SourceBoost C Code

Processor: PIC16F688

Filename: nichrome_wire_controller.c

Author: Bruce Rahn

Date: 19 Jan 2010

Version: 1.2

Purpose:

To control the "burn" of up to two independent nichrome wires based on an external command (trigger).
To provide "feedback" that the command has been received and acted upon.

Software Revision History:

Ver 1.0 12 July 2009
Orig Code based on the code developed for the prototype controller.

Ver 1.1 15 Dec 2009
Changes for the 900MHz CCT Experiment. Removed automatic trigger of device 2 from the code. TLM status bit is now real time...only shows when device is actually firing, not that it has fired.

Ver 1.2 19 Jan 2010
Due to issues with the nichrome wire failing to burn through the team has decided to increase the current going through the wire. This code has been modified to turn on both devices at the same time...hence acting as a single channel device allowing twice the current to safely flow to the nichrome wire. This function will be implemented in the program 2 jumper setting. The burn time has also been increased to 60 seconds.

Hardware: PIC 16F688

-------------

+5V  VDD  |  1  14| VSS  Ground
Trigger 1 In RA5  |  2  13| RA0  Programming
Trigger 2 In RA4  |  3  12| RA1  Programming
Programming RA3  |  4  11| RA2  Unused
TLM 1 Output RC5  |  5  10| RC0  Device 1 Fire Control
TLM 2 Output RC4  |  6  9| RC1  Device 2 Fire Control
Program Select RC3  |  7  8| RC2  Heartbeat LED
-------------

Hardware Revision History:

Ver 1.0 May 2009
PCB version

******************************************************************************

#include <system.h>
#include "nichrome_wire_controller.h"
#include <eeprom.h>

// Set the 16F688 device configuration bits
// Internal 8.0Mhz Crystal Oscillator is used.
#pragma DATA _CONFIG, _FCMEN_OFF & _BOD_OFF & _IESO_OFF & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON & _INTOSCIO
// 8.0 Mhz
#pragma CLOCK_FREQ 8000000

// Declare some logic definitions for later use
#define FALSE 0
#define TRUE 1
#define ON 0
#define OFF 1
#define FIRE 1
# define SAFE 0
# define INT_EEPROM_TRIGGER_1_COUNTER 0
# define INT_EEPROM_TRIGGER_2_COUNTER 1

unsigned short delay_counter;
volatile bit done;
char trigger_counter_1;
char trigger_counter_2;
unsigned short timer_device_1;
unsigned short timer_device_2;
volatile bit monitor_output @ PORTC . 2;
volatile bit trigger_device_1 @ PORTA . 5;
volatile bit trigger_device_2 @ PORTA . 4;
volatile bit device_1_fired_tlm @ PORTC . 5;
volatile bit device_2_fired_tlm @ PORTC . 4;
volatile bit fire_status_1;
volatile bit fire_status_2;
volatile bit program_select @ PORTC . 3;
volatile bit fire_device_1 @ PORTC . 0;
volatile bit fire_device_2 @ PORTC . 1;
volatile bit done_1;
volatile bit done_2;
volatile bit valid_input_device_1;
volatile bit valid_input_device_2;
unsigned short failsafe_timer;
volatile bit diagnostics_run;

void main()
{
    delay_counter=0;
    trigger_counter_1 = 0;
    trigger_counter_2 = 0;
    device_1_fired_tlm = OFF;
    device_2_fired_tlm = OFF;
    fire_status_1 = OFF;
    fire_status_2 = OFF;
    fire_device_1 = SAFE;
    fire_device_2 = SAFE;
    valid_input_device_1 = 0;
    valid_input_device_2 = 0;
    done_1 = 0;
    done_2 = 0;
    failsafe_timer = 0;
    diagnostics_run = FALSE;
    init();
    if(program_select)
    {
        program_1();
    }
    else
    {
        program_2();
    }
    while(done)
    {
        fire_device_1 = SAFE; // loop forever and ensure both outputs are in the off state.
        fire_device_2 = SAFE;
    }
} // End main()

void interrupt(void)
{
    if (test_bit(intcon, T0IF)) // timer0 interrupt
    {
        // Timer0 is used to generate 2 ms interrupts for timing
        // Reset the timer
```c
#include <avr/tim.h>
#include <avr/interrupt.h>

#define TIMER0_INIT

#define TMRO=0

#define TMR0_IF 0

#define INT_EEPROM_TRIGGER_1_COUNTER

#define INT_EEPROM_TRIGGER_2_COUNTER

#define OPTION_REG 0

#define CMCON0

#define WPUA

#define TRISA

#define TRISC

#define I OCA

#define MONITOR_OUTPUT

#define DELAY_MS

#define FIRE

#define SAFE

#define FALSE

#define TRUE

#define NULL

#define _H_ 0

#define _S_ 0

#define _P_ 0

#define _E_ 0

#define _F_ 0

#define _G_ 0

#define _I_ 0

#define _C_ 0

#define _A_ 0

#define _L_ 0

#define _R_ 0

#define _O_ 0

#define _N_ 0

#define _T_ 0

#define _H_ 0

#define _S_ 0

#define _P_ 0

#define _E_ 0

#define _F_ 0

#define _G_ 0

#define _I_ 0

#define _C_ 0

#define _A_ 0

#define _L_ 0

#define _R_ 0

#define _O_ 0

#define _N_ 0

#define _T_ 0

#define _H_ 0

#define _S_ 0

#define _P_ 0

#define _E_ 0

#define _F_ 0

#define _G_ 0

#define _I_ 0

#define _C_ 0

#define _A_ 0

#define _L_ 0

#define _R_ 0

#define _O_ 0

#define _N_ 0

#define _T_ 0

#define _H_ 0

#define _S_ 0

#define _P_ 0

#define _E_ 0

#define _F_ 0

#define _G_ 0

#define _I_ 0

#define _C_ 0

#define _A_ 0

#define _L_ 0

#define _R_ 0

#define _O_ 0

#define _N_ 0

#define _T_ 0

#define _H_ 0

#define _S_ 0

#define _P_ 0

#define _E_ 0

#define _F_ 0

#define _G_ 0

```
trigger_counter_1 = 0;
trigger_counter_2 = 0;
eeprom_write(INT_EEPROM_TRIGGER_1_COUNTER, trigger_counter_1);
eeprom_write(INT_EEPROM_TRIGGER_2_COUNTER, trigger_counter_2);
while (!trigger_device_1 || (!trigger_device_2))
{
    for ( delay_counter = 0; delay_counter < 5; delay_counter++)
    {
        monitor_output = ON;
        delay_ms(100);
        monitor_output = OFF;
        delay_ms(250);
    }
    diagnostics_run = FALSE;
}
delay_counter = 0;
tmr0 = TIMER0_INIT;
set_bit(intcon, T0IE); // enable timer interrupt
set_bit(intcon, GIE); // enable global interrupt

void program_1(void)
{
    while(!done)
    {
        if((timer_device_1 >= 500) && (!valid_input_device_1) && (trigger_device_1 == 0) &&
           (fire_device_2 == SAFE))
            // 500 counts equals 1 second (500 x 2.0 ms). To trigger, the input must be held low for at
            // least 1 second.
            {
                valid_input_device_1 = 1;
                timer_device_1 = 0;
                trigger_counter_1 = trigger_counter_1 + 1;
                fire_device_1 = FIRE; // fire device 1
                fire_status_1 = ON;
            }
        if((timer_device_1 == 15000) && (valid_input_device_1 == 1) && (done_1 == 0))
            // 15000 counts equals 30 seconds. The wire will burn for 30 seconds and shut off.
            {
                fire_device_1 = SAFE; // turn off device 1
                timer_device_1 = 0;
                done_1 = TRUE; // device 1 has been fired, we are done with this one!
                fire_status_1 = OFF;
            }
        if((timer_device_2 >= 500) && (!valid_input_device_2) && (trigger_device_2 == 0) &&
           (fire_device_1 == SAFE))
            // 500 counts equals 1 second (500 x 2.0 ms). To trigger, the input must be held low for at
            // least 1 second.
            {
                valid_input_device_2 = 1;
                timer_device_2 = 0;
                failsafe_timer = 0;
                trigger_counter_2 = trigger_counter_2 + 1;
                fire_device_2 = FIRE; // fire device 2
                fire_status_2 = ON;
            }
        if((timer_device_2 == 15000) && valid_input_device_2 && !done_2)
            // 15000 counts = 30 seconds. The wire will burn for 30 seconds before turning off.
            {
                fire_device_2 = SAFE; // turn off device 2
                timer_device_2 = 0;
                done_2 = TRUE; // device 2 has been fired, we are done with this one!
                fire_status_2 = OFF;
            }
        if(done_1 && done_2)
        {
            
        }
}
done = TRUE;
clear_bit(intcon, GIE);  // disable global interrupt
eeprom_write(INT_EEPROM_TRIGGER_1_COUNTER, trigger_counter_1);
eeprom_write(INT_EEPROM_TRIGGER_2_COUNTER, trigger_counter_2);
}

if ((trigger_device_1) && (!valid_input_device_1))
{
    timer_device_1 = 0;
}

if ((trigger_device_2) && (!valid_input_device_2))
{
    timer_device_2 = 0;
}

device_1_fired_tlm = fire_status_1;
device_2_fired_tlm = fire_status_2;

void program_2(void)
{
    while (!done)
    {
        if((timer_device_1 >= 500) && (!valid_input_device_1) && (trigger_device_1 == 0) && (fire_device_2 == SAFE))
            // 500 counts equals 1 second (500 x 2.0 ms). To trigger, the input must be held low for at least 1 second.
            
            valid_input_device_1 = 1;
timer_device_1 = 0;
trigger_counter_1 = trigger_counter_1 + 1;
portc |= 0b00000011;  //Fire both sides of the FET
fire_status_1 = ON;
fire_status_2 = ON;

    }

if((timer_device_1 == 30000) && (valid_input_device_1 == 1) && (done_1 == 0))
    // 30000 counts equals 60 seconds. The wire will burn for 60 seconds and shut off.
    
        portc &= 0b11111100;  //Safe (turn off) both sides of the FET
timer_device_1 = 0;
done_1 = TRUE;  // device 1 has been fired, we are done with this one!
done_2 = TRUE;  // We have fired both sides of the FET with this code.
fire_status_1 = OFF;
fire_status_2 = OFF;

    }

if(done_1 && done_2)
{
    done = TRUE;
clear_bit(intcon, GIE);  // disable global interrupt
eeprom_write(INT_EEPROM_TRIGGER_1_COUNTER, trigger_counter_1);
eeprom_write(INT_EEPROM_TRIGGER_2_COUNTER, trigger_counter_2);
}

if ((trigger_device_1) && (!valid_input_device_1))
{
    timer_device_1 = 0;
}

device_1_fired_tlm = fire_status_1;
device_2_fired_tlm = fire_status_2;
}