# Using PWM on the Vortex86

Application Note
AP0104 (v1) [Publish Date]

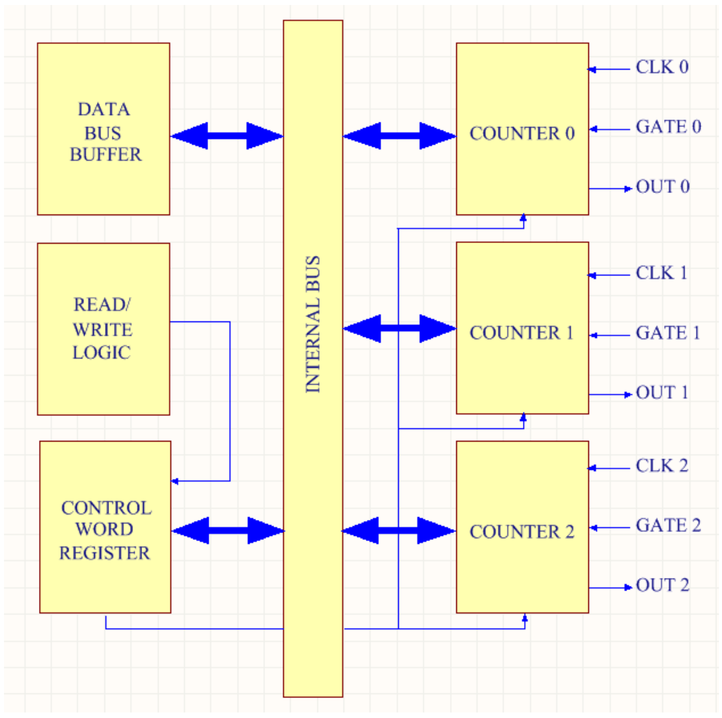
These DOS code examples were written and compiled using Borland Turbo C++.
There is a download link for this compiler at http://cc.embarcadero.com/item/26014

This document should be read in conjunction with application note AP0102 'Accessing North Bridge and South Bridge registers on the Vortex86.

The Vortex86 includes an 8254 counter which can be configured to provide a number of counting and timing applications. This application note will focus on providing a pulse width modulated output. Detailed information for the 8254 counter can be downloaded from the internet by searching for "8254 datasheet"

## 8254 Description

The 8524 consists of three programmable interval timers/counters and a control register which are accessed in IO space within the Vortex 86



The three counters are identical and fully independent 16-bit, pre-settable synchronous down counters. They are programmed by writing to the control word register and the individual count register. The four registers are mapped into IO space at the following addresses:

| IO Address | Description |
|---|---|
| 48H | Count Register 0 |
| 49H | Count Register 1 |
| 4AH | Count Register 2 |
| 4BH | Control Word Register |

Control Word Register

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| SC1 | SC0 | RW1 | RW0 | M2 | M1 | M0 | BCD |

SC – Select counter

| SC1 | SC0 | Description |
|---|---|---|
| 0 | 0 | Select counter 0 |
| 0 | 1 | Select counter 1 |
| 1 | 0 | Select counter 2 |
| 1 | 1 | Read back command[1] |

RW – Read/Write mode

| RW1 | RW0 | Description |
|---|---|---|
| 0 | 0 | Counter latch command[1] |
| 0 | 1 | Read/write least significant byte only |
| 1 | 0 | Read/write most significant byte only |
| 1 | 1 | Read/write least significant byte first then most significant byte |

M-Mode

| M2 | M1 | M0 | Description |
|---|---|---|---|
| 0 | 0 | 0 | Mode 0 |
| 0 | 0 | 1 | Mode 1 |
| 0 | 1 | 0 | Mode 2 |
| 0 | 1 | 1 | Mode 3 |
| 1 | 0 | 0 | Mode 4 |
| 1 | 0 | 1 | Mode 5 |

BCD

| | |
|---|---|
| 0 | Binary counter 16-bits |
| 1 | Binary coded decimal counter (4 decades) |

[1] Feature not described in this document. For further information download the 8254 datasheet.

**Mode description**

This application note will give a short description of the different modes – for a full description download the 8254 datasheet. Throughout this document.

- CLK refers to a rising, then a falling edge of a counter CLK input.
- TRIGGER refers to a rising edge on a counter GATE input.
- OUT is the output pin if the counter.

Mode 0 - Interrupt on terminal count

After the control word is written, OUT is low and will remain low until the counter reaches zero. OUT will then go high and remain high until either a new count or a new mode 0 control word is written.
If GATE is high, counting is enabled; if gate is low counting is disabled. The GATE input has no direct effect on OUT.

Mode 1 – Hardware retriggerable one-shot

After the control word is written, OUT will be high and will go low on the CLK following a TRIGGER; this is the start of the one shot pulse. OUT will remain low until the count reaches zero after which it will go high until the CLK after the next TRIGGER.

Mode 2 – Rate Generator

This mode functions as a divide-by-N counter.
After the control word is written, OUT will be high, when an initial count has decremented to 1, OUT will go low for one CLK and then will go high again. The counter then reloads the initial count and the sequence is repeated. For an initial count of N, the sequence repeats every N clock cycles.
If GATE is high, counting is enabled; if gate is low counting is disabled. Gate can be used to synchronize the counter as a TRIGGER reloads the counter with the initial count.

Mode 3 – Square wave mode

Mode 3 is similar in operation to mode 2 except for the duty cycle. After the control word is written, OUT will be high, when the count reaches half the initial count, OUT will go low until the count reaches zero, after which it will go high again. The counter then reloads the initial count and the sequence is repeated.
If GATE is high, counting is enabled; if gate is low counting is disabled. If the GATE goes low whilst OUT is low, OUT will immediately go high.
Gate can be used to synchronize the counter as a TRIGGER reloads the counter with the initial count.

Mode 4 – Software triggered strobe

After the control word is written, OUT will be high, when an initial count has decremented to 1, OUT will go low for one CLK and then will go high again. The sequence is retriggered by writing the initial count.

If GATE is high, counting is enabled; if gate is low counting is disabled. The GATE input has no direct effect on OUT.

Mode 5 – Hardware triggered strobe (retriggerable)

After the control word is written, OUT will be high,

Counting is triggered by a rising edge on GATE. When the counter reaches zero, OUT will go low for one CLK and then go high again.

TRIGGER causes the counter to be loaded with the initial count at any time (retriggerable)

Summary of GATE pin

| Mode | Low / falling | Rising | High |
|---|---|---|---|
| 0 | Disable Counter | - | Enable Counter |
| 1 | - | 1. Start Count<br>2. Reset output after next clock | - |
| 2 | 1. Disable Counter<br>2. Set OUT high | Start Counter | Enable Counter |
| 3 | 1. Disable Counter<br>2. Set OUT high | Start Counter | Enable Counter |
| 4 | Disable Counter | - | Enable Counter |
| 5 | - | Start Counter | - |

## Enabling the PWM pins

The Vortex86 multiplexes the PWM pins with COM2. By default the pins function as COM2 so the first task is to enable the PWM by programming the 'Internal Peripheral Control Register' in the south bridge.

| | |
|---|---|
| **Register Offset:** | C3-C0h |
| **Register Name:** | Internal Peripheral Feature Control Register |
| **Reset Value:** | 032C0500h |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rsvd | | IO16W | | | IO8W | | | MEM16W | | | MEM8W | | | ISACLK | Rsvd | | IDEIS | PWM2 | PWM1 | PWM0 | FCDC | MCLK | EMIQ | Rsvd | SFCE | Rsvd | PINS4 | CPS | PINS2 | PINS1 | PINS0 |

The relevant bits for PLL are shown in the following table:-

| Bit | Name | Description |
|---|---|---|
| 13 | PWM2 | PWM Timer 2<br>0: internal 1.19MHz (default)<br>1: external clock |
| 12 | PWM1 | PWM Timer 1<br>0: internal 1.19MHz (default)<br>1: external clock |
| 11 | PWM0 | PWM Timer 2<br>0: internal 1.19MHz (default)<br>1: external clock |
| 2 | PINS2 | Pin selection for COM2 and PWM<br>0: 9 pins for COM2 (default)<br>1:  9 pins for PWM |

## Example program

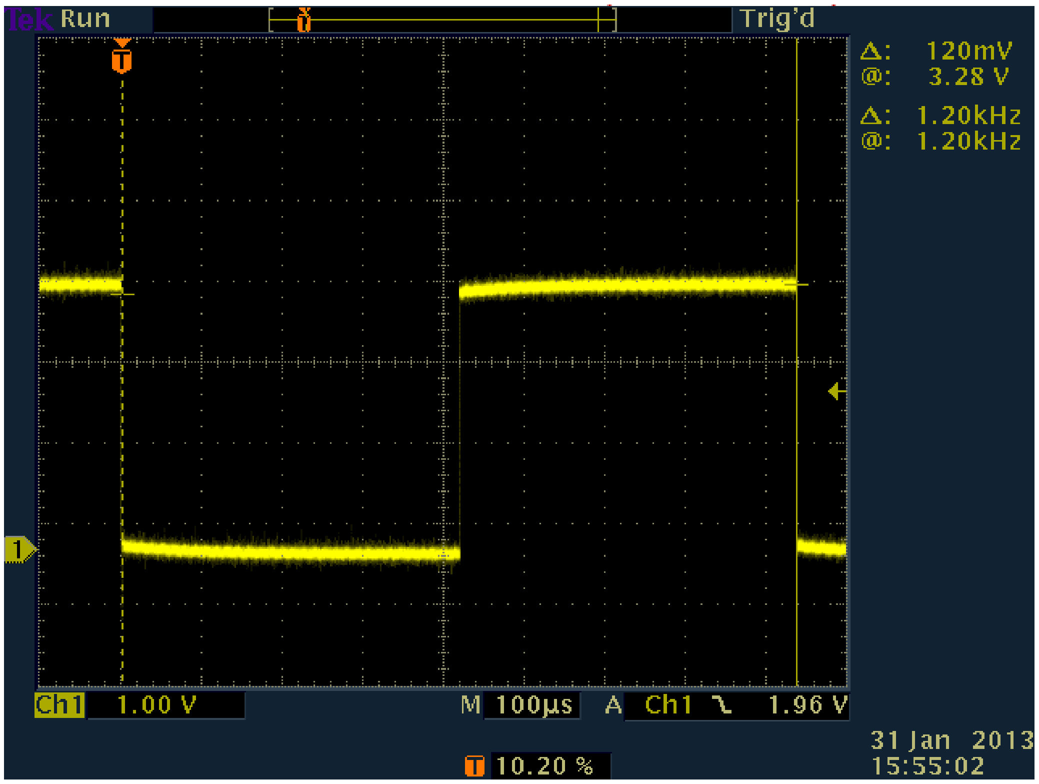
With the background information covered, the next step is to show some examples of how to program the interface. The first example will be to simply generate a square wave from PWM0.
The first thing is to set up the southbridge register to enable the interface.
These examples use inport32 and outport32 functions which are detailed in application note AP0102 'Accessing North Bridge and South Bridge registers on the Vortex86.

```
long Value;

Value=inport32(0x800038c0); // read in current register at offset C0h
Value &= 0xffffffc7ffL; // set bits 13-11 so all pll's use internal clk
Value |= 0x0000000004L; // set COM2 pins to be PWM
outport32(0x800038c0,Value); // write back to register
```

To make PWM0 generate a square wave requires the counter to work in mode 3 and setting an initial count value, say 1000.

```
cCnt=0; //counter number
cMode=3; // counter mode
lValue=1000;

cCmd=0; // clear to start
cCmd |= (cCnt << 6); // shift counter number into bits 7-6
cCmd |= (0x03 << 4); // set RW bits to 16 bit mode
cCmd |= (cMode << 1); // shift mode into bits 3-1
outp(0x4b,cCmd); // write to command register

//Value is 16 bits so sequentially send LSB followed by MSB
outp(0x48+cCnt,(unsigned char) (lValue & 0xff));
outp(0x48+cCnt,(unsigned char) (lValue >> 8));
```

Connect an oscilloscope to the COM2/PWM0_OUT pin and after running the program there should be a 1.2kHZ square wave on the pin. Notice that the square wave is generated in hardware, once the program has set up the counter no further action is required.



Full program listing for square wave generator:-

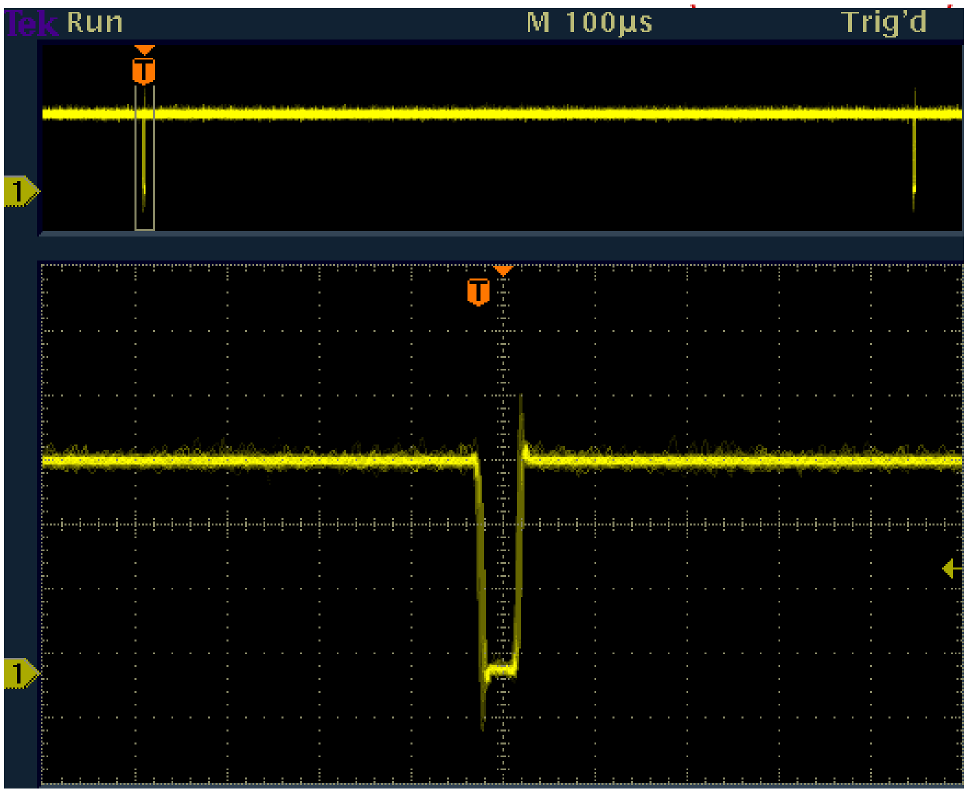
```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>

#define PCI_ADDRESS 0xcf8
#define PCI_DATA 0xcfc
#define PWM_COUNT_REG_0 0x48
#define PWM_CONTROL_REG 0x4B

// 32 bit operations
#define OPERAND32 asm db 66h

// prototypes
void outport32(unsigned long, unsigned long);
unsigned long inport32(unsigned long);

void main (void)
{
long Value;
int i;
unsigned char cCmd = 0x00;
long lValue = 0x00;
unsigned char cCnt = 0x00;
unsigned char cMode = 0x00;

// first set up the southbridge register
Value=inport32(0x800038c0);
Value &= 0xffffffc7ffL; // set bits 13-11 so all pll's use internal clk
Value |= 0x0000000004L; // set COM2 pins to be PWM
outport32(0x800038c0,Value);

// write to command register
cCnt=0; //counter number
cMode=3; // counter mode
lValue=1000;

cCmd=0; // clear to start
cCmd |= (cCnt << 6); // shift counter number to bits 7-6
cCmd |= (0x03 << 4); // set RW bits to 16 bit mode
cCmd |= (cMode << 1); // shift mode into bits 3-1
outp(PWM_CONTROL_REG,cCmd); // write to command register

//Value is 16 bits so sequentially send LSB followed by MSB
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue & 0xff));
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue >> 8));

return;
} // end of main
```

To make a PWM output we can use two counters; the first is to set the PWM frequency and the second to set the mark/space ratio.

To achieve this we will change the previous program so that counter 0 is set to mode 2

```
cMode=2; // counter mode
```

Running this program will give a waveform where the output is low only for a single CLK every 833uS



The next step is to use the rising edge of this single clock width pulse to trigger the second counter which must be set to expire at the end of the mark period.

A connection therefore needs to be made on the hardware between PWMO_OUT and PWM1_GATE.
Setting counter 1 to mode 1 with a value of 250 will demonstrate this.

```
// write to command register
cCnt=1; //counter number
cMode=1; // counter mode
lValue=250;

cCmd=0; // clear to start
cCmd |= (cCnt << 6); // shift counter number to bits 7-6
cCmd |= (0x03 << 4); // set RW bits to 16 bit mode
cCmd |= (cMode << 1); // shift mode int bits 3-1
outp(PWM_CONTROL_REG,cCmd); // write to command register

//Value is 16 bits so sequentially send LSB followed by MSB
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue & 0xff));
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue >> 8));
```

The output on PWM1_1 will then show a 25% mark space ratio.
Modifying the mark space ratio is achieved by adjusting the ratio between the initial count values of count1 and count 2.

Full software listing for PWM

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>

#define PCI_ADDRESS 0xcf8
#define PCI_DATA 0xcfc
#define PWM_COUNT_REG_0 0x48
#define PWM_CONTROL_REG 0x4B

// 32 bit operations
#define OPERAND32 asm db 66h

// prototypes
void outport32(unsigned long, unsigned long);
unsigned long inport32(unsigned long);

void main (void)
{
long Value;
int i;
unsigned char cCmd = 0x00;
long lValue = 0x00;
unsigned char cCnt = 0x00;
unsigned char cMode = 0x00;

// first set up the southbridge register
Value=inport32(0x800038c0);
Value &= 0xffffffc7ffL; // set bits 13-11 so all pll's use internal clk
Value |= 0x0000000004L; // set COM2 pins to be PWM
outport32(0x800038c0,Value);

// Set up counter 0
cCnt=0; //counter number
cMode=2; // counter mode
lValue=1000;

cCmd=0; // clear to start
cCmd |= (cCnt << 6); // shift counter number to bits 7-6
cCmd |= (0x03 << 4); // set RW bits to 16 bit mode
cCmd |= (cMode << 1); // shift mode into bits 3-1
outp(PWM_CONTROL_REG,cCmd); // write to command register

//Value is 16 bits so sequentially send LSB followed by MSB
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue & 0xff));
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue >> 8));

// Repeat for counter 1
// write to command register
cCnt=1; //counter number
cMode=1; // counter mode
lValue=250;

cCmd=0; // clear to start
cCmd |= (cCnt << 6); // shift counter number to bits 7-6
cCmd |= (0x03 << 4); // set RW bits to 16 bit mode
cCmd |= (cMode << 1); // shift mode into bits 3-1
outp(PWM_CONTROL_REG,cCmd); // write to command register

//Value is 16 bits so sequentially send LSB followed by MSB
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue & 0xff));
outp(PWM_COUNT_REG_0+cCnt,(unsigned char) (lValue >> 8));

return;
} // end of main
```