Progetto di reti Logiche

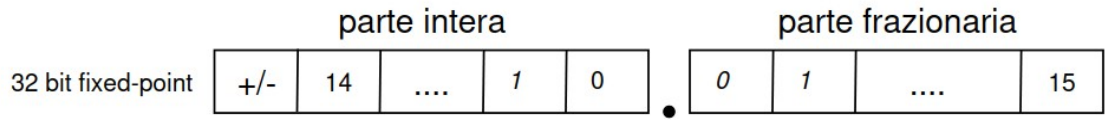
Filtro esponenziale fixed-point

Abate Kevin Pio

Indice

Introduzione

L'obiettivo di questo progetto è implementare un **circuito sequenziale** per la realizzazione di un *Filtro esponenziale* del *primo ordine* con lo scopo di elaborare una sequenza continua di campioni digitali di 32 bit fixed-point, il quale in *complemento a due* ha la seguente rappresentazione:



Come si può vedere dall'immagine il MSB indica il segno, i successivi 15 bit rappresentano la parte intera ed i 16 seguenti la parte frazionaria.

Il filtro elabora l'uscita Y_t in base all'ingresso corrente X_t e ai valori di uscita precedenti Y_{t-1} secondo la seguente equazione:

$$Y_t = \alpha X_t + \alpha(1-\alpha) Y_{t-1} + (1-\alpha)^2 Y_{t-1} .$$

Il coefficiente del filtro α è limitato alle potenze negative del 2:

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

dove K è un valore fornito in ingresso, limitato nell'intervallo [0 .. 7] , il quale si ipotizza costante durante l'elaborazione di una sequenza di ingressi.

Fixed-point

Fixed-point è un metodo per rappresentare i numeri frazionari, assegnando un predefinito numero di bit per la parte frazionaria (nel nostro caso in codifica binaria).

Le operazioni tra i binari fixed-point avverranno come di consuetudine con l'unica differenza che si dovranno compiere tutte le operazioni necessarie per centrare la virgola fissa durante le operazioni, per non perdere il significato della rappresentazione.

Specifica

Il filtro produce in uscita il segnale filtrato Y_t, calcolato secondo la seguente formula:

$$Y_t = \alpha X_t + \alpha(1-\alpha)Y_{t-1} + (1-\alpha)^2 Y_{t-1}$$

La progettazione inizia dalla sintetizzazione dell'equazione del filtro tramite i seguenti calcoli algebrici:

$$Y_t = \alpha X_t - \alpha^2 Y_{t-1} + \alpha Y_{t-1} + \alpha^2 Y_{t-1} - 2\alpha Y_{t-1} + Y_{t-1}$$

I termini di secondo ordine si elidono semplificando l'equazione, ottenendo:

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

Eseguendo il **complemento a due** del numero negativo si prosegue con la sintetizzazione:

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

Ora considerando che α è una potenza negativa del due, con equazione:

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

Si nota che i calcoli si possono semplificare tramite degli **shift right** di k posizioni, l'equazione quindi si può riscrive come:

$$Y_t = X_t \gg K + Y_{t-1} + (\neg Y_{t-1} + 1) \gg K$$

In questo modo la funzione risulta notevolmente semplificata.

Si può notare dall'equazione che i punti focali nella costruzione di questo circuito sono:

- l'operazione di ***shift right*** dei segnali X_t e Y_{t-1}, con particolare attenzione al *riporto del bit di segno* e *all'accentramento* della virgola fissa del numero shiftato
- la ***somma*** dei 3 numeri binari, la quale è operata tramite un *CSA*
- il ***complemento a due*** di Y_{t-1} per la rappresentazione del *numero negativo*

È importante precisare che la rete esegue il calcolo dell'equazione in un **singolo ciclo di clock**.

L'implementazione di queste funzioni è trattata in particolare nel paragrafo riguardante *l'architettura del sistema*.

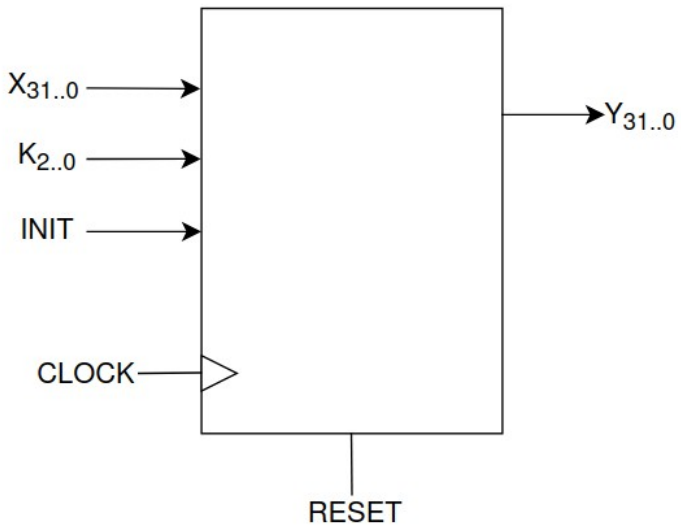
Il sistema inoltre ha un segnale di "*inizializzazione*", con il nome di "**INIT**", che permette di:

- riportare lo stato del sistema a quello iniziale, cioè con *l'uscita* pari a 0
- *resettare tutti i campionatori*
- permette il cambiamento del segnale **K** all'interno del circuito, il quale non è possibile modificare durante l'esecuzione della rete

Interfaccia del sistema

Il sistema è dotato dei seguenti ingressi:

- X_t, un segnale composto da *32 bit fixed-point*, è l'ingresso su cui lavora il circuito ad ogni ciclo di clock
- **K** è un numero limitato nell'intervallo [0 .. 7], quindi è codificato su 3 bit, determina la potenza negativa del 2 del coefficiente α
- **INIT** è un segnale asincrono di 1 bit ed è utilizzato per "*inizializzare*" e *resettare* il sistema
- **CLOCK**, il segnale di Clock



Mentre l'uscita è:

- Y_t **,** un valore di 32 bit fixed-point pari al *risultato* dell'equazione

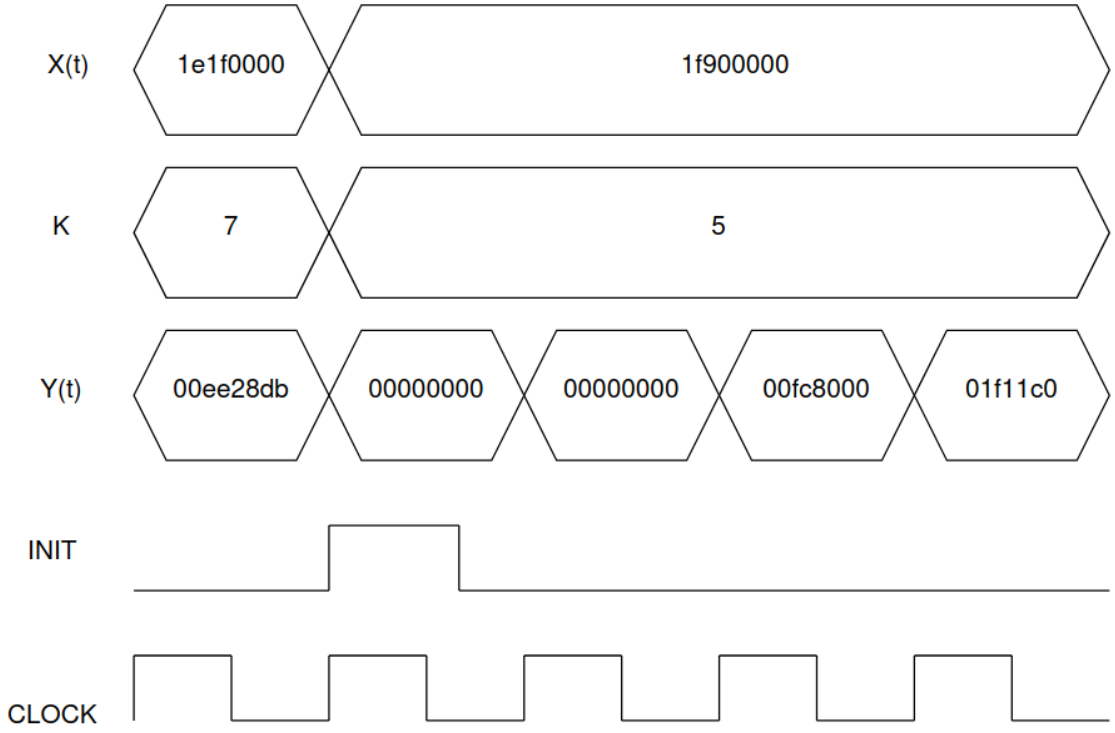
Utilizzo:

Per attivare il dispositivo è necessario aver completato la fase di configurazione, quindi rendere alto il segnale di ***INIT,*** quest'ultimo porterà il filtro in uno stato in cui il sistema avrà tutti i campionatori ed in particolare l'uscita pari a 0.

Dopo di che si porranno in ***ingresso*** i segnali X_t e K, per poi abbassare il segnale di INIT e quindi iniziare i calcoli all'interno del circuito.

Il valore di Y_t conseguente alla prima iterazione è pari a 0, dal momento che l'uscita del CSA è *campionata* ad ogni *fronte di salita del clock*, quindi dal ciclo successivo a quello iniziale saranno presenti in uscita le ***soluzioni*** *corrette* dell'equazione della rete logica.

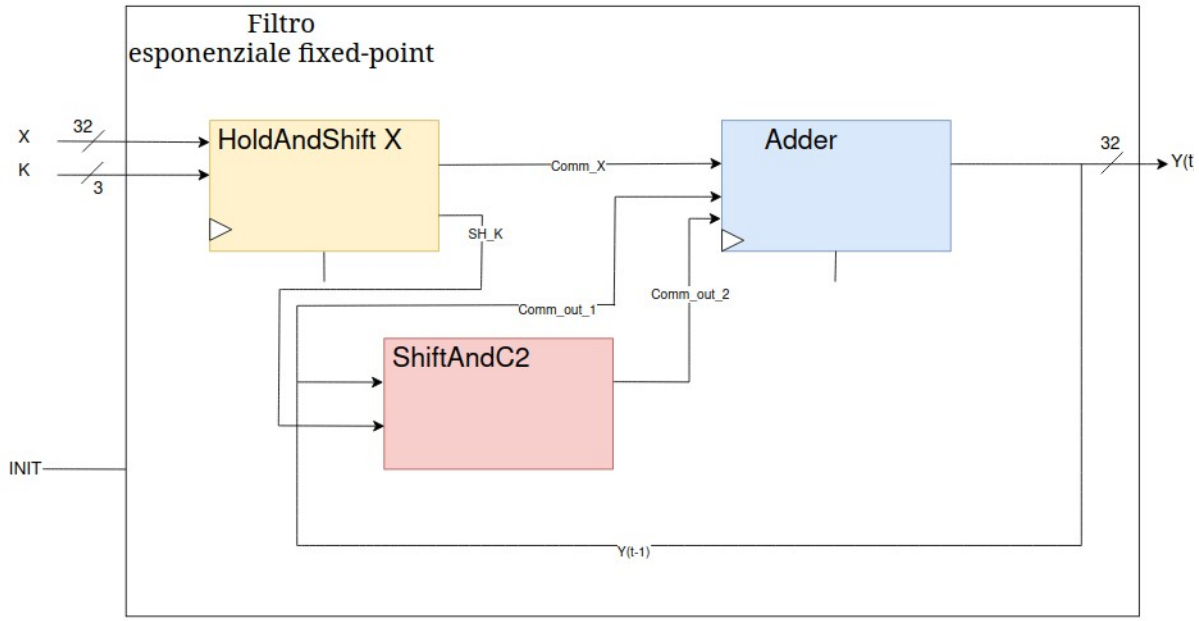
In seguito, è riportato il diagramma temporale del *reset* tramite INIT e la conseguente iterazione del circuito successiva ad uno risposta allo scalino:



NB: I valori di X_t , Y_t e K sono scritti in esadecimale per semplicità di lettura.

Architettura del sistema

Di seguito è riportato il circuito ad alto livello:

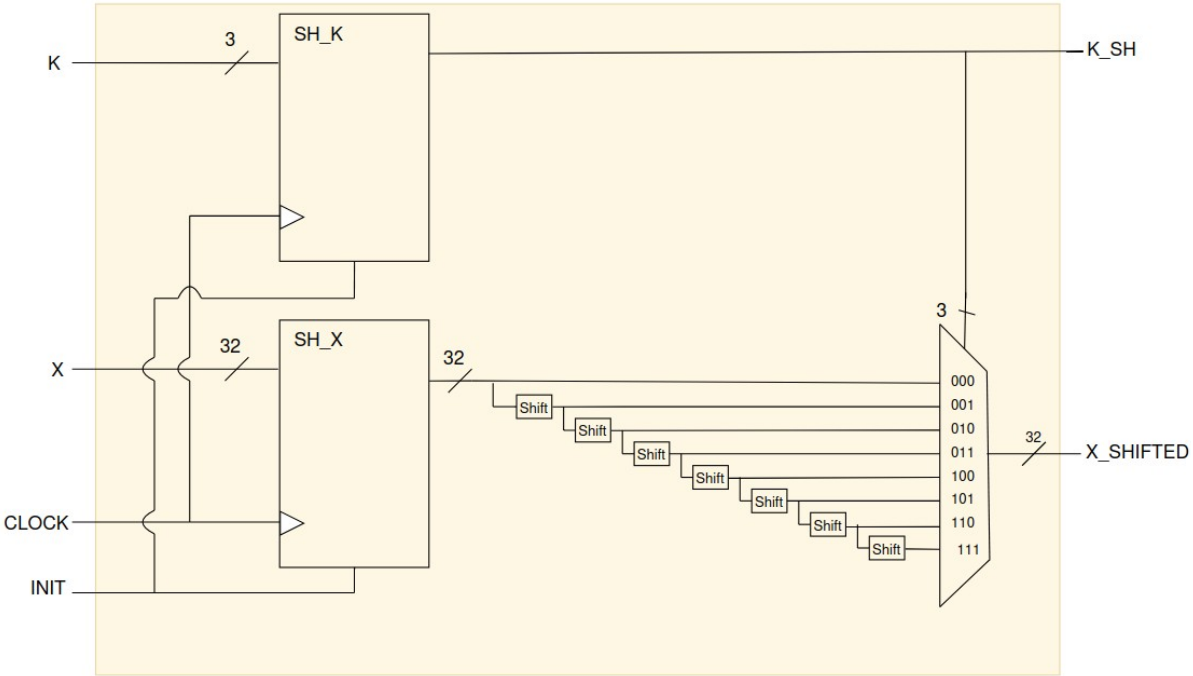


HoldAndShift X

Il modulo ***HoldAndShift X*** si occupa del campionamento dei segnali di ingresso tramite dei r*egistri* chiamati "***Memory***" e della successiva divisione tramite *Shift Right* del valore in ingresso X tramite un "***Shifter***" con un selettore da 3 bit a K ingressi.

Inoltre si può notare che tramite l'attivazione del segnalo alto "**INIT**", le istanze di *Memory* si porteranno nel loro stato di reset.

La struttura del modulo è la seguente:

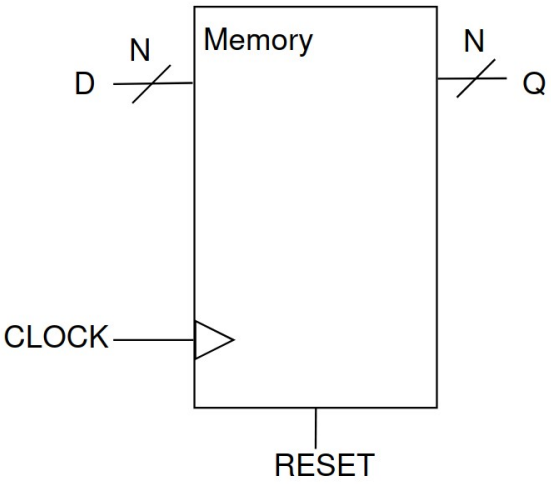


I componenti fondamentali per la realizzazione di questa rete logica sono il *registro denominato* "***Memory***" e il componente "***Mux8***".

Nel particolare:

MEMORY

Memory è l'istanza dei registri di memoria all'interno del circuito, presente sia in questo modulo che in "***Adder***".



Per lo sviluppo di questo modulo sono stati utilizzati i *costrutti generic,* per favorire la **riusabilità** del codice, vista la *differenza di dimensione* dei valori da memorizzare all'interno del circuito.

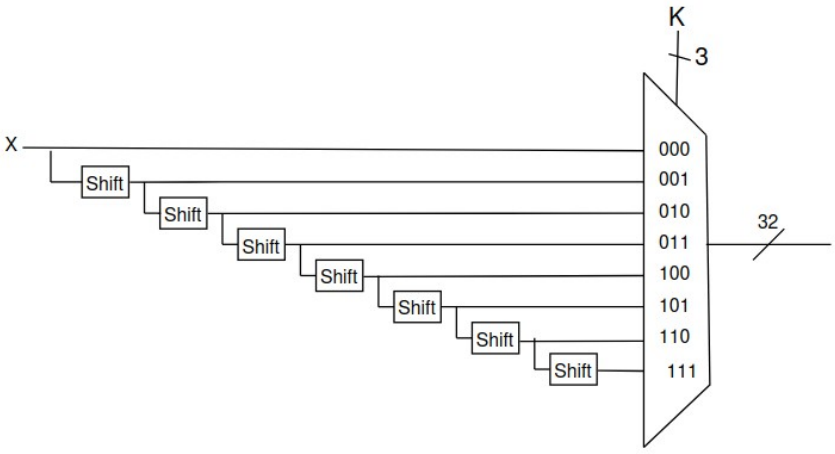
Questo modulo è quindi formato da ***N* Flip-Flop D**, di conseguenza da *N ingressi* ed *N uscite*, i quali costituiscono la cella di memoria di base all'interno del circuito.

Quest'ultimi aggiornano il loro valore con il *dato in ingresso* al singolo bistabile ad ogni *fronte di salita del clock*.

Memory ha collegato in ingresso anche il segnale INIT che, nel caso di attivazione, *resetta* tutti i bistabili portandoli a 0.

SHIFTER

Lo ***Shifter*** è il componente che si occupa della **divisione** di un determinato segnale in ingresso di 32 bit per una potenza negativa del 2 nell'intervallo [0..7] tramite il *selettore* **K**, facendo arrivare in ingresso ad ognuna delle 8 entrate del componente il valore in ingresso shiftato. Per la creazione di questo componente sono stati utilizzati *7 moduli di Shift* che hanno il compito di *shiftare* di un bit verso destra la cifra in

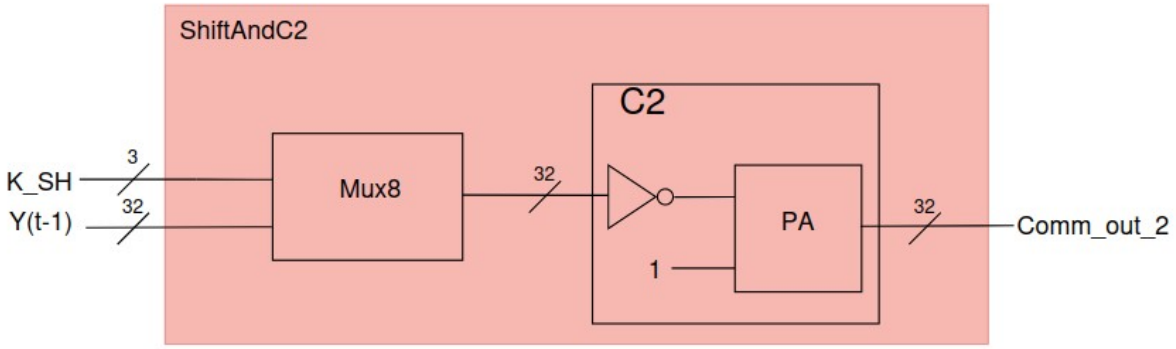


ingresso, e con particolare attenzione, dato che stiamo lavorando nel contesto di cifre *fixed-point*, di **centrare la virgola fissa** del componente tramite l'aggiunta di un bit (pari al MSB della cifra di ingresso) alla sinistra del numero.

Shift and C2

Il modulo ***Shift And C2*** si occupa prima dello shift del segnale Y_{t-1}, il quale è l'uscita del modulo "***Adder***", tramite il componente ***Mux8*** (come visto per il modulo precedente) ed in seguito del *complemento a due* del numero shiftato tramite un ***PA***.

Il modulo ha la seguente struttura:

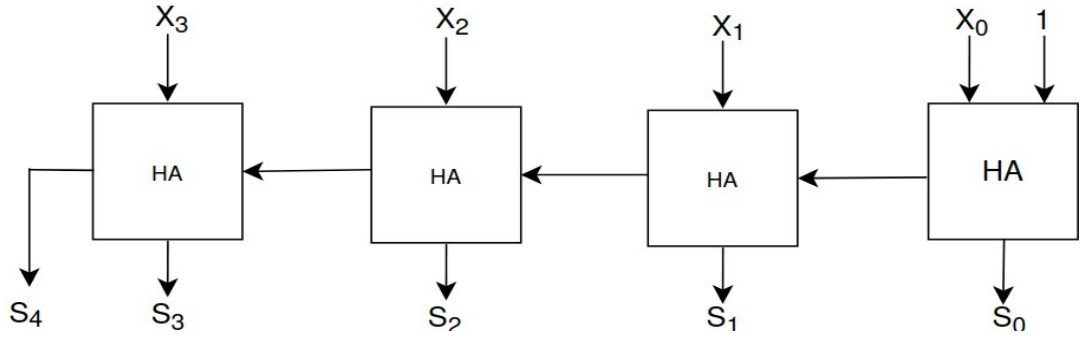


Il modulo C2 calcola il complemento a due del numero in ingresso:

$$C2 = \neg X + 1$$

Questa somma è eseguita tramite un ***PA*** (Partial Adder), il quale prende in ingresso una stringa di bit X e somma all'ingresso 1, tramite una serie di Half Adder che producono ognuno il risultato della somma tra un bit e il riporto della somma precedente, il modulo produce l'output S, che rappresenta il risultato della somma tra X e CIN, e COUT, che rappresenta l'eventuale riporto dell'operazione.

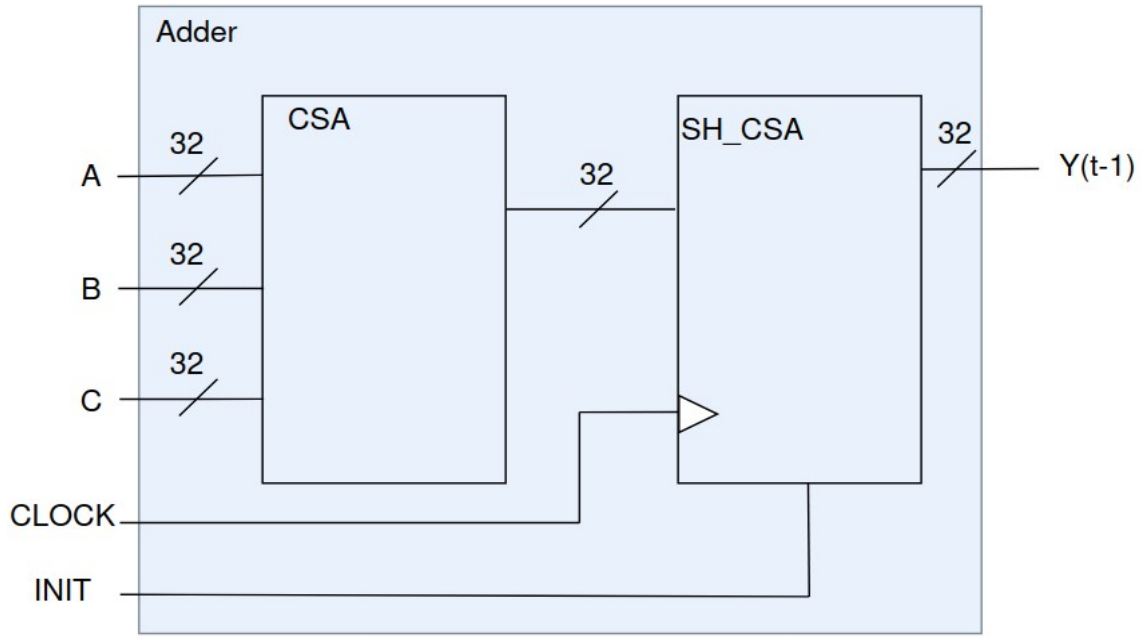
La struttura è la seguente:



Adder

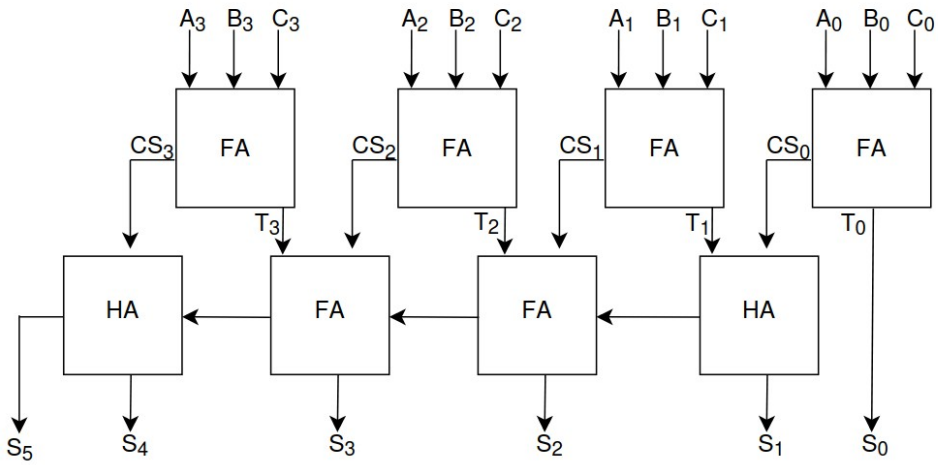
L'***Adder*** è composto da un *Carry Save Adder,* per eseguire la somma dei tre numeri da 32 bit in ingresso, e da una istanza del modulo **Memory** che *campiona* l'uscita ad ogni fronte di salita del clock per essere portata sia in uscita al modulo "***Adder*** " che al modulo "***Shift And C2 "***.

Il modulo ha la seguente struttura:



Il ***CSA*** (Carry Save Adder) è pensato per la somma di più operandi, rispetto agli altri sommatori ha la particolarità di *evitare la propagazione immediata* del carry, migliorando così la velocità complessiva delle operazioni aritmetiche . Dato che l'uscita deve essere su 32 bit, gli ultimi due in output dal *CSA* vengono eliminati troncando appunto l'uscita su 32 bit rispetto a 34.

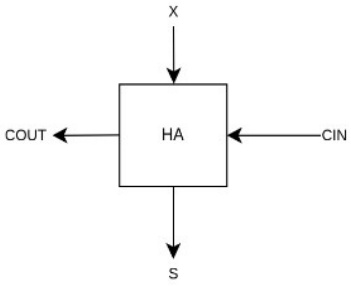
La struttura è la seguente:



Moduli Base

Inoltre per la realizzazione dei componenti trattati in precedenza sono stati utilizzati anche altri componenti, brevemente descritti di seguito.

HA (Half Adder)

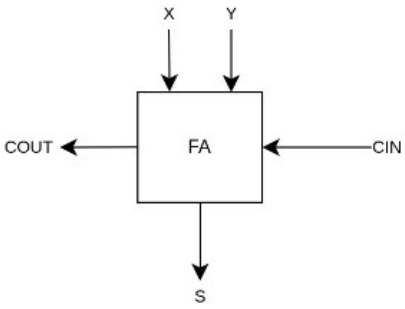


L'half Adder si occupare di sommare un bit X ed un riporto CIN, riportando il risultato della somma al segnale di uscita S e il riporto della somma sul segnale di uscita COUT. Le uscite seguono le seguenti relazioni:

$$S = X \ xor \ CIN$$

$$COUT = X \ and \ CIN$$

FA (Full Adder)



Il Full Adder si occupa di sommare due bit X e Y ed un riporto CIN, riportando il risultato della somma al segnale di uscita S ed il riporto della somma sul segnale di uscita COUT. Le uscite seguono le seguenti relazioni:

$$S = X \ xor \ Y \ xor \ CIN$$

$$COUT = (X \ and \ Y) \ or \ (Y \ and \ CIN) \ or \ (X \ and \ CIN)$$

Verifica

I test sono stati eseguiti prima sui *singoli moduli*, per assicurarsi del corretto funzionamento di quest'ultimi, in seguito si è **testato** il Filtro Esponenziale come ***unione dei moduli***.

Per testare *l'effettivo l'output* del filtro esponenziale è stata sviluppata una semplice applicazione in c, infatti inserendo X_t, K ed I (pari al numero di iterazioni che si desidera eseguire) restituisce in *output* i risultati desiderati per ogni iterazione.

```
/home/dracarys/CLionProjects/untitled1/cmake-build-debug/untitled1
Insert the X value:5678
Insert the K value:5

How many times do you want to iterate?10

Y: 177.4375000000000000  11628544    Binary: [00000000101100010111000000000000]

Y: 349.3300781250000000  22893696    Binary: [00000001010111010101010010000000]

Y: 515.8510131835937500  33806812    Binary: [00000010000000111101100111011100]

Y: 677.1681690216064453  44378893    Binary: [00000010101001010010101100001101]

Y: 833.4441637396812439  54620597    Binary: [00000011010000010111000110110100]

Y: 984.8365336228162050  64542247    Binary: [00000011110110001101011000100111]

Y: 1131.4978919471031986     74153846    Binary: [00000100011010110111111101110101]

Y: 1273.5760828237562237     83465082    Binary: [00000100111110011001001101111010]

Y: 1411.2143302355138985     92485342    Binary: [00000101100000110011011011011110]

Y: 1544.5513824156541887     101223719    Binary: [00000110000010001000110100100111]
```

Per semplificare la fase di debug, i risultati sono riportati e studiati anche in decimale.

Dato che l'esecuzione del filtro non è pensata con un limite di iterazioni, per i vari test i limiti vengono forniti e gestiti dall'utente.

NB: Il filtro *non* è stato definito con un *limite di iterazioni,* ma si nota durante la fase di debug che l'output Y_t, dopo un determinato numero di iterazioni (proporzionale al valore di K), si assesta al valore di X_t.

Sono state inoltre fatte diversi tipi di verifica:

Verifica behavioral:

Utilizzata per assicurarsi che il *componente rispetti le specifiche* trascurando gli effetti di ritardo dovuti ai transistor e componenti elettriche interne al circuito.

Verifica post-place & route:

Per *verificare* se effettivamente il componente è in grado di funzionare *nonostante ritardi* dovuti alle porte logiche. Questo modo ci permette di comprendere se il circuito sarà in grado di funzionare alle frequenze di clock desiderate.

Nel nostro caso si è stimato un best case pari a 12.866 ns con una frequenza di clock massima pari a 77.7 MHz.

Test-bench e casi d’uso

I test case del top-level del filtro si ispirano al funzionamento reale del circuito, valutando le diverse possibilità:

- **Verifica dell’effettivo funzionamento** del filtro tramite prove con diversi valori di X e K ed il conseguente confronto con i valori dell’output del programma in c
- **Test sul segnale di INIT** per verificare il corretto riporto del sistema allo stato iniziale.

Prestazioni

Dall’analisi temporale calcolata tramite simulazione si evidenzia un tempo di **risposta massimo** pari a 27,134 ns. Utilizzando un margine del 10% si conclude che il tempo minimo di clock del dispositivo deve essere di 30 ns e di conseguenza la **frequenza** di clock non deve essere superiore a 33,3 MHz.