DIY ion sensing ignition subsystem

Ville Vartiovaara
Ville.Vartiovaara@hut.fi

July 3, 2002

Contents

1 Preface **2**
- 1.1 Disclaimer . . . 2
- 1.2 General (Important) . . . 3
- 1.3 What is ion sensing (in this document)? . . . 3
  - 1.3.1 Simple.. . . . 3
  - 1.3.2 ..and efficient . . . 3
- 1.4 Benefits (to mention a few) . . . 3

2 Theory of operation **5**
- 2.1 Prerequisites . . . 5
- 2.2 General . . . 5
- 2.3 My version . . . 5

3 First implementation, laptop-based test bench **7**
- 3.1 Detailed hardware . . . 7
  - 3.1.1 High voltage side . . . 7
  - 3.1.2 Isolation . . . 7
  - 3.1.3 Noise reduction and signal conditioning . . . 8
  - 3.1.4 Analog-to-Digital conversion . . . 8
  - 3.1.5 Crank angle sender . . . 8
- 3.2 Software . . . 8
  - 3.2.1 Basic structure . . . 9

4 Second implementation, MSP430 microcontroller **10**

Chapter 1

Preface

1.1 Disclaimer

This document is a description of how the author has built an ignition timing feedback subsystem that uses ion sensing. The system is being used solely for his private experiments.

If you intend to use this information in any way, you should find out what you are allowed to do, since some companies have patented this technology. The author has nothing to do with these companies, and gives no guarantee for the authenticity or correctness of any information presented in this document.

All information in this document is for informational purposes only; to give an idea of how a simplified ion sensing feedback system can be built. If you use this information for anything, you do it at YOUR OWN RISK, the author takes NO RESPONSIBILITY for any harm or damage caused by doing so.

Neither can the author be blamed for using bad English.

1.2 General (Important)

This is a PRELIMINARY version of a project description and not complete at all; it is even missing several chapters. This version is published to inform the readers of the existence of the project and give some basic information about it, while more complete version is to come when the writer has time to make one..

1.3 What is ion sensing (in this document)?

1.3.1 Simple..

A method to get to know whether ignition is too early or too late, on a cycle-to-cycle basis. No extra intrusive sensors, no engine modifications. The technology can be applied to any internal combustion engine that is ignited by a spark plug (or has a similar pair of electrodes installed ;), independent of the engine layout and fuel.

1.3.2 ..and efficient

We get to know the crank angle where the pressure is at its maximum. If this PPP (Peak Pressure Position) is kept constant (depends only on motor design) under all conditions, we can suppose to have the ignition "perfectly tuned" all the time. The same could be achieved by installing a high pressure sensor in the cylinder head next to the spark plug, but it is seldom possible due to both limited space in the block and also the high mechanical stress applied to the assembly. Ion sensing uses the spark plug as an intrusive sensor. We apply a voltage of 100-400Volts across the spark gap just after the ignition, and as both the combustion flame and also ionized resultants conduct a little electricity, we can measure the current via the spark gap, and get a curve, where we can extract several parameters such as PPP.

1.4 Benefits (to mention a few)

Traditional ignition control relies on predetermined rules (tables of factors). These rules are coded in the control system (electronic or mechanical) upon manufacture. The correctness of ignition timing in road conditions depend on how well the designer has been able to reckon with all the parameters that affect the burn rate and thus the 'ideal' advance. Not to mention the effect of aging of the engine.

Usually the timing can be estimated quite well (with a modern ECU) during medium or high load, and when air humidity is relatively low, as in the prototyping lab. But, as these high load conditions in dry air are quite infrequent in normal use, the correct timing at light load (lean mixture) and varying humidity is of the highest importance, especially when economy, smooth operation, high instantaneous power, low emissions and durability are important factors. This is the case in all normal cars.

When an ignition control system uses ion sensing, it can, after every single combustion, "see", if the ignition was too early or too late. The predetermined

tables, which can be substantially simpler than with traditional predictive control, are then updated to reflect the current conditions. So a simple table may still be used, but if a fast enough feedback system is used, the table is only used upon cranking and similar conditions, where no reasonable ionization current may be present after ignition.The ignition control system is simplified a lot, as the amount of discrete sensors is reduced to a minimum of a crank position sender (TDC sensor or better).

Due to the quite demanding nature of the ion current signal (the shape of the current that flows across the spark gap during combustion) it is reasonable to use ion sensing only as a correction factor. It may be difficult to get reliable data after every combustion, so the system may be altered in a way that the feedback unit spits out an averaged error angle (how much the ignition is too late or early, in degrees) of e.g. ten revolutions, and the original control unit may then tune the overall ignition table, being able to do this after every cycle. With this technique the ignition tuning steps are 'blurred' to minimize the effect of a erroneous error angle, still maintaining short response delays to changing conditions such as mixture enrichment during acceleration.

Chapter 2

Theory of operation

2.1 Prerequisites

Prior to trying to understand the issues covered in this document it is advised that you spend a little time e.g. searching the web and finding general information about ion sensing in ignition control, especially if you are not familiar with this kind of technology. I could give a bunch of URLs here, but they would, sooner or later, become outdated. For your convenience, please use a search engine instead.

2.2 General

We can learn from deeply studying the nature of an internal combustion engine, that the signal we get from the spark plug can be approximated as a sum of two Gaussian curves [1], the first of which is due to the charge carried by particles in the flame (a.k.a flame ionization). The second presents the pressure in the cylinder (Fig. 2.1). This is because the voltage we apply across the spark plug electrodes ionize the results (not-yet-escaped exhaust gas), and the amount of charge carried by these ionized particles is proportional to the gas pressure around the electrodes. And as we are interested in the point (crank angle) where pressure reaches its maximum, it is the peak position of the latter term. The difficulty is to extract that position, and the first method that comes to mind is brute curve fitting. Gaussian curve is of type $ae^{-b(x-c)^2}$, so the whole curve in this model is a sum of two exponential functions. This leads to huge amount of calculations in curve fitting, so we either have to reduce resolution to an unusable amount, or simplify the process in some way. This is because the calculations have to be done approx. 50 times per second per cylinder at a resolution of one degree.

2.3 My version

The curve fitting method can be simplified mathematically to make it possible to do the calculations in real-time with moderate computing power (an ordinary microcontroller). Despite this I thought that it would be nice to be able to build

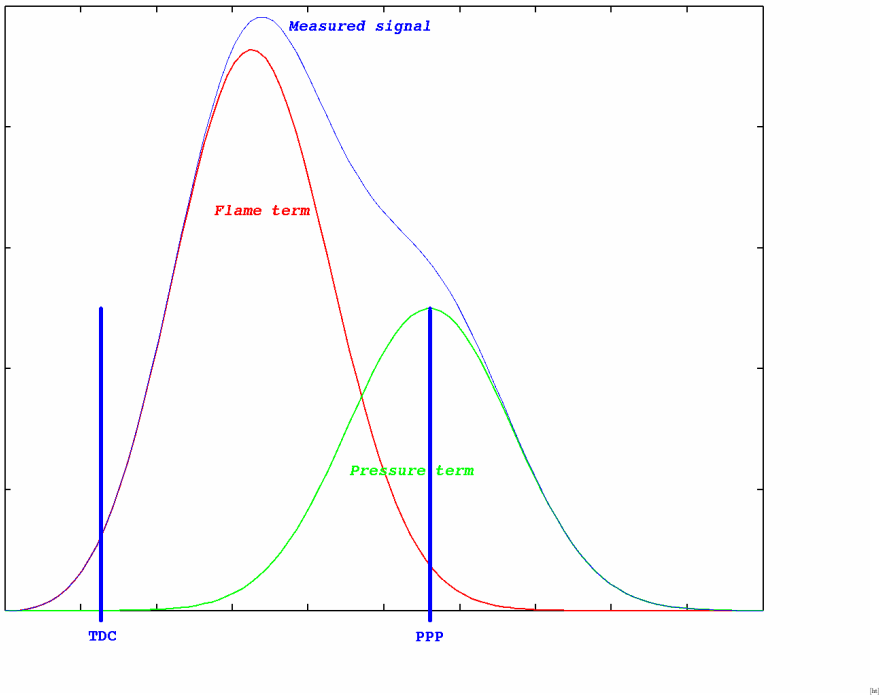

Figure 2.1: The measured signal and its components

a functional system without first having to bury oneself in maths. So I decided to try a simplified version of curve fitting, something similar to what I do myself when trying to approximate the peak position from a curve that is printed on paper. The model is:

- find the highest peak of the whole signal (giving approx. the peak of the flame term)
- do a primitive curve fit to find a Gaussian curve that fits to the first peak's rising slope
- extract that curve from the original signal
- find the highest peak of the remainders
- trust in the result

This method gives surprisingly correct results, at least so correct that I couldn't say them to be incorrect when visually comparing to the original signal. So it is time to put it to work. Oh, and of course, when the system is ready (all the hardware built), the software can always be enhanced to give more accurate results. The point here is to get a good enough algorithm to start with.

Chapter 3

First implementation, laptop-based test bench

Having written the algorithm in C on Linux I wrote a program that read signal from an 8-bit ADC attached to a laptop's parallel port, extracts the PPP and shows the result on screen. When the engine was idling, the laptop, that was on top of it, had a meter on its screen showing PPP that was varying from somewhere near 15 degrees to over 40 degs occasionally. The engine is not in perfect tune..;) All the time I had an oscilloscope attached to the signal cable to be able to verify the results, which seemed to be correct. When the engine was loaded a bit, the PPP got more stable, so the system was working. The problem was that as I only had an 8-bit ADC, and the dynamic range of the signal is much wider, I had to manually re-tune the preamplifier not to get the signal clipped during conversion at a higher load. Anyway, the algorithm was proofed to work as expected.

3.1 Detailed hardware

3.1.1 High voltage side

To be able to get the signal from the spark plug, you have to apply a relatively high voltage to it *after* ignition, and measure the resulting current, which is of order 1mA. For this purpose I built a simple switching flyback inverter based on the 555 timer and a 10A mosfet (high rating to allow for bad design without smoke :) Some systems use a plain zener diode and a capacitor in place, but this is only possible when the car is fitted with four terminal coils or similar with entirely isolated secondary side, which are not quite common. At least my car doesn't have one..:)

3.1.2 Isolation

The 400V must be applied *to* the spark plug, which means that we do *not* want to get any current (especially at 15kV) *from* the plug during spark. So we must have a diode that has reverse breakdown voltage substantially higher than the ignition system's spark voltage. I put 24 1kV avalanche diodes in series to get

Vr of 20kV, which is enough for my old coil ignition. The installation must be done with great caution, to make sure that no one gets hurt by either the high ignition voltage (> 10kV) or the ion sensing voltage (400V) when the engine is running. I used silicone tube to cover the diode chain.

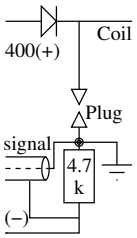

Figure 3.1: High voltage assembly to the spark plug

3.1.3 Noise reduction and signal conditioning

As the operation environment is quite demanding in the sense of protection from electromagnetic interference (EMI), the signal present across the resistor near the spark plug is transferred in a coaxial (microphone) cable to the filtering and preamp unit. I use a 4700Ω resistor as the current sensor, so the amplitude of the signal is somewhere between 3 and 5 Volts at its highest. The filtering must be minimal, because the useful band of the signal goes up to 30kHz, so I use a simple RC low-pass filter with -3dB point somewhere around 50kHz. Also minimal distortion is of utmost importance, as any phase shift in the major components result directly in wrong PPP estimate. The filter is built into a two-stage amplifier consisting of a unity-gain inverter and an adjustable-gain amplifier. This is because the signal present on the resistor leads is from 0 to -5V. To make the signal usable, it must first be inverted, and this is carried out at its simplest by constructing an op-amp-inverter with one half of a LM358. This design also provides us with a substantially high input impedance, which is important due to the nonlinear nature of the signal source. Then the other half is used to create the amplifier. The resulting signal is positive-up, 0-to-5V signal, which can be directly fed to the ADC.

3.1.4 Analog-to-Digital conversion

In the first version I used a 8 bit ADC from Texas Instruments they offered me as a sample. It was chosen because I wanted to get easy interface via the parallel port. The ADC was of parallel-out type, so I just needed to put wires between the parallel port pins and the pins on the ADC. The return path (ground) from the parallel port was blocked with a low-drop diode to prevent burning the port driver if some of the data pins were mistakenly pulled low while the ADC was sending high.

3.1.5 Crank angle sender

To make the sampled signal useful, we also need some timing information. I use a simple aluminum plate mounted to the generator belt pulley at the end of the crank angle.

The rotation of the plate is encoded with an optical switch and sent to a pin in the parallel port. The design of this sender is not robust at all; it is just simple to make, whereas the final version must have a hall effect detector and an appropriate metal plate. But for a prototype this is good enough.

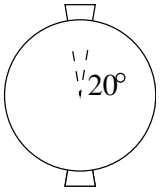

Figure 3.2: Crank angle timing plate

3.2 Software

I have no knowledge of an existing system that would be using this technology. When writing software for the laptop implementation, I tried to make things

as simple as possible, and, first of all, fast. The goal was to keep the amount of instructions per two engine revolutions (four-stroke) under 5000, and the result was approx 3000. This means that if the engine runs at 6000RPM, you need a processor capable of doing 150000 instructions per second for the signal interpretation. So my 100MHz 486 was really fast enough..;) So I put a nice analog meter on screen instead of a plain ASCII output.

I don't provide any sources at this point, as this first version was really a prototype in all respects. All I say is that it was coded in C with Linux gcc and basic X11 libs, with optional real-time scheduling. As soon as I saw that the algorithm was working, and didn't need all too much computing power to keep real-time, I moved forward to the next implementation. Anyway, the basic structure of the software is as follows:

3.2.1 Basic structure

Data logging cycle:

- keep discarding samples until a high spike due to spark occurs in the signal
- when the spark peak goes down, start logging sampled data for further processing
- log data until 120ATDC (approximated from the amount of samples collected during signal from timing plate encoder is active (from TDC to TDC+20 degrees)
- when ready, hand over the logged data for PPP extraction

PPP extraction routine:

- find the first peak location
- fit a Gaussian curve to that peak's rising slope
- extract that curve from the original data
- find the remaining signal's peak position
- transform the peak coordinate (sample number) into degrees ATDC with the help of knowing from which sample to which sample the timing plate signal was active (from TDC to TDC+20 degrees again)
- put the PPP on the screen ;)
- adjust sampling speed if the amount of samples was too small or too high (adapt sample rate to changing RPM)
- pass control to the sampling routine

And that's it. The routines can be enhanced in many ways, such as controlling the sampling with a PLL (Phase Locked Loop) so that no software RPM sensing needs to be done, adding automatic signal level control (when used with a 12-bit DAC) etc.. But it works.

As it became obvious that the algorithm is promising, and that I needed more resolution to the A/D conversion, it was time for the next implementation with Texas Instruments 16-bit RISC microcontroller MSP430.

Chapter 4

Second implementation, MSP430 microcontroller

Ok, now I'm working on an integrated version that is based on MSP430F149, the Texas Instruments' 16-bit RISC Flash microcontroller. The chip operates at clock frequencies of up to 8MHz and has many very powerful features that benefit this project. It has a 12-bit ADC (8 MUXed inputs), 2 timers and a hardware multiplier module on chip along with many other integrated peripherals. All the peripherals share the same 2048byte memory, so the API is very simple. F149 has 60kB of Flash memory with serial (in-system) reprogramming capability. The application code can also update the flash memory autonomously. These are the main points why I decided to use just this chip in my project.

The software is mostly assembler, but there are still some lines of C where speed isn't important. The program structure is roughly the same as in the laptop version, only the sampling routines are entirely different. The timing is done by having two IRQs sent by the timer and the ADC. So it's a bit more 'multi'tasking..;) Anyway, to keep this document preliminary, I won't write about this any more.. Hope I'll find the time to finish this document soon.. Just now I'm too busy debugging the new system that has all the basic functionality (sampling, PPP extraction, PPP sending via RS232 interface (gives correct numbers..:)) but needs more..

Bibliography

[1] L. Eriksson. *Spark-Advance Control by Ion-Sensing and Interpretation http://www.fs.isy.liu.se/ larer/Projects/main.html.* 25 Nov. 1998.