SAE Auto Shifter
Design Report

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

1. Design Overview

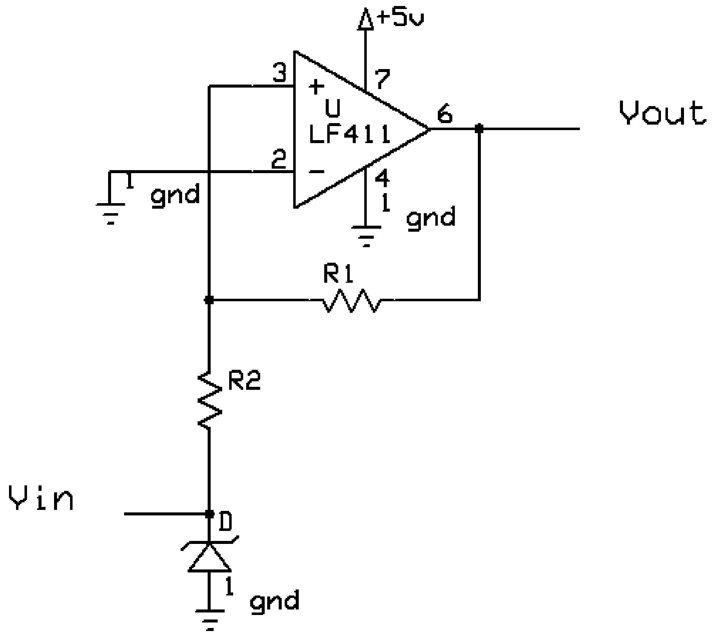
Our design is to create a shifting system for the SAE racecar. The system will be controlled by a PIC microcontroller. A heads up display will show the RPM, current gear, gear suggestion, and include a temperature and oil pressure warning lights. The goal is to allow the car driver to be able to shift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

2. 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 that data is easily read by the PIC. Care must also be taken to ensure that the engine circuitry is not disturbed and that signals from the engine sensors do not harm the PIC.

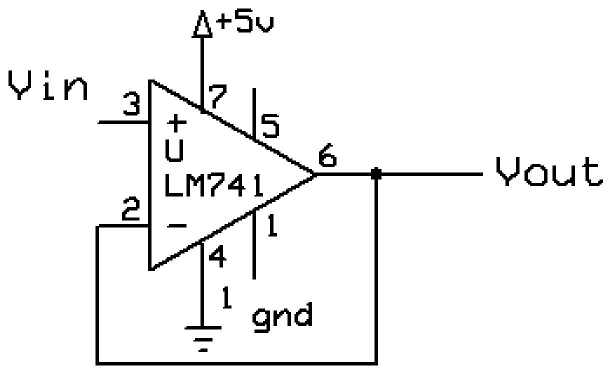
2.1 RPM Buffer

For the RPM sensor a Schmidt trigger will be used. The signal coming off of the RPM sensor is a distorted sine wave with a frequency of ten times the engine RPM. The Schmidt trigger will fix the amplitude of the signal at either +5V or 0V. This will allow the PIC to calculate the time between the rising edges of the RPM signal and thus find the frequency and RPM. Noise will also be rejected well by the Schmidt trigger by rejecting multiple zero crossings. A zener diode is included at the input to clamp the signal at an appropriate level.



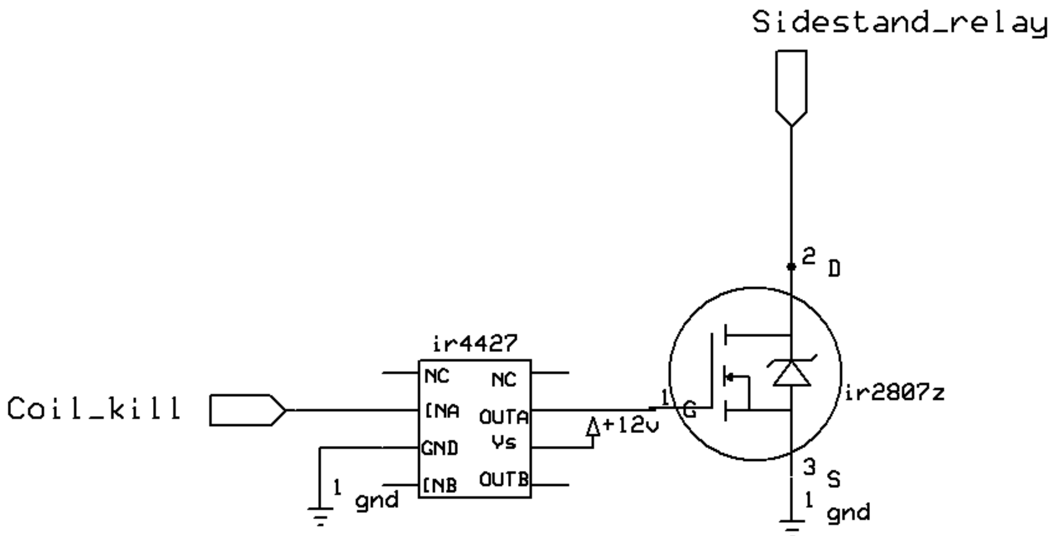
2.2 Temperature and Oil Pressure Buffers

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.



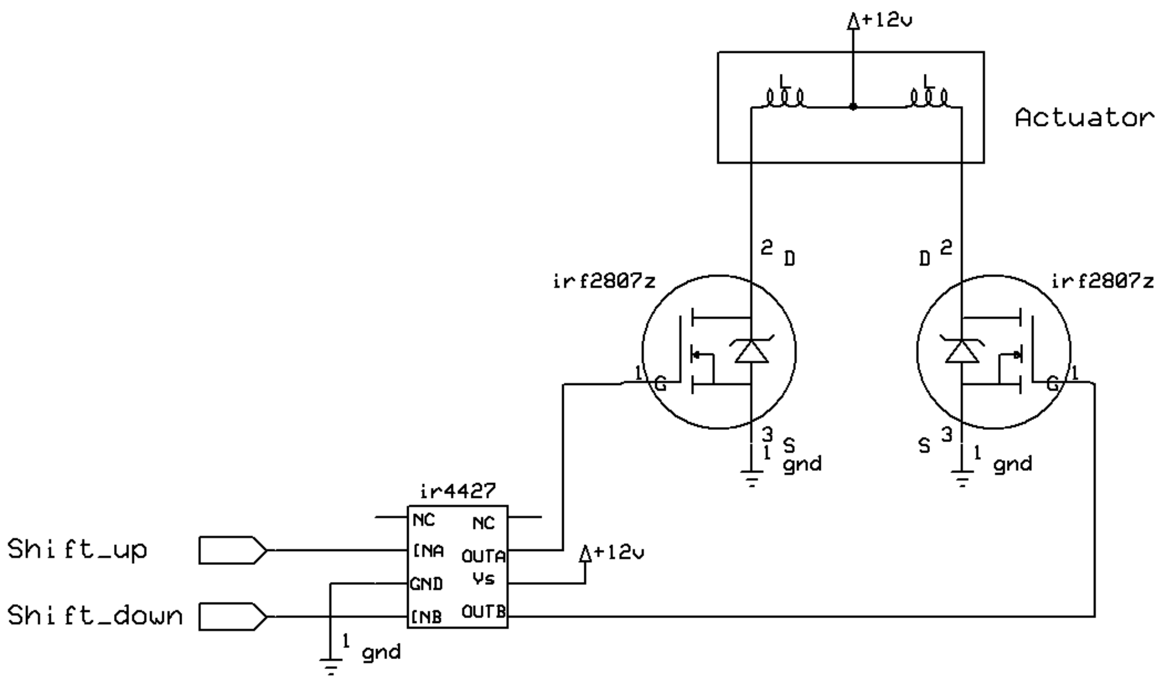
3. Engine Kill Circuit

To allow the manual engine to be shifted without engaging the clutch, the coil to the current 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 killed. 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.

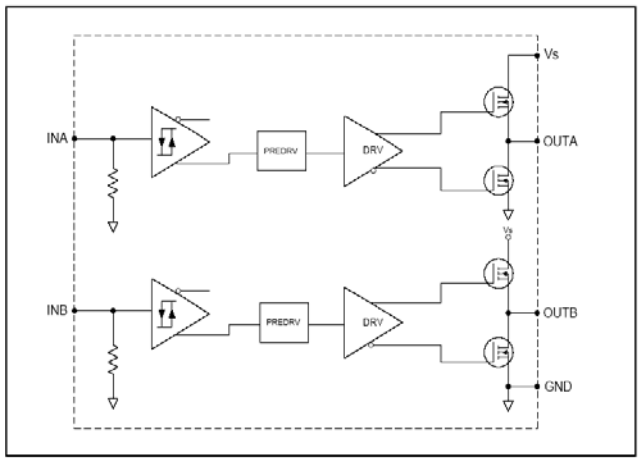


4. Actuator Driver

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 due to its large current capacity and low price compared to other option. The IR2807Z also includes a fly back diode to suppress voltage spikes induced when the actuator is turned off. To interface the PIC to the mosfets the IR4427 low side driver is used. This chip take 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.



Functional Block Diagram IR4427



5. Display

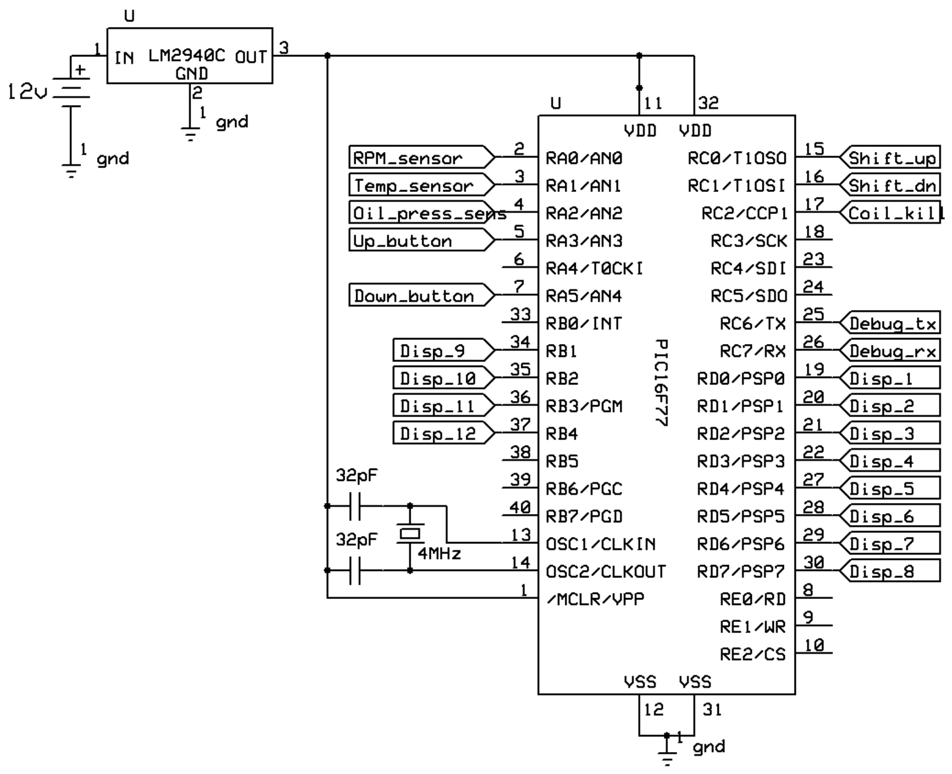
We will be using a CPLD to decode the various outputs of our PIC chip. We will use 11 signals as inputs to the CPLD (outputs of the PIC). Four will be used for the RPM display LEDs, three will be used to display the current gear position on a seven-segment LED, two will be used to display engine temperature, and two will be used for gear change suggestions. These signals, once decoded by the VHDL code provided in the Appendix, will be connected to various LEDs. The LEDs will be mounted on the dashboard of the SAE car so that the driver can closely monitor the data from the engine.

6. 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.

7. PIC Microcontroller

A PIC 16F877A will be used as a microcontroller. The 16F877A provides a total of 33 I/O pins with eight A/D channels. Data flash ROM is available on the chip. This will enable us to write parameters into the flash ROM for gear suggestions allowing for tweaking through the serial port without having to reprogram the entire microcontroller. Below is a diagram of the 16F877A pins and the appropriate connections.



8. Appendix

8.1 VHDL Code

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

entity decoders is
port(inputs : in std_logic_vector(10 downto 0);
     outputs : out std_logic_vector(25 downto 0));
end decoders;

architecture behavior of decoders is

begin
   process(inputs)
   begin
      case inputs(10 downto 7) is   --RPM
         when "0000" =>
            outputs(25 downto 12) <= "00000000000000";      --0 RPM
         when "0001" =>
            outputs(25 downto 12) <= "00000000000001";      --1000
         when "0010" =>
            outputs(25 downto 12) <= "00000000000011";      --2000
         when "0011" =>
            outputs(25 downto 12) <= "00000000000111";      --3000
         when "0100" =>
            outputs(25 downto 12) <= "00000000001111";      --4000
         when "0101" =>
            outputs(25 downto 12) <= "00000000011111";      --5000
         when "0110" =>
            outputs(25 downto 12) <= "00000000111111";      --6000
         when "0111" =>
            outputs(25 downto 12) <= "00000001111111";      --7000
         when "1000" =>
            outputs(25 downto 12) <= "00000011111111";      --8000
         when "1001" =>
            outputs(25 downto 12) <= "00000111111111";      --9000
         when "1010" =>
            outputs(25 downto 12) <= "00001111111111";      --10,000
         when "1011" =>
            outputs(25 downto 12) <= "00011111111111";      --11,000
         when "1100" =>
            outputs(25 downto 12) <= "00111111111111";      --12,000
         when "1101" =>
            outputs(25 downto 12) <= "01111111111111";      --13,000
         when "1110" =>
            outputs(25 downto 12) <= "11111111111111";      --14,000
         when others =>
            outputs(25 downto 12) <= "11111111111111";      --error
      end case;

      case inputs(6 downto 4) is    --Gear Display
         when "000" =>        --abcdefg
            outputs(11 downto 5) <=  "1111110"; --0, neutral
         when "001" =>        --abcdefg
            outputs(11 downto 5) <=  "1100000"; --1
         when "010" =>        --abcdefg
            outputs(11 downto 5) <=  "1011011"; --2
         when "011" =>        --abcdefg
            outputs(11 downto 5) <=  "1110011"; --3
         when "100" =>        --abcdefg
            outputs(11 downto 5) <=  "1100101"; --4
         when "101" =>        --abcdefg
            outputs(11 downto 5) <=  "0110111"; --5
         when "110" =>        --abcdefg
            outputs(11 downto 5) <=  "0111111"; --6
         when others =>       --abcdefg
            outputs(11 downto 5) <=  "0011111"; --E, error
      end case;

      case inputs(3 downto 2) is    --Gear Suggestion LEDs
         when "00" =>
            outputs(4 downto 3) <= "00";  --none on
         when "01" =>
            outputs(4 downto 3) <= "01";  --up shift
         when "10" =>
            outputs(4 downto 3) <= "10";  --down shift
         when others =>
            outputs(4 downto 3) <= "11";  --both on, error
      end case;

      case inputs(1 downto 0) is    --Temperature LEDs
         when "00" =>
            outputs(2 downto 0) <= "001"; --cold
         when "01" =>
            outputs(2 downto 0) <= "010"; --normal
         when "10" =>
            outputs(2 downto 0) <= "100"; --hot
         when others =>
            outputs(2 downto 0) <= "111"; --all on, error
      end case;
   end process;
end behavior;
```