PIC n' Mix

Mike Hibbett

Our periodic column for PIC programming enlightenment

Practical DSP – Part 1

HELLO again! While Mike O'Keeffe takes a break from writing to dive into the joys of parenthood, I am back to fill his boots for a short time. This provides an opportunity for a clean break from Mike's current subjects, so we thought it would be a great time to look into a topic that we have not yet covered in *PIC n' Mix* – namely, 'digital signal processing' – or 'DSP' for short. It's a fascinating area with some fairly complex mathematics behind it, but we will focus on practical applications, and have some fun along the way. First though, here's a bit of background.

What is digital signal processing?

DSP is the process of taking a signal, converting it into a digital form, and performing signal processing or analysis on that digital representation of the original signal. The types of input signals that are suitable for use with DSP algorithms various enormously – audio signals from a microphone, video from a camera, vibration signals from a sensor on a rotating machine shaft are just some examples. You can even apply DSP to slowly changing signals such as daylight levels, or changes in heart rate (very relevant in neonatal care.)

Real-world applications run from the tuning of church bells during manufacture to the detection of planets orbiting distant stars. The latter example has led to the detection of planets outside of our solar system.

There are a number of signal processing techniques that digital signal processing encompasses, but the technique which has fascinated the author since university days is the transformation of a signal from the 'time domain' to the 'frequency domain'.

So, what do these terms actually mean? A signal whose value (amplitude for example) is recorded over a period, a *time* interval, is being represented in the *time domain*. A signal whose value is recorded over a range of *frequencies* is being represented in the *frequency domain*. You can see an example of these two representations of the same signal in Fig.1.

The example in Fig.1 is very simple, but it clearly demonstrates how to visualise the translation of a signal in the time domain into the frequency domain. Consider, however, a situation where your input signal is a mix of many different frequencies. Being able to transform such a signal, in real time, into its component frequencies enables some very interesting processing capabilities. Examples include looking for signals at particular frequencies (akin to bandpass filtering) or even simply visualising the frequency components of a signal – such as for an audio spectrum display on a Hi-Fi amplifier.

The process of transforming a signal from the time domain to the frequency domain is called 'Fourier transformation'. The 18th century French scientist and mathematician Joseph Fourier invented this mathematical system, long before its practical applications were realised.

The computational method for performing the transformation is called the 'discrete Fourier transform' (DFT), which deals with translating a finite set of input signal levels (ie, a short sample period) into a corresponding set of frequency components. The actual algorithm that is used within digital systems is called the 'fast Fourier transform' (FFT.) This is an algorithm that yields the same results as a DFT, but is highly optimised. Surprisingly, the DFT algorithm was also discovered long before any practical applications

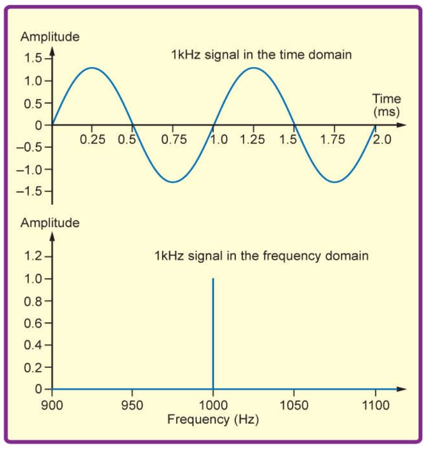

Fig.1. Two different ways of displaying the same signal – a 1kHz sinewave graphed in the time and the frequency domains

were available, and certainly well before the advent of computers.

In this series of articles we will be exploring the application of the DFT to analogue signals. Many of the applications of digital signal processing require expensive digital hardware and sensing systems, but there are many practical applications that can fall within a hobbyist's price range, while still being useful, educational and fun.

The consequence of limiting the cost of the system we create will be resolution, signal frequency and accuracy. There are however many applications where those specific limitations are perfectly acceptable.

Fast!

The mathematics behind FFT is complex, and beyond the scope of these articles. However, for our purposes all we need to know is some of the background and principles of application. Fortunately, the engineers at Microchip have created highly efficient DSP software libraries that we can access and build into our projects, and Microchip provide these libraries for free. If this series of articles whets your appetite to dig deeper into the theory there are a number of very good university tutorial videos available for free, on-line. An excellent introductory overview is given here: **https://youtu.be/spUNpyF58BY**. (Incidentally, the same YouTube author also offers some excellent introductions to calculus, neural networks, machine learning blockchain technology and other important and topical mathematical subjects – thoroughly recommended!)

The principle of operation is straightforward enough: take a series of samples of your data at regular time intervals, and store them into a buffer. Adjust those samples to match the format required by your FFT function and then call the FFT function to transform the data from the time domain to the frequency domain. Last, examine the resulting data and take an action on it. This workflow is shown in Fig.2, highlighting what operations occur within the hardware circuit, and what occurs on-chip.

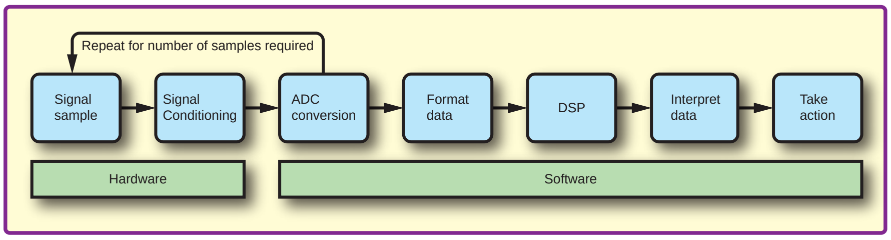

Fig.2. Overview of a DSP workflow (hardware and software)

The signal conditioning stage is a combination of scaling your signal to maximise the full range of the ADC input voltage swing, and also some low-pass filtering to minimise any high frequency signals or noise, above our sampling frequency, which many cause unexpected 'ghost' signals to appear in your data. This process is called 'anti-alias filtering'. We will explore the effects of aliasing next month.

Processor selection

Let's now move away from the theory and look at how we can start experimenting with a practical solution. Microchip offers a number of processors with DSP-assisting instructions built into the chip, but it basically comes down to a choice between three processor families: the PIC32MZ, dsPIC30 or dsPIC33. The PIC32MZ has the highest clock speed and largest memory availability, but we have selected the dsPIC33 family as this range offers devices with dual-in-line (DIL) packaging – ideal for simple breadboard tests. Specifically, we have selected the dsPIC33EP512GP502-I/SP, a 28-pin DIL device, which is cheap, readily available from distributors such as Farnell (part code 2406557) and has plenty of on-chip resources. It is also compatible with the PICKIT-3 debugger/programmer, our favourite low-cost interface. Here is a summary of what you get from this processor, all for the price of a pint of beer:

- 512KB Flash memory
- 48KB RAM
- 500,000 samples per second 12-bit ADC
- CAN bus interface
- 21 GPIO pins
- DSP-instructions
- On chip high-speed oscillator
- 70MHz clock speed
- 3.0V to 3.6V operation

Plus, all the usual timers, PWM, SPI, I²C interfaces that you would expect from Microchip. The pin-out for this device is shown in Fig.3.

With the on-chip high speed oscillator, calibrated to within 1%, there is no need to add an external crystal oscillator. This is ideal, as it allows us to build the test circuit on a breadboard, with just a few decoupling capacitors and pull-up resistor required to complete the minimal hardware.

Software

Many microcontroller vendors provide DSP algorithms within their free software libraries, even for processors that do not have special DSP instructions in hardware. Microchip provide two sources of DSP libraries: one within the Harmony software framework, designed specifically for the PIC32MZ family of processors, and second, a separate dsPIC DSP library, provided as part of the XC16 compiler, for the dsPIC family of processors. Since we have selected the dsPIC33 processor we will make use of the latter. Do be careful not to confuse the two should you search on-line.

We will build our software within the MPLAB-X IDE, using the free version of the XC16 compiler. With the addition of the PICKIT-3 debugger/programmer unit, this represents a very low cost set-up. We will cover the XC16 compiler installation next month.

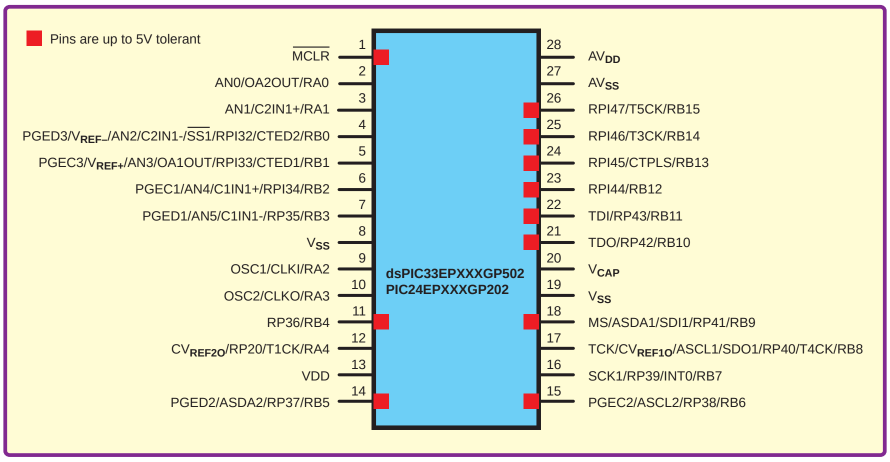

Fig.3. Pin out for our digital signal processing PIC of choice – the inexpensive 28-pin dsPIC33EP512GP502-I/SP, a DIL device.

Hardware design

Fig.4 shows the minimal components required to start testing the DSP capabilities of the dsPIC33 processor. Three 100nF capacitors, a 10µF tantalum capacitor and a 4.7kΩ resistor complete the basic requirements. Two LEDs have been added here to aid debugging; half a dozen hook-up wires need to be added to make this breadboard good to go and ready to connect to the PICKIT-3 debugger/programmer interface.

If you decide to follow along with us on this journey and are using a similar breadboard setup, make sure your breadboard is a good one, and not something you have had in your shed for the last ten years. While all the high frequency signals are on-chip, your ability to program the device reliably may be impacted by old, tarnished connections. Fault finding these issues can be frustrating!

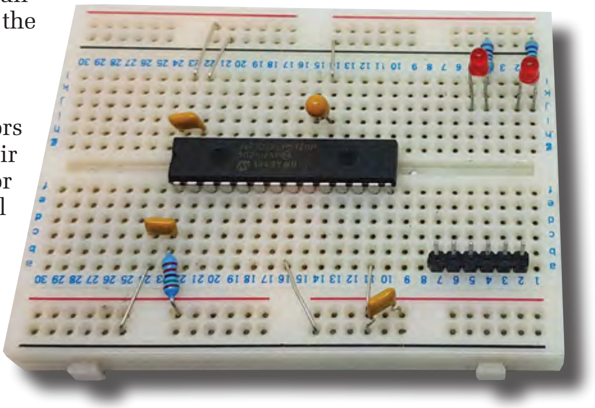

Fig.4. Our basic DSP hardware – details next month

The PICKIT-3 also serves as a power supply for the processor, freeing us from the need to provide an additional regulator circuit to the board. At a push, however, the circuit will run directly from a pair of AA cell batteries because the processor can operate over the 3.0V to 3.6V range.

As you can see from this board layout, initially, we are not connecting sensors. Our first software tests will be made with pre-computed data samples, allowing us to know exactly what data is going into the FFT function. This minimises the number of things that can go wrong. We will add sensor input via the ADC once the basics are working.

Our aim in this series of articles is to leave you with a reference hardware design and source code that will enable you to explore this fascinating branch of electronics to your own ends. We hope to hear from you on your progress in DSP!

Next month

In Part 2 we will discuss building the initial hardware and look at how to incorporate the DSP libraries provided by Microchip.