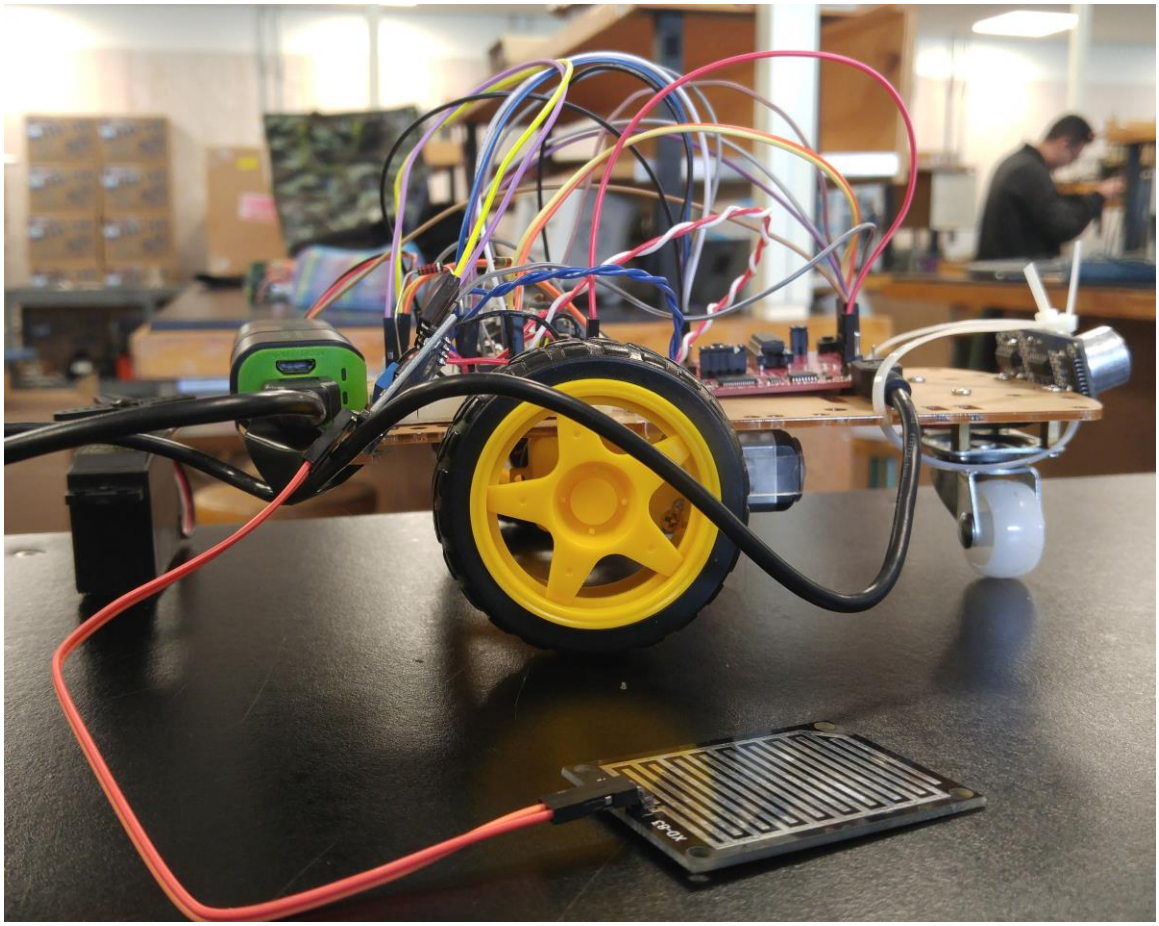
Mini Robot Car



Physics 319 Project

Anita Mahinpei
23586150

Table of Contents

Abstract

For this project, I built a mini robot car with two modes: automatic and remote. In automatic mode, the car could avoid obstacles using data from an ultrasonic distance sensor. It also had automatic windshield wipers, simulated by a servo motor, which would move at a speed proportional to the amount of water detected using a rain sensor. In the remote mode, the car could be controlled using an IR remote and receiver kit to move forward, turn right or stop. Originally, the goal was to use different buttons of the remote to make the car move. However, in the final project, the number of button presses was used to change the motion of the car.

Introduction

The motivation behind this project was to apply concepts learned during the first six labs in the course and learn to work with new electronics in order to build an interesting product. This was achieved in all portions of the project. Concepts such as working with the timer PWM and counter were used to run the servo motor for the wind shield wiper and time the data received by the ultrasonic sensor to measure distance. The UART was used for monitoring the signal from the IR receiver in order to set the appropriate thresholds in the final code. Finally, the ADC was used for receiving data from the sensors. The project also involved new concepts such as working with H-bridge, rain sensor, IR receiver and servo motor.

The project was inspired by other remote-controlled car projects and weather forecast projects that used rain and temperature sensors (reference 4 and 5).

Apparatus

The final project was built using the following electronics: Ivolador Robot Car Kit, analogue rain sensor, servo motor, IR receiver and controller kit, H-Bridge, ultrasonic sensor, 5V rechargeable battery pack and MSP430 microprocessor.

First the robot kit was assembled with the wires soldered appropriately. The H-Bridge was connected to the robot kit in order to drive the motor based on the signal from MSP430. The microprocessor was also connected to the analogue output of a rain sensor and to the signal pin of a servo motor in order to simulate the automatic windshield wipers being controlled based on the rain sensor signal. Finally, the IR receiver and the ultrasonic rain sensor were also connected to separate pins on MSP430 in order to provide data for the motion of the car in the remote and automatic modes respectively.

All the electronics were connected to 3V signal and ground from MSP430 pins. 5V power was also connected to the second VCC pin of the H-Bridge in order to drive the car motors. The diagram below shows the circuit setup in more details.

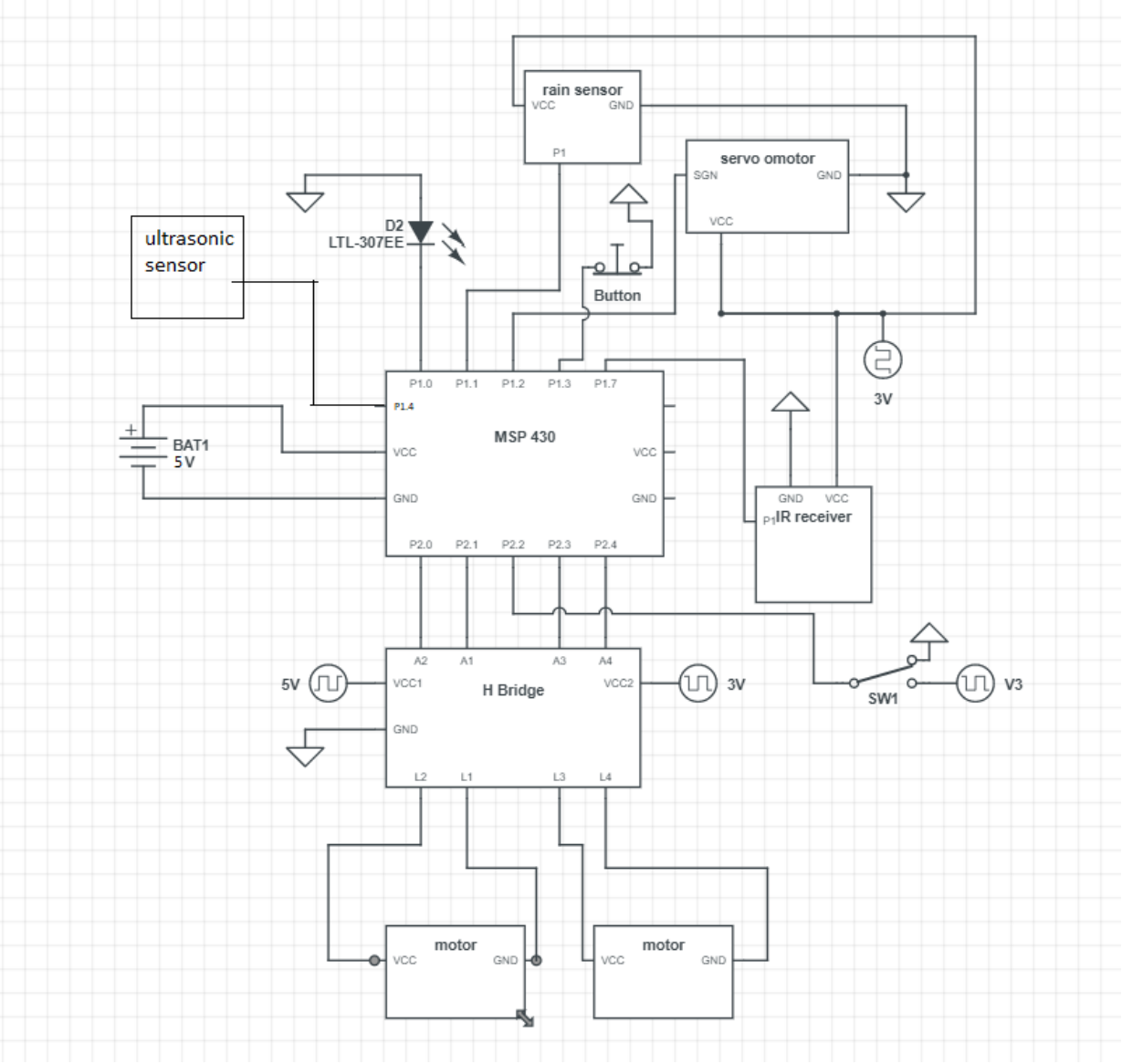


Figure 1: Diagram of the circuit setup for project

Procedure

Once powered up, the car goes in pre-application mode with the MSP430 LED's blinking. The user can select whether they want the car in automatic or remote-controlled mode by setting the P2.2 input to low or high respectively. Once the button connected to P1.3 is pushed, the device will be in the selected mode.

In the automatic mode, the car will start moving forward and receiving data from the ultrasonic sensor, which is used to measure distance. If the distance is more than 30cm, the car will keep moving forward, if it is between 15 and 30cm it will turn right, and if less than 15cm it will reverse. The last option was added to prevent the car from getting stuck in a corner for making the wrong turn. The device also receives data from the rain sensor in automatic mode. If the voltage received, drops below a given threshold, the servo motor will start moving with the delay time between each quarter-turn made by the motor, proportional to the voltage received.

In the remote-controlled mode, the device will be stationary. Once any button on the IR remote is pressed, the device will more forward, a second press of the button will make the device turn and another press will make it stop again.

The pre-application mode was setup using the code from lab 5 and 6 with the device changing its mode upon receiving a push button interrupt. The code for calculating distance from the ultrasonic sensor data was also adapted from lab 5 and 6. MSP430 would send a 15-microsecond long trigger and time the sensor signal using TA.1 counter. This time, however, instead of transmitting the data to a computer, the data was used to change the output to the H-Bridge. The H-Bridge had four pins for controlling each motor: two were connected to the motor and the others to the microprocessor. The direction of a single wheel’s rotation (forward or reverse) depended on whether pin 1 or 2 for that motor was set to high.

The servo motor moved using TA.0 PWM. The period of the cycle determined the speed and the duty cycle determined how many degrees the motor moved. The motor was set to make a 90 degree turn, return to its original position, delay and repeat. The delay time was

changed using the rain sensor voltage input read using the ADC. Finally, the IR receiver signal was used to determine the motion in remote-controlled mode. The initial 9ms drop in voltage from the receiver upon receiving a signal was used to trigger a change in the motion of the car.

Results

The windshield wiper and obstacle avoider portion of the project worked as intended. I had some difficulties with the rain sensor as the original sensor that I bought didn't work and the second one became less sensitive with use. However, I was eventually able to set the code properly and make the servo motor move based on the rain sensor data. The device was able to avoid obstacles in most cases, however if the sides of the car got stuck because of a peripheral obstacle, the device would stop moving.

I had to modify the remote-controlled aspect of the project due to difficulties reading the received signal. The IR receiver gets a signal based on the NEC protocol which is a sequence of ones and zeroes with the delay time for a one being longer than a zero. I tried decoding this signal by looking at the output using the UART, however I wasn't getting consistent data, so I decided to use the number of times a button was pressed to determine the motion of the car instead.

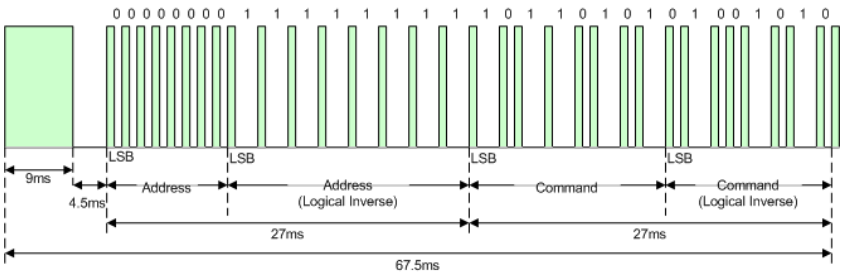


Figure 2: example IR signal in NEC protocol

Discussion

This project could be improved by adding another timing option, that measures the amount of time the car is stuck in one position for changing the car's direction of motion when it gets stuck to a peripheral obstacle that can't be detected using the ultrasonic sensor that is mounted in the front.

The project could also be improved by using the timer compare-and-capture interrupt mode rather than using the counter without interrupts in order to avoid wasting processor cycles on checking for readings. This could also potentially help with taking more reliable and consistent data from the IR receiver for decoding.

There is also the possibility of expanding the project to include more functionalities such as making the ultrasonic sensor turn to check for obstacles in multiple directions and make a decision about which way it should move depending on the data received rather than always turning in the same direction.

Conclusion

For this project, I was able to build a mini robot car with an automatic windshield wiper that could avoid obstacles and be controlled remotely with and IR controller. I was able to achieve my goals with a few alterations. The windshield wiper and obstacle avoiding aspects of the project performed as expected, but in order to make the car remote controlled I made it change direction based on the number of times a button was pressed.

I was able to apply the knowledge obtained from labs about PWM, ADC, and ultrasonic sensors and learn about new electronics such as: servo motors, rain sensors, IR receivers, DC motors and H-Bridges.

Reference

1) Altuim. (n.d.). NEC Infrared Transmission Protocol. Retrieved March 16, 2018, from http://techdocs.altium.com/display/FPGA/NEC Infrared Transmission Protocol
2) MSP430 Launchpad: Controlling a Servo with hardware PWM. (n.d.). Retrieved March 08, 2018, from http://gushh.net/blog/msp430-servo/
3) Richardson, M. (n.d.). Lab: DC Motor Control Using an H-Bridge. Retrieved March 08, 2018, from https://itp.nyu.edu/physcomp/labs/motors-and-transistors/dc-motor-control-using-an-h-bridge/
4) Instructables. (2017, October 03). Building a Robot Using MSP430 Launchpad. Retrieved February 27, 2018, from http://www.instructables.com/id/Building-a-Robot-using-MSP430-Launchpad/
5) I. (2017, October 13). Arduino Modules - Rain Sensor. Retrieved February 27, 2018, from http://www.instructables.com/id/Arduino-Modules-Rain-Sensor/

Appendix

```
/*
 * Mini Robot Car Code adapted from MSP-EXP430G2-LaunchPad User
Experience Application
 * for UART control
 * main.c
*/


#include  "msp430.h"

#define     LED1                  BIT0
#define     LED2                  BIT6
#define     BUTTON                BIT3
#define     PreAppMode            0
#define     RunningMode           1

// Servo motor definitions
#define MCU_CLOCK           1100000
#define PWM_FREQUENCY       50      // In Hertz
#define SERVO_STEPS         180     // Maximum amount of steps
#define SERVO_MIN           700     // The minimum duty cycle
#define SERVO_MAX           2000    // The maximum duty cycle


volatile unsigned int Mode;

unsigned int PWM_Period     = (MCU_CLOCK / PWM_FREQUENCY);
unsigned int PWM_Duty       = 0;
unsigned int delayTime            = 400; //delay for the windshield wiper


void InitializeButton(void);
void PreApplicationMode(void);

void main(void)
{
  float distance;
  int count=0;
  unsigned int servo_stepval, servo_stepnow;
  unsigned int servo_lut[ SERVO_STEPS+1 ];
  servo_stepval   = ( (SERVO_MAX - SERVO_MIN) / SERVO_STEPS );
  servo_stepnow   = SERVO_MIN;
  unsigned int i;

  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  /* next three lines to use internal calibrated 1MHz clock: */
  BCSCTL1 = CALBC1_1MHZ;                    // Set range
```

```
  DCOCTL = CALDCO_1MHZ;
  BCSCTL2 &= ~(DIVS_3);                         // SMCLK = DCO = 1MHz

  InitializeButton();

  // setup port for leds:
  P1DIR |= LED1 + LED2;
  P1OUT &= ~(LED1 + LED2);
  P1DIR |= BIT2;
  P1OUT |= BIT2;

  Mode = PreAppMode;
  PreApplicationMode();          // Blinks LEDs, waits for button press
  __enable_interrupt();          // Enable interrupts.



  // Fill up the LUT
  for (i = 0; i < SERVO_STEPS; i++) {
     servo_stepnow += servo_stepval;
     servo_lut[i] = servo_stepnow;
  }

  // TimerA.0 setup for PWM to use with the windshield wiper
  TA0CCTL1 = OUTMOD_7;            // TACCR1 reset/set
  TA0CTL   = TASSEL_2 + MC_1;     // SMCLK, upmode
  TA0CCR0  = PWM_Period-1;        // PWM Period
  TA0CCR1  = PWM_Duty;

  //setup the ADC
  ADC10CTL0 = ADC10SHT_2 + ADC10ON;
  ADC10CTL1 = INCH_1;
  ADC10AE0 |= 0x02; //p1.1 is ADC input read for rain sensor

  P1DIR   |= BIT2;          // P1.2 is servo motor output
  P1DIR    &= ~BIT1;                          // P1.1 is rain sensor
input
  P1SEL   |= BIT2;          // P1.2 = TA1 output
  P1DIR |= BIT0;
  P1DIR |= BIT6;
  P1OUT &= ~BIT0;           //turns off when wiper is on
  P1OUT &= ~BIT6;


  // Timer A.1 setup for using counter
  TA1CTL = TASSEL_2 | MC_2 | ID_0;

  // Setup P2 pins to send signal to H-Bridge and run the motor
  // pin 2 is used to determine whether it is automatic (low) or remote
(high)
  P2DIR = 0b00011011;
  P2OUT = 0x0;
  P1DIR |= BIT4;
```

```
  P1DIR &= ~BIT5;
  P1DIR &= ~BIT7;


  /* Main Application Loop */
  while(1)
  {

     if(!(P2IN & BIT2)){//automatic mode
    //ADC reading
          ADC10CTL0 |= ENC + ADC10SC;
          while (ADC10CTL1 &ADC10BUSY);
    // compute the time using ultrasonic sensor data
          delayTime  =
400*(((float)ADC10MEM)/((float)0x3a0))*(((float)ADC10MEM)/((float)0x3a0))
;

          if(ADC10MEM<0x3a0){ //if sensed rain
               P1OUT &= ~BIT0;

      // move the wiper forward
               for (i = 0; i < SERVO_STEPS; i++) {
                    TA0CCR1 = servo_lut[i];
                    __delay_cycles(2000);
               }

               // Move wiper backward
               for (i = SERVO_STEPS; i > 0; i--) {
                    TA0CCR1 = servo_lut[i];
                    __delay_cycles(2000);
               }

               //delay
               for(i =0; i<delayTime; i++){
                    __delay_cycles(5000);
               }
          }else{ //didn't sense rain
               P1OUT|=BIT0;
          }

      // send signal to the ultrasonic sensor
     P1OUT|=BIT4;
     __delay_cycles(10);
     P1OUT&= ~BIT4;

      // measure time of signal sent by sensor
     while(!(P1IN&BIT5));
     TA1R =0;
     while((P1IN&BIT5));
     distance=((float) TA1R)/((float)58.0);

          if(distance <=30 && distance >10){//turn right
               P2OUT = 0b00000010;
```

```
            }else if(distance <=15){//reverse
                  P2OUT = 0b00001001;
            }else{//move forward
                  P2OUT = 0b00010010;
            }
            __delay_cycles(60000);

    }else{//remote control mode

            while(P1IN&BIT7);//IR signal not received
            count = (count +1)%3;
            switch (count) {
                  case 1:
                        P2OUT = 0b00010010;//move forward
                        break;
                  case 2:
                        P2OUT = 0b00000010;//turn
                        break;
                  case 0:
                        P2OUT = 0b00000000;//stop
                        break;
            }
            for(int i=0; i<100; i++){
                  __delay_cycles(20000);
            }


     }


  }
}

void PreApplicationMode(void)
{
  P1DIR |= LED1 + LED2;
  P1OUT |= LED1;                  // To enable the LED toggling effect
  P1OUT &= ~LED2;

  /* these next two lines configure the ACLK signal to come from
     a secondary oscillator source, called VLO */

  BCSCTL1 |= DIVA_1;              // ACLK is half the speed of the source
(VLO)
  BCSCTL3 |= LFXT1S_2;            // ACLK = VLO

  /* here we're setting up a timer to fire an interrupt periodically.
     When the timer 1 hits its limit, the interrupt will toggle the
lights

     We're using ACLK as the timer source, since it lets us go into LPM3
     (where SMCLK and MCLK are turned off). */
```

```
  TACCR0 = 1200;                  //  period
  TACTL = TASSEL_1 | MC_1;        // TACLK = ACLK, Up mode.
  TACCTL1 = CCIE + OUTMOD_3;      // TACCTL1 Capture Compare
  TACCR1 = 600;                   // duty cycle
  __bis_SR_register(LPM3_bits + GIE);   // LPM3 with interrupts enabled
  // in LPM3, MCLCK and SMCLK are off, but ACLK is on.
}

// this gets used in pre-application mode only to toggle the lights:
#if defined(__TI_COMPILER_VERSION__)
#pragma vector=TIMER0_A1_VECTOR
__interrupt void ta1_isr (void)
#else
  void __attribute__ ((interrupt(TIMER0_A1_VECTOR))) ta1_isr (void)
#endif
{
  TACCTL1 &= ~CCIFG; // reset the interrupt flag
  if (Mode == PreAppMode){
    P1OUT ^= (LED1 + LED2); // toggle the two lights.
  }
  else{
    TACCTL1 = 0;                 // no more interrupts.
    __bic_SR_register_on_exit(LPM3_bits);        // Restart the cpu
  }

}

void InitializeButton(void)                 // Configure Push Button
{
  P1DIR &= ~BUTTON;
  P1OUT |= BUTTON;
  P1REN |= BUTTON;
  P1IES |= BUTTON;
  P1IFG &= ~BUTTON;
  P1IE |= BUTTON;
}

/* ************************************************************
 * Port Interrupt for Button Press
 * 1. During standby mode: to enter application mode
 *
 * ************************************************************ */

#if defined(__TI_COMPILER_VERSION__)
#pragma vector=PORT1_VECTOR
__interrupt void port1_isr(void)
#else
  void __attribute__ ((interrupt(PORT1_VECTOR))) port1_isr (void)
#endif
{

  /* this disables port1 interrupts for a little while so that
```

```
  we don't try to respond to two consecutive button pushes right
together.
  The watchdog timer interrupt will re-enable port1 interrupts

  This whole watchdog thing is completely unnecessary here, but its
useful
  to see how it is done.
*/
  P1IFG = 0;  // clear out interrupt flag
  P1IE &= ~BUTTON;          // Disable port 1 interrupts
  WDTCTL = WDT_ADLY_250;    // set up watchdog timer duration
  IFG1 &= ~WDTIFG;          // clear interrupt flag
  IE1 |= WDTIE;             // enable watchdog interrupts

  TACCTL1 = 0;              // turn off timer 1 interrupts
  P1OUT &= ~(LED1+LED2);    // turn off the leds
  Mode = RunningMode;
  __bic_SR_register_on_exit(LPM3_bits); // take us out of low power mode

}

// WDT Interrupt Service Routine used to de-bounce button press
#if defined(__TI_COMPILER_VERSION__)
#pragma vector=WDT_VECTOR
__interrupt void wdt_isr(void)
#else
  void __attribute__ ((interrupt(WDT_VECTOR))) wdt_isr (void)
#endif
{
    IE1 &= ~WDTIE;                    /* disable watchdog interrupt */
    IFG1 &= ~WDTIFG;                  /* clear interrupt flag */
    WDTCTL = WDTPW + WDTHOLD;         /* put WDT back in hold state */
    P1IE |= BUTTON;             /* Debouncing complete - reenable port 1
interrupts*/
}
```