How to use text editor software to edit PG Vector File (*.PGV)

Programmable Data Generator (PG in brief), it can read not only PG Waveform File (*PGW) but also PG Vector File (*PGV). You can use any text editor software to edit the PG Vector File by yourself and the contest of PG Vector File is the data and PG command. We'll explain the format of PG Vector File and 7 PG commands include NP (No Operation), JP (Jump), LP (Loop), BE (Branch if Event), LC (Loop Count), SE (Set Event) and WE (Set Event).

```
INPUTS PG_Function DATA;
ASSIGN DATA 3..0;
RADIX AUTO;
FREQUENCY 1000 Hz;
%INTERVAL 1ms;%                    『%..%』:comment
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              『//』:Comment
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
;
```

We'll explain the PG Vector File sample that it is a 4-bits-width, 1 KHz synchronous counter as above.

INPUT PG_Function DATA

Decide the signal name. Every signal name separates by a space and if the signal is the bus signal (Group), you can use sign [] to express, for example, A [3..0] means A3, A2, A1, A0, 4 signals.

Note:
"PG_Function" is a keyword; don't use "PG_Function" as your signal name. It means you'll use PG_Function command in your pattern here.

ASSIGN DATA 3..0

It indicates that which channel your signal output. It means DATA0 = CH-00, DATA1 = CH-01, DATA2 = CH-02, DATA3 = CH-03.

RADIX AUTO

Set the bus group radix. If the value in PATTERN section follows with radix-ID (h, d, o, b), the RADIX should be AUTO.

Ex. When the RADIX is AUTO, the pattern 35 (=35d) and 35h (=53d) are different: Set the RADIX to HEX, the pattern 35 and 35h are equal. When RADIX sets to DEC, the pattern 35h will treat as 35d.
The 5 kinds of RADIX as:

- AUTO: depending on radix-ID
- HEX: Hexadecimal
- DEC: Decimal
- OCT: Octal
- BIN: Binary

In AUTO mode, the value with radix-ID in PATTERN section: "h" is hexadecimal value, "o" is octal value, and "b" is binary value. The empty radix-ID value will treat as decimal value.

FREQUENCY 1000 Hz

It means PG clock frequency is 1000 Hz.

PATTERN

The section-keyword is the head of waveform pattern. There are two areas in the section: time scale (called Time Stamp) and wave data, using ">" to separate the two areas. In No Time Stamp mode, Time Stamp can be removed. The time scale is increased INTERVAL (or FREQUENCY) column by column. Only one section-keyword of INTERVAL and FREQUENCY can be chose in No Time Stamp mode. In Time Stamp mode, the time scale accord with Time Stamp, time unit accord with UNIT value, and these wave data describe what these INPUTS digital patterns are. See Time Stamp example as below:

```
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
;
```

There is a main difference between No Time Stamp example and Time Stamp example. In No Time Stamp example, FREQUENCY 1000 Hz or INTERVAL 1ms means that every interval of the data sample point is 1 KHz or 1ms. In Time Stamp example, UNIT ms means the unit of every Time Stamp.

```
000h      0h   // 10          START PATTERN
000h      1h   // 11
000h      2h   // 12
000h      3h   // 13
```

There is an extract from No Time Stamp example; it indicates that the tenth data sample point is 0h (Hex); the eleventh data sample point is 1h (Hex); the twelfth data sample point is 2h (Hex) and the thirteenth data sample point is 3h (Hex). Every interval of the data sample point is 1 KHz or 1ms.

```
0.0> 8FFh      0h
1.0> 2FFh      0h
2.0> 900h      0h
10.0>000h      0h
11.0>000h      1h
12.0>000h      2h
```

Here is another extract from Time Stamp example; it means that the data is 0h (Hex) when 0 ms, 1ms, 2~10 ms; the data is 1h (Hex) when 11 ms; the data is 2h (Hex) when 12 ms.

Note: 8FFh, 2FFh, 900h, 816h, 200h, 100h is PG_Function command. We will explain them later.

PG_Function:

Name	Instruction	Description	Clk*
NP	No Operation	No action	1
JP	Jump	Jump to a new address	3
LP	Loop	Reduce 1 of the LC value. Jump to a new address if LC >0; Go to next address if LC =0	3
BE	Branch if Event	Jump to a new address if receive SE. Else go to next address	3
LC	Loop Count	Set Loop Count (2～65536)	3
SE	Set Event	Set Event to be a trigger	1
WE	Wait Event	Stop for waiting Event received	1

*Clk: It is a machine cycle, reference to the Base Frequency.

There are several internal registers in the PG: RT, REX, RC, and ROE. They are controlled by PG_Function command. PG_Function is 12-Bits command set.

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
8	XX	MOV RL,XX	Move the LSB of the PG_Function into RL.
2	XX	MOV RH,XX	Move the LSB of the PG_Function into RH.
1	XX	JP RT	Jump new address to RT-12.

RT is a 16-bits-width register; it can be separated two 8-bits-width register RL and RH.

```
MOV     RL      16h     //Insert 16h in the RL
MOV     RH      00h     //Insert 0h in the RH
Is equal to
MOV     RT      016h
```

PG_Function commands told the internal register pointer of the PG to work according to the command you give. Show the detail of the PG_Function as below:

NP (No Operation):

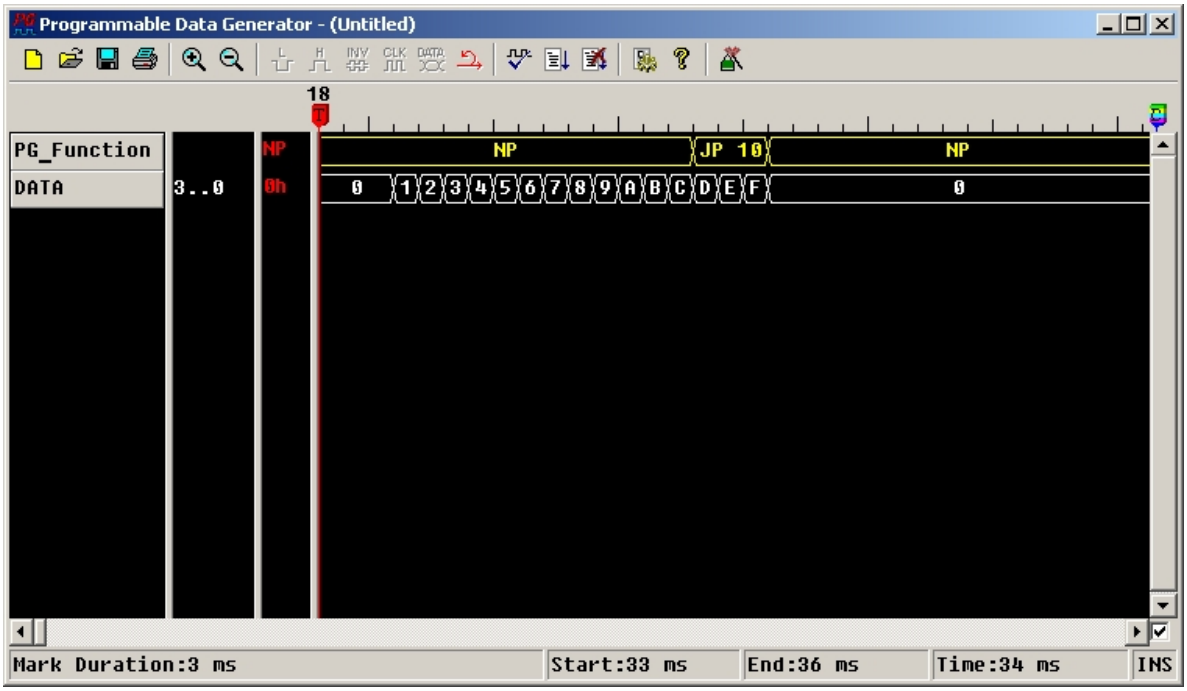
NP (No Operation) will affect nothing. The action is the same as MCU and CPU, NP means New Address = Address + 1.

JP (Jump):

JP (Jump) will affect the output flow. Ex. JP 35 means to jump a new address=35 without any condition.

```
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:
;
```

Note:
816h means that insert the value 22 (16h) into the RL register.
200h means that insert the value 0 into the RH register.
100h means that jump the new address RT-12 (22-12 = 10).



LP (Loop):

LP (Loop) is similar with JP. The different is that JP requires no condition but LP is a condition-jump decided by LC. There is a register in Acute PG called LC (Loop Counter). To set LC 32 will write 32 into Loop Counter. The LC legal value is 2～65536. It is illegal value about 0 and 1. (Note: Here is the different with most CPU and MCU.) Now, we can use the LP command after setting the LC value. The waveform output flow run across the LP command will reduce 1 of the LC.

Ex. Set LC 32 in address=3~4, set LP 16 in address=23~25

1. Run along address to LP 16, and then reduce 1 of the LC (LC=LC-1).
2. Check the LC at address=25
3. If LC =0, New Address = Next Address = 26
4. If LC >0, New Address = 16

Note: If the LC=0 already, and run across the LP, reduce the LC will cause unrespectable flow.

```
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
```

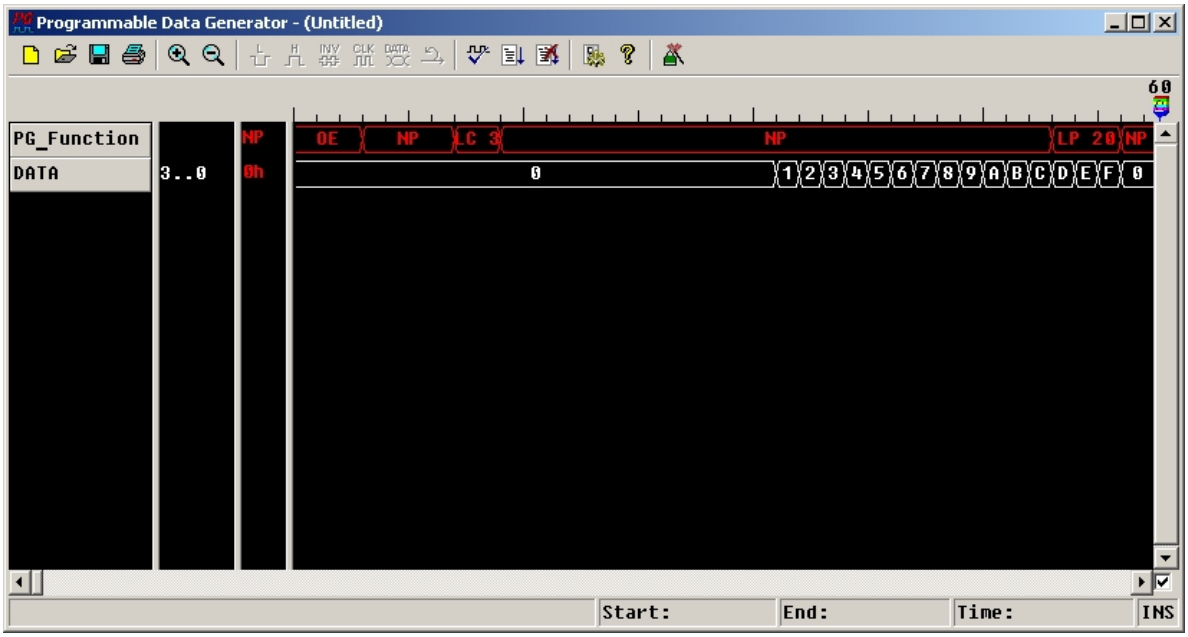
PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
3	XX	LP RT	Jump to new address RT-12.

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
4	XX	LC RC	Loop count of the RC.

Note:
200h means that insert 0 into the RH register.
401h means that insert 3 (1+2=3, loop count range: 2 ~65536) into the

RC register (16-bits-width).

RC	00000000 (00h)	00000001(1h)



SE (Set Event):

There are 4 events of PG, included 3 external events (Event_1, Event_2, Event_3) and 1 internal event (Keyboard Event). The PG interlaces the 4 events to be 16 conditions for controlling the output flow. These 16 conditions will be saved into the Event register of PG.

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

Note: PKPG series possess 2 external events (Event_1, Event_2) and 1 internal event (Keyboard Event).

The others 8 conditions are the inverse of these 8 items.

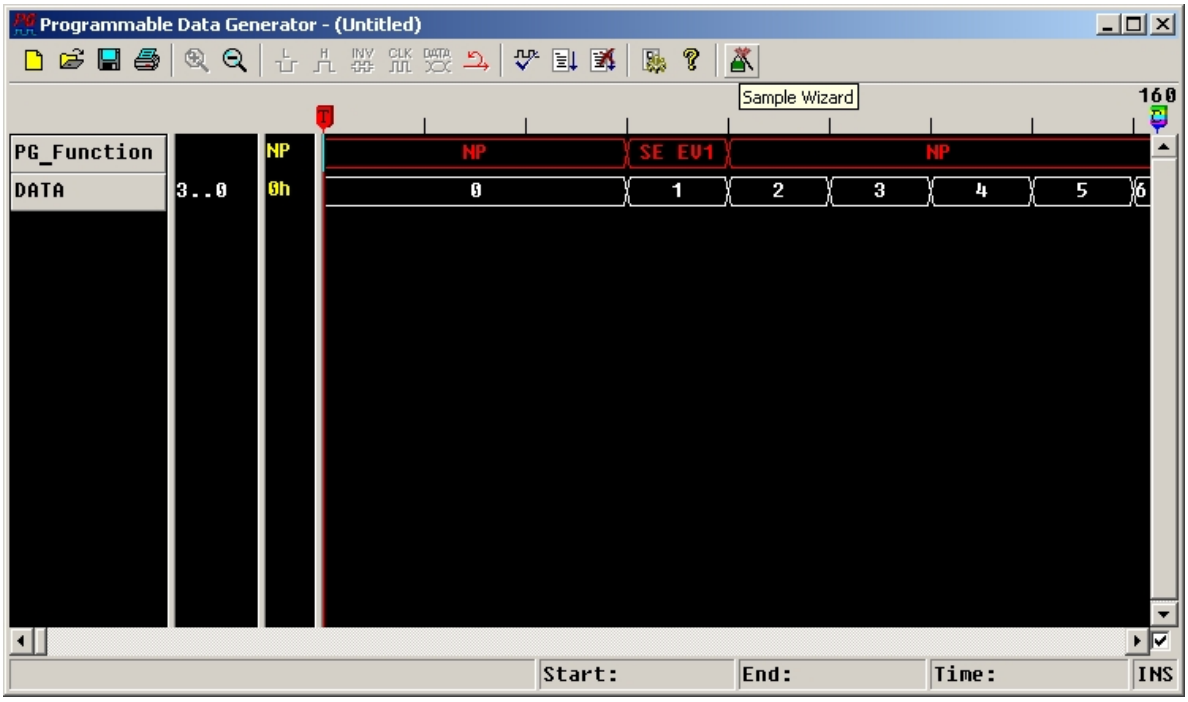
If Event registers set as above 8 conditions, PG will detect these event-channels and compare with Event register. To get the same value will set the Flag-Register-Event bit of PG to be true state. If got the different value, then set the bit to be false state. Nevertheless, invert conditions will detect these event-channels and compare with Event register. To get the same value will set the Flag-Register-Event bit to be false state; Got the different value will set the Event bit to be true state.

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

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
6	XX	SE EV	Insert event into the REX.

Note:
600h means that Set Keyboard Event.
601h means that Set Not Event_1.
602h means that Set Not Event_2.
603h means that Set Not Event_1 And Not Event_2.
604h means that Set Not Event_3.
605h means that Set Not Event_1 And Not Event_3.
606h means that Set Not Event_2 And Not Event_3.
607h means that Set Not Event_1 And Not Event_2 And Not Event_3.
608h means that Set Not Keyboard event.
609h means that Set Event_1.
60Ah means that Set Event_2.
60Bh means that Set Event_1 Or Event_2.
60Ch means that Set Event_3.
60Dh means that Set Event_1 Or Event_3.
60Eh means that Set Event_2 Or Event_3.
60Fh means that Set Event_1 Or Event_2 Or Event_3.

There are two-command sets actions depending on the Event bit: one is WE (Wait Event), the other one is BE (Branch If Event).

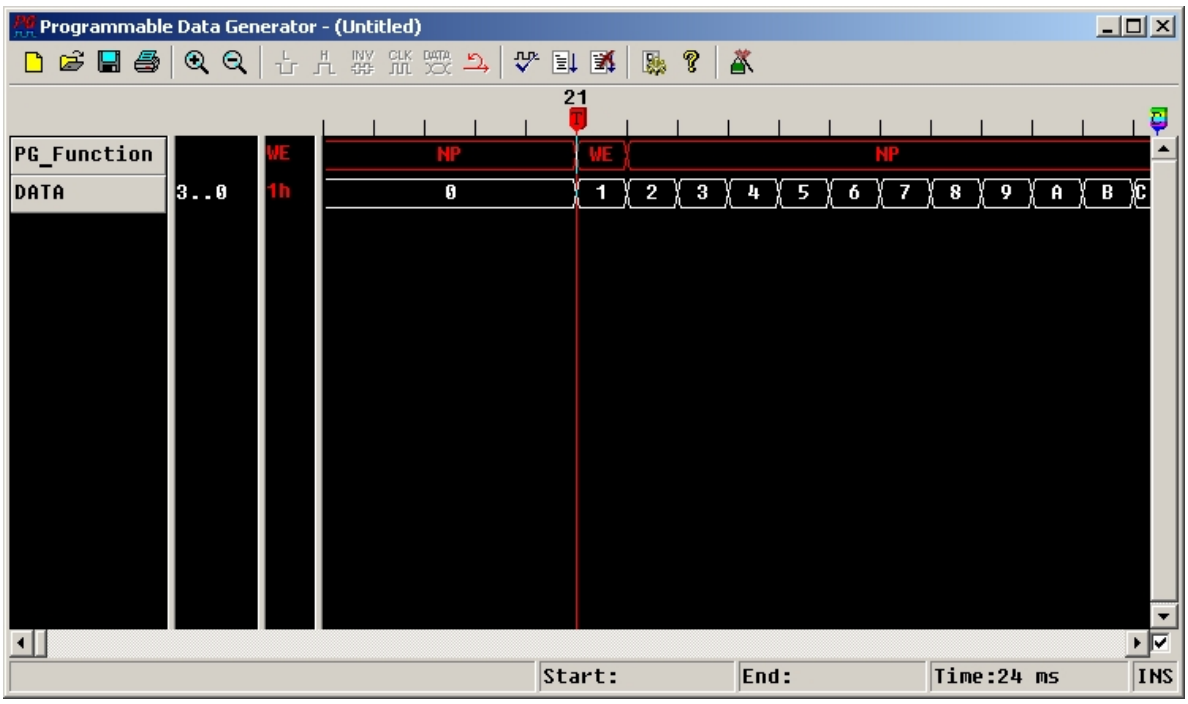


WE (Wait Event):

The WE (Wait Event) command will stop the PG flow at the address and do not go to the next address until Event bit =1.

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

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
7	XX	WE	PG paused and Wait event.



BE (Branch If Event):

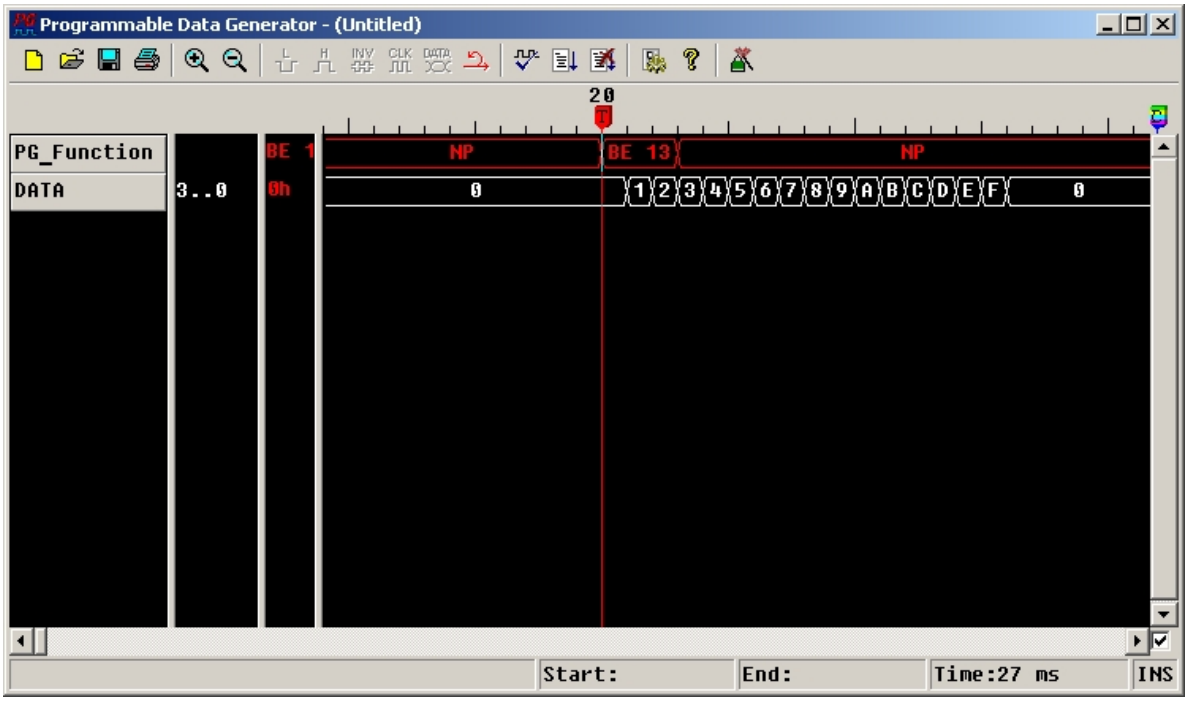
The BE (Branch If Event) command is similar with LP. Because they are both condition-jump. LP jumps by LC condition, BE jumps by Event bit state. When PG flow run across BE command, the PG will jump to BE address if the Event bit =1. It will go to next address when the Event bit =0.

```
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	Jump to the new address of the REX

Note:
80Dh means that insert 13 (0Dh) into the RL register.
200h means that insert 0 into the RH register.
500h means that jump to the new address in the REX register.

REX | 00000000 (00h) | 00001101(0Dh) |



Note:
There is a PG_Function command OE (Output Enable), it's only used in PKPG series.

```
8FFh 0h    // 00000:          (MOV RL, 255)
2FFh 0h    // 00001:          (MOV RH, 255)
900h 0h    // 00002:          OE   65535
000h 0h    // 00003:
000h 0h    // 00004:
```

PG_Function (12Bits)			
4Bits(MSB)	8Bits(LSB)		
9	XX	OE	Output Enable

8FFh means that insert 255 (FFh) into the RL register.
2FFh means that insert 255 (FFh) into the RH register.
900h means that enable channels output.

ROE	11111111(FFh)　　11111111(FFh)