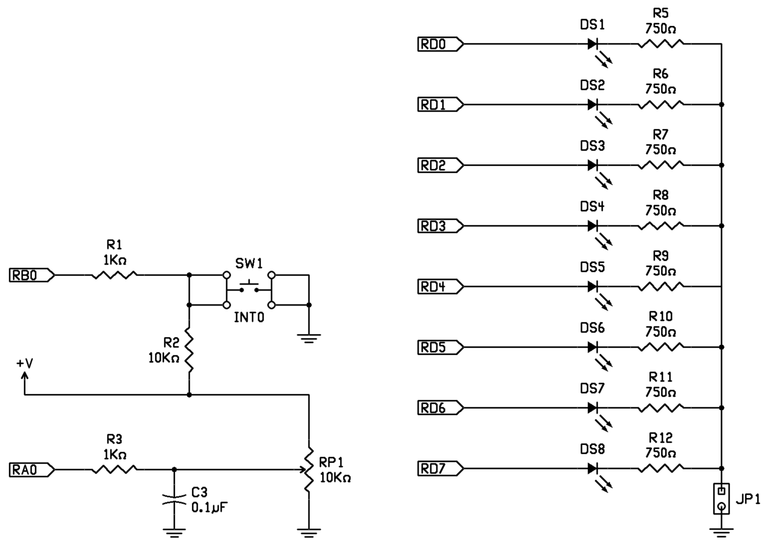
# Tricycle Lights
# Embedded System Design, Lab 6

Ben Lorenzetti

October 29, 2015

## Contents




(a) Potentiometer Input (b) Output LEDs

Figure 1: I/O Circuit Diagrams from the 44–Pin Demo Board User's Manual

# 1 Objectives and Problem Description

## 1.1 Tricycle Lights

Blink two LEDs, representing left and right turn signals, depending on the position of a rotary potentiometer, which represents a steering wheel. The following conditions should be met.

1. Use the potentiometer and LEDs on the 44–Pin Demo Board.
2. Use the LED connected to RD7 for the left turn signal, and RD0 for right.
3. When wheel is in neutral position (wiper in middle of pot.), both turn indicator lights should be off.
4. When turned counterclockwise or clockwise from neutral position, the left or right LEDs should blink at a rate proportional to the angular displacement from neutral position.

# 2 Procedure

## 2.1 Potentiometer and LEDs on the 44–Pin Demo Board

In this lab, we will need to measure a potentiometer input (the steering wheel) and blink two LEDs for output. The 44–pin demo board for the PIC16F887 has all of this hardware on board. The potentiometer and LEDs are connected to pins as shown in the circuit diagram in figure 1.

According to the specifications, the LED at RD7 should be the left blinker and the LED at RD0 should be the right blinker.

## 2.2 Clock Sources on the PIC16F887 and 44–Pin Demo Board

To use the analog to digital converter, we need to know the frequency of the system's clock. The PIC16F887 microcontroller can be configured to use an internal RC oscillator or external crystals/clocks with various prescalars. The block diagram below shows all of the possible options for the oscillator module of the PIC16F887.

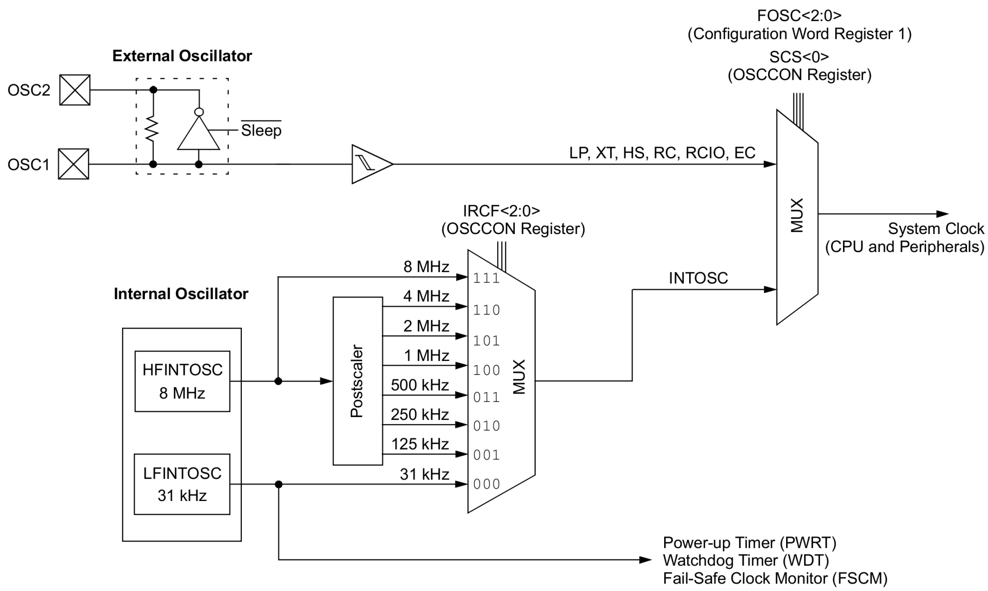



Figure 2: Oscillator Module Block Diagram from PIC16F887 Datasheet

The 44–pin demo board has 10 MHz and 32 kHz oscillators, which are connected to the $\mu$Controller as shown in figure 3. However, by default the PIC16F887 $\mu$Controller runs from its internal high frequency RC oscillator with the prescalar set to yield a 4 MHz clock rate.

$$F_{OSC} = 4MHz$$

## 2.3 PIC16F887 Analog to Digital Converter (ADC)

The PIC16F887 has a successive approximation analog to digital converter (ADC) which can be used to measure the voltage on `RA0` with 10–bit precision. Figure 4 and from the the PIC16F887 datasheet shows how the ADC fits into the $\mu$Controller.

Successive approximation is a process similar to binary search, where the input voltage being measured is compared with a series of binary decisions honing in on the actual value. The ADC has its own clock source, but usually it is just the main CPU/peripherals clock divided by a prescalar. The ADC requires $11 * T_{AD}$ to complete one conversion: 10 ADC clock cycles for 10 binary comparisons with 1 more ADC clock cycle in overhead. Figure 5, from the PIC16F887 datasheet, shows the 11–step conversion process and the recommended prescalar from the main CPU clock.

The ADC clock $T_{AD}$ should not exceed the recommended rate because parasitic capacitance in the pins and traces will cause aliasing. In section 2.2, we decide the main CPU clock ($F_{OSC}$) is 4 MHz; therefore, the prescalar `ADCS` should be *configured to* $F_{OSC/8}$:

$$\texttt{ADCON0} = 01\text{XX XXXX}$$

The result of an ADC operation is stored in registers `ADRESH` and `ADRESL` and is equal to

$$\texttt{[ADRESH:ADRESL]} = \frac{2^{\text{bit precision}} - 1}{V_{\text{REF+}} - V_{\text{REF+}}} * V_{in}$$

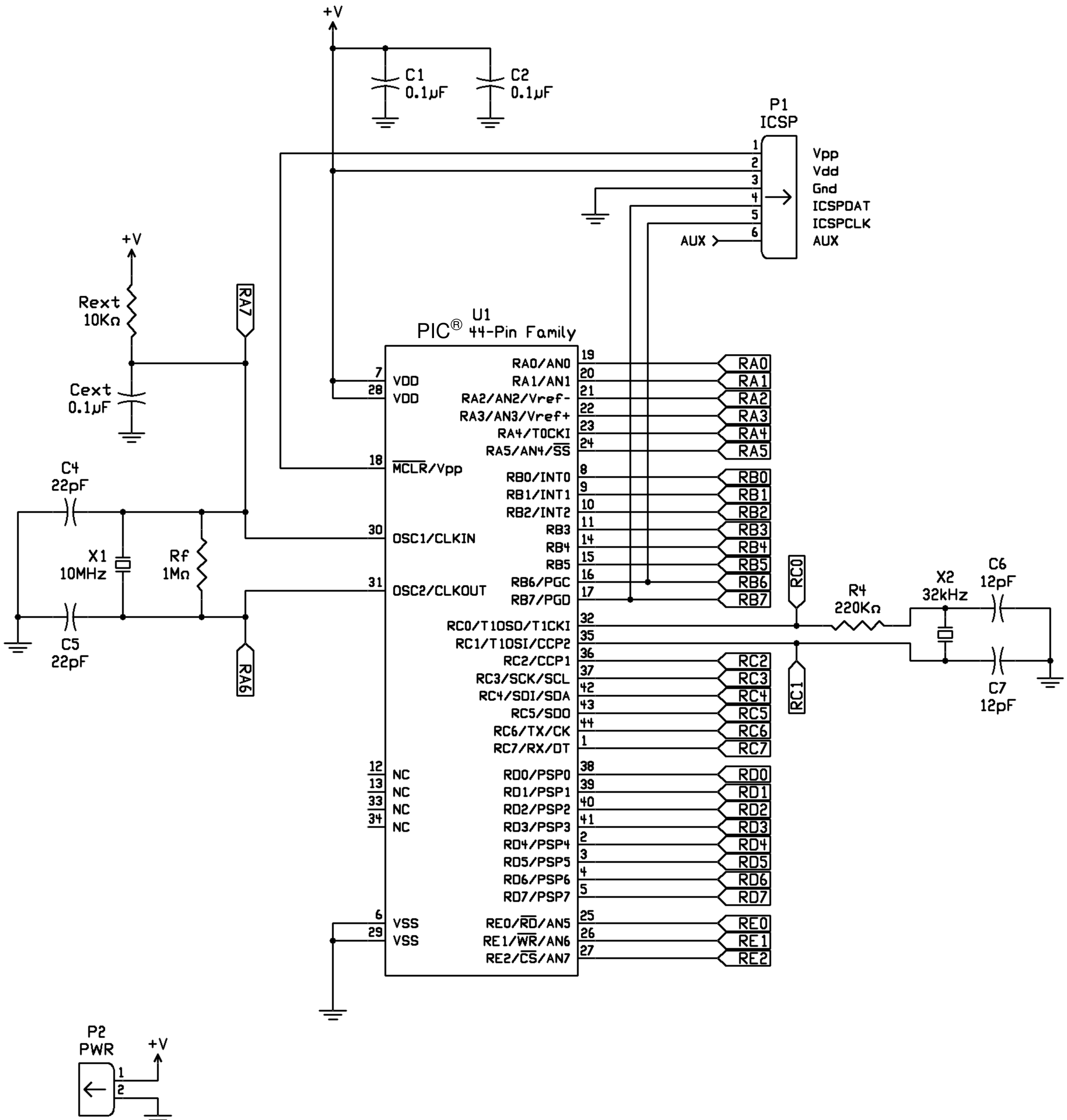



Figure 3: Oscillators, Power, and Programming Connections on the 44–Pin Demo Board

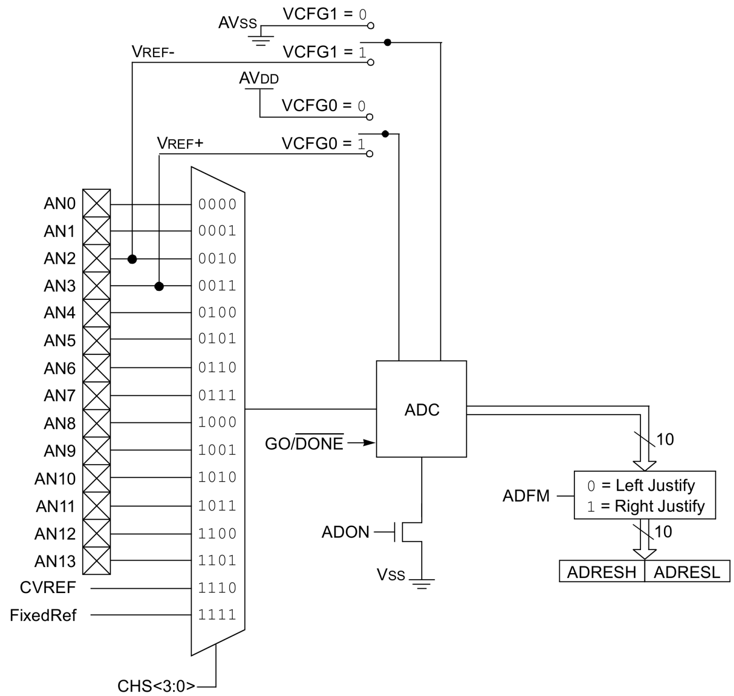



Figure 4: PIC16F887 ADC Block Diagram

**TABLE 9-1: ADC CLOCK PERIOD (TAD) VS. DEVICE OPERATING FREQUENCIES (VDD ≥ 3.0V)**

| ADC Clock Period (TAD) | | Device Frequency (FOSC) | | | |
|---|---|---|---|---|---|
| **ADC Clock Source** | **ADCS<1:0>** | **20 MHz** | **8 MHz** | **4 MHz** | **1 MHz** |
| FOSC/2 | 00 | 100 ns(2) | 250 ns(2) | 500 ns(2) | 2.0 μs |
| FOSC/8 | 01 | 400 ns(2) | 1.0 μs(2) | 2.0 μs | 8.0 μs(3) |
| FOSC/32 | 10 | 1.6 μs | 4.0 μs | 8.0 μs(3) | 32.0 μs(3) |
| FRC | 11 | 2-6 μs(1,4) | 2-6 μs(1,4) | 2-6 μs(1,4) | 2-6 μs(1,4) |

**Legend:** Shaded cells are outside of recommended range.

**Note 1:** The FRC source has a typical TAD time of 4 μs for VDD > 3.0V.

**2:** These values violate the minimum required TAD time.

**3:** For faster conversion times, the selection of another clock source is recommended.

**4:** When the device frequency is greater than 1 MHz, the FRC clock source is only recommended if the conversion will be performed during Sleep.

**FIGURE 9-2: ANALOG-TO-DIGITAL CONVERSION TAD CYCLES**

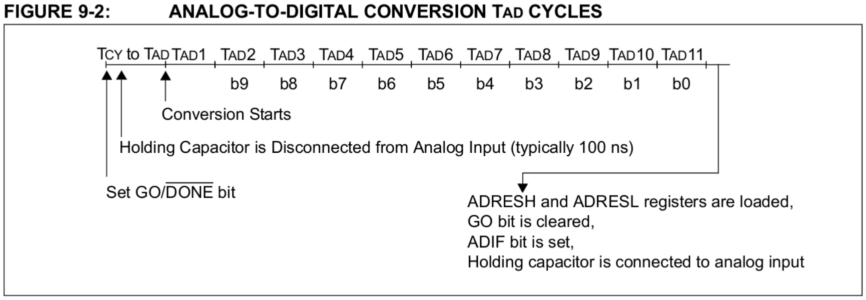



Figure 5: PIC16F887 Recommended ADC Clock Rate ($T_{AD}$) and Measurement Time

For this lab, a human would not be able to notice the difference between 10–bit precision and 8–bit precision in the small angular range of the rotary potentiometer. Furthermore, because the PIC16F887 uses an 8–bit data paradigm, reducing the ADC result to eight bits would make implementation much easier. This can be done by *configuring the ADC to justify left* and then taking only `ADRESH`.

$$\texttt{ADCON1} = 0\text{XXX XXXX}$$

According to the schematic in figure 1, the $10k\Omega$ potentiometer has terminals connected to `V+` and `Gnd`; and its wiper is connected to `RA0` (`PORTA, 0`) with a $1k\Omega$ current limiter. With this configuration, the voltage from the wiper can vary between $V_{SS}$ and $V_{DD}$. The ADC can be *configured to the upper and lower bounds of the expected input* for more accurate measurments. For $V_{SS}$ and $V_{DD}$, `VCFG[1:0]` should be

$$\texttt{ADCON1} = \text{XX00 XXXX}$$

With these configuration settings, the result of an ADC conversion is now stored in `ADRESH` and given by

$$\texttt{ADRESH} = \frac{255}{3.3V} * V_{in}$$

The general process for ADC conversion in code is:

1. Configure Port
```
BANKSEL TRISA
BSF TRISA, 0 ; set RA0 to input
BANKSEL ANSEL
BSF ANSEL, 0 ; set RA0 to analog
```
2. Configure ADC Module
   `; ADCON1 = 0xxx xxxx` - select result format as left justify
   `; ADCON1 = xx00 xxxx` - configure reference voltages to $V_{SS}$ and $V_{DD}$
   `; ADCON0 = 01xx xxxx` - set ADC conversion clock rate to $F_{OSC/8}$
   `; ADCON0 = xx00 00xx` - select input channel to AN0
   `; ADCON0 = xxxx xxx1` - turn ADC on `BANKSEL ADCON1`
```
MOVLW B'00000000'
MOVWF ADCON1
BANKSEL ADCON0
MOVLW B'01000001
MOVWF ADCON0
```
3. Configure ADC interupt (optional)
4. Wait for the required ADC settling time
5. Start conversion by setting the GO/$\overline{\text{DONE}}$ bit
```
BSF ADCON0, GO ; start conversion
```
6. Wait for ADC conversion to complete
```
BTFSC ADCON0, GO ; is conversion done?
GOGO $-1 ; if no, test gain
```
7. Read the ADC result from `ADRESH` (and `ADRESL`)

### 2.4 Implementation Flowchart

### 2.5 Delay Function

### 2.6 Linear Mapping

## 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 `tricycle-lights.asm`, the commands are:

```
$ cp /opt/microchip/mplabx/v3.10/mpasmx/p16f887.inc ./p16f887.inc
$ /opt/microchip/mplabx/v3.10/mpasmx/mpasmx -p16f887 tricycle-lights.asm
$ more tricytle-lights.ERR
```

## 5 Observations

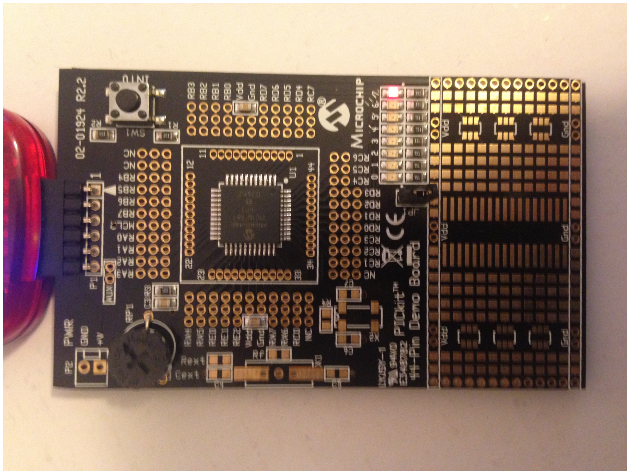


Figure 6: Demonstration of Tricycle Turn Lights

## 6 Discussion

# 7 Tricycle Lights Implementation Code

```
; tricycle-lights.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
```