EEL 4914 Senior Design

Final Design Report

April 21st, Spring 2008

Auto Rev Matcher

Team Name: "The Cowboys Lost Again"

Submitted by:

Monique Mennis
moniki@ufl.edu

Brad Atherton
masscles@ufl.edu

Abstract

Our device minimizes the frequency differential between the engine and transmission of a manual automobile for increased longevity of the clutch plate. In human terms our device can be seen as an automated RPM matcher. Technical challenges may entail finding an appropriate sampling rate for the inputs of our microprocessor, and correctly calibrating the appropriate RPM value for each gear from a series of tests. We expect our product to be a valuable asset in the car performance industry.

I. Introduction

In the high performance vehicle industry there is a strong demand for additional features that allow a driver to perform gear changes within milliseconds without having to reduce speed, decrease engine power or overuse the clutch. Current technology allows the driver to select the gear he / she wishes to shift into directly before or after the gear is currently engaged. Usually a shift lever is used to select the adjacent higher or lower gear. The shift lever operates like a ratchet mechanism that converts fore and aft motion into rotary motion.

There are various different types of products installed in today's high performance vehicles allowing the driver greater control over the shifting mechanism of the vehicle. With our "Auto Rev Matcher" we aim to allow the everyday driver similar control in their conventional vehicle.

II. Project Features

Main Objectives

- Maximized lifetime of clutch plate
- Minimized jerk from clutch engagement

Input Sensors and Switches:

- Speed sensor
- Up-shift / Down-shift switch
- Enter button
- Clutch pedal switch
- Emergency disable switch

Output Devices and Actuation:

- Throttle body controller servo motor
- LCD display

III. Concept / Technology

Atmega32 Microcontroller

We chose the Atmega32 over other microprocessors due to its wide availability and low cost. In the development stage this processor was seen as the best option given our resources and prior experience with other Atmel processors.

LCD Display

A basic LCD display is used to inform the user of their current speed and gear when the clutch is not engaged. When the clutch is engaged the LCD enables the user to see which gear he / she is switching into.

Speedometer Sensor

A 6.6 V powered speedometer senor signal is read in as an input to our microprocessor. This transitional input allows us to calculate current speed and rpm ranges.

Clutch Sensor

The clutch sensor is read as an input into our microprocessor allowing software to determine whether or not the clutch is depressed.

Up-shift / Down-shift Clicker

The up-shift / down-shift clicker input allows the user to specify which gear he / she intends to shift into next.

Enter & Reset Button

The enter button input allows the user to confirm his / her gear selection. The reset button input allows an emergency hardware reset that moves the servo motor controller back to its neutral position.

Servo Motor

The HS-985MG servo motor output allows the microprocessor control over the throttle cable on the vehicle.

IV. Product Comparison

BMW

The BMW M5 Sedan offers a “7-speed M Drivelogic sequential gearbox system.” It features gear change keys on the steering wheel and a selection lever on the central console. Gear changes are made within milliseconds and special function features such as slip recognition or hill recognition adapt to the gear shift points required in certain driving conditions.

Nissan

The Nissan r35 GTR has a 6-speed “Dual Clutch Transmission” with three driver-selectable modes. Normal mode allows for maximum smoothness and efficiency while snow mode allows for gentler starting and shifting on slippery surfaces. Lastly R mode gives the driver maximum performance with fastest shifts. The “Dual Clutch” design changes gears in less than 0.5 seconds. Other features are available such as “Downshift Rev Matching” (DRM) and the “Predictive pre-shift control” (in R-mode).

VW / Audi

The Volkswagon DSG Transmission delivers identical acceleration while putting the driver in closer contact with the rise and fall of the engine’s power curve. It allows manual shifting using a Tiptronic® shift lever or, when equipped, buttons in the steering wheel. The interaction between the clutches and shafts is such that the next higher gear is always permanently engaged and ready for activation.

Alfa Romeo

The Alfa Romeo Selespeed uses paddles or a joystick, with the joystick having a higher priority when shifting. The speed of the gear changes depends on the engine revs and the system also has a rev limiter to avoid over revving. The gearbox is made for sportive driving but a city mode option is also available that simulates automatic driving.

Lamborghini Gallardo

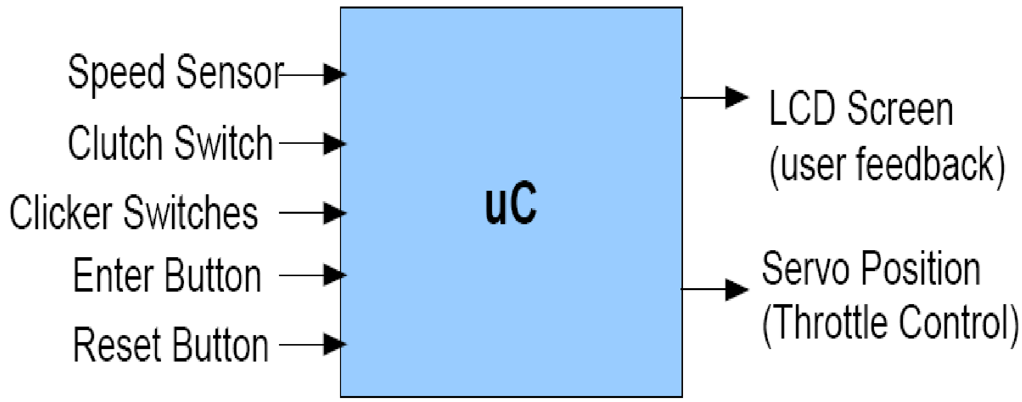
The new 2009 Lamborghini Gallardo uses an “e-Gear sequential transmission system.” This system now takes 40% less time to switch gears than previous models. The revised Gallardo can hit 60 mph in 3.7 seconds and can achieve a top speed of 202 mph.

Ferrari 599 GTB

The “F1-SuperFast Transmission” on the Ferrari 599 GTB is able to shift gears in 100 milliseconds. By overlapping the clutching and shifting tasks, harshness in shifting is reduced along with shift time.

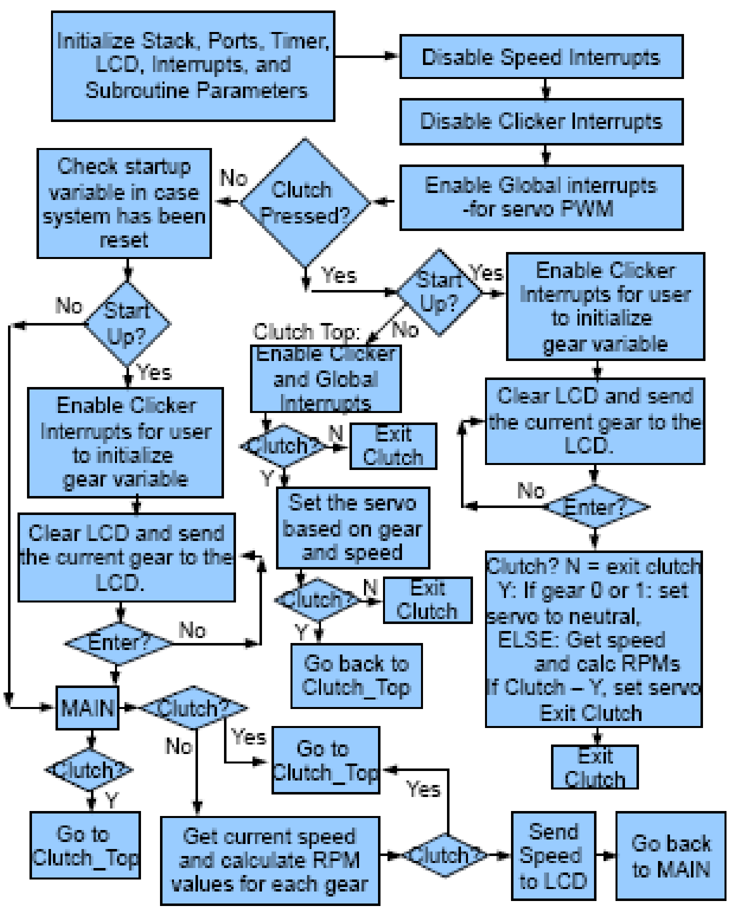
V. Project Architecture

The general I/O structure of the Auto Rev Matcher is shown in the figure below.


Speed Sensor
Clutch Switch
Clicker Switches
Enter Button
Reset Button
uC
LCD Screen
(user feedback)
Servo Position
(Throttle Control)

VI. Flowcharts and Diagrams

The system flowchart is show below. For additional upper-level understanding and organization, each box represents a subroutine in the software. Each subroutine has its own flowchart that can be found in the appendix that describes how the software is able to accomplish the task.



VII. Debugging Issues

The primary challenges faced during the programming phase of the project were related to interrupt timing and CPU issues. Hardware bugs discovered during the software stage also caused recursive issues where the errors were undetermineable (whether they were due to hardware or software) until a more detailed investigation of the hardware was performed.

General Interrupt Bugs

All stack operations (except for return addresses) in AVR microcontrollers are programmer controlled, so all data that may be necessary for program operation must be handled accordingly in the interrupt handler. This includes the status register and all registers that will be using in the handler. If interrupts are enabled during a section of the program where branching or status flag testing occurs, then the status register must be saved at the beginning of all the interrupt handlers that may be executed during this part of the program. The AVR does not do this automatically! Extensive debugging was performed until this was realized first through examination, and then validated by the microcontroller's data sheet. Always read the data sheets, they are your friends.

Solving other interrupt bugs required a macro-micro examintation of the overall program and a flowchart of interrupt timing to provide the macroscopic view of all possible interrepts and nested interrupts. For example, the PWM signal for the servo is interrupt-generated, so global interrupts must always be enabled for this to work properly, even during other interrupts. This places a significant risk of unplanned nested interrupts, especially during clicker switch interrupts due to bouncing. These problems were resolved by disabling the particular interrupts during their own interrupt handlers. Modifications of when to re-enable the particular interrupts were added to the flowcharts and software, with the minimal risk of possibly missing an interrupt. Thankfully humans are slow, the microcontroller is fast, and most of the interrupts are man-generated, so this did not pose a problem.

Speed Sensor Bugs

The majority of programming time was spent on the speedometer section of the program. The speed sensor does not feature much resolution; only four full sqaure waves represent one full revolution of the sensor. Because the speed sensor turns very slowly (over 8 seconds for a full revolution at 1 mph), initially two transition interrupts were used to catch a rising-then-falling or a falling-then-rising pair of edges to minimize the time required to capture a speed sample. This method only worked partially; a large percentage of the samples were spikes of speed changes that were not realistic values. After checking the interrupt timing and timer values (to ensure the error was not in software), it was determined that the sensor was causing the spikes. An initial attempt at signal averaging was experimented with, but an excessive amount of samples were spikes instead of the real (expected) value, so this method did not prove successful. Althought no datasheets were available to determine the internal operations of the sensor, cscilloscope measurements showed that the voltage was dropping out temporarily when the square was in a high state. Various capacitors were tested to hold the voltage high during the moments of drop-out. Too much capacitance take excess time to charge, causing an approximate ramp function at the signal pins. Too small of capacitance would not have enough energy storage to sustain the votlage during the drop-out period. This problem was resolved with a 0.1 uF capacitor. No further speed-code debugging was required after the capacitor was implemented.

Clicker Input Bugs

The bugs from the clicker switches were the typical bounce issues, but being momentary switches, bouncing is prone to occur twice. A software delay of more than a 3/8 second was implemented with the expectation that the user will press and release the momentary switch within that period.

VIII. Measurements

Speed Sensor

Voltage and current measurements supplied to and consumed by the speedometer sensor are show in figure “Speed Sensor Measurements”. Measurements at 7.5 volts, 6.6 volts, and 6 volts were the most important data. The sensor is supplied with 7.5 volts in Isuzu vehicles, but the signal voltage exceeds 5 volts, creating potential problems if connected directly to the uC. A 6.6 supply voltage provided a 5 volt (high) signal voltage which proved compatible with the sensor and uC. An LM317 voltage regulator was used to realize this voltage.

Speed Sensor Measurements

Supply Voltage	Signal Voltage		Supply Current	
(Volts)	High	Low	High (mA)	Low (mA)
3	0.57	0.565	1	1
4	0.58	0.58	2	2.5
5	3.7	0.59	3	4
6	4.52	0.59	4	5
6.6	5	0.6	5	5
7	5.35	0.6	5	6
7.5	5.78	0.6	6	6
8	6.26	0.61	6	6
9	7.21	0.61	6	6
10	8.16	0.61	6	6
11	9.12	0.615	6	7
12	10.07	0.62	6	7
13	11.02	0.62	6	7
14	11.98	0.62	6	7

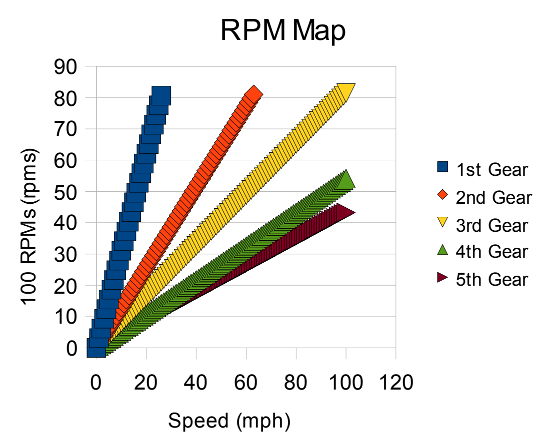
Speed-RPM

The ratios (of each gear) of the speed:rpm coordinates were measured with the vehicle's dashboard instrument panel gages. To reduce error, several points were recorded for each gear, and then a linear regression was used to minimize human error from “eye-balling” the measurements. Since the relationship between speed and rpm is linear and all lines converge at the null, the graph “RPM Map” below shows two points for each line, the null and the nearest integer ratio point. Note the emphasis on the nearest integer ratio point since

RPM Map

Gear – (RPM/100)

Speed (mph)	1	2	3	4	5
0	0	0	0	0	0
10	31	x	x	x	x
7	x	9	x	x	x
43	x	x	35	x	x
50	x	x	x	27	x
21	x	x	x	x	9



IX. Hardware / Software

Atmega32

We chose the Atmega32 microprocessor because of the following features:

– 131 Powerful Instructions – Most Single-clock Cycle Execution

– 32 x 8 General Purpose Working Registers

– Fully Static Operation

– Up to 16 MIPS Throughput at 16 MHz

– On-chip 2-cycle Multiplier

– 32K Bytes of In-System Self-programmable Flash program memory

– 1024 Bytes EEPROM

– 2K Byte Internal SRAM

– Write/Erase Cycles: 10,000 Flash/100,000 EEPROM

– Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes

– One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode

– Real Time Counter with Separate Oscillator

– Four PWM Channels

- 8 Single-ended Channels

- 7 Differential Channels in TQFP Package Only

- 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x

– Byte-oriented Two-wire Serial Interface

– Power-on Reset and Programmable Brown-out Detection

– Internal Calibrated RC Oscillator

– External and Internal Interrupt Sources

– 32 Programmable I/O Lines

– 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF

• Power Consumption at 1 MHz, 3V, 25°C for Atmega32L

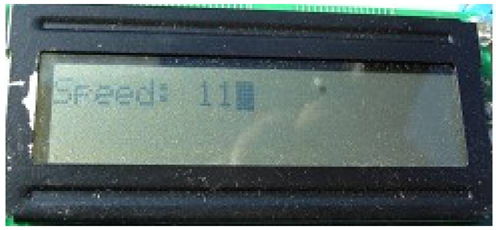
– Active: 1.1 mA

– Idle Mode: 0.35 mA

– Power-down Mode: < 1 µA

LCD Display

The LCD display provided us with two 16 character lines in 4-bit mode. Connecting to pins porta.0 through porta.6 on the Atmega32, current speed and gear options are displayed for the user while the product is enabled.



Speedometer Sensor

The speedometer sensor four cables consist of ground, signal, a no-connect, and 6.6 V power. The signal cable is connected to portb.2 of the Atmega32 microprocessor. With this signal cable as an input we are able to keep track of the time between the transitions of a square wave and calculate the current speed of the vehicle.

Clutch Sensor

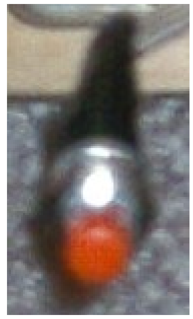
On portd.5 of the Atmega32 the clutch sensor input is connected allowing software to determine when the clutch is depressed enabling up-shifting or down-shifting options.

Up-shift / Down-shift Clicker

The up-shift / down-shift clicker allows the user to select which gear they would like to shift into.

Enter & Reset Button

The enter button is pulled low with registering true, allowing the user to confirm his / her gear selection. A complete hardware reset is always available to the user by means of a reset button. When clicked the reset button goes low and resets the Atmega32 microprocessor thus setting the servo motor back to its neutral position.



Servo Motor

The HS-985MG servo motor from servo city provides us with 180 degree rotation of 172 oz-in. of torque in 0.13 sec/60º. Using a pwm signal with a 3-5 volt peak to peak voltage we are able to control the throttle on our manual car. With the 5:1 aluminum gear wheel we are able to gain the resolution necessary to optimally operate.

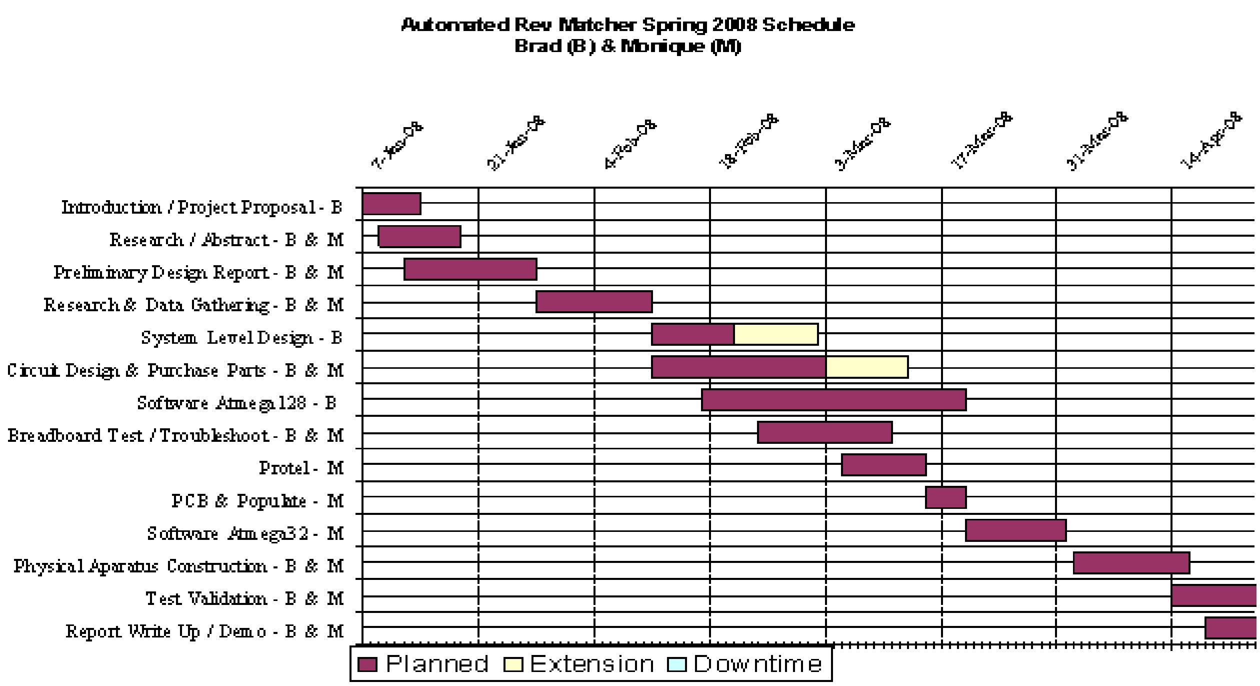
X. Bill of Materials

	Amount	Individual Price	Total
HS-985MG Servo Motor	1	$152.94	$152.94
Atmega32 µP	1	$5.50	$5.50 (Free)
Servo Mounting Brackets & Supplies			$13.74
Audio Jack Connectors	10	$2.99	$29.90
Audio Jack Plugs	10	$3.99	$39.90
LM317T Voltage Regulator	2	$2.29	$4.58
7805 Voltage Regulator	1	$3.75	$3.75(Free)
PCB Container	1	$1.62	$1.62
Wood	3	$3.50	$10.50
Misc.	4	$0.98	$3.92
LCD Screen	1	$25.00	$25.00 (Free)
24-Gauge Wire	1	$3.99	$3.99
Total			$295.34

The total cost of our product came to be $270.34. This price is well under the range of more sophisticated systems in high performance vehicles and allows a driver similar options. The servo motor was the most expensive part in this design. In searching for high torque motors, ones that were suited to our needs were in this higher price range. Additional costs may be incurred if our device were to be installed on a different vehicle.

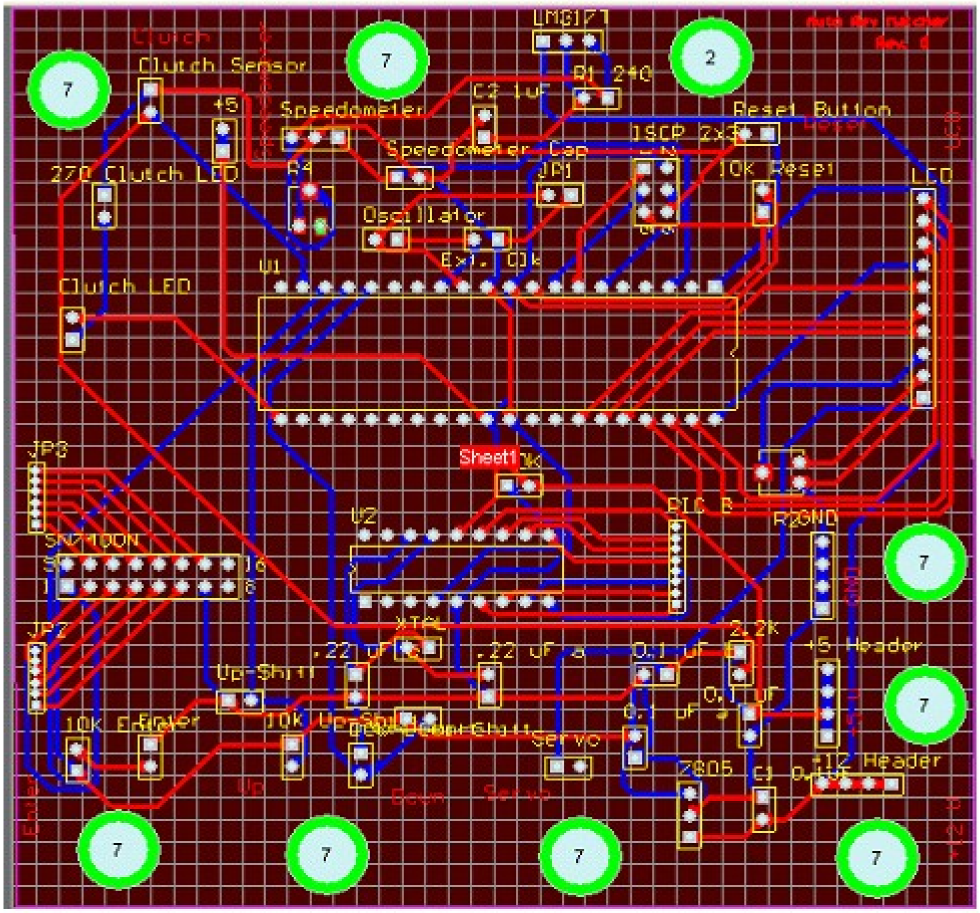
XI.Gantt Chart

Task Name & Assignment	Start Date	Planned	Extension	Downtime
Introduction / Project Proposal - B	7-Jan-08	7	0	0
Research / Abstract - B & M	9-Jan-08	10	0	0
Preliminary Design Report - B & M	12-Jan-08	16	0	0
Research & Data Gathering - B & M	28-Jan-08	14	0	0
System Level Design - B	11-Feb-08	10	10	0
Circuit Design & Purchase Parts - B & M	11-Feb-08	21	10	0
Software Atmega128 - B	17-Feb-08	32	0	0
Breadboard Test / Troubleshoot - B & M	24-Feb-08	16	0	0
Protel - M	5-Mar-08	10	0	0
PCB & Populate - M	15-Mar-08	5	0	0
Software Atmega32 - M	20-Mar-08	12	0	0
Physical Apparatus Construction - B & M	2-Apr-08	14	0	0
Test Validation - B & M	14-Apr-08	14	0	0
Report Write Up / Demo - B & M	18-Apr-08	10	0	0



Appendix A. Diagrams

PCB Layout



Appendix B. Software

```
;Demo Code
;Brad Atherton, Monique Mennis
;Sr Design EEL 4914
;4-18-08


;port a for atmega128
.equ porta = $3B
.equ ddra = $3A
;port b for atmega 128
.equ portb = $38
.equ ddrb = $37
;port d for atmega 128
.equ portd      = $32
.equ pind       = $30
.equ ddrd       = $31
;port e for atmega 128
.equ porte      = $23
.equ pine       = $21
.equ ddre       = $22
;port f for atmega 128
.equ portf      = $62
.equ pinf       = $20
.equ ddrf       = $61
;never put below $46 on atmega128 b/c interrupt handlers
.equ strings = $60
.equ main = $80
.equ data_variables = $100
;stack for atmega128
.equ sph = $5E
.equ spl = $5D
.equ stack_h = $10
.equ stack_l = $FF
;timer 1 equates
.equ TCCR1B = $4E
.equ TCNT1L = $4C
.equ TCNT1H = $4D
.equ TIFR       = $56
.equ OC1AL      = $4A
.equ OC1AH      = $4B
;timer 3 equates
.equ TCNT3H = $89
.equ TCNT3L = $88
.equ OC3AH = $87
.equ OC3AL = $86
.equ TCCR3A = $8B
.equ TCCR3B = $8A
.equ ETIFR = $7C
;timer 0 equates
.equ TIMSK      = $57
.equ TCNT0      = $52
.equ TCCR0      = $53   ;bit 2,1,0 = 000 for 1024 prescaler
;output compare equates
.equ OC0        = $51
;external interrupt equates
.equ EICRA      = $6A
```

```
.equ EICRB      = $5A
.equ EIMSK      = $59
.equ EIFR       = $58
;status register equate
.equ SREG       = $5F
;distance equate
.equ d_UB       = $a8
.equ d_LB       = $2b
;.equ d_UB      = $FC
;.equ d_LB      = $52
;servo equates
;.equ neutral = 27400
.equ offset = 0
.equ neutral = 27500
.equ    rpm_0   =       0
.equ    rpm_1   =       11
.equ    elevenhundred = 27350
.equ    rpm_2   =       12
.equ    twelvehundred = 27338
.equ    rpm_3   =       13
.equ    thirteenhundred = 27325
.equ    rpm_4   =       14
.equ    fourteenhundred = 27316
.equ    rpm_5   =       15
.equ    fifteenhundred = 27308
.equ    rpm_6   =       16
.equ    sixteenhundred = 27300
.equ    rpm_7   =       17
.equ    svnteenhundred = 27280
.equ    rpm_8   =       18
.equ    ateteenhundred = 27256
.equ    rpm_9   =       19
.equ    nineteenhundred = 27242
.equ    rpm_10  =       20
.equ    twenty = 27225
.equ    rpm_11  =       21
.equ    twentyone = 27200
.equ    rpm_12  =       22
.equ    twentytwo = 27125
.equ    rpm_13  =       23
.equ    twentythree = 27100
.equ    rpm_14  =       24
.equ    twentyfour = 27090
.equ    rpm_15  =       25
.equ    twentyfive = 27080
.equ    rpm_16  =       26
.equ    twentysix = 27050
.equ    rpm_17  =       27
.equ    twentysvn = 27042
.equ    rpm_18  =       28
.equ    twentyate = 27034
.equ    rpm_19  =       29
.equ    twentynine = 27025
.equ    rpm_20  =       30
.equ    thirty = 27000
.equ    rpm_21  =       31
.equ    thirtyone = 26950
.equ    rpm_22  =       35
.equ    thirtyfive = 26900
```

```
;.equ   thirtyfive = 23710
;rpm calculator equates
.equ first_num  = 31
.equ first_den  = 10
.equ second_num = 20;WAS 9
.equ second_den = 7
.equ third_num  = 35
.equ third_den  = 43
.equ fourth_num = 27
.equ fourth_den = 50
.equ fifth_num  = 9
.equ fifth_den  = 21



.def XL = r26
.def XH = r27
.def ZL = r30
.def ZH = r31

.dseg
        .org data_variables
oflo_cntr:
                .byte 1
start_up:
                .byte 1
edgecounter:
                .byte 1
Speed_H:                ;tens digit to be converted to ascii
        .byte 1
Speed_L:                ;ones digit to be converted to ascii
        .byte 1
Gear:                        ;ones digit to be converted to ascii
        .byte 1
RPM_Char:                ;already in ascii just a black box
        .byte 1
RS:
        .byte 1
LCDbyte:
        .byte 1
Speed_String_Count:
        .byte 1
Gear_String_Count:
        .byte 1
RPM_String_Count:
        .byte 1
RPM_Bar_Number:
        .byte 1
RPM_Row_Count:
        .byte 1
inner_delay:
        .byte 1
outer_delay:
        .byte 1
edge1_L:
                .byte 1
edge1_H:
                .byte 1
edge2_L:
```

```
                .byte 1
edge2_H:
                .byte 1
speed:
                .byte 1
speed1:
                .byte 1
speed2:
                .byte 1
speed3:
                .byte 1
speed4:
                .byte 1
speed5:
                .byte 1
speed6:
                .byte 1
speed_dec:
                .byte 1
Hexbyte:
                .byte 1
Decbyte:
                .byte 1
Num_H:  ;numerator input variable for Division subroutine
        .byte 1
Num_L:  ;numerator input variable for Division subroutine
        .byte 1
Den_H:  ;denominator input variable for Division subroutine
        .byte 1
Den_L:  ;denominator input variable for Division subroutine
        .byte 1
first_rpm:      ;variables for the rpm calculator subroutine
        .byte 1
second_rpm:
        .byte 1
third_rpm:
        .byte 1
fourth_rpm:
        .byte 1
fifth_rpm:
        .byte 1
quotient:
        .byte 1
sreg_temp:
        .byte 1



;----------INTERRUPT VECTORS------------
;----------INTERRUPT VECTORS------------
.cseg
        .org    $0000
        jmp             main

        .org    $0002
        jmp             downshift_interrupt     ;int0

        .org    $0004   ;
        jmp             upshift_interrupt       ;int1
```

```
        .org    $0008   ;int 3 interrupt
        RETI

        .org    $0010
        jmp             speed_interrupt ;int7

        .org    $0018   ;timer1 OC A Match
        jmp             tmr0_OC ;timer 0 output compare

        .org    $001A   ;timer1 OC B Match
        reti

        .org    $001C   ;timer 1 O'flow Interrupt
        jmp             tmr0_oflo

        .org    $001E
        jmp             tmr0_OC ;timer 0 output compare

        .org    $0020   ;TIMER 0 OFLOW INTERRUPT VECTOR
        jmp             tmr0_oflo
;^^^^^^^^INTERRUPT VECTORS^^^^^^^^^
;^^^^^^^^INTERRUPT VECTORS^^^^^^^^^


        .org    strings
speed_str:
        .db     "Speed: ", $D
gear_str:
        .db     "Gear: ", $D
enter_str:
        .db "Enter?  ", $D

        .org    main
;--------main program----------------
;---initialize stack
            ldi r23, stack_h
        sts sph, r23
        ldi r23, stack_l
        sts spl, r23
;---set port b0 for output
                ldi     r16, 0b00000001
                sts     ddrb, r16
;---set port f0 for output for PWM wave
                ldi     r16, 0b00000001
                sts     ddrf, r16
;---set port e for input since speed and enter
;---are connected to E7 and E0
                ldi     r16, 0
                sts     ddre, r16
;---set port d for input to read clutch from D3
                ldi     r16, 0
                sts     ddrd, r16
;---initialize delay paramters--------
        ldi r23, 0xff       ;initialize inner_delay parameter
        sts inner_delay, r23
        ldi r23, 0xff       ;initalize outer_delay parameter
        sts outer_delay, r23
                call delay_sub
```

```
;---initialize timer 1 prescalers for servo
                ;clear the overflow counting variable
                clr     r16
                sts     oflo_cntr, r16
                ;set prescalers to 001 (1 divider)
                lds     r16, tccr1b
                andi r16, 0b11111000
                ori     r16, 0b00000001
                sts     tccr1b, r16
;---clear timer 1 and set OC to neutral for servo-----
                clr     r18
                sts     tcnt1h, r18
                sts     tcnt1l, r18
                ldi     r18, high(neutral)
                sts     OC1AH, r18
                ldi     r18, low(neutral)
                sts     OC1AL, r18
;---enable timer 0 OC and oflow interrupts
;---intialize output compare and overflow interrupts for timer 1
                ;bit 4 of TIMSK = OCIE1A
                ;bit 2 of TIMSK = TOIE1
                lds     r16, timsk
                andi r16, 0b11101011
                ori     r16,  0b00010100
                sts     TIMSK, r16
;----------------------------
        call speed_disable
        call disable_clicker
        sei     ;olny pwm is enabled


;-----CHECKPOINT-----;
  ;---CHECKPOINT---;
hereee:
                ldi r16, 1
                sts portb, r16
                lds r16, pine
                andi r16, 0b00000001
                clz
                cpi     r16, 1
                breq hereee
  ;---CHECKPOINT---;
;-----CHECKPOINT-----;

                ;TURN OFF LED
                clr     r16
                sts     portb, r16

                ;DELAY FOR SHITS N GIGGLES--;
                ldi     r16, $FF                                ;
                sts     inner_delay, r16                ;
                sts     outer_delay, r16                ;
                ldi     r17, 15                                 ;
rpt:                                                            ;
                call delay_sub                          ;
                dec     r17                                             ;
                clz                                                     ;
                cpi     r17, 0                                  ;
                brne rpt                                        ;
```

```
                ;END OF DELAY---------------;

                ;


;---initialize LCD screen-------------
                call LCD_init
;---initialize clicker interrupts
                ;set int 1 and 0 to falling edge trigger
                ;bits 1,0 and 3.2 in EICRA to 1,0
                lds     r16, EICRA
                andi r16, 0b11111010
                ori     r16,  0b00001010
                sts     EICRA, r16
;---intialize start_up variable
                ldi     r16, $FF
                sts     start_up, r16
;--set-gear-and-speed-to-zero
                clr     r16
                sts     gear, r16
                sts speed, r16
;---TESTCODESTARTSHERE----------------------------

        ;the only interrupt that should be enabled
        ;are the PWM interrupts

;       jmp     hereee
        ;CLUTCH PRESSED?
        LDS     r16, pind
        andi r16, 0b00001000
        clz
        cpi     r16, 0  ;if clutch is pressed the z flag will be true (bit 1 of
sreg)
        lds     r16, sreg
        sbrc r16, 1
        call clutch_subroutine
        ;clutch is not pressed, check start up
        ;turn off servo
                ldi     r18, high(neutral)
                sts     OC1AH, r18
                ldi     r18, low(neutral)
                sts     OC1AL, r18
        lds     r16, start_up
        clz
        cpi     r16, $FF
        brne normal_prog
        ;if still in startup condition, use the clicker switches and enter
        ;to make sure current gear is acquired.  Do not pass until enter has
        ;been pressed
        call clear_screen
        call send_enter
        call send_gear
check_enter2:                   ;
                                ----
        call enable_clicker     ;now PWM and clicker are enabled
        lds r16, pine           ;
                                ----
        andi r16, 0b00000001    ;the interrupt from the clicker will----
        clz
```

```
        cpi     r16, 0  ;occur during this loop.
    ;------------------------
        ----
        brne    check_enter2    ;routine sets the appropriate value-
;this point is only reached after the enter key is pressed
;clear the startup variable
        clr     r16
        sts     start_up, r16
;-----clear the lcd of the enter string
        call clear_screen

;NORMAL-PROGRAM-OPERATION----------------------------
normal_prog:

        ;CLUTCH PRESSED?
        LDS     r16, pind
        andi r16, 0b00001000
        clz
        cpi     r16, 0  ;if clutch is pressed the z flag will be true (bit 1 of
sreg)
        lds     r16, sreg
        sbrc r16, 1
        call clutch_subroutine


        ;CLUTCH ISNT PRESSED
        ;get current speed, REMEMBER, PWM INTERRUPTS ARE ENABLED BUT CLICKER INTS
                                                ;ARENT BECAUSE THEY ARE DISABLED AT
THE END OF THEIR ihr
                ;turn off servo
                ldi     r18, high(neutral)
                sts     OC1AH, r18
                ldi     r18, low(neutral)
                sts     OC1AL, r18
        call disable_clicker
        ;get current speed
        call get_speed  ;the speed interrupt enable is in the get_speed subroutine
        call speed_disable
        ;calculate the rpm
        call rpm_calc

        ;CLUTCH PRESSED?
        LDS     r16, pind
        andi r16, 0b00001000
        clz
        cpi     r16, 0  ;if clutch is pressed the z flag will be true (bit 1 of
sreg)
        lds     r16, sreg
        sbrc r16, 1
        call clutch_subroutine

        ;turn off servo
                ldi     r18, high(neutral)
;               sts     OC1AH, r18
                ldi     r18, low(neutral)
;               sts     OC1AL, r18
        ;CLUTCH ISNT PRESSED, SEND SPEED OUT TO LCD
        call send_speed
        ;TEST CODE FOR THE SERVO:
```

```
        ;START AT THE NEUTRAL AND INCREMENT UPWARD TO THE MAX AND THEN BACK DOWN

        ;call set_servo

                ldi     r16, $FF
                sts     inner_delay, r16
                sts     outer_delay, r16
                ldi     r17, 10
rpt2:
                call delay_sub
                dec     r17
                clz
                cpi     r17, 0
                brne rpt2

        ;end
        jmp normal_prog



;-Get-Speed-Subroutine------------
get_speed:
        ;set bits 7,6 to 01 in EICRB (for transition interrupt)
        lds     r16, EICRB
        ori     r16, 0b11000000 ;11 for rising edge
        sts     EICRB, r16
        ;set timer3 prescalers to 1024 (bits 2,1,0 in the TCCR1B)
        lds     r16, TCCR3B     ;to 101
        andi r16, 0b11111000
        ori     r16, 0b00000101
        sts     TCCR3B, r16
        ;clear the edge counter
        clr     r16
        sts     edgecounter, r16
        ;TEST POINT-MAKE SURE THE TIMER WAS RESET
        lds     r16, tcnt3l
        lds     r17, tcnt3h
        ;check the edge counter
        ;if 2nd edge hasnt been captured,
        ;dont continue
        ;set bit 7 in EIMSK (to enable int 7 interrupt)

check_edge:
        call speed_enable
        sei
        ;if timer overflows, set speed to 0
        ;timer 1 overflow flag: bit 2 of tifr
        lds     r16, ETIFR
        sbrc r16, 2
        jmp     zero_speed
        lds     r16, edgecounter
        clz
        cpi     r16, 2
        brne    check_edge
        ;now two edges have been captured
        ;assume 16 bit time values for each,
        ;edge1_h, edge1_l, and edge2_h, edge2_l
```

```
        ;clear port b so i know it made it to this point
        clr     r16
        sts portb, r16

        ;edge 2 l and h contain the time difference
        ;between edges.  No subtraction is necessary
        ;since the timer was initialized at zero for edge1
        lds     r16, edge2_l
        lds     r17, edge2_h
        ;now the time difference is in r17 and r16
        ;divide the FC52 by the time difference
        ldi     r19, d_UB       ;distance upper byte    ;.equ d_UB = $FC
        ldi     r18, d_LB       ;distance lower byte    ;.equ d_LB = $52
        clr     r20     ;clear the subtraction counter
in_sub:
        ;first check if the value is 00.  if so, go to zero speed
;------------------------
;       clz
;       cpi r16, 0
;       lds     r21, sreg
;       sbrc r21, 1
;       cpi     r17, 0
;       sbrc r21, 1
;       jmp zero_speed
        ;subtract the lower bytes
;       inc     r20     ;increment the counter
;       sec     ;clear the carry flag first
;       sub     r18, r16
;       ;including the carry, subtract the higher bytes
;       sbc     r19, r17
;       ;check the carry flag, if not true, keep subtracting
;;----------------------
        ;subtract the lower bytes
        inc     r20     ;increment the counter
        clc     ;clear the carry flag first
        sub     r18, r16
        ;including the carry, subtract the higher bytes
        sbc     r19, r17
;       ;check the carry flag, if not true, keep subtracting
        brcc    in_sub
;SUBTRACTION-COMPLETE
        dec     r20
        lsr     r20
        lsr r20
        ;TEST LINE TO SEE LARGER SPAN OF SPEED.  do NOT PUT THIS IS FINAL CODE!
;       lsl r20
;       lsl r20
;DIVISION-COMPLETE
        ;r20 containS the integer quotient
                ;check if r20 is greater than decimal
        clc     ;clear the carry
        cpi     r20, $63        ;check if r20 is greater than 99
        ;if the carry is low, r20 is greater than 99
        ;and needs to be corrected
        brcs store_speed ;branch if carry is set
        ;if r20 is greater than 99, correct it
        ;to 99
        ldi     r20, $63
store_speed:
```

```
        sts     speed, r20
        ;NOTE: the value of speed in mph is stored in r20
        ;-END-OF-DIVISION-TECHNIQUE-----------------
        ret
zero_speed:
        ;clear the timer overflow flag
        ldi     r16, 0b00000100
        sts etifr, r16
        clr     r20
        sts     speed, r20
        ret

;       lds     r16, etifr
;       andi r16, 0b11111011
;       ori      r16, 0b00000100
;       sts etifr, r16
;       clr     r20
;       sts     speed, r20
;       ret
;
;end of get speed subroutine------------------


;-1--INTIALIZE-LCD-SUBROUTINE--------
LCD_init:
        ;STEP1: Enable PORTA(lower 6 pins)
        ldi        r23, ddra        ;
        ori        r23, 0x7F        ;
        sts        ddra, r23        ;PORTA 5-0 = R/W | RS | DB7 | DB6 | DB5 | DB4
                                        ;delay for 15ms to allow VCC to settle
        ldi r23, 200        ;200 x 75 x 1us = 15ms
        sts inner_delay, r23
        ldi r23, 75                ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 15 ms
        ;-------------------------
        ;STEP2: Enable 4-bit Mode

        ;remember, when writing to the LCD, first E,RW, & RS
        ;are low, then E goes high (no change to RW or RS) and the
        ;valid data is placed on db7:4, then E goes low again
        ;timing specs:
        ;RW must fall low first, with at least 150 ns before E goes high
        ;then the data must be on the line for at least 195 ns before E
        ;goes low, then the data must also remain on the line for at least
        ;10 ns after E goes low

        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x03        ;RS = 0, RW = 0, DB = 3
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
                call delay_sub        ;delay 1 us
            ;clear enable bit
```

```
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 200        ;200 x 25 x 1us = 5ms
        sts inner_delay, r23
        ldi r23, 25                 ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 5 ms
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x03        ;RS = 0, RW = 0, DB = 3
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 100        ;100 x 1 x 1us = 100us
        sts inner_delay, r23; outer_delay already set to 1
        call delay_sub        ;delay 100 us
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x03        ;RS = 0, RW = 0, DB = 3
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 200        ;200 x 25 x 1us = 5ms
        sts inner_delay, r23
        ldi r23, 25                 ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 5 ms
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x02        ;RS = 0, RW = 0, DB = 2
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 40                 ;40 x 1 x 1us = 40us
        sts inner_delay, r23; outer_delay already set to 1
        call delay_sub        ;delay 40 us
```

```
        ;------------------------
        ;STEP3: Enable 2 lines
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x02        ;RS = 0, RW = 0, DB = 2
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 200        ;200 x 25 x 1us = 5ms
        sts inner_delay, r23
        ldi r23, 25                ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 5 ms
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x08        ;RS = 0, RW = 0, DB = 8
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 40                ;40 x 1 x 1us = 40us
        sts inner_delay, r23; outer_delay already set to 1
        call delay_sub        ;delay 40 us
        ;------------------------
        ;STEP4: Diplay on, Cursor on, Blink on
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x00        ;RS = 0, RW = 0, DB = 0
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 200        ;200 x 25 x 1us = 5ms
        sts inner_delay, r23
        ldi r23, 25                ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 5 ms
        ldi r23, 0x01
```

```
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x0F        ;RS = 0, RW = 0, DB = F
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 40                ;40 x 1 x 1us = 40us
        sts inner_delay, r23; outer_delay already set to 1
        call delay_sub        ;delay 40 us
        ;-------------------------
        ;STEP4: Clear screen, Cursor home
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x00        ;RS = 0, RW = 0, DB = 0
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 200        ;200 x 25 x 1us = 5ms
        sts inner_delay, r23
        ldi r23, 25                ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 5 ms
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x01        ;RS = 0, RW = 0, DB = 1
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 82                ;82 x 20 x 1us = 1.64ms
        sts inner_delay, r23
        ldi r23, 20                ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 1.64 ms
                                        ;END of LCD_init subroutine
        ret
;-------------------------------------------------
```

```
;CLEAR LCD SCREEN SUBROUTINE---------------------------------
clear_screen:
        push r23
        push r25
        ;------------------------------------------
        ;------------inserted from lcd_init to clear screen
        ;-------------------------------------------        ;STEP4: Clear screen,
Cursor home
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x00        ;RS = 0, RW = 0, DB = 0
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 200        ;200 x 25 x 1us = 5ms
        sts inner_delay, r23
        ldi r23, 25                ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 5 ms
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        ldi r23, 0x01        ;RS = 0, RW = 0, DB = 1
        sts porta, r23
        call delay_sub        ;delay 1 us
                ;set enable bit high
                ori     r23, 0b01000000
                sts     porta, r23
        call delay_sub        ;delay 1 us
            ;clear enable bit
        andi r23, 0b10111111
                sts     porta, r23
        ldi r23, 82                ;82 x 20 x 1us = 1.64ms
        sts inner_delay, r23
        ldi r23, 20                ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub        ;delay 1.64 ms

        ;-----------------------------------------
        ;--end of insertion------------------------
        ;------------------------------------------
                pop r25
                pop r23
                ret
;--end of clear lcd subroutine

;-2--NIBBLE-PASSER-SUBROUTINE--------
nibbler_passer:
        ;check if the byte is for data or command
        lds r23, RS                ;load RS parameter value into r23
```

```
        lds r22, LCDbyte;load the byte for the LCD into r22
        swap r22                ;swap upper & lowe nibble
                andi r22, 0b00001111
                ;skip the next instruction if RS = 1
                sbrs r23, 0
                jmp data_upper_nib      ;this line is only executed when RS = 0
                ;otherwise, RS is 1, so set the RS bit
                ;in the upper nibble
                ori     r22, 0b00010000 ;the RS bit has just been set
data_upper_nib:
        sts porta, r22          ;send the 1st (upper) nibble to LCD
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        clz
        call delay_sub          ;delay 1 us

        ori     r22, 0b01000000         ;set enable bit high
                sts     porta, r22
        call delay_sub          ;delay 1 us

        andi r22, 0b10111111  ;clear enable bit
                sts     porta, r22

        ldi r23, 200          ;200 x 10 x 1us = 2ms
        sts inner_delay, r23
        ldi r23, 10                 ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
        call delay_sub          ;delay 2 ms

;load the lower nibble and check the RS bit
        ;check if the byte is for data or command
        lds r23, RS                 ;load RS parameter value into r23
        lds r22, LCDbyte;load the byte for the LCD into r22
        andi r22, 0b00001111
                ;skip the next instruction if RS = 1
                sbrs r23, 0
                jmp data_lower_nib      ;this line is only executed when RS = 0
                ;otherwise, RS is 1, so set the RS bit
                ;in the upper nibble
                ori     r22, 0b00010000 ;the RS bit has just been set
data_lower_nib:
        sts porta, r22          ;send the 2nd (lower) nibble to LCD
        ldi r23, 0x01
        sts inner_delay, r23
        sts outer_delay, r23
        clz
        call delay_sub          ;delay 1 us
        ori     r22, 0b01000000   ;set enable bit high
                sts     porta, r22
        call delay_sub          ;delay 1 us
        andi r22, 0b10111111  ;clear enable bit
                sts     porta, r22
        ldi r23, 200          ;200 x 10 x 1us = 2ms
        sts inner_delay, r23
        ldi r23, 10                 ;set inner_delay to largest number to make more
accurate
        sts outer_delay, r23
```

```
        call delay_sub        ;delay 2 ms
                                        ;END of 2nd nibble byte has been sent
                                        ;END of nibble_passer subroutine
        ret
;-------------------------------------------------

;-SEND-SPEED-TO-LCD-SUBROUTINE-------------------
send_speed:
;--send out speed characters to the LCD---

        ;convert the speed value to dec
        lds     r16, speed
        sts     Hexbyte, r16
        call Hex_2_Dec
        lds     r16, Decbyte
        sts     speed_dec, r16
        call clear_screen

                ;initialize the Z pointer for
                ;where the string is in prog memory
        ldi     ZH, high(speed_str<<1)
        ldi     ZL, low(speed_str<<1)
                ;dont forget to update the RS bit
        ldi     r23, 01
        sts     RS, r23

                ;load the character
send_byte:
        lpm     r23, Z
                ;check if its the end line
        clz
    cpi r23, $D
        breq    end_of_string
                ;otherwise (if not end), send character to
                ;the LCD screen
        STS     LCDbyte, r23
        call Nibbler_passer
                ;since it is not the end character,
                ;increment the pointer and go back
                ;to the load and send instructions
        inc     ZL
        jmp send_byte
end_of_string:
        ;send out speed to the LCD
        lds     r16, speed_dec  ;remember this must be the speed variable
        mov     r17, r16                ;that has been converted from hex to dec
        swap r16
        andi r16, 0b00001111
        ldi     r18, $30
        add     r16, r18        ;add 30 to format it in ascii
        sts     LCDbyte, r16
        call Nibbler_passer     ;send out the units characters
        andi r17, 0b00001111
        add     r17, r18        ;add 30 to format it in ascii
        sts     LCDbyte, r17
        call Nibbler_passer;send out the tens character
        ;exit
        ret
;end-of-send-speed-subroutine----------------------
```

```
;--SEND-GEAR-TO-LCD-SUBROUTINE-------------------------
send_gear:
;clear the LCD screen
        ;load the z pointer with the gear string address
        ldi     ZL, low(gear_str<<1)
        ldi     ZH, high(gear_str<<1)
        ;dont forget to update the RS bit
        ldi     r23, 01
        sts     RS, r23


        ;send the character
                ;load the character
send_byte2:
        lpm     r16, Z
                ;check if its the end line
        clz
    cpi r16, $D
        breq    end_of_string2
                ;otherwise (if not end), send character to
                ;the LCD screen
        STS     LCDbyte, r16
        call Nibbler_passer
                ;since it is not the end character,
                ;increment the pointer and go back
                ;to the load and send instructions
        inc     ZL
        jmp send_byte2
end_of_string2:
        lds     r16, gear
        ldi     r17, $30
        add     r16, r17
        sts     LCDbyte, r16
        call Nibbler_Passer
        ;exit
        ret
;-----end of send_gear subroutine---------------
;-SEND-ENTER-TO-LCD-SUBROUTINE-------------------
send_enter:
;--send out "Enter?" characters to the LCD---

   ;STEP4: Clear screen, Cursor home
;   ldi r23, 0x01
 ;  sts inner_delay, r23
  ; sts outer_delay, r23
   ;ldi r23, 0x00        ;RS = 0, RW = 0, DB = 0
 ;  sts porta, r23
;   call delay_sub        ;delay 1 us
;        ;set enable bit high
;        ori        r23, 0b01000000
;        sts        porta, r23
;   call delay_sub        ;delay 1 us
;   ;clear enable bit
;   andi r23, 0b10111111
;        sts        porta, r23
;   ldi r23, 200        ;200 x 25 x 1us = 5ms
;   sts inner_delay, r23
;   ldi r23, 25                ;set inner_delay to largest number to make more
```

```
accurate
;   sts outer_delay, r23
;   call delay_sub        ;delay 5 ms
;   ldi r23, 0x01
;   sts inner_delay, r23
;   sts outer_delay, r23
;   ldi r23, 0x01        ;RS = 0, RW = 0, DB = 1
;   sts porta, r23
;   call delay_sub        ;delay 1 us
;        ;set enable bit high
;        ori        r23, 0b01000000
;        sts        porta, r23
;   call delay_sub        ;delay 1 us
;   ;clear enable bit
;   andi r23, 0b10111111
;        sts        porta, r23
;   ldi r23, 82                ;82 x 20 x 1us = 1.64ms
;   sts inner_delay, r23
;   ldi r23, 20                ;set inner_delay to largest number to make more
accurate
;   sts outer_delay, r23
;   call delay_sub        ;delay 1.64 ms

        ;-----------------------------------------
        ;--end of insertion------------------------
        ;------------------------------------------

                ;initialize the Z pointer for
                ;where the string is in prog memory
        ldi        ZH, high(enter_str<<1)
        ldi        ZL, low(enter_str<<1)
                ;dont forget to update the RS bit
        ldi        r23, 01
        sts        RS, r23

                ;load the character
send_byte3:
        lpm        r23, Z
                ;check if its the end line
        clz
   cpi        r23, $D
        breq        end_of_string3
                ;otherwise (if not end), send character to
                ;the LCD screen
        STS        LCDbyte, r23
        call Nibbler_passer
                ;since it is not the end character,
                ;increment the pointer and go back
                ;to the load and send instructions
        inc        ZL
        jmp send_byte3
end_of_string3:
        ;"Enter? " has been sent to the LCD
        ;exit
        ret
;end-of-send-enter-subroutine----------------------
```

```
;-6--DELAY-SUBROUTINE--------
delay_sub:
                PUSH R24
                PUSH R25
        lds        r24, outer_delay
outer_top:
        ;the inner_delay variable is the number of
        ;1uS repitions to be competed
        lds        r25, inner_delay        ;2 cycles        :2
inner_top:        ;(we want 16 clock cycles total between here and the branch)
        ;-----------------------------------------
        dec        r25                                          ;1 cycle        :1
-        -
        nop                                                  ;1 cycle        :2
- 1        -
        nop                                                  ;
:3        -        -
        nop                                                  ;
:4        - u        -
        nop                                                  ;
:5        - S        -
        nop                                                  ;
:6        -        -
        nop                                                  ;
:7        - s        -
        nop                                                  ;
:8        - e        -
        nop                                                  ;
:9        - q        -
        nop                                                  ;
:10        - u        -
        nop                                                  ;
:11        - e -
        nop                                                  ;
:12        - n        -
        nop                                                  ;
:13        - c        -
        nop                                                  ;
:14        - e        -
        cpi        r25, 0                                  ;                         :
15        -        -
        brne        inner_top                ;1 cycle        :16        -        -
        ;----end-of-1uS-sequence--------------------
        clz        ;clear the Z flag
        dec        r24
        cpi        r24, 0
        brne        outer_top
        clz

                POP R25
                POP R24
        ret
;-------------------------------------------------

;Hex-To-Decimal-Conversion-Subroutine------------
Hex_2_Dec:
        ;assume the input variable is called "Hexbyte"
        ;and is located in data space.
```

```
        lds     r16, Hexbyte
        ;check if hexbyte is zero
        clz
        cpi     r16, 0
        breq zero_hex

        ldi     r17, 10
        clr     r18     ;use r17 to count (the integer quotient)
                ;formula: divide Hexbyte by 10, then add 6x
                ;that number to Hexbyte
subtract:
        clc     ;clear the carry flag beforehand
        sub     r16, r17
        inc     r18
        ;check if r16 is less than 0 (the carry goes true)
        ;if carry is not true, increment the counter and
        ;and go back to subtract
        lds     r19, SREG
        sbrs r19, 0     ;if the carry is true, skip the next instruction
        jmp     subtract
        dec     r18     ;decrement r18 since it is pre-incremented before
                        ;the condition test
        ;now the integer quotient is in r18
        ;MULTIPLY R18 by 6, R19 can be used since the carry test is over
        ldi     r19, 6  ;r18 * r19 = quotient * 6
        mul     r18, r19        ;resultant is in r1(high) r0 (low)
                        ;the product will be a 1 Byte number, only care
                        ;about the low byte R0
        ;R0 contains the product.  Add R0 to Hexbyte
        lds     r16, Hexbyte
        add r16, r0
        ;result is in r16
        sts     Decbyte, r16
        ;exit
        ret
zero_hex:
        ldi r16, $00
        sts Decbyte, r16
        ret
;-------End of Hex to Decimal Subroutine----------------------



;--RPM CALCULATING SUBROUINTE----------------------------------
RPM_Calc:
        push r16
        push r17
;first_gear:
        lds     r16, speed
        ;check if speed = 0
        clz
        cpi     r16, 0
        lds     r16, sreg
        sbrc r16, 1
        jmp speediszero
        ;multiply speed by the 1st gear factor
        ldi     r17, first_Num
        mul     r16, r17
        ;result is in r1, r0
```

```
        ;store the results in the Numerator variables
        ;for the division subroutine
        sts     Num_H, r1
        sts     Num_L, r0
        ;load and store the denominator for the division subroutine
        ldi     r16, first_den
        sts Den_L, r16
        clr     r16
        sts     Den_H, r16
        ;divide to calculate the RPM
        call Div_Sub
        ;the RPM is returned in variable 'quotient'
        lds     r16, quotient   ;rpm is in r16
        sts     first_rpm, r16
;second_gear:
        lds     r16, speed
        ;multiply speed by the 2st gear factor
        ldi     r17, second_num
        mul     r16, r17
        ;result is in r1, r0
        ;store the results in the Numerator variables
        ;for the division subroutine
        sts     Num_H, r1
        sts     Num_L, r0
        ;load and store the denominator for the division subroutine
        ldi     r16, second_den
        sts Den_L, r16
        clr     r16
        sts     Den_H, r16
        ;divide to calculate the RPM
        call Div_Sub
        ;the RPM is returned in variable 'quotient'
        lds     r16, quotient   ;rpm is in r16
        sts     second_rpm, r16
;third_gear:
        lds     r16, speed
        ;multiply speed by the 3rd gear factor
        ldi     r17, third_num
        mul     r16, r17
        ;result is in r1, r0
        ;store the results in the Numerator variables
        ;for the division subroutine
        sts     Num_H, r1
        sts     Num_L, r0
        ;load and store the denominator for the division subroutine
        ldi     r16, third_den
        sts Den_L, r16
        clr     r16
        sts     Den_H, r16
        ;divide to calculate the RPM
        call Div_Sub
        ;the RPM is returned in variable 'quotient'
        lds     r16, quotient   ;rpm is in r16
        sts     third_rpm, r16
;fourth_gear:
        lds     r16, speed
        ;multiply speed by the 2st gear factor
        ldi     r17, fourth_num
        mul     r16, r17
```

```
        ;result is in r1, r0
        ;store the results in the Numerator variables
        ;for the division subroutine
        sts     Num_H, r1
        sts     Num_L, r0
        ;load and store the denominator for the division subroutine
        ldi     r16, fourth_den
        sts Den_L, r16
        clr     r16
        sts     Den_H, r16
        ;divide to calculate the RPM
        call Div_Sub
        ;the RPM is returned in variable 'quotient'
        lds     r16, quotient   ;rpm is in r16
        sts     fourth_rpm, r16
;fifth_gear:
        lds     r16, speed
        ;multiply speed by the 2st gear factor
        ldi     r17, fifth_num
        mul     r16, r17
        ;result is in r1, r0
        ;store the results in the Numerator variables
        ;for the division subroutine
        sts     Num_H, r1
        sts     Num_L, r0
        ;load and store the denominator for the division subroutine
        ldi     r16, fifth_den
        sts Den_L, r16
        clr     r16
        sts     Den_H, r16
        ;divide to calculate the RPM
        call Div_Sub
        ;the RPM is returned in variable 'quotient'
        lds     r16, quotient   ;rpm is in r16
        sts     fifth_rpm, r16
        ;exit
        pop     r17
        pop r16
        ret
speediszero:
        clr r16
        sts     first_rpm, r16
        sts     second_rpm, r16
        sts third_rpm, r16
        sts fourth_rpm, r16
        sts fifth_rpm, r16
        sts     quotient, r16
        pop     r17
        pop r16
        ret
;--end of RPM-Speed calculator subroutine-----------------------

;--16-16-bit DIVISION SUBROUTINE--------------------------------
Div_Sub:
        push r16
        push r17
        push r18
        push r19
        push r20
```

```
        lds     r19, Num_H
        lds     r18, Num_L
        lds     r17, Den_H
        lds     r16, Den_L
        clr     r20     ;r20 is the subtraction counter
inc_subcounter:
        inc r20
        clc
        sub     r18, r16
        sbc     r19, r17
        brcc inc_subcounter
;carry is now true
        dec r20
        sts quotient, r20
        ;exit
        pop     r20
        pop r19
        pop r18
        pop r17
        pop r16
        ret
;-end of division subroutine----------------



;SET SERVO SUBBROUTINE------------------------------------------
set_servo:
        push r16
        push r17
        push r18
        push r19
;       ldi     r16, $5b
;       sts     OC1AH, r16
;       clr     r16
;       sts OC1AL, r16
;
;       pop     r18
;       pop r17
;       pop r16
;       ret

        lds     r16, gear
        clz
        ;check for neutral first
        cpi r16, 0
        lds     r18, sreg
        sbrc r18, 1
        jmp     pos_0                       ;z flag is true => go to pos0
        ;not in neutral, find the gear
        cpi     r16, 1
        breq load_first
        cpi     r16, 2
        breq load_second
        cpi     r16, 3
        breq load_third
        cpi     r16, 4
        breq load_fourth
        cpi     r16, 5
```

```
        breq    load_fifth
        ;otherwise, set servo to zero
                ;by setting the output compare to neutral
                ldi     r18, high(neutral)
                sts     OC1AH, r18
                ldi     r18, low(neutral)
                sts     OC1AL, r18
        ;exit
        ret
load_first:
        lds     r16, first_rpm
        jmp     find_range
load_second:
        lds     r16, second_rpm
        jmp     find_range
load_third:
        lds     r16, third_rpm
        jmp     find_range
load_fourth:
        lds     r16, fourth_rpm
        jmp     find_range
load_fifth:
        lds     r16, fifth_rpm

find_range:
        ldi     r17, rpm_1
        clc
        cp      r16, r17
        ;if carry goes true, r16 < rpm_1, set servo to neutral
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_0                           ;carry is true => go to pos0
        ;check 2nd position
        ldi     r17, rpm_2
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_1                           ;carry is true => go to pos1
        ;check 3rd position
        ldi     r17, rpm_3
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_2                           ;carry is true => go to pos2
        ;check 4th position
        ldi     r17, rpm_4
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_3                           ;carry is true => go to pos3
        ;check 5th position
        ldi     r17, rpm_5
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_4                           ;carry is true => go to pos4
        ;check 6th position
        ldi     r17, rpm_6
        cp      r16, r17
```

```
lds     r18, sreg
sbrc r18, 0
jmp     pos_5                          ;carry is true => go to pos5
;check 7th position
ldi     r17, rpm_7
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_6                          ;carry is true => go to pos6
;check 8th position
ldi     r17, rpm_8
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_7                          ;carry is true => go to pos7
;check 9th position
ldi     r17, rpm_9
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_8                          ;carry is true => go to pos8
;check 10th position
ldi     r17, rpm_10
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_9                          ;carry is true => go to pos9
;check 11th position
ldi     r17, rpm_11
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_10                         ;carry is true => go to pos10
;check 12th position
ldi     r17, rpm_12
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_11                         ;carry is true => go to pos11
;check 13th position
ldi     r17, rpm_13
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_12                         ;carry is true => go to pos12
;check 14th position
ldi     r17, rpm_14
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_13                         ;carry is true => go to pos13
;check 15th position
ldi     r17, rpm_15
cp      r16, r17
lds     r18, sreg
sbrc r18, 0
jmp     pos_14                         ;carry is true => go to pos14
;check 16th position
ldi     r17, rpm_16
```

```
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_15                          ;carry is true => go to pos15
        ;check 17th position
        ldi     r17, rpm_17
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_16                          ;carry is true => go to pos16
        ;check 18th position
        ldi     r17, rpm_18
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_17                          ;carry is true => go to pos17
        ;check 19th position
        ldi     r17, rpm_19
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_18                          ;carry is true => go to pos18
        ;check 20th position
        ldi     r17, rpm_20
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_19                          ;carry is true => go to pos19
        ;check 21st position
        ldi     r17, rpm_21
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_20                          ;carry is true => go to pos20
        ;check 22nd position
        ldi     r17, rpm_22
        cp      r16, r17
        lds     r18, sreg
        sbrc r18, 0
        jmp     pos_21                          ;carry is true => go to pos21
        ;else
        jmp             pos_22
        ;check 23rd position
;       ldi     r17, rpm_23
;       cp      r16, r17
;       lds     r18, sreg
;       sbrc r18, 0
;       jmp     pos_22                          ;carry is true => go to pos22


pos_0:
        ;set servo to neutral position
                ldi     r17, high(neutral)
                ldi     r16, low(neutral)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
```

```
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_1:
        ldi     r16, low(elevenhundred)
        ldi     r17, high(elevenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_2:
        ldi     r16, low(twelvehundred)
        ldi     r17, high(twelvehundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_3:
        ldi     r16, low(thirteenhundred)
        ldi     r17, high(thirteenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_4:
        ldi     r16, low(fourteenhundred)
        ldi     r17, high(fourteenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_5:
        ldi     r16, low(fifteenhundred)
        ldi     r17, high(fifteenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_6:
```

```
        ldi     r16, low(sixteenhundred)
        ldi     r17, high(sixteenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_7:
        ldi     r16, low(svnteenhundred)
        ldi     r17, high(svnteenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_8:
        ldi     r16, low(ateteenhundred)
        ldi     r17, high(ateteenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_9:
        ldi     r16, low(nineteenhundred)
        ldi     r17, high(nineteenhundred)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_10:
        ldi     r16, low(twenty)
        ldi     r17, high(twenty)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_11:
        ldi     r16, low(twentyone)
        ldi     r17, high(twentyone)
                ldi     r18, offset
                clc
```

```
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_12:
        ldi     r16, low(twentytwo)
        ldi     r17, high(twentytwo)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_13:
        ldi     r16, low(twentythree)
        ldi     r17, high(twentythree)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_14:
        ldi     r16, low(twentyfour)
        ldi     r17, high(twentyfour)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_15:
        ldi     r16, low(twentyfive)
        ldi     r17, high(twentyfive)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_16:
        ldi     r16, low(twentysix)
        ldi     r17, high(twentysix)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
```

```
                sts     OC1AL, r16
        jmp     end_set_servo
pos_17:
        ldi     r16, low(twentysvn)
        ldi     r17, high(twentysvn)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_18:
        ldi     r16, low(twentyate)
        ldi     r17, high(twentyate)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_19:
        ldi     r16, low(twentynine)
        ldi     r17, high(twentynine)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_20:
        ldi     r16, low(thirty)
        ldi     r17, high(thirty)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_21:
        ldi     r16, low(thirtyone)
        ldi     r17, high(thirtyone)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo
pos_22:
        ldi     r16, low(thirtyfive)
```

```
        ldi     r17, high(thirtyfive)
                ldi     r18, offset
                clc
                sub     r16, r18
                clr     r18
                sbc r17, r18
                sts     OC1AH, r17
                sts     OC1AL, r16
        jmp     end_set_servo

end_set_servo:
        pop r19
        pop r18
        pop     r17
        pop r16
        ret
;--end of set servo subrouine---------------------------------



        ;-ENABLE-CLUTCH-INTERRUPT-SUBROUTINE---------
clutch_enable:
                ;assume clutch is connected to pin d3, the int3 interrupt
                ;clear the clutch flag first
                lds     r16, EIFR
                ori     r16, 0b00001000
                sts     EIFR, r16
                ;enable the interrupt
                lds     r16, EIMSK
                ori     r16, 0b00001000
                sts     EIMSK, r16
                ret
        ;-DISABLE-CLUTCH-INTERRUPT-SUBROUTINE---------
clutch_disable:
                ;assume clutch is connected to pin d3, the int3 interrupt
                lds     r16, EIMSK
                andi r16, 0b11110111
                sts     EIMSK, r16
                ret
        ;-ENABLE SPEED INTERRUPT SUBROUTINE
speed_enable:
                ;clear the flag first
                lds     r16, EIFR
                ori     r16, 0b10000000
                sts     EIFR, r16
                ;enable the interrupt
                lds     r16, EIMSK
                ori     r16, 0b10000000
                sts     EIMSK, r16
                ret
        ;-DISABLE-SPEED-INTERRUPT-SUBROUTINE---------
speed_disable:
                ;assume clutch is connected to pin e7, the int7 interrupt
                lds     r16, EIMSK
                andi r16, 0b01111111
                sts     EIMSK, r16
                ret
        ;--enable clicker interrupts subroutine
enable_clicker:
```

```
                ;clear the clicker flags first
                lds     r16, EIFR
                ori     r16, 0b00000011
                sts EIFR, r16
                        ;ENABLE CLICKER INTERRUPTS
                ;set bits 0 and 1 in EIMSK (to enable int 0 and 1 interrupts)
                lds     r16, EIMSK
                ori     r16, 0b00000011
                sts EIMSK, r16
                ret
disable_clicker:
                ;clear the clicker flags first
                andi r16, 0b11111100
                sts     EIMSK, r16
                lds     r16, EIFR
                ori     r16, 0b00000011
                sts EIFR, r16
                lds     r16, EIMSK
                ret

;-SPEED-INTERRUPT-HANDLER--------------------------
Speed_Interrupt:
        push r16
        push r17
        push r18
        ;save the status register
        lds     r16, sreg
        sts     sreg_temp, r16
        ;FIRST DISABLE NESTED SPEED INTERRUPTS
        call speed_disable
        ;ENABLE GLOBAL INTERRUPTS
        ;FOR PWM INTERRUPTS
        SEI
        ;check which edge:
        lds     r18, edgecounter
        clz
        cpi     r18, 0
        breq edge1

edge2:
        ;clear bit 7 in EIMSK (to disable int 7 interrupt)
        ;turn off LED
        clr     r16
        sts     portb, r16
        ;if greater than FC52 OR if timer overflow occurred, set
        ;the edge time to FC52
                ;first check the overflow flag
        lds     r16, ETIFR      ;check bit 2, the overflow flag
        sbrc r16, 2
        jmp oflowed
;if overflow did not occur:
        ;load timer value
        lds     r16, TCNT3L
        lds     r17, TCNT3H
        ;check if greater than $FC52
                clc
                cpi     r17, $FC
                brlo check_out
                ;if FC is greater than or = to FC, check the lower byte
```

```
                clz
                cpi     r17, $FC
                ;r16 is greater than FC
                brne oflowed
                ;if it is FC, check the lower byte
                clc
                cpi     r16, $53
                brge oflowed
                ;otherwise, the lower byte is $52 or less
                jmp check_out


oflowed:
        ;set edge time to FC52
        ldi     r17, $FC
        ldi     r16, $52
check_out:
        ;store the timer value
        sts     edge2_H, r17
        sts edge2_L, r16
        ;increment the edge counter
        inc     r18
        sts     edgecounter, r18
        ;exit
        ;clear the external interrupt flag
        lds     r16, EIFR
        ori     r16, 0b10000000
        sts     EIFR, r16
        ;restore the sreg
        lds     r16, sreg_temp
        sts     sreg, r16
        ;exit
        pop     r18
        pop r17
        pop     r16
        reti
edge1:
        ;illuminate LED
        ldi     r16, 1
        sts     portb, r16
        ;if at first edge,
        ;reset timer 1
        clr     r16
        sts     TCNT3H, r16
        sts     TCNT3L, r16
        sts     edge1_H, r16
        sts edge1_L, r16
        ;and clear the timer overflow flag
        lds     r16, ETIFR
        sbr     r16, $04
        sts     ETIFR, r16
        ;increment the edge counter
        inc r18
        sts     edgecounter, r18
        ;clear the external interrupt flag
        ldi     r16, 0b10000000
        sts     EIFR, r16
        ;restore the sreg
        lds     r16, sreg_temp
```

```
        sts     sreg, r16
        ;exit
        pop     r18
        pop r17
        pop     r16
        reti
;--------------------------------------------------------

;---UPSHIFT INTERRUPT-----------------------------------
upshift_interrupt:
        ;NOTE: THE SWITCH USED REQUIRES A LOT OF DEBOUNCING
        ;BECUASE IT ALSO SENDS A PULSE WHEN THE MOMENTARY SWITCH
        ;IS RELEASED.  THIS CANNOT BE FULLY CORRECTED, BUT THE
        ;DELAY IS SET LONG ENOUGH FOR EVEN A LAZY FINGER (SHY OF A HALF SECOND)
        push    r16
        ;save the status register
        lds     r16, sreg
        sts     sreg_temp, r16
        ;disble clicker ints to avoid nested interrupts
        call disable_clicker
        ;4.enable global interrupts for PWM
        sei
        ;first check if at startup
        ;if at startup, also send the enter string
        ;to the lcd
        call    clear_screen
        lds     r16, start_up
        clz
        cpi     r16, $FF
        lds     r16, sreg
        sbrc    r16, 1
        call send_enter
        ;increment the gear and send it to the LCD
        lds     r16, gear
        ;check if gear is 5 before incrementing
        clz
        cpi     r16, 5
        breq send1
increment_gear:
        inc     r16
send1:
        sts     gear, r16
        call send_gear
        ;add some delay for deboucing (40 mS)
        ldi     r16, 250
        sts     inner_delay, r16
        ldi     r16, 250
        sts     outer_delay, r16
        call delay_sub
        call delay_sub
        call delay_sub
        call delay_sub
        call delay_sub
        call delay_sub
        ;clear the flag
        lds     r16, EIFR
        ori     r16, 0b00000010
        sts     EIFR, r16
        ;---------------------------
```

```
        ;restore the sreg
        lds     r16, sreg_temp
        sts     sreg, r16
        ;exit
        pop     r16
        reti
;--------------------------------

;---DOWNSHIFT INTERRUPT--------------------------------------
downshift_interrupt:
        ;NOTE: THE SWITCH USED REQUIRES A LOT OF DEBOUNCING
        ;BECUASE IT ALSO SENDS A PULSE WHEN THE MOMENTARY SWITCH
        ;IS RELEASED.  THIS CANNOT BE FULLY CORRECTED, BUT THE
        ;DELAY IS SET LONG ENOUGH FOR EVEN A LAZY FINGER (SHY OF A HALF SECOND)
        push    r16
        ;save the status register
        lds     r16, sreg
        sts     sreg_temp, r16
        ;disble clicker ints to avoid nested interrupts
        call disable_clicker
        ;4.enable global interrupts for PWM
        sei
        ;first check if at startup
        ;if at startup, also send the enter string
        ;to the lcd
        call    clear_screen
        lds     r16, start_up
        clz
        cpi     r16, $FF
        lds     r16, sreg
        sbrc    r16, 1
        call send_enter

        ;decrement the gear and send it to the lcd
        lds     r16, gear
        ;before decrementing, check if gear is zero
        clc
        cpi     r16, 0
        breq send2
decrement_gear:
        dec     r16
send2:
        sts     gear, r16
        call send_gear
        ;add some delay for debouncing
        ldi     r16, 250
        sts     inner_delay, r16
        ldi     r16, 250
        sts     outer_delay, r16
        call delay_sub
        call delay_sub
        call delay_sub
        call delay_sub
        call delay_sub
        call delay_sub


        ;clear the flag
        lds     r16, EIFR
```

```
        ori     r16, 0b000000001
        sts     EIFR, r16
        ;-----------------------------------------
        ;restore the sreg
        lds     r16, sreg_temp
        sts     sreg, r16
        ;exit
        pop     r16
        reti
;---------------------------------

;-CLUTCH-INTERRUPT----------------------------------------
Clutch_subroutine:
        push r16
        push r17
        push r18
        push r19
        push r20
        push r21
        push r22
        push r23
        push r24
        push r25
        push r26
        push r27
        push r28
        push r29
        push r30
        push r31
;check the start-up variable
        lds     r25, start_up
        clz
        cpi     r25, $FF        ;if at start-up, variable = FF

;--GO-T0-STEP-3-OF-FLOW-CHART-------------------------
        breq startup_code       ;
;if not at start up:
clutch_top:
        ;enable clicker interrupts

        call enable_clicker
        sei
        ;Find the Servo Position Value
        ;For the current Gear
;       call rpm_calc
        ;make sure the clutch is still pressed before setting the servo
        ;if clutch is not pressed anymore, exit the ihr
        lds     r16, pind
        sbrc    r16, 3  ;skip the reti if clutch is pressed (d2 = low when clutch
is pressed)
        ;clutch isnt pressed anymore
        jmp exit_clutch
        ;clutch is still pressed
        call set_servo
        ;TEST CODE: after setting servo, if speed is greater than 15, get new speed
and servo values
        lds     r16, speed
        clc
        cpi     r16, 15
```

```
        lds r17, sreg
        sbrs r17, 0
        jmp in_gear
        ;exit
        ;Clutch still pressed?
        lds     r16, pind
        sbrs    r16, 3  ;skip the reti if clutch is pressed (d2 = low when clutch
is pressed)
        ;clutch isnt pressed anymore
        jmp clutch_top
exit_clutch:
        ;restore the sreg
        lds     r16, sreg_temp
        sts     sreg, r16
        ;exit
        pop     r31
        pop r30
        pop r29
        pop r28
        pop     r27
        pop r26
        pop r25
        pop r24
        pop     r23
        pop r22
        pop r21
        pop r20
        pop     r19
        pop r18
        pop r17
        pop r16
        ret

;--STEP-3-OF-FLOW-CHART--FIRST-CLUTCH-INTERRUPT-ROUTINE---------
startup_code:           ;
                        ----
        ;enable clicker interrupts
        call clear_screen
        call send_enter
        call send_gear
        sei
        ;wait for enter button to be pressed
check_enter:                    ;
                        ----
        call enable_clicker
        lds r16, pine           ;
                        ----
        andi r16, 0b00000001    ;the interrupt from the clicker will----
        clz
        cpi     r16, 0  ;occur during this loop.  the interrupt-
    ;-------------------------
        ----
        brne    check_enter     ;routine sets the appropriate value-

;--If enter is pressed, disable the clicker and
        ;                                       CLEAR THE START-UP VARIABLE
                ----
        call disable_clicker
        call clear_screen
```

```
        lds     r16, start_up   ;
                        ----
        clr     r16                             ;
                                        ----
        sts     start_up, r16                   ;
                        ----
;read in port d2 to see if clutch is still pressed.
;if clutch is still pressed, set the servo.
;if clutch is not pressed anymore, exit the ihr
        lds     r16, pind
        sbrc    r16, 3  ;skip the reti if clutch is pressed (d2 = low when clutch
is pressed)
        ;clutch isnt pressed anymore
        jmp     exit_startup
;clutch is still pressed
        lds     r16, gear
;if gear = 1 or 0, set servo to 0
                ;use >= 2
        cln
        clv
        cpi     r16, 2
        brge    in_gear
;the car is in neutral or first gear, set servo to neutral
        ldi     r16, high(neutral)
        sts     OC1AH, r16
        ldi     r16, low(neutral)
        sts     OC1AL, r16
        ;exit
        jmp     exit_startup

in_gear:
        ;disable clicker ints
        call disable_clicker
        ;get the current speed
        call get_speed
        ;enable clickers
        call enable_clicker
        ;calculate the rpm values at each gear
        call rpm_calc
        ;make sure the clutch is still pressed before setting the servo
        ;if clutch is not pressed anymore, exit the ihr
        lds     r16, pind
        sbrc    r16, 3  ;skip the reti if clutch is pressed (d2 = low when clutch
is pressed)
        ;clutch isnt pressed anymore
        jmp     exit_startup
        ;clutch is still pressed
        call set_servo
        ;delay for a while so the driver can have a
        ;chance to press the clicker
        ldi     r16, $FF
        sts     inner_delay, r16
        sts     outer_delay, r16
        ldi     r17, 25
rpt3:
        call delay_sub
        dec     r17
        clz
        cpi     r17, 0
```

```
        brne rpt3
        ;exit the interrupt
exit_startup:
        ;check if clutch is still pressed
        ;if pressed, go to clutch top
        lds     r16, pind
        sbrs    r16, 3
        jmp     clutch_top
        ;restore the sreg
        lds     r16, sreg_temp
        sts     sreg, r16
        ;restore the registers
        pop     r31
        pop r30
        pop r29
        pop r28
        pop     r27
        pop r26
        pop r25
        pop r24
        pop     r23
        pop r22
        pop r21
        pop r20
        pop     r19
        pop r18
        pop r17
        pop r16
        ;exit
        ret
;end of start-up AND in gear section of clutch
interrupt--------------------------------


tmr0_OC:
        push r16
        ;save the status register
        lds     r16, sreg
        sts     sreg_temp, r16
        ;check the overflow counting variable: IF zero, turn off port f
        ;turn port f off
        lds     r16, oflo_cntr
        cpi     r16, 0
        brne skip_f
        clr     r16
        sts     portf, r16
skip_f:
        ;restore the sreg
        lds     r16, sreg_temp
        sts     sreg, r16
        ;exit
        pop     r16
        reti

tmr0_oflo:
        push r16
        ;save the status register
        lds     r16, sreg
        sts     sreg_temp, r16
```

```
        ;check the overflow counting variable.
        ;IF <10, increment the counter
        lds     r16, oflo_cntr
        clc
        cpi r16, 5
        brlo inc_cntr
        ;the counter has reached the maximum.
        ;Turn port f on and reset the counter
        clr     r16
        sts     oflo_cntr, r16
        ldi     r16, 1
        sts     portf, r16
        jmp end_oflo
        ;increment the overflow counting variable
inc_cntr:
        inc     r16
        sts     oflo_cntr, r16
end_oflo:
        ;restore the sreg
        lds     r16, sreg_temp
        sts     sreg, r16
        ;exit
        pop     r16
        reti
```