第四章 功率控制之機制

在本章將描述功率意識乘法器的功率控制機制，我們採用双節制技術來取得功率意識最佳化。所謂双節制技術，就是將節制(truncating)部位分為二種，一種是節制乘法器較低位元的乘積(product)，另外一種同時還節制乘法器的輸入:被乘數(multiplicand)與乘數(multiplier)。

圖 35 為乘法的部分乘積(partial product)示意圖，圖 35(a)為節制乘積，黑色區域即為節制的部分。因為 DSP 乘法器的輸出通常不會取所有的位元，而是取較高的位元。根據這個特性，我們節制乘積的較低位元，對於品質效能的影響也較小。

圖 35(b)則為節制輸入，其中斜的黑色區域為節制被乘數，橫的黑色區域為節制乘數。

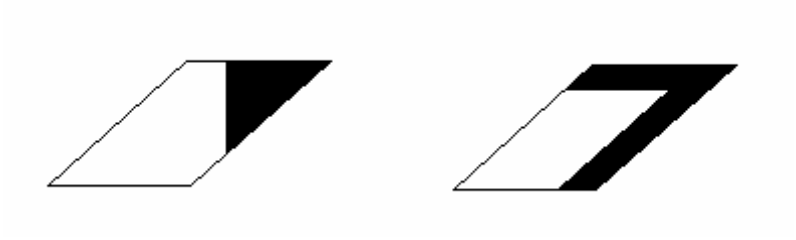

圖 35 (a)節制乘積 (b)節制輸入

我們除了將節制的部位分為二種之外，另外節制的方法又可分為二種，一種是節制的部位設為零，另一種是節制的部位保留前值，因

為這二種方法可減少加法器的開關動作(switching activities)，因此皆可降低功率的消耗。綜合二種節制部位與二種節制方法，總共有四種組合的方式。4.1 節與 4.2 節將分別介紹節制乘積與節制輸入，4.3 節介紹組合節制乘積與節制輸入的四種方式。

4.1 節制乘積

節制的方法可分為二種，一種是節制的部分設為零，另一種是節制的部分保留前值，本論文是以 10000 筆隨機輸入做實驗，保留前值我們定義為保留第一筆的資料，在本節將介紹節制乘積的二種方法。

1. 乘積設為零(Products remain Zero, PZ)

將部分乘積較小的次要位元(lower bits)設為零，如圖 36 所示。其中 Si, j= Xi AND Yj。節制乘積 P0 至 Pn-k-1 bit，將其部分乘積設為零，最後的乘積只取 n bit，亦即取 Pn 至 P2n-1 的乘積。

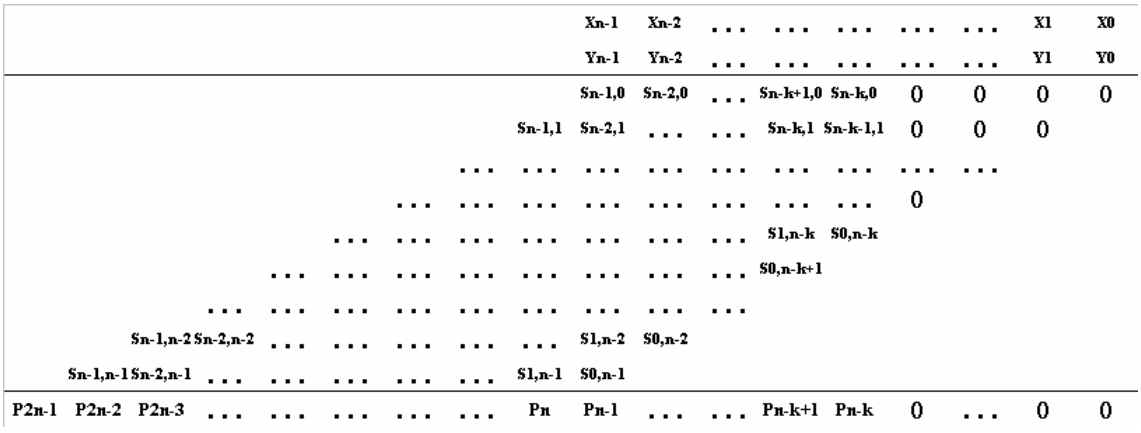

圖 36　乘積設為零之 n × n bit 乘法

以 16 × 16 bit 正負號乘法為例，因為 S15,0、S15,2、……、S15,14 以及 S0,15、S0,1、……、S14,15 為產生正負號位元(signed bit)的部分乘積，若節制 P15 到 P31 bit 的乘積將影響正負號位元的結果，因此最多節制乘積 P0 到 P14 bit。

在硬體的實現方面，我們將圖 32 的管線暫存器，加上 disable 訊號修改為防衛閂(guard latch)，如圖 37 所示，其中 disable 為節制乘積的控制訊號，因此我們將 1st 管線暫存器變成防衛閂以做為節制乘積之用。

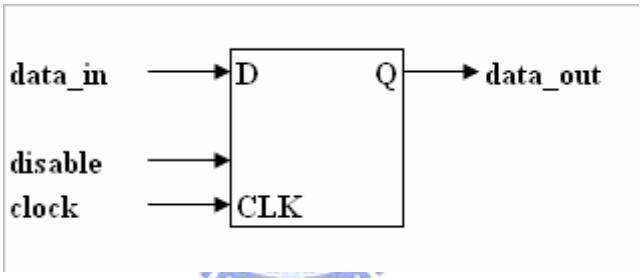

圖 37　節制乘積之防衛閂

乘積設為零的防衛閂 VHDL code 敘述如下:

```
library ieee;
use ieee.std_logic_1164.all;

entity guard_latch is
 port( data_in: in std_logic;
```

```
        disable,clk: in std_logic;
        data_out: out std_logic);
end guard_latch;
architecture  beha of guard_latch is
begin
 process(data_in,disable,clk)
 begin
   if clk'event and clk='1' then
     if disable='0' then
      data_out <= data_in;
     else
      data_out <=  '0' ;
     end if;
   end if;
 end process;
end beha;
```

乘積設為零的防衛閂動作機制為:當 disable＝0 時,則 data_out ＝data_in。當 disable＝1 時，表示啟動節制的機制，則 data_out

＝0。以本論文實驗的 16 × 16 bit 管線正負號 Dadda 乘法器，因為最多節制乘積 P0 到 P14 bit，所以將 P0 到 P14 bit 的 1st 管線暫存器修改為防衛閂，用以節制乘積，而 P15 到 P31 bit 的 1st 管線暫存器仍維持圖 32 的管線暫存器。

2. 乘積保留前值(Products remain Data, PD)

將部分乘積較小的次要位元保留第一筆的資料，相較於圖 36 的 PZ 法，設為零的部分改為保留第一筆的資料。在硬體的實現方面，跟 PZ 法一樣使用圖 37 的防衛閂，只是內部結構不同，其 VHDL code 敘述如下:

```
library ieee;
use ieee.std_logic_1164.all;

entity guard_latch is
 port( data_in: in std_logic;
       disable,clk: in std_logic;
       data_out: out std_logic);
end guard_latch;
```

```
architecture  beha of guard_latch is
begin
 process(data_in,disable,clk)
 begin
   if clk'event and clk='1' then
     if disable='0' then
      data_out <= data_in;
     end if;
   end if;
 end process;
end beha;
```

比較 PZ 與 PD 防衛閂的 VHDL code 差別，在於 PZ 多了 disable ≠0 時 data_out＝0 的描述，如 code 中黑體字所述。而 PD 是保留前值，因此不需要這層描述。

乘積保留前值的防衛閂動作機制為:當 disable＝0 時，則 data_out＝data_in。當 disable＝1 時，則 data_out 為前一個 clock 的 data_in。因為我們將保留前值設定為保留第一筆的資料，所以在第一個 clock cycle 將 disable＝0，存入第一筆的資料，在第二個

clock cycle 將 disable＝1，則 data_out 維持第一筆的資料。

如同 PZ 法,16 × 16 bit 的正負號乘法器最多節制乘積 P0 到 P14 bit，所以將 P0 到 P14 bit 的 1st 管線暫存器修改為防衛閂，用以節制乘積，而 P15 到 P31 bit 的 1st 管線暫存器仍維持圖 32 的管線暫存器。

4.2 節制輸入

乘法器的輸入為被乘數與乘數，以 I 代表輸入。我們將節制輸入代表同時節制被乘數與乘數。

16 × 16 bit 的正負號乘法器，其輸入的 MSB (I15)為 signed bit，因此輸入最多節制到 I13 bit。我們將節制輸入亦分為輸入設為零與輸入保留前值二種方法，下面將分別做介紹。

1.輸入設為零(Inputs remain Zero, IZ)

同時將被乘數與乘數較小的次要位元(lower bits)設為零，如圖 11 所示，將被乘數與乘數 I0 至 In-k-1 bit 設為零，最後的乘積只取 n bits，亦即取 Pn 至 P2n-1 的乘積。

在硬體的實現方面,我們將圖 34 的輸入暫存器(input register)設計成圖 38 所示，由修改過的輸入暫存器來做節制輸入之用，其中

X_in、Y_in 代表 16 bit 的被乘數與乘數，X、Y 為節制輸入之後 16 bit 的結果，而 sel_input 為節制輸入的控制訊號，因最多節制到 I13 bit，所以 sel_input 為 4 bit。

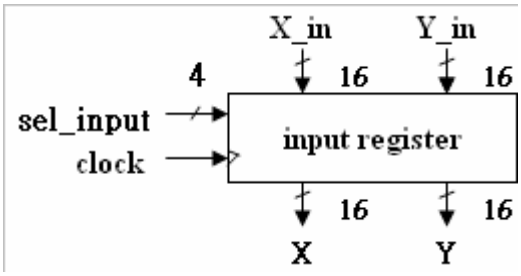

圖 38　節制輸入之輸入暫存器

輸入設為零之輸入暫存器動作機制為:當 sel_input＝0 時，節制輸入 I0 bit，則 X(0)＝Y(0)＝0，而且 X(15:1)＝X_in(15:1)，Y(15:1)＝Y_in(15:1)。當 sel_input＝1 時，節制輸入 I0 至 I1 bit，則 X(1:0)＝Y(1:0)＝0，且 X(15:2)＝X_in(15:2)，Y(15:2)＝Y_in(15:2)，……依此類推。因為輸入最多節制到 I13 bit，當 sel_input＝13 時，節制輸入 I0 至 I13 bit，則 X(13:0)＝Y(13:0)＝0，而且 X(15:14)＝X_in(15:14)，Y(15:14)＝Y_in(15:14)。若是未節制任何輸入，則設定 sel_input＝15，即 X(15:0)＝X_in(15:0)，Y(15:0)＝Y_in(15:0)。

2.輸入保留前值(Inputs remain Data, ID)

將被乘數與乘數較小的次要位元保留第一筆的資料，相較於圖

11 的 IZ 法，設為零的部分改為保留第一筆的資料。在硬體的實現方面，跟 IZ 法一樣使用圖 38 的輸入暫存器，只是內部結構不同。

輸入保留前值之輸入暫存器動作機制為:當 sel_input＝0 時，節制輸入 I0 bit，則 X(0)為前一個 clock 的 X_in(0)，Y(0)為前一個 clock 的 Y_in(0)，而且 X(15:1)＝X_in(15:1)，Y(15:1)＝Y_in(15:1)。當 sel_input＝1 時，節制輸入 I0 至 I1 bit，則 X(1:0)為前一個 clock 的 X_in(1:0)，Y(1:0)為前一個 clock 的 Y_in(1:0)，而且 X(15:2)＝X_in(15:2)，Y(15:2)＝Y_in(15:2)，……依此類推。因為輸入最多節制到 I13 bit，當 sel_input＝13 時，節制輸入 I0 至 I13 bit，則 X(13:0) 為前一個 clock 的 X_in(13:0)，Y(13:0)為前一個 clock 的 Y_in(13:0)，而且 X(15:14)＝X_in(15:14)，Y(15:14)＝Y_in(15:14)。若是未節制任何輸入，則設定 sel_input＝15，即 X(15:0)＝X_in(15:0)，Y(15:0)＝Y_in(15:0)。

因為我們將保留前值設定為保留第一筆的資料，所以在第一個 clock cycle 將 sel_input 設定為 15，存入第一筆的資料，在第二個 clock cycle 若節制輸入 n bit，則將 sel_input 設定為 n。

4.3 組合節制乘積與節制輸入的四種方式

綜合 4.1 節與 4.2 節的方法,可以歸納出四種組合方式:

1. 輸入設為零與乘積設為零(IZ-PZ)

將被乘數與乘數較小的次要位元(lower bits)設為零，並且將乘積較小的次要位元設為零。

2. 輸入保留前值與乘積設為零(ID-PZ)

將被乘數與乘數較小的次要位元保留第一筆資料，並且將乘積較小的次要位元設為零。

3. 輸入設為零與乘積保留前值(IZ-PD)

將被乘數與乘數較小的次要位元設為零，並且將乘積較小的次要位元保留第一筆資料。

4. 輸入保留前值與乘積保留前值(ID-PD)

將被乘數與乘數較小的次要位元保留第一筆資料，並且將乘積較小的次要位元保留第一筆資料。

我們將以這四種功率控制方式來做實驗，找出其中一種功率意識最佳的方式，做為功率意識乘法器。

另外，n × n bit 乘法的乘積只取 n bit，亦即取 Pn 至 P2n-1 的乘積，則必須做圍繞(rounding)修正，以降低誤差。我們是以圍繞至最接近(round-to-nearest)來做修正，即將 Pn-1 加到 Pn，再取 Pn 至 P2n-1 做為最後的乘積。以 16 × 16 ＝16 bit 乘法為例，如圖 8 所示，P0 到 P31 為 32 bit 的乘積，但是最後的乘積只取 16 bit，因

此我們保留 P15，將 P15 加到 P16，再取 P16 至 P31 成為輸出 Q0 至 Q15，以做為最後的乘積。

我們以圖 39 的輸出暫存器(output register)做圍繞修正，其中 P 為未做修正的 32 bit 乘積，Q 為修正之後 16 bit 最後的乘積。

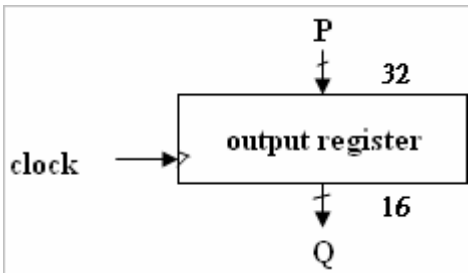

圖 39　輸出暫存器

輸出暫存器的 VHDL code 敘述如下：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all:

entity reg_out is
 port( p: in std_logic_vector(31 downto 0);
       clk: in std_logic;
       q: out std_logic_vector(15 downto 0));
end reg_out;
```

```
architecture  beha of reg_out is
begin
 process(p,clk)
 begin
   if clk'event and clk='1' then
      q <= p(31 down to 16) + p(15);
   end if;
 end process;
end beha;
```

組合節制乘積與節制輸入四種方式之功率意識乘法器線路圖如圖 40 所示， X_in、Y_in 分別代表 16 bit 被乘數與乘數，Q 代表 16 bit 乘積，sel_input 為節制輸入的訊號，disable 為節制乘積的訊號，sel_product 為控制 disable 的訊號，以下介紹各方塊的功用。

1. input register:

同 4.2 節介紹的節制輸入之輸入暫存器，由 sel_input 用以節制輸入。因為我們將保留前值設定為保留第一筆的資料，所以若是使用

ID 法則在第一個 clock cycle 將 sel_input 設定為 15，存入第一筆的資料，在第二個 clock cycle 若節制輸入 n bit，則將 sel_input 設定為 n。若是使用 IZ 法節制輸入 n bit，則直接將 sel_input 設定為 n。

2. and gate array:

AND 閘所構成之陣列，用以產生 256 個部分乘積，另外加上 NOT 閘，以產生正負號(signed)乘法器。

3. Dadda tree with 6 stages pipelined register:

Dadda tree 乘法器，包括 CPA 的 Brent-Kung CLA，並且插入 6 stages 管線暫存器(pipelined register)。其中 P0 至 P14 列的 1st 管線暫存器修改為防衛閂，加入 disable(14:0)控制防衛閂 P14 至 P0 bit，用以節制乘積。

4. output register:

同本節介紹的輸出暫存器，將 32 bit 乘積做圍繞至最接近(round-to-nearest)的修正來產生最後 16 bit 乘積。

5. combination circuit:

為了易於控制防衛閂的 disable 訊號，我們加入 combination circuit，以 sel_product 控制 disable 訊號。其動作機制為:當 sel_product＝0 時，節制乘積 P0 bit，則 disable(0)＝1，

disable(14:1)＝0。當 sel_product＝1 時，節制乘積 P0 至 P1 bit，則 disable(1)＝disable(0)＝1，disable(14:2)＝0，……依此類推。因為乘積最多節制到 P14 bit，當 sel_product＝14 時，節制乘積 P0 至 P14 bit，則 disable(14)＝……＝disable(0)＝1。若是未節制任何乘積，則設定 sel_product＝15，即 disable(14:0)＝0。

因為我們將保留前值設定為保留第一筆的資料，所以若是使用 PD 則在第一個 clock cycle 將 sel_product 設定為 15，存入第一筆的資料，在第二個 clock cycle 若節制乘積 n bit，則將 sel_product 設定為 n。若是使用 PZ 節制乘積 n bit，則直接將 sel_product 設定為 n。

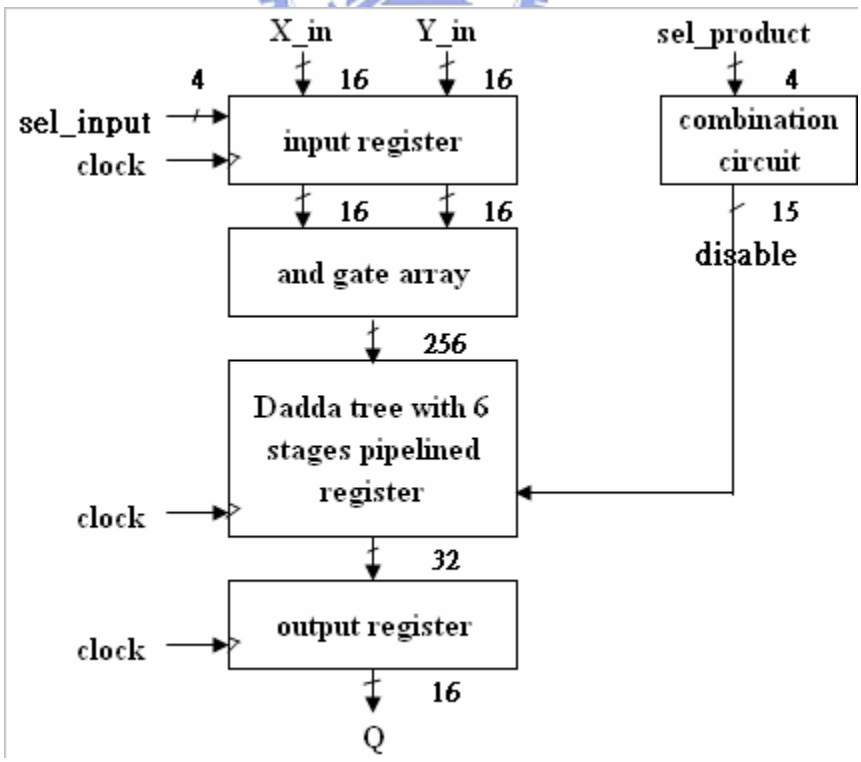

圖 40　組合節制乘積與節制輸入四種方式之功率意識乘法器