Outline of Lecture

- **Multiplication Algorithms**
- **Mutiplication Hardware**

Multiplication

- Now we want to include ***multiplication*** in the design of our ALU - multiplication is much more complicated than addition or subtraction.
- **More complicated than addition**

  - accomplished via shifting and addition
- **More time and more area**

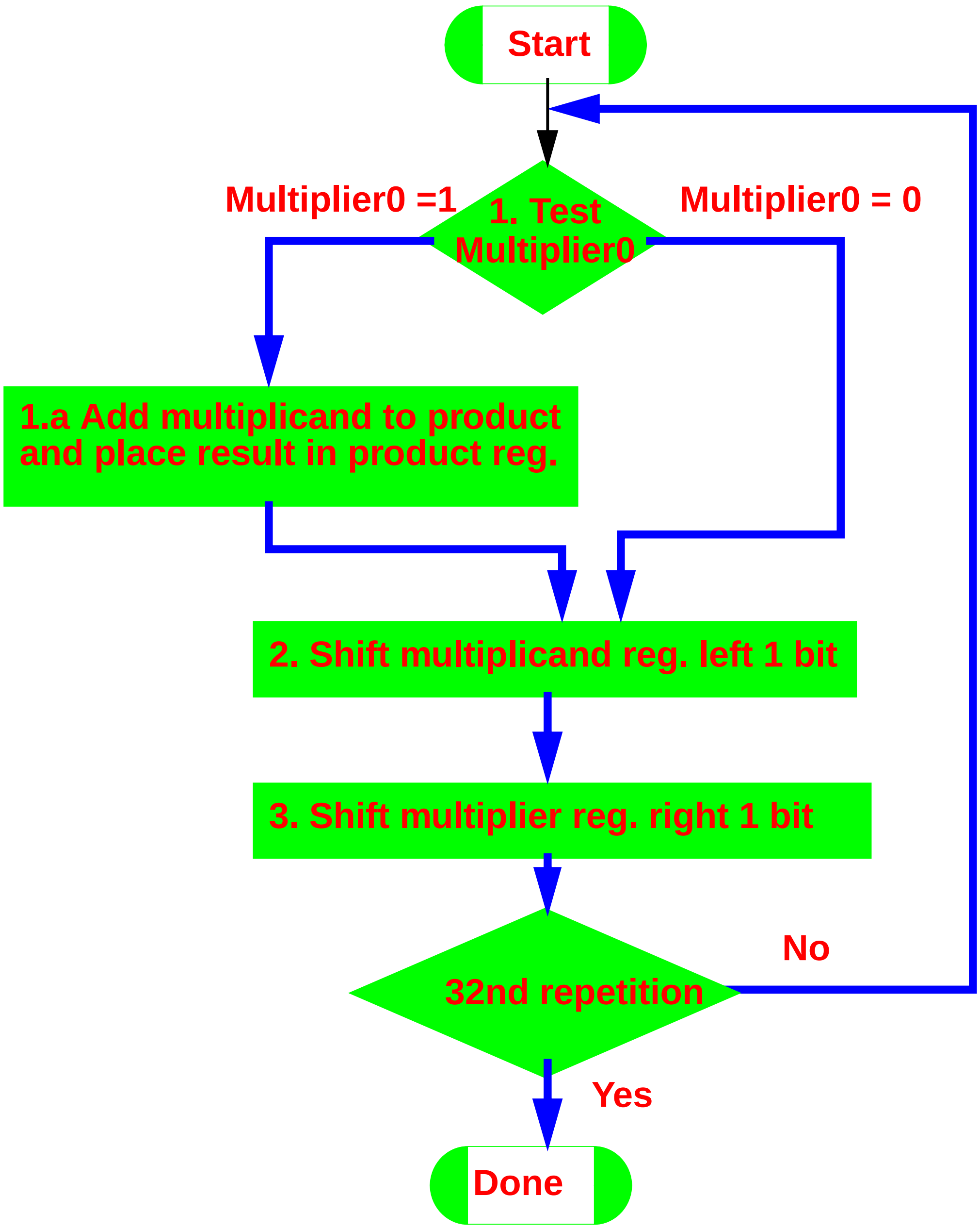
Simple Example

- **Paper-and-pencil example:**

```
Multiplicand          1 0 0 0
Multiplier            1 0 0 1
                    ---------
                      1 0 0 0
                    0 0 0 0
                  0 0 0 0
                1 0 0 0
                -------------
Product         1 0 0 1 0 0 0
```

- **m bits $\times$ n bits = $m + n$ bit product**
- **if the digit is 0 $\Rightarrow$ place (0 $\times$ multiplicand)**
- **if the digit is 1 $\Rightarrow$ place (1 $\times$ multiplicand)**

Multiplication



Multiplication Hardware

- **The hardware for multiplication simply follows the flow of the above paper-and-pencil example.**
- **We use a 64-bit Multiplicand register, 64-bit ALU, 64-bit Product register, and a 32-bit Multiplier register.**

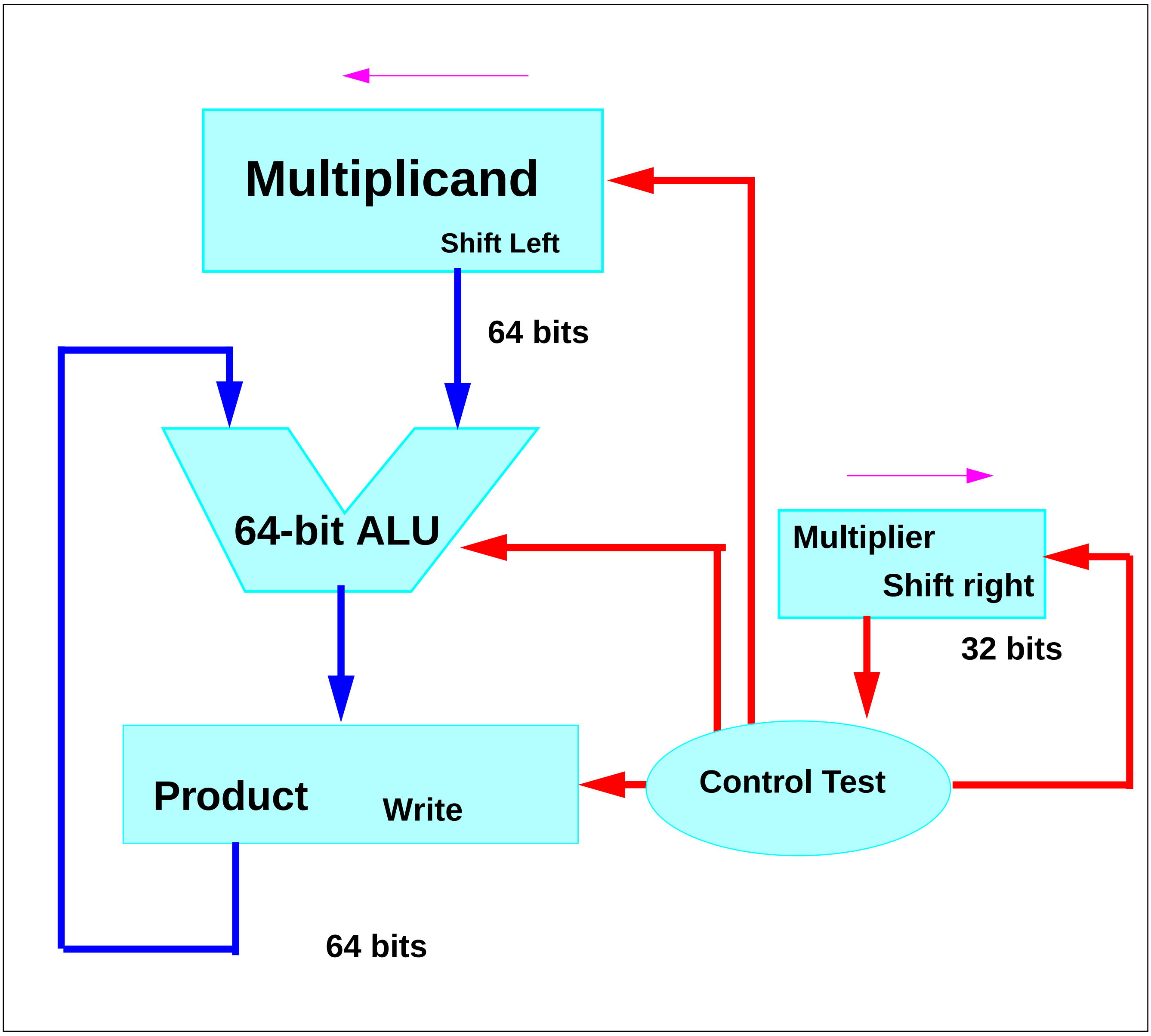
→ **The 32-bit multiplicand starts in the right half of the multiplicand register, and is shifted left 1 bit on each step.**

→ **The multiplier is shifted in the opposite direction at each step.**

→ **The product is initially 0.**

→ **The control decides when to shift the Multiplicand and Multiplier registers and when to write new values into the product register.**

Multiplication Hardware



Example

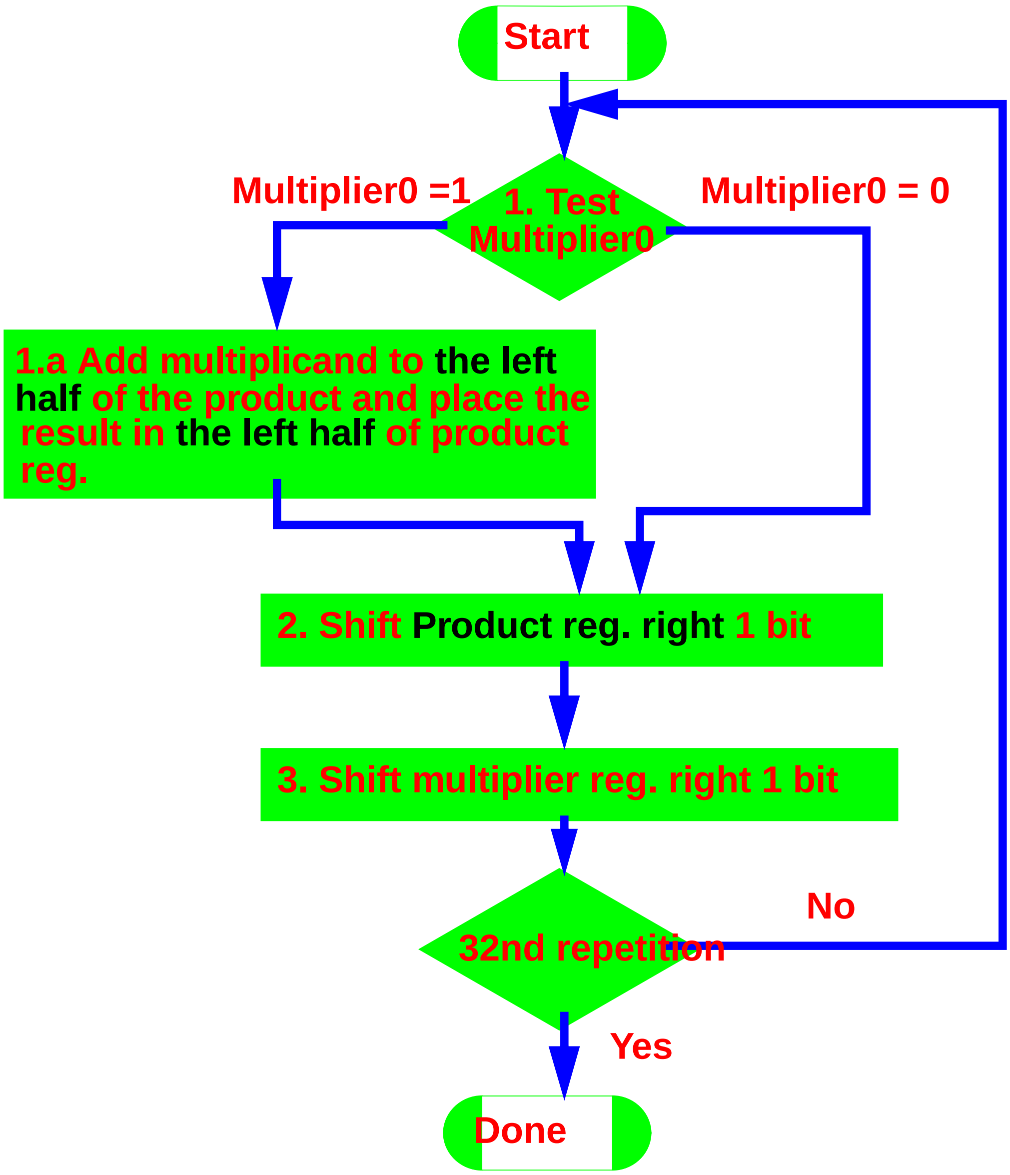
Using 4-bit numbers, multiply 0010_2 x 0011_2.

Solution

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial Values	0011	0000 0010	0000 0000
1	1a: 1⇒ Prod=Prod+Mcand	0011	0000 0010	0000 0010
	2: Shift left multiplicand	0011	0000 0100	0000 0010
	3: Shift right multiplier	0001	0000 0100	0000 0010
2	1a: 1⇒ Prod=Prod+Mcand	0001	0000 0100	0000 0110
	2: Shift left multiplicand	0001	0000 1000	0000 0110
	3: Shift right multiplier	0000	0000 1000	0000 0110
3	1: 0 ⇒ no operation	0000	0000 1000	0000 0110
	2: Shift left multiplicand	0000	0001 0000	0000 0110
	3: Shift right multiplier	0000	0001 0000	0000 0110
4	1: 0 ⇒ no operation	0000	0001 0000	0000 0110
	2: Shift left multiplicand	0000	0001 0000	0000 0110
	3: Shift right multiplier	0000	0010 0000	0000 0110

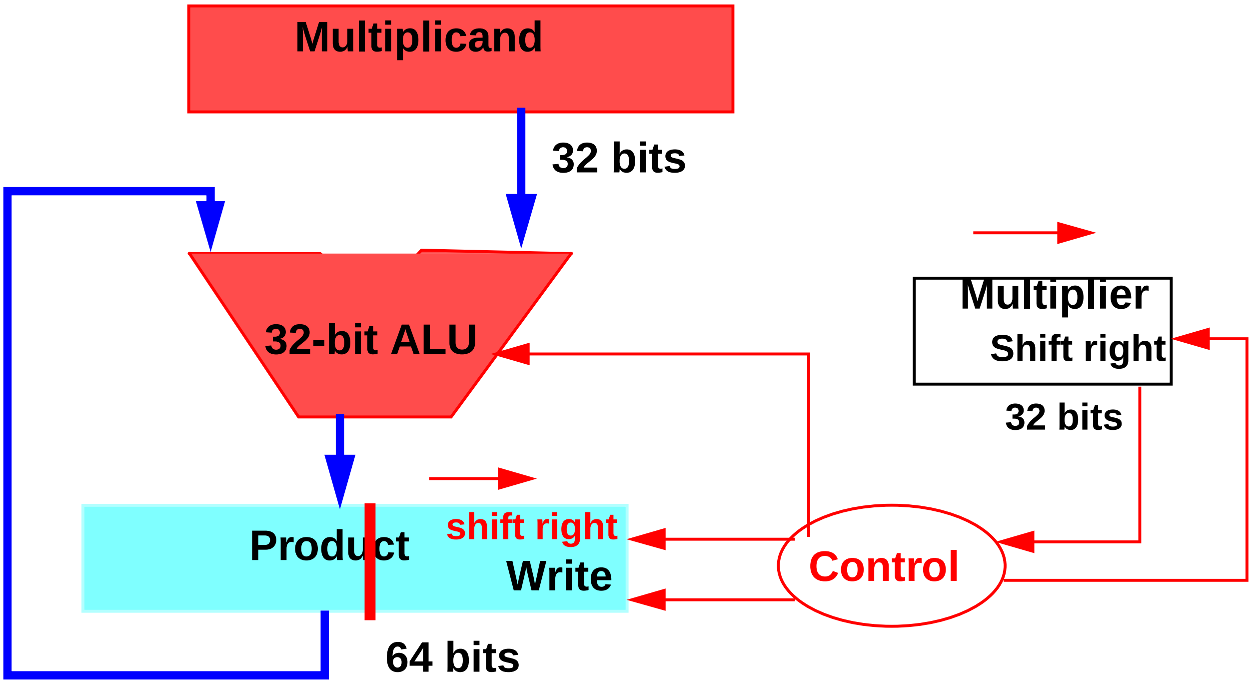
Second Version

- The hardware for the multiplication needs a 64-bit ALU. However, since half of the bits of the multiplicand are always 0 - we can achieve the same functionality by simply using a 32-bit ALU.



Second Version

- The hardware for this multiplication needs just 32-bit ALU - it can be implemented using our original MIPS ALU.



Signed Multiplication

- For ***signed multiplication***: we simply convert the multiplier and multiplicand to positive numbers, and we remember the original signs. The algorithm will run for 31 iterations. After that, we restore the sign.

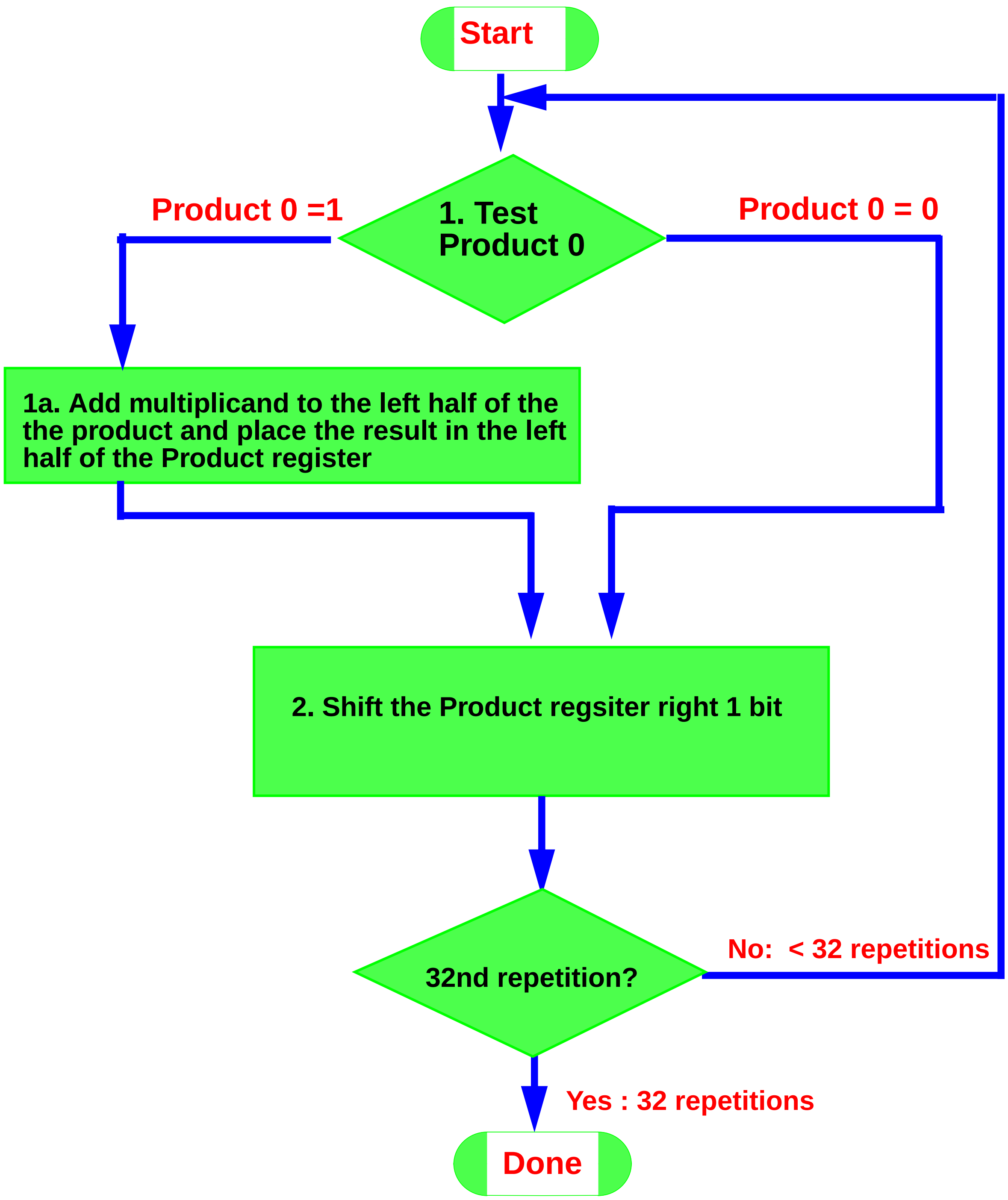
Example

Using 4-bit numbers, multiply 0010_2 x 0011_2.

Solution

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial Values	0011	0010	0000 0000
1	1a: 1⇒ Prod=Prod+Mcand	0011	0010	0010 0000
	2: Shift right product	0011	0010	0001 0000
	3: Shift right multiplier	0001	0010	0001 0000
2	1a: 1⇒ Prod=Prod+Mcand	0001	0010	0011 0000
	2: Shift right product	0001	0010	0001 1000
	3: Shift right multiplier	0000	0010	0001 1000
3	1: 0 ⇒ no operation	0000	0010	0001 1000
	2: Shift right product	0000	0010	0000 1100
	3: Shift right multiplier	0000	0010	0000 1100
4	1: 0 ⇒ no operation	0000	0010	0000 1100
	2: Shift right product	0000	0010	0000 0110
	3: Shift right multiplier	0000	0010	0000 0110

Final Version



Final Version Hardware