Using DOS and Quick BASIC to control hardware in real time.

DOS? Are you crazy Paul? :)

I was investigating Visual Basic for Windows and I was dismayed at the complexity. It is way over kill for what I want to do. Build an EFI system that is easy to program in BASIC. Also display engine parameters on a note book or net book screen in real time. Notebook and net book screens are very large in terms of number of pixels and cheap. They can display dozens of engine parameter in a relative small space.

Everything including a full blown EFI can be written in hours using simple to learn Quick BASIC. Quick BASIC is a compiled language and the programs are small and lightening fast. QB/DOS can handle real time hard ware interrupts which were used in my moving map program. The code is already written to read a USB GPS.

To do an EFI system it is necessary to add hardware of course. One must design a USB interface board with addressable bit ports to control injectors and coils. There are probably some on the market already if I bother to look. After that an amplifier must be added to control the injectors. That could be as little as an FET power transistor. The modern coils can be controlled with a computer level bit.

20 some odd years ago I wrote a fully functional aviation moving map program in Quick Basic that ran on DOS driven by a GPS using an RS232 serial interface. The GPS was interrupting the QB background program once a second. Unfortunately RS232 serial interfaces have disappeared off note book and net book computers.

DOS is a close as anyone is going to get for a real time OS. The down side to DOS is it's limited memory management capability. For for my use it is quite adequate. Running on a modern computer it is lightening fast bordering on instantaneous.

It is the combination of DOS and Quick Basic. Both allow direct connection with the hardware. Quick Basic has hardware interrupt handling built in.

That cannot be said about Windows or Linux. There is a multitasking scheduler in there that is making decision on how much of the resources you are allowed. You don't want a powerful multitasking OS. You want something small that gets out of the way of your real time program. DOS fits that almost perfectly.

I got the printer port working with the following real simple DOS QB program.

This 533 Mhz IBM Think Pad is ten times faster than most single chip computers used in your typical EFI system such as Tracy's or Megasquirt. It will do one instruction written in the high level QB programming language in only 2 microseconds.

```
ON KEY (1) GOSUB finished

KEY(1) ON

Start:

OUT &H3BC, 1
```

```
OUT &H3BC, 0

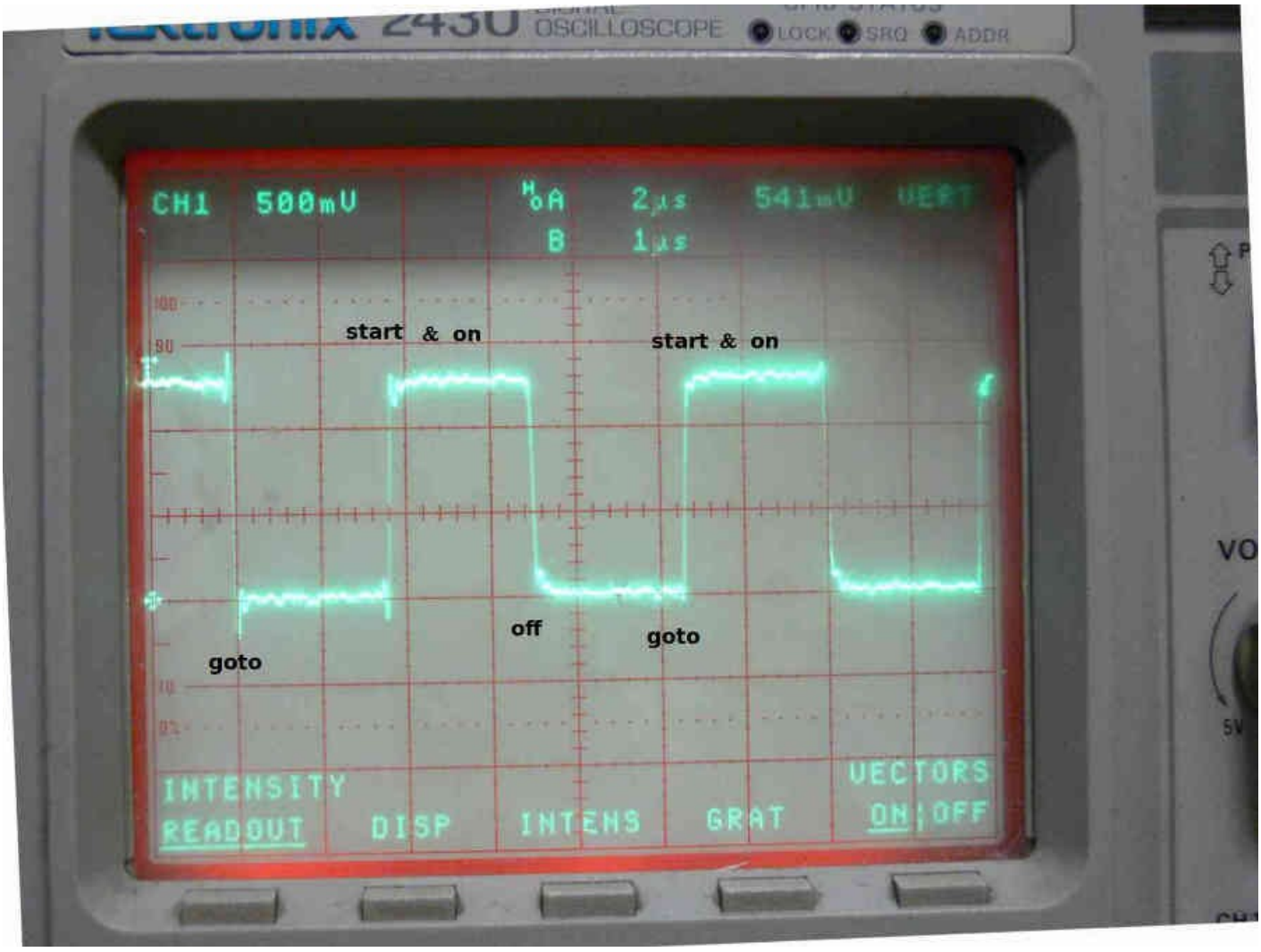
GOTO Start

finished:

END
```

It toggles the first bit (bit 0) on and off in less than 2 microseconds. This with 530 Mhz IBM Think Pad 560. It would be lot faster with some thing made 20 years later :)

Here is a scope picture of the program running.



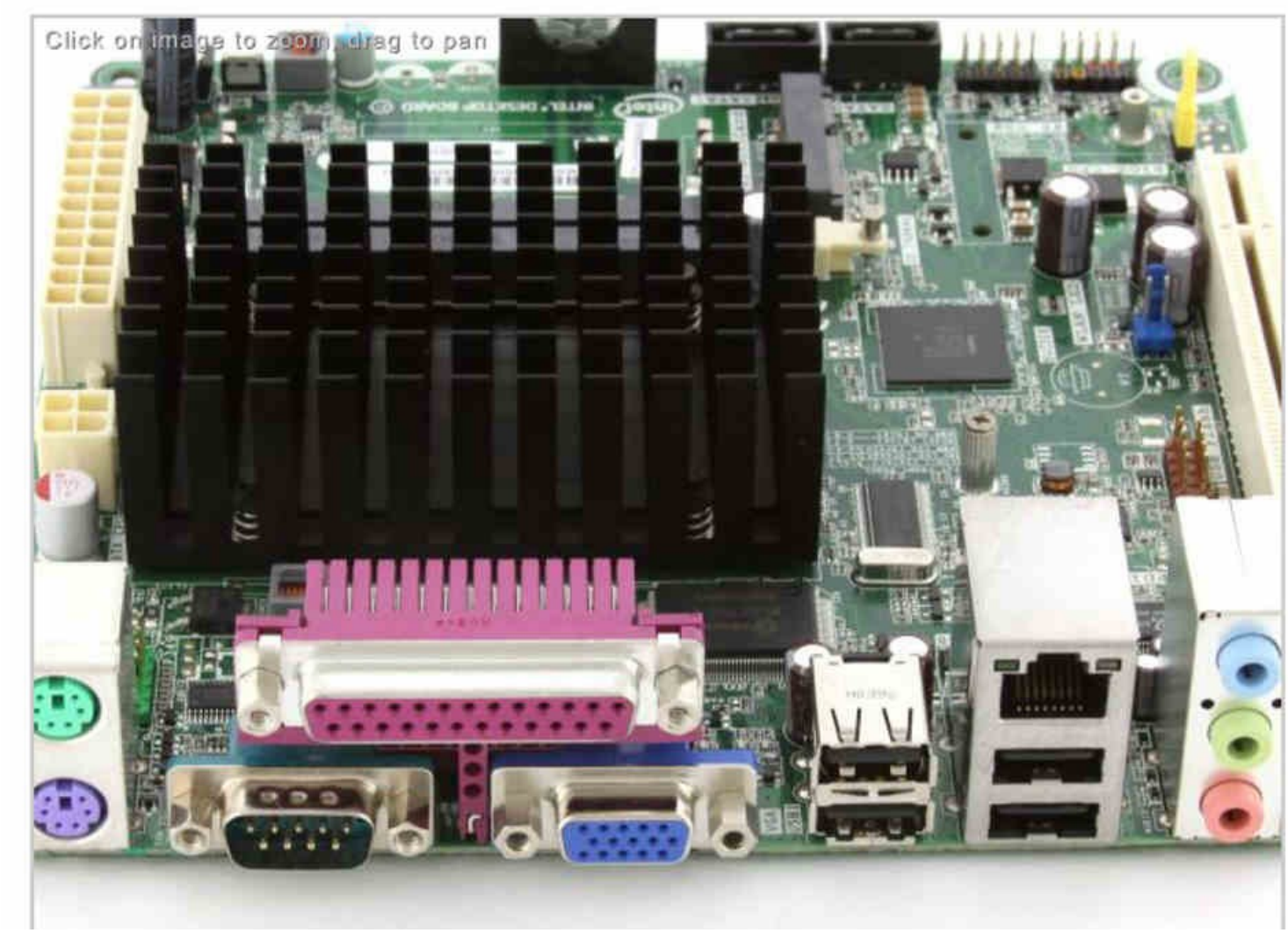
Since you need to delay a coil turning on and off for dwell 2 Milli seconds there is plenty of time to do other things.

The on key stuff is a hard ware interrupt so you can get out of the infinite loop. Press function key F1.

BTW this is with interpreted QB. Compiled would be a lot faster.

Note; it take only several hundred nano seconds more for the goto to execute.

I am fooling with a cheap PC miniATX mother board. They are ten times more powerful than the single chip computers and the cost is about $80. Running DOS they can be programed in easy to learn quick basic. http://www.rotaryeng.net/fuel-injection.html



If your micro ATX does not have a parallel printer port on board you can buy a PCI express card.

http://www.softio.http://www.softio.com/ic0652kb.htmcom/ic0652kb.htm

<

Make sure you get one with a DIN keyboard connector as USB keyboard won't work running DOS.

-

7500 rpm is 125 revs per second. That is TDC every 0.008 second. One micro second is 0.000001. That means 8000 micro seconds go by before the next TDC happens.

I am pretty close to getting the USB working with DOS and QB. That means you don't need a hard disk. You can boot DOS from a tiny thumb drive. The thumb drive is also large enough to store weeks of engine parameter data.

I have also figured out how to get 12 hardware interrupts working using a PC keyboard 8051 processor. One

can use a CD 4066 analog switch to simulate a function key press. QB supports these hardware interrupts.

Most single chip computers used in EFI systems only have 2 or 3 hardware interrupts.

Using hall effect sensors on the eshaft telling the computer when to do things with a hardware interrupt saves a lot of software.

The concurrence of the old software with new low cost hardware makes it possible for the first time to control complex hardware in real time with super easy to write applications in simple to program BASIC.

The big stumbling block as I see it is the lack of support for USB serial ports. That has recently changed. Apparently there are other people out there that see the elegant beauty of DOS.

Here is the break through. http://www.georgpotthast.de/usb/

Rather than spend 100's of hours learning VB I am going to look into this approach further. I have a real old IBM Think Pad that should run DOS no problem.

The less software between the ports and the software the faster the response. I got native DOS 5 running on that old IBM Think Pad I had in the hangar.

It has both RS232, USB and a printer port. A printer port in essences is an 8 bit parallel port so in theory you can use the Think Pad as an EFI computer. Four bits for coils and four bits for injectors. It is supported in Quick Basic by the POKE or OUT command. POKE address, value The address of the printer port in decimal and the value is a byte of data with the proper bits set to on. It is that simple.

The ON event is a hardware interrupt and there are several options that can interrupt the program on TDC e-shaft sensor. I need to test this and see if it works at 8000 RPM or 133 interrupts per second. Should be no problem for a 1.3 Ghz notebook running DOS and compiled QB.

The ON key event can trigger on any key board function key so you could hack the key board with a couple of small wires and short out a key with a solid state 4066 chip turned on by the the CAS. That gives at least ten hardware interrupts.

Excellent site on programming in quick basic in general.

http://www.petesqbsite.com/sections/tutorials/beginners.shtml

http://www.thinkpaddepot.com/

Here is the pin out for the 25 pin connector printer port on many old notebooks suitable for running DOS/QB and using it as an EFI computer. Many cheap, less than $100, small ATX mother boards also have a parallel printer port and would also work.

9.31. PS/2 PARALLEL PORT CONNECTOR

Connector	*Pin Number*	*Direction**	*Signal*
System end (DB25)	1	In/out	-STROBE
	2	In/out	D0
	3	In/out	D1
	4	In/out	D2
	5	In/out	D3
	6	In/out	D4
	7	In/out	D5
	8	In/out	D6
	9	In/out	D7
	10	In	-ACK
	11	In	BUSY
	12	In	PE
	13	In	SLCT
	14	Out	-AUTO FEED XT
	15	In	-ERROR
	16	Out	-INIT
	17	Out	-SLCT IN
	18		Ground
	19		Ground
	20		Ground
	21		Ground
	22		Ground
	23		Ground
	24		Ground
	25		Ground

Notes: *From computer

Source: IBM PS/2 Model 30 Technical Reference, page 1-126
IBM PS/2 Model 50 and 60 Technical Reference, page 4-179

See Also: 9.32. Centronics Parallel Connector
9.34. Parallel Printer Connector

Here are the addresses of the various ports.

Port	Address (Decimal)	Address (Hex)
Data Lines	888	378h
Control Lines	890	37Ah
Status Lines	889	379h

I need to experiment with input Pins 11 and 13 as potential CAS inputs. QB would see that as an error hardware interrupt. I am sure this goes through the assemble language BIOS routines and could be quite fast. I'll trigger these lines with a signal generator and measure the time it takes the Think Pad computer to respond. My guess it is down in the microseconds. The trick here is to get the HW interrupt and have a QB program toggle an out put pin with the OUT or POKE QB instructions. For those that don't have QB and are interested here it is in zip form. It will run in a DOS or command line window under XP.

All the I/O pins 2 through 9 are self explanatory and would be used to trigger coils and fuel injectors.

What I and several others are trying to do is add USB support to DOS. I am getting close. If I can get that to work it is a whole new paradigm in easy to program hardware controllers. However I would not wait for it as it might not happen. It makes no sense without DOS USB support.

.

H