

Making Waves

By John Dobson

Introduction

Many of you may be familiar with using the A/D inputs of the PICmicro microcontroller but using D/A's is less common. This short article examines how it is possible to use Flowcode and E-blocks to produce a simple sine wave oscillator for use in testing audio circuits.

Background theory

I have a requirement for a simple sine wave oscillator which I can use for testing audio circuits. The E-blocks system includes a D/A converter so I thought I would try to make a general purpose waveform generator with it.

To start with I needed to decide on the range of values I wanted to produce and do some calculations on the overall parameters to check the project is feasible. The D/A has an output range of 0 to 5V in 255 equal steps. It does not make sense to go too near the rails in case some headroom is needed for a buffer device, so I decided to keep things simple and go for a maximum of 4V peak to peak output. I can divide this down in software to get smaller signals. I also wanted a frequency of around 1kHz – mid range audio. A quick check of the specification of the MAX5385 D/A states that it has settling time of 20uS for output accuracy of half of the least significant bit. If we have 256 samples per waveform then this would give us a maximum output frequency of 200Hz. In theory that's not high enough, but as we are using a sine wave, with small incremental steps between samples, I reckon it has a fighting chance of coping and I can always resort to reducing the number of samples per waveform if it does not perform well.

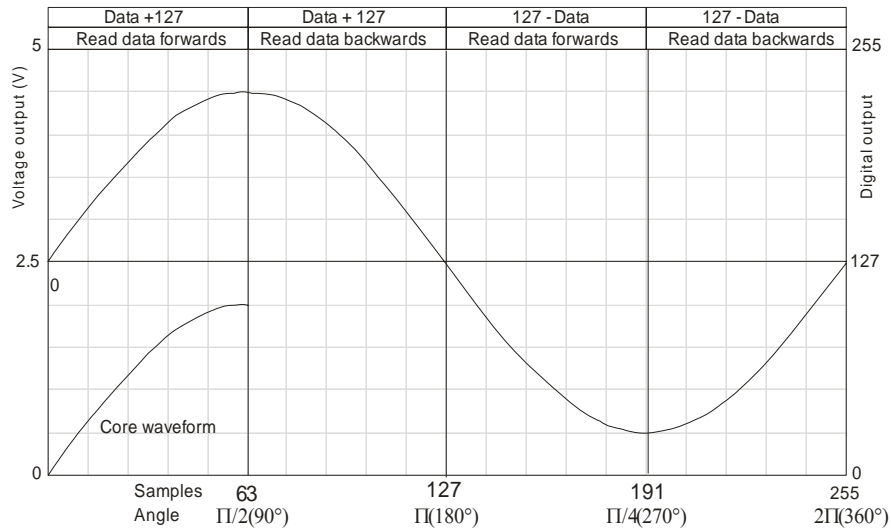


Fig 1 – initial specification sketch

D/As are relatively simple to use – you feed them a number and the analogue output corresponds to the full scale voltage output multiplied by ratio of the number you feed in and to the full scale digital input. From the graph in Fig 1 you can see my initial sketch

assuming 5V full scale output and 8 digital bits – 0 to 255 – input. (Actually the output of the DAC is only 0.9 Vcc so there will be a 10% voltage level error in the real output, but we can live with that.)

So I needed to generate a range of values corresponding to a sine wave output. With 256 steps that means 256 separate values. Fortunately sine waves are symmetrical I can actually just generate numbers for a quarter of a waveform and then use simple programming to reflect the data horizontally and vertically. We want a 4V peak to peak output: to make the numbers easy I decided to actually just use a range of 200 of the available 255 bits. This will give us close to 4V out of the full 5V range. The core waveform is shown in the bottom left of Fig 1: 64 values in x with a digital range of 0 to 100 or around 2V.

At this stage we need to use a spreadsheet to calculate the numerical output values for the core data:

Sample no	x	sin(x)	100sin(x)
0	0.024544	0.024541	2
1	0.049087	0.049068	5
2	0.073631	0.073565	7
...
61	1.521711	0.998796	100
62	1.546255	0.999699	100
63	1.570798	1	100

Table 1 – output values

The formula for a sine wave is $y = \sin(x)$. x varies from 0 to 360 degrees or 0 to 2π radians. I could not figure out how to get Excel to work in degrees so I just used radians. I have only shown the first 3 and last 3 of my 64 core values. To get the values we need to send to the DAC we just multiply $\sin(x)$ by our core range of 100.

Ok so that's the theory over – lets see it all put into practice.

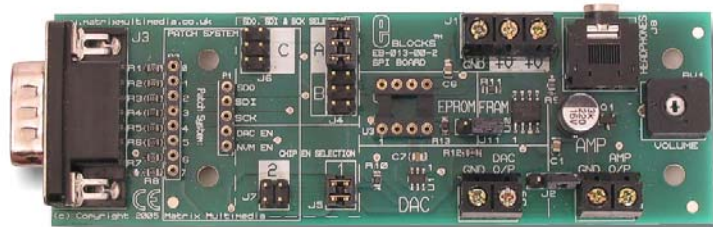


Fig 2 – SPI bus board containing NVM and DAC

In practice

First lets look at the hardware: I am going to be using the E-blocks SPI bus board shown in Gig 3. This goes onto my Multiprogrammer boards which happens to have a

PIC16F877 on it. You could use almost any PIC with a USART. The SPI board goes on port C and I have an LCD display on port B and a keypad on port D. The SPI board has a MAX5385 DAC as well as a 64Kbyte SPI bus Non Volatile Memory (NVM). Firstly I wanted to get the data into the NVM. As a medium term plan I intend to build up a portfolio of different waveforms inside the NVM and then call them up separately as needed using the keypad and LCD for selection. To populate the NVM I made program that stuffed the core 64 data values into the NVM: I simply established a counter from 0 – 63 and then wrote individual bytes of data to sequential NVM locations. In Flowcode this is quite easy – you just initialise the SPI bus and then write a program with 64 icons each of which writes a byte of data, as shown in fig 3.

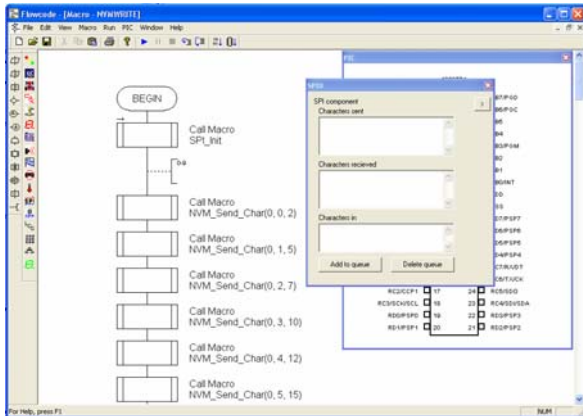


Fig 3 – NVM stuffing program

The next step was to write a separate program to check the core values produce the first quarter of the sine wave. This was fairly straight forward – I established a counter, read the value from NVM locations 0 to 63 in turn, and sent the data from the NVM to the DAC.

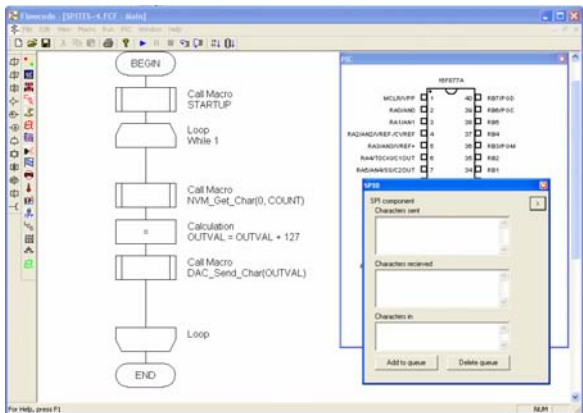


Fig 4 – Simple program to take NVM data and send it to the DAC

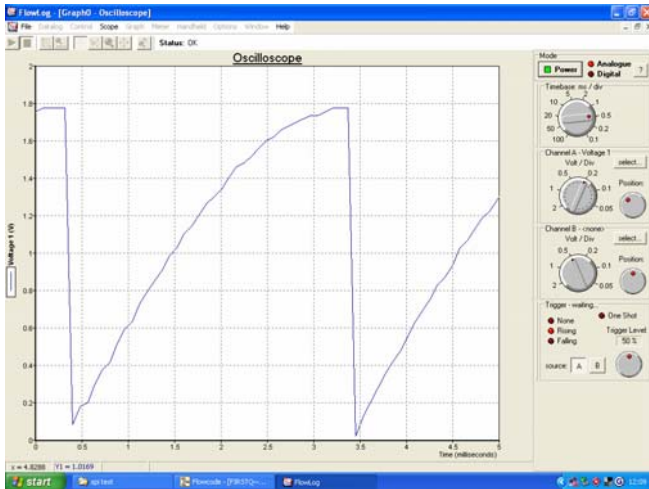


Fig 5 – first output waveform showing repeated core values output.

The oscillogram in Fig 5 shows you the first output waveform for the first quarter cycle. It's a bit rough around the edges but looks like a quarter of a sine wave. Unfortunately the time period for a quarter cycle was 3ms – giving a final frequency of under 100Hz: too slow for my purposes. Oh well – I have a backup plan.

Things are slowed up by the need to serially read the data from the external NVM. However the '877 device I am using has 256 internal bytes of EEPROM: what we can easily do is populate the internal EEPROM with our data which will speed up the process a great deal.

I added a new user macro to Flowcode, called LOADEEPROM, that read the external NVM and puts the 64 data bytes into internal EEPROM. Next I altered the program to read data forwards for the first quarter, and backwards for the second quarter of the waveform, and added the offset of 127. Similarly I manipulated the data for the second half of the cycle and then measured the results which you can see in Fig 6.

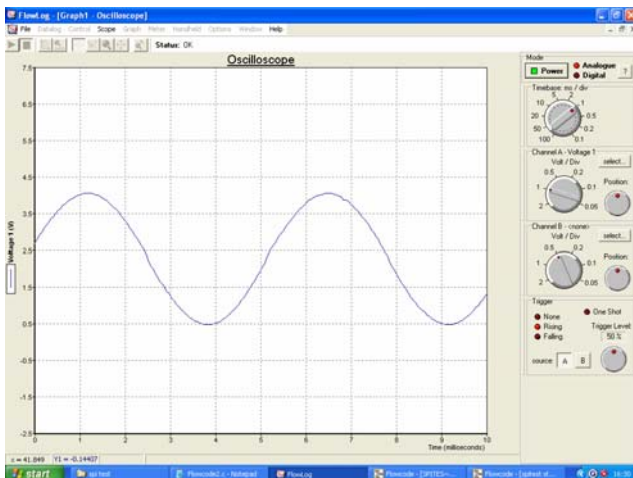


Fig 6 – the final output.

The final signal has a period of 2.5ms or a frequency of 400Hz, and a peak to peak voltage of around 3.5V. The waveform does not look too distorted although at the zero crossing points you can see a small discontinuity. This is presumably caused by program delays between each quarter cycle of data being read out. I plan to solve this by loading the internal EEPROM with the full 256 databytes and start the waveform at 90 degrees where distortion will have less effect. However it is now Christmas Eve and if I don't stop now then my family tell me I will be spending Christmas alone.....

Programs available for download:

SPI memory stuffing.fcf

First quarter.fcf

Sine wave gen.fcf

Copyright © John Dobson 2006