Implement digital filters efficiently with a specially configured computer

Digital Filters: Part One

In the same way that digital computers have left analog computers far behind, the technique of digital filtering promises to advance far ahead of analog filtering. An important step in the advance is computer architecture that optimizes digital filtering. In this article, author Karwoski takes us through his design of such an efficient signal-processing computer. Part Two will show how to implement specific filters with this computer.

To get the best results with digital filters, design a specialized computer specifically tailored to the task. Although general-purpose computers can do the job, generally they can't work fast enough to operate in real time. A dedicated unit, sequenced by firmware, can operate in real time, but it lacks the flexibility of the specialized computer's software control. Not only that, the specialized design can operate just as fast and also perform real-time tasks.

But no matter how it's implemented, the digital filter offers many advantages over analog units:

- *Freedom from parameter drift*—temperature changes and aging don't affect digital components as they do analog ones.
- *High repeatability*—component tolerances aren't a consideration as in analog units.
- *Vastly better performance*—much sharper cutoff rates and exact linear phase response can be attained with digital filters.
- *Filter time-sharing is easy*—many channels readily can share the same filter when digitally implemented.

And the main performance disadvantage—so-called quantum noise that results from sampling and the finite length of digital words—rapidly improves as computer word lengths get longer and a/d and d/a converters get better. Also, advances in computer architecture will lower costs and speed performance.

Richard J. Karwoski, TRW LSI Products, P.O. Box 1125, Redondo Beach, CA 90278.

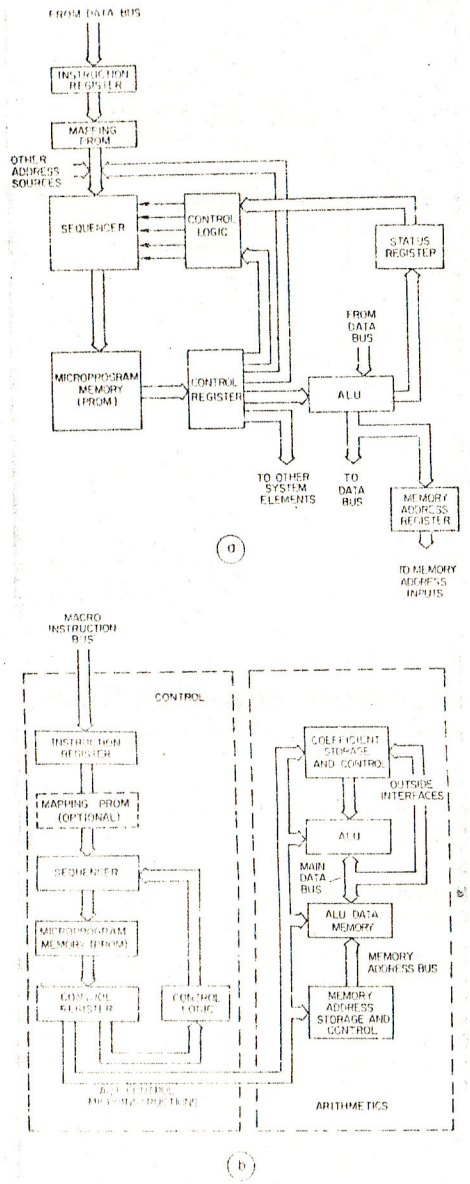
A specialized computer is simpler

Fortunately, for even the most complex digital filter, the computer doesn't need all the complications and flexibility of a general-purpose machine. For instance, data and instruction memories can be kept separate. As a result, the computer-control section of the processor can be isolated from the main arithmetic-logic-unit (ALU) data ports, because then the controller doesn't have to borrow the ALU for computing effective addresses for instruction or data-acquisition purposes. The upshot: A great amount of the considerable time that such inner computer activity consumes is saved.

Not only does separation of data and control functions save time, but it also allows computer-hardware simplification. Then, the ALU can be tailored specifically for signal processing without regard for possible controller requirements. And the separate control and address functions can be handled efficiently by a small amount of specialized hardware.

In addition, the software need place little emphasis on elementary micro operations such as load, shift, transfer, add, subtract or multiply—an inefficient way to implement digital filters. Even the most complex filter configuration can be "described" with relatively few high-level instructions, where the detailed operations, such as loading, adding and multiplying, are implied in a single statement. For example, a very often repeated operation in digital-filter synthesis involves reading data from memory, multiplying them by another number, adding them to a previous result stored in some working register and restoring the result in the same register. This operation, and even more complicated sequences, can be directed by a single code, within a single clock cycle in a specialized computer.

Fig. 1 shows in simplified form the differences between a general-purpose microprocessor and a fast, specialized microprogrammed digital-signal processor. Obvious is the separation of data, address and instruction buses in the specialized configuration in contrast to the general-purpose structure. In the specialized machine, all computational operations occur under the unconditional direction of the controller—practically no decisionmaking depends on ALU outputs. A general-purpose microprocessor, however, would lose much of its capability without ALU feedback.



1. **A general-purpose computer's processor** (a) is more complicated and flexible than needed for digital-filtering work (b). For filtering, the control and data functions can be isolated from each other, which allows speeding up the process for real-time applications.

Table 1. ALU data/instruction memories

- *Macro RAM:* Contains the over-all macroinstructions defining ALU operations. The instructions are stored sequentially in memory just as they are to be executed by the machine.
- *Control RAM:* Contains words that define such control functions as the scratch-pad location to be written or read, and the output/input ports to be activated.
- *Coefficient RAM:* Contains the sequence of multiplier coefficients needed for the implemented filter.
- *Offset RAM:* Contains the offset words required to generate the proper delays and memory-partitioning information to accommodate cascaded-filter configurations in time-shared applications.

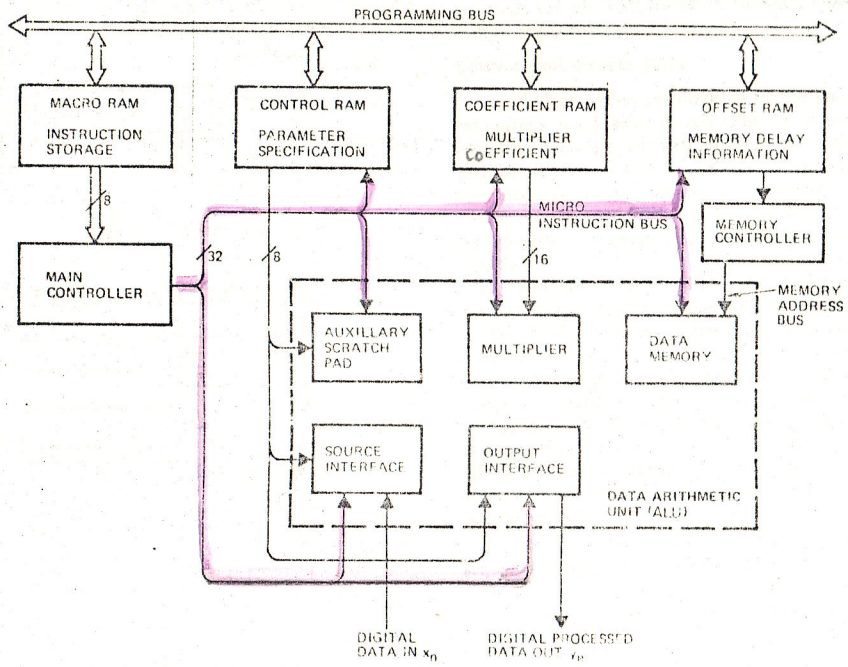
To implement a digital filter, most of the input instructions to a computer include a multiplier coefficient, memory-delay information, general-purpose scratch-pad locations, I/O control codes and over-all ALU instructions. In a computer optimized for implementing digital filters, these codes can be stored in four separate memories (see Table 1) allocated specifically for them (Fig. 2).

As a result, data can be fetched from each memory and directed to their proper destinations simultaneously and separately with minimal independent control logic, and in substantially less time than that required by a general-purpose machine.

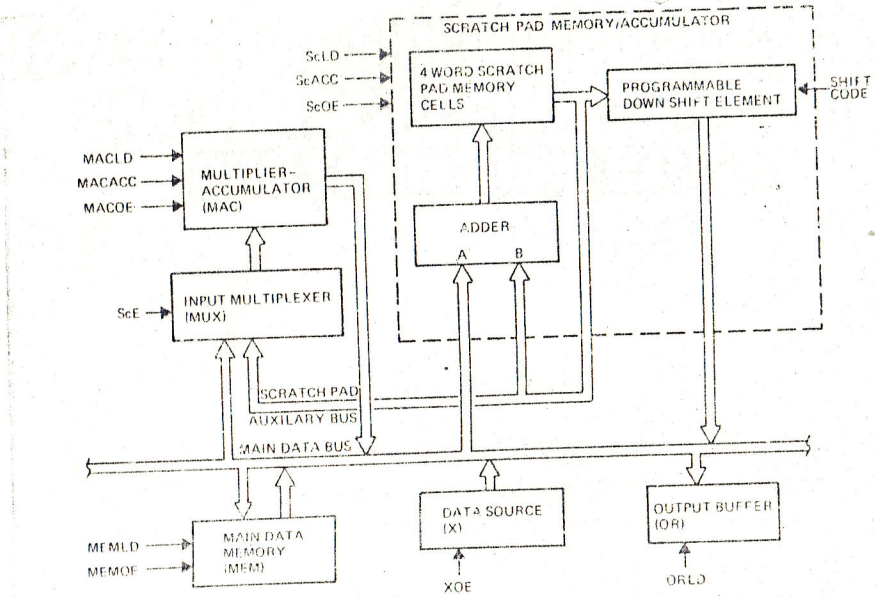
In a general-purpose microprocessor, such data must be stored in the main memory and processed sequentially through a central processing unit. Parallel processing in a specialized machine not only saves time, but also makes programming very simple: each item can be independently specified because it is processed by individual control circuitry and carried out during a single machine-clock interval. In a single interval, the sequencer in the main controller enables the specified ALU elements—multiplier, data memory, scratch-pad—via the microinstruction bus (Fig. 2). At the same time, the multiplier coefficient, memory address, control information and microinstruction also arrive at their destinations. By the next clock edge, the computed results of all these data from the ALU are ready to be strobed into the buffer register of a d/a output device.

Universal ALU not needed

Confronted with the myriad of possible digital filter configurations, a computer designer's first compulsion might be to create a machine with a completely



2. **In a computer optimized for filtering,** the data that define the filter are stored in four separate memories, which then direct the computer's activities. Local rapid-access storage is in a scratch-pad memory.



3. **The ALU for an efficient digital-filter computer** needs less hardware than for a general-purpose computer. A local scratch-pad memory and auxiliary-bus system greatly speeds performance by allowing parallel operation of the adder with the very busy multiplier/accumulator, which can be a 64-pin, TDC 1010J (TRW) DIP.

universal ALU bus structure, in which each ALU element has direct and independent access to the output of any other ALU element. But experience shows that such flexibility isn't necessary for the efficient execution of even the most complicated digital-filter algorithm.

With much less hardware, a specialized ALU (Fig. 3) is much more efficient for the execution of digital filters (Table 2). Data channels in and out of this ALU via input and output storage elements—Data Source (X) and Output Buffer (OR)—can be a/d and d/a interfaces or additional memory. Local rapid-access storage is in a scratch-pad memory (SC), out "bulk" storage is in the four ALU main-data/instruction memory units, one of which provides the data delays needed to implement the digital filter (Table 1).

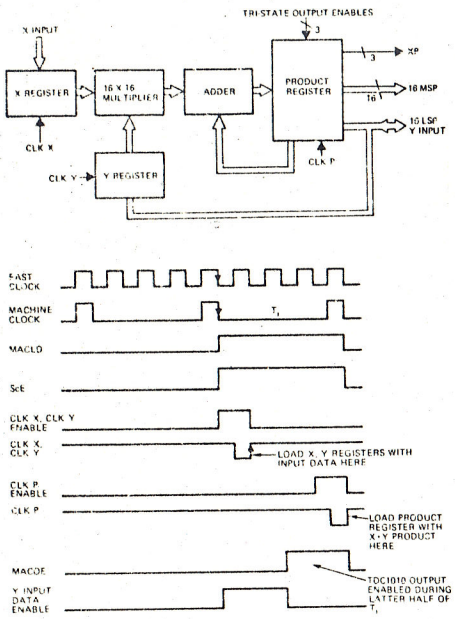
An extended-range adder directly links the scratch memory so that data can be accumulated efficiently, especially for a specific important class of filters (see box). This extra accumulator arrangement frees the very busy multiplier/accumulator (MAC) from the need to perform simple group-addition tasks and speeds overall performance of the ALU. A combinatorial shifter connected to the scratch-pad output then renders the accumulated data compatible with the main bus. Note that the scratch-pad-memory output also drives an auxiliary bus into the MAC via a multiplexer (MUX).

Indeed, the MAC is the heart of an ALU: It does most of the data computations. A block and timing diagram of the MAC (Fig. 4) shows a loading and outputting enabling sequence during a typical computer cycle. Consider T_i to be a machine cycle during which programmed instructions call for a multiplication (or multiply/accumulate) and for a previous product, stored in the product register, to be transferred to the main bus. During this interval, input registers for the new X and Y operands are loaded. Since a multiply/accumulate time is short, a single machine-clock interval can be divided into loading, multiplying, adding and output storage intervals—a complete sequence of microsteps in a single microcycle.

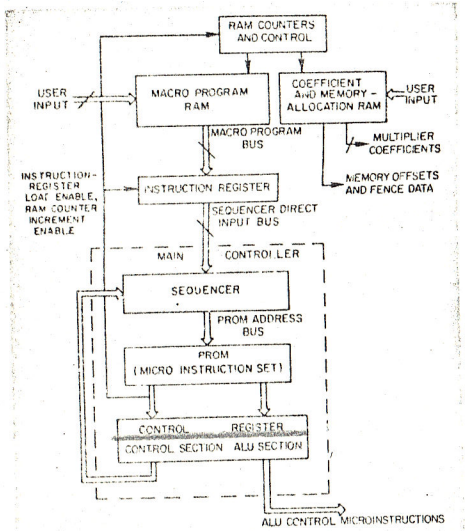
Unfortunately, during an output-storage cycle, the I/O bus to the main memory is tied up and can't provide data for other multiplications. However, such data, if stored in the scratch pad during a prior cycle, can be used via the auxiliary bus. The MAC otherwise would be idle during this time. With data running on both buses, the ALU thereby eliminates one microstep when implementing many filter types—a substantial savings in machine time, especially when just four to

Table 2. ALU components/nomenclature

- **Scratch pad memory/accumulator (SC)**
  - *Scratch pad Load* (SCLD)—Clock enable to load the memory.
  - *Scratch pad Read Address, Scratch Pad Write Address* (SRA, SWA)—Memory address directs data in and out of memory; addresses may be tied together or remain separate.
  - *Scratch Accumulate* (SCACC)—Activates scratch adder to perform accumulation. Data on the bus are added to data in a memory location specified by SRA and stored in memory location specified by SWA. Otherwise, only simple data write or read can be performed.
  - *Scratch Output Enable* (SCOE)—Enables shift of data onto the main bus.
  - *Shift Code (s)*—n-bit code specifies the number of places the scratch pad word is shifted down, in this way doing division by 2^n.
- **Main memory**
  - *Memory Output Enable* (MEMOE)—Enables memory data onto the main bus.
  - *Memory Load* (MEMLD)—Clock enables loading data into memory.
  - Memory Address (MEMA)—n-bit code specifies memory-word locations.
- **Multiplier/Accumulator (MAC)**
  - *Multiplier/Accumulator Load* (MACLD)—Clock enables loading into X, Y and accumulator registers.
  - *Multiply/Accumulate* (MACACC)—Directs an accumulate operation within the MAC after multiplication.
  - *Multiplier Output Enable* (MACOE)—Allows the multiplier product onto the main data bus.
- **Multiplexer input (MUX)**
  - *Scratch-Pad Enable* (SCE)—Enables data into the MAC from the scratch memory via the auxiliary bus.
- **Input source (X)**
  - *Source Output Enable* (XOE)—Enables X data onto the main bus.
- **Output buffer (OR)**
  - *Output Load* (ORLD)—Clock enables loading this element (register or buffer memory).



4. The multiplier/accumulator is the heart of the filter-computer's ALU. Since it does most of the computing, its high-speed capability contributes greatly to the system's over-all speed: just 120 ns for a 16 × 16 operation.



5. A few macroinstructions from an outside source are converted to a sequence of many microstep instructions in the computer's control unit to direct the detailed steps in the ALU needed to implement a digital filter.

six steps are all that are often needed to perform a complete algorithm.

Control unit directs ALUs

All ALU activities are directed by the specialized computer's main control unit (Fig. 5). It primarily receives user-supplied instructions, interprets them and then directs an ALU to perform the corresponding computations. In general-purpose computer systems, such controllers tend to be very complex; digital-filter configurations, however, can be much simpler, yet flexible.

The data for both the macro and ALU memories enter via the programming bus from an outside source (such as a μP). The main controller includes a software interpreter to direct data to each memory according to the filter program. With the memories loaded, the macro instructions trigger "canned" filter routines stored in a PROM, which delivers the required sequences of micro codes to the ALU. (Of course, the macroinstruction system also can program simple, one-cycle instructions such as executing delays and routing data.)

For example, binary codes that correspond to a sequence of macroinstructions might enter as follows:

Step No.	Macro-instruction	Binary code (hypothetical)
00	NOP	0 0 0 0 0 0 0 0
01	MemMult	0 0 0 0 0 1 1 0
02	XWM/XMAC	0 0 0 1 0 1 0 0
03	MAC ST01	0 0 1 0 0 1 0 1

The binary codes read sequentially from memory under the direction of a counter, enter an instruction register and provide a macroinstruction through a sequencer (when not busy) to address a PROM that contains the desired ALU microcode.

The sequencer cuts corners

The sequencer is important when implementing with many identical sections, or for time-sharing individual filters. When a particular microinstruction enters the control register (and is therefore being executed), a control portion of its code directs the sequencer's mode: Single-cycle macroinstructions allow the sequencer to pass the next address through to the PROM, but the starting instruction (or intermediate instructions) for a "canned" filter sequence directs the sequencer to generate a succession of PROM addresses until the final control-register microinstruction calls for the next address from the instruction register.

This simple control unit and the specialized ALU architecture can perform fast implementation of virtually any filter structure. But some applications may not need all this flexibility and speed. In that case, elimination of the scratch-pad memory (and auxiliary bus) can considerably simplify both the hardware and instruction set. Also for batch-processing, although a buffer memory for the input would be needed, the scratch-pad would not be necessary. ■■