Multiport register file simplifies and speeds digital signal processing

A byte-wide eight-register chip with five independent ports overcomes bit-slice barriers and puts digital data into registers simultaneously with processing operations.

Bit-slice processors have achieved broad acceptance in digital signal processing. However, inflexibility and a small memory-to-register bandwidth limit their effectiveness for many applications.

For example, the transfer of data between registers and memory cannot often occur simultaneously with ALU processing.

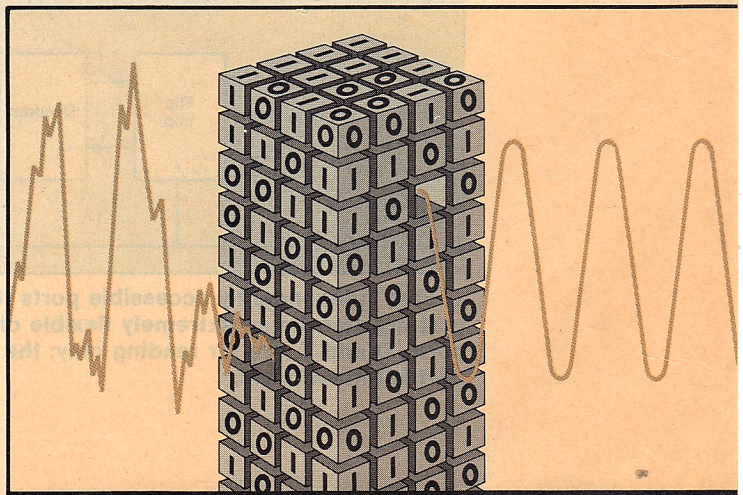
These limitations can be overcome with a multiport CMOS register file IC that not only increases signal-processing bandwidths, but also adds a new dimension of flexibility.

The file, the LFR08, contains eight registers of eight bits each and is easily expandable to more registers and wider words. The device has five independent parallel ports, each of which may be individually addressed to access any of the eight internal registers on a given clock cycle.

Two of the five ports, B and C, are write-only memory ports. Two, D and E, are read-only ports; and the fifth, A, is bidirectional. With so many ports, microprogrammable digital signal-processing systems take on new flexibility.

A closer look

Each of the five parallel ports has an 8-bit data bus, three address lines, and one or two control lines (Fig. 1). All address and control-line inputs are latched on the rising edge of the clock signal. During the following clock period, the addressed data is available at the read ports. Input data is latched on the rising



SIGNAL PROCESSING

Joel H. Dedrick, Logic Devices Inc.

Joel H. Dedrick, director of product development, joined Logic Devices in Sunnyvale, Calif., in January. Earlier he worked with Texas Instruments, where he helped develop CMOS LSI signal processors and other digital signal-processing systems for military use. He earned his BSEE from the University of Nebraska and his MSEE from Southern Methodist University.

Signal Processing: Multiport register chip

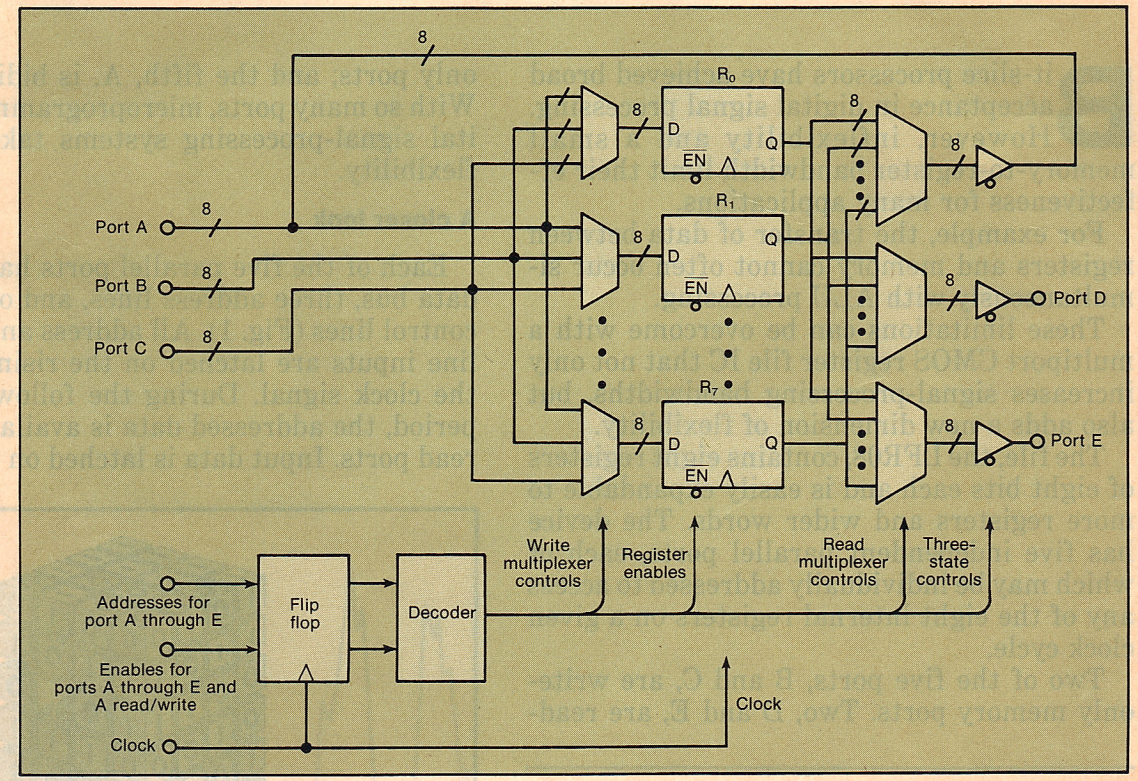
edge following application of the address. This timing allows for efficient use of the LRF08 in horizontally microcoded architectures with pipelining.

A logic low on the enable lines of ports D and E enables the corresponding three-state outputs, which in turn allows these ports to produce the contents of the register selected by address lines 0 through 2.

Input ports B and C are enabled in the same way. A logic low on their enable lines allows the selection of the target register for the write operation on the next rising clock signal after addressing. The addressing is similarly performed by B and C port address lines 0 through 2.

Because it is bidirectional, port A is served not only by address lines 0 through 2 and an enable line, but also a read/write line. The last line is latched with all other control lines to determine when port A will be used for reading or writing. When the read/write line is high, the A port is in the read mode, and its enable line functions as a three-state control line. When the read/write line goes low, the A port is in the write mode. Here its enable line functions the same as the port B and C enable lines for write clock cycles.

The five independent ports of the LRF08 also allow data to be transferred between the register file and external system memory. The transfer occurs while arithmetic operations



1. Five independently accessible ports that address eight 8-bit registers make the LRF08 multiport register file an extremely flexible chip in digital signal-processing circuits. The two output ports, D and E, are for reading only; the two input ports B and C, are for writing only; and the A port is bidirectional.

are being performed on data in the file and while the results are being returned to the file. If a register addressed for reading is the target of a write operation during the same clock period, the data at the output port will be the register contents prior to the write operation. Since the user can write to and read from any register during the same clock cycle, any of the eight registers can be used as accumulators for arithmetic operations.

With this five-port, eight-register flexibility, a word-slice approach to digital signal processing is available. It is superior to the traditional bit-slice approach, embodied in such parts as those in the Am2900 family, and a closer examination of the two architectures will show why.

In the bit-slice architecture of the Am2903, for example, each chip has a slice of the ALU and register file memory (Fig. 2a). The registers are in a three-port RAM—two read ports and one write. The read ports supply operands to the ALU, and the ALU feeds back results to the write port for storage.

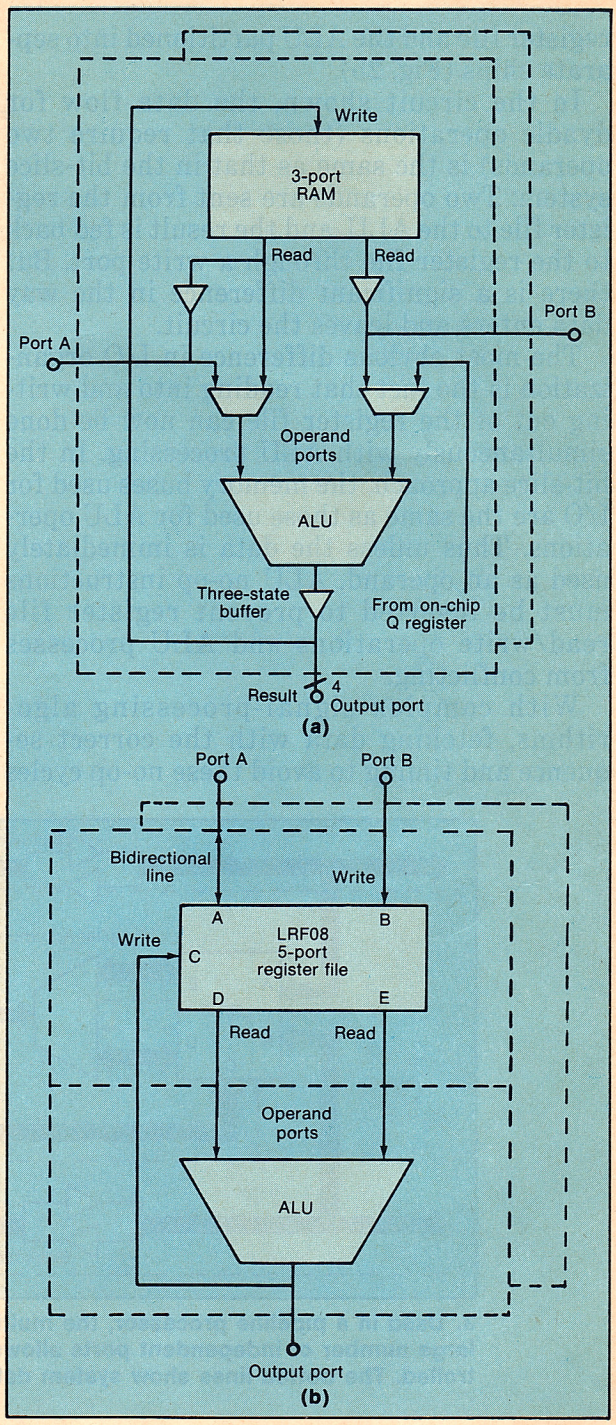
Writing external data to the registers of a bit-slice system can be performed in two ways. The ALU's output port can be disabled by use of an off-chip, three-state enable circuit. This allows the user to write in external data to the RAM through the feedback line.

Alternatively, off-chip multiplexers can be used to allow external data into the system through the two operand ports, A and B. Data is passed through the ALU and stored through the feedback line in the three-port RAM.

In many cases, the ALU's operation must be suspended, with resulting computational delays, while the external data is brought in.

As for reading data from a bit-slice system, it can be done at the output of the ALU or at the ALU's operand ports. In the latter case, ALU results are read out at the A and B ports simultaneously with ALU operations. This operation is useful for such applications as address generation for vector processing. Here, the address is read to the A port from the RAM and is simultaneously incremented by the ALU. The result is stored in the RAM in place of the old address.

The word-slice system, in contrast, is divided along functional boundaries, with the



2. The bit-slice architecture of processors like the Am2903 (a) limits the memory-to-ALU bandwidth and leads to delays when outside data is introduced, because the ALU's operation may have to be suspended while the external data is written into the three-port RAM. Using the LRF08 allows complete overlap of an ALU operand and result transfers with external read/write of the register set (b).

Signal Processing: Multiport register chip

register file and the ALU partitioned into separate chips (Fig. 2b).

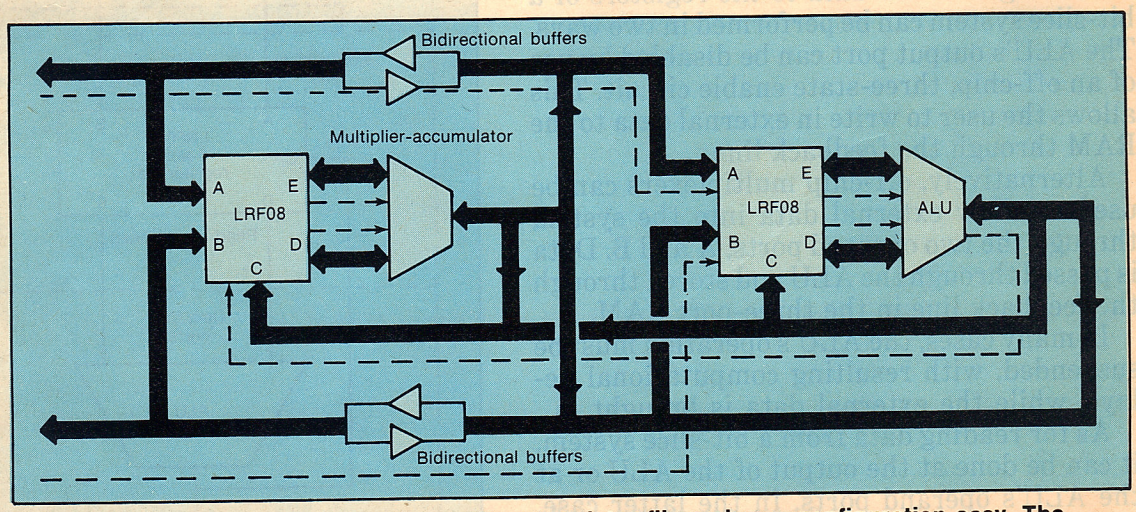
In the circuit shown, the data flow for dyadic operations (those that require two operands) is the same as that in the bit-slice system: Two operands are sent from the register file to the ALU, and the result is fed back to the register file through a write port. But there is a significant difference in the way data enters and leaves the circuit.

The most obvious difference in I/O organization is the fact that reading into and writing out of the register file can now be done simultaneously with ALU processing. In the bit-slice approach, the memory buses used for I/O are the same as those used for ALU operations. Thus unless the data is immediately used as an operand, ALU no-op instructions must be inserted to prevent register file read/write operations and ALU processes from conflicting.

With complex signal-processing algorithms, fetching data with the correct sequence and timing to avoid these no-op cycles is difficult, since only a limited number of independent address generators are available. True, the no-op states can be held down by the use of wide-word or complex (real plus imaginary) memory organizations, but then additional circuitry is needed to store the data temporarily, perhaps even rearrange it, for the ALU. With the completely independent I/O ports in the LRF08, all these problems are eliminated.

A more subtle advantage of the word-slice approach results from timing considerations in the design of the processor-memory interface. When data is multiplexed into the ALU at the operand ports in the bit-slice approach, provision must be made for sufficient setup time so that data can propagate through the ALU before the next rising edge of the clock signal.

Depending on the complexity of the instruction being executed, this setup time can range from 100 ns upward. Since many digital signal-processing system clock periods are 150 ns or less, a staging register is required be-



3. Used in a pipeline processor, the multiport register file makes reconfiguration easy. The large number of independent ports allows data routing through the system to be software controlled. The dotted lines show system data flow for a finite-impulse-response filter algorithm.

Signal Processing: Multiport register chip

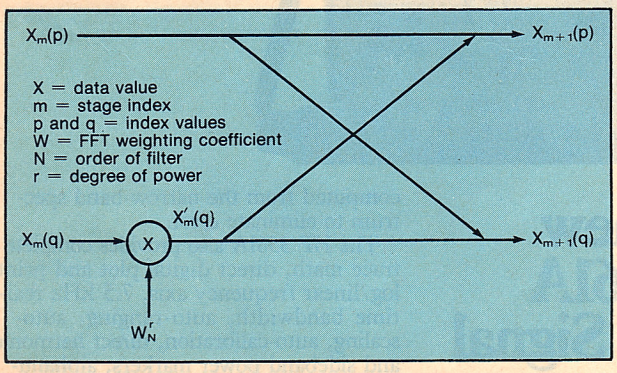
tween the external system memory and the ALU. The LRF08 essentially performs the function of such a staging register, reducing effective memory data setup time to 15 ns.

A reconfigurable pipeline processor

Consider the utility of the LRF08 in a reconfigurable pipeline processor, where two of the multiport register file ICs are paired with an ALU and a multiplier-accumulator (Fig. 3). The outputs of the ALU and multiplier-accumulator are fed back to their associated register files, and they may also be selectively gated onto auxiliary data buses running the length of the pipeline.

Bidirectional buffers on the data buses make them reconfigurable. System memory, which is accessed through ports A and B of each register file, can also be reconfigured for greater I/O bandwidth by the addition of memory ports to the buses. Likewise, the number of arithmetic elements, like the ALU and multiplier-accumulator, can be increased by a simple extension of each bus.

A typical application of the circuit shown is



4. In a radix-two decimation-in-time butterfly, the basic unit of the fast Fourier transform, four real multiplications and six real additions take place.

the finite-impulse-response (FIR) filter algorithm, frequently encountered in digital signal processing. For a nonrecursive Nth-order filter algorithm, each output sample consists of the sum of the past N points, each weighted by the appropriate filter coefficient. This can be expressed as follows:

$$Y_n = \sum_{k=0}^{N-1} h_k(x_{n-k})$$

where:

Y_n = the nth output sample
n = the data index
k = the coefficient index
N = order of filter
h_k = the kth coefficient
x = the input sample
x_{n-k} = input sample delayed by k sample periods

A key feature of the FIR filter algorithm is its linear-phase transfer characteristic. A necessary and sufficient condition for linear phase is that the coefficients of the filter be symmetric or antisymmetric about the center of the impulse response. Thus, for a filter with N coefficients, where the first coefficient has the value h_0, coefficient h_{N-1} must equal $\pm h_0$, and so on, from h_1 through $h_{(N/2)-1}$, with N being an even number. Such a filter can be expressed as:

$$Y_n = \sum_{k=0}^{(N/2)-1} h_k(x_{n-k} + x_{n-N+k+1})$$

The symmetry of the filter coefficients offers the possibility of computational shortcuts. By adding the input points corresponding to coefficients that are equal prior to weighting, the designer can reduce the total number of multiplications required for each output point to N/2. Although this reduction comes at the expense of an extra addition for each pair of input points, it offers the advantage of a 2:1 adder-to-multiplier ratio, typical of many signal-processor architectures.

The flow of data to implement the FIR filter is shown by dotted lines in Fig. 3. It is assumed that the filter coefficients are stored in the multiplier register file prior to entering the kernel (the smallest processing loop or itera-

Signal Processing: Multiport register chip

tion). Data enters the system from the memory ports and is placed in the ALU register file.

Points corresponding to a single coefficient are assumed to be read from memory simultaneously. If memory limitations preclude simultaneous readings, the designer can obtain an equivalent throughput by doubling the kernel length from one to two clock periods and processing input points four at a time. In this way adjacent points can be fetched from memory by a single address generator and a double-width memory organization.

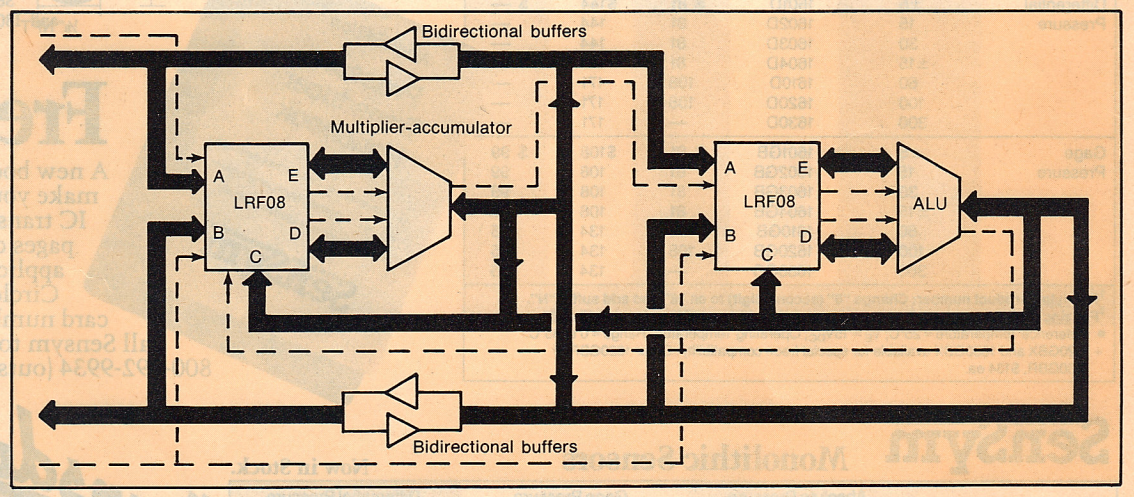
The addition of the two input values is accomplished in the ALU, and the result is fed back to the multiplier-accumulator register file. The file performs weighting and accumulation and holds the result until all pertinent input points have been processed.

Taking on the FFT

Yet another illustration of the flexibility of the LRF08 is its use for calculating fast Fourier transforms, a type of algorithm encountered frequently in digital signal processing. The fundamental unit of the FFT, the butterfly, is of interest. One of several common forms is the radix two, decimation-in-time butterfly (Fig. 4).

For complex (real plus imaginary) input data, the decimation-in-time butterfly requires four real multiplications and six real additions. The multiplications and two of the additions are used in the complex multiplication which phase-rotates the lower input, $x_m(q)$. The remaining four additions combine the upper input, $x_m(p)$, with the weighted lower input to produce the pair of complex output points, $x_{m+1}(p)$ and $x_{m+1}(q)$.

For this example, the weighting coefficients, W, are assumed to be stored in the multiplier-accumulator register file of Fig. 5 at kernel entry. In practice, the kernel is usually four or eight butterflies long. It takes advantage of the extensive symmetry in the weighting coefficients ($\pm 90°$ and $\pm 180°$ rotations, and reflections about the 45° axes). Thus, given one coefficient, three others may



5. The LRF08 multiport register file allows reconfiguration of the pipeline architecture for a different algorithm, in this case, the complex decimation-in-time FFT butterfly (Fig. 4).

Signal Processing: Multiport register chip

be calculated by sign changes in the real and imaginary parts of the complex input data. In this way the butterfly can be computed efficiently without the need for a memory dedicated to storage of the complex weights.

The data flow of the decimation-in-time butterfly in the figure is heavily pipelined. It has a net throughput of one complex output pair for every four clock pulses. The total pipeline delay is 12 clock periods. The table (below) shows a symbolic listing of the various data movements and arithmetic operations.

During the first two clocks of every four-clock iteration, the real and imaginary parts of $x_m(p)$ are read from memory to the multiplier-accumulator register file. This transfer of data occurs through the lower I/O port. The last two clocks are used for reading the real and imaginary parts of $x_m(q)$ from memory to the ALU register file. The paths of the data flowing from memory to the register are represented by dotted lines in the figure.

The second column in the table shows the complex multiplication of the weighting coefficient and the $x_m(q)$ input from the previous iteration. Results are passed to the ALU register file, where the additions needed to produce the real parts of the output values occur during the latter half of the iteration. This transfer of data is accomplished over the upper bus in the figure, a bus isolated from the memory port by bidirectional buffers.

The last column in the table shows the final iteration, when the imaginary portions of the results are calculated. These results are returned to the multiplier register file through the C port bus. Finally, the results stored in the multiplier file are sent out to the upper memory bus, one word during each of the four clocks of the iteration.□

How useful?	Circle
Immediate design application	556
Within the next year	557
Not applicable	558

Symbolic listing of operations for an FFT decimation-in-time butterfly iteration

Clock cycle	1	2	3
1	Multiplier $Re[X_m(p)] \rightarrow$ register file	$Re[X_m(q)] + Re[W_n^r] \rightarrow$ accumulator	$Re[X_{m+1}(t)] \rightarrow$ output $Im[X_m(s)] + Im[X'_m(t)] \rightarrow$ multiplier register file
2	Multiplier $Im[X_m(p)] \rightarrow$ register file	Accumulator $- Im[X_m(s)]^*$ ALU $Im[W_m^r] \rightarrow$ register file	$Im[X_{m+1}(t)] \rightarrow$ output $Im[X_m(s)] - Im[X'_m(t)] \rightarrow$ multiplier register file
3	ALU $Re[X_m(q)] \rightarrow$ register file	$Re[X_m(s)^* Im[W_n^r] \rightarrow$ accumulator $Re[X_m(r)] + Re[X'_m(s)] \rightarrow$ multiplier register file	$Re[X_{m+1}(U)] \rightarrow$ output
4	ALU $Im[X_m(q)] \rightarrow$ register file	Accumulator $+ Im[X_m(s)]^*$ $Re[W_n^r] \rightarrow$ ALU register file $Re[X_m(r)] - Re[X'_m(s)] \rightarrow$ multiplier register file	$Im[X_{m+1}(U)] \rightarrow$ output