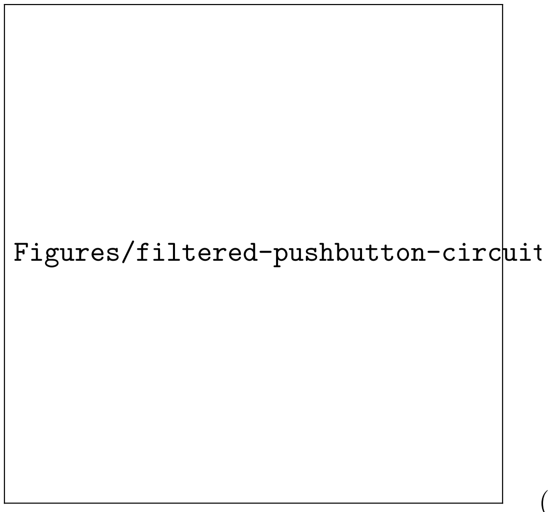
Switch Bounce & Catch the Clown Game
Embedded System Design, Lab 6

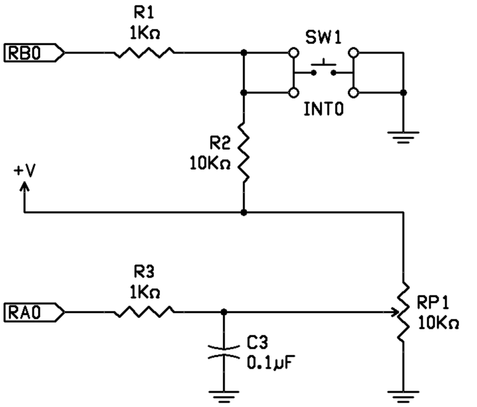
Ben Lorenzetti

November 5, 2015

Contents



(a) A Hardware Debounced Pushbutton: *R_3 and C form a low pass filter.*



(b) PIC16F887 Demo Board Schematic: *pushbutton SW1 is unfiltered and requires software debouncing.*

Figure 1: Hardware vs Software Debouncing for Mechanical Switches

1 Objectives and Problem Description

1.1 Does the Switch Bounce?

1.2 Catch the Clown!

Build a game for testing reaction times with an 8 LED rotating display, a pushbutton trigger, and a knob for adjusting the speed/difficulty. If the player presses the trigger in sync with the LED display, then the display stops rotating to indicate victory. The specifications can be summarized in the four points below:

1. For an 8 LED display, one LED should be illuminated at a time and the illuminated position should rotate right one digit every period.

2. The period should be adjustable on the fly with the rotatry potentiometer knob.

3. If the user triggers the switch while the topmost (most significant bit) LED is illuminated, then the LED display should stop rotating until the switch is released. The LED rotation loop should also continue–including through the topmost state–if the switch is active but was triggered during the wrong state.

4. The pushbutton switch should be debounced based on the results from part 1.

2 Procedure

2.1 Switch Bounce

3 Expected Results

4 Experiment and Design Revisions

4.1 Command Line Assembly

My `.asm` source files were assembled on the command line so please do this if they don't compile nicely in the IDE. On Ubuntu, with the default MPLAB installation location, from the directory containting `catch-the-clown.asm`, the commands are:

```
$ cp /opt/microchip/mplabx/v3.10/mpasmx/p16f887.inc ./p16f887.inc
$ /opt/microchip/mplabx/v3.10/mpasmx/mpasmx -p16f887 catch-the-clown.asm
$ more catch-the-clown.ERR
```

5 Observations

6 Discussion

7 Implementation Code

7.1 Does the Switch Bounce?

```
; debounce-tim.asm
; Ben Lorenzetti
; Embedded Systems Design, Fall 2015

#include <p16f887.inc>
        __CONFIG        _CONFIG1, _LVP_OFF & _FCMEN_OFF & _IESO_OFF & _BOR_OFF
            & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON & _WDT_OFF &
           _INTRC_OSC_NOCLKOUT
        __CONFIG        _CONFIG2, _WRT_OFF & _BOR21V

#define NEUTRAL_POS             0x80
#define INNER_DELAY_TIME        0x8F
#define MIDDLE_DELAY_TIME       0x0F
#define MINIMUM_HALF_PERIOD     0x06
#define OSC8_CHANNEL0_NOGO_ADON B'01000001'
#define LEFTJUSTIFY_VSS_VDD     B'00000000'
#define RESOLUTION_MASK         B'11111100'

;-------------------------Organize Program Memory-------------------------------;
Reset_Vector
        ORG 0
        GOTO Initialize

Interupt_Vector
        ORG .4

;---------------------Allocate Static Variables----------------------;
        cblock  0x20
        adc_result
        turn_signal
        delay_time
        outer_delay_counter
        middle_delay_counter
        inner_delay_counter
        endc

;----------Pause (INNER_DELAY * MIDDLE_DELAY * delay_time)----------;
Delay_Function
        MOVF    delay_time, W           ; copy delay_time to
        MOVWF   outer_delay_counter     ;   outer_delay_counter
        MOVLW   INNER_DELAY_TIME        ; initialize
        MOVWF   inner_delay_counter     ;   inner_delay_counter
        MOVLW   MIDDLE_DELAY_TIME       ; initialize
        MOVWF   middle_delay_counter    ;   middle_delay_counter
Inner_Loop
        DECFSZ  inner_delay_counter, f
        GOTO    Inner_Loop
        MOVLW   INNER_DELAY_TIME
        MOVWF   inner_delay_counter
Middle_Loop
```

```
        DECFSZ  middle_delay_counter, f
        GOTO    Inner_Loop
        MOVLW   MIDDLE_DELAY_TIME
        MOVWF   middle_delay_counter
Outer_Loop
        DECFSZ  outer_delay_counter, f
        GOTO    Inner_Loop
        RETURN

;------------------------------Initialize Data Memory------------------------------------------;
Initialize
        ;-------------------- Initialize I/O ----------------------------------------------;
        BANKSEL TRISD               ; select Register Bank 1
        CLRF    TRISD               ; set all LED pins to output
        BANKSEL PORTD               ; back to Register Bank 0
        CLRF    PORTD               ; set all LED pins to low
        BANKSEL TRISA
        CLRF    TRISA               ; clear TRISA
        BSF     TRISA, RA0          ; set port A pin 0 to input
        ;-------------------- Initialize ADC-----------------------------------------------;
        BANKSEL ADCON1
        MOVLW   LEFTJUSTIFY_VSS_VDD
        MOVWF   ADCON1              ; left justify result,
                ; use VSS and VDD for Vref- and Vref+
        BANKSEL ADCON0
        MOVLW   OSC8_CHANNEL0_NOGO_ADON
        MOVWF   ADCON0              ; ADC clock rate = Fosc/8,
                ; ADC input channel = 0, ADC on
        MOVLW   10
        MOVWF   delay_time          ; initialize delay_time
        CALL    Delay_Function      ; Pause to allow ADC to settle

;-----------------------Begin Main Program Loop------------------------------------------------;
Main
        ;------------------- Measure Potentiostat Input ----------------------------;
        BANKSEL ADCON0
        BSF     ADCON0, GO          ; start convertion
        BTFSC   ADCON0, GO          ; is converstion done?
        GOTO    $-1                 ; go back to BTFSC instruction
        BANKSEL ADRESH
        MOVFW   ADRESH              ; store ADC result in W
        BANKSEL PORTA               ; go back to bank 0
        ;---------- Calculate Angular Displacement from Neutral ----------;
        MOVLW   RESOLUTION_MASK     ; reduce number of steps by
        ANDWF   ADRESH, 1           ;   truncating lower bits in ADRESH
        MOVLW   NEUTRAL_POS
        SUBWF   ADRESH, 1           ; compute displacement from Neutral
         ; Z = 1 if ADRESH == NEUTRAL_POS; C = 1 if ADRESH >= NEUTRAL_POS
        ;----------------------- Perform Conditional Logic -------------------------;
        BTFSC   STATUS, Z           ; test zero flag, skip next if clear
        GOTO    Main                ; if (ADRESH == NEUTRAL_POS)
        BTFSS   STATUS, C           ; if (ADRESH < NEUTRAL_POS), invert
        COMF    ADRESH, F           ;   angular displacement
        MOVLW   1 << RD7            ; assume left turn (ADRESH < NEUTRAL)
        BTFSC   STATUS, C           ; if actually (ADRESH > NEUTRAL_POS),
        MOVLW   1 << RD0            ;   then fix it to be right (RD0)
        ;------------------- Blink LEDs ------------------------------;
        MOVWF   PORTD               ; turn on LED
        MOVLW   MINIMUM_HALF_PERIOD
        MOVWF   delay_time          ; keep LED on for fixed delay time
        CALL    Delay_Function      ;
        CLRF    PORTD               ; turn off LEDs
        MOVF    ADRESH, W           ; compute appropriate delay time
        SUBLW   NEUTRAL_POS + MINIMUM_HALF_PERIOD
        MOVWF   delay_time          ; (from angular displacement value)
        CALL    Delay_Function      ; delay
        ;---------- End of Main Function Loop ------------------------;
        GOTO    Main
;------------------------------ End of File -------------------------------;
        END
```

7.2 Catch the Clown!