Managing Multiple Tasks

Introduction. This application note presents a program in TechTools assembly language that demonstrates a technique for organizing a program into multiple tasks.

Background. Like most computers, the PIC executes its instructions one at a time. People tend to write programs that work the same way; they perform one task at a time.

It's often useful to have the controller do more than one thing at a time, or at least seem to. The first step in this direction is often to exploit the dead time from one task—the time it would normally spend in a delay loop, for instance—to handle a second task. The PIC's speed makes this quite practical in many cases.

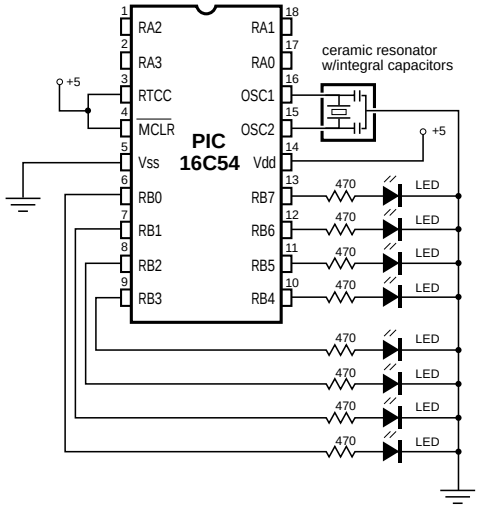
When several tasks must be handled at once, this approach can quickly become unworkable. What we need is a framework around which to organize the tasks. We need an operating system.

The program in the listing illustrates an extremely simple operating system that runs each of eight small subprograms in turn. When the subprograms finish their work, they jump back to the system. Notice that this method does not require the call instruction, so it leaves the two-level stack free for the use of the subprograms.

How it works. The circuit and program comprise an eight-LED flasher. Each of the LED's flashes at a different rate. While this could be accomplished differently, the program is easier to understand and maintain because the code that controls each LED is a separate task.

The "system" portion of the program acts like a spinning rotary switch. Each time it executes, it increments the task number and switches to the next task. It does this by taking advantage of the PIC's ability to modify the program counter. Once the task number is loaded into the working register, the program executes the instruction *jmp pc+w*. The destinations of these jumps contain *jmp* instructions themselves, and send the program to one of the eight tasks. Not surprisingly, a list of *jmp* instructions arranged like this is called a "jump table."

Modifications. For the sake of simplicity, this task-switching program



lacks one important attribute: fixed timing. The individual tasks are permitted to take as much or as little time as they require. In some real applications, this wouldn't be acceptable. If the system maintains a master timer (like the variable *ticks* in this program) it should increment at a consistent rate.

With many possible paths through the code for each task, this may seem like another problem. A straightforward solution is to use the PIC's RTCC to time how long a particular task took, then use that number to set a delay to use up all of the task's remaining time. All you need to know is the worst-case timing for a given task.

Program listing. This program may be downloaded from our Internet ftp site at ftp.tech-tools.com. The ftp site may be accessed directly or through our web site at http://www.tech-tools.com.

; PROGRAM: TASK.SRC

```
; This program demonstrates task switching using the PIC's relative addressing
; mode. It handles eight tasks, each of which flashes an LED at a different rate. The
; flashing routines base their timing on a master clock variable called ticks.

; Remember to change device info if using a different part.
                device      pic16c54,xt_osc,wdt_off,protect_off
                reset       start

LEDs            =           rb

; Put variable storage above special-purpose registers.
                org          8

task            ds          1                   ; The task number used by the
                                                ; system.
ticks           ds          1                   ; Master time clock, increments
                                                ; once for each system cycle.
time0           ds          1                   ; Timer for task 0.
time1           ds          1                   ; Timer for task 1.
time2           ds          1                   ; Timer for task 2.
time3           ds          1                   ; Timer for task 3.
time4           ds          1                   ; Timer for task 4.
time5           ds          1                   ; Timer for task 5.
time6           ds          1                   ; Timer for task 6.
time7           ds          1                   ; Timer for task 7.

; Set starting point in program ROM to zero.
                org         0

start           mov         !rb,#00000000b      ; Set port rb to output.
                mov         task, #7            ; Set task number.
                clr         ticks               ; Clear system clock.
                clr         LEDs                ; Clear LEDs

system          inc         task                ; Next task number.
                cjne        task, #8, :cont     ; No rollover? Continue.
                clr         task                ; Rollover: reset task and
                inc         ticks               ; increment the clock.
:cont           mov         w, task             ; Prepare to jump.
                jmp         pc+w                ; Jump into table, and from there
                jmp         task0               ; to task #.
                jmp         task1
                jmp         task2
                jmp         task3
                jmp         task4
                jmp         task5
                jmp         task6
                jmp         task7
```

```
task0   cjne    ticks, #255,:cont   ; Every 255 ticks of system clock
        inc     time0               ; increment task timer. Every 3
                                    ; ticks
        cjne    time0, #3, :cont    ; of task timer, toggle LED, and
        clr     time0               ; reset task timer.
        xor     LEDs, #00000001b
:cont   jmp     system

task1   cjne    ticks, #255,:cont
        inc     time1
        cjne    time1, #8, :cont
        clr     time1
        xor     LEDs, #00000010b
:cont   jmp     system

task2   cjne    ticks, #255,:cont
        inc     time2
        cjne    time2, #6,:cont
        clr     time2
        xor     LEDs, #00000100b
:cont   jmp     system

task3   cjne    ticks, #255,:cont
        inc     time3
        cjne    time3, #11,:cont
        clr     time3
        xor     LEDs, #00001000b
:cont   jmp     system

task4   cjne    ticks, #255,:cont
        inc     time4
        cjne    time4, #12, :cont
        clr     time4
        xor     LEDs, #00010000b
:cont   jmp     system

task5   cjne    ticks, #255,:cont
        inc     time5
        cjne    time5, #4, :cont
        clr     time5
        xor     LEDs, #00100000b
:cont   jmp     system

task6   cjne    ticks, #255,:cont
        inc     time6
        cjne    time6, #23,:cont
        clr     time6
        xor     LEDs, #01000000b
```

```
:cont       jmp     system

task7       cjne    ticks, #255,:cont
            inc     time7
            cjne    time7, #9,:cont
            clr     time7
            xor     LEDs, #10000000b
:cont       jmp     system
```