

```

/*****
/* Program      :   RCSWITCH.C                               */
/* Function     :   R/C Power Switch (SMT Version)           */
/* Author      :   Engr. John F. Fitter B.E.                 */
/* Language    :   HiTech C                                  */
/* Platform    :   PIC 12C508 or 12C509                      */
/* Revisions   :   28jun98 Original                          */
/*                                                     */
/* Copyright c 1998 Eagle Air Australia Pty. Ltd. All rights reserved */
/*****

// This program takes an input signal and switches a protected FET according to the pulse
// width of the input signal. The input signals is a pulse train repeated at 20 to 25ms
// intervals. Each pulse is nominally 1.5ms +/- 0.5ms.

// The Pulse is represented in the program by a signed integer representing the pulse width
// deviation from the neutral pulse width in processor clock cycles (fosc/4) and to a
// resolution of 5uS.

// A DIL4 configuration switch is used for field alteration of operating parameters.
// Each switch is wired as normally open / close to ground.

// Switch      On                                Off

// 1           FET turns off at trip point        FET turns on at trip point
// 2           Two trip points                    One trip point
// 3,4         FET trip point
//            1,2,3 or 4 fifths of
//            maximum pulse width deviation

#define      _RCSWITCH_C
#include     <stdlib.h>
#include     <stdio.h>
#include     <pic.h>
#include     "rcswitch.h"

__CONFIG(FOSC1 | WDTE | CP);

/*****
/* Main program.                                           */
/*****

main() {

    // Initialize chip and get switch states. The switches are read at program startup only.
    // Since GP4 and GP5 do not have weak pullups then they are read by applying a logic high
    // and reading the pin status. If the switch is closed the result will be low else it
    // will be a logic high. This is dangerous and must be performed very quickly.
    OPTION = 0x8a; // WDT 1:4, pullups enabled
    GPIO = 0x30; // set latches, GP4,5 high, others low
    TRIS = 0xcf; // set GPIO dir's - GP4,5 output
    switches.sw_byte = GPIO; // read the port into switch structure
    GPIO = 0; // reset the latches
    TRIS = 0xfb; // make GP2 output, others inputs
    OPTION = 0xca; // WDT 1:4, pullups disabled
    pswt_ctr = PWRCNT; // initialize the switch counter

    switches.sw_byte = ~switches.sw_byte; // The switches have negative logic due
    // to the pullups so invert the logic.

    // Set the trip points based on the switch settings.
    sw_point_l = (PW_CTR-PW_DEVN+(PW_DEVN*2)/5)/5;
    sw_point_h = (PW_CTR+PW_DEVN-(PW_DEVN*2)/5)/5;
    if(switches.sw_bit.trp0_s) {
        sw_point_l += (PW_DEVN*2)/25;
        sw_point_h -= (PW_DEVN*2)/25;
    }
    if(switches.sw_bit.trp1_s) {
        sw_point_l += (PW_DEVN*4)/25;
        sw_point_h -= (PW_DEVN*4)/25;
    }

#ifdef _DEVELOPMENT_
    OSCCAL = 0x60; // set the oscillator calibration
#endif

    while(true) { // main loop runs every input pulse

```

```

clrwdt(); // keep the dog tied up
if(pswt_ctr) --pswt_ctr; // decrement frame counter, stop on 0

// Get the input pulse width in 5uS increments. No signal will result in a
// watchdog timeout which is ok and will not turn the power switch on.
ch_in = 0;
asm("wait1 btfsc 6,3"); // ensure signal input is low
asm("\tgoto wait1");
asm("wait2 btfss 6,3"); // wait for signal to go high
asm("\tgoto wait2");
asm("t_pulse btfss 6,3"); // count 5uS increments
asm("\tgoto end_tpulse");
asm("\tincfsz _ch_in");
asm("\tgoto t_pulse");
asm("\tincf (_ch_in+1)");
asm("\tgoto t_pulse");
asm("end_tpulse");

// Process the input pulse.
temp_pswt = false;

// Tri-state switch ??
if(switches.sw_bit.swtyp_s) {
    if(ch_in > sw_point_l) {
        if(ch_in < sw_point_h) temp_pswt = true;
    }else if(ch_in < sw_point_h) {
        if(ch_in > sw_point_l) temp_pswt = true;
    }else temp_pswt = true;
}

// Only a 2 state switch.
}else if(ch_in > sw_point_l) temp_pswt = true;

// Ensure that the new switch state is steady for PWRcnt frames, ie. the new switch
// state is the same as the last switch state and the frame counter is zero.
if(temp_pswt == old_pswt) {
    if(!pswt_ctr) {

        // Set the switch output based on the selected logic sense.
        if(switches.sw_bit.logic_s) pswt = temp_pswt;
        else pswt = !temp_pswt;

        // Reset the frame counter.
        pswt_ctr = PWRcnt;
    }
}

// New switch state is different to the last switch state so save it
// and reset the frame counter.
}else {
    old_pswt = temp_pswt;
    pswt_ctr = PWRcnt;
}
}
}

// ***** EOF RCTSWITCH.C *****

```