

# 1 PIC16F884 Timing Programs

```
1 delay
2     movwf    delay_count
3 delay_decrement
4     decfsz  delay_count ,F
5     goto    delay_continue
6 delay_end
7     nop      ; m2 nop instructions go here
8     ...
9     return
10 delay_continue
11    nop      ; m1 nop instructions go here
12    ...
13    goto    delay_decrement
```

**Timing Analysis:** Let  $n$  be the value in the `delay_count` register. Whenever  $n > 2$ , the `decfsz` instruction in line 4 will not result in 0. It will then take only a single instruction cycle to execute and the next instruction to execute will be the `goto` instruction in line 5. After this, the  $m_1$  `nop` instructions starting in line 11 will be executed, along with the `goto` instruction in line 13. This will consume  $1 + 2 + m_1 + 2 = 5 + m_1$  cycles.

This series of instructions will be repeated  $n - 1$  times for a total of

$$\begin{aligned} N &= (n - 1)(5 + m_1) \\ &= 5n - 5 + nm_1 - m_1 \end{aligned}$$

cycles.

When  $n = 1$ , the `decfsz` instruction in line 4 does finally result in a zero, causing it to consume two instruction cycles instead of just one and causing the  $m_2$  `nop` instructions beginning in line 7 to be executed, followed by the `return` instruction in line 9. This series of instructions takes  $2 + m_2 + 2 = m_2 + 4$  instruction cycles. When added to the earlier total, to the single cycle consumed by the `movwf` instruction in line 2, and to the two cycles consumed by the `call` instruction used to enter the subroutine in the first

place, we get

$$\begin{aligned} N &= 1 + 5n - 5 + nm_1 - m_1 + m_2 + 4 + 2 \\ &= 2 + 5n + nm_1 - m_1 + m_2. \end{aligned}$$

**Performance with  $f_{\text{XTAL}} = 4$  MHz:** Suppose  $f_{\text{XTAL}} = 4$  MHz, so  $f_{\text{INST}} = 1$  MHz and  $T_{\text{INST}} = \frac{1}{f_{\text{INST}}} = 1 \mu\text{s}$ . The time consumed by the routine, therefore, is

$$\begin{aligned} T_{\text{DELAY}} &= nT_{\text{INST}} \\ &= N \mu\text{s} \\ T_{\text{DELAY}} &= (2 + 5n + nm_1 - m_1 + m_2) \mu\text{s}. \end{aligned} \tag{1}$$

If we call the subroutine with the value  $n$  in the W register, how can we cause this to consume time  $T = (n \times 10) \mu\text{s}$ ?

What we want to do is set

$$\begin{aligned} 10n &= 2 + 5n + nm_1 - m_1 + m_2 \\ 5n &= m_1n + m_2 - m_1 + 2. \end{aligned}$$

This is an easy equation to solve if we choose  $m_2 - m_1 + 2 = 0$ :

$$\begin{aligned} 5n &= m_1n - m_1 + m_2 + 2 \\ &= m_1n \\ m_1 &= 5. \end{aligned}$$

Substituting this back into (1) gives

$$\begin{aligned} T_{\text{DELAY}} &= (2 + 5n + nm_1 - m_1 + m_2) \mu\text{s} \\ &= (5 + m_1)n - m_1 + m_2 + 2) \mu\text{s} \\ &= (5 + 5)n) \mu\text{s} \\ &= 10n \mu\text{s}, \end{aligned}$$

as desired.

Now to get a delay of, say,  $100 \mu\text{s}$ , all we need to do is place the value 10 in the W register and then call this delay routine.

Note that putting the number 0 into the W register will be equivalent to specifying  $n = 256$  because, after the first decrement of line 4, `delay_count` will have the value 255. The delay actually is specified as

$$T_{\text{DELAY}} = \begin{cases} 10n \mu\text{s} & \text{if } n \in \{1, 255\} \\ 2.56 \text{ ms} & \text{if } n = 0. \end{cases}$$

Longer delays can be achieved either by modifying the number of **nop** instructions (that is, changing  $m_1$  and  $m_2$ ,) or by calling this subroutine multiple times. Since doing this will introduce extra overhead not accounted for in this analysis, the delay may not be an exact multiple of  $10 \mu\text{s}$ . Calling the subroutine one time fewer than needed and then adding some additional delay can correct the problem if this is important.

Of course, this method ties up the processor completely while the delay takes place and so is a completely unsuitable method to use if the processor has other work to do while waiting. In such a case, it makes much more sense to use interrupt processing and the PIC16F884's built-in timer hardware.