ARCHITECTURE AND INSTRUCTION SET OF A PROGRAMMABLE LSI DIGITAL FILTER

Stephen Terepin , Paul Loewenstein

Fairchild Advanced Research & Development Laboratory

4001 Miranda Avenue, Palo Alto, CA 94304

ABSTRACT

This paper describes an LSI data stream processor that has been optimised for filtering algorithms. Its novel serial architecture achieves high throughput by performing many operations concurrently, and permits long data wordlengths in a compact NMOS implementation. The instruction set supports recursive and nonrecursive filters and a variety of nonlinear signal processing functions.

INTRODUCTION

The benefits of implementing signal processing functions digitally have been recognised for some time. Now that it is becoming feasible to integrate enough hardware on a single die to perform useful functions cheaply, we can expect digital solutions to see more widespread use.

So far, most signal processor chips have used microprogrammed computer-like architectures [1,2] that retain general purpose capabilities. High throughput has been achieved by providing hardwired multipliers and addressing units, separating the program, data and coefficient memories, and adding multiple buses to avoid bottlenecks. Chip sizes have been kept within reasonable bounds by using comparatively narrow wordlengths (around 16 bits) and having on-board ROM to store the program and coefficients.

These design choices make perfect sense for voice-band telecommunications systems. However, there are important applications where higher accuracy is required and production volumes do not justify a mask ROM.

Our goals in developing the LSI data stream processor were (a) to provide high quality filtering and related signal processing functions in a low cost device, and (b) to permit the use of external PROM or RAM program/coefficient memories without a speed penalty.

Present circuit densities are barely adequate for the amounts of memory and arithmetic logic that one would like to have in a single chip signal processor. Instead of trimming down the data wordlengths, we have compromised in a different direction - by forfeiting the ability to do general purpose processing and bit manipulation. This has allowed the use of a serial architecture that is well matched to signal processing functions and can provide operating wordlengths from 20 up to 32 bits to suit different system requirements. High throughput is obtained from the serial data path by using functional parallelism.

The architecture supports IIR filters, FIR and ladder structures, LMS equalisers, correlators, signal generators and several nonlinear functions. The device is optimised for some specific applications in instrumentation and digital audio, but is equally suitable for use in telecommunications and speech processing systems.

GENERAL DESCRIPTION

Figure 1 shows a typical system configuration. The data stream processor reads input data through its serial port and filters it by executing instructions in sequence. It updates the state variables stored in internal memory, computes the

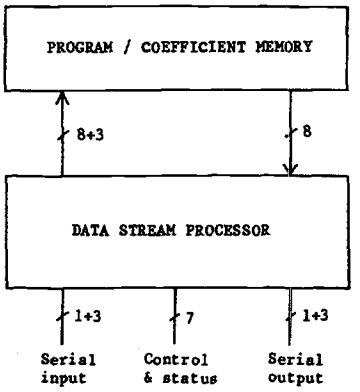

FIGURE 1 - SYSTEM CONFIGURATION

ICASSP 83, BOSTON CH1841-6/83/0000-0435 $1.00 © 1983 IEEE

new output, and then loops back to the beginning of the program to await the next sample. The maximum sampling rate is determined by the number of program steps.

Each program step implements a complete macro operation such as a two-pole resonator. The execution time varies from 1.0-1.4uS depending on the data wordlength, which may be set to either 20,24,28 or 32 bits. Several operations are carried out simultaneously during each instruction cycle - three multiplications and two additions, nonlinear processing, accessing and updating of state variables, register transfers, accumulation of partial products, data I/O, and fetching and decoding of the next instruction.

This high degree of parallelism means that the arithmetic section can be kept active all the time. Each component has a dedicated serial data path, so bus bottlenecks are eliminated. In conventional bit-parallel designs, multiplier cycles are inevitably wasted while state variables are being shuffled between the data memory and the registers along a limited number of buses.

Functional building blocks such as filter sections, peak detectors and oscillators map directly into single instructions, requiring 27 bits of microcode and 37 coefficient bits. The instruction bandwidth is low enough to permit the use of an external program/coefficient memory without a speed penalty: each 64-bit instruction frame is stored in the program memory as a block of eight bytes, and accessed sequentially. A new instruction byte is fetched every 125nS.

The program memory can be a mixture of PROM and RAM. Handshake control lines are provided to facilitate updating a coefficient RAM whilst the processor is waiting for data I/O. An external controller can change the algorithm or the filter parameters dynamically.

Operating with 20 bit precision, a single processor can execute the equivalent of 50 biquadratic sections at a 10KHz rate. Higher complexity or faster sampling rates (up to 500KHz) can be obtained by sharing the computational load among a number of devices, connected together in pipeline fashion.

ARCHITECTURE

The data path contains the components necessary to execute a two-pole filter section within a single instruction time: a sequential access memory with two data channels, and a triple multiplier/adder. A set of data switches edits the raw result from the multiplier to obtain saturation and other nonlinear effects. A register bank is used for short term storage. Data is routed between sources and sinks of data by a crossbar switching network that can be reconfigured for each instruction.

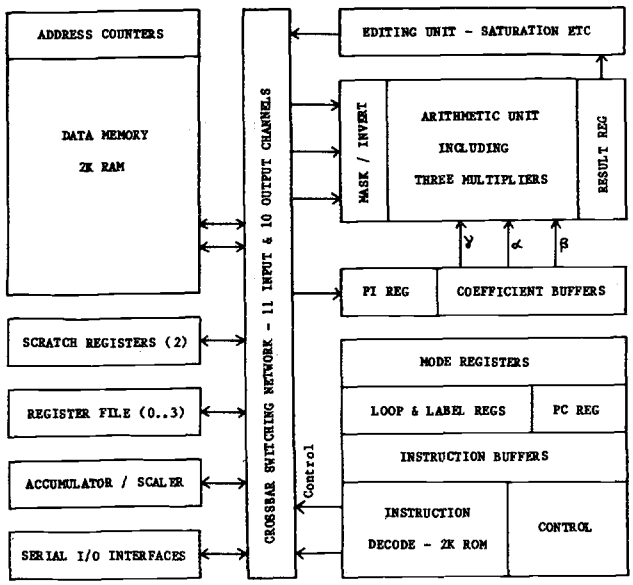

FIGURE 2 - ARCHITECTURE BLOCK DIAGRAM

The data memory is a 2K RAM array equipped with serial/parallel converters and circulating address logic. It simulates a dual channel shift register whose length is determined by a mode register. The two channels accommodate the pairs of state variables that arise in second order sections.

The serial access memory restricts us to algorithms in which the data flows in an ordered stream, such as filters and correlators. However there are a number of benefits associated with sequential addressing. Microprogram address fields and address decoding hardware are eliminated, making both the instruction bandwidth and the chip itself smaller. Also, it becomes feasible to provide a variable precision feature which is just not practical for parallel word accessible memories. The important parameter is the total number of BITS of delay.

The arithmetic unit contains three multipliers and two adders. During each instruction cycle, it forms the sum of products:

$$R = X0.Gamma + X1.Alpha + X2.Beta$$

The X's are signal data streams emerging from crossbar switch. Alpha and Beta are 16 bit coefficients representing -2...+2 and -1...+1 respectively. Gamma is normally a power of two scaling factor, but it is possible to feed signal data to the Gamma coefficient input via the Pi register, to effect the signal-by-signal multiplications required for modulation, VCO's and time varying filters.

Conditional arithmetic operations performed within the triple multiplier form the basis of many of the nonlinear functions. Signals from the control section can cause the 'X' values to be masked to zero or inverted as they enter the arithmetic unit. Rectification and sign dependent gain are achieved using these circuits alone; the more complicated nonlinearities such as peak detection require further processing which is carried out in the editing section.

Normally the multiplier rounds the final sum of products. Recursive filtering instructions, however, may invoke random switching between rounding and truncation to defeat limit cycles.

When overflows are detected, the spurious result is replaced with a saturation value. The substitution occurs whilst 'R' is being clocked out of the result register through the editing unit during the following instruction cycle. The editing unit contains a set of switches that can substitute either constant values (-1,0,+1) or signal data from the register bank. A control PLA sets the switches according to the FUNCTION code and the sign/overflow flags, to achieve either saturation or the more complex behaviour required for the other nonlinear operators.

The register bank is a group of seven variable length shift registers. There are two scratch registers that serve as working stores. User programs can manipulate the 4-word dual port register file and the accumulator. The latter is used to sum partial products in FIR and parallel IIR filters. It has six extension bits which permit accumulation of up to 64 values without the risk of overflow. There is a power-of-two scaling circuit at the output, controlled by the ASCAL mode register.

The data I/O interfaces contain double buffers which facilitate conversion between internal and external data formats and clock rates. A simple two-line handshake is used to synchronise word transfers.

The crossbar switch takes the serial outputs of the data memory, the multiplier and the registers, and routes them all simultaneously to new destinations. It contains a set of double buffered control registers, one for each output channel, which determine the connection pattern. The data paths can be configured differently for each instruction. The sequence of connection patterns determines the signal processing algorithm.

The instruction fetch section contains an 8-bit program counter and a label register that stores the branch address. The loop register is useful for controlling block computations such as DFT's and autocorrelations. Instruction decoding consists essentially of mapping a 6-bit FUNCTION code into control signals for the crossbar switch and the arithmetic unit.

Mode registers loaded during an INIT instruction define various fixed operating parameters such as the data wordlength, the I/O formats and the effective delay through the data memory. Another special instruction loads the jump-address and loop-count registers and sets up the accumulator scaling factor.

PROGRAMMING

Programming is simple: each instruction applies a FUNCTION to the selected SOURCE data to produce a resulting VALUE. The functions can be high-level operators such as POLES or PEAKDETECT, or may just perform GAIN or OFFSET adjustment. Selecting VALUE as the source for the next function gives rise to a cascade structure. More complicated interconnections can be programmed using the register file and accumulator.

Figure 3 summarises the microcode format. All but 27 bits in the 64-bit instruction frame are multiplier coefficients. Conventional bit parallel architectures typically require 3-5 times as many control bits to achieve the same amount of signal processing.

The "SRC" (source) microcode field specifies which of eight possible data sources each function should operate on. "SNL" can specify simple source nonlinearities such as full wave rectification, useful for detecting amplitudes in filter bank analysers. The remaining fields control the program counter, the register file, the accumulator and the output port, and specify the multiplier coefficients.

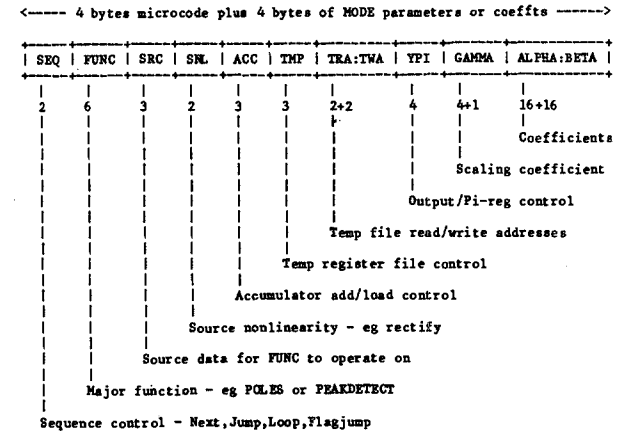

FIGURE 3 - MICROCODE / COEFFICIENT FORMAT

The FUNCTION code invokes one of 64 high level building blocks that will be familiar to analog designers:

* Linear filters (IIR/FIR/Ladder)
* Nonlinearites with state (eg peakdetect)
* Point nonlinearities (eg companders)
* Signal generators (sines and ramps)

Special instructions are included to support adaptive equalisation, Fourier analysis and autocorrelation estimates. Figure 4 summarises the processor's capabilities.

* two pole resonators with optional +1/-1 zeroes
  direct/transposed/coupled biquadratic sections
  zero pair, real pole & zero, pole with gain term
  lowpass poles with rectify/square at the input
  cascade or branched (parallel form) configurations
  normalised form ladder filters with optional taps

  FIR filters, interpolators, decimators allowing sample rate changes by arbitrary integer ratios

* full and half wave rectification, sign restoration, conditional gain/offset, limiting, centre clipping and infinite clipping, piecewise linear companding, windowing, minimum or maximum value selection, near zero detection, polynomial functions

* peak detect function, comparator with hysteresis, zero crossing detectors, monostable pulse generators, sampling function, resettable integrators

* sinusoidal and quadrature oscillators and VCO's, noise sources, ramp & pulse & sawtooth generators

  amplitude, frequency or phase modulation, Fourier transforms using Goertzels method, estimation of autocorrelations for LPC analysis adaptive equalisation using the LMS algorithm

FIGURE 4 - SUMMARY OF FUNCTION REPERTOIRE

Many of the building blocks require just one instruction cycle, but filtering functions that require more than three multiplications may take several cycles. Figure 5 indicates how two primitive instructions are combined to realise a direct form biquadratic section. During the first execution cycle the data memory is accessed and the crossbar switch makes connections as shown, to exercise the poles. The next instruction generates the final output by combining the new VALUE (resonator output) with the delayed state variables that were saved in two scratch registers. The new state variables (V,B) are copied into registers B & C temporarily when cycle #2 executes, and will be written into the memory when it is next accessed.

Provided that the serial memory is set up correctly the updated variables will appear again at just the right time, when the 'poles' function is executed on the next loop around the program.

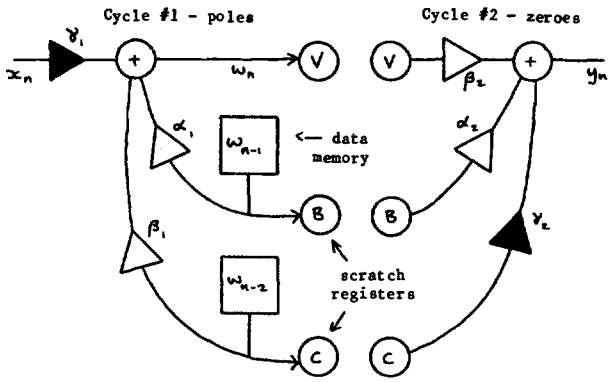

FIGURE 5 - DATA FLOW IN A BIQUADRATIC SECTION

Complex signal processing functions can be programmed by combining building blocks in an appropriate sequence, using the register file to store important node values and the accumulator to sum the outputs of parallel branches. The control section dovetails instructions together, and arranges for state variables to be accessed and updated when necessary. With these housekeeping tasks taken care of, it is comparatively straightforward both to write microprograms, and to write a compiler that translates a user's block diagram description into optimum code. A signal processing compiler is being developed as part of the software support package.

SUMMARY

We have described an LSI signal processor with a novel serial architecture that is optimised for digital filtering. Making the instruction set specific rather than general has allowed us to take advantage of the constraints inherent in filtering algorithms and use large primitives such as a triple multiplier and a circulating data memory. The result is a compact NMOS circuit which need not compromise on issues such as wordlength and the provision of external program storage. As figure 4 indicates, using large primitives need not unduly restrict flexibility [2] in a filtering context.

REFERENCES

[1] Broderson, "VLSI for signal processing", Trends & Perspectives in Signal Processing, Jan 81

[2] Thompson & Tewksbury, "LSI signal processor architecture for telecommunications applications", IEEE Trans ASSP 30.4, Aug 82