

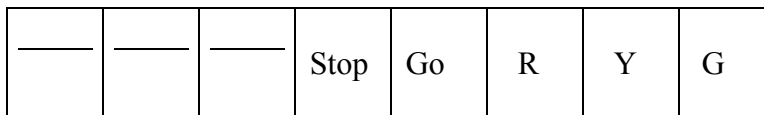
From *Embedded Software: The Works* by Colin Walls, 2006, Section 3.4

The Application: Pedestrian Crossing with a Traffic Light (Modeled on U.K. system)

1. The walker presses a button.
2. The traffic lights cycle through YELLOW to RED; the pedestrian crossing light changes from STOP to GO.
3. After a delay, the pedestrian YELLOW light and GO are set flashing.
4. After more delay, the traffic light is set back to GREEN; the pedestrian light changes to STOP.
5. The pedestrian crossing button is then inoperable for a period of time (to avoid frequent interruption of traffic).

Hardware Configuration: The traffic light controller is based on a microcontroller with 2 memory-mapped I/O ports.

1. Output port at address \$1000 controls the 5 lights; setting the bit turns on the light.



2. Input port at address \$2000 corresponds to the pedestrian button.
 - Bit 0 is set (and latched) when the button is pressed.
 - Bit 0 is cleared by writing 0 to the port.
3. Timer interrupt (every 250 ms) is set up on a vector at address \$100

Software Implementation

1. Main Loop, which waits for the pedestrian button press and sequences the lights.
2. Interrupt Service Routine to manage the time delays and lamp flashing.

Main Loop Pseudo Code

```
Initialize lights to Green/Stop;
Initialize button;
Repeat
    Wait for button to be pressed;
    Clear button register;
    Set lights to Yellow/Stop;
    Delay 2 seconds;
    Set lights to Red/Stop;
    Delay 2 seconds;
    Set lights to Red/Go;
    Delay 5 seconds;
    Set lights to flashing Yellow/Go;
    Delay 3 seconds;
    Set lights to Green/Stop;
    Delay 1 minute;
End
```

Main Loop Code

```
/* Bit patterns for lights */

#define GREEN          1    /* traffic */
#define YELLOW        2
#define RED           4
#define FLASH_YELLOW  0x200

#define GO             8    /* pedestrian */
#define STOP          0x10
#define FLASH_GO      0x800

/* Control word for lights */
volatile int lights;      /* global variable for ISR too */

/* Button read/clear definitions - bit 0 of $2000 - macros */
#define READ_BUTTON    ((* (int *)0x2000) & 1)
#define WRITE_BUTTON   (*(int *)0x2000)

/* Timing facilities */
volatile int countdown;  /* global variable for ISR too */

#define WAIT(s)        countdown=s*4; while(countdown) ; /* macro */
```

ECE 4001 / 7001 A Simple Embedded Software Example

```
main ()
{
    /* Initialization */
    lights = GREEN | STOP;          /* set lights */
    WRITE_BUTTON = 0;               /* init button */
    asm(" MOVE #$2000,SR\n");       /* enable interrupts */

    /*** Main loop ***/

    while (1) /* go round for ever */
    {
        /*** Wait for button ***/
        while (!READ_BUTTON)
            ;                       /* just hang */
        WRITE_BUTTON = 0;

        /*** Change lights to allow crossing ***/
        lights = YELLOW | STOP;
        WAIT(2);
        lights = RED | STOP;
        WAIT(2);
        lights = RED | GO;
        WAIT(5);                    /* crossing now */

        /*** Change lights back ***/
        lights = FLASH_YELLOW | FLASH_GO;
        WAIT(3);
        lights = GREEN | STOP;

        /*** Let traffic flow for a while ***/
        WAIT(60);
    }
}
```

Interrupt Service Routine – runs every 250 ms to support:

- Countdown timer for time delays in the main loop, using the global variable `countdown`.
- Output to the lamp-driving hardware, taking care of flashing; uses the global variable `lights`.

Interrupt Service Routine Code

```
/*Lights port - $1000 */
#define LIGHTS (*(int *)0x1000)

interrupt void timer() /* interrupt keyword -> context saving */
{
    int flash_bits;

    if (countdown != 0) /* timer */
        countdown--;

    flash_bits = lights >> 8; /* update lights */
    lights ^= flash_bits;
    LIGHTS = lights & 0xff;
}

/* Interrupt vector */

#pragma asm /* assembly language insert to set up vector */
    ORG $100 /* store the address of timer at $100 */
    DC.L _timer
#pragma endasm
```

Caution on Using Global Variables: The use of global variable to communicate between mainline code and ISR's is often unreliable. To use them safely, Walls suggests the following conditions:

- The application must be simple
- Variable must be declared as `volatile`
- There must be a well-defined protocol for reads and writes.