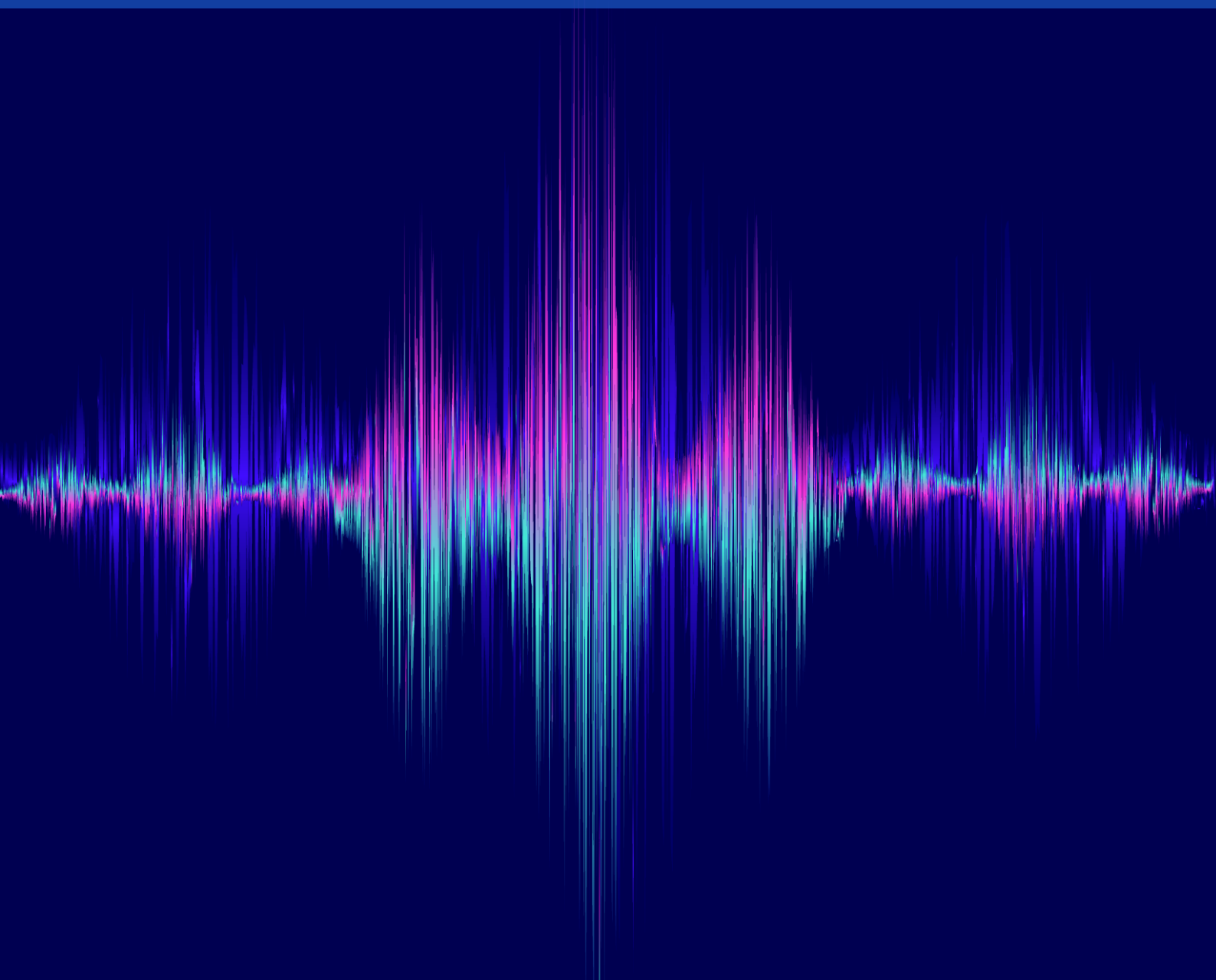
8 DSP Fundamentals Every Electronics Engineer Should Know



John Edwards & Dunstan Power

8 DSP Fundamentals Every Electronics Engineer Should Know

Integrating Digital Signal Processing Algorithms into Your Future Application

John Edwards
Sigma Numerix Ltd

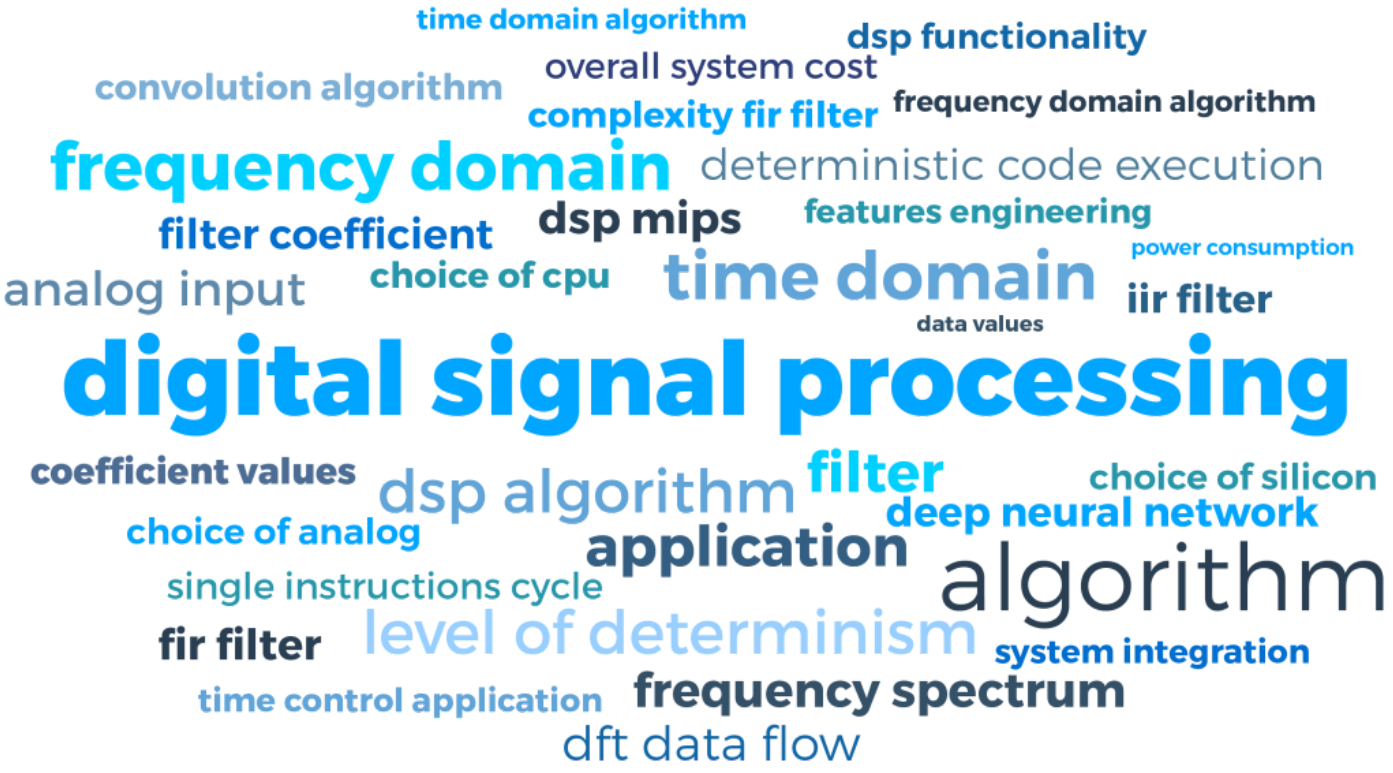
Dunstan Power
ByteSnap Design Ltd

8 DSP Fundamentals Every Electronics Engineer Should Know

Table Of Contents

8 DSP Fundamentals Every Electronics Engineer Should Know

Introduction



Digital Signal Processing (DSP) has been a key part of many real time applications for several decades but over the last few years the use of hybrid devices, which are sometimes called bridge devices, is becoming increasingly common.

It is no longer necessary to use a dedicated DSP device for these applications so the choice of architecture to use has become much more complex.

A good example of where these new devices are being utilized is the enhancement of signals for later processing by algorithms such as Deep Neural Networks (DNNs).

In applications from voice recognition to vibration analysis, Feature Extraction, or Feature Engineering as it is increasingly being called, is an application of DSP algorithms that needs to connect to a wider network for the cloud based back-end processing.

Feature Engineering is the art of pre-processing signals before passing them to the training engine or the classifier. Extracting the best features for an ML model to classify is an art and can make the difference between success and failure.

Feature Engineering includes processes such as event counting, period or statistics; filtering; statistics analysis such as mean and variance; peak or zero crossing detection; frequency spectrum or histogram estimation.

Implementing real-time signal processing algorithms requires the design engineer to traverse two potential minefields; choice of algorithm and choice of hardware. The two are inextricably linked - the wrong choice of either will often end up with a sub-optimal decision regarding the other.

Historically, DSP algorithms have been run on specialized DSP devices that are optimized and targeted towards these applications. In more modern times, DSP functionality has become far more commonly integrated into general purpose microprocessors and microcontrollers so the term “DSP device” refers to any processor executing signal processing algorithms.

So, for integrating digital signal processing algorithms into your future application, here are **the top 8 DSP fundamentals every electronics engineer should know**:

8 DSP Fundamentals Every Electronics Engineer Should Know

Let's go through them now in greater detail...

1. Time Domain Algorithms

Signal processing algorithms can be split into two main categories for the majority of applications - **Time Domain** and **Frequency Domain**.

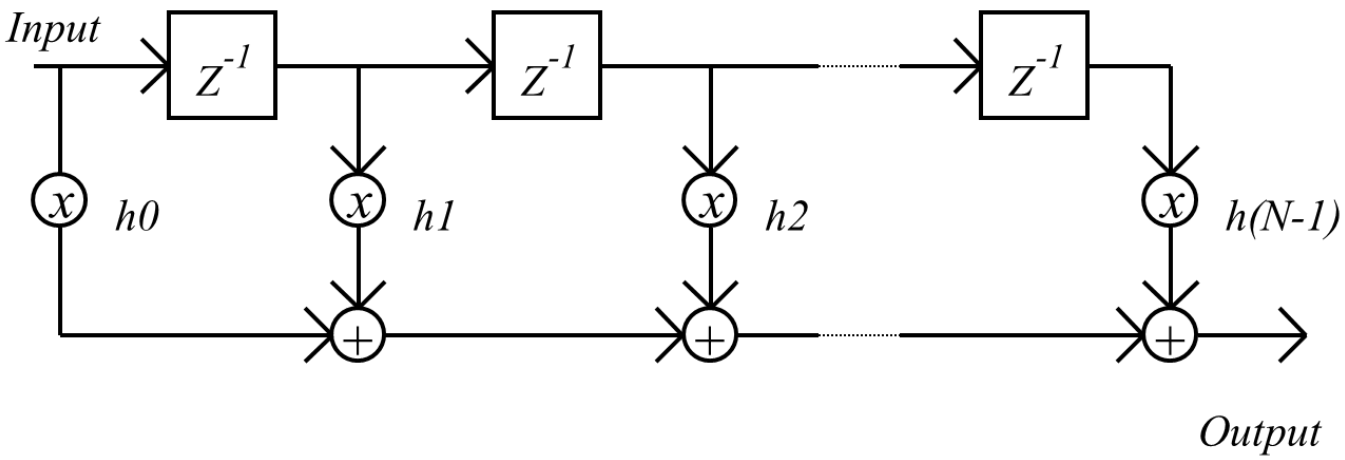
The most common time domain algorithms are filters, of which there are two basic types. The most basic filter uses a feedforward path for all of the data processing and these filters are often referred to as Finite Infinite Response (FIR) filters, also known as convolution filters.

The more advanced type of filter is the Infinite Impulse Response (IIR) filter, which incorporates feedback.

This equation shows the FIR filter or convolution algorithm:

$$y(n) = \sum_{}^{M-1} h(k).x(n-k)$$

Which is implemented using the following flow diagram:



The FIR filter or convolution algorithm is the combination of 2 mathematical operators – the multiplication of the data values by the filter coefficients and then the accumulation of the products to get the final result.

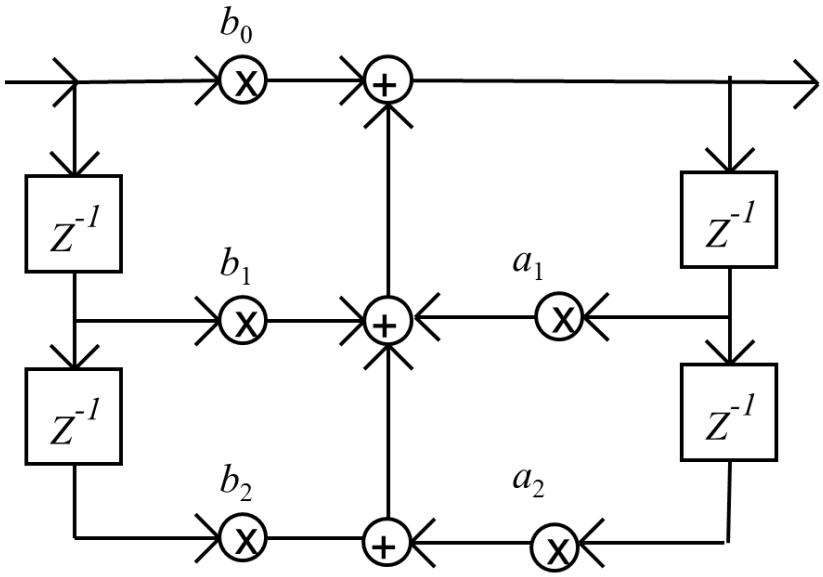
This Multiply-Accumulate (MAC) operation is performed over all the filter coefficients for every data sample so it requires a lot of operations for large filters.

Despite their high operational complexity, FIR filters are - by their nature - easy and safe to use because they are unconditionally stable.

IIR filters on the other hand include feedback elements, as this equation shows:

$$y(n) = \sum_{k=1}^{N} a_k . y(n-k) + \sum_{r=0}^{M} b_r . x(n-r)$$

Here is the corresponding the flow diagram:



IIR filters are, therefore, an extension of the same Multiply-Accumulate architecture used in FIR filters. The inclusion of the feedback path makes these filters much more efficient than the FIR filter. However, it also leads to possible instability and hence they require much more testing.

FIR filters can have a linear phase response, when the impulse response (coefficients) is symmetrical, and this is a useful property for some applications. Most applications, though, do not require linear phase so the enhanced performance, lower memory requirements and higher efficiency of IIR filters means that they are the most commonly used. All modern digital filter design packages can support both FIR and IIR filter design.

2. Frequency Domain Algorithms

Real-world signals are sampled in the time domain, using an Analog To Digital Converter, but the time domain is not necessarily the most efficient domain for processing the data.

Most engineers are familiar with the frequency domain being used for spectral analysis; but we should also consider that our time domain data stream can be converted to the frequency domain for processing as well as analysis.

The Discrete Fourier Transform (DFT) - typically implemented using the Fast Fourier Transform (FFT) algorithm - is the most common method for performing the transform from the time domain to the frequency domain.

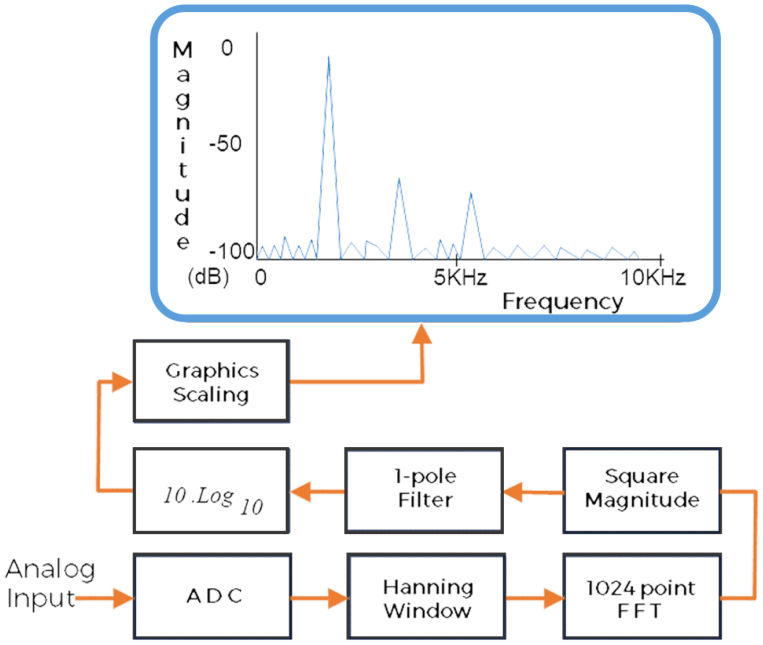
The DFT algorithm is shown in the following equation:

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{-j\frac{2\pi nk}{N}} \qquad for\ 0 \leq k \leq N-1$$

While we are not going to dive into the details of this algorithm in this book, it is useful to consider it from a high level.

If you look at this algorithm, you will see how similar it is to the convolution algorithm, that we featured earlier, and this observation is true because the DFT is essentially just a bank of digital filters.

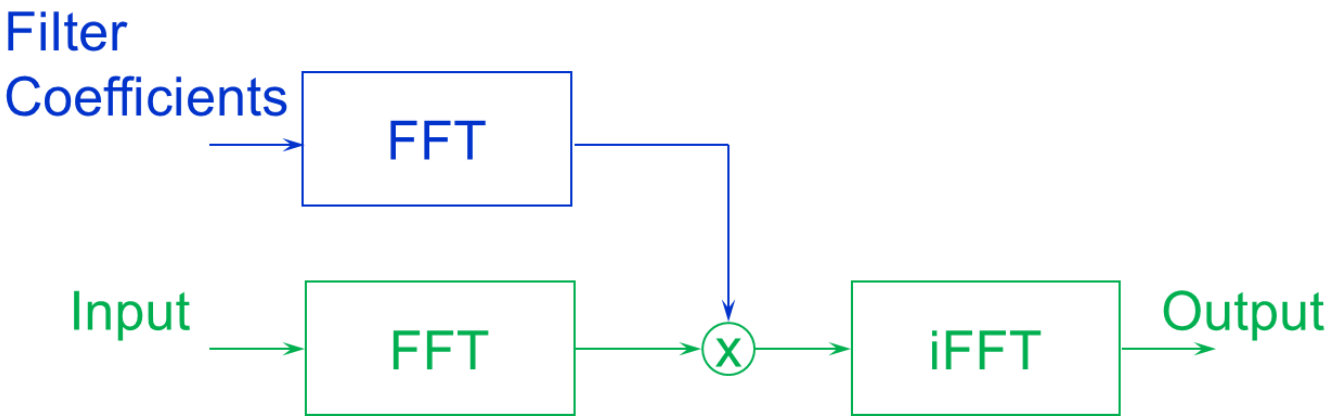
The FFT, being an optimization of the DFT data flow for minimizing the MIPS requirements, is typically used to convert from the time domain to the frequency domain for analysis - as can be seen here:



But the frequency domain can also be used for processing, using a property that convolution in the time domain is the equivalent to multiplication in the frequency domain.

As we have seen previously, convolution is a Multiply-Accumulate operation so it is much more MIPS intensive than simply cross multiplying two arrays in the frequency domain.

This diagram illustrates Frequency domain filtering:



The Fourier Transform of a system impulse response (i.e. the coefficients of an FIR filter) is the transfer function of that filter. This can be pre-calculated at development time to improve performance, as highlighted in blue.

The multiplication operation used in the frequency domain process is much more efficient than the convolution process used in the time domain FIR filter.

Therefore, the primary benefit of using the frequency domain for processing is reduced MIPs, but this is at a cost of memory and latency which are due to the use of the forward and inverse FFTs in the pipeline.

So, the choice of whether or not to perform the DSP functions in the time or frequency domain comes down to a trade-off between MIPS and memory, which can clearly have an impact on the choice of CPU architecture, as we will see in the next section...

3. Hardware Selection

Choosing the correct hardware to implement our signal processing algorithms can be a complex choice. It is quite common, however, for the final decision to be driven by what may at first appear to be secondary issues, such as I/O capabilities or memory capacity.

As we have seen in the first part of this book, the most important capability for a CPU being used to implement DSP algorithms is the ability to efficiently perform the Multiply-Accumulate operation. Historically, DSP algorithms have been run on specialized DSP devices that are optimized and targeted towards these applications.

In more modern times, DSP functionality has become far more commonly integrated into general purpose microprocessors and microcontrollers. So, in some ways, the choice of silicon has become easier because there is a wider selection of devices that can be used. This breadth of choice can also make the decision harder.

For the rest of this book we will refer to the device executing the algorithms as the **DSP Device.** However, this could just as easily be an embedded microcontroller that is performing Feature Engineering on vibration data, that is being analysed by a Deep Neural Network in order to monitor the status of a machine.

The first stage of selecting our target device is to choose our algorithms and see which devices can execute the code fast enough to meet the sample rate timing requirements. These devices must, by definition, include the required DSP MIPS and memory capabilities for the application, but also be able to provide a level of determinism to our processing pipeline.

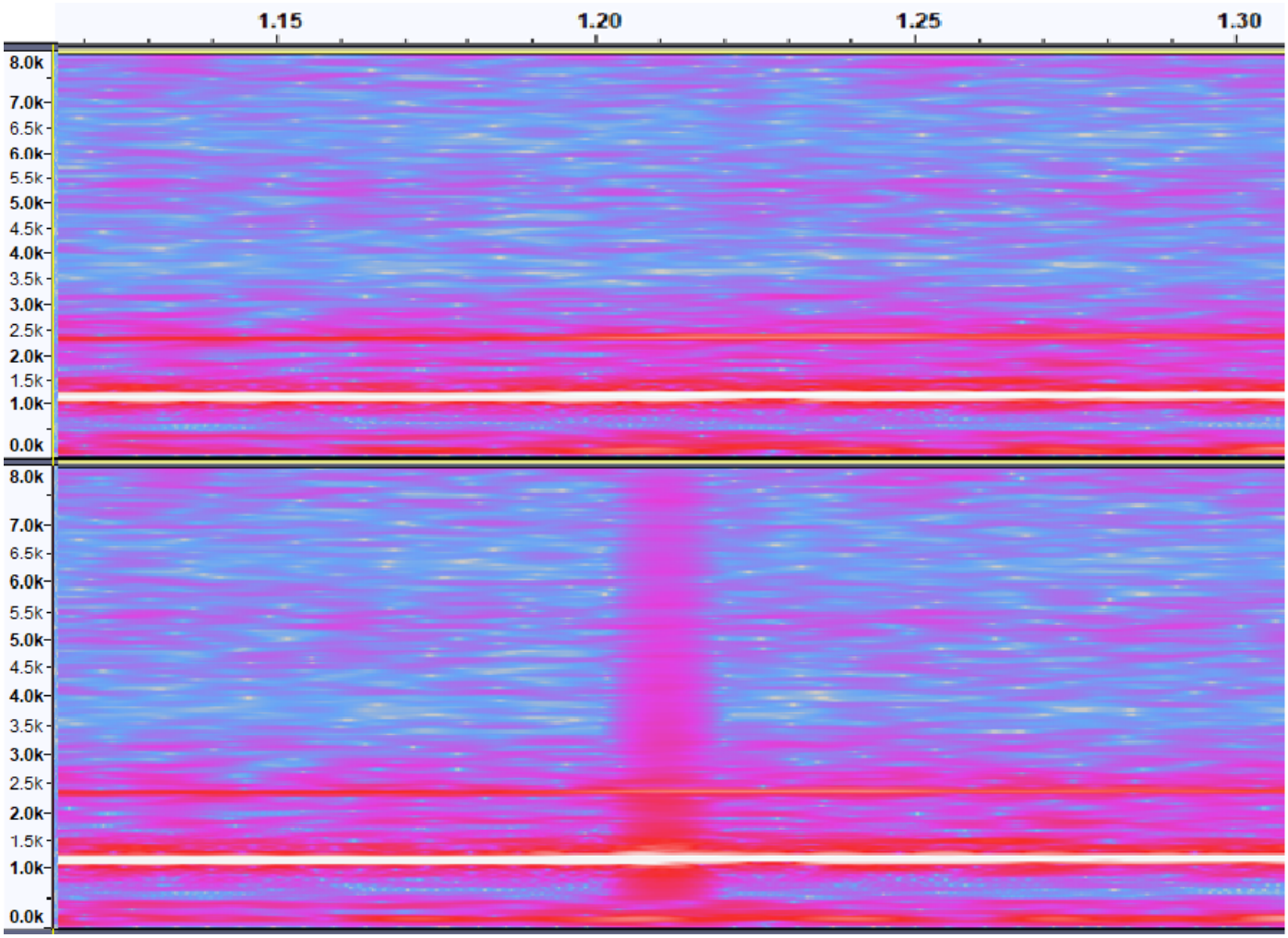
4. Deterministic Code Execution

Deterministic performance of the DSP is one of the most important things for a real-time system, because missing a sample interrupt can cause dramatic changes to the results.

At the simplest level, if a noise cancelling algorithm misses an input sample then it will need to re-train, which may just be an annoyance for the user.

At the other end of the spectrum, dropped samples in an industrial control system can cause equipment damage or worse.

In the image on the next page, you can see the frequency spectrum of two identical audio streams sampled via an ADC.



The top stream is perfectly sampled – however, the lower stream is dropping individual samples every few seconds.

This can be clearly seen at just after the 1.2 second mark where there is a very large spectral smearing.

If this signal was being provided to a DNN for vibration monitoring then this could easily cause False Accepts in the final results.

5. DSP MIPS

As we saw in the first part of this book that the filter and Fourier transform functions, we are going to implement our highly optimized algorithms that all have the same basic form:

$$y(n) = \sum_{}^{M-1} h(k).x(n-k)$$

DSP MIPs (Millions of Instructions per Second) are very different to MIPS measurements for regular CPUs or Microcontrollers. That's because DSP MIPS refer to the performance of a processor to execute the Multiply-Accumulate operation.

As mentioned previously, the heart of the convolution algorithm is the Multiply-Accumulate (MAC) algorithm where we are multiplying the data values (***x(n - k)***) by the coefficient values (***h(k)***) and summing the results into an accumulator register before saving the output.

So, there are 4 basic stages to this process:

1. **Load the data value(s)**
2. **Load the coefficient value(s)**
3. **Multiply the data value(s) by the coefficient value(s)**
4. **Add the results of the Multiply into the Accumulator**

Many modern DSPs incorporate a MAC instruction to perform all of these operations in a single instruction cycle or they can be initiated on every instruction cycle using pipelined CPU architectures.

Note: we are specifically highlighting value(s) in the above four stage process because many modern DSPs can process multiple values in parallel Single Instruction Multiple Data (SIMD) operations. This will have an important impact on the performance of the processor.

6. Memory Architecture

When considering memory issue for DSP applications, most engineers are familiar with laptops and workstations that have a flat memory space.

DSP and embedded microcontrollers often have a hierarchical memory architecture that will have different layers of internal (on-chip) or external (typically SDRAM or SBSRAM) memory.

Each of these memory types provide a different trade-off between size and speed (memory access bandwidth and latency).

Comparing different DSP memory architectures with respect to simple criteria such as bandwidth and latency is quite straightforward, but this comparison can become more difficult when issues such as power consumption are considered.

A common trade-off, for example, is between choosing a DSP with larger internal memory versus a device with smaller internal memory but which requires an external memory device.

Analysing which solution leads to a lower overall system cost and power consumption is an important part of the DSP solution choice.

7. Analog Input/Output

Analog I/O connectivity is one capability that often drives DSP device choice.

For many applications, such as audio and speech, high quality ADCs and DACs are relatively cheap and connect to the DSP using the (Inter-IC Sound (I^2S) bus. In these applications, choosing a DSP that natively incorporates I^2S connectivity is a logical solution.

ADCs and DACs designed for audio and voice applications often use Sigma-Delta converter technology, which means they may not be suitable for all applications. For example, Sigma-Delta converters designed for consumer applications typically exhibit the following:

Conversion Latency – This is often in the multi-millisecond range so this does not make them suitable for real-time control applications.

D.C. Offset – High D.C. offsets often preclude these devices from use in many medical and instrumentation applications.

Making the correct choice of analog I/O technology can make the difference between success and failure.

8. System Integration

The eighth DSP fundamental, system integration, is a very broad topic and includes items such as how will the DSP module integrate with elements, for instance:

- Networking
- Security Module
- Human Machine Interface (HMI)

As we have already seen, DSP architectures are optimized for processing the Multiply-Accumulate function. This often leads to pipelined architectures that are often not optimized for functions such as networking, security or HMI. DSPs are particularly inefficient and handling IP protocols such as TCP, UDP and higher layer protocols such as HTTP and MQTT.

As a consequence, one common solution is to choose a System on Chip (SoC) solution that integrates a general-purpose CPU or microcontroller core such as ARM Cortex M-Series or A-Series alongside the DSP core. These integrated devices can perform and optimum trade-off between DSP performance and system integration functionality.

Summary

Here, we have covered a variety of DSP algorithms and device architecture options, and examined how the choices we make will have a major impact on the cost of a project, as well as its success.

When starting a project that requires DSP functionality, understanding the algorithms required will lead onto choosing the most cost-effective hardware on which to implement those algorithms.

The cost metric for a DSP subsystem may not necessarily be the price of the CPU. Other factors - external memory, power supply or external I/O, for instance - may also lead to increased system cost. Therefore, giving consideration to the trade-offs available, for example between MIPS and memory, is essential - right from the start of the project.

Modern microcontrollers regularly include MAC instructions and other functionality to implement DSP algorithms efficiently, so you'll find these devices can often be an ideal fit for many embedded systems.

About John Edwards

Since obtaining his BEng (Hons) from the University of Bradford, in the 1980s, John has worked as a Signal Processing Applications Engineer for companies such as Loughborough Sound Images, Blue Wave Systems, Motorola, Picochip Designs and XMOS Semiconductors.

John now works as a Digital Signal Processing and Embedded Systems Consultant for Sigma Numerix Ltd.

He has worked with DSPs in a wide range of applications including wireless (2G, 3G WCDMA and 4G LTE), Voice Over IP, voice band and broadband modems, control, medical instrumentation and noise and vibration analysis.

John is a member of the IET, IEEE, and is a regular contributor of papers at international DSP conferences.

In addition, John has been a visiting lecturer at the University of Oxford since 1993 and teaches the annual Digital Signal Processing Course, as part of the Summer Engineering Program, for industry.

About Dunstan Power

Dunstan Power is a chartered electronics engineer providing design, production and support in electronics to ByteSnap Design's clients.

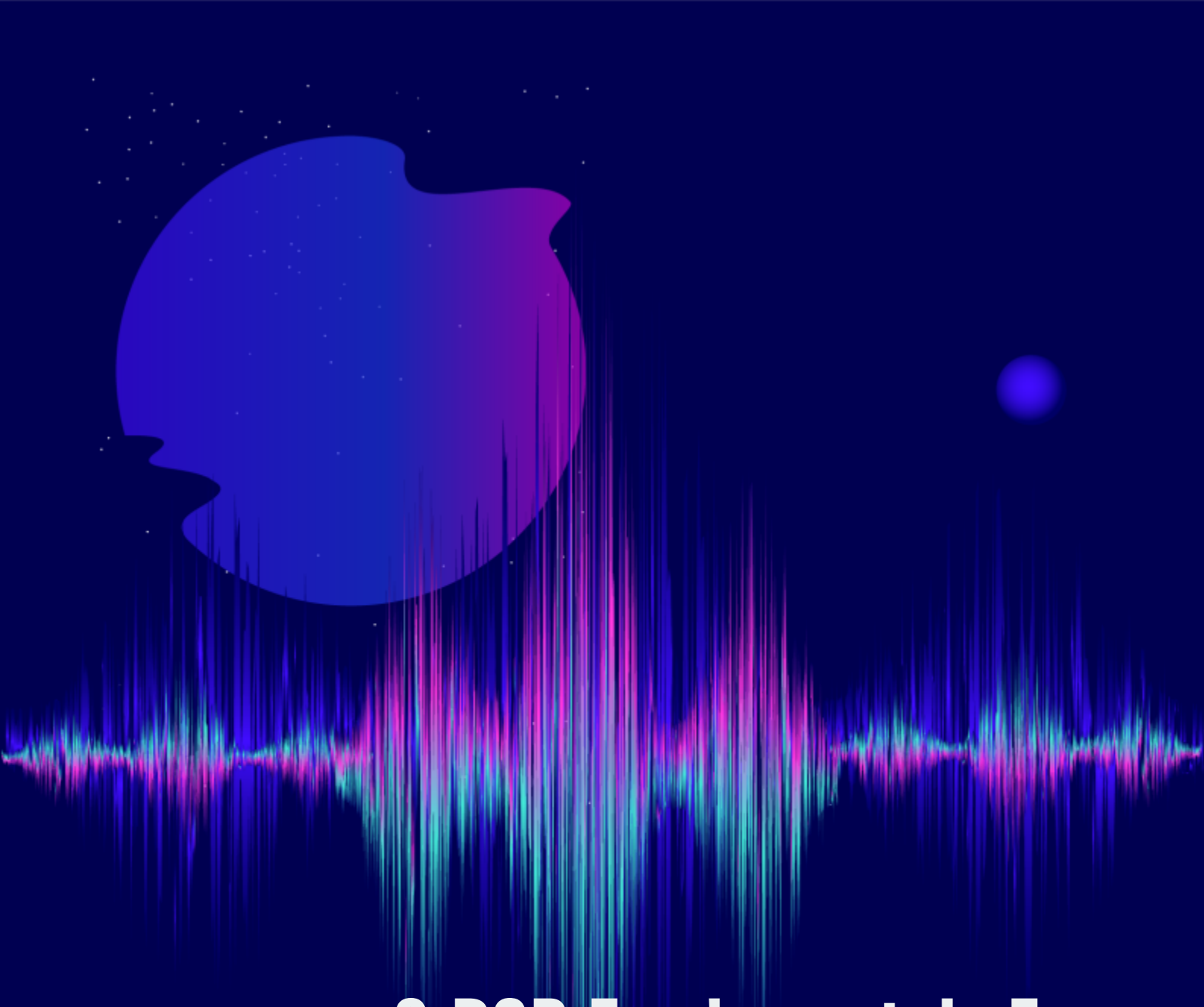
Having graduated with a degree in engineering from Cambridge University, Dunstan has been working in the electronics industry since 1992.

In 2008, Dunstan teamed up with his former colleague Graeme Wintle to establish the multi award-winning electronics engineering consultancy ByteSnap Design.

ISO 9001:2015 certified, ByteSnap's technical portfolio includes electronic circuit design, WinCE migration, Linux and embedded software development.

ByteSnap continues to successfully deliver multiple embedded systems projects, including Android BSPs and video processing applications, to clients around the world.

There's more to discover about ByteSnap Design at www.bytesnap.com.



8 DSP Fundamentals Every Electronics Engineer Should Know