**Pulse Width Modulation**

Many embedded systems are built to have PWM as a hardware function since PWM is a cheap way to do digital to analog conversion. In PWM we start with a base frequency of say 1,000Hz and each cycle we output a single pulse whose width varies from 0% of the period to 100% of the period.

![Figure 1](image)

**Figure 1**

Pulse width modulation. In this case the signal is 50% of maximum.

The base frequency of the PWM is \( f = \frac{1}{T} \) so that the frequency content of a PWM signal will have frequencies at \( f \) and higher due to the base frequency. The signal information is transmitted by the duty cycle and will be, generally, a much lower frequency signal. Thus a relatively simple low pass filter can eliminate the frequencies at the base frequency and above and recover an analog signal from the PWM.

The Philips LPC2138 has 6 channels of pulse width modulation. There are 7 registers to accommodate the PWM with register 0 being used to set the base frequency. Thus, since there is only one base frequency register all 6 channels must have the same base frequency.

PWM on the LPC2138 can be single edge or double edged. Single edge PWM is as shown in Figure 1. In double edge PWM the ending point of the pulse and the starting point of the pulse is a variable and can be set each cycle. In single edge PWM only the ending edge of the PWM is a variable and the starting edge is always set at the base frequency.

Single edge PWM requires one register (plus the base register) so that you can have 6 single edge channels of PWM on the LPC2138. For double edged PWM you need a base register plus a starting edge register plus an ending edge register so that only 3 channels of double edged PWM is available on the LPC2138. You can mix single and double edged channels as long as you don't run out of the 6 available registers.

PWM on the LPC2138 also has a *latch enable register* which is used to clock the match register data into the shadow registers at the starting point of the base frequency cycle. If this is not done, the PWM could have a glitch (technical term) because the duty cycle
could change in the middle of a cycle. The figure below shows the LPC2138 PWM block diagram.

Note that each PWM channel has its own timer/counter so that it does not use Timer0 or Timer1.

![PWM block diagram](image)

**Figure 39: PWM block diagram**

Setting up a single channel PWM channel requires loading values into Match Register 0 (base frequency), Match Register x (for channel x) to determine the duty cycle, the Latch Enable Register, the Timer Counter register, the Prescale Register, the Timer Control Register, the PWM Control register, the periperal clock prescaler, and the IO direction register.

Here is an outline of the steps necessary to set up PWM on the LPC2138 an output a single value:

- Set up the pin select register to select PWM for the port you want.
• Set up the clock divider register for the PCLK. (VPBDIV)
• Set up the PWM timer prescale register. (PWMPR)
• Set up the IO Direction for the PWM pin you plan to use as output (a 1).
• Enable the particular PWM channel you want to use in the PWM Control register (PWMCR).
• Set the maximum number of counts in one cycle. This is done in match register 0. (PWMMR0).
• Set the value for the duty cycle in the particular match register you want to use.
  (PWMMRX where X is from 1 to 6).
• Set the PWM match control register to cause a counter reset when the match occurs. (PWMMCR).
• Set the PWM Latch enable register to enable the use of match values. (PWMLECR)
• Reset the timer counter using a bit in the PWM Timer Control Register. (PWMTCR)
• Enable the timer counter and enable the PWM mode using the PWM Timer Control Register. (PWMTCR)

The number of registers available and the similarity in their acronyms makes them hard to remember. The figure on the next page shows the PWM register map for the LPC2138 an all of their names with a short explanation of what they do.

For the ARM7 board being used in class PWM ports 1, 2, 4, 5, and 6 are brought out to pins on CN2. (PWM port 3 is multiplexed with the programming serial-in port and is not brought out.) For example, PWM Channel 4 which comes out on port 0 at P0.8 is on Pin 10 of CN2. This particular port is also used as the second UART on this chip so it is routed through the Max 563 communications chip. The output from the Max 563 goes to connector CN2 at pin 14. Thus, for this class, you can set up PWM on channel 4 and see the output on CN2 at either pin 10 or pin 14. Note that since the output at pin 14 goes through and RS-232 amplifier it will range roughly from +9 volts to -9 volts while the output on pin 10 will range from 0 to +3 volts.

To set up the registers begin with the IO direction and pin select registers. The pin select register values for port 0 are shown below:

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Function when 0</th>
<th>Function when 1</th>
<th>Function when 10</th>
<th>Function when 11</th>
<th>Reserved</th>
<th>Real value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>P0.0</td>
<td>GPIO Port 0.0</td>
<td>GPIO Port 0.0</td>
<td>GPIO Port 0.0</td>
<td>GPIO Port 0.0</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.1</td>
<td>P0.1</td>
<td>GPIO Port 0.1</td>
<td>GPIO Port 0.1</td>
<td>GPIO Port 0.1</td>
<td>GPIO Port 0.1</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.2</td>
<td>P0.2</td>
<td>GPIO Port 0.2</td>
<td>GPIO Port 0.2</td>
<td>GPIO Port 0.2</td>
<td>GPIO Port 0.2</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.3</td>
<td>P0.3</td>
<td>GPIO Port 0.3</td>
<td>GPIO Port 0.3</td>
<td>GPIO Port 0.3</td>
<td>GPIO Port 0.3</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.4</td>
<td>P0.4</td>
<td>GPIO Port 0.4</td>
<td>GPIO Port 0.4</td>
<td>GPIO Port 0.4</td>
<td>GPIO Port 0.4</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.5</td>
<td>P0.5</td>
<td>GPIO Port 0.5</td>
<td>GPIO Port 0.5</td>
<td>GPIO Port 0.5</td>
<td>GPIO Port 0.5</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.6</td>
<td>P0.6</td>
<td>GPIO Port 0.6</td>
<td>GPIO Port 0.6</td>
<td>GPIO Port 0.6</td>
<td>GPIO Port 0.6</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.7</td>
<td>P0.7</td>
<td>GPIO Port 0.7</td>
<td>GPIO Port 0.7</td>
<td>GPIO Port 0.7</td>
<td>GPIO Port 0.7</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.8</td>
<td>P0.8</td>
<td>GPIO Port 0.8</td>
<td>GPIO Port 0.8</td>
<td>GPIO Port 0.8</td>
<td>GPIO Port 0.8</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.9</td>
<td>P0.9</td>
<td>GPIO Port 0.9</td>
<td>GPIO Port 0.9</td>
<td>GPIO Port 0.9</td>
<td>GPIO Port 0.9</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.10</td>
<td>P0.10</td>
<td>GPIO Port 0.10</td>
<td>GPIO Port 0.10</td>
<td>GPIO Port 0.10</td>
<td>GPIO Port 0.10</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.11</td>
<td>P0.11</td>
<td>GPIO Port 0.11</td>
<td>GPIO Port 0.11</td>
<td>GPIO Port 0.11</td>
<td>GPIO Port 0.11</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.12</td>
<td>P0.12</td>
<td>GPIO Port 0.12</td>
<td>GPIO Port 0.12</td>
<td>GPIO Port 0.12</td>
<td>GPIO Port 0.12</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.13</td>
<td>P0.13</td>
<td>GPIO Port 0.13</td>
<td>GPIO Port 0.13</td>
<td>GPIO Port 0.13</td>
<td>GPIO Port 0.13</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.14</td>
<td>P0.14</td>
<td>GPIO Port 0.14</td>
<td>GPIO Port 0.14</td>
<td>GPIO Port 0.14</td>
<td>GPIO Port 0.14</td>
<td>Reserved</td>
<td>00</td>
</tr>
<tr>
<td>1.15</td>
<td>P0.15</td>
<td>GPIO Port 0.15</td>
<td>GPIO Port 0.15</td>
<td>GPIO Port 0.15</td>
<td>GPIO Port 0.15</td>
<td>Reserved</td>
<td>00</td>
</tr>
</tbody>
</table>

Pin Select register for P0 which contains the PWM ports.
Here is a detailed example of how to set up PWM on channel 4 and output a single value.

- **Set up the pin select register to select PWM for the port you want.**

  From the pin select register table we see that for PWM4 we need to set bit 17 to 1 and bit 16 to 0. This will allow PWM on channel 4 at P0.8.

  ```c
  PINSEL0 |= 0x00020000; // Change GPIO P0.8 to PWM4
  ```
• Set up the clock divider register for the PCLK. (VPBDIV)
  With the CCLK at 60MHz we can divide it by 4 to get a 15MHz PCLK. This
  requires that VPBDIV by all zeros.
  VPBDIV = 0x00000000; //Set up peripheral clock for 15MHz

• Set up the PWM timer prescale register. (PWMPR)
  We will set the prescale register to 0 which means no prescaling to the timer for
  PWM4 will be running at the PCLK rate of 15MHz.
  PWMPR = 0x00000000; // Set Pre Scale to 0

• Set up the IO Direction for the PWM pin you plan to use as output (a 1).
  We are using PWM channel 4 which comes out on P0.8. The I/O direction of that pin
  needs to be set to output.
  IODIR0 |= 0x00000100; // Set Direction of PWM4 Pin to be output

• Enable the particular PWM channel you want to use in the PWM Control register
  (PWMPCR).

<table>
<thead>
<tr>
<th>PWMPCR</th>
<th>Function</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>Reserved</td>
<td>Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>PWMSel2</td>
<td>When zero, selects single edge controlled mode for PWM2. When one, selects double edge controlled mode for the PWM2 output.</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>PWMSel3</td>
<td>When zero, selects single edge controlled mode for PWM3. When one, selects double edge controlled mode for the PWM3 output.</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>PWMSel4</td>
<td>When zero, selects single edge controlled mode for PWM4. When one, selects double edge controlled mode for the PWM4 output.</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>PWMSel5</td>
<td>When zero, selects single edge controlled mode for PWM5. When one, selects double edge controlled mode for the PWM5 output.</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>PWMSel6</td>
<td>When zero, selects single edge controlled mode for PWM6. When one, selects double edge controlled mode for the PWM6 output.</td>
<td>0</td>
</tr>
<tr>
<td>8:7</td>
<td>Reserved</td>
<td>Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.</td>
<td>NA</td>
</tr>
<tr>
<td>9</td>
<td>PWMA1</td>
<td>When one, enables the PWM1 output. When zero, disables the PWM1 output.</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>PWMA2</td>
<td>When one, enables the PWM2 output. When zero, disables the PWM2 output.</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>PWMA3</td>
<td>When one, enables the PWM3 output. When zero, disables the PWM3 output.</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>PWMA4</td>
<td>When one, enables the PWM4 output. When zero, disables the PWM4 output.</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>PWMA5</td>
<td>When one, enables the PWM5 output. When zero, disables the PWM5 output.</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>PWMA6</td>
<td>When one, enables the PWM6 output. When zero, disables the PWM6 output.</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td>Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.</td>
<td>NA</td>
</tr>
</tbody>
</table>

PWM Control Register bits.

The PWM Control Register (PWMPCR) is used to determine whether the PWM
signal is single edge or double edged. To enable channel 4 for single edge
operation we could do the following:
PWMPCR = 0x00001000; //Set PWM4 to enable with single edge

• Set the maximum number of counts in one cycle. This is done in match register 0.
  (PWMMR0).
  We will set the maximum count to 100H = 256 decimal.
  PWMMR0 = 0x00000100; // Max Value of Pulse Width

• Set the value for the duty cycle in the particular match register you want to use.
  (PWMMRX where X is from 1 to 6).
  This register determines the duty cycle of the PWM signal. For example, if we want
  a duty cycle of 1/8 or 12.5% we would set this register to 32 decimal or 20H since
  32/256 = 1/8.
  PWMMR4 = 0x00000020; // PWM4 Match Value

• Set the PWM match control register to cause a counter reset when the match occurs.
  (PWMMCR).

Table 132: PWM Match Control Register (PWMMCR - 0xE014014)

<table>
<thead>
<tr>
<th>PWMMCR</th>
<th>Function</th>
<th>Description</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt on PWMMR0</td>
<td>When one, an interrupt is generated when PWMMR5 matches the value in the PWMTTC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Reset on PWMMR0</td>
<td>When one, the PWMTC will be reset if PWMMR0 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Stop on PWMMR0</td>
<td>When one, the PWMTC and PWNPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR0 matches the PWMTC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Interrupt on PWMMR1</td>
<td>When one, an interrupt is generated when PWMMR1 matches the value in the PWMTC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Reset on PWMMR1</td>
<td>When one, the PWMTC will be reset if PWMMR1 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>Stop on PWMMR1</td>
<td>When one, the PWMTC and PWNPC will be stopped and PWMTCR[1] will be set to 0 if PWMMR1 matches the PWMTC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Interrupt on PWMMR2</td>
<td>When one, an interrupt is generated when PWMMR2 matches the value in the PWMTC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Reset on PWMMR2</td>
<td>When one, the PWMTC will be reset if PWMMR2 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Stop on PWMMR2</td>
<td>When one, the PWMTC and PWNPC will be stopped and PWMTCR[2] will be set to 0 if PWMMR2 matches the PWMTC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Interrupt on PWMMR3</td>
<td>When one, an interrupt is generated when PWMMR3 matches the value in the PWMTC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>Reset on PWMMR3</td>
<td>When one, the PWMTC will be reset if PWMMR3 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>Stop on PWMMR3</td>
<td>When one, the PWMTC and PWNPC will be stopped and PWMTCR[3] will be set to 0 if PWMMR3 matches the PWMTC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>Interrupt on PWMMR4</td>
<td>When one, an interrupt is generated when PWMMR4 matches the value in the PWMTC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>Reset on PWMMR4</td>
<td>When one, the PWMTC will be reset if PWMMR4 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>Stop on PWMMR4</td>
<td>When one, the PWMTC and PWNPC will be stopped and PWMTCR[4] will be set to 0 if PWMMR4 matches the PWMTC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>Interrupt on PWMMR5</td>
<td>When one, an interrupt is generated when PWMMR5 matches the value in the PWMTC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>Reset on PWMMR5</td>
<td>When one, the PWMTC will be reset if PWMMR5 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
</tbody>
</table>

PWM Match Control Register bits

For this example we want the PWM timer to be reset when a match occurs. We can
choose to use match register 0 so we need to set bit 1 of of PWMMCR to 1.
PWMMCR = 0x00000002; // Reset PWMTC when Matched with PWMMR0
• **Set the PWM Latch enable register to enable the use of match values. (PWMLER)**

The latch enable register enables the match register to update the shadow register only at the end of a base frequency cycle.

For our case we need to enable only the Match 0 latch and the Match 4 latch but we could also just enable all of them. For example

PWMLER = 0x7F; //Enable latch in all match registers

• **Reset the timer counter using a bit in the PWM Timer Control Register. (PWMTCR)**

Finally we need to reset and start the PWM timer counter register.

<table>
<thead>
<tr>
<th>PWMTCR</th>
<th>Function</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Counter Enable</td>
<td>When one, the PWM Timer Counter and PWM Prescaler Counter are enabled for</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>counting. When zero, the counters are disabled.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Counter Reset</td>
<td>When one, the PWM Timer Counter and the PWM Prescaler Counter are</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>synchronously reset on the next positive edge of pwm. The counters remain</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>remain reset until TON[1] is returned to zero.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>Reserved, user software should not write ones to reserved bits. The value</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>read from a reserved bit is not defined.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PWM Enable</td>
<td>When one, PWM mode is enabled. PWMM mode causes shadow registers to</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>operate in connection with the Match registers. A program write to a Match</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>register will not have an effect on the Match result until the corresponding</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit in PWMLER has been set. Followed by the occurrence of a PWM Match 0 event.</td>
<td></td>
</tr>
</tbody>
</table>

**PWM Timer Control Register**

Bit 1 does the reset and bit 3 and bit 0 to the enable.

For example,

PWMTCR = 0x02; //Reset PWMTCR

• **Enable the timer counter and enable the PWM mode using the PWM Timer Control**

Bit 3 and bit 0 are used to enable the counter and the PWM mode.

PWMTCR = 0x09; //Counter Enable, PWM Mode Enabled(bit0, bit3)
The program below is a complete example. It runs with VPBDIV not set so its reset value divides the 60MHz clock by 4. So the PCLK runs at 15MHz. There are 256 clocks per base cycle so the PWM base frequency is 15MHz/256 = 58.59375KHz.

```c
/*PWMTest.c
 This program illustrates how to use the PWM on Channel 4 in a
 single edge mode. PCLK is set to 15MHz and there is no prescaling
 so that the PWM clock is at 15MHz. The base rate is set to 100H
 which is 256 base 10 so the base PWM frequency is at 15MHz/256 =
 58.59375 KHz. The duty cycle is set to 20H = 32. The resulting
 PWM is signal will be high for 32 of every 256 tics.
 */
#include<LPC213x.h>
void main()
{
    PINSEL0 |= 0x00020000; // Change GPIO P0.8 to PWM4
    VPBDIV = 0x00000000;  //Set up peripheral clock for 15MHz
    PWMPR = 0x00000000;    // Set Pre Scale to 0
    PWMPCR = 0x00001000; // Enable PWM4 Output (bit12)
    PWMMR0 = 0x00000100; // Max Value of Pulse Width
    PWMMR4 = 0x00000020; // PWM4 Match Value
    PWMMCR = 0x00000002; // Reset PWMTC when Matched with PWMMR0
    PWMLER = 0x7F; // Latch in all PWMMR's
    PWMTCR = 0x02; // Counter Enable, PWM Mode Enabled(bit0, bit3)
    PWMTCR = 0x09; // Reset PWMTC, (bit1)
    while(1);
}
```

Note that the duty cycle in this case is set to 20H = 32 clock tics. On the ARM 7 board you can observe this signal on pin 10 of CN2. You can see it inverted (through the RS-232 amplifier) on pin 14.

**Class exercise 8-1:** Modify PWMTest.c so that the base frequency is 100KHz and the duty cycle is 60%.

**PWM Modulation**

To modulate the PWM signal with say a sinusoid, it is necessary to periodically update the match register (PWMMR4 in our case). For a signal to have sinusoidal modulation you need to write the sinusoid amplitude, properly scaled, into the match register and update the latch enable register (PWMLER). The latch enable register allows the match register to be written to the shadow register at the end of the PWM base frequency cycle. (If this is not done, the match register could get updated in mid cycle causing a glitch in the output.)

For example, if x and incr are floats with incr = 2π/100, the following segment modulates the PWM signal in the program above as a sinusoid.
/*PWMSine.c
This program illustrates how to use the PWM on Channel 4 in a single edge mode to output a sinusoid on the PWM channel. PCLK is set to 15MHz and there is no prescaling so that the PWM clock is at 15MHz. The base rate is set to 100H which is 256 base 10 so the base PWM frequency is at 15MHz/256 = 58.95375 KHz. */
#include<LPC213x.h>
#include <MATH.H>
const float PI = 3.141592654;
int main()
{
    int i;
    float x, incr;
    incr = 2*PI/100;
    PINSEL0 |= 0x00020000; // Change GPIO P0.8 to PWM4
    VPBDIV = 0x00000000;  //Set up peripheral clock for 15MHz
    FWMPR = 0x00000000; // Set Pre Scale to 0
    FWMPCR = 0x00001000; // Enable PWM4 Output (bit12)
    PWMMR0 = 0x00000100; // Max Value of Pulse Width
    PWMMCR = 0x00000000; // Reset PWMMTC when Matched with PWMMR0
    FWMLER = 0x7F;  // Latch in all PWMMMR's
    PWMTCR = 0x002; // Reset PWMMTC, (bit0, bit3)
    FWMTCR = 0x09; // Counter Enable, PWM Mode Enabled(bit0, bit3)
    while(1)
    {
        x = 0;
        for(i=0;i<100;i++)
        {
            x = x + incr;
            //Scale x value and write it to the match register 4
            PWMMR4 = (int)(40*(sin(x) + 1));
            FWMLER = 0x00000010; //update the latch enable register
        }
    }
}

Note that PWMSine.c is updating the PWMMR4 match register much more than is necessary. This is because the PWMLER setting is such that the match register is only changed at the beginning of each base cycle which is 100 times per second. Since the program loop is running as fast as it can go, many of the values are being written but have no effect since the latch enable register does not copy them into the match register.

You can use the μVision 3 IDE Logic Analyzer to look at this signal. To set up the logic analyzer, click on View → Logic Analyzer Window while in a Debug session. From the logic analyzer window click on Setup to get the screen shown below. Push Insert and enter Port0.8. (Note that it is important that you type Port0.8 even though the window shows only Port0). Be sure the Display type is set to Bit and choose your favorite color.
Logic Analyzer Setup
Click on Insert and enter Port0.8.

Close the set up window and click on run in the debugger. Select the Logic Analyzer window and click on Zoom In or Zoom Out to get the display shown below.

Logic Analyzer Window
Sinusoidal modulation

Class exercise 8-2: Modify PWMSine.c to output a ramp function. Use a base frequency of 100KHz.