DIY ion sensing ignition subsystem

Second edition

Ville Vartiovaara
Ville.Vartiovaara@hut.fi

November 25, 2002

Contents

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

This second edition covers, in addition to the basics found in the first one, also the MSP430-based implementation of the ion-sensing module. It is not written in great detail, but the main structure of the system is being discussed. This is a compromise between a poster and a tutorial, the former being useless and the latter taking ages to write..

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 100-400 Volts 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. 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 (weighed) 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 an erroneous angle, still maintaining fast response 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 (or three for some purposes) 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 represents the pressure in the cylinder (Fig. 2.1). This is because the voltage we apply across the spark plug electrodes ionize the resultants (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

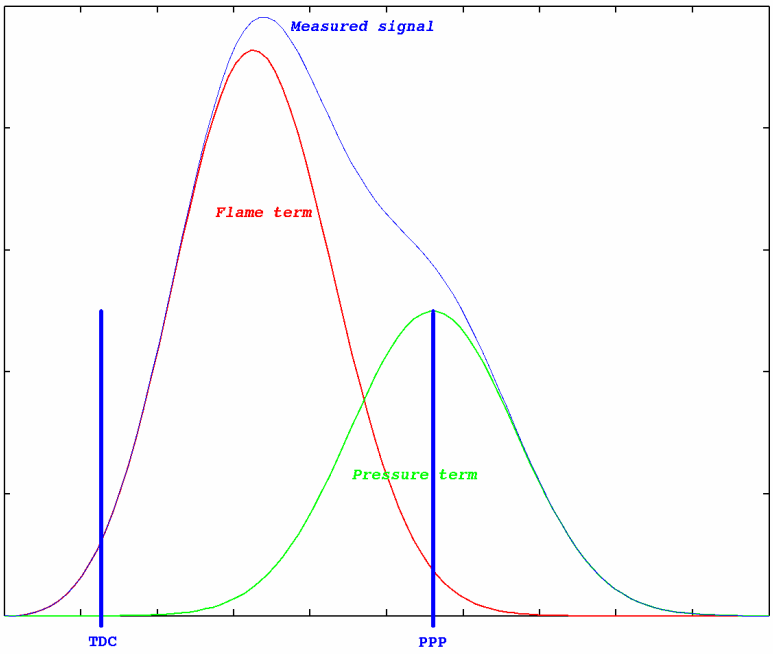

Figure 2.1: The measured signal and its components

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 working (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

Laptop-based test bench

Having written the algorithm in C on Linux I wrote a program that reads signal from an 8-bit ADC attached to a laptop's parallel port, extracts the PPP and shows the results 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..:)

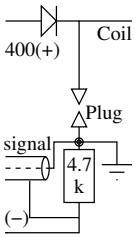

Figure 3.1: High voltage assembly to the spark plug

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 controlled 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.

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. *In general, this way of getting the ion current converted to voltage is not optimal, especially as a low-noise high-voltage isolated floating-output DC/DC-converter is needed. For better performance (and less words is module descriptions), the MSP430 version implements an enhanced version of the high-voltage circuit. See later in this document.*

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.

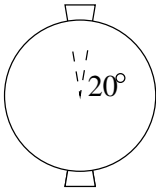

Figure 3.2: Crank angle timing plate

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

Stand-alone system on MSP430

4.1 General

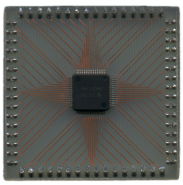

Figure 4.1: DIY PQFP64-to-0.1in pitch -adapter

MSP430 is a family of 16-bit RISC microcontrollers made by Texas Instruments. I use the MSP430F149 that has 60kB of FLASH and 2048 bytes of RAM in addition to a range of peripherals, of which the ones in use in the system are being discussed later. The hardware of the system is now fully functional, the development being mostly focused on the algorithms. The software works too, but work needs to be done so make it more robust in varying conditions. The latest change in the software was an algorithm rewrite where all routines became 16-bit ones, thus rendering automatic gain control obsolete. The reason for some routines being initially written in 8-bit world is that otherwise the sampled window has to be stored with 16-bit accuracy, where only 8 of them are actually needed for precise results, so we're wasting memory. But as fast transient response is critical, it has become obvious that the best solution is to get rid of bit-shifting etc. and do everything in the 16-bit world.

4.2 Hardware

4.2.1 General

The hardware of the system gets often minor improvements that would make a whole bunch of schematics useless in a split second. That's why I think the only reasonable way to make this document is to give the main characteristics of the whole system and then concentrate on the modules and describe them in detail, because they are what distinguishes this system from any other mixed signal processing unit, not the signal wiring between them. The most important thing around here (too) is EMI protection and suppression, which definitely can't be described in the schematics and depends on so many factors that it's out of the scope of this story. All the important pieces should be there, and if they aren't, feel free to ask for them.

4.2.2 Overview

First of all, as the packaging of the MSP430 (QFP) is quite uncomfortable when one needs to use scope probes with it, an adapter has been made (Fig. 4.1) that has 64 traces each going from a leg to a pin on the edge. The adapter is then inserted into a 4x16-pin square socket on the proto board. The board is mounted in an aluminum enclosure, and all external *and* internal wiring is done with shielded cable, either coax or multiwire. The in the enclosure is a bunch of 2.8mm coax, but without it, no protoboarding with long wires and space for experimenting is possible due to severe crosstalk between digital and extremely low-power analog signal wires. The board is divided into two separate cabinets by an anuminium ground plane. The first one is populated by the power supply and serial communications, timing and signal conditioning hardware. This means that there are two regulators, one MAX232, one CMOS hex inverter, one NPN transistor and a handful of passive electronics and connectors. The other cabinet is for the MSP430, its peripheral components, diagnostic leds and a pushbutton that puts the system into program mode allowing for in-system firmware update. Figure 4.2 gives an overview on the basic structure of the system [1].

4.2.3 Peripherals in detail

Ion sensor

Power supply design The process of ion sensing begins with applying the bias voltage across the spark plug gap and measuring the resulting current. The simplest method when counting amount of electronics involved, is to get the bias voltage by charging a capacitor that is connected in parallel with a zener diode between the ignition coil and the plug. Although being the simplest and cheapest method available, it still poses some restrictions to the design of the ignition system to work properly. It is required that the system is such that the plug wire is permanently connected to the secondary of the coil, which in turn is always grounded, thus eliminating the possibility to have a distributor. So I decided to stay in the basic idea of having a separate DC/DC-converter to do the job. What is new to this version in comparison to the first one, is that no floating-output power supply is needed anymore. Instead, by utilizing a simple current mirror, a non-floating one can be used. An additional benefit of the ability to use a simpler design in the converter, is that noise reduction is much more efficient as a big capacitor (and a small one also) can be connected between the power lead and ground, which is not the case with flying outputs, where the resulting noise must be eliminated later, with dramatical effect on signal quality.

Current mirror The ideal model of a current mirror (fig. 4.4) is that it makes a current I_{sense} that is equal to a 'programming current', I_{ion}, as can be easily seen when paying attention to the base voltages of the transistors that are equal in type. There are several parasitic effects that make the real-world case

[1] At the date of publishing this document there is also a Microchip MCP2510 CAN2.0B transceiver and a 3.3V CAN bus driver for it on the board. This additional hardware enables the module to be hooked to a max. 1Mb/s CAN network.

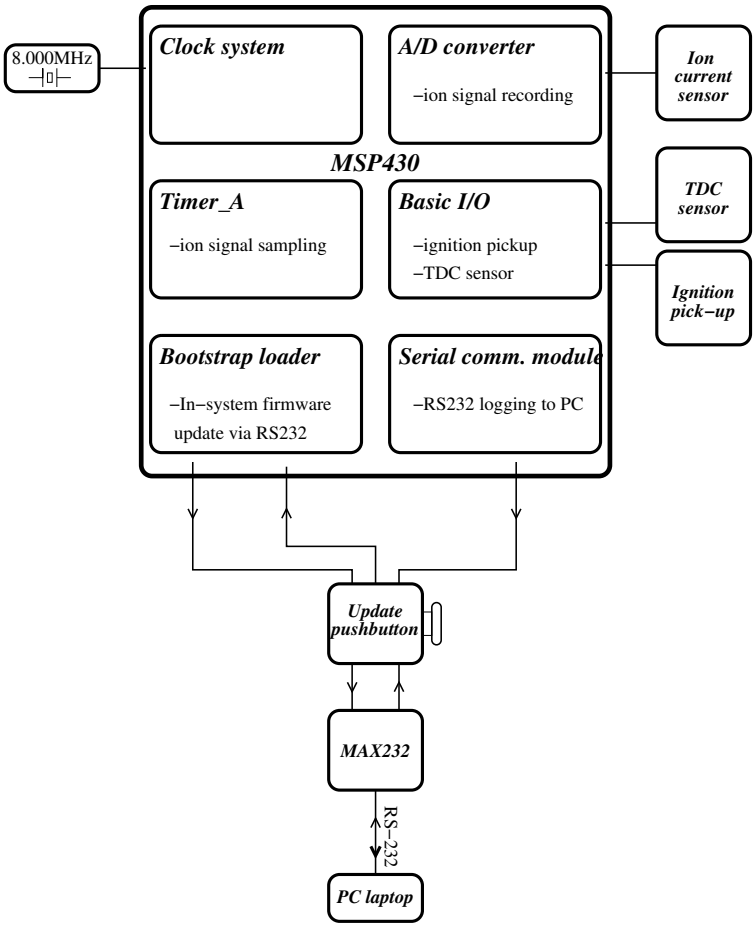

Figure 4.2: Subsystem block diagram

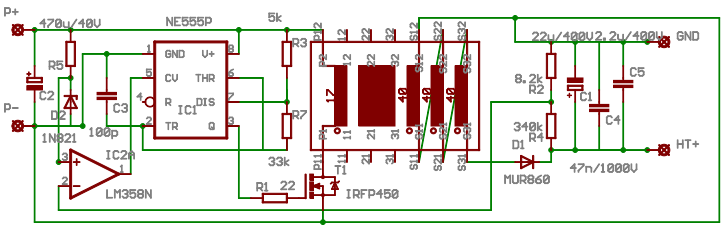

Figure 4.3: Power supply diagram

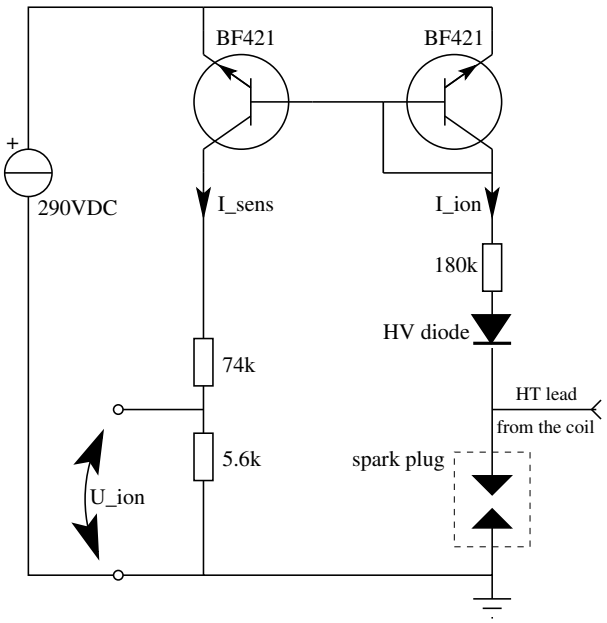

Figure 4.4: Current mirror connected to the spark plug

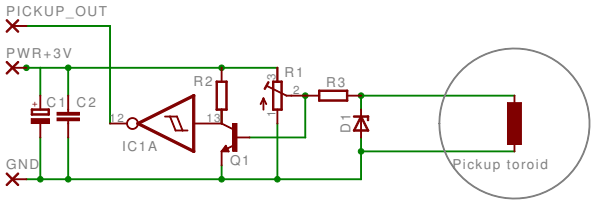

Figure 4.5: Ignition pickup module

differ from the ideal one (as usual), but in this configuration the error remains neglible throughout the dynamic range of the signal. The U_ion in the figure is the voltage signal to be brought to the ADC via a coaxial cable. In the ideal case, this voltage could be expressed as $U_{ion} = 5600 * I_{ion}$, and the real-world case isn't far from that. The U_{ion} is then split with a resistor divider (trimmer potentiometer) on the main board, just after the input voltage clamp diodes, so the basic setting of signal level can easily be adjusted. The 180kΩ resistor is there to prevent the power supply output capacitor from discharging via the spark plasma thus lengthening the spark by several milliseconds and frying the transistors is seconds (nice fireworks)..

Ignition pick-up

When the cylinder fires, the sampling routine waits for a preset time, e.g. 1ms, and starts data logging. In order to give the routine the timing information, an inductive pick-up sensor is fitted. It consists of a toroidal inductor (120 turns on a noise suppression ferrite core) fitted around the spark plug lead and

a transistor with biasing trimmer potentiometer on the base. The pulse from the inductor triggers the transistor, which in turns drives an inverter to achieve square-wave output. The sensitivity of the sensor can be adjusted with the biasing trimmer.

TDC sensor

The TDC timing plate is the same as in the first version, and shown in figure 3.2. An optical switch turns off when the nose on the plate passes by, and gives a TTL-pulse that lasts exactly 20degrees. This signal is then driven through an inverter, and further to the MSP430. The signal is used for TDC and RPM detection and sample rate correction upon RPM changes.

4.3 Software

The basic structure of the software is similar to the one in the laptop version. The major difference between the two is the hardware timer/interrupt -based sampling routine in the MSP code, where the software only updates certain variables to control the sample rate when needed, and then fetches words from the AD in the interrupt handler. Another significant change is the 16-bit arithmetics in the peak-finding algorithm. Due to the rewrite, there is a lot of C code left, so the next thing to do is to put the most critical parts into assembler and optimize. Another thing to do is to make use of the MCP2510 CAN controller that is now on board and connected to the MSP430 via 4Mbps SPI line. The controller supports CAN baud rates of up to 1Mbps and extended messages. The goal is to make the subsystem send PPP information via this medium to other nodes in the system, e.g. the ECU and maybe a laptop, instrument panel etc...

4.3.1 The code itself

It's probably the best solution that I don't put any actual code available with this document, because it'd become outdated the day I next change something in the code. But in case someone still wants to get it (licensed under GPL), tell me and I'll send it.

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.