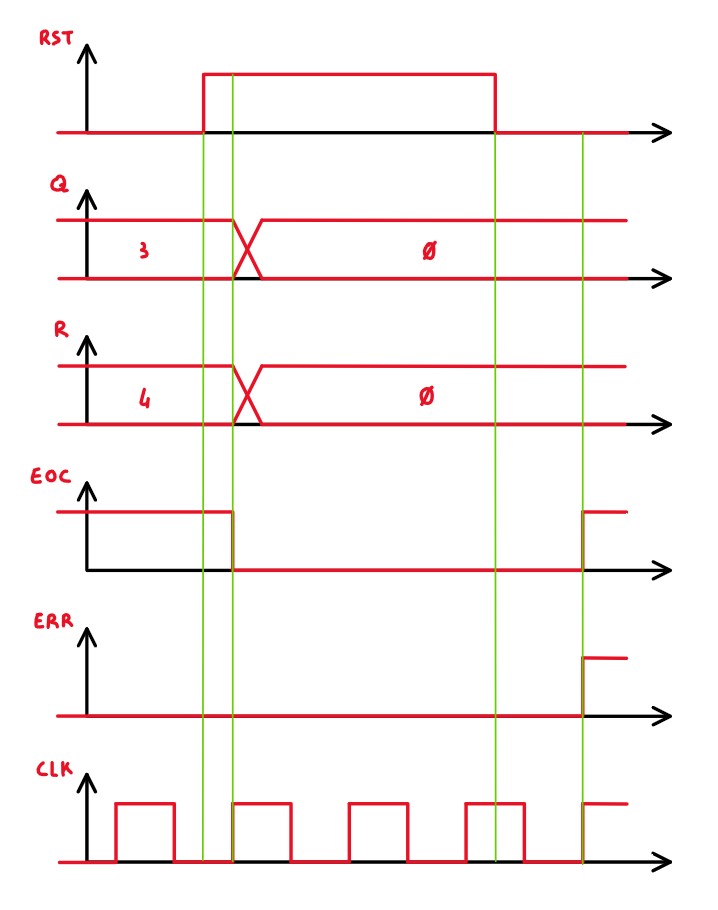


Figura 2. *Convenzione di utilizzo sul Reset* – Se il segnale di RST è alto, tutti i segnali di uscita sono posti pari a 0

1. **Inserimento dei dati in ingresso e utilizzo del segnale di start STR**:

Nel momento in cui l’utente vuole calcolare il quoziente *Q* ed il resto *R* di due numeri, deve effettuare l’inserimento dei nuovi dati in ingresso di cui vuole conoscere il risultato; in particolare, deve modificare il valore dei segnali d’ingresso che rappresentano il *dividendo* N ed il *divisore* D. Tuttavia, la computazione, ovvero il calcolo del quoziente *Q* e del resto *R* relativi ai dati inseriti, non può avere inizio fintanto che il segnale di *Start* STR è basso. L’utente può, di conseguenza, decidere quando far cominciare la computazione dei dati inseriti ponendo il segnale di *Start* STR alto per un ciclo di *clock*, come mostrato nella successiva *figura 3*.

*Nota*. Se una computazione è già in corso, il segnale di *Start* STR non avrà, come effetto, l’inizio della computazione successiva; infatti, bisogna aspettare che la computazione precedente sia giunta al termine per poterne iniziarne un’altra. Ciò significa che i dati inseriti in ingresso al componente durante una computazione in corso, se modificati più volte, andranno perduti.

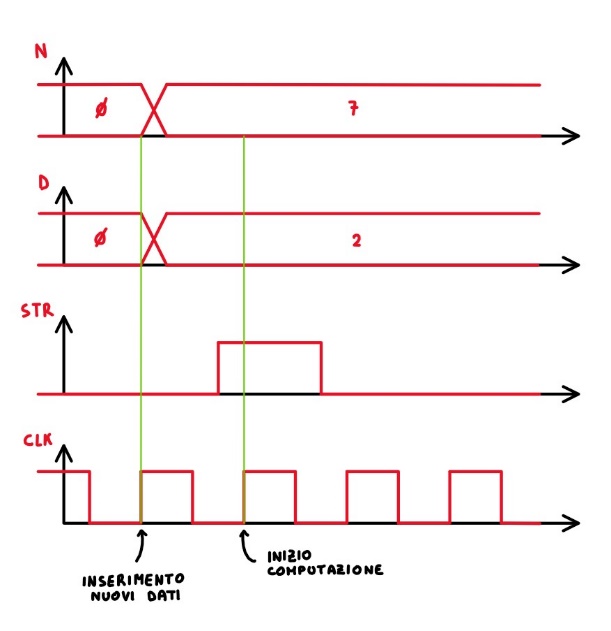


Figura 3. *Utilizzo dello Start e inizio della computazione*

1. **Lettura dei dati in uscita e utilizzo del segnale di End Of Computation EOC**:

La computazione di due numeri, il dividendo *N* e il divisore *D*, voluta dall’utente ha inizio quando quest’ultimo alza il segnale di Start *STR*. I relativi risultati, ovvero il *quoziente* Q ed il *resto* R dei numeri in ingresso, vengono presentati all’uscita dell’architettura, salvo errori per i quali si rende necessario l’utilizzo del segnale di Reset *RST*, dopo un tempo pari a 32 cicli di *clock* dall’inizio della computazione. Non è necessario, tuttavia, che l’utente tenga conto del numero di cicli di *clock* trascorsi dall’inizio del calcolo, bensì viene messo a disposizione dello stesso un apposito segnale, il segnale di *End Of Computation* EOC, il quale indica se è in corso una computazione oppure no.

In particolare, quando il segnale *End Of Computation* EOC è alto significa, in primo luogo, che la computazione precedente è terminata e l’utente può quindi leggere i relativi dati presenti sull’uscita, ma indica anche che è ora possibile eseguire il prossimo calcolo. Al contrario, quando il segnale *End Of Computation* EOC è basso si notifica l’utente che una computazione è già in corso, perciò non è possibile avviarne un’altra, anche con l’utilizzo del segnale di Start *STR*.

*Nota*. Dopo aver fatto uso del segnale di Reset *RST* per resettare lo stato dell’architettura, il segnale *End Of Computation* EOC risulterà alto al fine di indicare all’utente che il componente è nuovamente disponibile ad effettuare una computazione (si veda la *figura 2*).

L’utilizzo del segnale *End Of Computation* EOC è meglio illustrato dalla seguente *figura 4*:

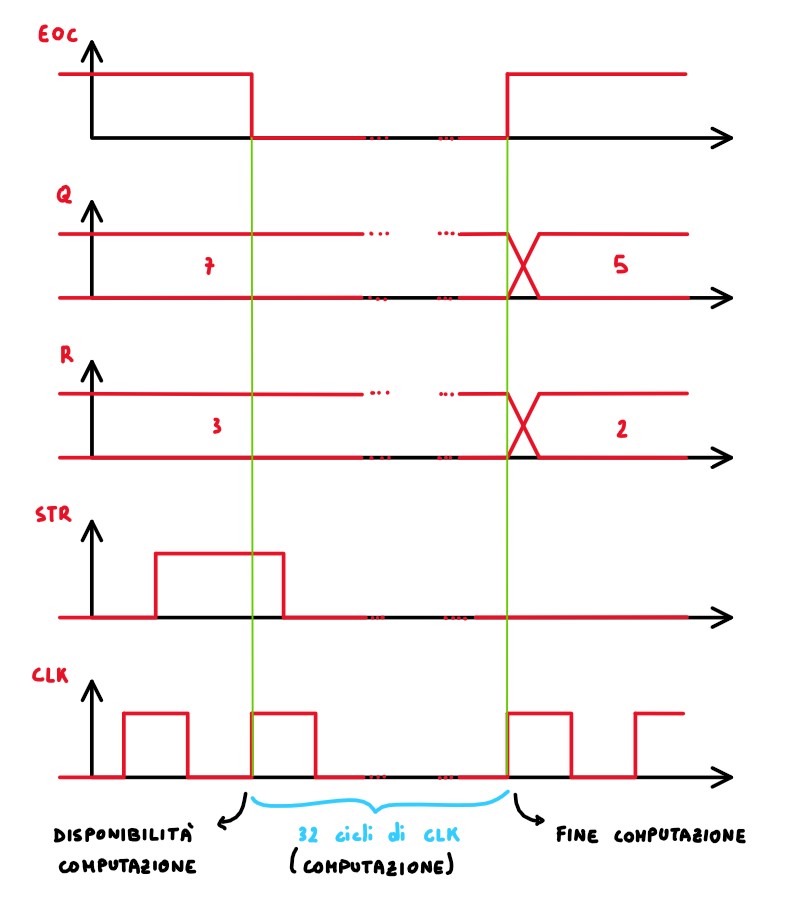


Figura 4. *Utilizzo dell’End Of Computation e disponibilità della computazione*

1. **Condizione d’errore e utilizzo del segnale di Errore ERR**:

Nel caso in cui l’utente inserisca come divisore *D* il numero zero, l’architettura notifica l’impossibilità di effettuare la divisione per zero attraverso il segnale *Error* ERR.

In particolare, una volta terminata la computazione di tale condizione, il segnale *Error* ERR si porterà alto come mostrato in *figura 5*.

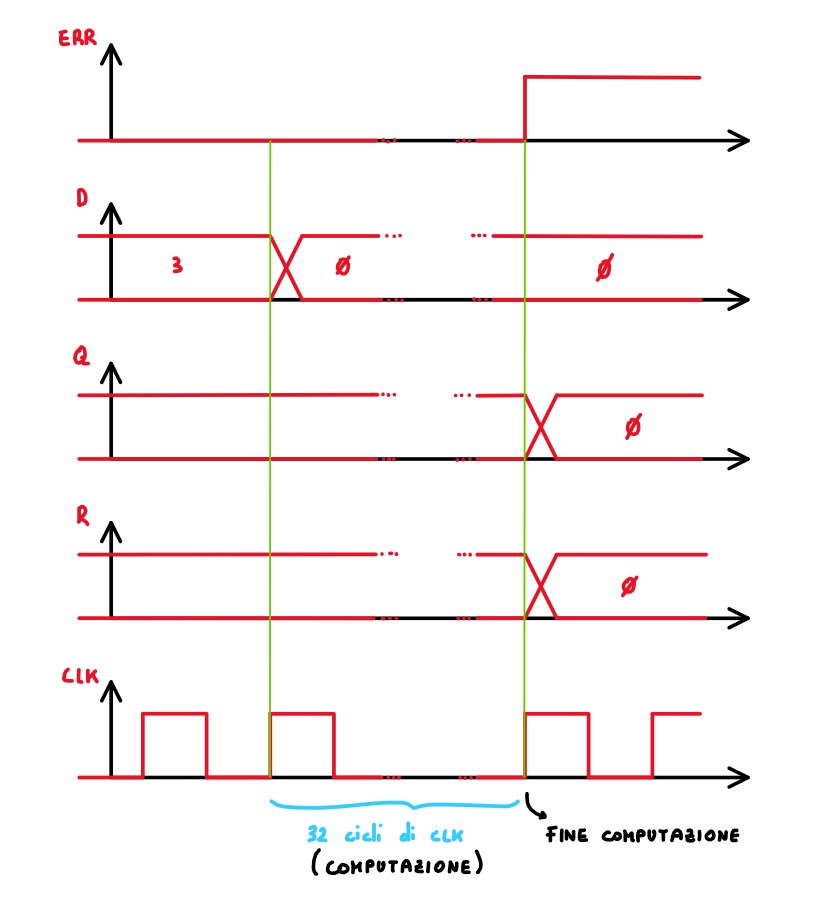


Figura 5. *Convenzione di utilizzo sull’Errore* - Se il segnale di ERR è alto, i segnali di uscita del quoziente Q e del resto R sono posti pari a 0

*Nota*. Dopo aver fatto uso del segnale di Reset *RST* per resettare lo stato dell’architettura, il segnale *Error* ERR risulterà alto in accordo col fatto che, dopo aver resettato, l’architettura vede come divisore *D* il numero 0 (si veda la *figura 2*).

## Architettura del sistema

L’architettura del divisore, nelle sue generalità, si presenta come di seguito illustrato:

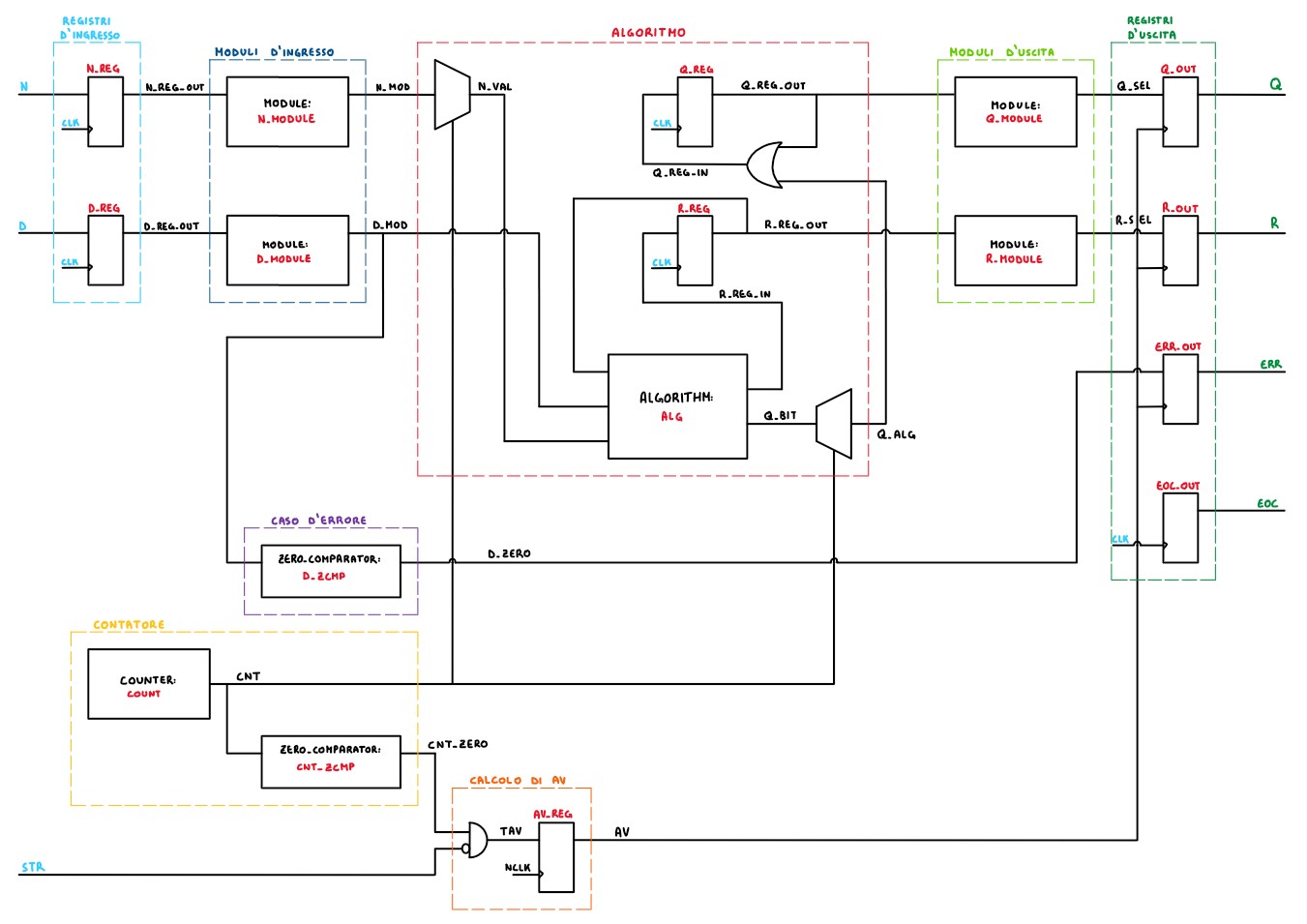


Figura 6. *Architettura del divisore*

La *figura 6* mostra come l’architettura del componente possa essere concettualmente suddivisa in varie parti, di seguito approfondite:

1. **REGISTRI D’INGRESSO**: registri posti all’inizio dell’architettura per permettere all’utente, quando non sia già in corso una computazione, di inserire nuovi dati in ingresso, il dividendo *N* ed il divisore *D*, di cui si vuole conoscere il risultato;
2. **MODULI D’INGRESSO**: i componenti *MODULE* (vedi sotto) sono moduli che prendono in ingresso i dati inseriti dall’utente e che vengono utilizzati per calcolarne il modulo, necessario per la computazione degli stessi. Questa operazione si rende necessaria in quanto il calcolo del quoziente *Q* e del resto *R* viene effettuato tramite i moduli del dividendo *N* e del divisore *D*; perciò, se questi ultimi sono positivi, vengono utilizzati direttamente, altrimenti, se negativi, è necessario calcolarne il complemento a due per ottenerne il modulo;
3. **CASO D’ERRORE**: il modulo del divisore *D* viene anche impiegato per verificare la condizione di errore, ovvero quando il divisore *D* risulta pari a zero. Questa verifica viene effettuata tramite il modulo *ZERO\_COMPARATOR* (vedi sotto);
4. **ALGORITMO**: questa parte dell’architettura è composta da diversi componenti: innanzitutto, il *MUX* utilizzato per la selezione del bit *N\_VAL* del dividendo *N* necessario alla computazione, e il *DEMUX*, usato invece per la creazione del segnale *Q\_ALG*, il quale indica il bit del quoziente *Q* appena calcolato. Si rendono necessari, poi, i registri intermedi del quoziente *Q* e del resto *R* al fine di memorizzare i valori dei due numeri durante ogni passo della computazione. Infine, si utilizza il componente *ALGORITHM* (vedi sotto) per calcolare i nuovi valori del resto *R* e del bit *Q\_BIT* del quoziente *Q* ad ogni passo della computazione;
5. **CONTATORE**: il contatore decrescente a modulo 32 *COUNTER* viene utilizzato per “tenere traccia” dei passi della computazione, infatti la sua uscita, il segnale *CNT*, funge da selettore per il *MUX* ed il *DEMUX*. Inoltre, anche in questo caso si rende necessario l’utilizzo di un modulo *ZERO\_COMPARATOR*, il quale verifica che l’uscita del contatore sia nello stato di Reset, ovvero pari a 31;
6. **CALCOLO DI AV**: il segnale *AV* indica se è in corso una computazione oppure no; in particolare, se *AV* è alto significa che l’architettura è disponibile per un nuovo calcolo dei dati in ingresso, mentre se *AV* è basso si rappresenta la situazione nella quale è in corso una computazione. Il calcolo del segnale *AV* viene effettuato tramite un apposito registro;
7. **MODULI D’USCITA**: i componenti *MODULE* vengono utilizzati anche per effettuare la selezione della corretta espressione dei dati in uscita quali il quoziente *Q* ed il resto *R*. Infatti, se questi ultimi devono risultare positivi, si seleziona direttamente l’uscita dei registri intermedi utilizzati durante la procedura dell’algoritmo, altrimenti, se devono invece essere considerati negativi, si necessita del loro complemento a due;
8. **REGISTRI D’USCITA**: registri posti alla fine dell’architettura per permettere all’utente di leggere i dati in uscita risultanti dalla computazione dei relativi dati in ingresso.

## Modulo ALGORITHM

Il modulo ALGORITHM presenta la struttura mostrata in *figura 7*, nella quale i segnali d’ingresso vengono raffigurati in **azzurro**, mentre per quelli d’uscita si utilizza il colore **verde**.

**Lista dei segnali I/O**:

* **RIN**: segnale d’ingresso che rappresenta il valore del resto *R*, memorizzato nell’apposito registro intermedio, che deve essere utilizzato per il calcolo dell’attuale computazione; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit;

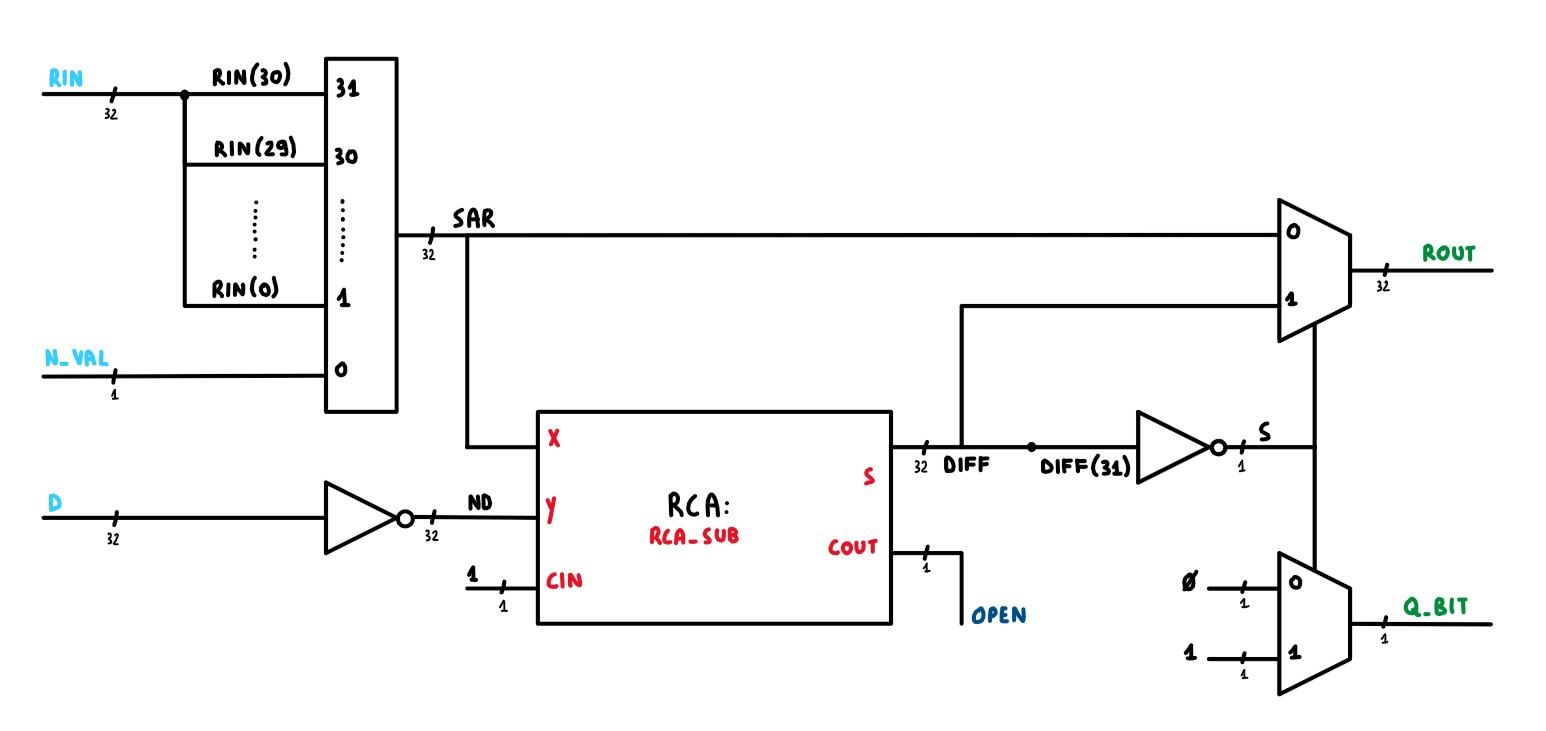


Figura 7. *Struttura del modulo Algorithm*

* **D**: segnale d’ingresso che rappresenta il modulo del divisore *D*, il quale deve essere sottratto dal valore corrente del segnale *SAR* (vedi paragrafo successivo); nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit;
* **N\_VAL**: segnale d’ingresso che rappresenta il bit del dividendo *N* selezionato per l’attuale passo della computazione; nel linguaggio VHDL viene definito dal tipo *std\_logic*; perciò, si tratta di un segnale di 1 bit;
* **ROUT**: segnale d’uscita che rappresenta il valore del resto *R* calcolato durante l’attuale computazione e che costituisce l’ingresso del relativo registro intermedio; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit;
* **Q\_BIT**: segnale d’uscita che rappresenta il bit del quoziente *Q* calcolato durante l’attuale computazione; nel linguaggio VHDL viene definito dal tipo *std\_logic*; perciò, si tratta di un segnale di 1 bit.

**Comportamento del modulo**:

Il modulo ha la funzione di implementare la parte dell’algoritmo iterativo che sta alla base del metodo di “divisione lunga” di seguito riportata:

Immagine che contiene testo

Descrizione generata automaticamente

Al fine di implementare tale porzione di pseudocodice attraverso un apposito circuito è necessario, innanzitutto, calcolare il valore del resto *R* che verrà successivamente confrontato con il modulo del divisore *D*: per fare ciò, bisogna effettuare le operazioni di *shifting* e di *assegnamento*, le quali sono rappresentate dal segnale *SAR* (*Shifted and Assigned R*), composto dai primi 31 bit del segnale d’ingresso *RIN* e dal bit *N\_VAL* selezionato dal modulo del dividendo *N* come bit meno significativo.

In secondo luogo, la decisione su quale ramo dello *statement if* seguire viene effettuata a seconda del risultato della differenza tra l’espressione appena calcolata del resto *R*, rappresentata dal segnale *SAR*, e il modulo del divisore *D*. Per svolgere tale differenza viene impiegato un RCA (*Ripple Carry Adder*), istanziato in questa sede con il nome *RCA\_SUB*, al quale vengono forniti in ingresso il minuendo, ovvero il segnale *SAR*, ed il sottraendo, la realizzazione del quale viene effettuata negando il modulo del divisore *D*, che dà origine al segnale *ND* (*Negated D*) e rappresenta l’ingresso Y, al quale, poi, si deve sommare 1, considerato come il riporto in ingresso *CIN* del componente.

L’uscita dell’RCA fornisce il segnale *DIFF*, il quale rappresenta la differenza appena calcolata; attraverso quest’ultimo viene effettuata la decisione sul ramo dello *statement if* da seguire: in particolare, questa viene presa sulla base del valore negato del bit più significativo del segnale, al quale viene assegnato il nominativo *S*.

Nel caso in cui la differenza risulti un numero positivo, ovvero il bit più significativo del relativo segnale *DIFF* sia pari a 0, consegue che *S* risulta pari a 1, perciò viene selezionato il primo ramo dello *statement if*, per il quale il segnale d’uscita *ROUT* assume il valore dello stesso segnale differenza *DIFF* calcolato in precedenza, mentre *Q\_BIT* risulta alto. Al contrario, quando la differenza risulta negativa, cioè il bit più significativo di *DIFF* sia pari a 1, segue che *S* assume valore 0; di conseguenza, il segnale d’uscita *ROUT* assume il valore del segnale *SAR*, mentre *Q\_BIT* prende il valore 0, a rappresentare il secondo ramo dello *statement if*.

## Modulo MODULE

Il modulo MODULE presenta la struttura mostrata in *figura 8*, nella quale i segnali d’ingresso vengono raffigurati in **azzurro**, mentre per quelli d’uscita si utilizza il colore **verde**.

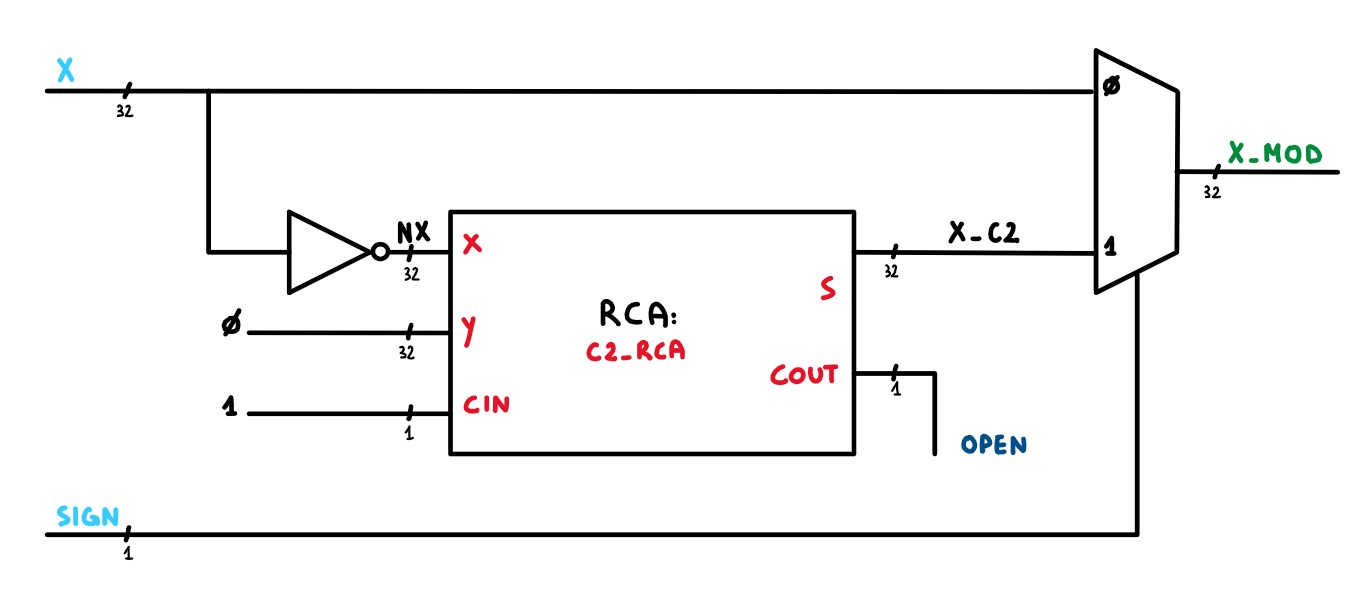


Figura 8. *Struttura del modulo Module*

**Lista dei segnali I/O**:

* **X**: segnale d’ingresso che rappresenta il numero di cui si vuole selezionare la rappresentazione positiva o negativa a seconda del segno; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit;
* **SIGN**: segnale d’ingresso che rappresenta il segno grazie al quale è possibile selezionare una delle due rappresentazioni; nel linguaggio VHDL viene definito dal tipo *std\_logic*; perciò, si tratta di un segnale di 1 bit;
* **X\_MOD**: segnale d’uscita che rappresenta la rappresentazione selezionata a seconda del segno; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(31 downto 0)*, perciò si tratta di un segnale a 32 bit.

**Comportamento del modulo**:

Il modulo è atto a selezionare la rappresentazione corretta e voluta del segnale d’ingresso *X*. Infatti, l’uscita del componente, ovvero il segnale d’uscita *X\_MOD*, può assumere direttamente il valore di *X* quando si pone basso il segnale *SIGN*; altrimenti, in caso contrario, è necessario calcolare la negazione del segnale d’ingresso attraverso un apposito RCA (*Ripple Carry Adder*), denominato *C2\_RCA*, il quale fornisce il segnale *X\_C2*, selezionato quando *SIGN* risulta pari a 1.

*Nota*. All’interno dell’architettura del divisore, il componente *MODULE* viene riutilizzato diverse volte: in primo luogo, è necessario selezionare i moduli del dividendo *N* e del divisore *D*, necessari per la computazione; perciò, se i due numeri in ingresso sono positivi, vengono selezionati direttamente, altrimenti si rende necessario negarli per ottenerne il modulo. Inoltre, *MODULE* viene impiegato anche per selezionare la corretta rappresentazione dei dati in uscita, il quoziente *Q* ed il resto *R*, sulla base del loro segno.

## Modulo ZERO\_COMPARATOR

Il modulo ZERO\_COMPARATOR presenta la struttura mostrata in *figura 9*, nella quale i segnali d’ingresso vengono raffigurati in **azzurro**, mentre per quelli d’uscita si utilizza il colore **verde**.

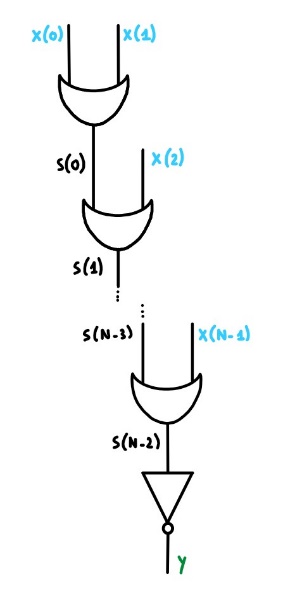


Figura 9. *Struttura del modulo Zero\_Comparator*

**Lista dei segnali I/O**:

* **X**: segnale d’ingresso che rappresenta il numero di cui si vuole verificare l’uguaglianza rispetto a zero; nel linguaggio VHDL viene definito dal tipo *std\_logic\_vector(N-1 downto 0)*, perciò si tratta di un segnale a *N* bit, dove *N* è un numero generico che può essere specificato durante l’istanziazione del componente;
* **Y**: segnale d’uscita che indica se l’uguaglianza risulta verificata; nel linguaggio VHDL viene definito dal tipo *std\_logic*; perciò, si tratta di un segnale di 1 bit.

**Comportamento del modulo**:

Il modulo viene impiegato per verificare se un numero in ingresso *X*, composto da un generico e assegnabile numero di bit *N*, risulta essere pari a zero, ovvero se ogni suo bit è 0. Per fare ciò si impiegano una serie di porte OR in cascata tra loro; in tal modo, se tutti i bit del segnale d’ingresso *X* sono 0, il risultato della computazione delle OR in cascata è ancora 0, mentre, se almeno un bit di *X* risulta pari a 1, l’uscita della cascata è ancora 1.

L’uscita *Y* del componente assume valore alto nel caso in cui la condizione risulti verificata, ovvero se tutti i bit del segnale d’ingresso *X* sono 0 e l’uscita della cascata è 1, mentre assume valore pari a 1 in caso contrario. Il segnale d’uscita *Y*, perciò, non è altro che la negazione del risultato delle porte OR in cascata.

*Nota*. All’interno dell’architettura del divisore, il componente *ZERO\_COMPARATOR* viene impiegato un paio di volte. Per prima cosa, il modulo risulta necessario per verificare il caso di errore in cui il divisore *D* sia pari a zero. In secondo luogo, viene utilizzato per appurare se l’uscita del contatore risulta pari a 31, ovvero nel suo stato di Reset.

# Verifica

## Test-bench

Il Test-bench *TB\_DIVIDER*, attuo a verificare la correttezza del sistema, ovvero del divisore, è principalmente caratterizzato dal processo di stimolo dell’architettura stessa, denominato *STIM\_process*. All’interno di questo processo, vengono simulati diversi casi d’uso del sistema, descritti nel prossimo paragrafo.

## Casi d’uso

Dopo aver inizializzato l’architettura del divisore e averla portata nel suo stato di reset tramite l’attivazione del segnale di *RST* alto, convenzionalmente per un tempo pari a 500 ns, si considerano i seguenti casi d’uso del sistema:

* **CASO 1** **-** Numero basso negativo su numero basso positivo: vengono inseriti come dividendo *N* il numero -8, mentre come divisore *D* il numero +3. Quello che l’architettura produce in uscita sono i numeri -2 come quoziente *Q*, -2 come resto *R* ed *ERR* pari a 0.

*Nota*. Questo caso d’uso serve anche per verificare come la divisione tra un dividendo *N* negativo e un divisore *D* positivo e diverso da zero, produca in uscita un quoziente *Q* ed un resto *R* entrambi negativi.

* **CASO 2 –** Numero medio positivo su numero basso positivo: vengono inseriti come dividendo *N* il numero +134.532, mentre come divisore *D* il numero +8. Quello che l’architettura produce in uscita sono i numeri +16.816 come quoziente *Q*, +4 come resto *R* ed *ERR* pari a 0.

*Nota*. Questo caso d’uso serve anche per verificare come la divisione tra un dividendo *N* positivo e un divisore *D* positivo e diverso da zero, produca in uscita un quoziente *Q* ed un resto *R* entrambi positivi.

* **CASO 3 –** Numero alto positivo su numero alto negativo: vengono inseriti come dividendo *N* il numero +2.113.879.799, mentre come divisore *D* il numero -932.691.771. Quello che l’architettura produce in uscita sono i numeri -2 come quoziente *Q*, +248.496.257 come resto *R* ed *ERR* pari a 0.

*Nota*. Questo caso d’uso serve anche per verificare come la divisione tra un dividendo *N* positivo e un divisore *D* negativo e diverso da zero, produca in uscita un quoziente *Q* negativo ed un resto *R* positivo. Inoltre, lo stesso test viene utilizzato anche per mostrare come la computazione non possa avere inizio fintanto che non si ponga il segnale *STR* pari a 1.

* **CASO 4 –** Numero alto negativo su numero medio negativo: vengono inseriti come dividendo *N* il numero -2.006.236.987, mentre come divisore *D* il numero -1.229.065. Quello che l’architettura produce in uscita sono i numeri +1.632 come quoziente *Q*, -402.907 come resto *R* ed *ERR* pari a 0.

*Nota*. Questo caso d’uso serve anche per verificare come la divisione tra un dividendo *N* negativo e un divisore *D* negativo e diverso da zero, produca in uscita un quoziente *Q* positivo ed un resto *R* negativo.

* **CASO 5 –** Dividendo *N* più piccolo in modulo del divisore *D*: vengono inseriti come dividendo *N* il numero +180, mentre come divisore *D* il numero -457. Quello che l’architettura produce in uscita sono i numeri 0 come quoziente *Q*, +180 come resto *R* ed *ERR* pari a 0.

*Nota*. Questo caso d’uso serve per verificare come la divisione tra un dividendo *N* minore in modulo rispetto al divisore *D* diverso da zero, produca in uscita un quoziente *Q* pari a zero ed un resto *R* pari al dividendo *N*.

* **CASO 6 –** Condizione d’errore: vengono inseriti come dividendo *N* il numero -97.814.939, mentre come divisore *D* il numero 0. Quello che l’architettura produce in uscita, seguendo la convezione adottata in caso d’errore, sono i numeri 0 come quoziente *Q* e resto *R* ed *ERR* pari a 1, ad indicare la condizione d’errore.
* **CASO 7 –** Utilizzo del Reset durante una computazione: dopo l’inizio di una computazione e prima che la stessa termini, viene attivato, ponendo pari a 1, il segnale *Reset* RST al fine di resettare lo stato del sistema. L’architettura, appena osserva il segnale di *Reset* RST alto, pone tutti i segnali d’uscita pari a 0, seguendo la convenzione adottata in caso di Reset.
* **CASO 8 –** Utilizzo del Reset alla fine di una computazione: dopo la fine della computazione precedente e prima che cominci la successiva, viene attivato, ponendolo pari a 1, il segnale di *Reset* RST al fine di resettare lo stato del sistema. L’architettura, seguendo la convenzione adottata in caso di Reset, pone tutti i segnali d’uscita pari a 0.
* **CASO 9 –** Utilizzo dello Start dopo il Reset dell’architettura: dopo il Reset dello stato del sistema, viene effettuata una nuova computazione ponendo il segnale di *Start* STR alto. Questo caso d’uso serve per verificare come la prima computazione dopo il Reset dell’architettura venga effettuata sugli ultimi due numeri inseriti come dividendo *N* e divisore *D*.
* **CASO 10 –** Inserimento e computazione degli ultimi numeri inseriti in ingresso: questo caso d’uso serve per verificare come un secondo e nuovo inserimento degli ultimi dati in ingresso, produca una nuova computazione, la quale fornisce in uscita gli stessi risultati.
* **CASO 11 –** Inserimento e computazione di due numeri durante un’altra computazione in corso: questo caso d’uso serve per verificare come, se una computazione è già in corso, l’inserimento di due nuovi numeri durante la stessa non fa iniziare una nuova computazione, anche ponendo a 1 il segnale di *Start* STR.