A Bit-Serial VLSI Architectural Methodology for Signal Processing

Richard F. Lyon

VLSI System Design Area, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304 U.S.A.

INTRODUCTION

Applications of signal processing abound in the modern world of electronics. Telephones, stereos, radios, and televisions are the most common examples. In the general electronic communication and control market, there is a widespread demand for higher quality and lower cost signal processing components of all sorts. For many years, digital signal processing (DSP) techniques have been touted as "the way of the future," but have consistently failed to make a big impact on the market. We discuss here an *architectural methodology* designed to make digital signal processing "the way of the present."

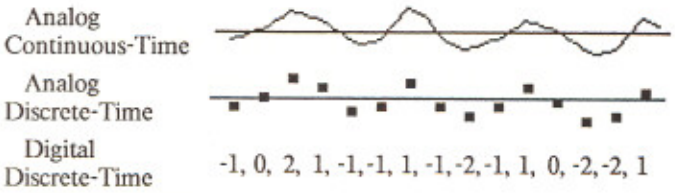
An architectural methodology is a style, or school of design, that provides a basis for a wide range of architectures for different functions. The architectural methodology presented here is built on top of the logic, circuit, timing, and layout levels of VLSI system design methodology presented by Mead and Conway (1980). It includes a large component that is independent of the underlying technology.

Researchers working on DSP theory and applications have had a hard time implementing their ideas in cost-effective hardware, because the IC industry has not figured out how to support their needs. We hope to change that, based on the family of components described, in conjunction with the ability for designers to easily create their own more specialized system components from the silicon layout macros and composition rules that characterize our methodology. The standard-chip way of life has caused many inappropriate architectures to be proposed and tried in the past; the new freedom for non-specialists to easily design custom integrated systems will enable a wave of new applications and new architectures.

SIGNALS and SIGNAL PROCESSING

Signals are time-varying measurements or simulations of real-world phenomena; for example, an audio signal represents minute changes in air pressure as a function of time. In most familiar signal processing equipment, a signal is represented by an electrical analog, such as a continuously changing voltage; such analog signal representations can be directly processed through continuous-time components such as resistors, capacitors, inductors, diodes, amplifiers, etc. The modern alternative is to sample the signal at equally spaced

instants of time, and to represent those sampled values either as discrete-time electrical analogs (e.g. amount of charge held on a capacitor) or as numbers (in some kind of a digital computing machine).



The theory of sampling and discrete-time signal processing is quite well developed, and applies to either analog or digital representations of signals. Switches, capacitors, and amplifiers are typical building blocks for analog discrete-time signal processing. For digital signal representations, the basic building blocks are memories, adders, and multipliers. Analog noise, analog drift, and digital roundoff effects of these components are also well understood.

Digital representations other than sequences of sample values are sometimes used; for example, delta-modulation and its variants use a fast sequence of one-bit values indicating whether the signal is above or below its predicted value at each time instant. These representations will not be considered here.

Using modern MOS VLSI technologies, large amounts of signal processing can be done with a small amount of silicon, compared to the more mature continuous-time analog technologies. Yet, signal processing devices remain relatively expensive and in various ways limited. We believe that difficulties in both design and usage of these devices is the reason. Therefore, we base our architectural decisions on our desire to jointly minimize design difficulty and usage (programming) difficulty, subject to maintaining the high performance promised by the technology. We also believe that it is not possible, for the range of applications we are considering, to get good enough performance from analog VLSI technologies; nor is it easy to design the required analog circuits. Accordingly, we have arrived at these basic decisions:

Question:	*Answer:*
Continuous-time or discrete-time? (so that MOS VLSI techniques, particularly digital techniques, can be used)	Discrete
Analog or digital sample representation? (to achieve highest quality and tractable design)	Digital
Bit-parallel or bit-serial number representation? (for high clock rate, and maximum flexibility and extensibility)	Bit-serial
Bus-oriented or dedicated signal paths? (for maximum extensibility, and efficiency)	Dedicated
General-purpose programmability or specialized? (for efficiency and ease of application)	Specialized
Fixed or variable filter parameters? (to cover the widest range of applications, including time-varying filters)	Variable
What number system? (for efficiency and ease of logic design)	Fixed point 2's comp. LSB first

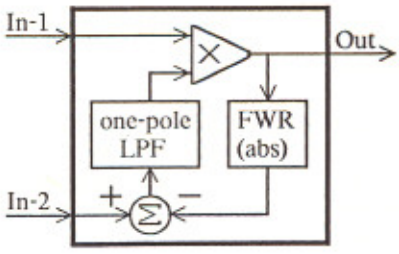
DIGITAL FILTERS

Probably the most commonly needed signal processing component is a filter. It is simply a linear (usually time-invariant) system with memory (i.e. the output is a linear function of some history of the input). Most commonly, the input and output are single scalar signals, though this is not required; complex- and vector-valued signals will not be discussed, but are easily accommodated in this architecture. In a digital discrete-time architecture, a filter is a computation that operates on a sequence of numbers as input, and produces another sequence of numbers as output. The usual purpose of such computations is to pass some frequencies of signals, while attenuating others. See Moore (1978) for an introduction to the mathematics of digital filters and related signal processing components; for more detailed information, including four chapters on hardware implementations, see Rabiner and Gold (1975).

COMPONENTS AND HIERARCHY

The remainder of this paper explains how to implement signal processing components (operators and systems) in a style evolved from that presented by Jackson, Kaiser, and McDonald (JK&M) in their 1968 paper "An Approach to the Implementation of Digital Filters." JK&M's notion of an *approach* is similar in some respects to our more developed notion of an architectural methodology, but lacks the notions of standardized interfaces and hierarchical composition of operators.

Our system-building strategy is to design top-down, decomposing blocks functionally into more detailed block diagrams, until we get down to low-level operators, such as adders and multipliers; then, to construct the system from the bottom up, by assembling operators into higher-level operators. To do this, we need conventions for the hierarchical definition and construction of operators. This methodology does not severely constrain the range of possible architectures, but unifies many architectures, allowing them to share components at many levels. The general features of the methodology and conventions are described in following sections.

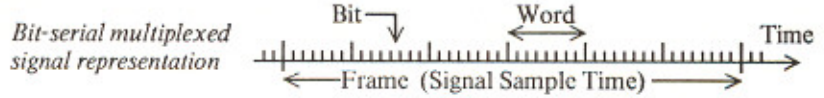


An automatic gain control operator, composed of other operators.

The architectural methodology involves the use of heavily pipelined bit-serial arithmetic processors, all operating at a fixed throughput rate (words per second), and *multiplexed* over several channels or functions to match processing hardware rate to the signal sample rates. In this approach, it is important that the throughput of an adder be the same as that of a multiplier, which is in turn the same as that of a second-order filter section, or a delay element, or any other component (and the same as a wire, the elementary data path).

MULTIPLEXING

Digital time-division multiplexing is a technique developed by the Bell System to send several digital sampled data signals on a single wire pair (one-bit data path), by dividing time into periodically recurring time-slots, each of which could carry one signal sample. The number of signals that can be carried on one data path is then the word rate divided by the signal sample rate (the word rate is the bit rate divided by the number of bits per word, since a data path carries one bit at a time at a fixed rate). Bell's T1 carrier system is a good introductory example; it carries 24 voice-band signals, sampled at 8000 samples per second (8 ksps), with 8 bits per sample, on a wire pair with 1.544 Mbps total data rate (that's 24*8000*8 plus a few extra bits). These multiplexing concepts are easily extended from transmission of signals to processing of signals.



For many operators, multiplexing is as simple as time-interleaving the samples of several input signals, resulting in interleaved answers. But when the definition of an operator requires some *state* associated with a signal, multiplexing that operator requires that more state be saved, separately for each of the interleaved input signals. In digital signal processing, all state information is conventionally saved inside the unit-delay operator (called Z^{-1}, the inverse of the unit advance operator of Z-transform notation), which simply produces an output value equal to what the input value was one sample time earlier. The problem of saving state reduces to the problem of multiplexing the Z^{-1} operator. This operator must store one word for each multiplexed signal, so it is logically just a long shift register (if the signals being multiplexed do not all have the same sampling rate, things are more complicated, but still tractable).

CONVENTIONS

System-wide clocking, signalling, timing, and format conventions are needed in order to hierarchically design a system of potentially very high complexity. The basic approach we have adopted is to have system-wide synchronous clocks at the bit rate, and both centralized and distributed timing and control signal generation, as described below.

Clocking

The synchronous clocking scheme will include several versions of the bit clock, provided to accommodate the different types of technology from which the system will be constructed. These technologies and clock types are (1) LSTTL with positive edge-triggered clocking, and (2) high-performance NMOS with nonoverlapping two phase clocking. The relative clock phasing and data timing across technology boundaries will be fixed system-wide. In the NMOS part, the first clock gate encountered by a signal entering a subsystem will be designated Phi-2 (Phase-In), which will be high during the latter (low) part of the TTL clock cycle, when data bits are stable. Phi-1 (Phase-Out) will be the last clock gate encountered by a signal leaving an NMOS subsystem, and it will be timed such

that output data, stable during Phi-2, meets the TTL setup and hold requirements (see chapter 7 of Mead and Conway 1980).

Signalling

Signalling refers to the electrical representation of bits on wires. We adopted the electrical convention that a low voltage level would represent numerical digit 1, and a high level would represent numerical digit 0 (so that unconnected pulled-up inputs default to a zero signal). This is true within and between operators of any scale (subsystems, chips, or whatever). Bits are transmitted with a non-return-to-zero (NRZ) representation, which just means that the voltage changes to the voltage representing the new bit value after the appropriate clock edge, and remains throughout the clock period.

Timing signals are all active-high; the higher voltage state means logically true, asserting the named time state (e.g. the *LSBtime* signal discussed below is high during the least significant bit time of a word).

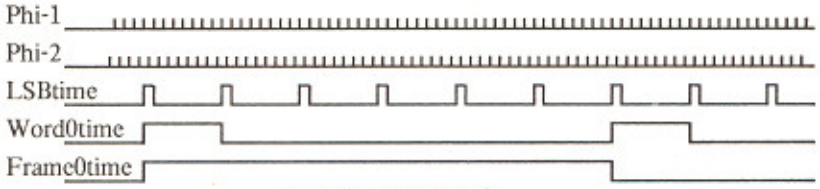
More electrical conventions for between-chip signalling are embodied in the I/O pad designs used in the NMOS parts. An informal statement of the conventions is that the NMOS parts be "compatible" with LSTTL (fanout of 10) in levels, noise tolerance, and transition times. The signalling conventions are about the only conventions that would need to be changed to accommodate new and different VLSI technologies as they become available. Some changes in clocking conventions may also be desirable at some point.

Timing

The centralized control signal generator is a time counter that keeps track of bits, words, etc. on a cycle equal to the slowest period of interest in the system. This control section may be thought of as a microprogram store with no conditional branching, a very wide horizontal control word output, and a long linear program (but the actual PLA encoding will be much more efficient).

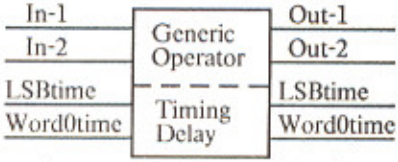
The distributed control scheme is more widely used, and is more suitable for actually addressing the large-system issues, where design of a single controller for the many functions would be a task beyond the capabilities of a human designer, and would add too many interrelated constraints between parts of the system.

The basic distributed control strategy is that each data path be associated with a timing signal (which is used and generated locally) that represents time within a word; optionally, data paths may be associated also with other timing signals that represent time within a larger *frame*, as appropriate for a particular operator. The standard representation of time is simply for the signal to be true during the first time-slot of its frame, and false at all other times. The signal called *LSBtime* is true during the first (least significant) bit of each word, and the signal *Word0time* is true during the first word of each frame, for example.



Typical timing signals

Most arithmetic processing units need only clock, data, and one control input to identify the first bit of a new word. Thus, we adopt a convention to use an *LSBtime* signal in association with *every* operator input and output: operators accept *LSBtime* as a control input, and produce a delayed version as the *LSBtime* of their output(s). Thus data paths from one unit to another carry their own timing information, and units can be arbitrarily interconnected without concern about connection to a central controller. Still, the designer must take care to assure that merging signal paths (e.g. at an adder) and loops (e.g. in a recursive filter) have the correct total delay (instead, a special stretchable queue could be designed to automatically adjust delays, but this would greatly decrease the efficiency of the methodology at the chip level).



An operator with timing signals.

Format and numerical value restrictions

Some operators, such as multipliers, may take two input operands of different lengths. In such cases, the convention is that both operands are aligned to the same *LSBtime* signal (i.e. words of different lengths are right-adjusted). For short words, the bit slots between the sign bit of one word and the LSB of the next should be filled with sign extensions. Then values of words of any length can be described by the two's-complement binary integer interpretation of the entire 24-bit word (and can be written in binary, octal, or decimal as convenient).

Operators usually require that input values be limited to a range which is less than the total number of bits would allow. For example, the multiplier design for 24-bit words requires that the X input be in the range $[-2^{21}, 2^{21})$, while the Y input must be in the range $[-2^{2k-1}, 2^{2k-1})$, where k is the number of actual hardware stages in the multiplier (not over 12); both X and Y conform to the same format (24 bits per word, LSB first, two's complement), but have different value restrictions. Value restrictions can be enforced by either scalers or limiters, if necessary.

On consistent formats for signals and coefficients

We have deliberately chosen to have signals and coefficients conform to the same format, with their distinction being only in possibly different value restrictions. Many signal processing architectures do not have this property, thereby making it difficult to multiply (mix) two signals together, or to use a filter's output as another filter's coefficient input. These uses of multipliers are not the common time-invariant linear system uses, but are nevertheless widely needed for nonlinear and time-varying processing such as modulation, demodulation, automatic gain control, adaptive filtering, correlation detection, etc. Of particular importance to us is the application of a time-varying filter for speech synthesis; the coefficients of the synthesis filter are themselves the outputs of lowpass (interpolation) filters.

On interpretation of coefficients as fractional values

We have spoken of signal samples with integer values, but we often need to consider the representation of fractional values, especially for coefficients. The interpretation of operands as fractional values derives from the definition of the

function performed by the multiplier. For example, the "5-level recoded" or "modified Booth's algorithm" pipelined multiplier design of Lyon (1976) computes $XY/2^{2k-2}$ for a k-section layout, with Y restricted to $[-2^{2k-1}, 2^{2k-1})$. So if we simply regard Y as a value in $[-2, 2)$, we can say that the multiplier computes XY (as long as we interpret X and XY consistently, either both as integers or both as fractions). Another way of saying this is that a multiplier which accepts n-bit coefficients regards the radix point as being $n-2$ places from the right. Multipliers can easily be designed with other scale factors, to give other range interpretations to the operands. The scaling described here is popular for signal processing.

Pipelining delay and composition of operators

Another important convention is that all operators have some constant positive integer number of clock cycles of delay (pipeline delay or transport delay) between the LSB of an input word and the LSB of the corresponding output word. For some operators, such as adders, each output bit depends only on the corresponding and lower-order input bits, so the delay can be very small--but not zero. We require that no output bit be combinationally related to any input bit, so at least one cycle of delay is required. This allows the system clock rate to be fast, independent of how many adders and other operators are cascaded (long ripple propagation delays are excluded, unlike typical bit-parallel systems).

The delay of an operator is required to be a constant, known at design time, rather than being dependent on control parameters. The designer must satisfy delay constraints, such as equal *LSBtime* for signals that merge in an adder, by inserting null operators with carefully chosen delay wherever needed (i.e. shift registers are added to make signals line up in time). Loop delay constraints are sometimes difficult or impossible to satisfy if operators in the loop have too much delay. Fortunately, all delay information is known at design time, and is easy to represent and design with, by using simple notations.

FUNCTIONAL PARALLELISM and MULTIPLEXING ALTERNATIVES

We have briefly discussed the use of multiplexing to accommodate many separate operations on relatively slow signals by using a small amount of fast hardware. In the other direction, *functional parallelism* can be used to perform operations on very fast signals, by using a larger number of hardware operators running in parallel, with appropriate techniques for combining their partial computations. Thus, it is possible to apply the serial-arithmetic architectural methodology to very wideband applications, such as radar and video processing.

There is really no limit on the bandwidth obtainable, as illustrated by the VFFT series of Fourier transform processors described by Powell and Irwin (1978), which use slow 3 Mbps (custom PMOS) serial chips to compute transforms of up to 10 Msps radio telescope data (i.e. the sample rate is actually faster than the bit rate!).

The techniques for use of functional parallelism in nonrecursive (finite impulse response) filters are straightforward. Similar techniques for recursive filters have been demonstrated by Moyer (1976). The point is that the bit-serial architectural techniques do not overconstrain the system-level architecture, and do not limit the applications to low-bandwidth areas.

Six performance regimes

We are familiar with signal processing system designs that span six distinct regimes, in terms of the amount of multiplexing and/or functional parallelism that they incorporate. Several chips mentioned below have been designed by members of the author's team at Xerox, and will be described in forthcoming papers.

The low-performance regime (level 0) is characterized by a lack of multiplier hardware or other special signal processing features, and is exemplified by software systems running on simple general-purpose processors.

Bit-serial architectures begin to look interesting at level 1, characterized by the use of a single multiplier for different functions. This level is exemplified by Jim Cherry's "Synth" chip, a tenth-order lattice filter for speech synthesis; its block diagram has two multipliers per lattice stage and one overall gain control multiplier, while its implementation consists of one multiplier, one adder, one delay shift register, a little control logic, and a handful of data path switches.

When more total performance is required, level 2 is appropriate. This is basically the simple multiplexed filter approach described by JK&M, which uses one hardware multiplier for each multiplier in the block diagram of a section, but shares them over several sections by multiplexing. Lyon's 32-channel second-order "Filters" chip exemplifies this regime; it has almost no data path switching or control logic. Rich Pasco's "FOS" (first-order section) is another example, which uses scalars and switches instead of multipliers, for a range of specialized applications.

For signals of higher bandwidth, the multiplexing factor may be reduced to unity. We call this level 3, meaning one hardware filter per signal, with no sharing. Rich Pasco's "NCO" chip is a rather simplified example; it is a first-order integrator with no multiplexing, so the signal sample rate can be very high (500 ksps).

When the application requires sample rates higher than the word rate obtainable with the target technology, more parallelism is needed. At level 4, systems take several bits per clock cycle into a distributor, which de-interleaves and sends simple serial data to several operators in parallel; their results can later be combined into the net answer, in a partial-result combiner. Jim Cherry's "FIR" chip is intended to be used this way in the implementation of two-dimensional video filters for image understanding. The chip is one of the operators to be run in parallel; the associated distributor, control, combiner, and line memories are quite simple.

Finally, when the sample rate is higher than the achievable bit rate (level 5), another level of parallelism is needed, as in GE's "VFFT-10" system mentioned above. In this "very fast Fourier transform" system, blocks of signals are transmitted along many serial paths in parallel; signal samples are delivered on a collection of data paths by some faster signal source that can deal with the high bandwidths.

THE BUILDING BLOCKS — LOGIC DESCRIPTION

When designing the logic of many of the low-level and system-level operators, it is not important to know what technology will be used to implement those operators. Thus, this level of the design is relatively long-lived and independent of scale of integration, etc.

For most operators, such as adders, multipliers, and filters, we design logic in the style popularized by synchronous edge-triggered TTL MSI families; that is, the exact nature of the clocking is suppressed, and all memory is in D-type flip-flops, sometimes connected as parallel registers or shift registers. Some of the operators, however, such as the larger memories, cannot be adequately represented by a logic-level design; most of the design work for those parts is in the technology-dependent circuit and layout level.

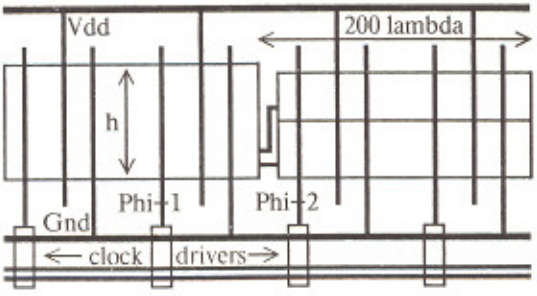
Given logic designs of this form, it is straightforward to apply a collection of techniques to translate them into circuits for the target technology. The techniques we use for NMOS are those described by Mead and Conway (1980), making heavy use of pass-transistors and dynamic storage to produce designs that are much more efficient than gate-based circuits. The logic designs therefore do not imply any particular gate structure or circuit design. Examples of CMOS implementations of multipliers and serial memories for digital filtering can be found in Ohwada, *et. al.* (1979).

There is not room here to cover details of logic designs, so we just mention some the more useful operators that have been built. They are signal combiners (adder, multiplier, and variations and combinations), point operators (scaler, overflow detector/corrector or limiter, full-wave and half-wave rectifiers, square, and sine), serial memories (Z^{-1} block, random-access serial register file, and serial ROM), and filters (configurable biquadratic sections, parallel FIR blocks, lattice structures, and multipurpose first-order sections for zero-order hold, interpolation, differentiation, sum-and-dump, oscillators, etc.)

An interesting aspect of the memory-intensive operators is the serial-parallel-serial (SPS) memory organization for both read-write and read-only memories, random or sequential access, that simultaneously minimizes area and power while keeping a high input and output bandwidth. Since memory is always accessed for full words, the parallel internals of the memory can operate on a cycle that is much slower than the serial bit clock, allowing minimum-size dynamic memory cells and slow, low power addressing and bus logic. Only the input and output shift registers and a small control PLA need to run at the bit rate.

ON VLSI LAYOUT STYLES

When actually creating layouts for a system designed through our architectural methodology, it is important to use a simple layout style that facilitates placement and interconnection of components, at many levels of the hierarchy. For NMOS, we have chosen a standard grid style with fixed width (200λ) and variable height cells. A component may occupy any number of 200-lambda wide cells, subject to chip size constraints. The grid may be thought of as a distribution network for VDD, Ground, and two clock phases, into which components and wires are dropped (possibly automatically).



This style is comparable to the polycell layout style, except that the cells being placed and interconnected are considerably larger than gates and registers, and are already specialized to our architectural methodology. This allows considerably more efficient layouts than are possible with polycells, by leaving room for flexible optimizations of logic, circuits, and layout of low-level operators. System designers need never deal with this intra-component level of optimization, but there is always the opportunity for "wizards" to design efficient new low-level components to add to the available library.

Some operators are not designed on the 200-lambda grid, for sake of their layout efficiency. For example, the serial sequential-access memory used in the "Filters" chip is designed with a custom cell array style, appropriate for the large collection of memory cells and the controller PLA.

CONCLUDING REMARKS

As JK&M predicted in 1968, serial arithmetic is alive and well as a simple and efficient way to implement digital filters and other signal processing systems in silicon. By extending their approach into an architectural methodology, and by combining this with a VLSI design and implementation methodology, we suddenly enable designers everywhere to build their own novel low-cost signal processing systems. The methodologies provide a sensible context for the accumulation of a library of component and system designs, at logic and layout levels. So start now, and design the future accordingly.

REFERENCES

Jackson, L. B., Kaiser, J. F., and McDonald, H. S. (1968), An Approach to the Implementation of Digital Filters, *IEEE Trans. on Audio and Electroacoustics* AU-16, 413-421 (*reprinted in* Rabiner and Rader, *Digital Signal Processing*, IEEE Press).

Lyon, R. F. (1976), Two's Complement Pipeline Multipliers, *IEEE Trans. on Communications* COM-24, 418-425 (*reprinted in* Salazar, *Digital Signal Computers and Processors*, IEEE Press).

Mead, C. A. and Conway, L. A. (1980), *Introduction to VLSI Systems.* Addison-Wesley, Reading, Massachusetts.

Moore, F. R. (1978), An Introduction to the Mathematics of Digital Signal Processing, *Computer Music Journal* Vol. 2, No. 1 and 2.

Moyer, A. L. (1976), An Efficient Parallel Algorithm for IIR Filters, *IEEE International Conference of Acoustics, Speech, and Signal Processing.*

Ohwada, N., Kimura, T., and Doken, M. (1979), LSI's for Digital Signal Processing, *IEEE Journal of Solid-state Circuits* SC-14, 214-220.

Powell, N. R. and Erwin, J. M. (1978), Signal Processing with Bit-serial Word-parallel Architectures, *Real-Time Signal Processing*, Proc. SPIE Vol. 154, 98-104.

Rabiner, L. R., and Gold, B. (1975), *Theory and Application of Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey.