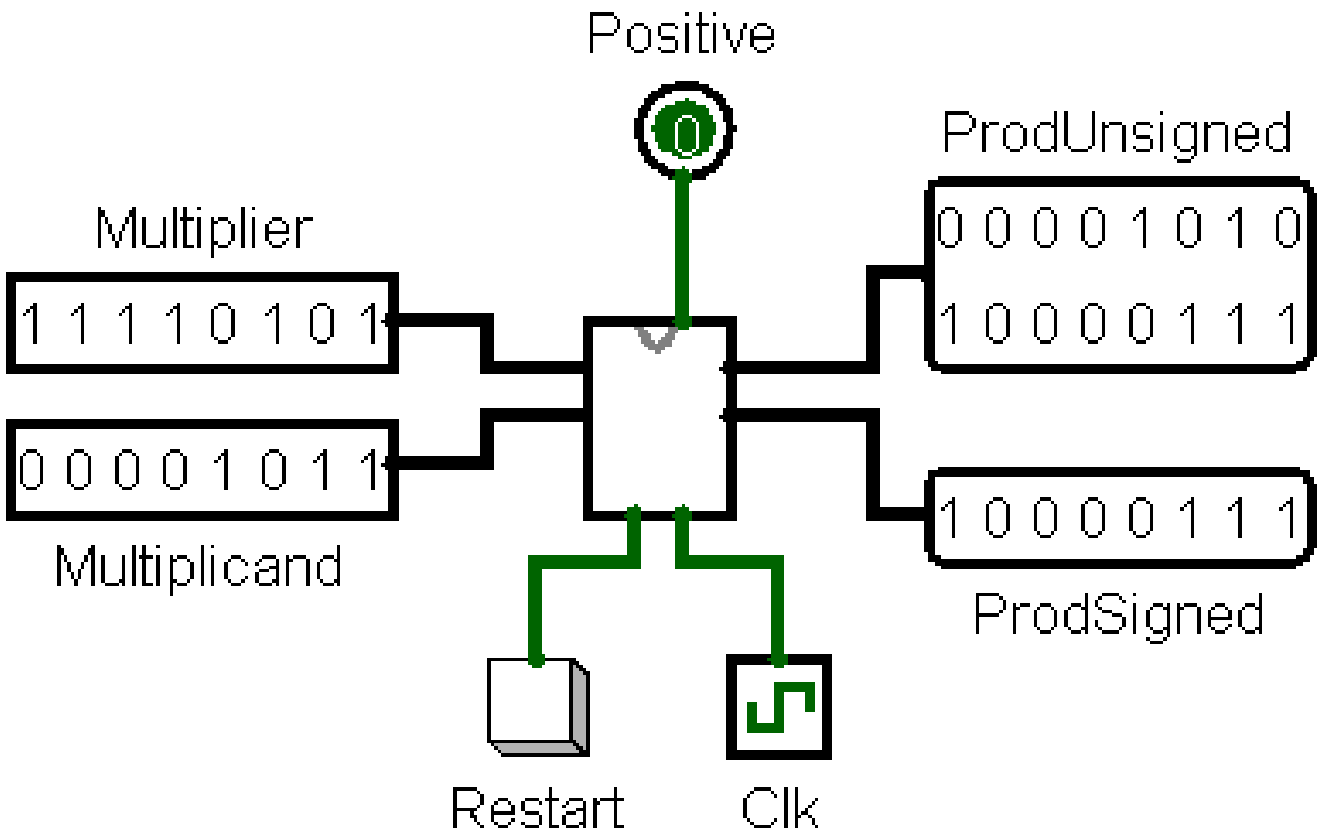
MOLTIPLICATORE BINARIO



Corso di Laurea in Informatica

Anno Accademico 2018-2019

Passini Serena

Terenziani Andrea

MOLTIPLICATORE BINARIO

La moltiplicazione

La moltiplicazione è una delle quattro operazioni aritmetiche fondamentali e, dati due valori a e b, l'espressione $a \times b$ corrisponde a sommare $b + \cdots + b$ per a volte.

$$
\begin{array}{cccc}
 & 1 & 0 & 1 \quad \times \\
 & 1 & 1 & 0 \quad = \\
\hline
 & 0 & 0 & 0 \quad + \quad \rightarrow 101 \times 0 \\
1 & 0 & 1 & \quad + \quad \rightarrow 101 \times 1 \\
1 & 0 & 1 & \quad = \quad \rightarrow 101 \times 1 \\
\hline
1 & 1 & 1 & 1 \quad 0
\end{array}
$$

Il primo operando è chiamato *moltiplicando* e il secondo *moltiplicatore*. Il risultato è detto *prodotto*. Come si sa, il prodotto è calcolato come *somma di prodotti parziali*. Ciascuno è ottenuto con il seguente algoritmo:

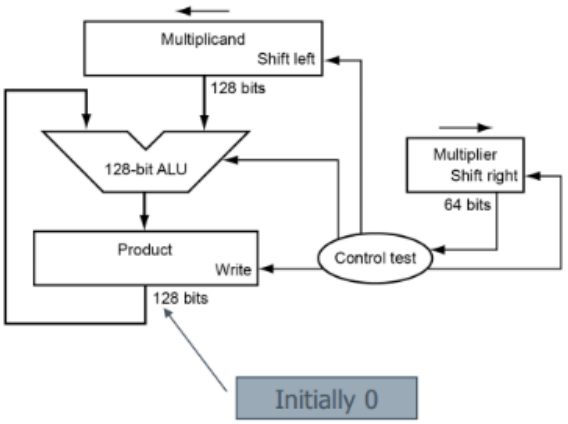
1. si moltiplica la cifra *j*-esima (partendo da destra) del moltiplicatore per il moltiplicando;
2. viene shiftato tale risultato di 1 posizione più a sinistra del prodotto parziale precedente.

Calcolati così gli *n* prodotti parziali, essi sono sommati per ottenere il risultato finale.
Si può osservare che, nel caso in cui si stiano moltiplicando operatori binari, per ogni prodotto parziale si hanno solo due possibilità:

1) Se il bit meno significativo del moltiplicatore è 1, il prodotto parziale è semplicemente il moltiplicando shiftato a sinistra per il numero di volte necessario;
2) Se il bit meno significativo del moltiplicatore è 0, invece, il prodotto parziale è chiaramente 0.

Una prima possibile implementazione può consistere nel replicare letteralmente questo algoritmo, ossia:

- Memorizzare il moltiplicatore in un registro da n bit e il moltiplicando in uno da 2n bit (poiché deve essere shiftato n volte a sinistra evitando di perderne le cifre);
- Conservare la somma dei prodotti parziali in un registro da 2n bit, inizializzato a 0 e che conterrà, al termine, il prodotto finale.
- Usare una ALU da 2n bit per sommare il *j*-esimo prodotto parziale al registro del prodotto.



Questo algoritmo consiste in:

a) controllare se l'ultima cifra del moltiplicatore è uguale a 1. In caso affermativo, si aggiunge il moltiplicando al registro del prodotto e si shifta il moltiplicando a sinistra di un bit. In caso contrario, si esegue solo l'operazione di shift.
b) shiftare il moltiplicatore a destra di un bit;
c) ripetere questo processo finché non siamo all'*n-esima* iterazione.

Si può migliorare, dal momento che somma e shifting sono eseguiti ciascuno in un ciclo di clock e che la memoria utilizzata non è ottimizzata.

Possiamo effettuare i seguenti miglioramenti :

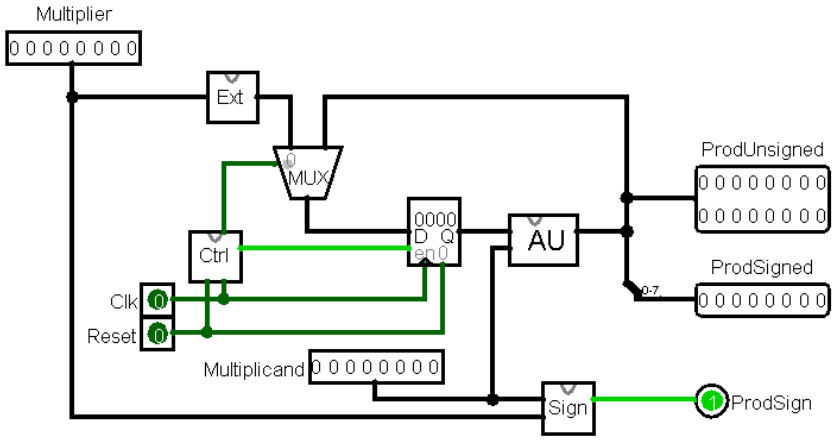
1) conservare in un unico registro a 2n bit il prodotto e il moltiplicatore, invece di utilizzare due registri separati.
2) una ALU a n bit rispetto ai 2n bit descritti precedentemente.

Indi per cui, il nuovo algoritmo consisterà nel :

a) controllare se l'ultima cifra del registro contenente il prodotto e il moltiplicatore è uguale ad 1. In caso affermativo, si aggiunge il moltiplicando alle n cifre più significative di questo registro e lo si shifta a destra di 1 bit. In caso contrario, si esegue solo l'operazione di shift.
b) ripetere questo processo finché non siamo alla n-esima iterazione.

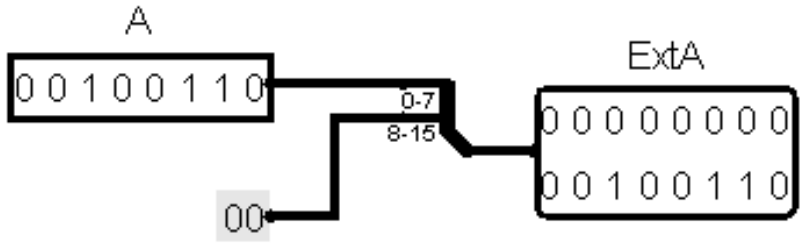
Dal punto a), possiamo osservare che l'azione di shift a destra sul prodotto incorpora l'azione di shift a sinistra del moltiplicando poiché permette che il moltiplicando sia via via sommato alle cifre più significative del moltiplicatore.

Il nostro circuito



Nel dettaglio, si illustrano in seguito i sotto-circuiti.

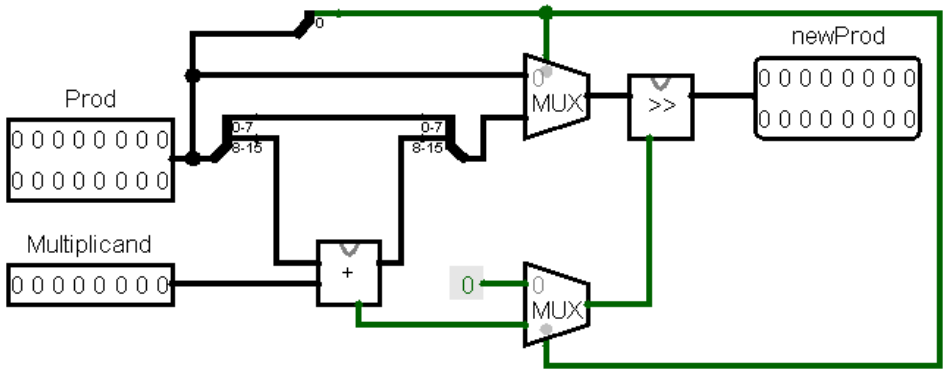
1) ***Extender:***



È un particolare circuito che prende in input un numero ad 8 bit e lo estende a 16 bit. Tramite uno splitter si inizializzano a 0 le cifre più significative dell'output, e al numero in input le 8 cifre meno significative.

Serve ad inizializzare il registro contenente il prodotto e il moltiplicatore.

2) **Unità aritmetica:**



Per chiarezza, definiamo *K* il valore del bit meno significativo di Prod.

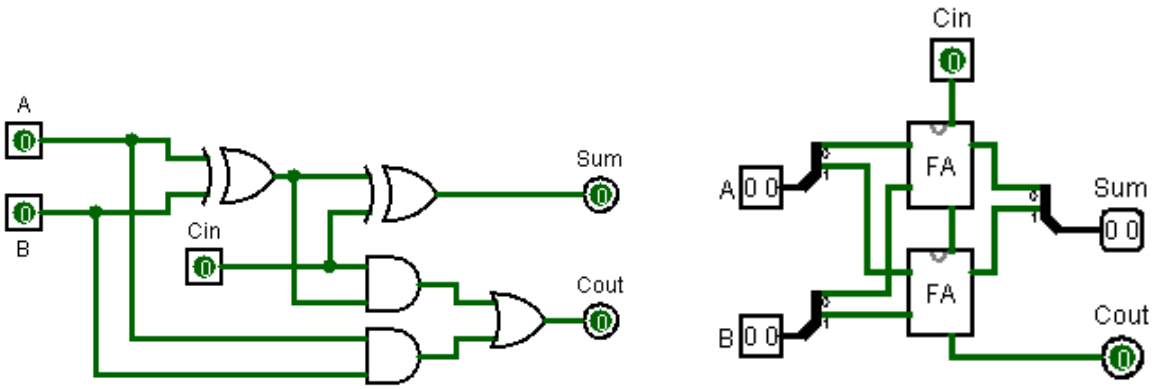
Se K è 0, allora il contenuto di Prod viene shiftato il contenuto stesso, altrimenti viene shiftato Prod con il moltiplicando sommato alle 8 cifre più significative.

La cifra più significativa di newProd sarà:

-0, se K=0;

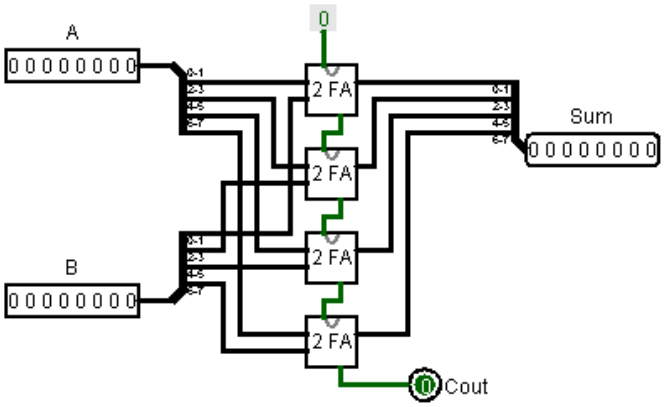
-Il carry out della somma tra le cifre più significative di Prod e il moltiplicando, se K=1;

3) **Adders:**



Entrambi prendono due numeri in input, qui detti A e B, e ne calcolano la somma. Cin e Cout rappresentano, rispettivamente, il riporto in ingresso e in uscita dell'operazione.

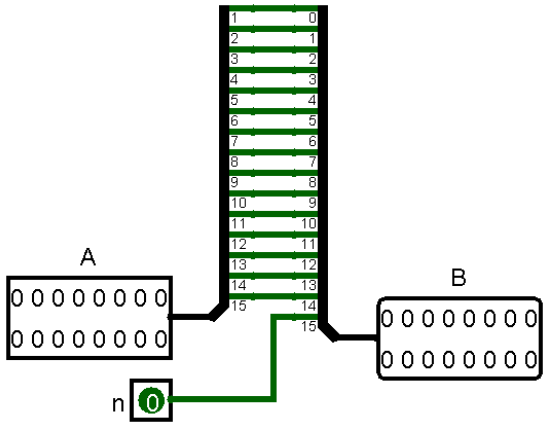
Per questioni di modularità e compattezza, abbiamo creato l'adder finale come combinazione di 4 full adders a 2 bit (invece di 8 adders a 1 bit).



2FA sono full adder a 2 bit. Il carry in della somma è rappresentato dalla costante 0 posta in alto nel primo adder, mentre il carry out è inserito come pin di uscita nell'ultimo adder.

Con uno splitter abbiamo selezionato i primi 2 bit (siccome i nostri adder sono a 2 bit) per ogni adder inserito nel circuito.

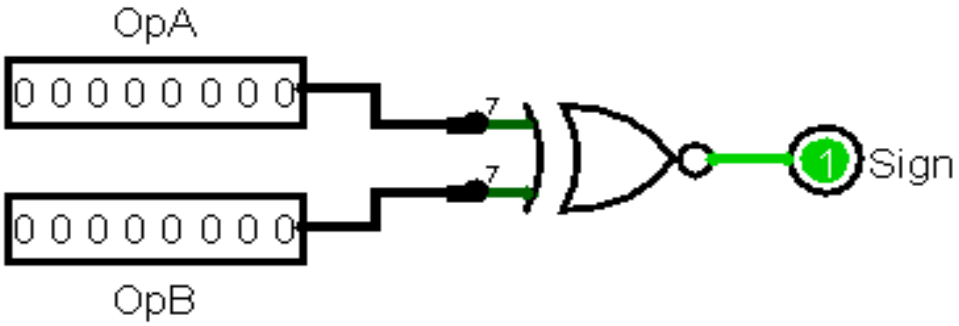
4) Bit shifter:



Siccome l'output della moltiplicazione è ottenuto shiftando a sinistra per un numero di volte necessario il moltiplicando, abbiamo anche implementato un cosiddetto shifter:

Esso riceve in ingresso **A**, numero a 16 bit, e **n**, un numero da 1 bit. Esegue quindi uno shift a destra di **A** di una posizione, impostando **n** come nuovo valore del bit più significativo.

5) **Segno**:

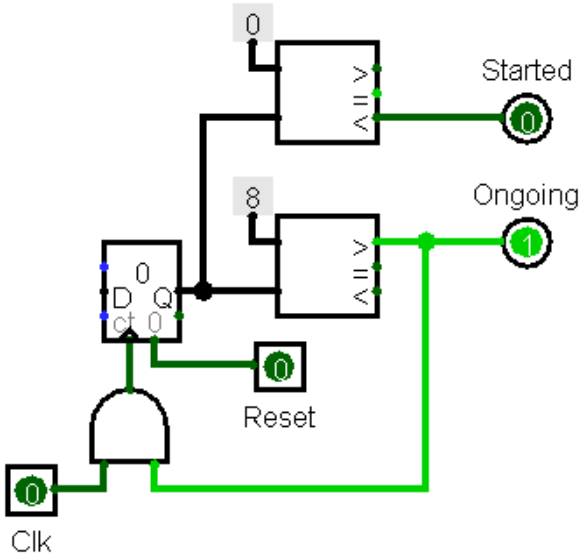


Determina il segno della moltiplicazione. Se gli operandi hanno lo stesso segno (ovvero la stessa cifra più significativa) allora il risultato sarà positivo. Negativo, altrimenti; poiché se un operando ha il primo bit a 1, significa che esso non è altro che un complemento a 2 di un numero, quindi sarà sicuramente un numero negativo.

a	b	x
0	0	1
0	1	0
1	0	0
1	1	1

Abbiamo implementato questa funzione con un gate XNOR, la cui tabella di verità è rappresentata a fianco.

6) **Unità di controllo**:

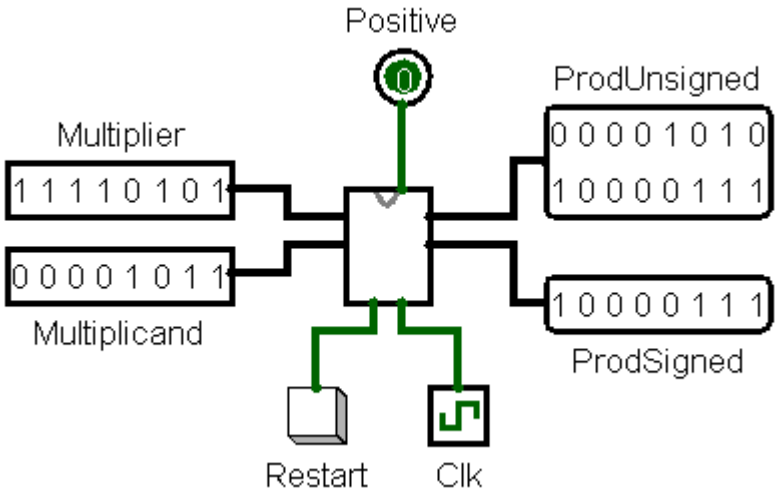


Regola l'esecuzione del programma, contando le iterazioni dei prodotti parziali. Se esse sono minori strette di 8, allora significa che possiamo procedere ad una n-esima iterazione. Altrimenti, il conteggio si ferma poiché abbiamo raggiunto il massimo numero di cifre binarie moltiplicabili.

Questo circuito prende in ingresso un clock, per scandire il conteggio, e un segnale di reset, che ripristina il circuito a 0. Abbiamo fatto uso di comparatori, componenti dedicati al confronto di due variabili in ingresso.

Il primo comparatore ci serve per determinare se il conteggio è maggiore di 0, in tal caso *Started* avrà valore 1; il secondo comparatore, invece, serve per determinare se il conteggio è minore di 8. *Ongoing* è un pin di uscita che ha valore 1 se il conteggio è appunto minore di 8.

Il circuito nel complesso



Questa è la rappresentazione complessiva del nostro circuito. Esso è regolato da un clock interno, prende in input due numeri a 8 bit ciascuno e restituisce in output il prodotto con segno a 8 bit, e il prodotto considerato come unsigned a 16 bit.

Come pin di uscita abbiamo *Positive*, che ci indica il segno del prodotto, come pin di ingresso, invece, abbiamo *Restart* che reinizializza il contatore e il registro ProdottoMoltiplicatore.

Tuttavia, il nostro moltiplicatore non è esente da difetti, al di là del modello implementativo, che sono principalmente descritte come:

a) una maggiore complessità, unita a una minore modularità in termini di circuiteria, rispetto per esempio alle semplici operazioni di somma e sottrazione.
b) Non si ha un risultato immediato, dal momento in cui ogni prodotto parziale viene calcolato ad ogni ciclo di clock ed il risultato finale viene prodotto al termine degli *n* cicli di clock necessari, dove *n* rappresenta le cifre degli operandi da sommare. Il ritardo introdotto dalle implementazioni di moltiplicatori basate su iterazioni di somme e di shift è spesso inaccettabile.

Esempi di altri moltiplicatori

Esistono molteplici versioni di moltiplicatori: facciamo un breve cenno alle moderne implementazioni.

- Moltiplicatori veloci: per risolvere il problema del ritardo, sono stati creati moltiplicatori capaci di eseguire il prodotto di due numeri in un solo ciclo di clock.
  Per esempio:
  - Moltiplicatori a 'look up table': questo tipo di moltiplicatori fa uso di circuiti di memoria dove sono immagazzinati i valori pre-calcolati di tutti i possibili prodotti di due numeri a n bit (unsigned). L'esecuzione della moltiplicazione consiste quindi in un semplice accesso alla memoria in corrispondenza del valore cercato. La crescita della dimensione della memoria è *esponenziale* con il numero di bit n degli operandi (questo può quindi diventare rapidamente un problema).
  - Moltiplicatori a matrice: consiste nella generalizzazione a n bit del circuito combinatorio che realizza la moltiplicazione di due numeri a 1 bit. Si tratta di una matrice di n sommatori a n bit. La struttura è regolare quindi semplice da realizzare.

$1101_2 \cdot 0110_2 = 01001110_2$

1101	0011	00100111
1101	0100	00110100
1101	0101	01000001
1101	0110	01001110
1101	0111	01011011