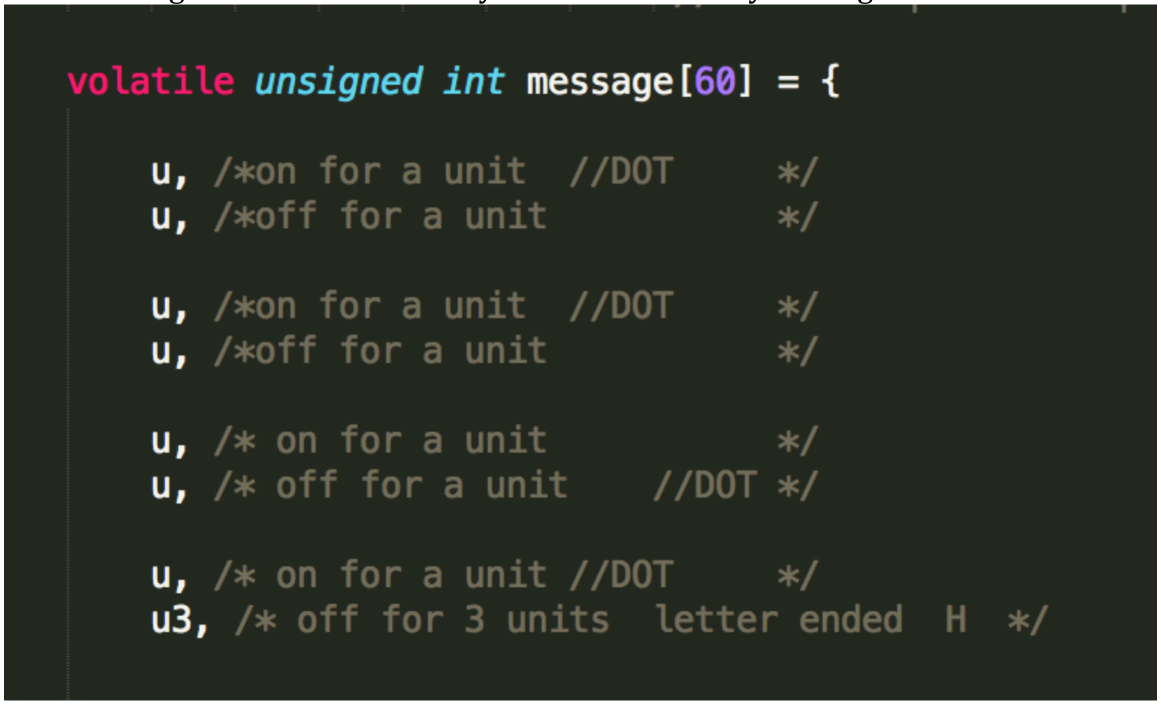
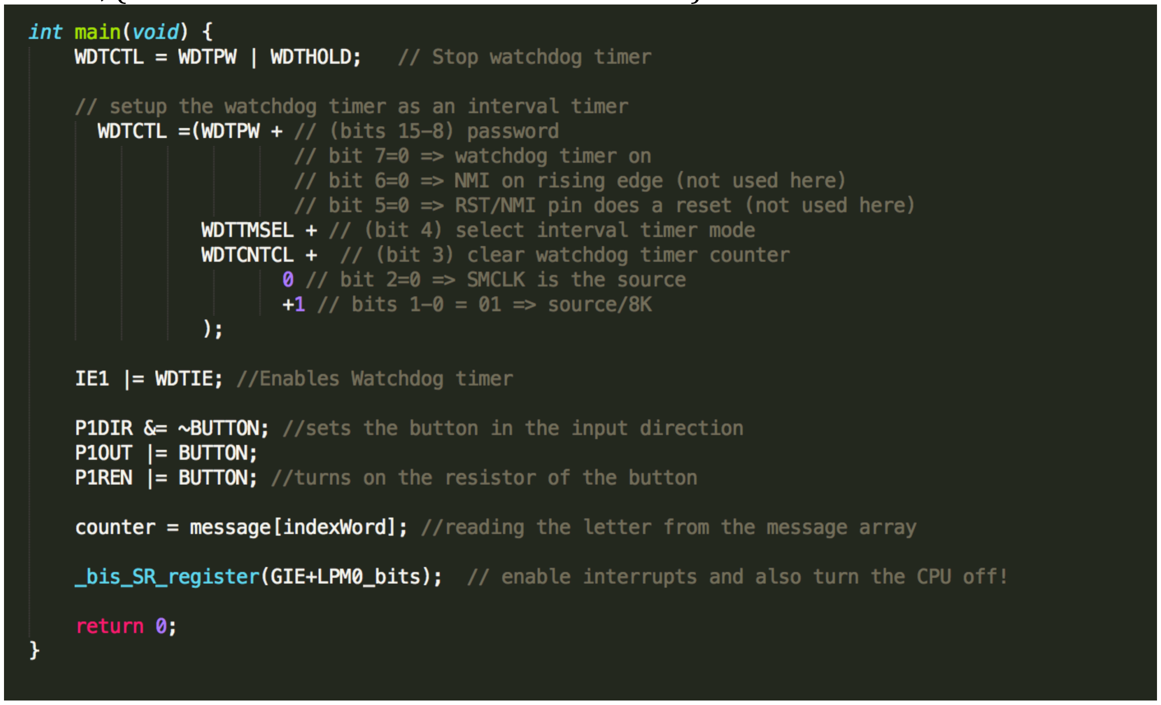
For this homework we had to encode our name in morse code and use the watch dog timer interrupt to trigger a service routine that would both allow us to blink our name and also control which LED the message came out of.

For this assignment I made an array of values that is my message.



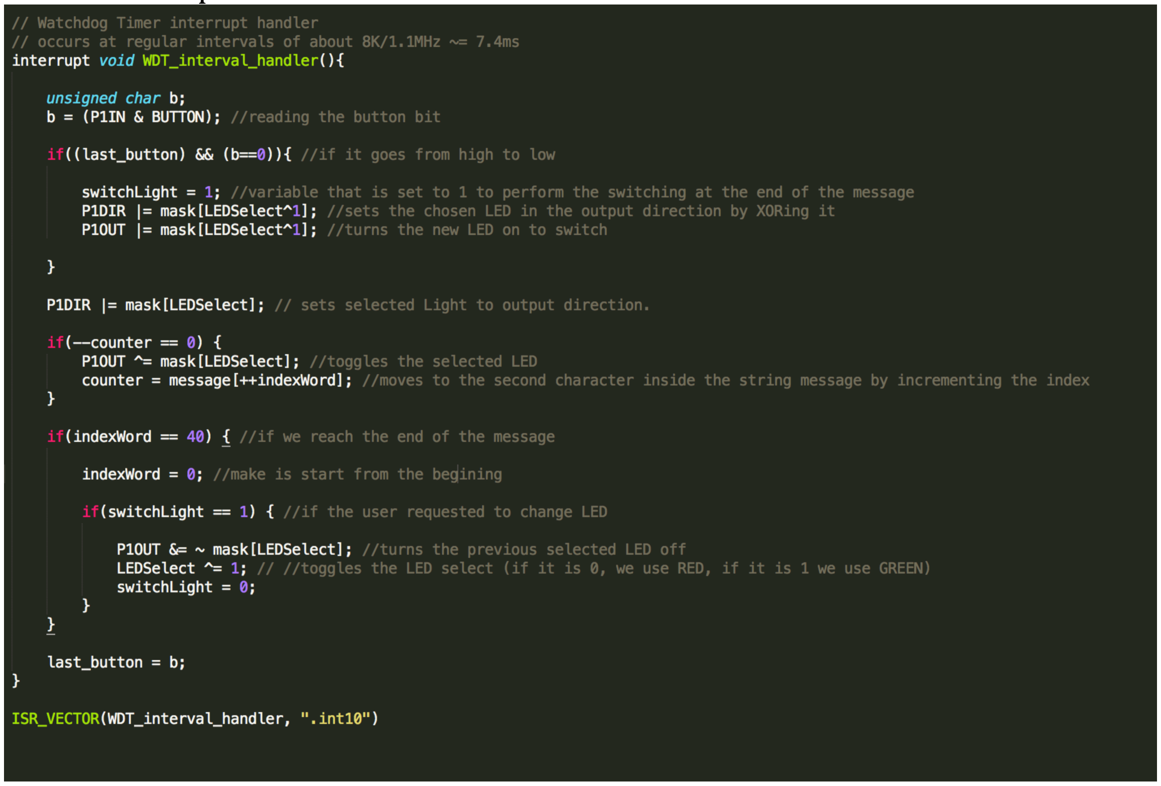
```
volatile unsigned int message[60] = {

    u, /*on for a unit  //DOT     */
    u, /*off for a unit           */

    u, /*on for a unit  //DOT     */
    u, /*off for a unit           */

    u, /* on for a unit           */
    u, /* off for a unit    //DOT */

    u, /* on for a unit //DOT     */
    u3, /* off for 3 units  letter ended  H  */
```

Like so, (more information can be seen in the code).



```
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer

    // setup the watchdog timer as an interval timer
      WDTCTL =(WDTPW + // (bits 15-8) password
                       // bit 7=0 => watchdog timer on
                       // bit 6=0 => NMI on rising edge (not used here)
                       // bit 5=0 => RST/NMI pin does a reset (not used here)
               WDTTMSEL + // (bit 4) select interval timer mode
               WDTCNTCL +  // (bit 3) clear watchdog timer counter
                      0 // bit 2=0 => SMCLK is the source
                      +1 // bits 1-0 = 01 => source/8K
               );

    IE1 |= WDTIE; //Enables Watchdog timer

    P1DIR &= ~BUTTON; //sets the button in the input direction
    P1OUT |= BUTTON;
    P1REN |= BUTTON; //turns on the resistor of the button

    counter = message[indexWord]; //reading the letter from the message array

    _bis_SR_register(GIE+LPM0_bits);  // enable interrupts and also turn the CPU off!

    return 0;
}
```

In my main function I enable the button as an input, I drove an output value of 1 to the button to know what the value was. Then I drove a 1 to the pin in the P1REN register to enable the pullup resistor on the board so that when pressed the button will have a value of 0 when pushed (pulled up to Vcc if not pressed) . Then I make my global variable 'counter' the current time delay of my array. The delays come in three flavors, there is u, u3, and u7. U is the base unit and is 30 and the other ones are u times three and u times 7. All together my delays are 30-90-210. Finally I enable the interrupt for the watchdog timer in the IE1 register. This lets the watchdog timer interrupt be able to be called. Finall, I go into a low power mode and enable interrupts so that my interrupt code can take over everytime the watchdog timer wakes up.



```
// Watchdog Timer interrupt handler
// occurs at regular intervals of about 8K/1.1MHz ~= 7.4ms
interrupt void WDT_interval_handler(){

    unsigned char b;
    b = (P1IN & BUTTON); //reading the button bit

    if((last_button) && (b==0)){ //if it goes from high to low

        switchLight = 1; //variable that is set to 1 to perform the switching at the end of the message
        P1DIR |= mask[LEDSelect^1]; //sets the chosen LED in the output direction by XORing it
        P1OUT |= mask[LEDSelect^1]; //turns the new LED on to switch

    }

    P1DIR |= mask[LEDSelect]; // sets selected Light to output direction.

    if(--counter == 0) {
        P1OUT ^= mask[LEDSelect]; //toggles the selected LED
        counter = message[++indexWord]; //moves to the second character inside the string message by incrementing the index
    }

    if(indexWord == 40) { //if we reach the end of the message

        indexWord = 0; //make is start from the begining

        if(switchLight == 1) { //if the user requested to change LED

            P1OUT &= ~ mask[LEDSelect]; //turns the previous selected LED off
            LEDSelect ^= 1; // //toggles the LED select (if it is 0, we use RED, if it is 1 we use GREEN)
            switchLight = 0;
        }
    }

    last_button = b;
}

ISR_VECTOR(WDT_interval_handler, ".int10")
```

in the actual interrupt code. I register the function WDT_interval_handler as the function pointer to call when the interrupt in the 10th register gets called.
Finally inside the code above, I advance through my message array and work out delays and pin settings so that when the button is pressed the button can readily reply to the input. The specifics of the code can be seen above and in my main.c file attached.