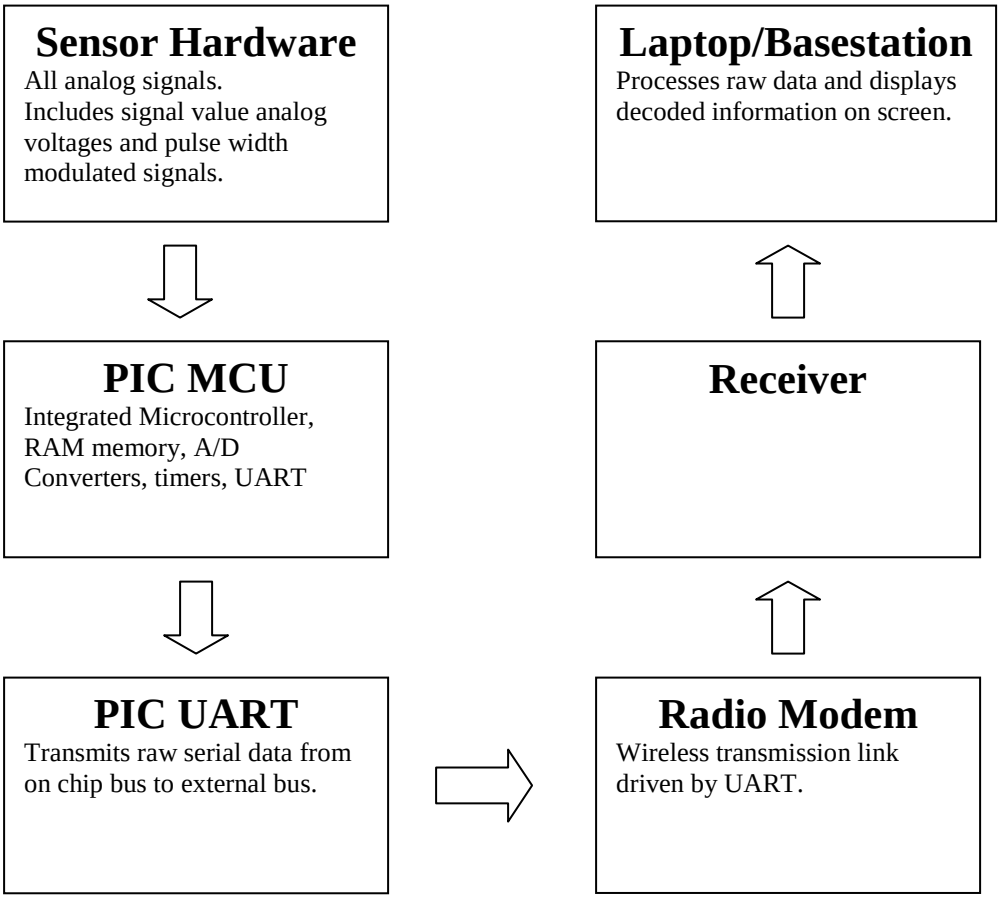
DAQ-T: Telemetry System Design Document
Laxmikant Joshi, Sangeeta Das, Stephen Tarzia, Waclaw Godycki

Implementation Overview

Analog signals are generated by sensor located on various parts of the engine and chassis or by outputs of the EFI-ECU. Please refer to Project Description for details on sensor names/implementations. These signals will need some analog interface circuitry to attach to the ADC or timer channels of the PIC Microcontroller (MCU). Single value signals (such as thermistor signals/strain gauges) will go to ADC channels, and time based signals will go to timer channels. The data will be converted into 8-bit PCM words by the MCU. The basis of the processing is a simple polling loop. On every polling loop cycle, the MCU will initiate A/D conversion on each ADC channel, store the results in memory, store pulse width timer data to memory, then construct a frame of data by retrieving the most recent data from memory and sequentially transmitting it to the radio modem via the UART. The MCU has a tri-state bus, with all data path and peripheral elements on the bus. All peripherals are controlled by memory mapped control registers. The radio modem link sends the data to a receiver at the base station, most likely a laptop computer. The raw data will then be processed by the laptop on site and displayed in user readable format either graphically or textually.

Simplified Block Dataflow Diagram



Sensors and analog interface circuitry

Since all sensor signals are piggybacked from the stock ECU built into the engine, all voltage signals will be buffered by high input impedance op-amp circuitry to ensure signal strength/integrity and low attenuations. Some signals (e.g. temperature) are single valued voltage signals. Others, such as RPM and injector controls, are time based signals that require pulse width timing. All signals are referenced to a common ground point. The injector pulse control signals are slightly trickier since there are two of them. However, only one of them is on at a given instant of time. All that is needed for fuel consumption is the running sum of the pulse widths. The two signals feeding the digital timer will simply be added using an analog adder circuit so only one PIC timer channel is needed. The pulse will also be fed into an AND gate with the chip clock signal because of the nature of the on chip timer circuitry.

The dynamic range of most signals will be between 0 and 5V. However the injector pulses are 0 or 12 V digital signals. To preserve the dynamic range constraints imposed by a 5V power supply, the injector pulse buffers will be designed to clamp the signals to 5V. By the nature of the RPM sensor, the RPM signal is not a constant amplitude frequency modulated signal and can have a large dynamic range. To account for variations in amplitude, a 5V hard limiter will be used to saturate the RPM signal to constant amplitude. While the original signal is sinusoidal, the amplitude values are irrelevant and the frequency data will be preserved by the hard limiter.

The power supply could prove to be the most difficult component to construct. The standard automotive power supply (nominally 12 V lead-acid battery) is charged by an alternator and the actual battery voltage can vary a few volts depending on a myriad of conditions. A regulated 5V power supply is needed for the MCU and most other components. It is desired to try to use single supply 5V opamp chips if possible. If dual rail power supply is necessary, an inverting switched-capacitor voltage regulator can be used to generate a -5V signal. A low-dropout voltage three terminal voltage regulator will be used to clamp the battery voltage to 5V. Large decoupling capacitors (50-300 uf electrolytic) will be placed near the regulator to filter large noise spikes. Smaller valued (1uf and 1nf) caps will be placed near chip inputs to filter out small higher frequency noise spikes. Worst-case scenario, if power supply noise cannot be removed, a provision will be made to use independent 5V batteries.

Ultimately a PCB will have to be made for the final circuitry. This will be at least a two layer board, but ideally a dedicated ground plane may be desired to reduce noise problems. This will be designed after the working prototype is tested.

MCU structure

A PIC16F77 microcontroller will be used to sample, process, and initiate the serial data transmission sequence. It has built in memory-mapped 8channel x 8bit A/D converter, two digital timer channels, and a UART. Each peripheral is controlled by its respective control and status registers. It has a very small fixed instruction set that can be used to

make assembly level programs which will be downloaded to an on chip instruction cache by a programmer. The program memory is 8Kx14bit word length, and the datapath has 360 bytes of RAM.

Wireless radio link

A Maxstream 9X PKG-R serial radio modem link will be used to transmit the data sampled and processed by the MCU. It has a maximum of 9600kps of serial data throughput and can transmit data wirelessly up to 3km with a direct line of sight. It readily interfaces to the MCU UART via output pins using standard 5V TTL/CMOS voltage values. The two components will be connected using a DB-9 null modem cable.

MCU Programming and Program Structure

The central component in our system is a Microchip PIC16F77 microcontroller. Programming will be done in assembly and instruction data will be stored in the on-chip EEPROM. The PIC will execute a loop consisting of a polling phase and a serial transmission phase.

There are two different types of signals to be polled. The first, and simplest, are the discrete voltage signals attached to the eight ADC inputs. The PIC16F77 has only one ADC. All eight inputs connect to this single ADC through an eight position switch, controlled by a status register. In the first part of the polling phase, this switch will be cycled through all eight positions, storing the value read at each position to a dedicated memory location.

In the second part of the polling phase, the pulse timers are polled. There are two pulse signals. One is the EFI (electronic fuel injector) signal and the other is the RPM (revolutions per minute) signal. After the two pulse timers are polled, they are reset to zero. The EFI signal is set whenever the fuel injectors are active. The integral of this signal gives the total fuel consumption. To approximate this integral, one of the PIC's timers counts whenever the EFI signal is active (EFI AND CLK is the counter's clock signal). The value in this timer will be proportional to the amount of time that the EFI signal was active during the latest execution loop. The sum of these values would be proportional to the total fuel consumption. After conditioning, the RPM signal is a square-wave. To measure the frequency of this signal, a second timer on the PIC uses this square-wave directly as a clock. When polled, the value of the timer will be proportional to the average frequency of the RPM signal during the latest execution loop.

RS232 serial data transmission occurs after the polling phase completes.

The total duration of the execution loop should be long enough to allow at least several dozen incrementations of the RPM counter. This translates to 50 to 100 ms. Therefore, the PIC16F77's maximum allowable 5 MHz instruction frequency is certainly overkill for out application. A slower clock may be used to cut down on no-op time latency.

ADC and Timer calibration

In order to interpret the raw sampled data signals from the MCU, the PCM values will need to be correlated to analog values. Initially this will be largely done on the bench test using either a function generator for time based signals or using a test scheme for physical value sensors (e.g. MAP sensor).

Timing considerations

The PIC micro controller communicates with the modem via a serial RS232 link at 9600 baud. This corresponds to 1200 bytes/second, which is .8333 ms/byte. The PIC on the other hand, has a 200ns instruction cycle. After each byte is collected, the data be transmitted to UART. It remains to be investigated, but it seems like we will be able to do put data straight from ADC to UART, bypassing the memory. We estimate a worst-case scenario, in which the time needed in order to sample the data and send it to UART will last about 20 instructions plus the ADC time. However, this still takes only 8us/byte, compared with a much slower .833ms/byte serial data. Since we cannot send more, 1ms/byte is also the limiting factor for the amount of date we can process. This also means that data will be obtained from the sensors roughly every .9ms, which is sufficient for most data, except for the fuel injector control signal pulses. The injector pulse widths last for 2-4ms per pulse and appear approximately every 5-10ms depending on engine RPM. The injector pluses will be have to be sampled and summed continuously, during the “dead” time (roughly 7ms) that we have to wait to send serial data. Finally the value will be transmitted as the last byte, after all the other data. If the register that holds the sum will overflow, the counter clock will have to be pre-scaled by some amount.

Data packets and encoding

We will send a header consisting of one or more bytes, followed by 8 data bytes in the order that is specified by the polling loop. The last piece of data will be the current value of the pulse width of the injection pulses. This way we will receive 8 data bytes plus the header byte(s) per frame of data. Considering a transmission speed of .833ms/byte, we get 7.5ms/frame, or 133 frames/second. This is actually more than we need/care to have, so most of data will be averaged out before they are displayed. This will help reduce the errors in data due to dropped packets or bit errors caused by transmission. Ideally, data should be synchronized to an RPM value. However, since the RPM changes on the order of a few seconds, the data bandwidth should be sufficient to guarantee close to “real-time” accuracy.

Post Processing

Once the raw data is received from the MCU, we would like to process it and display it textually and/or graphically on a laptop computer screen. After calibration the raw data will be accessed in a stream and processed and displayed as quickly as possible. The data will then be stored for to be used for additional analysis at a later. The data stream can be used to drive software “gauges” arranged as a simple GUI.