



Author: Jeremy Leach Email: [ukc802139700@btconnect.com](mailto:ukc802139700@btconnect.com)

December 2006

## Introduction

This article describes the special integer maths behind a joint tachometer project that I've completed with Rex Lantz. Rex has done a general write-up of the project at <http://www.tach.rex-deb.net/>

The Tachometer reads the Pulse-width of an incoming pulse from the ignition, then calculates the RPM from this.

At first sight, this might seem an easy thing to achieve with simple maths. However because the PICAXE is limited to integer maths, getting an accurate conversion from Pulse-width to RPM is no simple task.

This article gives an in-depth study of how to get a conversion from the Pulse-width accurate to 1 unit of RPM. The basic principles involved can be used in similar maths problems.

A simulation showing the accuracy of the maths is given in the associated Excel™ spreadsheets:

*RPMMathsResults\_LowRPM.xls*

*RPMMathsResults\_HighRPM.xls*



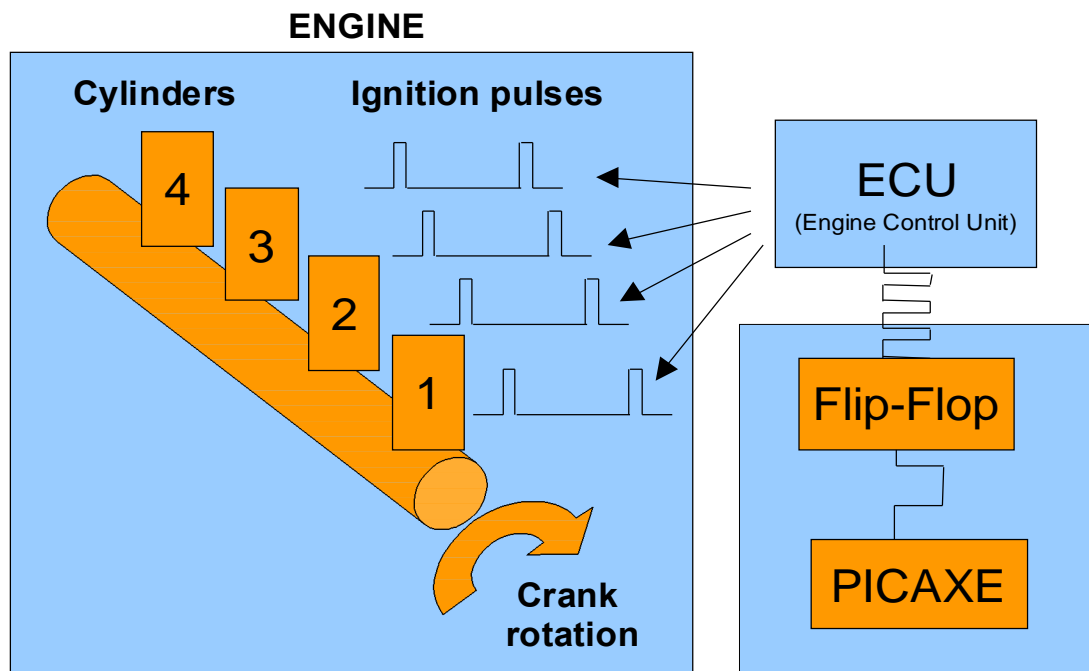
## IN THIS ARTICLE

- 2 So what's the problem?
- 2 An aside about the RPM formula
- 3 Bringing some order
- 3 The high revs ( $\geq 367$  RPM)
- 4 The low revs ( $< 367$  RPM)
- 5 The results

## ACKNOWLEDGEMENTS

- PICAXE is a trademark of Revolution Education Ltd.
- PICAXE Forum <http://www.rev-ed.co.uk/picaxe/forum/>

Figure 1:Source of the Tach Pulses



## So what's the problem?

The formula for the RPM turns out to be this:

$$\text{RPM} = 24,000,000 / (\text{Cylinders} * \text{PulseWidth})$$

..Eqn1

Where :

RPM = Revs Per Minute

Cylinders = The number of cylinders in the engine

Pulsewidth = The output of the *Pulsin* command using an 8MHz internal PICAXE clock.

The challenge with PICAXE maths is work within the following two constraints:

- A PICAXE uses integer Word maths (16bit) and not floating point maths.
- Calculation statements are evaluated from left to right. If the calculation of a result (intermediate or final) is greater than can fit into a Word (>65,535) then the calculation will overflow and the result of the calculation will be incorrect.

Evaluation of Equation1 is therefore very tricky!

Another problem when using integer maths comes

with division operations. If just the division operator "/" is used, then the remainders used by the equivalent floating point calculations get discarded.

Fortunately the PICAXE also possesses a modulus operator "%" that can be used to grab hold of these remainder values, and this operator is central to the methods used below.

## An aside about the RPM formula

Before delving into the detail to solving the problem, here's the explanation of the RPM formula:

Firstly, with typical four-stroke engines, there will be an ignition-pulse every two revs for each cylinder. The pulses to each cylinder are offset. See [http://en.wikipedia.org/wiki/4\\_stroke#The\\_Otto\\_cyc le](http://en.wikipedia.org/wiki/4_stroke#The_Otto_cycle)

Assuming our Tachometer taps into a signal that captures the pulses to all cylinders, then the more cylinders the more pulses per revolution of the crank. By timing the duration between ignition pulses you can work out the RPM of the engine.

Rex and I have used a flip-flop circuit to clean up the incoming signal. The pulse-width from the Flip-

flop output is the interval between ignition pulses. See Figure 1.

If each cylinder has an ignition pulse every 2 revs, then the ignition pulses per rev = Cylinders / 2.

The pulse-width we are measuring is the interval between ignition pulses, therefore:

$$\begin{aligned} \text{Pulse-width} &= \text{TimeForOneRev} / \\ &\text{IgnitionPulsesPerRev} \\ \text{Pulse-width} &= \text{TimeForOneRev} * 2 / \text{Cylinders} \end{aligned}$$

The Pulsin command used to measure the pulse-width has an integer result that I call PulseWidth. Each unit of the PulseWidth represents 10uS at a PICAXE clock of 4MHz and 5uS at a PICAXE clock of 8MHz. We want the Tachometer to be as accurate as possible, so we use an 8MHz clock.

So if TimeForOneRev is in seconds :

$$\begin{aligned} (\text{PulseWidth} * 5) / 1,000,000 &= \text{TimeForOneRev\_S} * \\ &2 / \text{Cylinders} \\ \text{TimeForOneRev\_S} &= (\text{PulseWidth} * 5 * \\ &\text{Cylinders}) / 2,000,000 \\ \text{But Revs per second} &= 1 / \text{TimeForOneRev\_S} \\ &= 2,000,000 / (\text{PulseWidth} * 5 * \text{Cylinders}) \\ \text{So Revs Per Minute} &= 60 * \text{Revs Per Second} \\ &= (60 * 2,000,000) / (\text{PulseWidth} * 5 * \text{Cylinders}) \\ \mathbf{RPM} &= \mathbf{24,000,000 / (\text{Cylinders} * \text{PulseWidth})} \\ &\mathbf{\dots Eqn1} \end{aligned}$$

Because our Tachometer is aimed to be a general-purpose device, we need to leave Cylinders as a variable. The Cylinder value is actually determined by the position of DIP switches on the circuit board.

## Bringing some order

First, let's express the equation as:

$$\begin{aligned} \text{RpmW} &= (\text{Constant1W} * \text{Constant2W}) / \\ &(\text{CylindersB} \\ &* \text{PulseWidthW}) \dots \text{Eqn2} \end{aligned}$$

Where the end character 'W' or 'B' indicates a Word or Byte value.

The value of PulseWidthW represents the pulse-width between ignition pulses and can vary from 1 to 65,535. So the minimum RPM that can possibly be measured is 24,000,000 / (8 \* 65,535) = 45 RPM.

It makes sense to pre-calculate the expression CylindersB \* PulseWidthW and store the result in a Word variable before attempting the overall calculation. However this is only possible if CylindersB \* PulseWidthW doesn't exceed 65,535. This only happens for RPM values greater than 24,000,000 / 65,535 = 367.

After some experimentation, the best code turns out to be using separate calculation strategies for whether the RPM is greater or less than 367.

## The high revs (>=367 RPM)

$$\text{Assume: CylMultPulseWidthW} = \text{CylindersB} * \text{PulseWidthW}$$

The simplest calculation of the RPM to avoid overflow using PICAXE arithmetic is :

$$\begin{aligned} \text{RpmW} &= (\text{Constant1W} / \text{CylMultPulseWidthW}) \\ &* \text{Constant2W} \dots \text{Eqn3} \end{aligned}$$

In integer arithmetic it's always best to try to make the result of division operations as large a number as possible to preserve accuracy. Which means that Constant1W should be as large as possible. So we need to choose Constant1W to be as close to 65,535 as possible, but also ensure that Constant1W \* Constant2W is as close to 24,000,000 as possible. The best choice I've found through experimentation is:

$$\begin{aligned} \text{Constant1W} &= 65217 \\ \text{Constant2W} &= 368 \end{aligned}$$

This gives: Constant1W \* Constant2W = 23999856, which is 144 difference to 24,000,000 and so a tiny error. So now Eqn3 becomes...

$$\begin{aligned} \text{RpmW} &= 65217 / \text{CylMultPulseWidthW} * 368 \\ &\dots \text{Eqn4} \end{aligned}$$

But even choosing the right constants won't give a very good result when using Eqn4 directly in PICAXE code, because the integer division is done before the multiplication and so the rounding error that takes place in the division is scaled up in the final result.

But in PICAXE maths we've got the ability to obtain the remainder of a division, using the modulus operator. So Eqn4 can act as a good first approximation, to which we can add the result of a calculation involving the remainder.

If the division of 65217 by CylMultPulseWidthW gives a WholeW and RemainderW then expanding Eqn4 we have:

$$\begin{aligned} \text{RpmW} &= (65217 / \text{CylMultPulseWidthW}) * 368 \\ &= (\text{WholeW} + (\text{RemainderW} / \text{CylMultPulseWidthW})) * 368 \\ \text{RpmW} &= [\text{WholeW} * 368] + [(\text{RemainderW} / \text{CylMultPulseWidthW}) * 368] \dots \text{Eqn5} \\ \text{RpmW} &= \text{Expression1} + \text{Expression2} \end{aligned}$$

Expression1 is just the result of the simple integer division.

If Expression2 is performed in integer maths, it will always give a result of zero because the remainder is going to always be less than CylMultPulseWidthW ! So we need to manipulate the maths here.

A first observation is that the remainder value will never be greater than 32767 (ie half of 65,535), for whatever value CylMultPulseWidthW takes within the Word range. So to improve the division it will be helpful to pre-multiply the RemainderW by 2 before the division takes place...

$$\begin{aligned} \text{Expression2} &= [(2 * \text{RemainderW}) / \\ &\text{CylMultPulseWidthW}] * 184 \end{aligned}$$

Notice how the end number of 368 has been halved to 184 to compensate.

But this won't give a useable expression on it's own. A further trick is to scale-down the denominator of Expression2 and then divide the result of the division by the same factor:

$$\begin{aligned} \text{Expression2} &= [(2 * \text{RemainderW}) / \\ &(\text{CylMultPulseWidthW} / \text{Factor})] * (184 / \text{Factor}) \end{aligned}$$

The fact that this trick can greatly improve calculation accuracy isn't that intuitive to most people (or me at first!) but it does work! However choosing the best factor to use is a bit of trial and error. It makes sense to use a factor that divides into 184 perfectly, because this will reduce any further rounding errors.  $84 = 46 * 4$  and this seems to work well, giving...

$$\begin{aligned} \text{Expression2} &= [(2 * \text{RemainderW}) / \\ &(\text{CylMultPulseWidthW} / 46)] * 4 \dots \text{Eqn6} \end{aligned}$$

But this can further be improved by looking at a new remainder value for this division! I won't expand on this, but you'll get the idea from the code below.

Putting the whole RPM calculation into code gives:

$$\begin{aligned} \text{CylMultPulseWidthW} &= \text{CylindersB} * \\ &\text{PulseWidthW} \\ \text{RpmW} &= 65217 / \text{CylMultPulseWidthW} * 368 \\ \text{RemainderW} &= 65217 // \text{CylMultPulseWidthW} \\ \text{TempW} &= \text{CylMultPulseWidthW} / 46 \\ \text{RpmW} &= 2 * \text{RemainderW} / \text{TempW} * 4 + \\ &\text{RpmW} \\ \text{RemainderW} &= 2 * \text{RemainderW} // \text{TempW} \\ \text{RpmW} &= \text{RemainderW} * 4 / \text{TempW} + \text{RpmW} \end{aligned}$$

## The low revs (<367 RPM)

Here a different approach is taken, that is actually much simpler and very accurate. This would be ideal to use for all RPM values, but unfortunately it isn't accurate for the higher RPM calculations.

In the discussion above we've used :

$$\begin{aligned} \text{CylMultPulseWidthW} &= \text{CylindersB} * \\ &\text{PulseWidthW} \end{aligned}$$

However for low RPMs this calculation will overflow. So it's necessary to use the \*\* PICAXE operand to work out what this overflow is, and express the CylMultPulseWidth value in terms of a Most-Significant and Least-Significant Word.

$$\begin{aligned} \text{MSW} &= \text{CylindersB} ** \text{PulseWidthW} \\ \text{LSW} &= \text{CylindersB} * \text{PulseWidthW} \end{aligned}$$

Using Eqn1 we have:

$$\text{RpmW} = 24,000,000 / [(\text{MSW} * 65536) + \text{LSW}]$$

And using our previously determined constants ...

$$\begin{aligned} \text{RpmW} &= (65217 * 368) / [(\text{MSW} * 65536) + \text{LSW}] \\ &= 65217 / [[\text{MSW} * (65536/368)] + \\ &(\text{LSW}/368)] \\ &= 65217 / [(\text{MSW} * 178) + (\text{LSW}/368)] \end{aligned}$$

The denominator of this division can be worked out easily. The numerator constant has been kept at the largest value possible to preserve accuracy.

Putting this into code gives:

```
MSW = CylindersB ** PulseWidthW
LSW = CylindersB * PulseWidthW
TempW = LSW/368
TempW = MSW * 178 + TempW
RPM = 65217/ TempW
```

## The results

The results are very satisfying:

*RPMMathsResults\_LowRPM.xls* analyses the low RPM calculations and shows that the maximum absolute error is 1 RPM (max percentage error 7.8%).

*RPMMathsResults\_HighRPM.xls* analyses the high RPM calculations and shows that the average absolute error is 1 RPM, the maximum absolute error is 4 RPM and the max percentage error is 0.8%.

These results seem very good considering we are using integer maths to do the calculation of Equation1.

This probably seems like one heck of a lot of work for such a simple Equation1, but with the constraints of integer maths there doesn't appear to be a simpler solution to achieve the same accuracy.

The final code isn't really that long, which is the main thing, even though it's been a slog to work it out!

❖