

# Switch Bounce & Catch the Clown Game

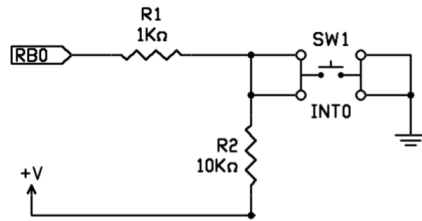
## Embedded System Design, Lab 6

Ben Lorenzetti

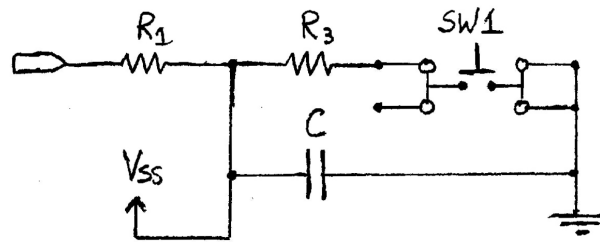
November 5, 2015

### Contents

<b>1</b>	<b>Objectives and Problem Description</b>	<b>2</b>
1.1	Part 1: Does the Switch Bounce? . . . . .	2
1.2	Part 2: Catch the Clown! . . . . .	2
<b>2</b>	<b>Procedure</b>	<b>2</b>
2.1	Switch Bounce Background . . . . .	2
2.2	Part 1: Debouncing SW1 . . . . .	3
2.3	Part 2: Catch the Clown Game . . . . .	3
<b>3</b>	<b>Expected Results</b>	<b>3</b>
3.1	Debounce Time . . . . .	3
<b>4</b>	<b>Experiment and Design Revisions</b>	<b>3</b>
4.1	Command Line Assembly . . . . .	3
4.2	PORTB Input . . . . .	3
4.3	Adjusting Debounce Time . . . . .	5
<b>5</b>	<b>Observations</b>	<b>6</b>
5.1	44-Pin Demo Board SW1 Bouncing Time . . . . .	6
<b>6</b>	<b>Discussion</b>	<b>6</b>
<b>7</b>	<b>Implementation Code</b>	<b>7</b>
7.1	Does the Switch Bounce? . . . . .	7
7.2	Catch the Clown! . . . . .	9



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



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

Figure 1: Hardware vs Software Debouncing for Mechanical Switches

## 1 Objectives and Problem Description

### 1.1 Part 1: Does the Switch Bounce?

### 1.2 Part 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 rotary 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 Background

Mechanical switches do not always produce clean, step function electrical signals. Instead, when a switch is thrown, the electric signal often ‘bounces’ several times before settling at the active level. This is caused by electric arcing and breaking as the contacts near one other and from mechanical vibrations of contacts from the throwing force.

Bouncing can cause problems if the device’s function depends on the number of times a switch is thrown. For example, if your keyboard bounced, you would find yourself typing repeat letters and unpredictable backspacing. Traditionally switches were ‘debounced’ with low pass filters in hardware, but with fast microcontrollers debouncing can be done in software to reduce hardware costs. Figure 1 shows how PIC did this with their PIC16F887 44-pin demo board.

## 2.2 Part 1: Debouncing SW1

## 2.3 Part 2: Catch the Clown Game

# 3 Expected Results

## 3.1 Debounce Time

From Professor Vemuri's lecture, figure 3 shows the timescale at which some random switch bounces. Based on this plot, the expected bouncing time of SW1 was on the order of  $500\mu s$ .

# 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 containing 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
```

## 4.2 PORTB Input

On the 44-pin demo board, the pushbutton is connected to RB0 on PORTB, in an active low configuration with a  $1k\Omega$ ,  $10k\Omega$  voltage divider, as shown in figure 1.

When I first wrote my implementation for part 1 of the lab, I could not get the  $\mu$ Controller to respond to pressing the pushbutton switch (SW1). After inspecting the hardware with a ohmmeter, I knew the problem had to be in software. I created a knew 'hello world' program for debugging the pushbutton switch.

It turns out the bug was a configuration problem: RB0 is connected to one of the 16 analog inputs, AD12. By default, the pin configured use with the ADC and the digital input amplifiers are turned off. Here is a nugget that was buried on page 49 of the PIC16F887 datasheet:

<b>Note:</b> The ANSELH register must be initialized to configure an analog channel as a digital input. Pins configured as analog inputs will read '0'.
---

Figure 4: Port B Configuration Note

With this nugget, the working 'pushbutton hello world' program was:

```
; pushbutton-test.asm
; Test active-low pushbutton on RB0 with active-high LED on RD0

#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
```

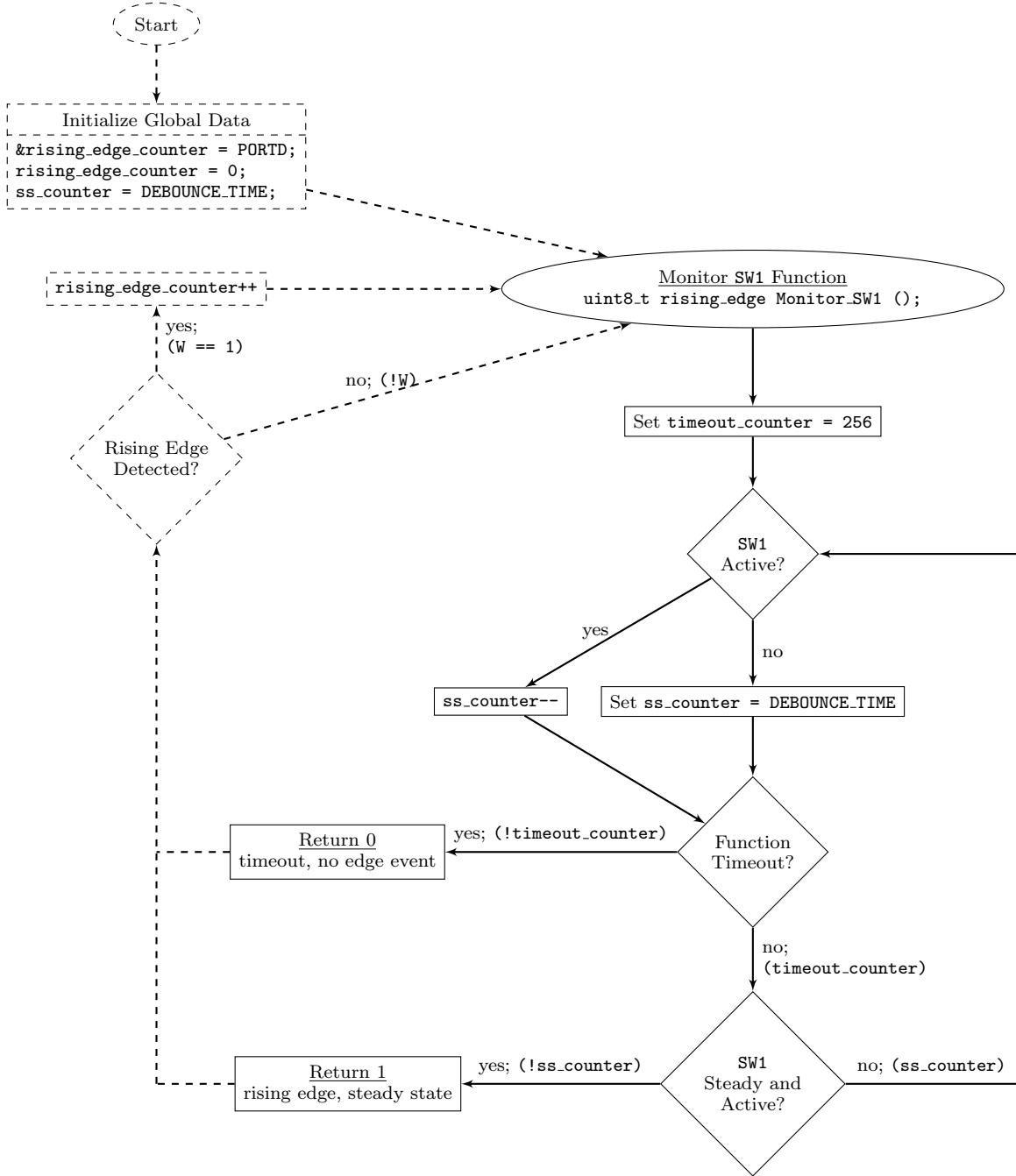


Figure 2: Implementation of rising\_edge Monitor\_SW1() Function

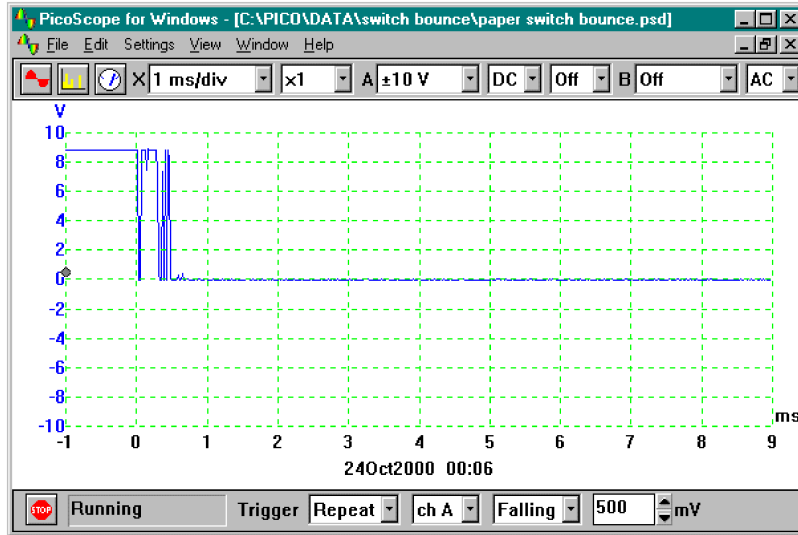


Figure 3: A Bouncing Input Signal from a Switch

```

Initialize_Data_and_IO
    CLRF    PORTB
    CLRF    PORTD
    BSF     STATUS, RP0      ; Switch from Bank 0 to Bank 1
    BSF     PORTB, RB0       ; configure RB0 as input
    BCF     PORTD, RD0       ; configure RD0 as output
    BSF     STATUS, RP1      ; Switch from Bank 1 to Bank 3
    BCF     ANSELH, ANS12    ; by default RB0/AN12 is configured as analog
                                ; input. Set to '0' to enable digital input
    BANKSEL 0x00             ; Switch to Bank 0

Main_Loop
    MOVF    PORTB, W         ; copy pushbutton input into W
    XORLW   1 << RB0         ; invert active-low pushbutton for active-high
                                ; output
    MOVWF   PORTD            ; update LED display
    GOTO    Main_Loop
    END

```

### 4.3 Adjusting Debounce Time

The results from part 1 of the lab are presented in [hyperref\[part-1-results-section\]](#)section 5.1. From these results, the `DEBOUNCE_TIME` for the part 2 implementation was adjusted to `XXX` to require a steady state, debouncing period of

$$T = X * 14 \mu s = XXX$$

before a switch activation is accepted by the program.

## **5 Observations**

### **5.1 44-Pin Demo Board SW1 Bouncing Time**

## **6 Discussion**

## 7 Implementation Code

### 7.1 Does the Switch Bounce?

```
; debounce-time.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 DEBOUNCE_TIME    .1        ; software debounce sampling period is
                                    ; T = DEBOUNCE_TIME * 14 microseconds

#define rising_edge_counter    PORTD
#define timeout_counter        0x20
#define ss_counter              0x21    ; steady state counter for debouncing

Reset_Vector
    ORG          0
    GOTO         Main

;----- uint8_t rising_edge Monitor_SW1 () -----;
Monitor_SW1_Function
                                ; static uint8_t ss_counter;
    CLRF         timeout_counter ; uint8_t timeout_counter = 256;
Debounce_Loop
;----- Monitor Pushbutton Input -----;
    BTFSC        PORTB, RB0      ; if (switch == 1)
    MOVLW        DEBOUNCE_TIME   ; ss_counter = DEBOUNCE_TIME;
    BTFSS        PORTB, RB0      ; if (!switch) // SW1 is active low
    DECF         ss_counter, W    ; ss_counter--;
    MOVWF        ss_counter
;----- Check for Timeout -----;
    DECF         timeout_counter, F ; debounce_counter--;
    BTFSC        STATUS, Z        ; if (!debounce_counter)
    RETLW        0                ; return 0;
;----- Check if SW1 has been steady for debounce time -----;
    ANDLW        0xFF             ; Z = !ss_counter;
    BTFSC        STATUS, Z        ; if (Z)
    RETLW        1                ; return 1;
                                ; else
    GOTO         Debounce_Loop    ; continue;

Main
Initialize_Variables
    CLRF         rising_edge_counter
    CLRF         timeout_counter
    MOVLW        DEBOUNCE_TIME
    MOVWF        ss_counter
```

```

Configure_IO
    BSF      STATUS, RP0      ; switch from bank 0 to bank 1
    CLRF     TRISD            ; configure PORTD for 8-LEDs output
    BSF      TRISB, RB0       ; configure PORTB Pin 0 for pushbutton input
    BANKSEL  ANSELH           ; by default, RB0/AN12 is configured as analog
    BCF      ANSELH, ANS12    ; input. Reconfigure to digital.
    BANKSEL  0x00             ; return to bank 0

Forever_Loop
    CALL     Monitor_SW1_Function
    ADDWF    rising_edge_counter
    GOTO     Forever_Loop

END

```



## 7.2 Catch the Clown!

```

; catch-the-clown.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 OUTER_MIN_PERIOD    .4
#define MIDDLE_SCALAR      .8
#define DEBOUNCE_TIME      .8
#define CLOWN_STATE        1 << 7
#define OSC8_CHANNEL0_NOGO_ADON 0x41
#define LEFTJUSTIFY_VSS_VDD 0x00

#define led_state          PORTD
#define period              0x20
#define outer_counter      0x21
#define middle_counter     0x22
#define timeout_counter    0x23    ; timeout counter local to Monitor_SW1 ()
#define ss_counter         0x24    ; static, steady state time counter

Reset_Vector
    ORG    0
    GOTO   Main

;----- uint8_t rising_edge Monitor_SW1 () -----;
Monitor_SW1_Function
    ; static uint8_t ss_counter;
    CLRF    timeout_counter ; uint8_t timeout_counter = 256;
Debounce_Loop
    ;----- Monitor Pushbutton Input -----;
    BTFSC   PORTB, RB0      ; if (switch == 1)
    MOVLW   DEBOUNCE_TIME   ; ss_counter = DEBOUNCE_TIME;
    BTFSS   PORTB, RB0      ; if (!switch) // SW1 is active low
    DECF    ss_counter, W    ; ss_counter--;
    MOVWF   ss_counter
    ;----- Check for Timeout -----;
    DECF    timeout_counter, F ; debounce_counter--;
    BTFSC   STATUS, Z        ; if (!debounce_counter)
    REILW   0                ; return false;
    ;----- Check if SW1 has been steady for debounce time -----;
    ANDLW   0xFF             ; Z = !ss_counter;
    BTFSC   STATUS, Z        ; if (!ss_counter)
    REILW   0xFF             ; return true;
    ; else
    GOTO    Debounce_Loop    ; continue;

Main
Initialize_IO

```

```

        BSF      STATUS, RP0      ; switch from Bank 0 to Bank 1
        CLRF     TRISD            ; configure port D to output for LEDs
        BSF      TRISB, RB0       ; configure pushbutton on RB0 for input
        BSF      TRISA, RA0       ; configure potentionmeter on RA0 for input
        BANKSEL  ANSEL            ; go to bank 3
        BSF      ANSEL, 0
        BANKSEL  ANSELH           ; by default, RB0/AN12 is configured as analog
        BCF      ANSELH, ANS12    ; input. Reconfigure to digital.
        BANKSEL  0x00            ; return to Bank 0

Initialize_Analog_to_Digital_Converter
        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

ADC_Acquisition_Time
        CLRF     outer_counter
        CLRF     middle_counter

One_Off_Delay_Loop
        DECFSZ   middle_counter, F
        GOTO     One_Off_Delay_Loop
        DECFSZ   outer_counter, F
        GOTO     One_Off_Delay_Loop

Initialize_State_Machine
        MOVLW    CLOWN_STATE
        MOVWF    led_state
        MOVLW    DEBOUNCE_TIME
        MOVWF    ss_counter

State_Machine_Loop

Measure_Potentiometer
        BSF      ADCON0, GO       ; start conversion
        BTFSC    ADCON0, GO       ; is conversion done?
        GOTO     $-1             ; go back to is conversion done?

Set_Period
                                ; swap upper nibble of ADC result
        SWAPF    ADRESH, W        ; into lower 4 bits of W
        ANDLW    0x0F            ; keep only the lower 4 bits
        ADDLW    OUTER_MIN_PERIOD;
        MOVWF    period          ; period = (ADC_result >> 4) + MIN_PERIOD

; for (outer_counter=OUTER_SCALAR; !outer_counter; outer_counter--)
Open_Outer_Loop
        MOVF     period, W
        MOVWF    outer_counter    ; outer_counter = OUTER_SCALAR

Outer_Loop

; for (middle_counter=MIDDLE_SCALAR; !middle_counter; middle_counter--)

```

```

Open_Middle_Loop
    MOVLW    MIDDLE_SCALAR
    MOVWF    middle_counter    ; middle_counter = MIDDLE_SCALAR
Middle_Loop

Monitor_SW1
    CALL     Monitor_SW1_Function

Clown_Caught_Logic
    ADDLW    0                ; Z = !return_value
    BTFSC    STATUS, Z        ; if (!return_value) // timeout occurred
    GOTO     Close_Middle_Loop;    continue;

    MOVF     led_state, W
    SUBLW    CLOWN_STATE      ; Z = if (led_state == CLOWN_STATE)
    BTFSC    STATUS, Z        ; if (Z)
    GOTO     Initialize_State_Machine; // you win!

Close_Middle_Loop
    DECFSZ   middle_counter, F
    GOTO     Middle_Loop

Close_Outer_Loop
    DECFSZ   outer_counter, F
    GOTO     Outer_Loop

Next_State_Transition
    BCF      STATUS, C
    RRF      led_state, F      ; led_state = state >> 1;
    BTFSS    STATUS, C        ; if (state)
    GOTO     State_Machine_Loop;    continue; // continue forever loop
                                ; else {
    RRF      led_state, F      ; led_state = 128;
    GOTO     State_Machine_Loop
                                ; continue; // continue forever loop

END

```