Timing in microcontroller applications

Microcontrollers are often required to measure time intervals. Typical applications include:

- Flow meters
- Speedometers
- Time of day clocks

If applications such as these are to function correctly then an accurate measure of time is required. How can this be achieved? The first ingredient is a stable, predictable clock signal. In the digital world, a clock signal is one that switches from low to high and back again at a particular rate. There are several ways to achieve this, one of which is shown in Figure 1 below

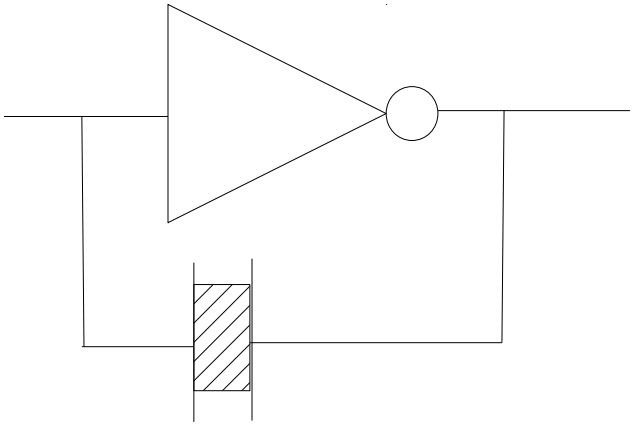

Figure 1 A Pierce oscillator. If you take the output from an inverter (NOT gate) and connect it back to its input you will create an oscillator. The frequency of this oscillator is not very predictable however, if you also include a crystal resonator which operates just like a very high frequency tuning fork then the inverter will switch at a rate that is determined by the size and geometry of the crystal. Manufacturers can produce crystals that are trimmed to resonate at particular frequencies with an error of only a few parts per million.

Having established a stable clock signal, the next ingredient required is a digital counter. A digital counter is a logic circuit (a collectiono f lip flops with some "glue" logic) that counts pulses (actually most counters count clock signal edges i.e. A transition from low to high or high to low).
Example:
A microcontroller has a clock signal that runs at 1 million cycles per second (1MHz). Assuming the counter starts at zero, what value will be in the counter after 0.1 of a second?
Answer: Counter Value = Interval x Clock frequency
Counter Value = 0.1 x 1000000
Counter Value = 100000

Digital counters have a finite number of bits. An 8 bit counter can count up to a maximum value of 255. A 16 bit counter has a maximum count of 65535. What happens when you send another clock pulse into a 16 bit counter which has already counter up to 65535? The counter will overflow and its new value will be **0**. Applications that use counters to measure time intervals must be conscious of the number of bits in these counters. If overflows occur, corrective action must be taken by the application software.

Example:

A microcontroller has a clock signal that runs at 32768Hz (very common in digital watches). The counter also has a 16 bit counter coupled to the clock that can be accessed using the symbol TAR in application code. Write a function that takes 1 millisecond to execute.

Answer

```
void delay_milli() {
// Approximately one millisecond delay routine
     unsigned short start;
     start = TAR;
     while ( (TAR-start) < 32); // wait for time to expire
}
```

How accurate is this routine (neglect the overhead of entering the function and managing the variables)?

Measuring intervals between events.

Very frequently, microcontrollers are required to measure the time between two events. Suppose a microcontroller needs to measure the length of time a signal is high on bit 0 (B0) of Port 1 in an MSP430. The following code could be used to measure this width.

```
unsigned short measure_pulse( )
{
  unsigned short start,stop;
  while ((P1IN & 0x1) == 0); // wait for input signal to go high
  start = TAR;
  while ((P1IN & 0x1) != 0); // wait for input signal to go low again
  stop = TAR;
  return stop-start;
}
```

Once again, it should be noted that care must be taken to ensure that the counter does not overflow.