

LESSON 7: WRITING PROGRAM2: ONBOARD GPIO

Objectives:

1. Study hardware and software drivers for onboard GPIO
2. Write a program that interfaces GPIO
3. Draw State Transition Diagram

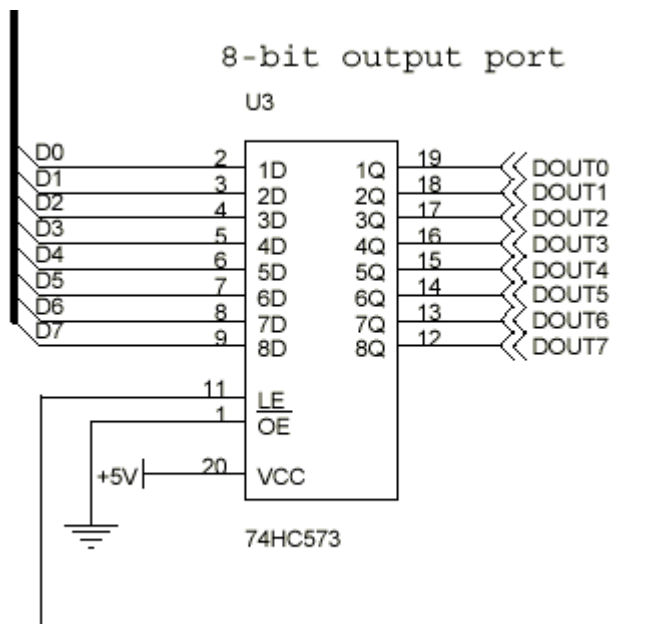
The 8051SBC provides a number of I/O devices for easy learning. We can write the driver for these devices with assembly code. This lesson shows how to write the driver for onboard GPIO. Writing the I/O drivers, we must know details of hardware beforehand.

GPIO1

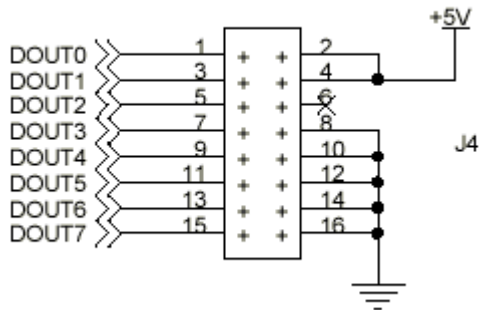
GPIO stands for General Purpose Input Output. The most common specification of such GPIO is simple digital input and output port. The 8051SBC has two ports: 8-bit input port and 8-bit output port. The locations of both ports are mapped into external data memory with address below:

GPIO1	8-bit OUTPUT PORT	100H
GPIO2	8-bit INPUT PORT	200H

Let us see the hardware for GPIO1.



GPIO1 is built with 8-bit D type Flip-Flop, 74HC573. The input signals are tied to 8051's data bus, D0-D7. The LE, Latch Enable signal activates when writing to the external data memory at address 100H. The 8-bit output signals are terminated at J4 header.



Pin1 of the header J4 has square pad, we may see the bottom pcb to find it.

Examples

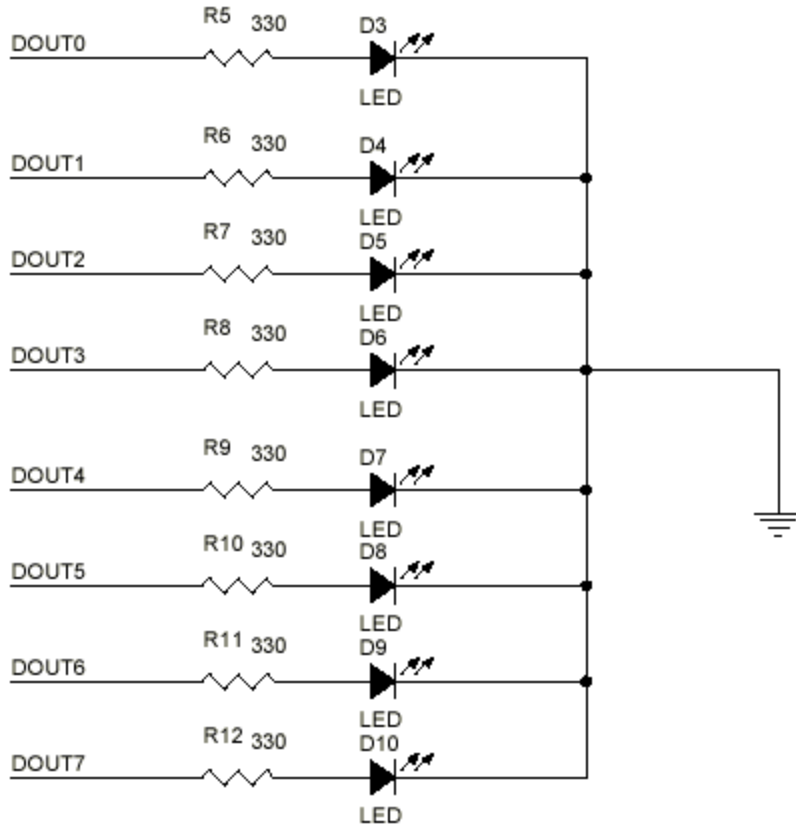
To write a byte to GPIO1, we can use below code,

```

mov a,#10101010b
mov dptr,#100h
movx @dptr,a

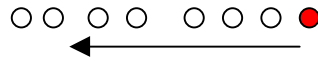
```

We can test the output logic appeared at J4 header using dot LED series with 330Ohms current limiting resistor. For single bit, we can use logic probe to test it also.

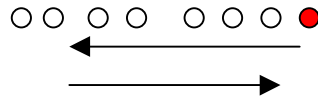


EXERCISE 7-1:

1. Write a program that shifts a bit from right-to-left. Each bit will be shifted every 100ms.



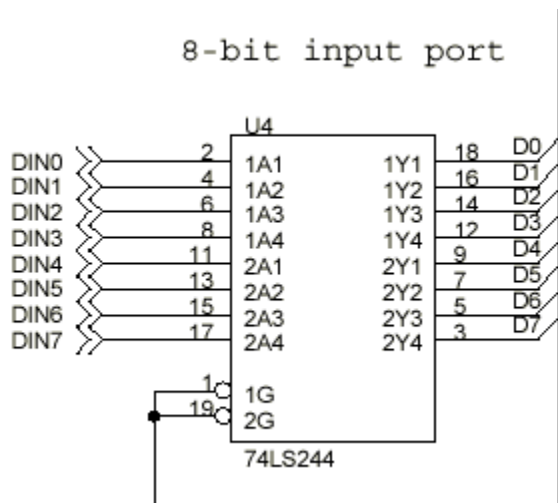
2. Similarly for 1, now shift it back and forth from right-to-left and left-to-right the same speed.



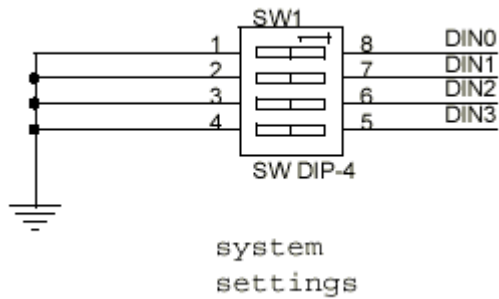
Hint: use 10ms timebase and RR and RL instructions.

GPIO2

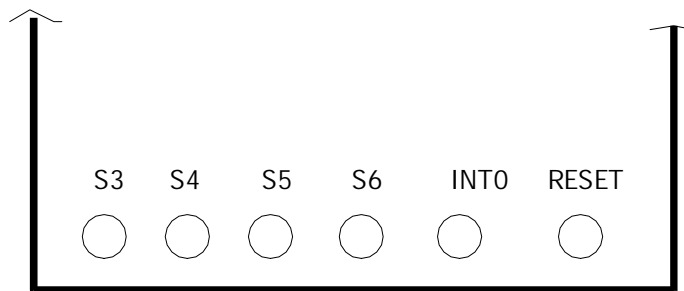
GPIO2 is 8-bit input port, built with 3-state 8-bit buffer, 74HC244. The circuit is shown below.

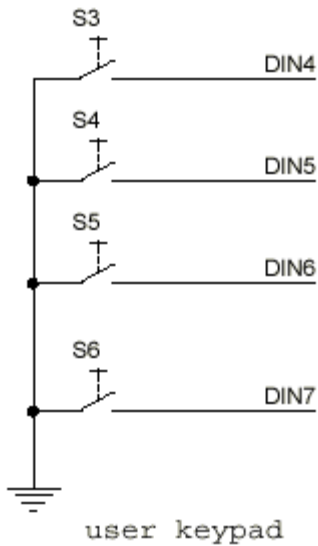


DIN0-DIN7 accepts TTL compatible logic signal. DIN0 to DIN3 are tied to DIP switch, SW1 for system setting. When each switch was closed, the logic input will be '0'. The 8051SBC uses position 1 for boot loader feature. Position1 has a dot indicator.

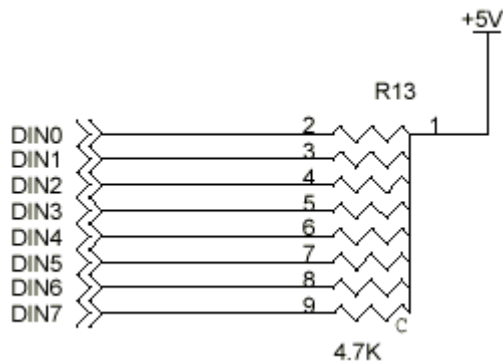


DIN4 to DIN7 are tied to user keypad, S3 to S6. We can experiment with these push buttons.





All input bits are pull-up with 4.7k to make logic input for CMOS circuit to be '1' when all switches are OPEN circuit.



The location of GPIO2 is at address 200H.

Examples

To read 8-bit data from GPIO2, we can use the code below.

```

mov dptr, #200h
movx a, @dptr

```

EXERCISE 7-2:

1. Write a program that reads 8-bit data and prints it on screen in binary format.

1111 1111

When press a given key onboard, the value should show '0'.

EXERCISE 7-3:

1. Add task below to the main loop of digital clock program from Lesson 6. Assemble the program and test run. What happen when we press S6 key?

```
;----- read key S6 every 10ms -----
read_s6:  mov dptr,#gpio2
          movx a,@dptr
          jb acc.7, exit_read_s6
          setb flag1.7    ; to tell next task
                   ; that S6 has been pressed
          ret

exit_read_s6:
          clr flag1.7
          ret

;----- service key S6 -----
execute_s6:
          jnb flag1.7,exit_execute_s6

          mov command,#'m'
          call adjust_min

exit_execute_s6:
          ret

;----- main loop running -----
main:    call wait_tick
          call update_clock
          call LED_on
          call LED_off
          call print_time
          call get_command
          call adjust_min
          call adjust_hour
          call read_s6           ; read S6 push button
          call execute_s6
          jmp main
```

We see that S6 key was read every 10ms by task read S6. And it will make minute to be updated every 10ms.

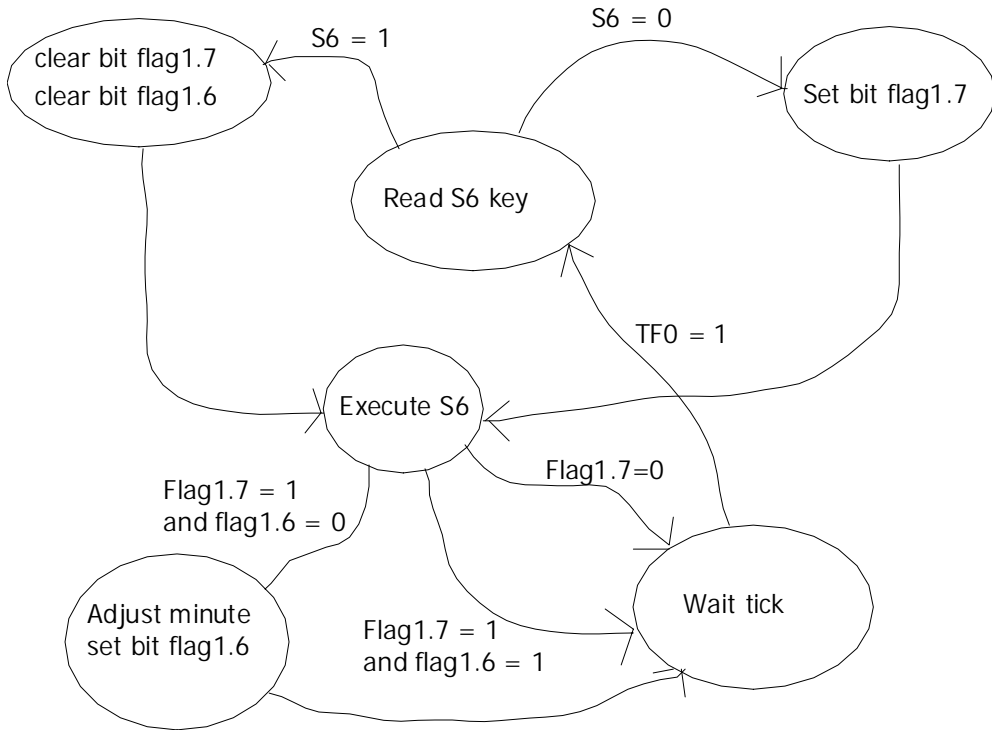
How can we stop reading S6 and repeat executing update minute while S6 has been pressed?

We will need more bit to stop repeating service key S6. Now let use see a new modification code.

```
;----- read key S6 every 10ms -----  
  
read_s6:  mov dptr,#gpio2  
          movx a,@dptr  
          jnb acc.7, exit_read_s6  
          setb flag1.7 ; to tell next task  
                ; that S6 has been pressed  
          ret  
  
exit_read_s6:  
          clr flag1.7  
          clr flag1.6  
          ret  
  
;----- service key S6 -----  
execute_s6:  
          jnb flag1.7,exit_execute_s6  
          jnb flag1.6,just_pressed  
          ret  
  
just_pressed:  
          setb flag1.6 ;to tell execute_s6 has been executed  
          mov command,#'m'  
          call adjust_min  
  
exit_execute_s6:  
          ret
```

EXERCISE 7-4:

1. Modify above tasks with flag1.6 adding. Assemble the code and test run, now let press S6 again. What does minute incrementing look like?



We can learn more easily with above state transition diagram. The tasks are shown only three tasks. The actual main loop has about 10 tasks.

EXERCISE 7-5:

1. Add more keys for hour adjusting with S5 key.
2. Draw a state transition diagram.

Sample code for Exercise 7-1

```
$mod51
#include(mypaulm2.equ)

gpio1      equ 100h
gpio2      equ 200h

        dseg at 20h

flag1:    ds 1

        dseg at 50h

tick:     ds 1
counter1: ds 1
timer3:   ds 1
temp:     ds 1

        cseg at 8000h
        jmp start

        org 8100h

;----- initialization code -----
start:   orl tmod,#1 ; set timer0 to mode1
        setb tr0
        mov flag1,#1
        mov counter1,#0
        mov temp,#1
        mov timer3,#0

;----- main loop running -----
main:    call wait_tick
        call shift_left
        call shift_right

        jmp main

;----- subroutines -----
wait_tick: jnb tf0,$
        clr tf0
        orl th0,#0dch
        ret

shift_left:
        jnb flag1.0,exit_left
        inc timer3
        mov a,timer3
        cjne a,#10,exit_left
```

```

    mov timer3,#0
    mov a,temp
    mov dptr,#100h
    movx @dptr,a

    rl a
    mov temp,a

    inc counter1

    mov a,counter1
    cjne a,#7,exit_left

    mov counter1,#0
    setb flag1.1
    clr flag1.0

exit_left:
    ret

shift_right:
    jnb flag1.1,exit_right
    inc timer3
    mov a,timer3
    cjne a,#10,exit_right

    mov timer3,#0
    mov a,temp
    mov dptr,#100h
    movx @dptr,a

    rr a
    mov temp,a

    inc counter1

    mov a,counter1
    cjne a,#7,exit_right
    mov counter1,#0
    setb flag1.0
    clr flag1.1

exit_right:
    ret

    end

```