State Machine IO Programming

The loop structures of Figures 8.18 and 8.19 wait for an IO event (switch press and release) and then perform an action. A common task in microcontroller applications is to perform a sequence of events that span a series of IO actions. A finite state machine approach for code structure is useful for these types of problems. Figure 8.20 shows a state machine specification of an LED/switch IO problem. Each state accomplishes an action, such as turning the LED off, turning the LED on, or blinking the LED. Transitions between states are controlled by an event on the pushbutton, which is a press, a release, or both a press and release. State OFF turns the LED off and transitions to state ON by a press and release. State ON turns on the LED and transitions to the next state on a press and release. The next state from the ON state is state OFF if the RB7 input is 0; else the next state is the BLINK state. The BLINK state flashes the LED until the pushbutton is pressed, at which point it transitions to state STOP. The stop STATE freezes the LED on as long as the pushbutton is pressed. State STOP transitions to state OFF when the pushbutton is released.

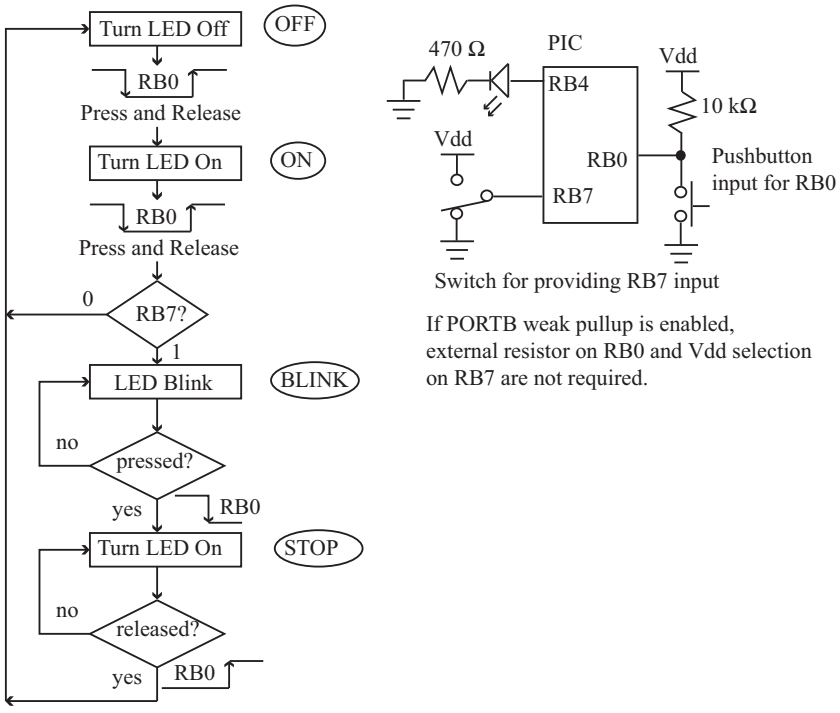

FIGURE 8.20 State machine specification of LED/switch IO.

Figure 8.21 gives a *C* code implementation of the LED/switch IO state machine of Figure 8.20. The `#define` statements define labels for each state with the state assignment arbitrarily chosen to start at 0. In a software state machine, the state assignments are usually unimportant, unlike a hardware finite state machine in which state assignments affect the logic generated for the state machine implementation. The `unsigned char state` variable is used to keep track of the current state.

```
// State definitions
#define LED_OFF     0  // turn off
#define LED_ON      1  // turn on
#define LED_BLINK   2  // start blinking
#define LED_STOP    3  // stop blinking
unsigned char state;
main(void){
  serial_init(95,1); // 19200 in HSPLL mode, crystal = 7.3728 MHz
  pcrlf(); // this subroutine prints a newline to the terminal
  printf("Led Switch/IO started");pcrlf();
  // RB4 is the output, RB7, RB0 are inputs
  TRISB = 0xEF; LATB = 0x00; STATE = LED_OFF;
  // enable the weak pullup on port B
  RBPU = 0;
  while(1) {
    switch (state) {
      case LED_OFF:
         printf("LED_OFF");pcrlf();
          LATB4 = 0;
          while(RB0);DelayMs(30);  // wait for press
          while(!RB0);DelayMs(30); // wait for release
          state = LED_ON;
          break;
      case LED_ON:
         printf("LED_ON");pcrlf();
          LATB4 = 1;
          while(RB0);DelayMs(30);  // wait for press
          while(!RB0);DelayMs(30); // wait for release
          if (RB7) state = LED_BLINK;
           else state = LED_OFF;
          break;
      case LED_BLINK:
        printf("LED_BLINK");pcrlf();
        while (RB0) { // while not pressed
          // toggle LED
          if (LATB4) LATB4 = 0;
             else LATB4 = 1;
          DelayMs(250);
        }
        DelayMs(30);
        state = LED_STOP;
        break;
      case LED_STOP:
        printf("LED_STOP");pcrlf();
        LATB4 = 1;  // freeze on
        while(!RB0);DelayMs(30); // wait for release
        state = LED_OFF;
        break;
    }
  }
}
```

State Definitions

Variable for tracking current state

Enable weak pullup

printf statements in each state are included for debugging.

Could use RB4 here as well

Change state variable so next time through loop will execute new **case** block.

Chooses next state based on RB7 value

Toggles LED each time through the loop, delay so we can see LED blink

Must have **break** at end of each **case** block or will execute next **case** block!!!!

FIGURE 8.21 *C* code for LED/switch IO.

The `main()` code performs initialization of the serial port and PORTB, and then enters a `while(1){}` loop that uses a *C* `switch` statement to execute different code segments based upon the `state` variable. A `printf()` statement that prints the state name is the first statement in each `case` block and is included for debugging purposes. Each `case` block performs its associated action and only changes the state variable to the next state once its specified pushbutton event is detected. It is very important to end each `case` block with a `break` statement, or else the next `case` block code is executed regardless of the `state` value. Reading the current state of LB4 (data latch port B, bit 4) and complementing it toggles the LED in the LED_BLINK state. A read of RB4 can be used here instead of LB4, because the external pin value will be the same as the data latch value because there are not multiple drivers on the RB4 external pin, so no possibility of driver conflict exists. However, in general, if you need to read the last value written to an output port, the data latch register should be read instead of the port register.

Figure 8.22 shows console output while testing the *C* code of Figure 8.21. The RB7 input was "1" for the first two times that the LED_ON state was exited, causing the next state to be LED_BLINK. After this, the RB7 input was low the next two times that the LED_ON state was exited, causing the following state to be LED_OFF.

```
Led Switch/IO started
LED_OFF   <-------- Initial state
LED_ON    <-- Press & Release
LED_BLINK <-- Press & Release, RB7 = 1, so blink LED
LED_STOP  <-- Press, stop blinking
LED_OFF   <-- Release, turn off
LED_ON    <-- Press & Release
LED_BLINK <-- Press & Release, RB7 = 1, so blink LED
LED_STOP  <-- Press, stop blinking
LED_OFF   <-- Release, turn off
LED_ON    <-- Press & Release
LED_OFF   <-- Press & Release, RB7 = 0, so go back to OFF
LED_ON    <-- Press & Release
LED_OFF   <-- Press & Release, RB7 = 0, so go back to OFF
```

FIGURE 8.22 Console output for LED/switch IO *C* code.

8.11 INTERFACING TO AN LCD MODULE

A liquid crystal display (LCD) is often used in microcontroller applications, as they are low power and can display both alphanumeric and graphics. Disadvantages of LCDs are that they have low viewing angles, are more expensive than LED displays, and must be lit by either ambient light or a back light. LCD *modules* display multiple characters; with part numbers using a *k* x *n* designation where *k* is the number