

# Displaying data on LED display

We have learned a lot so far, we can call ourselves programmers now. Engineers of electronics have always wanted to display values on LED displays; Its always either voltage, current, or frequency.

This chapter will help us reach those, so far unattainable dreams. A novice programmer will typically have problems displaying data on a display. This is why I chose to use the Clock0.bas program, originally written by Mr.Mirko Pelcl and modified it for this publication.

As usual the program starts with variable declaration.

```
Dim Clock As Byte , Clock1 As Byte , Mux As Byte , Seconds As Byte , X As Byte
Dim Var As Byte , Segments As Byte , Ones As Byte , Tenths As Byte
Dim Display As Bit , Calc As Bit

Config Timer0 = Timer , Gate = Internal , Mode = 2           'configuration of Timer

'Timer0           we will use timer 0
'Gate = Internal  without external interrupt
'Mode = 2         8 bit auto reload
```

If you've been paying attention you probably noticed the new structure. That is the **Config** command. **Config** specifies what will happen with **Timer**. And what is **Timer**? Let's have another look at the internal diagram of the microcontroller at figure 1 and figure 2.

Note **Timer1**, **Timer2** etc with matching inputs in the upper right corner. Let's assume that **Timer** works as a preload counter, counting up to 100. Lets make it use internal CPU clock. We want it to signal with an output bit when it finishes counting to 100, and then we want it to reset itself back to 0 and count to 100 again. This output bit is also called **Interrupt**. Why is it called **interrupt**; because **Timer**

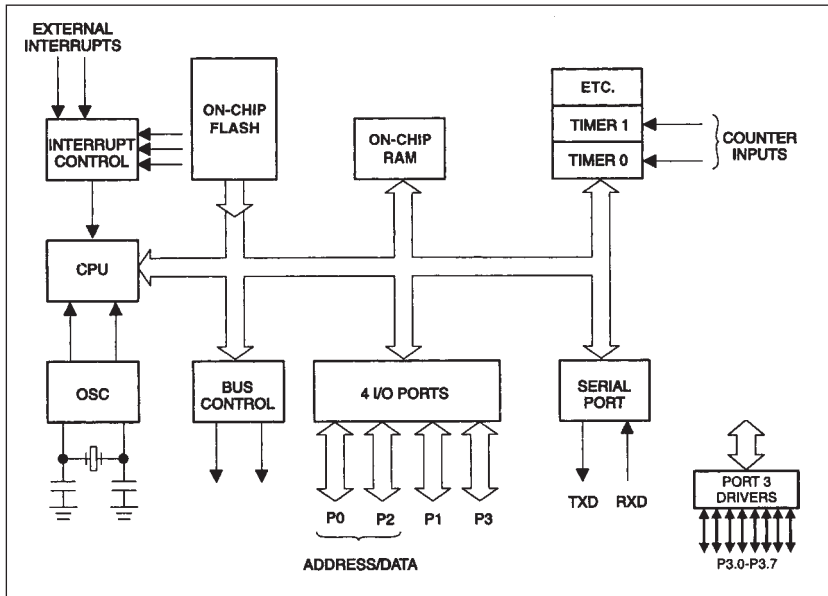


Figure 1: Block diagram of the microcontroller

has higher priority than other parts of the program! Executing of the program will stop whenever an interrupt bit occurs. The program will continue execution in the interrupt subroutine, complete the subroutine and return to the initial program exactly to the same point. You can read more on Timers and Counters in next chapters of this manual.

This is how **Timer** works. But **Timer** can also work as **Counter**. In this case we won't set the **Counter** to count up to a certain value, we just set it to count. And when it finishes counting its 16 bits (that's all its got, up to  $2^{16} = 65536$ ), it signals this with a bit at the output. **Counter** shall count pulses, brought in from the outside via one of the ports. **Counter** generates an interrupt bit. Here, I came close to describing a **Timer** and **Counter**. Still, I'm going to use **Timer** here.

Command:

```
On Timer0 Timer_0_int      'interrupt routine
```

When the **Timer0** counts to the preset value, the program jumps to the **Timer\_0\_int** routine every time the **Timer0** generates an interrupt bit.