如何使用文字編輯器編輯文字向量檔(*.PGV)

可程式化資料產生器**(Programmable Data Generator)**以下簡稱 PG, 除了可讀取波形檔**(*.PGW)**之外, 還可以讀取以文字編輯器所編輯的文字檔, 而該文字檔**(.*PGV (PG Vector File))**的內容主要是您想要觀察的一些資料以及給 PG 的指令等。

而該文字檔有一定的格式, 以下的內容就該文字檔的格式及使用方法做說明, 首先以下為擷取某文字向量檔的部分來做為範例並透過它來說明 **PGV** 檔的格式部分。

文字向量檔還有根據您所下的指令(Keyword)的不同而有不一樣的檔案內容, 在定義波形資料的區域, 可以區分為使用『Time Stamp』與否。除了介紹基本指令, 還有許多在應用上會使用到的 PG_Function 訊號組的波形指令, 整個波形指令總共有 7 個，包括 **NP(No Operation)**、**JP(Jump)**、**LP(Loop)**、**BE(Branch if Event)**、**LC(Loop Count)**、**SE(Set Event)**和 **WE(Wait Event)**等。以下會針對這些指令來做說明:

```
INPUTS PG_Function DATA;
ASSIGN DATA 3..0;
RADIX AUTO;
FREQUENCY 1000 Hz;
%INTERVAL 1ms;%                    『%..%』:註解符號
PATTERN

8FFh      0h   //  0              ( MOV RL, 255 )
2FFh      0h   //  1              ( MOV RH, 255 )
900h      0h   //  2               OE 65535
000h      0h   //  3
000h      0h   //  4
000h      0h   //  5          『//』:註解符號
000h      0h   //  6
000h      0h   //  7
000h      0h   //  8
000h      0h   //  9
000h      0h   // 10            START PATTERN
000h      1h   // 11
000h      2h   // 12
000h      3h   // 13
000h      4h   // 14
000h      5h   // 15
000h      6h   // 16
000h      7h   // 17
000h      8h   // 18
000h      9h   // 19
000h      Ah   // 20
000h      Bh   // 21
000h      Ch   // 22
816h      Dh   // 23            ( MOV RL, 22 )
200h      Eh   // 24            ( MOV RH, 0 )
100h      Fh   // 25             JP 10
000h      0h   // 26
;
```

上述的範例為一個 4Bits、1KHz 計數器的應用。我們就以上述範例先來說明 PGV 的指令:

INPUT PG_Function DATA

設定訊號名稱。每一個名稱用空白分隔開來，名稱可以為文字或是數字如果為 Bus(Group)時，可以用中括號來指定。例如: 某個 Bus 為 4 個通道所組成那就可以用 A[3..0]來代替，A[3..0]就代表 A3、A2、A1、A0 等 4 個訊號。

注意！『PG_Function』是一個保留字，如果在 INPUTS 指令的定義中有『PG_Function』，就代表 PATTERN 中會包含 PG 的專用指令。而且『PG_Function』也不能配合 ASSIGN 指令使用。

ASSIGN DATA 3..0

用來指定 INPUTS 指令所定義的訊號名稱之通道值。結果如下

DATA0 = CH0 , DATA1 = CH1……DATA3= CH3 等。

RADIX AUTO

RADIX 為區域的進位值。設定 PATTERN 區域的進位值。如果 PATTERN 區域的數值有進位識別符號時，此時就要將 RADIX 的值設成 AUTO。例如 RADIX 設定成 AUTO 時，PATTERN 區域的某一數值為 35 與 35h 是不一樣的，但是當 RADIX 設成 HEX(十六進位)時，這兩個值就是一樣的。但是當 RADIX 設成 DEC 時，而 PATTERN 區域的值為 35h 時，

卻是會被當成 35。

RADIX 共有五種定義:

AUTO ： 由數值的進位識別符號決定。

HEX ： 16 進位。

DEC ： 10 進位。

OCT ： 8 進位。

BIN ： 2 進位。

而在 PATTERN 區域的進位識別符號定義是『h』代表 16 進位、『o』代表 8 進位和『b』代表 2 進位，而不加任何識別符號就代表 10 進位。

FREQUENCY 1000 Hz

這裡是指定使用 PG 主頻為 1000Hz, 主要為了設定計數器的頻率 1KHz。

PATTERN

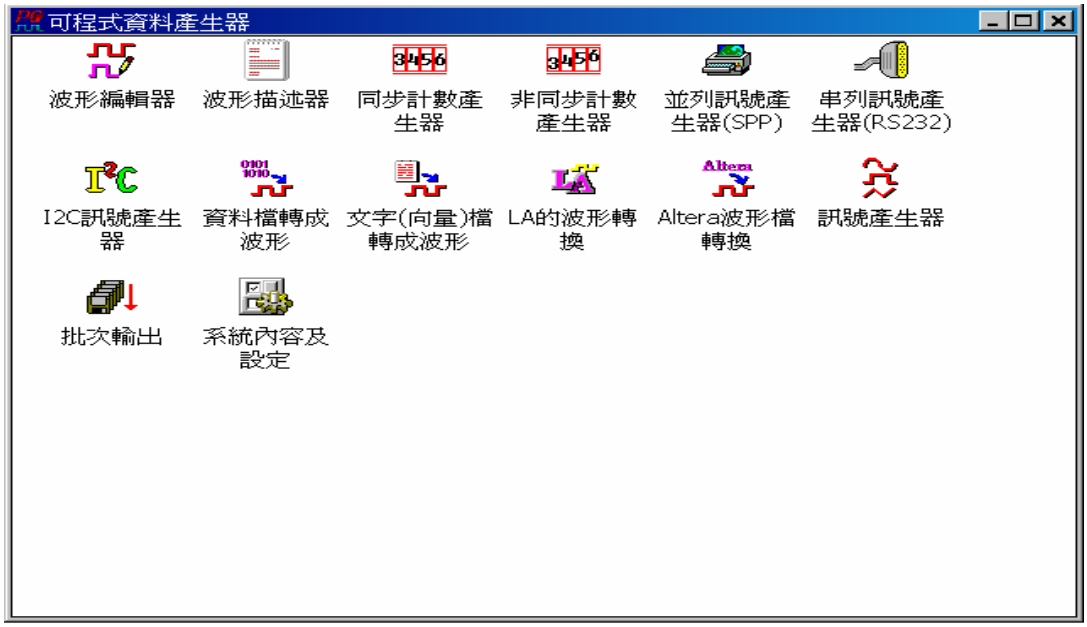
是用來定義波形資料的區域。這個區域包含了兩個部份，一個是時間部份一個是波形部份。時間部份稱為『Time Stamp』，時間部份和波形部份用『>』(大於符號)來做分隔。時間部份也可以省略，此時波形部份每一行的時間就會由 INTERVAL 或是 FREQUENCY 來指定，

所以 INTERVAL 和 FREQUENCY 只能選用一種。如果使用 Time Stamp 方式，時間部份的單位就是 UNIT 所設定的值。波形部份就是根據 INPUTS 的定義順序來描述波形。上述 PGV 的檔案中實際的資料為 10~25 共 16 筆，您可以觀察到從第 10 筆開始一直到第 25 筆結束，資料值為 0h, 1h, 2h, 3h……4Fh，上述為十六進制值，如果以十進制表示，則 0, 1, 2, 3……15。

PS:在第 25 筆的註解中的 JP 10 是表示在輸出第 25 筆的資料值之後，在跳回第十筆來重複的輸出資料。

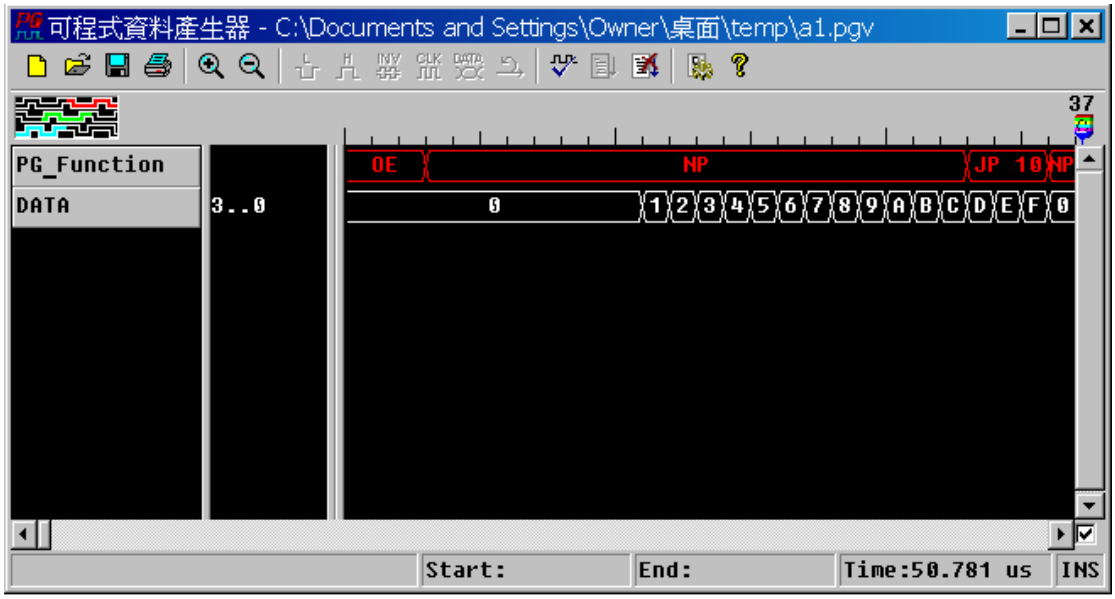
我們使用 PG EDIT 來觀察上述計數器的文字向量檔所產生的結果，

使用步驟如下:1. 執行 PG 程式



2. 執行**文字(向量)檔轉成波形**該程式  介面如下,並且選取編輯好的文字向量檔並載入該檔案

然後再按下『**波形轉換**』會出現以下畫面,該畫面為 PG EDIT 的程式介面:



上述為文字向量檔 PGV 格式中的其中一種,主要的分別為以上的 PATTERN 的波形資料部分使採用 FREQUENCY 的例子,您可以看到 PGV 檔檔頭的部分: FREQUENCY 1000 Hz; %INTERVAL 1ms;%

而另一種格式為上述 PATTERN 說明中關於 Time Stamp 的部分,底下為使用 Time Stamp 的例子:

```
INPUTS PG_Function DATA;
ASSIGN DATA 3..0;
RADIX AUTO;
UNIT   ms;
PATTERN
0.0> 8FFh      0h
1.0> 2FFh      0h
2.0> 900h      0h
10.0>000h      0h
11.0>000h      1h
12.0>000h      2h
13.0>000h      3h
14.0>000h      4h
15.0>000h      5h
16.0>000h      6h
17.0>000h      7h
18.0>000h      8h
19.0>000h      9h
20.0>000h      Ah
21.0>000h      Bh
22.0>000h      Ch
23.0>816h      Dh
24.0>200h      Eh
25.0>100h      Fh
26.0>000h      0h
;
```

以上為 Time stamp 格式的簡單範例,您可以看到與 FREQUENCY 的不同之處在於檔頭與 PATTERN 內容開頭的部分:

UNIT　ms

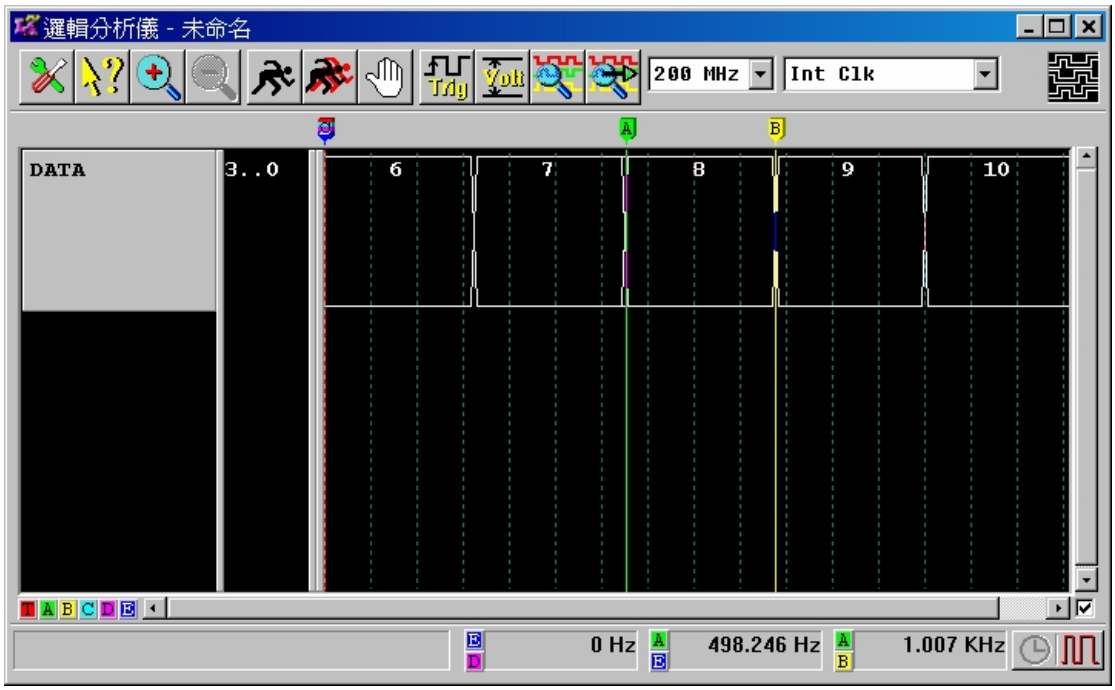
1.0>　1h

如上所說明的,UNIT ms 是指定資料的時間單位 ms(毫秒),而 1.0> 1h 是表示在 1.0ms 這個時間點有 PG_Function 指令的改變或是資料的改變,在 Time Stamp 的例子中都是記錄狀態有改變的時間點。『>』是區隔時間部分以及波形資料部分。而關於每筆資料採用的時間單位為 1.0> 主要是要配合所設定的計數器頻率 1KHz,您可以注意到只要是 PG_Function 的指令或是 DATA 改變,Time Stamp 才會加 1 ms,例如說時間 3.0>~9.0> ms 之間並沒有任何的 PG_Function 或是 DATA 改變,所以沒有記錄在檔案內,您可以跟使用 FREQUENCY 的 PGV 檔案內容做比較。

註 1:8FFh、2FFh、900h、816h、200h、100h 為 PG_Function 指令,該指令的功能都有在該行末的註解說明,關於 PG_Function 指令的完整說明請參考底下註 2

以下我們藉由 Acute LA(Logic Analyzer)來觀察結果,您可以看到起始值為 0、結束值為 15、增加值為 1、頻率為 1KHz 的計數器的簡單

的應用：



註 2:關於 PG_Function 指令的說明:

表一

指令	指令全名	說　　明	週期
NP	No Operation	沒有動作	1
JP	Jump	跳躍到設定點繼續執行	3
LP	Loop	將迴圈數減一，如果迴圈數不爲 0 就跳躍到設定點，如果爲 0 就跳到下一個位置。	3
BE	Branch if Event	如果以設定的 Event 訊號出現就跳躍到設定點，否則就會跳躍到下一個位置。	3
LC	Loop Count	設定迴圈數，設定值爲 2～65536。	2
SE	Set Event	選擇觸發的事件。	1
WE	Wait Event	等待事件被啓動。	1

PG 內部有下列幾個暫存器: RT, REX, RC, ROE。使用時都是透過 PG_FUNCTION 的指令控制。而 PG_FUNCTION 是一個 12Bits 的指令集。

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
8	XX	MOV RL,XX	將 PG_Function 的 LSB 8Bits 值移入 RL 暫存器中。
2	XX	MOV RH,XX	將 PG_Function 的 LSB 8Bits 值移入 RH 暫存器中。
1	XX	JP RT	跳至 RT-12 的新位址。

RT 有 16 個 Bits，可拆成兩個 8 Bits 的暫存器來用分別是 RL 及 RH。RT 的用途主要是用來傳遞資料用的。填入 RT 值的方法是藉由對 RL 以及 RH 這兩個 8Bits 的暫存器填值,例如:

```
MOV     RL      16h             //對暫存器 RL 填入 16h
MOV     RH      00h             //對暫存器 RH 填入 0h
會等同於:
MOV     RT      016h
```

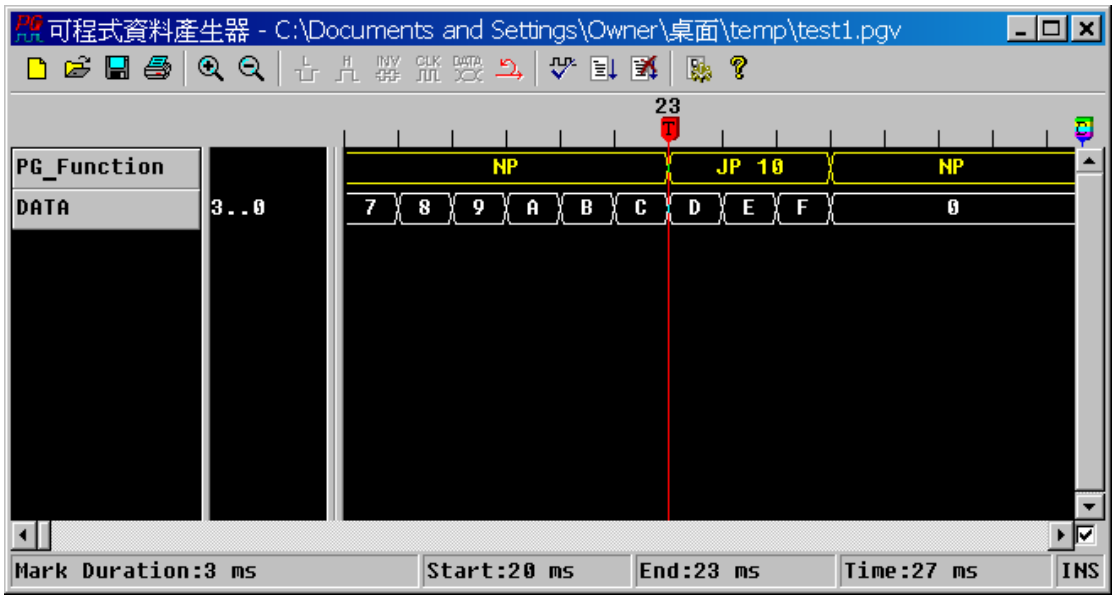
PG_Function 的指令主要是告知 PG 內部的暫存器指標做什麼對應的動作,然後由記憶體取出資料傳送出去。以下就 PG_Function 內部的指令說明如下:

NP(No Operation):在波形執行時，並不會有任何控制流程的改變或是任何內部暫存器改變。用 CPU 的觀點來看，那 NP 指令的動作就是 New Address = Address + 1，也就是指標移到下一個位置，其他不做任何改變。

JP(Jump):是一個改變控制流程的指令。例如『JP 35』，這個指令的動作就是無條件跳至新位址 35(相當於 New Address=35)。而在文字向量檔裡(*.PGV)該如何使用這個指令,以下皆使用前述 1KHz 的計數器的 PGV 檔案內容做說明:

```
000h Bh    // 00021:
000h Ch    // 00022:
816h Dh    // 00023:          (MOV RL, 22)
200h Eh    // 00024:          (MOV RH, 0)
100h Fh    // 00025:          JP 10
000h 0h    // 00026:
;
```

以上三行紅色字體的十六進制的 PG_Function 的指令, 816h:對暫存器 RL 填入數值 22(MOV RL, 22),200h: 對暫存器 RH 填入數值 0 (MOV RH, 0), 100h: 跳至 RT-12 的新位址(22 - 12 = 10),也就是說送出第 25 點資料後,跳躍到第十點後再繼續傳送資料。以下是透過 PG EDIT  來觀察文字向量檔(*.PGV)內的資料與 PG_Function 的指令:



LP(Loop): LP 和 JP 指令很類似，其最大差異就是 JP 是一個無條件跳躍指令，而 LP 是一個有條件跳躍，它必須與『LC』指令配合使用。PG 有一個內部暫存器稱為迴圈暫存器(Counter)，這個暫存器就是用來記錄迴圈數目的暫存器。這個暫存器是用 LC(Loop Count)指令來設定，例如『LC 32』就是將迴圈暫存器設定成 32。迴圈暫存器可以設定的值為 2～65536，所以不能設定成 0 與 1，這種設定方式與一般

的 CPU 或 MCU 的用法有些許的差異，需注意。設定好迴圈以後就可以使用 LP 指令了，而每次執行 LP 指令，迴圈暫存器的值就會減 1，而跳躍的新指標就會參考 LP 指令的設定值來決定。

例如指令『LC 32』、『LP 16』，的整個動作步驟如下：

一、　將迴圈暫存器 32 減 1。

二、　檢查迴圈暫存器是否為 0。

三、　如果迴圈暫存器為 0，New Address = Next Address。

四、　如果迴圈不為 0，New Address = 16。

```
200h 0h    // 00007: (MOV RH, 0)
401h 0h    // 00008: LC 3
000h 0h    // 00009:
000h 0h    // 00010:

==========================


820h Dh    // 00033: (MOV RL, 32)
200h Eh    // 00034: (MOV RH, 0)
300h Fh    // 00035: LP 20
```

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
3	XX	LP RT	跳至 RT-12 的新位址。

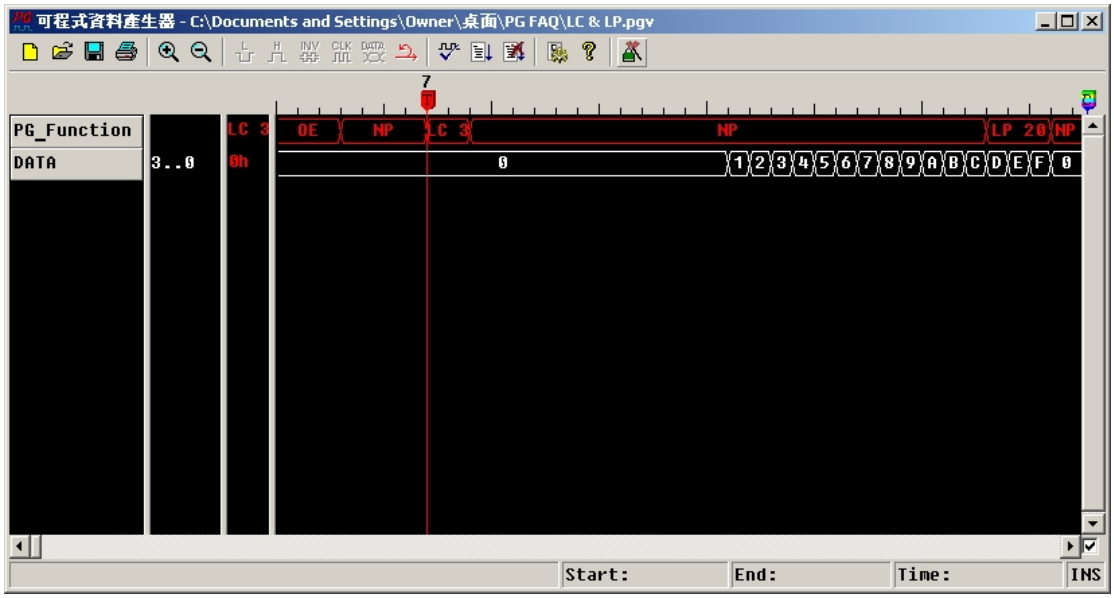
先看到前三行紅色字體的十六進制的 PG_Function 的指令, 820h: 對暫存器 RL 填入數值 32(MOV RL, 32), 200h: 對暫存器 RH 填入數值 0 (MOV RH, 0), 300h: 跳至 RT-12 的新位址(32 - 12 = 20)做迴圈的動作。

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
4	XX	LC RC	設定 RC 暫存器的迴圈數

接下來看到後兩行的部分: 200h: 對暫存器 RH 填入數值 0(MOV RH, 0), 401h:對迴圈暫存器 RC 填入數值 3(1 + 2 = 3, 因為迴圈數從 2 開始), 也就是說暫存器 RC 裡的值為 1, 因為 RC 也是 16Bits 的暫存器, 所以這兩行所代表的意義為 :

RC	00000000 (00h)	00000001(01h)

以下為透過 PG EDIT  來觀察文字向量檔(*.PGV)內的資料與 PG_Function 的指令:



有一點要注意的是如果當迴圈暫存器已經被減成 0，又遇上 LP 指令時，這時候迴圈暫存器是不會再被減成-1，而是會產生特殊情況。所以迴圈暫存器被減成 0 以後，LP 指令的工作是不可預期的，所以要使用 LP 指令之前請注意是否已經設定 LC 指令了。

SE(Set Event):是一個選擇事件的指令，PG 有 3 個外部事件通道(Event_1、Event_2、Event_3)及 1 個內部鍵盤事件(Keyboard Event)，所以總共有 4 個事件來源。PG 將這 4 個事件來源編輯成 16 種事件型態以茲利用，而這個事件型態會被存到一個 PG 的內部暫存器，稱之為事件暫存器(Event Register)。這些事件型態包括:

1. Keyboard Event
2. Event_1
3. Event_2
4. Event_3

5. Event_1 or Event_2

6. Event_1 or Event_3

7. Event_2 or Event_3

8. Event_1 or Event_2 or Event_3

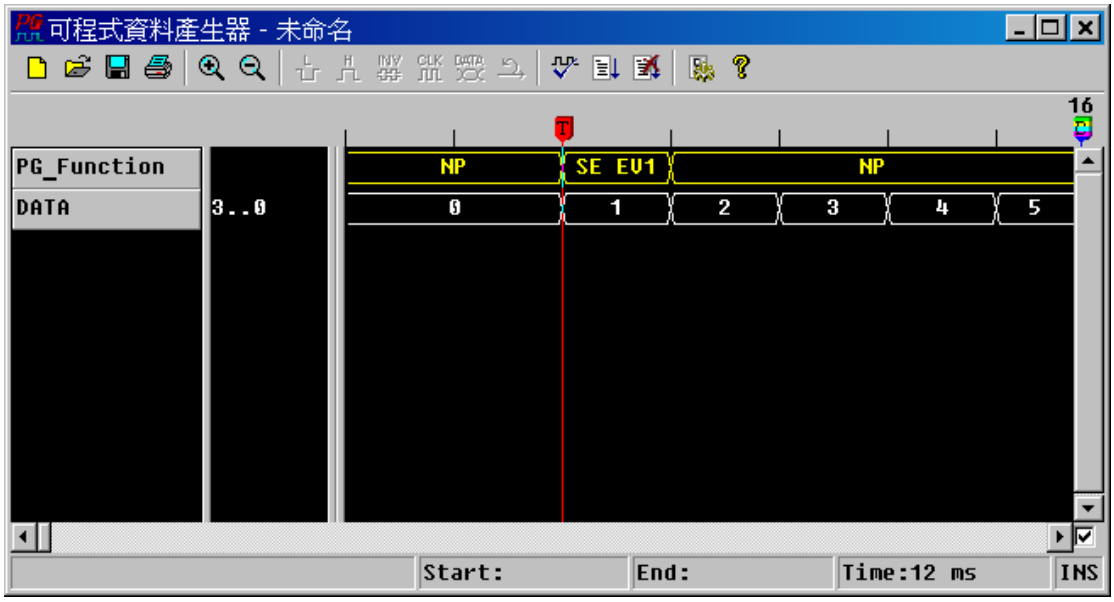
另外 8 種事件型態就是這 8 種型態的反相。例如事件暫存器的設定值為上述 8 種型態之一，此時不管從外部或是內部的事件通道都會被 PG 所監控，PG 會隨時比較目前的事件通道的狀態是否與事件暫存器的事件型態相同。如果相同就會將 PG 內部的旗標暫存器(Flag Register)的 Event Bit 設成 1(True)，如果不同就會將 Event Bit 設成 0(False)。然而反相型態就是當事件通道的狀態與事件暫存器的事件型態相同時，PG 會將 Event Bit 設成 0，不同時則會將 Event Bit 設成 1。

```
000h 0h    // 00010:
609h 1h    // 00011:                              SE EV1
000h 2h    // 00012:
```

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
6	XX	SE EV	設定 REX 暫存器事件值

8Bits 的 LSB 中:00h 代表內部鍵盤事件(Keyboard Event)、01h:代表外部事件通道 1 的反相型態(!EV1)、02h: 代表外部事件通道 2 的反相(!EV2)型態、03h: 代表外部事件通道 1 的反相型態與外部事件通道 2 反相型態的交集(!EV1 & !EV2) 、04h:代表外部事件通道 3 的反相型態(!EV3)、05h:代表外部事件通道 1 的反相與外部事件通道 3 的反相的交集(!EV1 & !EV3)、 06h:代表外部事件通道 2 的反相與外部事件通道 3 的反相交集(!EV2 & !EV3) 、07h:代表設定外部事件通道 1 的反相與外部事件通道 2 的反相與外部事件通道 3 的反相的交集(!EV1 & !EV2 & !EV3)、08h:代表內部鍵盤事件的反相(!KEY)、09h:代表外部事件 1(EV1)、0Ah:代表外部事件 2(EV2)、0Bh:表示 Event_1 or Event_2,也就是!(!EV1 & !EV2)……其他的依此類推,總共會有 16 種不同的事件。

您只要在 8Bits 的 LSB 中輸入您想要的情況即可。以下為上述例子之圖示:

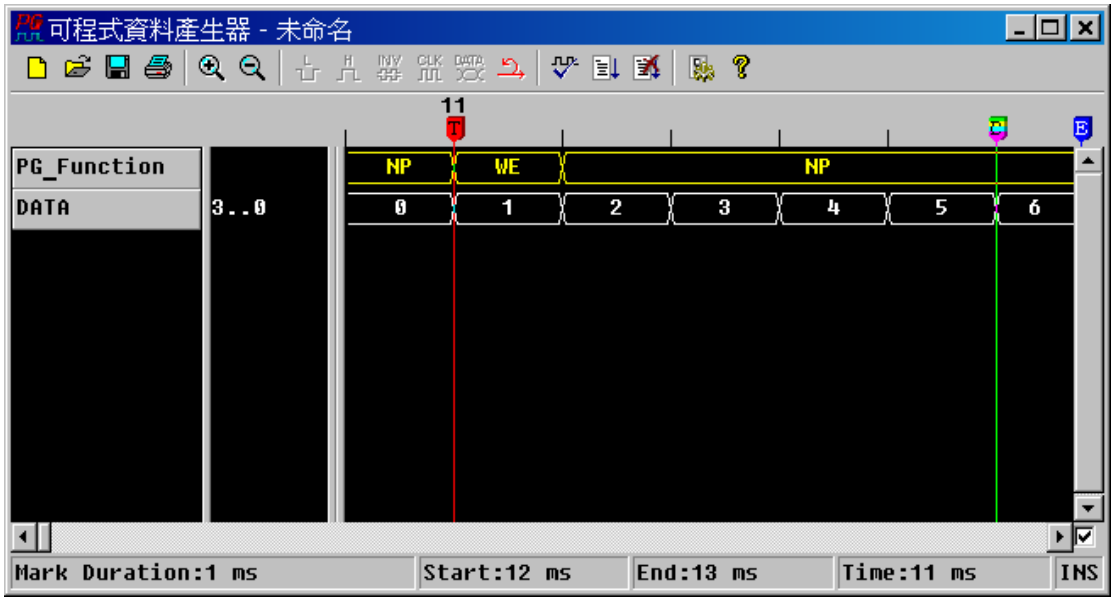


而運用 Event Bit 的指令有兩個，一是 WE(Wait Event)另一是 BE(Branch If Event)。

WE(Wait Event):指令是當波形執行到這個指令時，整個 PG 會因此而暫停，而波形會停留在 WE 指令當時的波形，一直到 Event Bit 為 1 時，才會再繼續往下執行。

```
000h 0h    // 00010:
700h 1h    // 00011:                WE
000h 2h    // 00012:
```

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
7	XX	WE	PG 暫停,等待事件發生



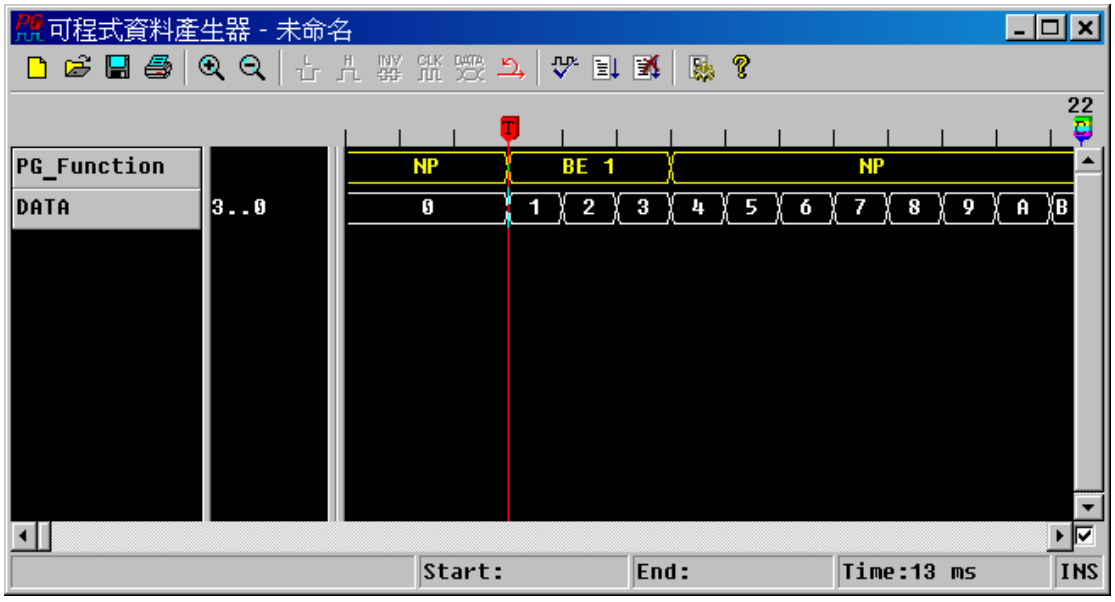
BE(Branch If Event):指令和 LP 指令又有點類似，因為他們都是一個有條件跳躍的指令。LP 的跳躍條件是迴圈暫存器，而 BE 的跳躍條件是 Event Bit。當波形執行到 BE 指令時，PG 會立即判斷 Event Bit，如果 Event Bit 為 1 就會跳到 BE 所設定的新位址，Event Bit 為 0 的話，則繼續執行下一個位址。

```
000h 0h   // 00010:
80Dh 1h   // 00011:                (MOV RL, 13)
200h 2h   // 00012:                (MOV RH, 0 )
500h 3h   // 00013:                BE 13
000h 4h   // 00014:
```

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
5	XX	BE	跳至 REX 的新位址。

而 80Dh: 對暫存器 RL 填入數值 13(MOV RL, 13), 200h: 對暫存器 RH 填入數值 0(MOV RH, 0),500h: 跳至 REX 的新位址。

以下為上述例子之圖示:



在 PKPG 系列中,PG_Function 的指令還有一個 Output Enable 『OE』,說明如下:

```
8FFh 0h   // 00000:          (MOV RL, 255)
2FFh 0h   // 00001:          (MOV RH, 255)
900h 0h   // 00002:          OE   65535
000h 0h   // 00003:
```

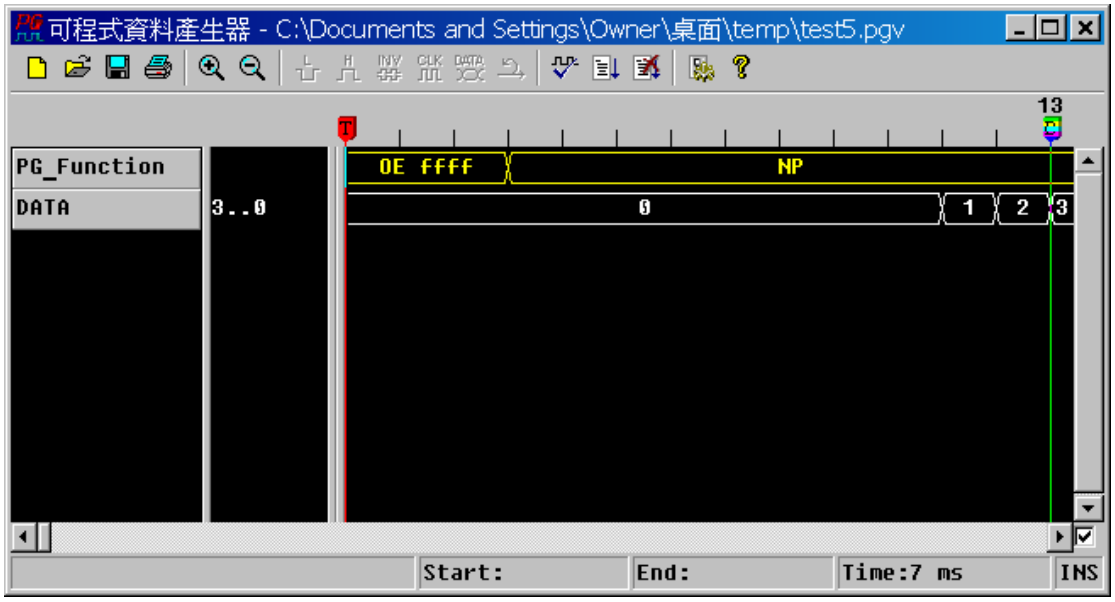
PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
9	XX	OE	將通道輸出致能

8FFh: 對暫存器 RL 填入數值 255(MOV RL, 255),2FFh:對暫存器 RH 填入數值 255(MOV RH, 255), 900h: 輸出致能。

暫存器 ROE 就如同暫存器 RT 一樣,都是由 8Bits 的 RL 與 RH 組成,所以整個 ROE 暫存器也可以看成如下所示:

暫存器 ROE	11111111(FFh)	11111111(FFh)

底下為圖示:


可程式資料產生器 - C:\Documents and Settings\Owner\桌面\temp\test5.pgv
13
T
PG_Function
DATA
3..0
OE FFFF
NP
0
1
2
3
Start:
End:
Time:7 ms
INS