



# Firmware Multiplier

Prof. Alberto Borghese  
Dipartimento di Informatica  
[borgnese@di.unimi.it](mailto:borgnese@di.unimi.it)

Università degli Studi di Milano

Riferimenti sul Patterson 5a ed.: B.6 & 3.4



## Sommario

**Il moltiplicatore firmware**

Ottimizzazione dei moltiplicatori firmware



## L'approccio firmware



Nell'approccio firmware, viene inserita nella ALU una unità di controllo e dei registri. L'unità di controllo attiva opportunamente le unità aritmetiche ed il trasferimento da/verso i registri. Approccio "*controllore-datapath*".

Viene inserito un microcalcolatore dentro la ALU.

Il primo microprogramma era presente nell'IBM 360 (1964).



## L'approccio firmware vs hardware



La soluzione HW è più veloce ma più costosa per numero di porte e complessità dei circuiti.

La soluzione firmware risolve l'operazione complessa mediante una sequenza di operazioni semplici. E' meno veloce, ma più flessibile e, potenzialmente, adatta ad inserire nuove procedure.

La soluzione HW è percorsa per le operazioni frequenti: la velocizzazione di operazioni complesse che vengono utilizzate raramente non aumenta significativamente le prestazioni (legge di Amdahl).



## Algoritmi per la moltiplicazione



Il razionale degli algoritmi firmware della moltiplicazione è il seguente.

Moltiplicando  $11011 \times$   
 Moltiplicatore  $101 =$

Si analizzano sequenzialmente i bit del moltiplicatore e:

- 1) Si mette 0 nella posizione opportuna (se il bit analizzato del moltiplicatore = 0).
- 2) Si mette una copia del moltiplicando nella posizione opportuna (se il bit analizzato del moltiplicatore è = 1).

```

-----
      11011 +
  00000 -
-----
      11011
  11011 - -
-----
  Prodotto 10000111
  
```



## Shift (scalamento)

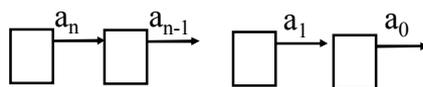


Dato A su 32 bit:  $a_j = a_{j-k}$  k shift amount ( $>$ ,  $=$ ,  $<$  0).

Effettuato al di fuori delle operazioni selezionate dal Mux della ALU, da un circuito denominato *Barrel shifter*.

Tempo comparabile con quello della somma.

Operazioni codificate in modo specifico nell'ISA.



Shift dx 1



Il bit  $a_0$  si "perde".  
Il bit  $a_n = 0$ .



# Moltiplicazione utilizzando somma e shift



Utilizzo un registro prodotto da 64 bit, inizializzato a 0.

$$\begin{array}{r}
 11011 \times A \\
 111 = B
 \end{array}$$

$$\begin{array}{r}
 00000+ \\
 11011
 \end{array}$$

Itero per ogni bit del moltiplicatore:

A) Sommo il moltiplicando al prodotto se il bit = 1.

$$\begin{array}{r}
 11111 \\
 11011+ P \\
 11011- A
 \end{array}$$

B) Shift a sx di un bit il moltiplicando ( $A' = A * \text{base}$ ).

$$\begin{array}{r}
 1 \\
 1010001+ P \\
 11011- - A_1 \\
 \hline
 10111101
 \end{array}$$

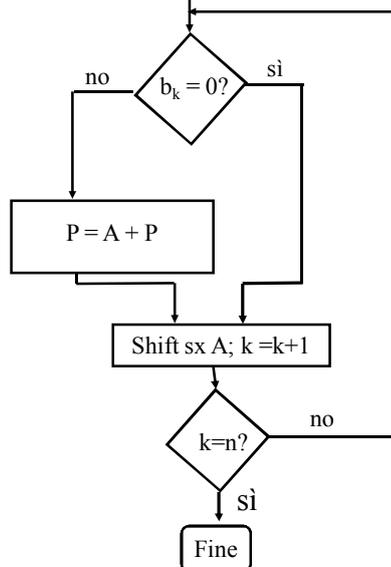
$$27 \times 7 = 189$$



# L'algoritmo



Inizio:  $P = 0; k = 0$



$$\begin{array}{r}
 A \rightarrow 11011 \times \\
 B \rightarrow 111 =
 \end{array}$$

$$\begin{array}{r}
 00000+ P \\
 11011 A
 \end{array}$$

$$\begin{array}{r}
 P_1 = 0 + A \\
 A_1 = A * 2 \\
 11111 \\
 11011+ P \\
 11011- A_1
 \end{array}$$

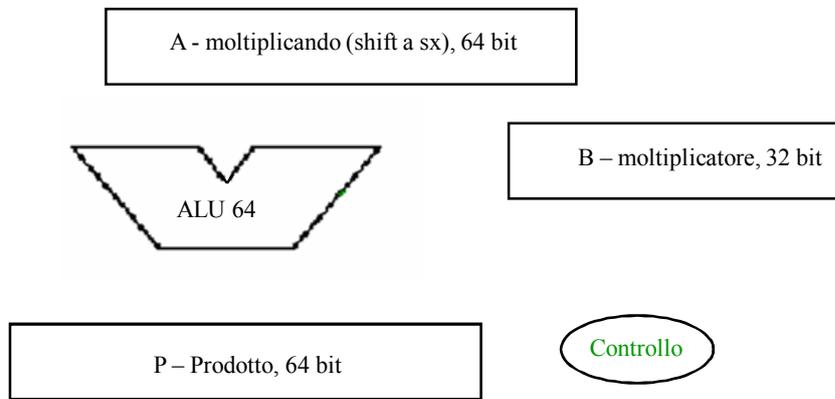
$$\begin{array}{r}
 P_2 = P_1 + A_1 \\
 A_2 = A_1 * 2 = A * 4 \\
 1 \\
 1010001+ P \\
 11011- - A_2
 \end{array}$$

$$\begin{array}{r}
 P_3 = P_2 + A_2 \\
 10111101
 \end{array}$$

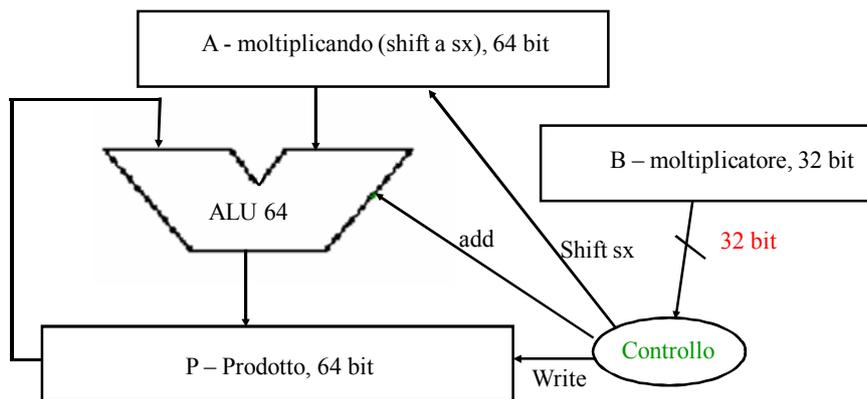
P contiene le somme parziali, al termine conterrà la somma totale, cioè il risultato del prodotto.



## Implementazione circuitale – gli attori



## Implementazione circuitale



Qual'è il problema?



## Esercizi



Costruire il circuito HW che esegui la moltiplicazione  $7 \times 9$  in base 2.

Eeguire la stessa moltiplicazione secondo l'algoritmo visto, indicando passo per passo il contenuto dei 3 componenti: A che contiene il moltiplicando, B che contiene il moltiplicatore e P che contiene somme parziali ed il risultato finale.



## Sommario

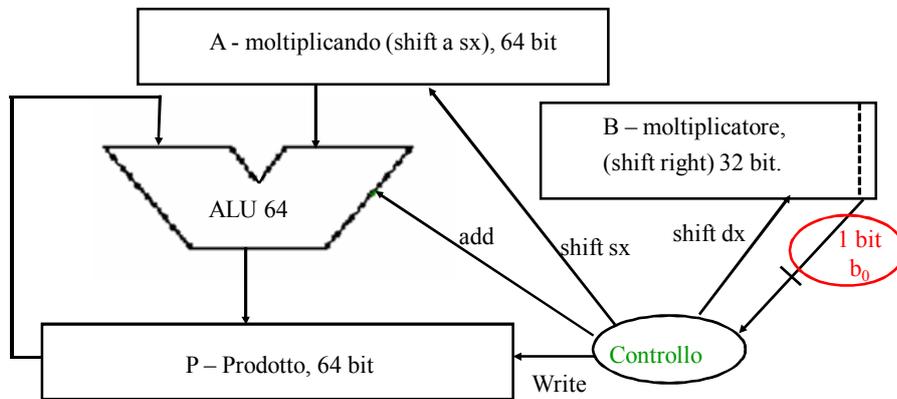


I moltiplicatori firmware

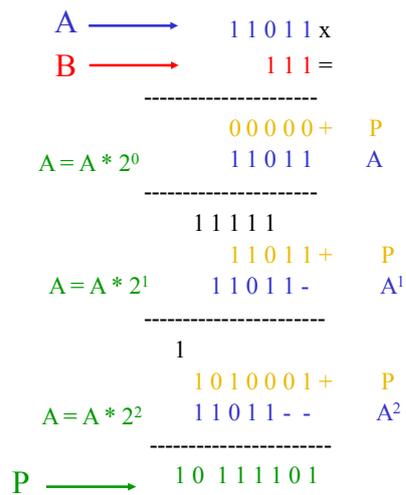
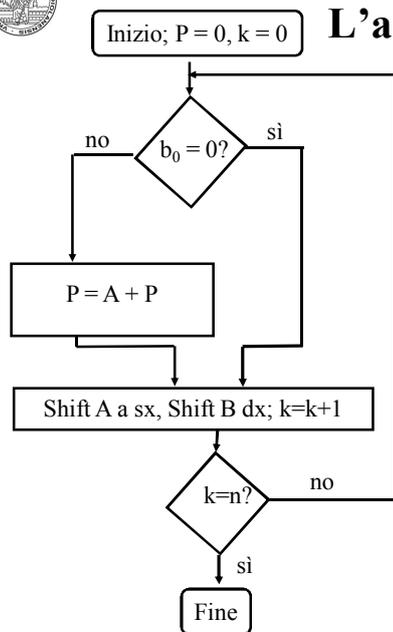
**Ottimizzazione dei moltiplicatori firmware**



# Implementazione circuitale ottimizzata - I



# L'algoritmo - I





## Esempio - I

Iterazione	Passo	Moltiplicatore	Moltiplicando	Prodotto
0	Valori iniziali	0010	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcando	0011	0000 0010	0000 0010
	2: Scala a sinistra Moltiplicando	0011	0000 0100	0000 0010
	3: Scala a destra Moltiplicatore	0000	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcando	0001	0000 0100	0000 0110
	2: Scala a sinistra Moltiplicando	0001	0000 1000	0000 0110
	3: Scala a destra Moltiplicatore	0000	0000 1000	0000 0110
3	1: 0 ⇒ Nessuna operazione	0000	0000 1000	0000 0110
	2: Scala a sinistra Moltiplicando	0000	0001 0000	0000 0110
	3: Scala a destra Moltiplicatore	0000	0001 0000	0000 0110
4	1: 0 ⇒ Nessuna operazione	0000	0001 0000	0000 0110
	2: Scala a sinistra Moltiplicando	0000	0010 0000	0000 0110
	3: Scala a destra Moltiplicatore	0000	0010 0000	0000 0110

$$\begin{array}{r}
 0010 \times \\
 0011 = \\
 \hline
 \text{-----}
 \end{array}$$

Moltiplicazione su 4 bit.



## Razionale per una seconda implementazione

Meta' dei bit del registro moltiplicando vengono utilizzati ad ogni iterazione.

Ad ogni iterazione si aggiunge 1 bit al registro prodotto.

Ad ogni iterazione sommo N cifre (pari al numero di cifre del moltiplicando).

Spostamento della ALU sul registro prodotto.

Oppure

Si sposta la somma dei prodotti parziali verso dx di 1 bit ad ogni iterazione.

$$\begin{array}{r}
 11011 \times \\
 111 = \\
 \hline
 00000 + \quad P \\
 11011 \quad A \\
 \hline
 11111 \\
 11011 + \quad P \\
 11011 - \quad A^1 \\
 \hline
 1 \\
 1010001 + \quad P \\
 11011 - - \quad A^2 \\
 \hline
 10111101
 \end{array}$$

**Implementazione ottimizzata - II**

1<sup>a</sup> implementazione

11011 x  
111 =

-----  
00000+  
11011

-----  
11111  
11011+  
11011-

-----  
1  
1010001+  
11011--

-----  
-  
10111101

2<sup>a</sup> implementazione

11011  
11011

Sposto a sx il moltiplicando | Sposto a dx il prodotto  
P<sup>1</sup> primo prodotto parziale

**Qual'è il problema?**

A - moltiplicando, 32 bit

ALU 64

B - moltiplicatore, (shift right) 32 bit.

P - Prodotto, 64 bit

Controllo

add

shift dx

w

shift dx

**Implementazione ottimizzata - II**

1<sup>a</sup> implementazione

11011 x  
111 =

-----  
00000+  
11011

-----  
11111  
11011+  
11011-

-----  
1  
1010001+  
11011--

-----  
-  
10111101

2<sup>a</sup> implementazione

11011  
11011

Sposto a sx il moltiplicando | Sposto a dx il prodotto  
P<sup>1</sup> primo prodotto parziale

A - moltiplicando, 32 bit

ALU 32

B - moltiplicatore, (shift right) 32 bit.

P - Prodotto, 64 bit

Controllo

add

shift dx

w

shift dx

Parte modificabile

Parte non modificabile



## Razionale dell'implementazione - III



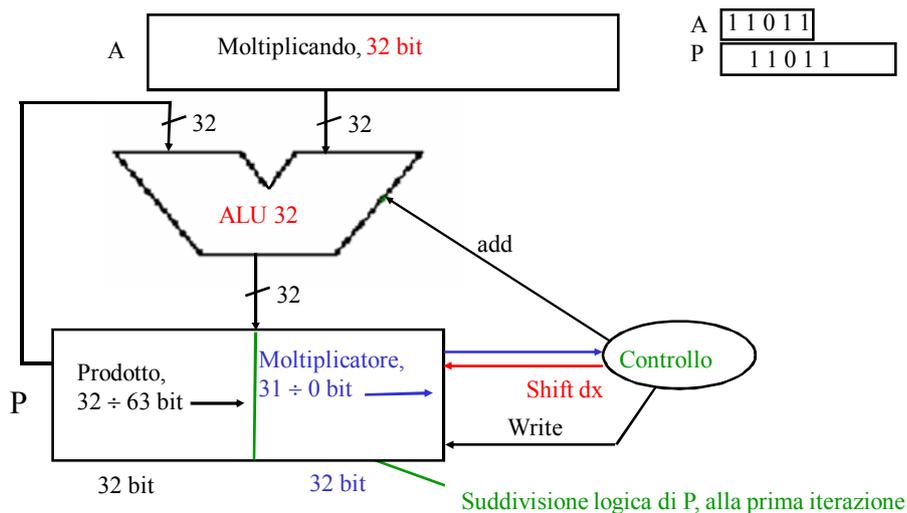
Il numero di bit del registro prodotto corrente (somma dei prodotti parziali) più il numero di bit da esaminare nel registro moltiplicando rimane **costante** ad ogni iterazione (pari a 64 bit).

Si può perciò eliminare il registro moltiplicatore.

$$\begin{array}{r}
 11011x \\
 111= \\
 \hline
 00000+ \\
 11011 \\
 \hline
 11111 \\
 11011+ \\
 11011- \\
 \hline
 1 \\
 1010001+ \\
 11011- - \\
 \hline
 - \\
 10111101
 \end{array}$$



## Circuito ottimizzato - III



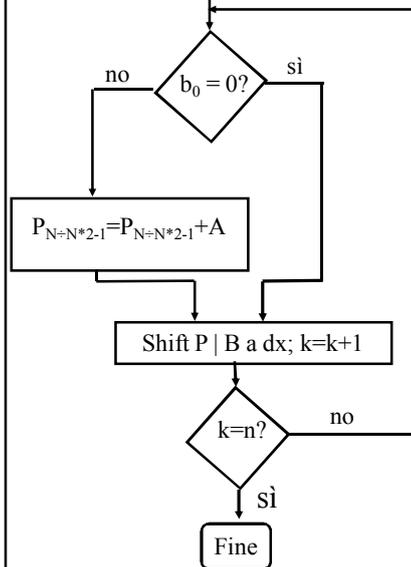
Il moltiplicando è allineato sempre ai 32 bit più significativi del prodotto.  
Ad ogni iterazione, il prodotto si allarga, il moltiplicatore si restringe.



## L' algoritmo ottimizzato - III



Inizio;  $P = 0 \mid B, k = 0$



```

A → 11011x
B → 111=
-----
00000+
11011
-----
11111
11011+
11011-
-----
1
1010001+
11011- -
-----
-
P → 10111101
  
```



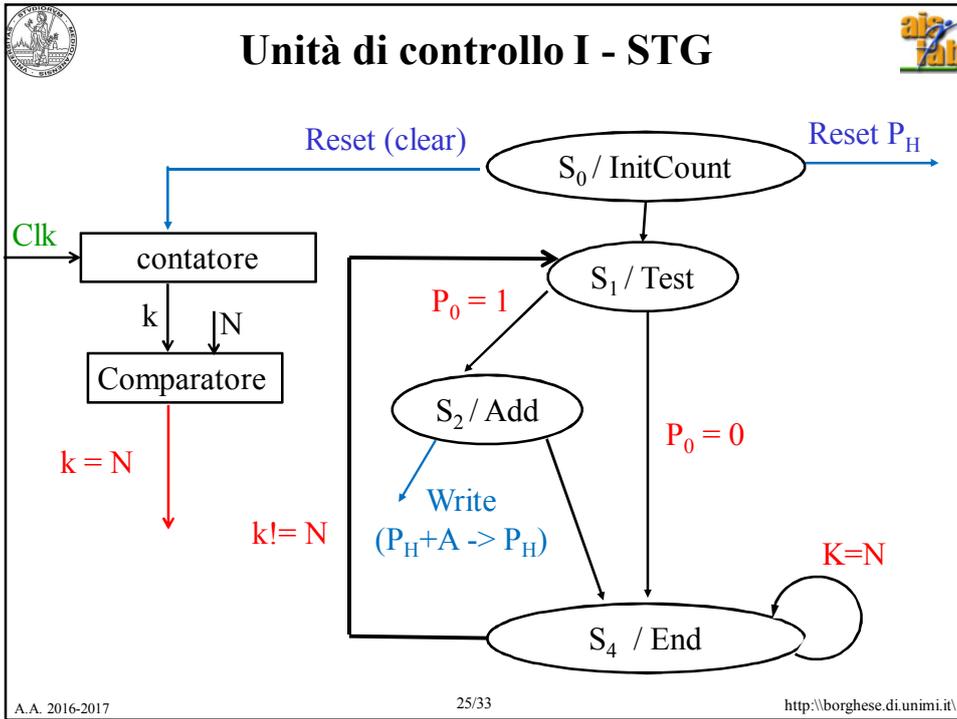
## Esempio di esecuzione dell'algoritmo ottimizzato - III



Iterazione	Passo	Moltiplicando	Prodotto
0	Valori iniziali	0010	0000 0010
1	1a: 1 ⇒ Prod = Prod + Mcando	0010	0010 0011
	2: Scala a destra Prodotto	0010	0001 0001
2	1a: 1 ⇒ Prod = Prod + Mcando	0010	0011 0001
	2: Scala a destra Prodotto	0010	0001 1000
3	1: 0 ⇒ Nessuna operazione	0010	0001 1000
	2: Scala a destra Prodotto	0010	0000 1100
4	1: 0 ⇒ Nessuna operazione	0010	0000 1100
	2: Scala a destra Prodotto	0010	0000 0110

Il moltiplicando è allineato (e sommato) ai bit più significativi del prodotto.



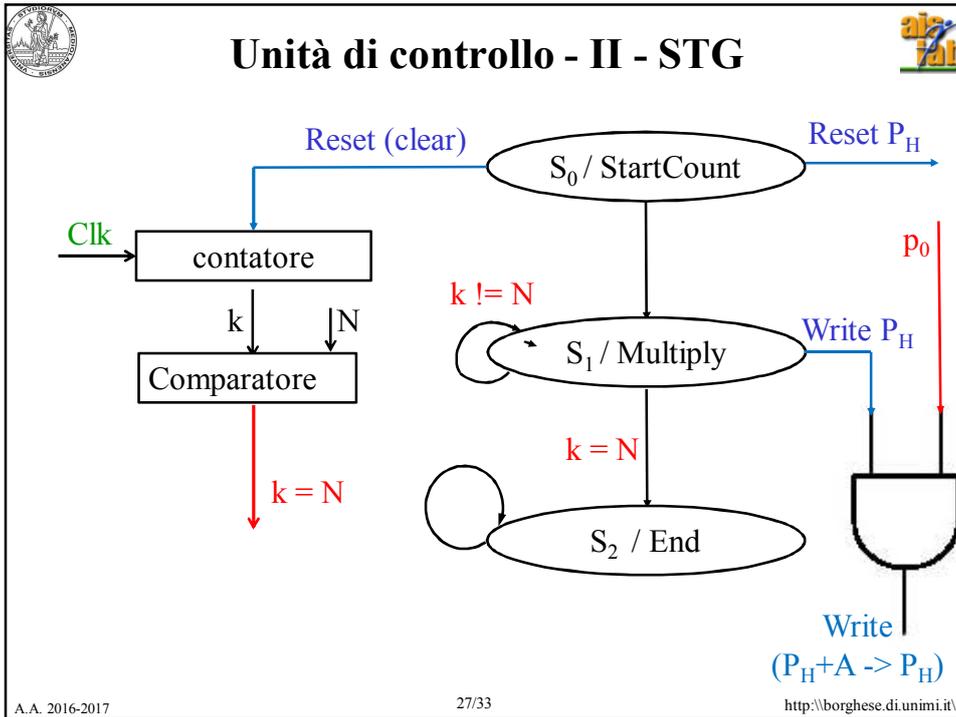


### Unità di controllo I - STT

$S = \{\text{InitCount, Test, Add, End}\}$   
 $I = \{k, P_0\}$   
 $Y = \{\text{Write } P_H, \text{Reset } P_H\}$

	$k \neq N$ $P_0 = 0$	$k = N$ $P_0 = 0$	$k \neq N$ $P_0 = 1$	$k = N$ $P_0 = 1$	Uscita
InitCount	Test	Test	Test	Test	Clear counter Reset $P_H$
Test (sub-step a)	End	End	Add	Add	
Add	Test	Test	Test	Test	Write $P_H$
End (sub-test b)	Test	End	Test	End	

A.A. 2016-2017 26/33 http://borghese.di.unimi.it/



**Approcci tecnologici alla ALU**

Quattro approcci tecnologici alla costruzione di una ALU (e di una CPU):

- **Approccio strutturato.** Analizzato in questa lezione.
- **Approccio hardware programmabile** (e.g. PLA). Ad ogni operazione corrisponde un circuito combinatorio specifico.
- **Approccio ROM.** E' un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l'uscita. E' utilizzabili per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.
- **Approccio firmware** (firm = stabile), o microprogrammato. Si dispone di circuiti specifici solamente per alcune operazioni elementari (tipicamente addizione e sottrazione). Le operazioni più complesse vengono sintetizzate a partire dall'algoritmo che le implementa.

A.A. 2016-2017 28/33 <http://borghese.di.unimi.it/>



## L'approccio firmware



Nell'approccio firmware, viene inserita nella ALU una unità di controllo e dei registri.

L'unità di controllo attiva opportunamente le unità aritmetiche ed il trasferimento da/verso i registri. Approccio "*controllore-datapath*".

Viene inserito un microcalcolatore dentro la ALU.

Il primo microprogramma era presente nell'IBM 360 (1964).



## L'approccio firmware vs hardware



La soluzione HW è più veloce ma più costosa per numero di porte e complessità dei circuiti.

La soluzione firmware risolve l'operazione complessa mediante una sequenza di operazioni semplici. E' meno veloce, ma più flessibile e, potenzialmente, adatta ad inserire nuove procedure.

La soluzione HW è percorsa per le operazioni frequenti: la velocizzazione di operazioni complesse che vengono utilizzate raramente non aumenta significativamente le prestazioni (legge di Amdahl).



## Sommario



I moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware