SAE Auto Shifter
Design Report

Wade Brzozowski (wwb2103)
Joe Carey (jac2127)
Ron Alleyne (rja2001)

1. Abstract

Our design is to create a shifting system for the SAE racecar. The system will be controlled by a PIC microcontroller. A heads-up display will show the RPM, current gear, gear suggestion and include a temperature and oil pressure warning lights. The goal is to allow the driver to be able to shift with the push of a button. Movement of the physical shifter will be achieved with a specialized actuator designed for this particular application.

2. Design Overview

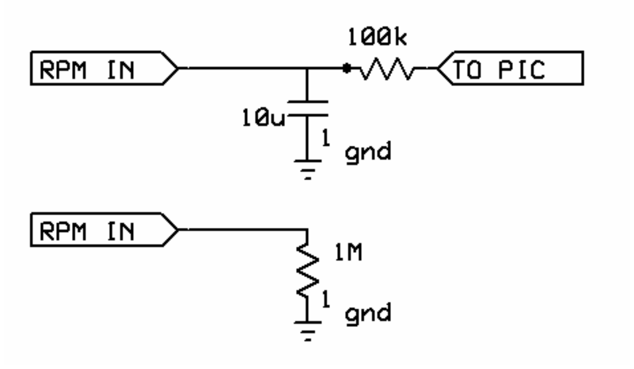
Based on data taken from engine sensors, the main idea is to have a heads-up display show the driver the current transmission gear and to make an appropriate recommendation on when to shift the gear up or down. Shifting the gear in our system would involve simply pushing a button. The main inputs to our system would be gear position sensor, RPM signal taken off of the crankshaft position sensor, temperature, oil pressure and buttons for shifting. All of these signals require some sort of circuitry to integrate them properly with the microcontroller. A PIC microcontroller is used to process the data and to provide control signals for the actuator as well as the display. Software development for the microcontroller was done in assembly. Circuitry driving the actuator utilizes power MOSFETs and off-the-shelf chips specialized to drive power MOSFETs from the microcontroller.

3. Input Circuitry

Signals coming off the engine are not suitable for input directly to the PIC. Input circuitry is needed to properly condition signals from the engine to ensure data is easily read by the PIC. Care must also be taken to ensure the engine circuitry is not disturbed and signals from the engine sensors do not harm the PIC.

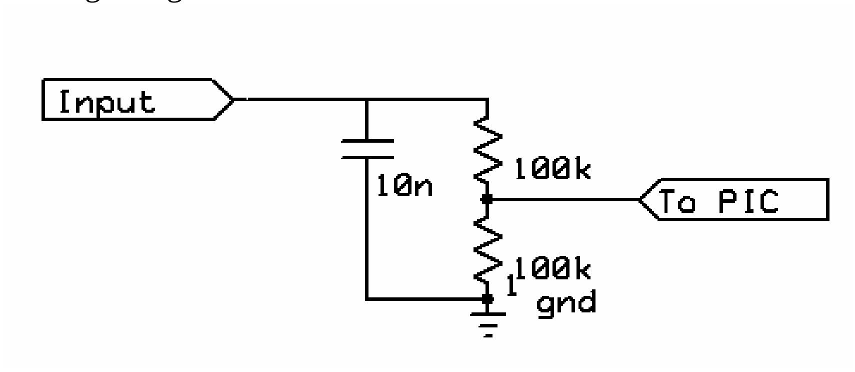
3.1 RPM Buffer

The RPM signal was one of the most challenging to measure successfully; this was because the RPM is a frequency dependant signal. The basic idea is to time the delay between rising edges of the signal. From this, the period is know and the frequency is easily determined—thus the RPMs. Most of the trouble came from the signal being differential. First attempts included an instrumentation amplifier followed by a Schmidt trigger. Here the idea was to convert the differential input to single ended while rejecting any common mode. The Schmidt trigger would further filter out any noise that might create multiple zero crossings thus giving an inaccurate reading. Preliminary tests showed the design to work; when used with the actual RPM signal, however, erroneous readings were shown on the RPM display. This was due impart to faulty opamps. Eventually a simpler approach was adopted. This design included grounding one of the RPM signals through a one-mega ohm resistor. The other RPM signal was passed through a one-kilo ohm resistor to the input on the PIC. Additionally a 10nF capacitor is used to filter out high frequency noise in the signal. This simplified approach cut down drastically the number of components and made debugging much easier.



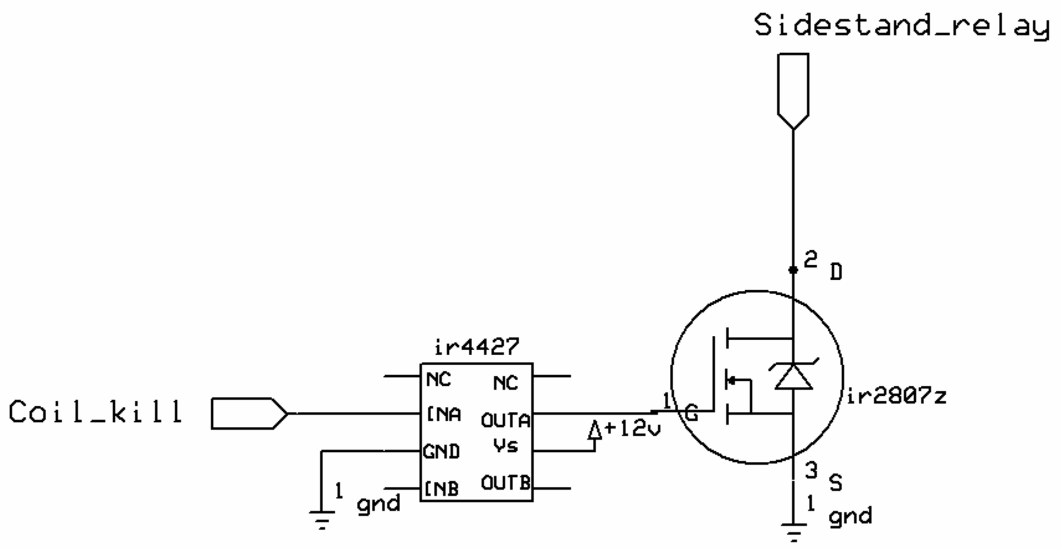
3.2 Buffers

Signals coming from the temperature and oil pressure sensors are simple analog voltages. A simple unity gain buffer is used to present the engine wiring harness with a high impedance. Analog to digital conversion is then done on the PIC.



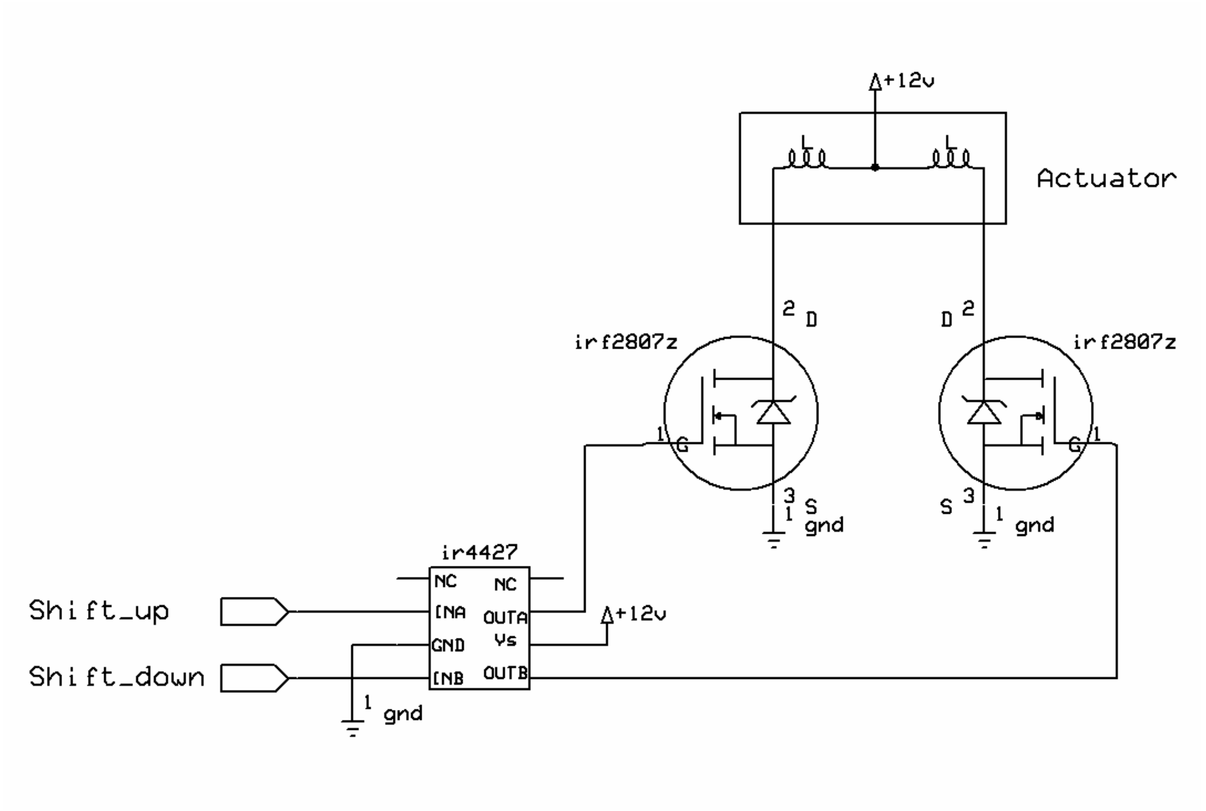
4. Engine Kill Circuit

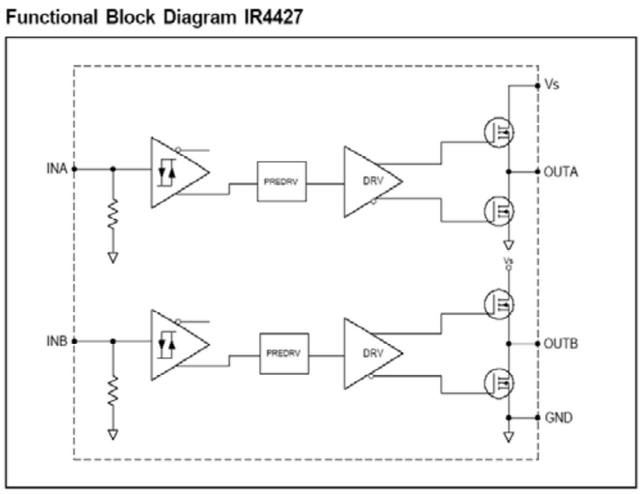
To allow the manual engine to be shifted without engaging the clutch, the current to the ignition coils must be killed for a small period of time. This can be achieved by using part of the project's motorcycle engine's existing electrical harness. By shorting the unused "side-stand" relay to ground, the engine can be briefly shut off. Our implementation will connect the lead of the side stand relay to the drain of a power MOSFET, which has a common source. When the PIC output signal that is fed to the gate of the MOSFET goes high, the side-stand relay will be shorted to ground. This solution was easy to implement, as the wiring harness to car did not have to be modified. The original idea was to kill the ignition right at the individual coils, but this would require cutting into the existing wiring harness at four separate locations. By utilizing methods already present on the car, a safer implementation was achieved without worry of damaging the ignition system.



5. Actuator Driver

A commercial actuator designed for auto shifting applications is used to engage the shift lever. The actuator is a solenoid that requires a large amount of current. International Rectifier's IR2807Z power MOSFET was chosen to switch the current on and off due to its large current capacity. This device is designed for automotive applications and is rated for 75A, well in excess of actuators rated 45A. Compared to the other option of using a relay the MOSFET proved to be the superior choice due to its physically smaller package size as well as nearly one-fourth the cost. The IR2807Z also includes a fly back diode to suppress voltage spikes induced when the actuator is turned off, making the overall design simpler and cheaper by lowering the overall part count. A SPICE simulation was done with values for resistance and inductance of the actuator found using an LCR meter. The results of the simulation were used to determine to maximum amount of current through the actuator as well as the total energy that would be dissipated by the MOSFET to ensure that the device was operating within its specification. To interface the PIC to the MOSFETs the IR4427 low side driver is used. This chip takes a logic level input straight from the PIC and drives the power MOSFETs. The IR4427 supplies enough current to quickly charge the gate capacitance on the MOSFETs allowing for a quick turn on time.





6. Display

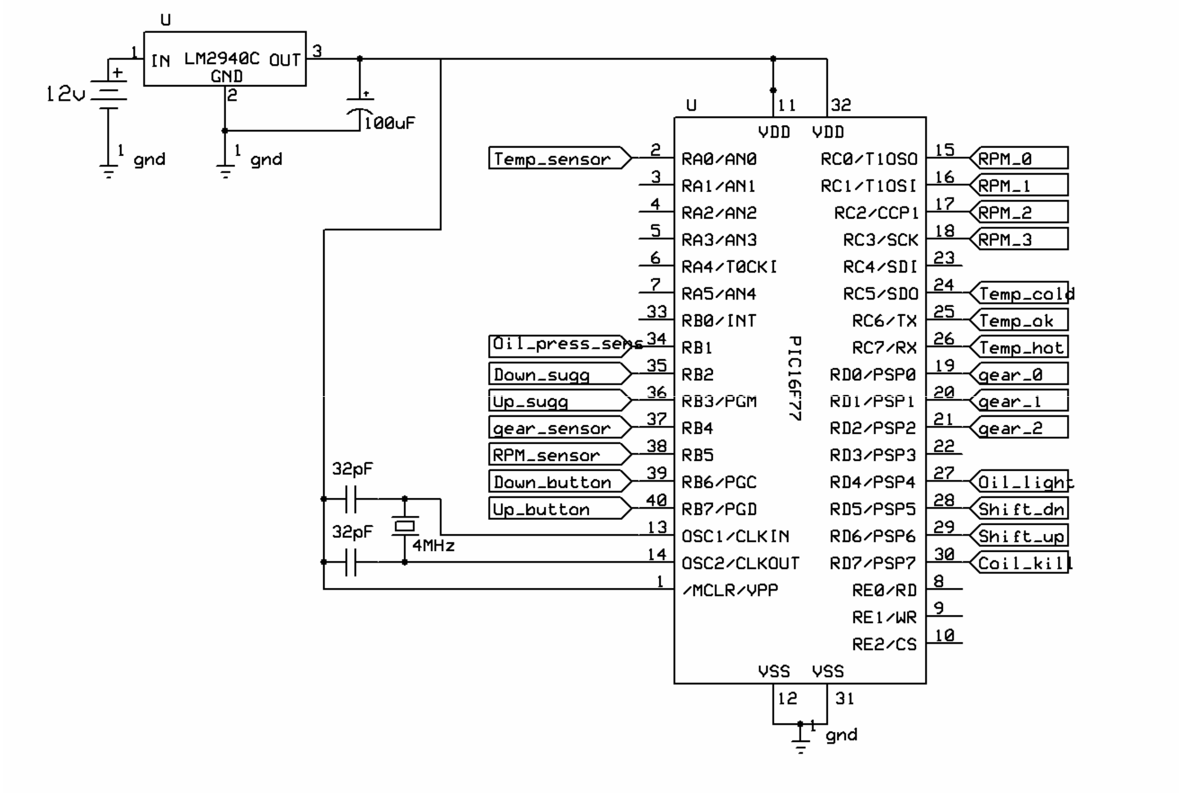
The main display is the dashboard. A Complex Programmable Logic Device (CPLD) was located directly behind the dash to decode the various outputs of our PIC chip. This was done to cut down on the number of wires that had to be run from the main box containing the PIC to the dashboard. Eleven signals were used as inputs to the CPLD (outputs of the PIC). Four will be used for the RPM display LEDs, three will be used to display the current gear position on a seven-segment LED, two will be used to display engine temperature, and two will be used for gear change suggestions. These signals, once decoded by the CPLD will drive 28 LEDs located on the dash. VHDL was utilized to program the CPLD.

7. Power Supply

The supply available on the car is 12V. A commercial voltage regulator package will be used to supply 5V to the circuits of the board. A large 100μF was used as a coupling capacitor. This was especially important due to the large current draw of the actuator. What we were experiencing was that when the actuator fired the supply voltage on the battery would drop and cause the microcontroller to reset. This was undesirable as the system was then out of sync.

8. PIC Microcontroller

A PIC 16F877A was used as a microcontroller. The 16F877A provides a total of 33 I/O pins with eight A/D channels. The large number of I/O was needed due to the large number of outputs. Using a PIC microcontroller was an easy decision as it allowed us a lot of flexibility in which pins are inputs and which are outputs. Also, the large number of built in features of the PIC allowed for an easy implementation of kinds of measurements of the real time data that we wanted to take. Data flash ROM is also available on the chip. This will enable us to write parameters into the flash ROM for gear suggestions allowing for tweaking. Below is a diagram of the 16F877A pins and the appropriate connections.



9. Software

At the heart of the racer’s gear control unit is a Microchip Technologies PIC16877A flash programmable microprocessor. The PIC is clocked at 4MHZ with a an instruction execution rate of 1MHZ. We used two of the PIC’s on-chip peripherals: the analog-to-digital converter (ADC) and the timer.
The operation of the PIC’s ADC module results in a corresponding 10-bit digital number. This module was used to convert the voltage readings delivered by the temperature sensor of the electrical harness. The use of the ADC required the setting of the appropriate control registers and the processing of the registers that stored the ADC result.

The PIC’s timer module was use to find the instantaneous period of the RPM signal. With each instruction execution, the 8-bit timer was incremented with the result being stored in a special-purpose register.

The actual assembly code written to control the PIC begins with the initialization of the chip’s pins as output or input ports and the setup of registers needed to control the aforementioned peripherals. Additionally, registers used to implement the gear control algorithm were also instantiated. Since the gear control unit would “boot” when the car was powered, a delay[1] was introduced to allow the electrical harness’ sensor signals to settle before reading and processing them.

The main control loop of the gear control unit follows the initial display. The first part of the loop measures the period of the RPM signal. With it being a base-band frequency modulated signal, the useful information can be found in the period of this signal. This information is later used to determine how the dashboard LED tachometer should be lit up to represent the current RPM reading. The code next uses the gear position sensor to determine if the system is calibrated. With this sensor being nothing more than a binary signal, it is impossible to know which gear the unit is in if it starts in anything other than the neutral gear. Fortunately, the stock control unit ensures that the engine’s electrical harness will usually not be powered unless the car is in neutral. The results could be problematic if the gear control unit tried to actuate the gear shift lever down from the first gear or up from the sixth gear. This possibility is eliminated by essentially disabling most functioning of the main control loop when the system has not been calibrated or more specifically, when the system has not sensed a neutral reading on the gear position sensor at least once since receiving power. The only code component that continues to function while the unit is an un-calibrated state is the dashboard display.

After calibration has been confirmed, the steering wheel gear shift buttons are polled to see if the driver is requesting a shift. With a 1MHZ processor, the duty cycle of the control loop is short enough to keep the unit from missing requests for gear shifts. The buttons are de-bounced in the sense that the code will not register a button press until it

[1] Delay code for any period of time written in assembly for the PIC was obtained easily with the use of http://www.sxlist.com/techref/piclist/codegen/delay.htm

senses the button have been pressed and then released in separate iterations of the control loop. Once a button is press, its validity is confirmed by doing a bounds-check on the current gear. The system will only allow the current gear to be greater than or equal to 1 and less than or equal to 6. A valid press is processed by incrementing the current gear, setting the appropriate port bit to kill the engine for a specified period of time, writing to the appropriate port bit to actuate the signal for another period of time and then clearing the bit signaling that the engine no longer be killed. By introducing time-precise delays during the design phase, the system could be later tweaked to meet the needs of the engine.

Next, the system updates the RPM display by comparing the current RPM signal period to know RPM signal period benchmarks. Also, the temperature sensor readout, the oil pressure sensor readout and the gear suggestion for the current RPM sensor reading and current gear reading are also refreshed. All of the display outputs are encoded and written to ports that the CPLD decodes and uses to power the appropriate dashboard display LEDs. The RPM sensor readout is translated to a 4-bit number that is written on four ports to signal the CPLD about how many RPM tachometer LEDs to power. The temperature readout made available by the ADC is translated to a hot, cold and warm LED. It should be noted the gear readout is only accurate when the gear control unit is used to automatically shift the engine. Otherwise, this readout and the gear suggestion are inaccurate. Following the code implementing the main loop are the code blocks used to implement small operations that support the functioning of the main loop.

10. Fabrication

Our board was fabricated using the simple but time-consuming method of wire wrapping. This process consists of taking 30-gauge wire and wrapping it around header pins to make the various connections. To make sure that these connections were secure, we also soldered every wire wrap directly to the header pins. We chose to do this because of time constraints and the cost was much lower than the alternatives. The main drawback of wire wrapping is the wires remain exposed on the backside of the board. In a project such as this, where drilling and welding are occurring constantly, having these exposed wires is not desired because they have a higher likelihood of being damaged or severed. Ideally, we would have designed a printed circuit board (PCB) and had it fabricated professionally. However, since we only needed one board, the cost was not economical. If this project were to be mass-produced, a PCB would be used as the cost per board would go down as the number of boards fabricated increased.

11. Results

Initially, while working in the lab, every test we ran went well. It wasn’t until we had to interface with the car’s wire harness that we began having problems. First, op-amps began failing, so we decided to replace them with resistors in a voltage divider configuration. They provide the same functionality as a unity gain buffer; however, they only reduced the signals to 6V. After reading the PIC datasheet we decided that 6V was

a low enough voltage and that it would not damage our processor. Another circuit of ours that we had planned on using also became useless, due to op-amp failures, was our Schmitt Trigger. The trigger was supposed to reduce the “ringing” of the RPM signal and thus making it into a more suitable square wave. We noticed this problem when we first went to test our LED tachometer. All 14 LEDs would light up periodically even though the RPM level was well below redline. To fix this, we implemented a low pass filter in our assembly code. In the code we stated if the time between two rising edges of the RPM signal were effectively less than what would be equivalent to 15,000 RPMs, we would ignore that signal because it was actually the ringing of the impure signal. Once these fixes were made our dashboard performed amazingly well.
We encountered surprisingly few problems when it came to our actuator control. Most problems occurred when we hooked the actuator up to our circuit incorrectly and blew a power MOSFET. However, as of this paper, we have not been able to test to see if the engine kill time is long enough to result in a proper shift. We are still looking for a safe way to test this without having to actually drive the car.

12. Conclusions

This project plainly illustrated to us the difficulty encountering when interfacing electronics with a real-world mechanical peripheral. Considering the difficultly during integration, we are very pleased and somewhat surprised at the ability to produce a functioning product. We learned how invaluable it is to have definite arrangements for circuit topology and fabrication in hand before the integration phase. Unfortunately, we found ourselves still designing and building after the time came for integration. Another useful engineering lesson learned is the ease with which we could solve problems in software. While software solutions limit efficiency, they were easier to deal with and provided the immediate assurance hardware solutions do not always provide. We also learned how much fabrication decisions factor into the durability and debugging of such an application. Finally, we learned how important it is to set modest goals when it comes to the scope of such a project. While we achieved functionality in all the significant areas of our design goals, there were fine details that did not receive the attention because of the pressure to solve bigger problems. All in all, this project was a great learning experience and provided us with a measure of real-world experience that would not have been as accessible in some of the more conventional embedded systems projects.

13. Appendix

13.1 VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity decoder is
port(inputs : in std_logic_vector(6 downto 0);
     RPM    : out std_logic_vector(13 downto 0);
     gear : out std_logic_vector(2 downto 0));
end decoder;
```

```
architecture behavior of decoder is

begin
  process(inputs)
  begin
       case inputs(6 downto 3) is      --RPM
          when "0000" =>
               RPM(13 downto 0) <= "00000000000000";   --0 RPM
          when "0001" =>
               RPM(13 downto 0) <= "00000000000001";   --1000
          when "0010" =>
               RPM(13 downto 0) <= "00000000000011";   --2000
          when "0011" =>
               RPM(13 downto 0) <= "00000000000111";   --3000
          when "0100" =>
               RPM(13 downto 0) <= "00000000001111";   --4000
          when "0101" =>
               RPM(13 downto 0) <= "00000000011111";   --5000
          when "0110" =>
               RPM(13 downto 0) <= "00000000111111";   --6000
          when "0111" =>
               RPM(13 downto 0) <= "00000001111111";   --7000
          when "1000" =>
               RPM(13 downto 0) <= "00000011111111";   --8000
          when "1001" =>
               RPM(13 downto 0) <= "00000111111111";   --9000
          when "1010" =>
               RPM(13 downto 0) <= "00001111111111";   --10,000
          when "1011" =>
               RPM(13 downto 0) <= "00011111111111";   --11,000
          when "1100" =>
               RPM(13 downto 0) <= "00111111111111";   --12,000
          when "1101" =>
               RPM(13 downto 0) <= "01111111111111";   --13,000
          when "1110" =>
               RPM(13 downto 0) <= "11111111111111";   --14,000
          when others =>
               RPM(13 downto 0) <= "11111111111111";   --error
       end case;

       case inputs(2 downto 0) is      --Gear Display
          when "000" =>                --gfedcba
               gear(2 downto 0) <=  "1000000"; --0, neutral
          when "001" =>                --gfedcba
               gear(2 downto 0) <=  "1111100"; --1
          when "010" =>                --gfedcba
               gear(2 downto 0) <=  "0010010"; --2
          when "011" =>                --gfedcba
               gear(2 downto 0) <=  "0011000"; --3
          when "100" =>                --gfedcba
               gear(2 downto 0) <=  "0101100"; --4
          when "101" =>                --gfedcba
               gear(2 downto 0) <=  "0001001"; --5
          when "110" =>                --gfedcba
               gear(2 downto 0) <=  "0000001"; --6
          when others =>               --gfedcba
               gear(2 downto 0) <=  "0000011"; --E, error
       end case;
  end process;
end behavior;
```

13.2 PIC assembly code

```
;
        Title "SAE autoshifter."
;
        list P = 16F877A
;
        include "P16F877A.inc"  ; use definition file for 16F877A
;
; --------------------
; USER RAM DEFINITIONS
; --------------------
;
        CBLOCK 0x20   ; RAM starts at address 20h
gear
flag
rpm_flag
neutral_last
neutral
calibrated
NaHi
clkl
clkh
d1
d2
d3
Xhi
Xlo
        ENDC
;
      org 0x0000      ; start address = 0000h
      goto Initialize

; INITIALISE PORTS
; binary used to see inividual pin level

Initialize
      movlw b'00000000'
      movwf INTCON
      movlw 0x00
      movwf NaHi
      movwf 0x07
      movwf gear
      movlw b'00000000'       ; all port pins = low
      movwf PORTA
      movlw b'00000000'
      movwf PORTB
      movlw b'00000000'
      movwf PORTC
      movlw b'00000111'
      movwf PORTD
      movlw b'00000000'
      movwf PORTE

      ;start a/d code
      bsf     STATUS,RP0      ;bank 1
      bcf     STATUS,RP1
      movlw   b'00001110'
      clrf    ADCON1          ;left justified, input an0
      bcf     STATUS,RP0      ;bank 0
      movlw   B'01000001'     ;Fosc/8 [7-6], A/D ch0 [5-3], a/d on [0]
      movwf   ADCON0

      bsf STATUS,RP0  ; set RAM Page 1 for TRIS registers
      bcf STATUS,RP1
      ; INITIALISE PORTS
      ; binary used to see individual pin IO status

      movlw b'11111111' ;1=input, 0=output
      movwf TRISA
      movlw b'11110010'
      movwf TRISB
      movlw b'00000000'
      movwf TRISC
      movlw b'00000000'
      movwf TRISD
      movlw b'00000000'
      movwf TRISE

      bcf STATUS,RP0  ; back to RAM page 0

      movlw 0x00
      movwf TMR1L
      movwf TMR1H
      movwf TMR1L
      movwf flag
      movwf rpm_flag
      movwf neutral
      movwf calibrated
      movwf neutral_last
      movlw b'00000001'
      movwf T1CON

      ;delay to allow everything to settle
      call startup_delay

;main loop in which displays are updated
Main

rpm1  btfsc PORTB,5
      goto rpm1
rpm2  btfss PORTB,5
      goto rpm2
      movlw 0x00
      movwf TMR1L
      movwf TMR1H
      movwf TMR1L
rpm3  btfsc PORTB,5
      goto rpm3
```

```
rpm4  btfss PORTB,5
      goto rpm4
      movf TMR1L,W
      movwf clkl
      movf TMR1H,W
      movwf clkh

      btfss PORTB,4
      bsf calibrated,0 ; neutral must be low
      btfss calibrated,0
      goto button_done ; must not be calibrated

      ;check if we are in neutral
      btfss PORTB,4
      bsf neutral,0 ; must not be in neutral anymore

      btfsc PORTB,4
      bcf neutral,0 ; must not be in neutral anymore

      btfss neutral,0
      goto here

      clrf gear   ;set display to '0'
      clrf PORTD
      goto button_done

here
      btfss neutral_last,0
      goto here2
      movlw 0x01
      movwf gear
      movwf PORTD

here2
      btfsc PORTB,5
      call rpm_check
      btfss PORTB,5
      bcf  rpm_flag,0

      ;check if the buttons are pressed
      btfss PORTB,7
      goto button_up
      btfss PORTB,6
      goto button_down

button_done
      ;clear the flag if needed
      btfss PORTB,7
      goto  $+3
      btfsc PORTB,6
      bcf flag,0

      movf neutral,W
      movwf neutral_last

      ;update the rpm
      movlw 0x17 ; put 6000 in x
```

```
movwf Xhi
movlw 0x70
movwf Xlo
call compare_unsigned_16 ;test if count is less than 6000
btfsc STATUS, C
goto rpm_0 ; if count is greater than 6000 we have 0 RPM
movlw 0x0b ; check 3000
movwf Xhi
movlw 0xb8
movwf Xlo
call compare_unsigned_16
btfsc STATUS, C
goto rpm_1000
movlw 0x07 ; check 2000
movwf Xhi
movlw 0xd0
movwf Xlo
call compare_unsigned_16
btfsc STATUS, C
goto rpm_2000
movlw 0x05 ; check 1500
movwf Xhi
movlw 0xdc
movwf Xlo
call compare_unsigned_16
btfsc STATUS, C
goto rpm_3000
movlw 0x04 ; check 1200
movwf Xhi
movlw 0xb0
movwf Xlo
call compare_unsigned_16
btfsc STATUS, C
goto rpm_4000
movlw 0x03 ; check 1000
movwf Xhi
movlw 0xe8
movwf Xlo
call compare_unsigned_16
btfsc STATUS, C
goto rpm_5000
movlw 0x03 ; check 857
movwf Xhi
movlw 0x59
movwf Xlo
call compare_unsigned_16
btfsc STATUS, C
goto rpm_6000
movlw 0x02 ; check 750
movwf Xhi
movlw 0xee
movwf Xlo
call compare_unsigned_16
btfsc STATUS, C
goto rpm_7000
movlw 0x02 ; check 667
movwf Xhi
```

```
      movlw 0x9b
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      goto rpm_8000
      movlw 0x02 ; check 600
      movwf Xhi
      movlw 0x58
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      goto rpm_9000
      movlw 0x02 ; check 545
      movwf Xhi
      movlw 0x21
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      goto rpm_10000
      movlw 0x01 ; check 500
      movwf Xhi
      movlw 0xf4
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      goto rpm_11000
      movlw 0x01 ; check 462
      movwf Xhi
      movlw 0xce
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      goto rpm_12000
      movlw 0x01 ; check 429
      movwf Xhi
      movlw 0xad
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      goto rpm_13000

      ;low pass filter
      movlw 0x01 ; check 400
      movwf Xhi
      movlw 0x90
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      goto rpm_max
      goto rpm_impossible

rpm_0
      movf  PORTC,W
      andlw 0xf0
      addlw   b'00000000'
      movwf       PORTC
      goto led_finish
```

```
rpm_1000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00000001'
      movwf PORTC
      goto led_finish
rpm_2000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00000010'
      movwf PORTC
      goto led_finish
rpm_3000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00000011'
      movwf PORTC
      goto led_finish
rpm_4000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00000100'
      movwf PORTC
      goto led_finish
rpm_5000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00000101'
      movwf PORTC
      goto led_finish
rpm_6000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00000110'
      movwf PORTC
      goto led_finish
rpm_7000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00000111'
      movwf PORTC
      goto led_finish
rpm_8000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00001000'
      movwf PORTC
      goto led_finish
rpm_9000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00001001'
      movwf PORTC
      goto led_finish
rpm_10000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00001010'
      movwf PORTC
      goto led_finish
rpm_11000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00001011'
      movwf PORTC
      goto led_finish
rpm_12000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00001100'
      movwf PORTC
      goto led_finish
rpm_13000
      movf  PORTC,W
      andlw 0xf0
      addlw b'00001101'
      movwf PORTC
      goto led_finish
rpm_max
      movf  PORTC,W
      andlw 0xf0
      addlw b'00001110'
      movwf PORTC
      goto led_finish

rpm_impossible

led_finish
      movf gear,W
      sublw 0x01
      btfsc STATUS,Z
      call one_sug
      movf gear,W
      sublw 0x02
      btfsc STATUS,Z
      call two_sug
      movf gear,W
      sublw 0x03
      btfsc STATUS,Z
      call three_sug
      movf gear,W
      sublw 0x04
      btfsc STATUS,Z
      call four_sug
      movf gear,W
      sublw 0x05
      btfsc STATUS,Z
      call five_sug
      movf gear,W
      sublw 0x06
      btfsc STATUS,Z
      call six_sug

;do an a/d
```

```
ad_portc
      bsf     ADCON0,GO ;Start A/D conversion
Wait
      btfsc   ADCON0,GO ;Wait for conversion to complete
      goto    Wait

      movf    ADRESH,W ;Write A/D result to W register
      addlw 0x6b ;overflow will occur if voltage on PORT RA0 > 2.9V
      btfsc   STATUS,C
      goto  ad_hot

      movf    ADRESH,W ;Write A/D result to W register
      addlw 0x94 ;overflow will occur if voltage on PORT RA0 > 2.1V
      btfsc   STATUS,C
      goto  ad_ok

      goto  ad_cold ;if we get here PORT RA0 < 2.1V

ad_hot
      bsf   PORTC,7
      bcf PORTC,6
      bcf   PORTC,5
      goto ad_done
ad_ok
      bcf   PORTC,7
      bsf PORTC,6
      bcf   PORTC,5
      goto ad_done
ad_cold
      bcf   PORTC,7
      bcf PORTC,6
      bsf   PORTC,5
ad_done

      ;check oil pressure, change display appropriately
      btfsc PORTB,1
      bsf   PORTD,4 ;pressure too low
      btfss PORTB,1
      bcf   PORTD,4 ;pressure OK

      goto Main

button_up
      ;if button still pressed, exit
      btfsc flag,0
      goto  done_up
      bsf   flag,0

      movf  gear,W
      sublw 0x06
      btfsc STATUS,Z
      goto  done_up

      movlw 0x01
      addwf gear,W
      movwf gear
      movwf PORTD
      bsf PORTD,7 ; engine kill

      call e_kill_delay ; 25ms
      bsf PORTD,6       ; actuate up

      call actuate_delay ; 100ms
      bcf PORTD,6 ; stop actuation

      call e_kill_delay ; 25 ms
      ;total shift time = 150ms
      bcf PORTD,7; engine unkill
done_up
      goto button_done

button_down
      ;if button still pressed, exit
      btfsc flag,0
      goto  done_down
      bsf   flag,0

      decfsz gear,W
      goto  $+2
      goto done_down
      movwf gear
      movwf PORTD

      bsf PORTD,7 ; engine kill

      call e_kill_delay ; 25ms
      bsf PORTD,5       ; actuate down

      call actuate_delay ; 100ms
      bcf PORTD,5 ; stop actuation

      call e_kill_delay ; 25ms
      ;total shift time = 150ms
      bcf PORTD,7; engine unkill

done_down
      goto button_done

compare_unsigned_16:
      movf Xhi,w
      subwf clkh,w ; subtract Y-X
Are_they_equal:
      ; Are they equal ?
      skpz
      goto results16
      ; yes, they are equal -- compare lo
      movf Xlo,w
      subwf clkl,w      ; subtract Y-X
results16:
      ; if X=Y then now Z=1.
      ; if X>Y then now C=0.
      ; if X<=Y then now C=1.
      return
```

```
rpm_check
      btfsc rpm_flag,0
      goto rpm_done
      movf TMR1L,W
      movwf clkl
      movf TMR1H,W
      movwf clkh
      movlw 0x00
      movwf TMR1L
      movwf TMR1H
      movwf TMR1L
      bsf rpm_flag,0
rpm_done
      return

one_sug
      ;if gear=1 and rpm>9000, shift up
      movlw 0x02
      movwf Xhi
      movlw 0x58
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      bcf PORTB,3
      btfss STATUS,C
      bsf PORTB,3
      return

two_sug
      ;if gear=2 and rpm>9500, shift up
      movlw 0x02
      movwf Xhi
      movlw 0x58
      movwf Xlo
      call compare_unsigned_16
      btfsc STATUS, C
      bcf PORTB,3
      btfss STATUS,C
      bsf PORTB,3

      ;if gear=2 and rpm<4500, shift down
      movlw 0x05
      movwf Xhi
      movlw 0xdc
      movwf Xlo
      call compare_unsigned_16
      btfss STATUS, C
      bcf PORTB,2
      btfsc STATUS,C
      bsf PORTB,2
      return

three_sug
      ;if gear=3 and rpm>9000, shift up
      movlw 0x02
      movwf Xhi
      movlw 0x58
```

```
        movwf Xlo
        call compare_unsigned_16
        btfsc STATUS, C
        bcf PORTB,3
        btfss STATUS,C
        bsf PORTB,3

        ;if gear=3 and rpm<4000, shift down
        movlw 0x05
        movwf Xhi
        movlw 0xdc
        movwf Xlo
        call compare_unsigned_16
        btfss STATUS, C
        bcf PORTB,2
        btfsc STATUS,C
        bsf PORTB,2
        return

four_sug
        ;if gear=4 and rpm>9500, shift up
        movlw 0x02
        movwf Xhi
        movlw 0x58
        movwf Xlo
        call compare_unsigned_16
        btfsc STATUS, C
        bcf PORTB,3
        btfss STATUS,C
        bsf PORTB,3

        ;if gear=4 and rpm<3750, shift down
        movlw 0x05
        movwf Xhi
        movlw 0xdc
        movwf Xlo
        call compare_unsigned_16
        btfss STATUS, C
        bcf PORTB,2
        btfsc STATUS,C
        bsf PORTB,2
        return

five_sug
        ;if gear=5 and rpm>8000, shift up
        movlw 0x02
        movwf Xhi
        movlw 0x58
        movwf Xlo
        call compare_unsigned_16
        btfsc STATUS, C
        bcf PORTB,3
        btfss STATUS,C
        bsf PORTB,3

        ;if gear=5 and rpm<4000, shift down
        movlw 0x05
```

```
      movwf Xhi
      movlw 0xdc
      movwf Xlo
      call compare_unsigned_16
      btfss STATUS, C
      bcf PORTB,2
      btfsc STATUS,C
      bsf PORTB,2
      return

six_sug
      ;if gear=6 and rpm<3000, shift down
      movlw 0x05
      movwf Xhi
      movlw 0xdc
      movwf Xlo
      call compare_unsigned_16
      btfss STATUS, C
      bcf PORTB,2
      btfsc STATUS,C
      bsf PORTB,2
      return

startup_delay
      ;2 second delay to let signals settle
      movlw 0x11
      movwf d1
      movlw 0x5D
      movwf d2
      movlw 0x05
      movwf d3
Delay_0
      decfsz d1, f
      goto $+2
      decfsz d2, f
      goto $+2
      decfsz d3, f
      goto Delay_0

      ;4 cycles
      goto  $+1
      goto  $+1
      return

e_kill_delay
      ;25ms delay
      movlw 0x4F
      movwf d1
      movlw 0xC4
      movwf d2
Delay_1
      decfsz d1, f
      goto $+2
      decfsz d2, f
      goto Delay_1

            ;2 cycles
      goto  $+1
      return

actuate_delay
      ;100ms delay
      movlw 0x03
      movwf d1
      movlw 0x18
      movwf d2
      movlw 0x02
      movwf d3
Delay_2
      decfsz d1, f
      goto $+2
      decfsz d2, f
      goto $+2
      decfsz d3, f
      goto Delay_2

      ;6 cycles
      goto  $+1
      goto  $+1
      goto  $+1
      return

      end
```