乘法器种類:

①. 無号數乘法: ┬ 傳統乘法 (traditional sequential multiplication)
└ 硬体最佳化乘法 (hardware - friendly multiplication)

②. 有号數乘法: Booth's Algorithm

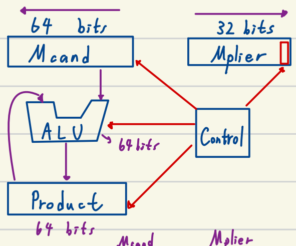
(1). 傳統乘法 (traditional sequential multiplication)

設: $a \times b = c$: a為 multiplicand c: product
b為 multiplier

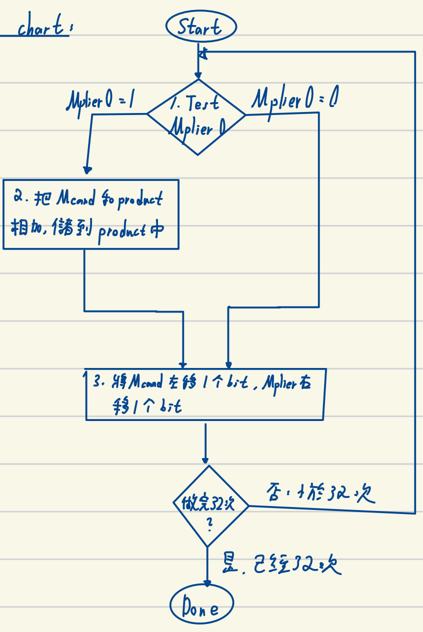
e.g
```
      0101     Mcand
    × 1111     Mplier
    ------
      0101
     0101
    0101
   0101
  ---------
  01011011     Product
```

需要元件: ①. Mcand register
②. Product register
③. ALU
④. control test

乘法器硬体:



Flow chart:



Start
Mplier 0 = 1
1. Test Mplier 0
Mplier 0 = 0
2. 把 Mcand 和 product 相加, 儲到 product 中
3. 將 Mcand 左移 1 个 bit, Mplier 右移 1 个 bit
做完32次?
否: 小於32次
是, 已经32次
Done

Tracing: e.g Mcand $0010_{(2)}$ × Mplier $0011_{(2)}$

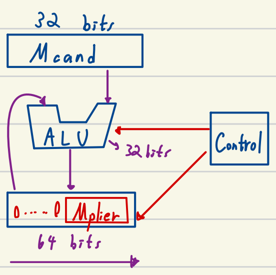
1. 需: Mcand: 8 bit, Mplier 4 bit, Product: 8 bit
2. 需: 4 輪

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial value	0011	00000010	00000000
1	1 → Product + Mcand	0011	00000010	00000010
	Multicand left shift	0011	00000100	00000010
	Multiplier right shift	0001	00000100	00000010
2	1 → Product + Mcand	0001	00000100	00000110
	Mcand left shift	0001	00001000	00000110
	Mplier right shift	0000	00001000	00000110
3	0 → No operation			00000110
4	0 → No operation			
Result				$\boxed{00000110}$

(2). Hardware - friendly multiplication

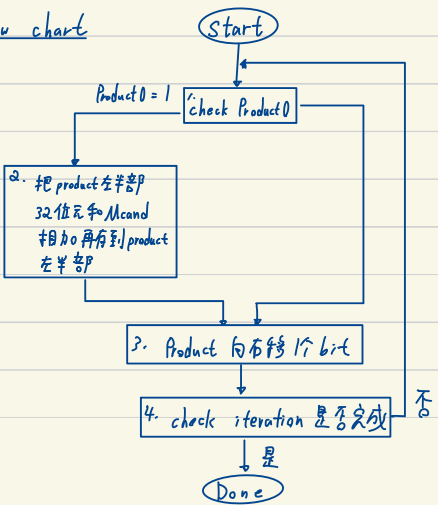
(1).
```
      0001
    × 1111
    ------
      0001
      ──→
       0001
    +  0001
    --------
      00011
      ───→
       00011
    +  0001
    ---------
      000111
      ────→
       000111
    +  0001
    ----------
    [00001111]
```

(2). 乘法器設計:



(3). 優點: ①. ALU 只需要32位元
②. 不用有額外的 register 存放 Mplier
③. Control 不用控制 Mcand

(4). Flow chart



Start
Product 0 = 1
1. check Product 0
2. 把 product 左半部 32位元和 Mcand 相加再存到 product 左半部
3. Product 向右移 1个 bit
4. check iteration 是否完成
否
是
Done

Tracing: $0010_{(2)}$ × $0011_{(2)}$

iteration	step	Mcand	Product
0	initial state	0010	00000011
1	1 → prod = prod + Mcand	0010	00100011
	prod right shift	0010	00010001
2	1 → prod = prod + Mcand	0010	00110001
	prod right shif	0010	00011000
3	0 → No operation + rll		00001100
4	0 → No operation + rll		00000110

(3). Booth's Algorithm:

①. a
$\times\ b = 00111100$

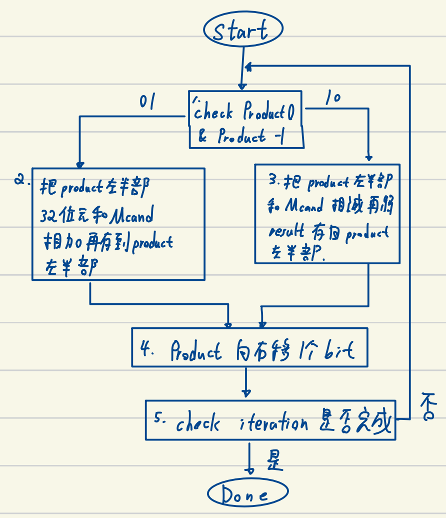
傳統: $a \times (2^5+2^4+2^3+2^2)$
$= a \times 60$

Booth's: $a \times (2^6 - 2^2)$
$= a \times 60$

eg. 0011 × 0110

其中: 01: 作加法運算
10: 作減法運算

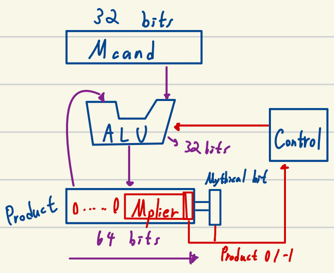
②. flow chart:



Start
01
10
check Product 0 & Product -1
2. 把 product 左半部 32位元和 Mcand 相加再存到 product 左半部
3. 把 product 左半部 和 Mcand 相減再將 result 存回 product 左半部.
4. Product 向右移 1 bit
5. check iteration 是否完成
否
是
Done

③. 加法器:

※ 注意有 Mythical bit 存在, 來做 dummy 0



⑥. Tracing. 0010 × 1101 ⇒ 4 iterations

iteration	step	Mcand	Product
0	initial step	0010	00001101 0
1	10 → prod = prod - Mc	0010	11101101 0
	prod rll	0010	11110110 1
2	01 → prod = prod + Mc	0010	00010110 1
	prod rll	0010	00001011 0
3.	10 → prod = prod - Mc	0010	11101011 0
	prod rll	0010	11110101 1
4.	X		
	prod rll		11111010 ✗

$\underline{11111010}$
⇓
11111010

0000
+ 1110
$\overline{1110}$

1111
0010
$\overline{0001}$

⑤. Booth's Algo 的優點:

(1). 可以做有號數乘法

(2). 平均上 速度較傳統 & 硬件最佳化乘法快
(但不是 Always, ∵ 當 Multiplier 為: 0101 時 需做 4次 加法運算, 但傳統只需 2次)

⑤. 有效性證明

設為兩 8 bit 有號數相乘

b
$\times\ a \quad a_7a_6\ldots a_3a_2a_1a_0\overset{\text{dummy zero}}{a_{-1}}$

$\Rightarrow (a_{-1}-a_0)\times b\times 2^0$
$+ (a_0 - a_1)\times b \times 2^1$
$\vdots$
$+ (a_6 - a_7)\times b \times 2^7$

$b[a_{-1} - a_0 + 2a_0 - 2a_1 + 4a_1 - 4a_2 + 8a_2 \cdots]$
$= b[a_{-1} + a_0 + 2^1a_1 + 2^2a_2 + \cdots + 2^6a_6 - 2^7a_7]$
$= b[-2^7a_7 + 2^6a_6 + \cdots + 2^0a_0]$
$= b \times a_7a_6a_5\cdots a_1a_0$
$= b \times a$ #.

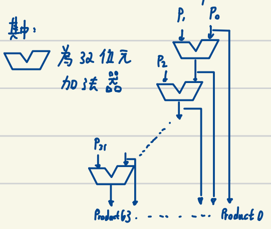
a_i a_{i-1}	result
1 1	X
0 0	X
0 1	b
1 0	-b

(4). Difference between sequential & combinational multiplication

sequential: 如 傳統, 硬体最佳化, Booth's Algorithm 設計之乘法器
需利用 sequential circuit 設計, 會依賴同步訊號更新變數的值
因此在做乘積的每一位元時, 需花一个 clock 的時間來同步所有記憶元件改变, 速度較慢

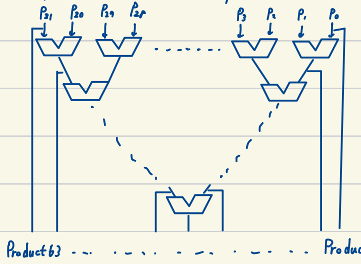
combinational: 如: fast, parallel fast multiplier
電路中的任何訊号改变會直接反映在其它訊号上
∴ 不需要 clock 來進行同步, 速度較快.
且較易優化和管線化, 但只能用作無号數乘法

①. fast multiplier



設加法時間為 T, 則此乘法器為 31T
∵ 共有 32-1 个加法器

②. parallel fast multiplier



為 1个有 31个 Node 的 full BT
∴ $2^i - 1 = 31$ $i = 5$ 為 height of tree
∴ 設加法器運算時間為 T, 則 parallel fast multiplier 的時間為 5T
但加法器同為 31个, 硬体成本一致

Sequential 加法器統整 Check $\longrightarrow$ Action $\longrightarrow$ Shift

	Traditional		Hardware-friendly		Booth's Algorithm			
check	Mplier 0		Product 0		product 0. $\underline{-1}$ mythical bit			
	0	1	0	1	00	01	10	11
action	X	Product ← Product + Mcand	X	Product 左半部 ← Product 左半部 + Mcand	X	L(p) ← L(p) + Mcand	L(p) ← L(p) - Mcand	X
shift	Mcand ← Mplier →		Product →		Product →			
優點	X		1. 不需要 Mplier register 2. ALU 只要 32 bit 3. 不需要 shift Mcand		1. 可作有號數乘法 2. 平均上運算速度較快			

實例比較乘法器速度

一. Traditional

基本假設: 設兩 A bits 的數在作乘法, 且每個 operation 需 B time unit
且總是做加
若用 HW 來 shift register 的話可同時做

Step:
0. check Multiplier 0 (可省略, ∵總是為1)
①. Multiplicand + Product : 1 clock
②. shift Multiplier & Mcand : 1 clock
③. check 是否完成: 1 clock (∵要實作一定有一 counter register
既然為 memory unit, 需要 clock 控制)

∴ 共需: $(3 \times A) \times B$ 個 time units.

二. Fast Multiplication:
需做 A-1 次的加法, ∴為: $(A-1)B$ 個 time units.

三. Parallel Multiplication
需做: H 次的加法, 其中: $2^H = A$ ∴ $H = \lceil \lg A \rceil$, 為 $\lceil \lg A \rceil B$ 個 time units.
(∵只需 A-1 個 Adder
∴ $2^H - 1 = A - 1 \Rightarrow 2^H = A$)