GENE FRANTZ AND RAY SIMAR, TEXAS INSTRUMENTS

DSP: Of Processors

and Processing

There's more than
one way to DSP.

Digital signal processing is a stealth technology. It is the core enabling technology in everything from your cellphone to the Mars Rover. It goes much further than just enabling a one-time breakthrough product. It provides ever-increasing capability; compare the performance gains made by dial-up modems with the recent performance gains of DSL and cable modems. Remarkably, digital signal processing has become ubiquitous with little fanfare, and most of its users are not even aware of what it is. Therefore, it is worthwhile to look at the development history of DSP, an explanation of what the technology is, and a review of the many technologies that are used to implement modern digital signal processing systems.

DSP: Of Processors and Processing

HISTORY AND APPLICATIONS

Digital signal processing is a wonderful blend of the theoretical and the practical. It is this blend that helps to explain much of its historical development. In the broadest sense, digital signal processing is the transformation of signals that have a digital representation. Today that has come to mean a large number of diverse processing tasks, as varied as voice compression, image recognition, and robotic control systems.

But the beginnings were humble indeed. In the late 1940s the now legendary figures of Shannon and Bode discussed the feasibility of digital elements to construct a filter. Because of practical concerns of cost, size, and reliability, however, the nod went to the continued use of analog filtering and spectrum analysis techniques. In the 1950s, with the increasing access to mainframe computers, some digital signal processing applications began to appear. Most notably, seismic scientists began to apply a limited set of digital signal processing techniques to their problems.¹

In the mid-1960s, it was becoming apparent that the integrated circuit offered a pathway to complete digital signal processing systems. This drove the development of a more formal theory of digital signal processing. This era saw the significant contributions of Kaiser in the area of digital filter design and of Cooley and Tukey in the development of a fast method of computing the DFT (Discrete Fourier Transform). The many variations and extensions in this area are referred to as FFTs (Fast Fourier Transforms).

These seminal works in the time domain (digital filters) and the frequency domain (FFTs) showed that this fledgling technology could apply to the duality of time and frequency representations of signals. These early, significant successes clearly demonstrated that digital techniques could offer cost, size, and performance advantages that were not available just 20 years earlier. By the 1970s, the technology was becoming more widely disseminated. This is best exemplified by the release in 1975 of two important textbooks: *Theory and Application of Digital Signal Processing*² by Rabiner and Gold targeted the practicing engineer and advanced course work; *Digital Signal Processing*³ by Oppenheim and Schaffer targeted graduate students.

Today, digital signal processing has become a ubiquitous technology. It has enabled the growth in cellular technology through cellular phone and digital base sta-

A Simple DSP System

Signal Flow Graph for an FIR Filter

Difference Equation for an FIR Filter

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) + \dots + b_Mx(n-M)$$
$$= \sum_{m=0}^M b_mx(n-m)$$

FIG 3

tions, and DSL and cable modems for the home and their end-point connections in the central office. Digital signal processors (DSPs) power consumer products from digital still cameras to PDAs to high-performance home audio receivers.

Let's consider a simple DSP system (see figure 1), which will help to illustrate many of the components found in most DSP subsystems. Of course, to do digital signal processing, you must have a signal represented digitally. Many systems have an A/D (analog to digital) converter that samples the analog signal and converts it into a digital representation. At this point, the DSP takes over and processes the signal. Then the signal must be converted back into an analog representation by a D/A (digital to analog) converter.

To become familiar with the types of computations performed in digital signal processing applications, consider the two equivalent representations of the FIR (finite impulse response) filter shown in figure 2.

The first representation is a signal flow graph that shows the flow of data in the operation of a FIR filter. The z^{-1} elements represent delay elements. These can be seen more easily by the difference equation representation of the FIR filter in figure 3. The FIR filter illustrates the most common characteristics of many DSP algorithms:

- Large number of data memory accesses: $2 \times (M+1)$ in this case.
- Large number of computations: $M+1$ multiplies and M adds in this case.
- Repetitive calculations: multiply and add in this case.

Certainly not all digital signal processing applications look like the FIR filter. IIR (infinite impulse response) filters have a different structure from FIR filters. FFTs have a different mix of multiplies and adds. Many sophisticated applications, such as voice coders and image coders, have a large portion of control code. Because of these varying demands, there are, as you might expect, a number of

implementation options.

PRECISION AND DYNAMIC RANGE

A practical implementation of a digital signal processing algorithm, using fixed-point arithmetic, must address issues such as the precision of variables, the dynamic range of results, and the importance of handling overflow conditions. To illustrate some of these issues, let's consider a practical example.

When performing a multiplication between two 16-bit numbers, a 32-bit product is created. At the heart of many DSP algorithms is a loop that performs a series of additions of these 32-bit products. It was recognized long ago that this series of additions would quickly overflow a 32-bit register. To address this problem, DSPs have used two basic strategies: one is to add guard bits; the other is to saturate upon overflow or underflow.

Guard bits extend the storage of the product to more than twice the size of the multiplier input. For example, when multiplying two 16-bit numbers, their 32-bit product is added to a register that is wider than 32 bits, typically a 40-bit register. This is the approach used by most modern DSPs. Early DSPs, like the 32010 and 320C25 (as shown in figure 4), had no guard bits, but they did use saturation.⁴

A number of VoIP (voice over IP) and standard telecom applications require guard bits—that is, 40 bits—to work well, most notably, echo cancellation, LMS (least mean square) filter tap update, and power calculations using exponential averaging.

In a processor, it is necessary to represent numbers with a finite number of bits. Multipliers, register files, and ALUs (arithmetic logic units) are all built to handle data of a finite size. This finite size introduces effects that must be understood when implementing an algorithm.

This condition was recognized in the earliest digital signal processing work. One of the most studied cases is the impact of overflow on digital signal processing calculation.

Figure 5 shows the two situations that can occur. Overflow results in dramatic discontinuities. To see how this is so, consider a radio whose volume knob has the natural overflow characteristics shown in figure 5. As we turn the volume knob down, it reaches its lowest volume; but if we turn the knob down just a bit more, we would suddenly be at the peak volume. Not a very desirable behavior! On the other hand, the saturation characteristics shown in figure 5 provide a much more reasonable behavior. As the volume knob of the radio is turned down, we reach the lowest volume and stay there. These

DSP: Of Processors and Processing

effects are problematic since they turn well-behaved linear systems into often difficult-to-analyze nonlinear systems.

In a system as simple as an IIR filter, overflow is a very

important consideration. For a stable IIR system, implemented with infinite-precision arithmetic, if the input becomes zero and remains zero, the output will decay asymptotically to zero. But such is not the case for a real

system that must necessarily deal with finite-register arithmetic, since digital signal processing algorithms are implemented on computers that have storage of a prescribed word width.

For the same IIR filter, implemented with finite-register arithmetic without the benefit of saturation, the output may continue to oscillate indefinitely. This effect is referred to as limit-cycle behavior and is a consequence of overflow. Because overflow inserts a gross error in the output, the filter output can thereafter oscillate between large-amplitude limits. These types of limit cycles are referred to as overflow oscillation. Overflow oscillations can be avoided by using the saturation overflow characteristics of figure 5.

DSP IMPLEMENTATIONS

Up to this point we have considered digital signal *processing*. Now we turn our attention to its implementation, the digital *signal processor*.

Realtime digital signal processing is a key enabler

TMS320C25 Block Diagram

Note: Shaded areas indicate a bus.

for more and more products. At the same time, these products are under significant development pressures. For example, the consumer space is full of new products such as PDAs, cellphones, and digital still cameras that are smarter, faster, smaller, and more interconnected than ever. Yet every time these products reach a new plateau in terms of capability, customers inevitably begin to ask for more—greater speed, effectiveness, and portability—and they want it now.

Clearly, this puts tremendous pressure on design engineers who are asked to satisfy these varied demands. They must reduce cost and power consumption while increasing performance and flexibility. And they must do all of this in increasingly complex development environments and within a design cycle that is ever shrinking.

In addition, designers are faced with a myriad of implementation technologies, all claiming to best execute realtime operations for a given application. There are at least six different implementation technologies to consider for building a DSP:⁵

1. ASIC (application-specific integrated circuit)
2. ASSP (application-specific standard part)
3. DSP (digital signal processor)
4. FPGA (field-programmable gate array)
5. MCU (microcontroller unit)
6. RISC/GPP (reduced instruction set computer/general-purpose processor)

Each of these options has its strengths and weaknesses. To provide some perspective, we examine these six popular architectures using the following criteria:

Time to market. This is the amount of time it takes to go from idea conception to system implementation to volume production.

Performance. Some systems want maximum performance. Others want just enough to meet the system requirements.

Price. This is the price of the end system, not the development cost.

Development ease. This is one of the biggest influences on time to market.

Power. Different applications may express “power” requirements in different ways.

Feature flexibility. Once the system has been deployed, how hard is it to change—that is, upgrade or enhance?

REALTIME SIGNAL PROCESSING OPTIONS

Before choosing among the six popular DSP implementation technologies, designers need to weigh the strengths and weaknesses of each for their particular applications. Here is a summary of each option.

ASIC. ASIC implementations have some significant benefits, all of which spring from the fact that, since this is a bottom-up design, the implementation can be targeted for specific performance, price, and power goals. But this ability to tune is where we see one of the challenges for ASIC design.

Today’s designers not only have to get the function logically correct, they must also achieve the desired clock rates. Missing clock-rate goals can often mean redoing an implementation that is logically equivalent to the target

DSP: Of Processors and Processing

but is very different in terms of implementation.

The major trade-off made when choosing an ASIC implementation path is the impact on time to market. ASIC design cycles, design validation, fabrication, and qualification can easily take more than a year.

Finally, although the final ASIC produced may be inexpensive, ASIC approaches are the most expensive when it comes to overall development cost. This is attributed to the amount of logic design necessary to create the application on the chip, coupled with the ever-increasing cost of silicon processing, with multiple hundreds of thousands of dollars per full reticle revision of silicon. In terms of development help, ASIC provides general support but does not offer any application-specific help because of the general lack of application-specific content knowledge by ASIC suppliers.

So the ability to tune an ASIC brings about significant advantages in terms of performance, product cost in high volume, and power or energy, but this has to be weighed carefully against its weakness of lengthy development times, high development costs, and often a reduced flexibility design.

ASSP. Application-specific products have been designed to perform specific functions and are available off the shelf. An example would be an MPEG decoder. With an ASSP, the implementation technology used—that is, ASIC, custom, or programmable—is not its distinguishing characteristic. Rather, its distinguishing characteristic is that it has been optimized for a specific application. This allows for potential advantages in terms of power (energy), performance, and price. Since the product has already been designed, it avoids the lengthy design times of ASIC and allows for rapid incorporation into an end system.

If the target market for an ASSP already exists and the ASSP already exists, the time to market could be

good to excellent. As an ASSP is specifically tuned to a particular application, it should offer high application-specific performance. Similarly, it should be good in terms of price and power.

Of course, like all of the approaches we are considering, ASSPs have a weakness. Often they are somewhat poor in flexibility, as they are inherently specific to their application and, in particular, to their unique solution approach to the target application. This specific focus and optimization is a trade-off for flexibility.

DSP. DSPs and other programmable solutions such as RISC processors and MCUs are very good in addressing time-to-market issues. Their advantage of software programmability to achieve different functions and features saves time to market compared with more hard-wired implementations, such as ASIC.

Because DSPs are specifically targeted at digital signal processing applications, their toolsets are often tuned to meet the needs of those applications. DSPs are also strong in terms of performance since they are tuned to specific application areas and specific performance levels. DSP is not as cost effective as ASIC or MCU, but not far off from MCU.

DSPs are very power efficient, especially when you consider DSP platforms designed specifically for low-power, handheld applications such as TI's TMS320C5000.

DSPs are typically complemented by powerful and easy-to-use programming and debug tools, and by technical support networks and applications engineers who understand the realtime world and are ready to help customers achieve their realtime designs.

In terms of development cost, DSP programmability allows for faster development cycles for the desired function versus developing application-specific chips. With proper use of high-level programming and/or standard code modules,

Realtime signal processing is taking the digital revolution to the next step.

you can cut development time significantly and thus save development cost.

Since a DSP can use software programmability to achieve different functions and features, it offers shorter time to market than similar hard-coded logic implementations. For realtime signal processing, DSP is rated the best among programmable processors (DSP, RISC, MCU) because it has the best and most relevant toolset and value web to achieve realtime signal-processing-relevant functions.

FPGA. FPGAs offer strength in terms of time to market. They allow modifications in the field to support a modest amount of functional changes, but their flexibility is not as high as the software-programmable alternatives—DSP, MCU, and RISC. FPGAs have better support and faster design cycle time than ASSP or ASIC, however, and thus can claim faster time to market than those alternatives.

FPGAs are also strong in terms of performance. With FPGAs, developers can tune hardware gates specific to the application, thus delivering high application-specific performance.

FPGAs have some notable drawbacks. They can be costly; from a logic gate perspective, they are the most expensive alternative discussed here. FPGAs tend to be in larger packages than other solutions, so in systems where board area is at a premium, they may not be a good fit.

Also, the power consumption of an FPGA can be high. Technology advances will lower FPGA power, but likely not enough to change its place in the relative ranking on power efficiency.

FPGAs originally had significant challenges when it came to ease of development. While they are still not as strong as programmable processors, they have made significant improvements in their development tools in recent years. FPGAs would rate the best on development cost assuming two situations: that the toolset for FPGA programming is not too expensive; and, assuming the developer is dealing primarily with hardware, that the engineer is involved in the development. If development leans toward software engineers, then FPGAs would increase in effort and relative cost. In terms of development help, the tools and support structure for FPGA-based designs seems to be well established and acceptable

to OEMs.

MCU. Like other programmable solutions, MCUs can use software programmability to achieve different functions and features, saving time to market, as opposed to similar hard-coded logic implementations. Compared with RISC/GPP, the MCU has lower mathematical processing resources and typically slower operating frequency. The MCU typically has a small chip size and thus a relatively low price. Typically, MCUs are general in nature, making them less power efficient than DSPs or ASSPs.

The simplicity of MCUs results in lower performance and clock speed but also less hardware dedicated to performance-enhancing feature support and a simple memory architecture. Typically using less silicon resources than RISC or FPGAs, MCUs are more power-efficient than those alternatives. MCU programmability of existing chips allows for faster development cycles for the desired function, versus having to develop application-specific chips or ASICs. With proper use of high-level programming and/or standard code modules, development time can be significantly reduced, saving development cost. In terms of development help, most MCU suppliers have a network of support as well, although it does not score an excellent rating because much of that network is not experienced with realtime applications, merely embedded applications.

RISC/GPP. RISC/GPP processors are programmable, allowing them to employ different functions and features, saving time to market as opposed to similar hard-coded logic implementations.

RISC's typically high megahertz levels yield decent signal processing, but its lack of mathematically specific single-cycle instructions and DSP-specific features limit its realtime performance. RISC processors have significant general-purpose functions on board that tend to make them well suited for general-purpose applications, but less so on cost effectiveness for realtime signal processing.

TABLE 1 Summary of Implementation Technologies

	Time to Market	Performance	Price	Ease of Use	Power	Flexibility
ASIC	Longest	High	Low	Hardest	Low	Low
ASSP	Shortest	High	Low	Easiest	Low	Low
DSP	Short	High	Low	Easy	Low	High
FPGA	Short	High	High	Hard	High	High
MCU	Short	Lowest	Low	Easy	Low	High
RISC/GPP	Short	Low	High	Easy	High	High

DSP: Of Processors and Processing

Typically, because RISCs are general in nature, they are less power efficient than DSPs or ASSPs.

RISC programmability of existing chips allows for faster development cycles for the desired function than developing application-specific chips or ASICs. Proper use of high-level programming and/or standard code modules can cut development time significantly and thus save development cost.

CONCLUSION

Digital signal processing applications are so diverse that they make it necessary to have a number of implementation alternatives. These are summarized in table 1. Clearly, no one solution is best in all cases. The challenge for the system implementers is to choose the solution that best meets their system and market requirements.

Realtime signal processing is taking the digital revolution to the next step, making equipment that is more personal, more powerful, and more interconnected than most people ever imagined possible. Over the years, different technologies have powered the most innovative creations from the mainframe and minicomputer eras to the PC and today's Internet era. Consumers are driving realtime functionality, demanding equipment that is extremely fast, portable, and flexible. To meet those needs, designers are facing more pressures than ever, but they also have more options than ever to address them.

Careful evaluation of each option clearly shows several viable alternatives for embedded applications. For implementing today's realtime signal processing applications, however, DSP is very often the best choice. No digital technology has more strengths than DSP nor better meets the stringent criteria of today's developer. Certainly, other digital options can address any one of these relevant problems well, but only with clear trade-offs.

DSP gives designers the best combination of power, performance, price, and flexibility and allows them to deliver their realtime applications quickly to the market. □

REFERENCES

1. Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*. Prentice Hall, Englewood Cliffs: NJ, 1975.

2. See reference 1.
3. Oppenheim, A. V., and Schaffer, R. W. *Digital Signal Processing*. Prentice Hall, Englewood Cliffs: NJ, 1975.
4. TMS320C2x User's Guide. Texas Instruments, Document number SPRU014C (January 1993).
5. Adams, L. Choosing the right architecture for realtime signal processing designs. Texas Instruments, Document Number SPRA879 (2002).

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

GENE FRANTZ is a principal fellow at Texas Instruments, heading TI's Technical Advisory Board providing long-term guidance to top management about emerging technical trends that may impact TI's business and products. He is also the DSP business development manager, responsible for creating new businesses within TI using digital signal processing technology. His documentation of the relationship between power dissipation and performance is becoming broadly accepted as "Gene's Law." Frantz joined TI in 1974 in the Consumer Products division leading the educational products development team to create the Speak & Spell learning aid. He holds 30 patents in the areas of memories, speech, consumer products, and DSP. He has a B.S.E.E. from the University of Central Florida, an M.S.E.E. from Southern Methodist University, and an M.B.A. from Texas Tech University. He is a fellow of the IEEE.

RAY SIMAR is responsible for enhancing Texas Instruments' DSP solutions leadership position by developing advanced architectures for diverse applications. In 1997, Simar was elected as a TI fellow in recognition of his pioneering work in DSP technology. He is the chief architect and program manager of the TMS320C6x, based on VelociTI, an advanced VLIW architecture, to achieve very high performance at low cost. Simar joined TI in 1984. Prior to the 'C6x, he was the chief architect and program manager for the floating-point TMS320C3x and TMS320C4x DSP devices. He received his B.S. at Texas A&M University and M.S.E.E. from Rice University. Simar holds more than 10 patents in DSP technology.

© 2004 ACM 1542-7730/04/0300 \$5.00