

# PC BASED IEEE STD SYNTHESIS, ENHANCED THROUGH DEDICATED DEVICES

**Richard Hazlewood**  
**Racal Instruments Group Limited**  
**29-31 Cobham Road**  
**Wimborne. Dorset. BH21 7PF**  
**+44 (0)1202 872800**  
**richard.hazlewood@racalinstrumentsgr**  
**oup.co.uk**

**Matt Cornish**  
**Racal Instruments Group Limited**  
**29-31 Cobham Road**  
**Wimborne. Dorset. BH21 7PF**  
**+44 (0)1202 872800**  
**matt.cornish@racalinstrumentsgr**  
**oup.co.uk**

**Abstract – In the paper ‘PC Based, IEEE Signal & Test Definition Standard, Synthetic Instruments’, we introduced our successful implementation of a PC-based signal generation framework. This document describes our continuing research and development into this area, and the techniques we have developed.**

**We outline many of the factors that enforce limits on the system, with examples regarding the processing power of the PC and the output device used to generate the signals.**

**We discuss the use of external digital memory devices and integrated dedicated hardware and how we have established techniques that enable our framework to take advantage of such technology in order to overcome the limitations of the system.**

**Throughout the document we relate how our system adheres to signal definitions as defined in *P1641 of the IEEE Signal and Test Definition* standard.**

## INTRODUCTION

The previous paper titled ‘PC Based, IEEE Signal & Test Definition Standard, Synthetic Instruments’ [1] describes our successful implementation of a framework that uses signal definitions, as defined in *P1641 of the IEEE Signal and Test Definition* standard (P1641) [4], to construct Filter Graphs based on the DirectX® Streaming (DirectShow®) framework [2].

The framework is designed to stream the signal data through various DirectShow®-based filters that implement the behaviour, and perform the mathematical calculations, of the different Basic Signal Components (BSC) of P1641 [4]. With current technology we are able to construct software-based complex signals, in real-time, at frequencies over 2MHz.

Inevitably, as the processing power of PCs increases, our framework will be able to handle frequencies at much higher rates. However, we have established some hardware-assisted techniques that enable the framework to produce very high signal frequencies today; making use of instrumentation that has been available for some time.

## LIMITING FACTORS

The main limiting factors of our system at present are:

- The processing power of the PC.
- The rendering device used for the digital to analogue (D/A) conversion.

## Processing Power

In order for the framework to continue to operate at real-time speeds, then, once the streaming has started, the rendering device (as known in streaming) must have a constant supply of digital data available to be converted to the analogue signal. If the framework cannot provide the required data stream, then the rendering device will not be able to produce a continuous output signal.

Unavoidably, there is a delay from the time at which the Filter Graph sources the stream of digital data and the time at which the signal is output from the rendering device. This delay represents the *latency* of the system. Whilst the latency remains below the time interval between samples (as determined by the sampling rate) the system can operate in real-time.

As an example, if the sampling rate was only 1Hz, then the latency is 1 second, i.e. each sample has 1 second to progress through the system. If we require a 1MHz sampling rate, then our latency is down to just 1µs. Of course, the streaming framework does not process one sample at a time through the system, but rather blocks of samples, however, the principle is the same: if we have a 1MHz sampling rate, where 1 second's worth of samples are processed in each block, then we have a latency of 1 second in which to process 1 million samples. I.e. there is an *average* latency of 1µs per sample.

Figure 1 shows the latency of the system divided into specific timing areas. The time taken for the Filter Graph to process the waveform data, and present it to the rendering device, is denoted by  $T_{\text{Graph}}$ . This time is currently dependant on the processing power of the PC.

The processing power of the PC limits the system in both scalability and maximum signal frequency that can be generated.

## Scalability

As the Signal Graph becomes more complex, more associated DirectShow® filters are added to the Filter Graph. This means that the signal data has to be passed through more stages of, possibly, time-consuming processing, which, effectively, imposes a limit on the number of filters in the Filter Graph (and hence BSCs in the Signal Graph).

## Frequency Limit

Whereas for measurement we can sample the signal at a rate greater than twice the frequency in order to acquire the signal's frequency [3], for generation we require a higher rate if we are to avoid signal degradation.

The effect of under-sampling is demonstrated in Figure 2. Here, we have a sinusoidal signal that is sampled at two rates. The second case, at the rate of 10 samples per cycle, manages to capture the waveform, i.e. the data that is delivered to the rendering device can be smoothed to reconstruct the required sinusoidal signal. The first case, at the lower sampling rate, does not.

If we wish to generate a 1MHz sinusoidal signal, we may require a sampling rate of 5MHz to truly reproduce it. For complex waveform patterns (or even square waveforms, in order to capture the transition point) the sampling rate may have to be much higher.

Figure1. Signal Streaming Process

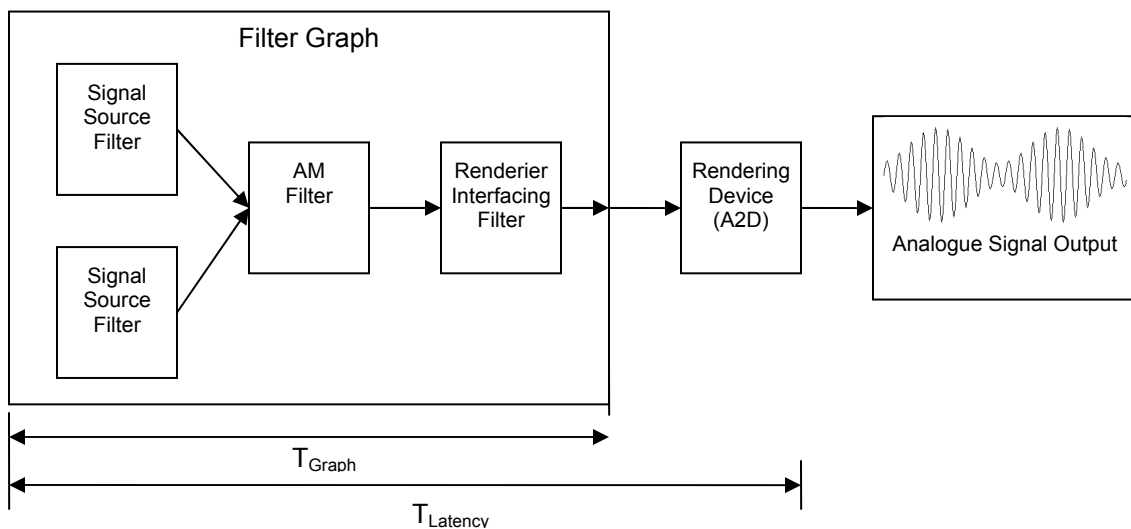
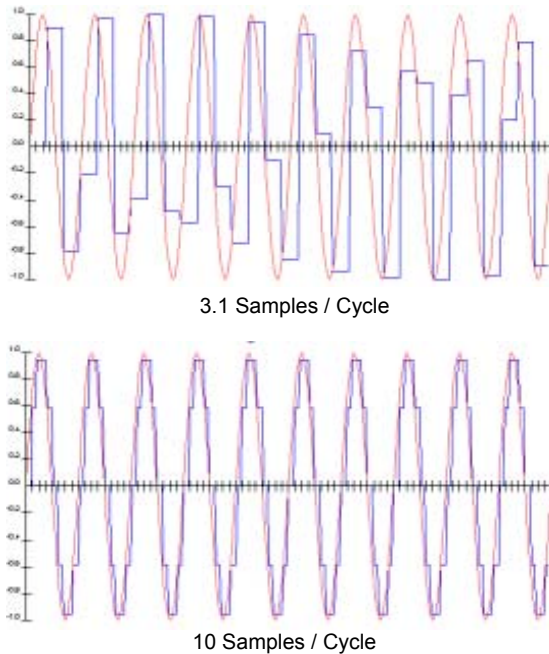


Figure 2. Sampling a Signal



As the sampling rate is increased, the number of samples that are processed by the Filter Graph increases. We then have a proportional increase in the number of calculations performed by each filter, and hence an increase in system latency. Thus, a limit is enforced on the signal frequency that can be rendered in real-time.

## Rendering Device

In order for the generated signal to be useful, it has to output to the real world via a method of D/A conversion. This is the task of the rendering device. Our initial development was based around using the most commonly available D/A on a PC, the sound card.

With the sound card we have been able to produce some genuinely usable Signal Graphs; however the analogue output is limited to the audio frequency spectrum (i.e. less than 25 kHz). For more specialised cases we might require much higher frequency rates, and hence a need for alternative rendering devices.

## DIGITAL MEMORY DEVICES

Towards the end of our last paper [1], we touched upon the ability to download the signal waveform data, as generated by the Filter Graph, into an arbitrary waveform generator (arb). Our initial

research into this area came about from a requirement to generate a fixed length, cyclic waveform. To achieve this, we created a specific rendering filter that dumped the waveform point information into a file on the local storage. In other words, the data was rendered into a file.

We also enhanced our top-level signal generating (or source) filters so that they could be instructed to generate a single cycle of a signal, and then terminate. This was achieved by utilising the end-of-stream [2] mechanism available in the DirectShow<sup>®</sup> framework. We could now construct a file of waveform points of a complex Signal Graph representing one complete cycle.

Using the software (or drivers) provided with the arb, we could upload the waveform point data, and then instruct the arb to generate the analogue signal at far greater frequencies than we could manage on the PC.

Because the generation of the waveform points was not required in real-time, the Signal Graph could be made as complex, and with as high a sampling rate, as required. The limitations were mainly imposed by the arb itself, not by our framework.

## Driver

Having realised the ability to generate high frequency complex signals, we concentrated our effort on creating a DirectShow<sup>®</sup> filter that unified the rendering of the waveform points with the software driver for the arb. By creating a DirectShow<sup>®</sup> filter that could load the arb with the waveform points, we could have a dynamic system with high frequency complex signal capability.

## Applicability

For the majority of the signals that will be generated in a real-world solution, the time it takes to generate the waveform points by our framework will be less than the time it takes to upload the waveform data to the arb. As an example, we can take the Test Signal Framework (TSF) as defined in P1641 [4], which is itself based on the C/ATLAS nouns [5]. Here is defined a common set of signals that are used in real-world scenarios, where the defining TSFs are rarely very complex; few having more than 8 BSCs. Our framework can rapidly generate the

waveform data for these; generally quicker than the time to upload the data to an arb.

The limit imposed on the system is that the time required for the arb to change to a different signal pattern cannot be less than the time it takes for our framework to generate a single cycle of waveform points plus the time it takes to upload the waveform points to the arb:

$$T_{Change} \geq T_{Generate} + T_{Upload}$$

Or, as we would expect the arb to output a fixed number of cycles before changing to another cyclic pattern, the time for a number of discrete cycles output from the arb has to be greater than the total time to generate, and upload, the waveform points:

$$N_{Cycles} \times T_{Cycle} \geq T_{Generate} + T_{Upload}$$

In the real world, this is a reasonable, if not discountable, limit. If an arb was the instrument of choice of a system designer creating a synthetic instrument solution, then it would have been chosen knowing it would always be limited by the amount of time that it takes to upload the (complex) waveform data into the arb.

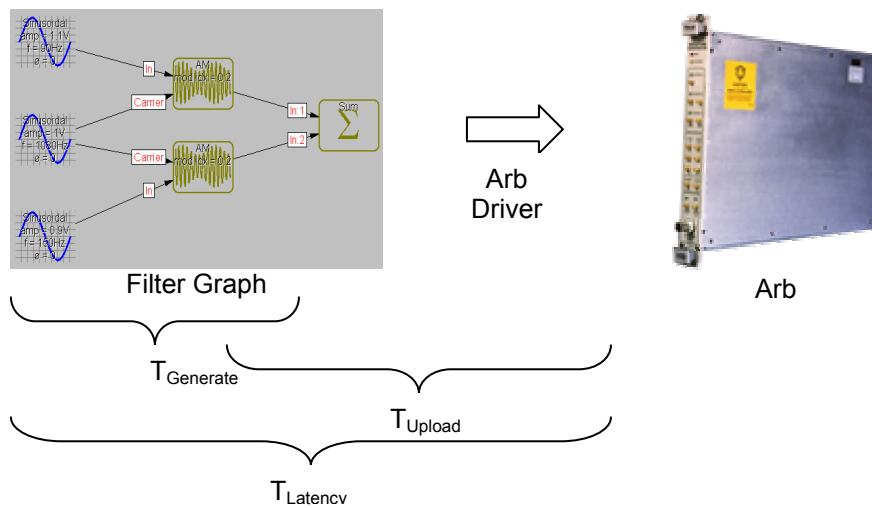
For example, a typical testing scenario would be to apply a stimulus, i.e. the generated signal, to the unit under test (UUT), and measure some output of the UUT to see if it is performing as expected. The stimulus would be applied as long as was necessary to perform the test, and would not be required to dynamically change for that part of the test.

When the discrete test was completed *then* it may be a requirement to apply a different signal. The new waveform data could be calculated and uploaded to the arb following the last test, or, for speed of testing purposes, our framework could calculate the waveform and have it ready to be uploaded to the arb when the last test was completed.

If the arb has support for multiple buffers (or segregation of the main buffer), then the process is speeded up further. Whilst a signal is being generated by the arb, from the waveform data previously loaded into one of its buffers, our framework calculates the new waveform data and uploads it into one of the arb's other buffers. At the relevant time, the arb is instructed to switch its output to the data in the newly loaded buffer.

We further speed up this process by, effectively, overlapping the generation of the signal's waveform data (by the Filter Graph) and the upload of the waveform data to the arb. We can achieve this by making use of multi-threading, i.e. while the Filter Graph is calculating the waveform data points, the arb driver is a small step behind, uploading the data. Because the generation and upload times are overlapped, we are reducing the latency of the system (see Figure 3). (Note: the small start-up delay of the arb, i.e. from the point the arb has received all of the waveform data to actually starting the output of the signal, should be taken to be included in the upload time.)

Figure 3. Latency of an arb-driving Filter Graph



What we have created by encapsulating our framework and the arb driver, is a P1641-based synthetic instrument. We can give our system a P1641 Signal Graph, and the specified signal is generated and output to the real world.

Further, where we have used the PC sound-card, as a capture device, in order to successfully perform sensing, measurement and analysis of audio frequencies, we can use another digital memory device, the digitiser, as the capture device for higher-frequency signals. We then have a system, integrated with digital memory devices, capable of high-frequency source and response.

## **DYNAMIC SIGNALS AND DEDICATED HARDWARE**

In order for our framework to be able to generate dynamic signals, the total time it takes to generate the signal (or latency of the system) must not be an increasing value. As Figure 1 shows, the latency of the system is represented by the time that the signal generation begins, and, after several stages of processing, is finally output from the D/A at time  $T_{\text{Latency}}$ .

In our arb-based system, we have achieved a fully functional synthetic instrument. However, the latency of our system is highly dependant on the arb's capabilities. Namely, the time it takes to upload the signal's waveform data, and, the maximum size of the data that can be uploaded. There are a number of possible reasons why overcoming these limitations would be beneficial.

If we want to generate a signal with a large number of samples, the buffer(s) available in the arb may not have enough memory to hold all of the waveform data. This would mean that the data would have to be uploaded in a number of stages, and hence the signal now becomes fragmented. Fragmenting the signal into stages would be problematic if the upload duration exceeded the output duration, as this would lead to pauses in the output. For example, a system outputting a long, high sampling rate, non-periodic stream (such as speech).

Another case where a dynamic system would be beneficial is if the source of the signal generation had to respond to events from outside of the system. As information was passed into the system, a new signal would be generated. The

efficiency of the system might depend on how quickly the new signal could be generated; therefore eliminating any delays, such as loading the arb's buffers, would be valuable.

In order to create this dynamic system, there are two main targets to achieve: to accelerate the processing of the DirectShow<sup>®</sup> filters, and to provide a fast D/A device.

There exists technology that would allow us to fulfil these targets, and hence be capable of generating arbitrarily changing signals, in real-time, without the delay incurred by using the arb for the signal generation.

## **Configurable Integrated Circuits**

Currently, all of our implementation of the BSCs is through DirectShow<sup>®</sup> filters implemented in software. By making use of field-programmable gate arrays (FPGA) to perform the time-consuming, mathematical, processing of the signal data, we can increase the throughput of data within the framework. This then allows us to process larger amounts of signal data (i.e. higher frequency waveforms) without increasing the latency within the system.

## **High Frequency Rendering Devices**

As we have outlined, using an arb, in conjunction with our framework, has provided us with one method of achieving high-frequency signal synthesis. However, we can integrate some efficient dedicated devices to achieve a far more elegant rendering device.

There are already many third-party D/A devices available, where even relatively basic units have the capability of processing 14-bit data at rates of 400MHz. Such a device would allow our PC-based system to overcome many of the limiting factors inherent in the current framework.

## **Framework Enhancement**

With a custom PC-interface board, containing FPGA units and the D/A device, we then have DirectShow<sup>®</sup> filters that, effectively, act as hardware controllers. The filters configure the FPGAs to process the waveform data as defined in the BSCs [4]. The rendering filter acts as the controller for the D/A device.

## INSTRUMENT CAPABILITY DESCRIPTION

At AutoTestCon '02 we presented a paper titled 'Synthesis of Complex Signals on Test Equipment'. In this paper we considered how the BSCs of the P1641 standard might be used to describe instrument capability. For example, an r.f. synthesiser (Figure 4) may be considered as a combination of **sinusoid, triangle & squarewave** sources, combine by **amplitude, frequency & phase** modulators.

Some test standards, such as ATML, are beginning to consider the use of instrument capability descriptions to provide general-purpose automatic test systems (ATS) which can answer the question, posed by a test, 'can you support the signals and measurements that I need?'

So far, in our discussions, we have only been concerned with digital techniques for generating the signals and measurements that we want. However, there has to come a point where we must recognise that we cannot produce signals above a certain frequency.

We propose a super instrument driver, which can incorporate a mixture of digital memory devices and traditional instruments and describe their signal and measurement capabilities in terms of P1641 BSCs (Figure 5). It would be the task of the driver to manage instrument resources, allocating these on the basis of capability requirement, as requested by a test program.

Figure 5 P1641 Super Instrument Driver

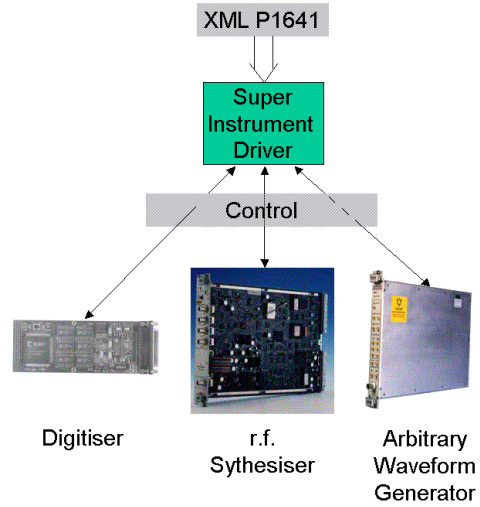
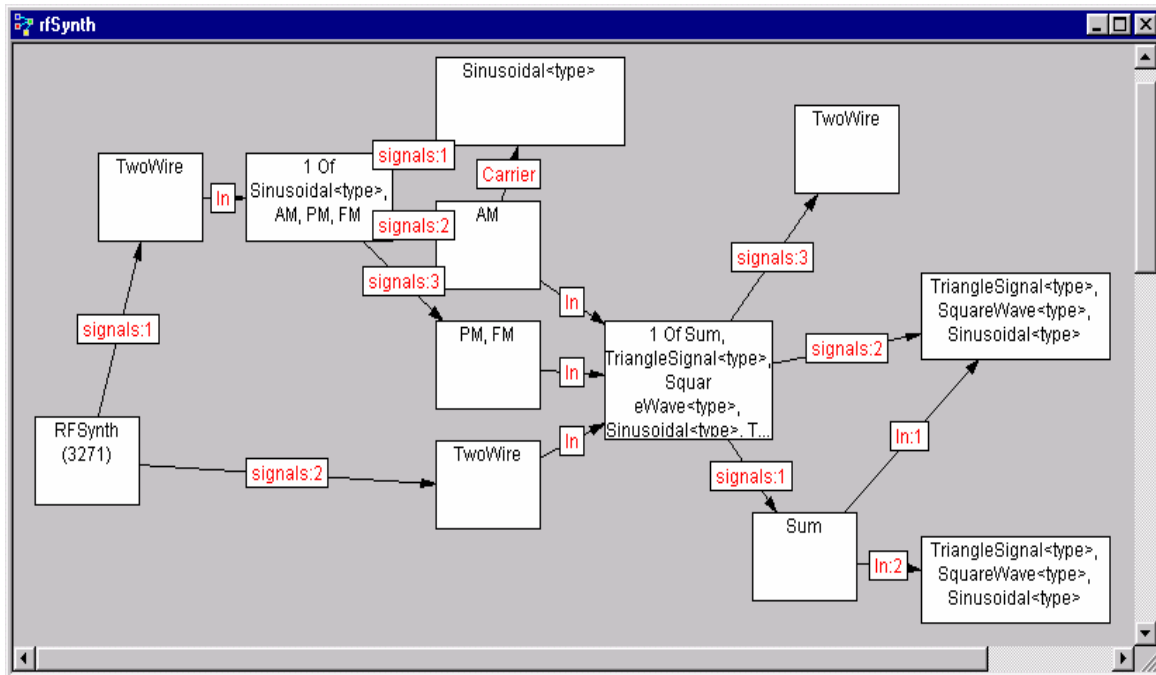


Figure 4. RF Synth Description



In the case of traditional instruments, it has already been shown how capability might be described [6]. For such instruments it is reasonably clear-cut to include specifications such as 1kHz, +/- 4Hz, since these are typically documented by the manufacturer.

Digital memory devices can, by their nature, produce all signals and measurements described within P1641. What is required, however, is a description of their limitations. For example, they cannot produce a sinusoid of exactly 1kHz, rather a **sinusoid** of 1kHz, +/- 3Hz and within a certain **range**, though it is a little more difficult to characterise these specifications than with traditional instruments.

A test that requires a certain stimulus signal can now query an ATS through our driver to determine whether it is capable of producing the signal and, thus, executing the test.

## CONCLUSION

We have developed a PC-based, synthetic signal generating system. The system has been designed to conform to *P1641 of the IEEE Signal and Test Definition Standard*, and the software framework is based upon the publicly available interfaces of Microsoft's DirectShow<sup>®</sup> streaming technology (part of the DirectX<sup>®</sup> Application Programming Interfaces).

As a totally software-based solution, our system has practical uses within the automatic test and measurement field. However, we have investigated the factors that enforce limits on the system and established hardware-assisted techniques that overcome these limits, hence advancing the evolution of the system

By combining digital memory devices and traditional instruments in a common framework, describing instrument capability through P1641, the versatility of P1641 can be realised, while retaining the specialist capabilities that currently only exist in the domain of traditional instruments.

Our framework, along with the humble PC, is well on the way to becoming a, standards-based, mini-ATE.

## 7 REFERENCES

- [1] *PC Based, IEEE Signal & Test Description Standard, Synthetic Instruments*, AutoTestCon03, Matt Cornish, et al.
  - [2] *DirectX 9.0 Software Development Kit*, Microsoft Corporation.
  - [3] *Nyquist-Shannon sampling theorem*. Harry Nyquist (1928) and Claude E. Shannon (1949).
  - [4] *\*\*\* IEEE P1641/D1, Draft Standard for Signal and Test Definition*, IEEE, October 2003.
  - [5] *\*\*\* ANSI/IEEE Std 416-1984, IEEE Standard ATLAS Test Language*
  - [6] *Synthesis of Complex Signals on Test Equipment*, AutoTestCon02, Matt Cornish, et al.
- 
- DirectShow and DirectX are registered trademarks of Microsoft Corporation.