The ARM architecture features two timer/counters named Timer0 and Timer1. Each timer has the following features:

- A 32-bit Timer/Counter with a programmable 32-bit Prescaler. The prescaler allows the user to divide the clock rate coming into the timer.
- Counter or Timer operation.
- Up to four 32-bit capture channels per timer, that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- Four 32-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Up to four external outputs corresponding to match registers, with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Toggle on match.
  - Do nothing on match.

All of the timers are driven by \( PCLK \) which is the Peripheral Clock. The system is generally driven by the \( CCLK \) or Crystal Clock.

\( PCLK \) is derived from \( CCLK \) which is the processor clock. In our case, \( CCLK \) runs at 60MHz. The VPBDIV register determines the rate of \( PCLK \). VPBDIV is not the same as the prescaler register although its action is similar. VPBDIV is an 8-bit register and only the lower two bits of it are used as follows:

<table>
<thead>
<tr>
<th>B1</th>
<th>B0</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( PCLK = CCLK/4 )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( PCLK = CCLK )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( PCLK = CCLK/2 )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Not used</td>
</tr>
</tbody>
</table>

**VPBDIV values and PCLK**

For example

\[
\text{VPBDIV} = 0x02; \quad \text{//Set up peripheral clock for 30MHz}
\]

sets bit 1 and bit 0 to 10 making the PCLK run at \( 60\text{MHz}/2 = 30\text{MHz} \)

PCLK goes into a prescaler which further scales the clock going to the timer. The output of the prescaler actually drives the timer register.
Derivation of the peripheral clock (PCLK)

* Note that Capture Register 3 cannot be used on TIMER0

Figure 48: Timer block diagram
To use the prescaler you load the Prescale register. The PCLK is scaled by the value in the prescale register plus 1.

For example

```
VPBDIV = 0x02;        //Set up peripheral clock for 30MHz
T0PR = 0x0000001D;    //Set prescaler to 29 so prescaled by 30 which
                      //corresponds to -> 1MHz timer clock
```

The timer control register is used to reset the Timer Counter and enable or disable the prescaler.

**Timer Control Register (TCR: TIMER0 - T0TCR: 0xE0004004; TIMER1 - T1TCR: 0xE0008004)**

The Timer Control Register (TCR) is used to control the operation of the Timer Counter.

<table>
<thead>
<tr>
<th>TCR</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 Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled.</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Counter Reset</td>
<td>When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of poll. The counters remain reset until TCR[1] is returned to zero.</td>
<td>0</td>
</tr>
</tbody>
</table>

For example:

```
T0TCR = 0x00000002; //Reset Timer0 & prescaler; disable prescale counter
```

The register T0TC is the timer counter register itself. You can read this register and run the timer in a polling mode.

For example, PolledTimer.c outputs a square wave to P1.16 using timer 0 in the polled mode. Each microsecond it complements the bit at P1.16 giving a 2 μsec cycle time.

```c
/*PolledTimer.c                                            July 4, 2007
   This program outputs a 2 KHz square wave on pin P1.16 using
   Timer 0.  The timer is polled.
*/
#include <LPC213X.H>

int main(void)
{
  VPBDIV = 0x02;        //Set up peripheral clock for 30MHz
  PINSEL2 = 0x00000000;  //Sets all pins on Port 1 as GPIO
  T0PR = 0x0000001D;    //Set prescaler to 29 -> 1MHz timer clock
  T0TCR = 0x00000002;   //Reset Timer0 & prescaler;
                         // disable prescale counter
  IODIR1 = 0x00010000;  //Set P1.16 to output. All others are input
  T0TCR = 0x00000001;   //Start counter 0
  while(1)
  {
    while(T0TC != 0x000003E8); //Wait until counter 0 gets to 1000
    T0TCR = 0x00000002; //Reset Timer0 & prescaler;
                         // disable prescale counter
    T0TCR = 0x00000001;  //Start counter 0
    IOPIN1 = ~IOPIN1;    //Complements output bit on P1.16
  }
}
```

Polled Timer example
The following example called Poll2Timer.c outputs a waveform to P1.16 which has a 1 μsec high time and a 5 μsec low time. This is done with two different timers.

```c
/*Poll2Timer.c                                             July 4, 2007
This program outputs a wave on pin P1.16 using two timers. The
high time is 1 usec and the low time is 5 usec.
Both timers are polled.
*/
#include <LPC213X.H>

int main(void)
{
    VPBDIV = 0x02;        //Set up peripheral clock for 30MHz
    PINSEL2 = 0x00000000;  //Sets all pins on Port 1 as GPIO
    T0PR = 0x0000001D;    //Set prescaler 0 to 29 -> 1MHz timer0 clock
    T1PR = 0x0000001D;    //Set prescaler 1 to 29 -> 1 MHz timer1 clock
    T0TCR = 0x00000002;   //Reset Timer0 & prescaler;
    //  disable prescale counter0
    T1TCR = 0x00000002;   //Reset Timer1 & prescaler;
    //  disable prescale counter1
    IODIR1 = 0x00010000;  //Set P1.16 to output. All others are input
    T0TCR = 0x00000001;   //Start counter 0
    while(1)
    {
        IOSET1 = 0x00010000;       //P1.16 = 1;
        while(T0TC != 0x000003E8); //Wait until counter 0 gets to 1000
        IOCLR1 = 0x00010000;       //P1.16 = 0;
        T0TCR = 0x00000002;       //Reset Timer0 & prescaler;
        // disable prescale counter
        T1TCR = 0x00000001;       //Start counter 1
        while(T1TC != 0x00001388); //Wait until counter 1 gets to 5000
        T1TCR = 0x00000002;       //Reset Timer1 & prescaler;
        // disable prescale counter
        T0TCR = 0x00000001;       //Start timer 0
    }
}
```

Example using two polled timers

**Class Exercise 6-1:** Rewrite the example using two polled timers to using only one timer.

**Class Exercise 6-2:** Use floating point arithmetic calculate the value of a 1KHz sinusoid at 10 steps/cycle. Scale these values and output them to the D to A converter. Use Timer0 to send out one sample every 100 μsec.

**Class Exercise 6-3:** WavArray.c from Lecture 4 shows how to output half sinusoid by storing the values in an array. Modify this program so that the half wave sinusoid has a period of 200 μsec using a polled timer.
This program outputs an arbitrary waveform where the waveform samples are stored in an array.

```c
#include <LPC213X.H>

const int arr[] = {0, 316, 601, 828, 973, 1023, 973, 828, 601, 316,
                   0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

int main(void)
{
    int i, delay;
    PINSEL1 = 0x00080000; // P0.25 set to DA Out
    while(1)
    {
        for(i=0;i<20;i++)
        {
            DACR = arr[i] << 6;
            for(delay=0;delay<10000;delay++);
        }
    }
}
```