Touch switch with PIC16F627A

time = 0; TRISB4 = 0; RB4 = 1; wait(); TRISB4 = 1; while (RB4==1) { time++; }

// CONFIG

#pragma config FOSC = INTOSCIO // Oscillator Selection bits (INTOSC oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config MCLRE = OFF // RA5/MCLR/VPP Pin Function Select bit (RA5/MCLR/VPP pin function is digital input, MCLR internally tied to VDD)

#pragma config BOREN = ON // Brown-out Detect Enable bit (BOD enabled)

#pragma config LVP = OFF // Low-Voltage Programming Enable bit (RB4/PGM pin has digital I/O function, HV on MCLR must be used for programming)

#pragma config CPD = OFF // Data EE Memory Code Protection bit (Data memory code protection off)

#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)

#include <xc.h>

// where are our LEDs?

#define LED1 RB0

#define LED2 RB1

#define LED3 RB2

#define LED4 RB3

// where are our buttons?

#define SW1 RB7

#define SW2 RB6

#define SW3 RB5

#define SW4 RB4

// shortcuts for tristate registers of the buttons

#define TRIS1 TRISB7

#define TRIS2 TRISB6

#define TRIS3 TRISB5

#define TRIS4 TRISB4

// auxiliary function, see below

void wait (void);

// variables

int time;

int avg1, avg2, avg3, avg4;

char thr1, thr2, thr3, thr4;

int buf1, buf2, buf3, buf4;

// you can adjust the PWM brightness levels here

char pwm\_values[] = {0, 20, 100, 255};

char pwm = 0, pwmlevels = 4;

// main function

void main (void) {

}

// set the entire PORTB as an output on startup

TRISB = 0;

// calibrate all buttons

// (find average discharge times)

for (char c=0; c<16; c++) {

avg1=0; time=0;

TRIS1 = 0; SW1 = 1; wait(); TRIS1 = 1; while (SW1==1) { time++; }

avg1 += time / 16;

avg2=0; time=0;

TRIS2 = 0; SW2 = 1; wait(); TRIS2 = 1; while (SW2==1) { time++; }

avg2 += time / 16;

avg3=0; time=0;

TRIS3 = 0; SW3 = 1; wait(); TRIS3 = 1; while (SW3==1) { time++; }

avg3 += time / 16;

avg4=0; time=0;

TRIS4 = 0; SW4 = 1; wait(); TRIS4 = 1; while (SW4==1) { time++; }

avg4 += time / 16;

}

// set threshold values (play around with these values to make it work for you)

thr1 = 15; thr2 = 15; thr3 = 15; thr4 = 15;

// configure PWM module, frequency: 3.9kHz, resolution: 10bit

// (more details at www.friendlywire.com/tutorials/pwm)

// PWM resolution: 10 bit

PR2 = 0xff;

// turn on timer 2 and set its prescaler to 1:1

T2CON = 0b100;

// turn on PWM module

CCP1CON = 0b1100;

// set startup duty cycle value to first entry in our custom PWM array

CCPR1L = pwm\_values[pwm];

// main loop

while (1) {

}

// momentary mode

time = 0; TRIS1 = 0; SW1 = 1; wait(); TRIS1 = 1; while (SW1==1) { time++; }

if (time-avg1 > thr1) {

buf1 = 20;

LED1 = 1;

} else {

if (buf1 > 0) {

buf1--;

} else {

LED1 = 0;

}

}

// toggle mode

time = 0; TRIS2 = 0; SW2 = 1; wait(); TRIS2 = 1; while (SW2==1) { time++; }

if (time-avg2 > thr2) {

if (buf2 == 0) {

LED2 = ~ LED2;

}

buf2 = 20;

} else if (buf2 > 0) {

buf2--;

}

// timer mode (adjust duration with variable "buf3")

time = 0; TRIS3 = 0; SW3 = 1; wait(); TRIS3 = 1; while (SW3==1) { time++; }

if (time-avg3 > thr3) {

buf3 = 500;

LED3 = 1;

} else {

if (buf3 > 0) {

buf3--;

} else {

LED3 = 0;

}

}

// dimmer mode (adjust dimming values in the array "pwm\_values")

time = 0; TRIS4 = 0; SW4 = 1; wait(); TRIS4 = 1; while (SW4==1) { time++; }

if (time-avg4 > thr4) {

if (buf4 == 0) {

pwm++;

if (pwm >= pwmlevels) {

pwm = 0;

}

CCPR1L = pwm\_values[pwm];

}

buf4 = 20;

} else if (buf4 > 0) {

buf4--;

}

// this function waits for eight cycles and does nothing

// (we need this so the stray capacity can charge up enough)

void wait (void) {

NOP(); NOP(); NOP(); NOP();

NOP(); NOP(); NOP(); NOP();

}