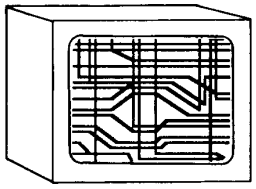


The TMS34010: An Embedded Microprocessor

Aimed at graphics systems, this 32-bit processor's large address reach, bit-field processing, and DRAM interface make it suitable for many other embedded processing applications.

Karl M. Guttag, Thomas M. Albers, Michael D. Asal, and Kevin G. Rose

Texas Instruments

The TMS34010 is a high-performance 32-bit microprocessor with special instructions and hardware for handling the bit-field data and address manipulations often associated with computer graphics. With integrated control and addressing for dynamic random access memory (DRAM), it supports a lower system cost than would normally be associated with a 32-bit microprocessor. Internal features such as an instruction cache, thirty-one 32-bit registers, and an independent memory control unit maintain a high degree of parallelism while efficiently utilizing lower cost DRAM.

Embedded processing for graphics was the target application for the 34010 from its inception. This led to a set of cost and performance decisions that are applicable to a wide variety of systems in addition to graphics systems.

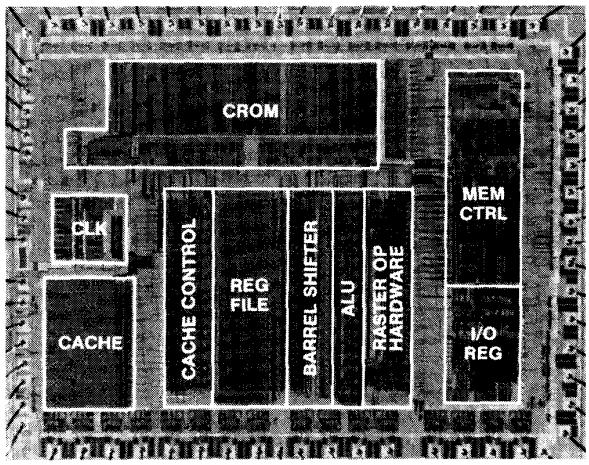
The chip contains over 180,000 transistors fabricated in 1.8-μm CMOS, consuming approximately one-half watt while executing in excess of six million instructions per second. In addition to a full 32-bit microprocessor core, the 34010 contains on-chip video random access memory (VRAM) display support, DRAM control, and a host interface.

History

Embedded microprocessors grew out of the need in the late 1960's for more advanced calculators.[1] Calculator designers recognized that as the variety and complexity of applications for calculators grew, programmable rather than fixed-function processors would be advantageous. Furthermore, engineers at Texas Instruments and Intel both recognized that these processors, once designed, could be used for much more than just calculators. The same chips could be used as general-purpose controllers, replacing gears, tubes, and relays with solid-state control. Texas Instruments initially pursued the single-chip microcomputer with its TMS1000 line, while Intel focused on the multichip microprocessor market with the 4004 and then the 8008.

Similarly, in the late 1970's and early 1980's designers began studying the processing needs of more advanced applications (such as digital signal processing, local area networks, and graphics). The comprehensive nature of these applications led designers at our company to the conclusion that even application-specific processors should be generally pro-

072-1732/88/0600-0039$01.00 © 1988 IEEE



Photomicrograph of the TMS34010 embedded microprocessor.

grammable. The design decision to support a completely general-purpose instruction set on an application-specific device means that the device will be well suited for many new application areas as well.

In display graphics, for example, the demands on graphics subsystems are growing rapidly. As desktop systems advance, a wide range of both graphic and nongraphic functions are being required of the graphics subsystem. The advent of graphical user interfaces for bitmapped graphics systems and the growing complexity of the interfaces between the system processor and its graphics subsystem dictate that the embedded graphics device should be programmable.

With advancements in processor technology, embedded control has expanded considerably. As many applications needed and could afford more processing power, 8-bit and, later, 16-bit CPUs came into use for embedded control applications. Today, many embedded applications are migrating to 32-bit microprocessors for their speed and advanced feature sets.

A 32-bit microprocessor offers significant advantages over older 8-bit and 16-bit chips, both in speed and ease of use. It has a large linear address reach for larger program and data requirements. The linear approach simplifies address management and tool requirements, and the larger address reach extends the application limits of the device. Additionally, 32-bit processors generally have much better bit-field processing capabilities than the older processors. Many 32-bit machines have internal instruction and/or data caches for faster program execution.

With the growing size and complexity of applications, the need for high-level-language (HLL) support has increased. However, for speed-critical portions of an application, the need for strong assembly language support for embedded control systems still exists. Thus, a processor that can be programmed easily at both levels is beneficial.

Embedded processors

The term *embedded processor* covers a wide range of processors and their applications. The term *embedded* implies that the user is not aware of the presence of the processor. That is, the user does not directly interact with or program the processor, but rather the processor provides a convenient method of implementing the control function. Microwave controllers, electronic games, and computer peripherals are typical examples of systems using embedded processors. With the falling cost and increasing power of microprocessor systems, the range and capabilities of embedded processors are expanding.

As 32-bit devices with their improved software support and speed move into the embedded control arena, it is natural to see them applied to the same applications currently supported by 8- and 16-bit devices. This trend tends to "raise the intelligence" of control systems while maintaining their embedded nature.

The evolution of printer systems clearly shows how the nature of embedded processors can change. The simple impact-head printers were implemented with 8-bit microcomputers as embedded processors. Today, high-resolution laser printers are emulating the impact printers they replaced and are providing the added capability of interpreting high-level page description languages. Consequently, the processing system of the laser printer is often more powerful than the host system it is connected to. Although the printer and its software interface have become more advanced, the processor is still being used in an embedded fashion.

In general, embedded systems are more cost sensitive than host systems, and large-volume embedded-system applications require relatively few chips. In embedded applications the processing is more a means to an end than an end in itself as in host applications. The embedded-system designer's goal is to provide the level of processing power necessary for the application with the highest system reliability at the lowest cost.

Applying host processors to embedded-processor systems. Most 32-bit microprocessor design to date has focused on host systems. These designs assume large systems such as multitiered memory systems with virtual demand paging memory management. Most of the new RISC (reduced instruction set computer) machines also require specialized memory systems such as fast memory subsystems (in some cases special external caches), very wide buses, and sometimes multiple buses. Another important factor is the bus speeds that many of these new processors require; these speeds are beyond most available application-specific ICs (ASICs) and can mean requiring very fast external logic with relatively low levels of integration.

These features are desirable for larger host applications, but for most embedded applications they are either unnecessary or too expensive. The new pro-

cessors, for example, typically require ceramic packages of 100 pins or more, resulting in higher component and system costs.

Thus, a large practical gap in system complexity and cost lies between most 32-bit microprocessors and their 16- and 8-bit predecessors. The needs of embedded controller applications for faster and easier-to-use microprocessors without the unnecessary impediments associated with host systems are not being addressed by most 32-bit processors.

Figure 1 diagrams the trade-offs in the current microprocessor market. On one axis is the relative system cost of the processor and its intended memory system, and on the other axis is relative performance. The general-purpose 32-bit microprocessors lie in the high-performance and high-system-cost region of the graph. The processors alone for these systems cost over $300 and require fast memory in the form of external static RAM (SRAM) caches to achieve optimum performance. The RISC chips on today's commercial market likewise have been designed for high-performance and high-cost systems. At the lower cost and lower performance end are the 8-bit and 16-bit microprocessors and microcomputers. In the case of microcomputers the memory is built into the processor chip. The 8- and 16-bit microprocessors typically have SRAM interfaces and require external logic to connect to lower cost DRAM. Consequently, high-cost SRAM has been used for lower chip count memory systems.

The 34010 is positioned between the two extremes in the cost-performance trade-off. It offers substantially more performance than the 16-bit microprocessors and lower system cost than the 32-bit general-purpose processors. The two positions of the 34010 on the graph in Figure 1 depict its relative performance in general-purpose processing and its advantage in graphics due to its special graphics processing hardware.

Application microprocessors. An application microprocessor is a general-purpose microprocessor designed with special hardware and instruction support for a specific system application. These microprocessors provide significant performance and cost advantages for applications needing their special capabilities. Application microprocessors grew out of the recognition that certain functions occur more frequently in embedded applications than in general-purpose processing.

The TMS320 digital signal processor (DSP) is a good example of an application processor family aimed at a specific application area. Designed for DSP applications, it optimizes the performance of high-speed multiplication operations typical in these applications. Many other applications have made use of the unique capabilities of DSP chips, including such varied applications as voice recognition, adaptive suspension, and image compression for graphics.

The TMS340 graphics system processor (GSP) family, of which the 34010 is the first processor, is aimed at

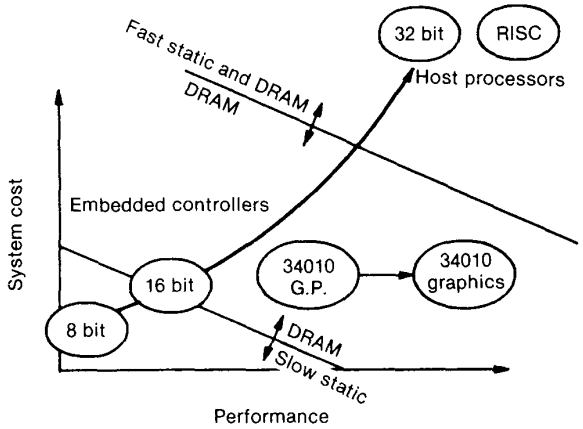


Figure 1. Embedded controller system trade-offs.

the bit-field processing and large memory spaces associated with graphics rendering. Its large address reach, bit-field processing capability, on-chip timers, and DRAM interface make the processor well suited to many embedded-processing applications.

Focus on embedded processing. Because it was intended to be an embedded processor or second processor in many systems, the 34010 has a set of features focused on reducing system complexity. It assumes a small system model with a single external memory hierarchy. The silicon budget was put into such features as DRAM control, timers, bit-field processing, pixel processing, and simple connection to a host processor or communication channel.

Figure 2 shows the distribution of the processor's 68 pins. Table 1 lists the functions of these pins. Note the

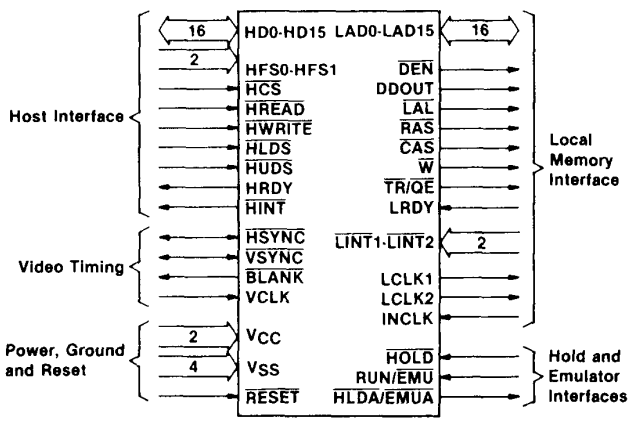


Figure 2. TMS34010 pin diagram.

Table 1. TMS34010 pin description.

Name	Pin	I/O	Description
Host Interface Bus Pins			
$\overline{HCS}$	66	I	Host chip select
HD0–HD15	44–51,53–60	I/O	Host bidirectional data bus
HFS0,HFS1	67,68	I	Host function select
$\overline{HINT}$	42	O	Host interrupt request
$\overline{HLDS}$	63	I	Host lower data select
$\overline{HUDS}$	62	I	Host upper data select
HRDY	43	O	Host ready
$\overline{HREAD}$	64	I	Host read strobe
$\overline{HWRITE}$	65	I	Host write strobe
Local Interface Bus Pins			
$\overline{RAS}$	38	O	Local row-address strobe
$\overline{CAS}$	39	O	Local column-address strobe
DDOUT	36	O	Local data direction out
$\overline{DEN}$	37	O	Local data enable
LAD0–LAD15	10–17,19–26	I/O	Local address/data bus
$\overline{LAL}$	34	O	Local address latched
LCLK1,LCLK2	28,29	O	Local output clocks
$\overline{LINT1}$,$\overline{LINT2}$	6,7	I	Local interrupt request pins
LRDY	9	I	Local ready
$\overline{TR/QE}$	41	O	Local shift-register transfer or output enable
$\overline{W}$	40	O	Local write strobe
INCLK	5	I	Input clock
Hold and Emulation			
$\overline{HOLD}$	8	I	Hold request
RUN/$\overline{EMU}$	2	I	Run/Emulate
$\overline{HLDA}$/$\overline{EMUA}$	33	O	Hold acknowledge or emulate acknowledge
Video Timing Signals			
$\overline{BLANK}$	32	O	Blanking
$\overline{HSYNC}$	30	I/O	Horizontal sync
VCLK	4	I	Video clock
$\overline{VSYNC}$	31	I/O	Vertical sync
Miscellaneous			
$\overline{RESET}$	3	I	Device reset
V_{CC}	27,61	I	Nominal 5-volt power supply
V_{SS}	1,18,35,52	I	Ground

emphasis both on adequate local bus control and on a host interface for attached processing. More than one third of the pins on the device are dedicated to this embedded function support. Almost half the pins support the local address/data bus designed to make DRAM interfacing straightforward. The device performs the row/column address multiplexing necessary to interface efficiently to DRAM.

Overview of the internal architecture

As shown in Figure 3, the internal architecture of the TMS34010 consists of six major blocks: main CPU, instruction cache, memory controller, host interface, display controller, and internal clocks.[2-4]

Main CPU. The CPU is controlled by an 808 microstate control ROM (CROM). It can execute simple instructions in a single cycle when in cache, and it can perform complex microsequences such as the pixel block transfer instructions (PIXBLTs). Each CROM word has 166 bits, which support highly parallel operations within the CPU.

The main internal data paths are 32 bits wide and contain the key elements for efficient execution of both graphics and general-purpose instructions. The thirty-one 32-bit registers are organized as two register files, each with 15 registers sharing a common stack pointer register. The ALU, adder/subtracter, and barrel shifter have separate inputs and control and can all operate in parallel.

Instruction cache. The instruction cache, transparent to the programmer, was designed to support fast execution while using DRAM for the system memory. During the execution of time-critical loops, the cache helps in two ways: It supports fast instruction fetches, and it frees the memory bus for reading and writing. The programming model is a single memory space for instructions and data, and the cache is used to separate them for parallel access.

The 256-byte cache uses a four-way set associative with four segments (sets), eight subsegments per segment, and four words per subsegment. Each subsegment has a "present" bit, and direct replacement of "misses" within a subsegment is made for the four words. The four segments use a four-location CAM (content-addressable memory) to determine whether a segment is present and a four-location LRU (least recently used) stack to determine which segment is replaced on a miss.

Memory controller. The memory controller is actually a separate microcoded processor with its own control ROM that coordinates all accesses to local memory. CPU, host, and video requests are prioritized and scheduled by the memory controller along with DRAM refresh cycles. The memory controller is responsible for generating the control signals for the local memory bus.

All the necessary DRAM and VRAM control signals are generated by the memory controller. The row and column addresses along with data are triple-multiplexed on the 16-bit local address/data (LAD) bus under control of the memory controller. Only one external buffer/latch is required to connect the processor to DRAM.

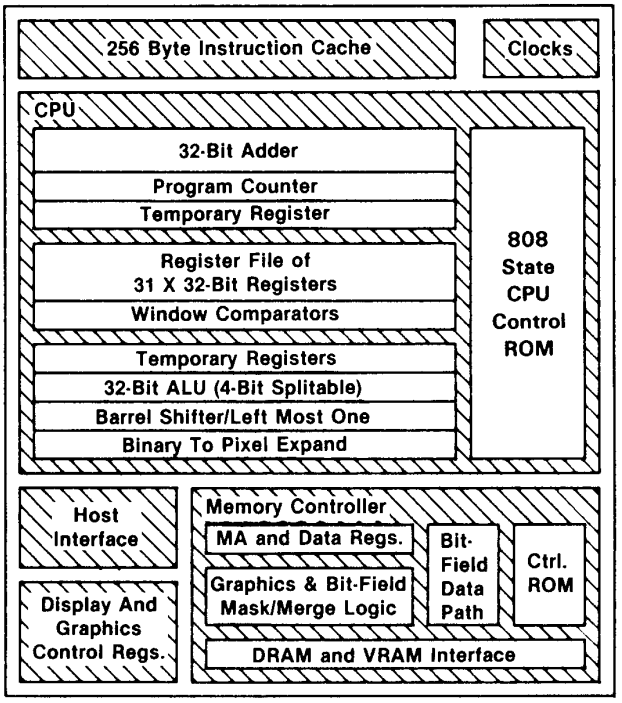


Figure 3. TMS34010 internal architecture.

Figure 4 shows the local bus timing. The signals LCLK1 and LCLK2 are the local bus clocks generated by the processor for timing on the bus. The $\overline{RAS}$ (row address strobe) and $\overline{CAS}$ (column address strobe) signals directly generate the timing required by DRAM. The $\overline{LAL}$ (local address latch) signal, which falls after the column address is valid, is used to latch the column address. For static memory interfacing, the $\overline{RAS}$ signal latches the upper address bits, and the $\overline{LAL}$ signal latches the lower address bits. $\overline{W}$ (write enable) is designed to give the proper write signal for DRAM. The $\overline{DEN}$ (data enable) and DDOUT (data direction out) signals enable and control the direction of bus transceivers if necessary in the system. $\overline{LRDY}$ (local ready) is a processor input used to lengthen memory cycles for slower memories and peripherals.

The $\overline{TR}/\overline{QE}$ (shift register transfer/output enable) signal, controlling a corresponding input on the VRAM, serves two dissimilar functions. It initiates the shift register transfer cycle on the VRAM, and it is timed to control the output enables on the VRAM and the 4-bit-wide DRAM devices.

The mixing of the dissimilar functions of the VRAM's shift register transfer signal and its output enable on $\overline{TR}/\overline{QE}$ (alternatively named $\overline{TR}/\overline{G}$) came about because the 34010 was defined in conjunction with the original VRAM, the TMS4161.[5] The purpose of the output enable function was to make it un-

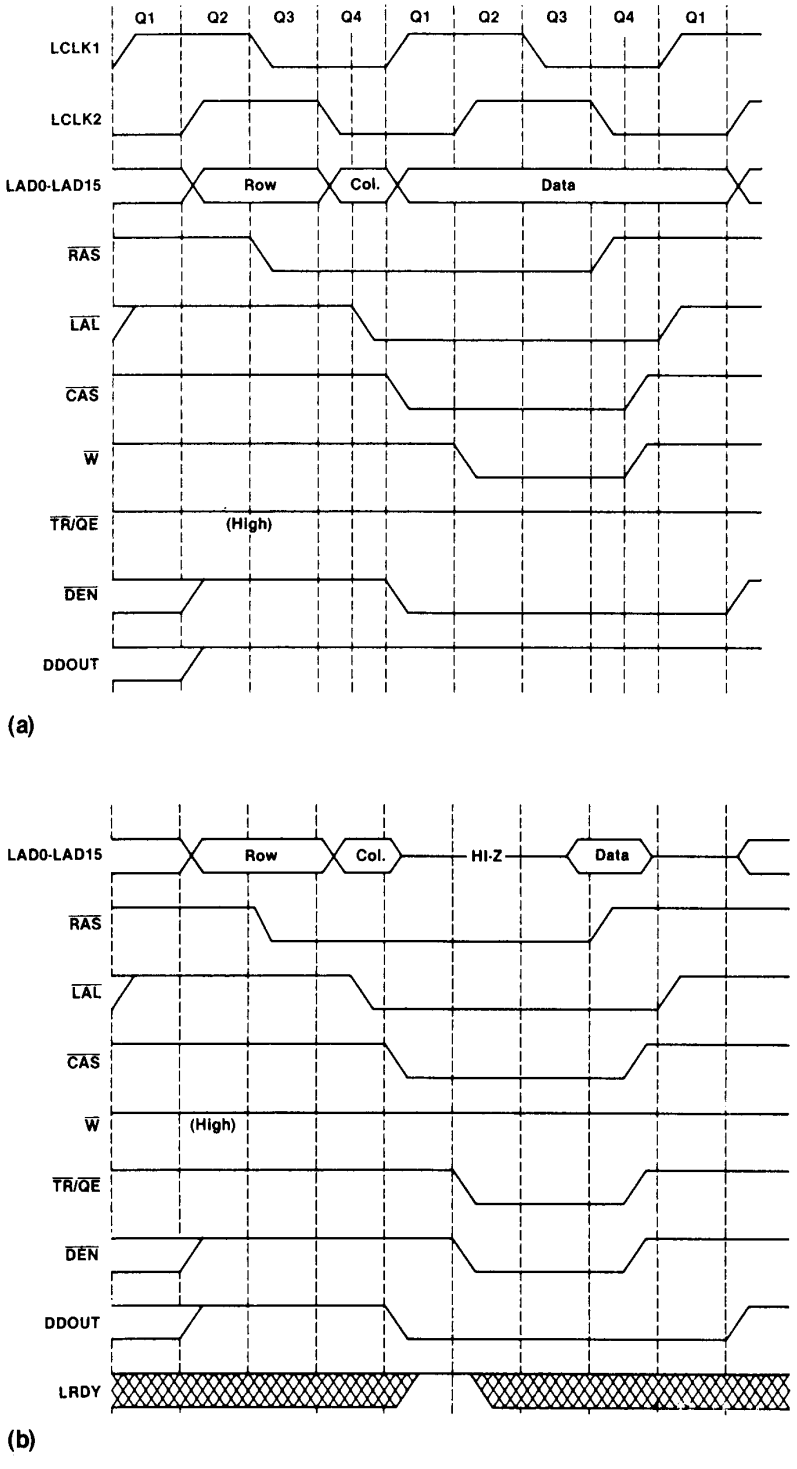


Figure 4. Local bus cycle timing: (a) write cycle; (b) read cycle.

necessary to add extra buffers between the VRAM and the 34010, but there were no pins left on the first VRAM. The $\overline{TR}$ signal already existed, so the designers decided to time-multiplex the signal. This invention of necessity has become standard on all VRAMs designed since.

The memory controller also plays an important role in offloading the main CPU from the burden of bit-field processing. On bit-field operations (including any move instructions), the CPU simply passes the starting bit address, the field size, and the data to the memory controller; then the CPU is free to execute the next instruction out of cache. Before sending the data, the CPU uses its barrel shifter to get the data in the proper bit alignment, but the memory controller actually performs the masking and merging operations to insert the field. On the basis of field size and address alignment, it computes and schedules as many read and write cycles as necessary, requiring no further interaction with the main CPU.

Host interface. Unlike other microprocessors, the 34010 has a dedicated interface port to allow another processor to gain access to its memory. The host interface is actually a communication channel into the 34010's memory space and can be used for other purposes. Accesses requested via this 8- or 16-bit data interface are scheduled at higher priority than the 34010's CPU by the memory controller. The local memory can also be directly accessed by a conventional hold interface.

Four internal memory-mapped registers are dedicated to the host. These registers are loaded from the 8/16-bit host bus under the control of two function select pins (HFS0, HFS1). Two of the 16-bit registers combine to form a 32-bit address into local memory. Another register holds the data written to and

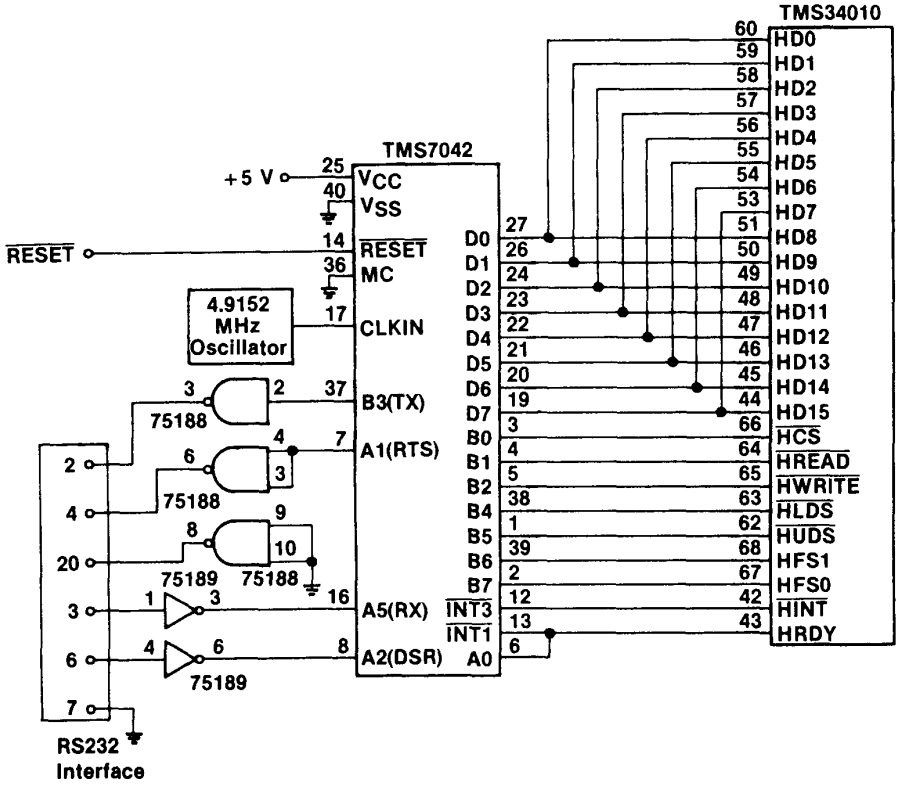


Figure 5. TMS7042 as serial port host.

read from memory, and the fourth contains control information.

In addition to the data transfer registers, the host processor has access to a 16-bit control word. Using this register, the host has complete control of the 34010. The control register can be set up for automatic incrementing of the address register on reads and/or writes for block access throughput of five megabytes per second.

The management of these resources and the memory pointer registers can be controlled by the 34010 or the host. In a 34010-controlled interface scheme, the host does not need to have any knowledge of the local memory organization. The 34010 is capable of loading the address registers locally, thus decoupling the host from the local memory implementation. The host interface's control register provides the host access to dedicated message-passing bits, a register-controlled interrupt in and out, and a nonmaskable interrupt. The control register also supports halting of the 34010's CPU and flushing of the contents cache, particularly useful when downloading code. The halt control can also be very useful in capturing the full bandwidth of the local bus for time-critical data transfers.

Using the host interface for other functions. Although intended to be a port for a host processor's commands and data, the host interface provides an economical way to access the local memory for any purpose. One system, for example, uses an 8-bit microcomputer as a front-end serial controller. The microcomputer, operating as a serial port or network interface, attaches very easily to the 34010. Figure 5 shows the connection of the 8-bit TMS7042 microcomputer with serial port to the processor host port.

The host has access to the interrupt vector table via the host interface. Thus, the host or other local processor can dynamically install interrupt vectors and their associated interrupt routines. The external processor can then initiate interrupts via either the 34010's interrupt pins or the host control register. For externally generated interrupts, there are both maskable and nonmaskable interrupts in addition to Reset. Two external pins, LINT1 and LINT2, are dedicated external interrupt inputs. Two bits in the host control register control the operation of the nonmaskable interrupt. One bit invokes the interrupt itself while the other enables or disables the stacking of the program counter and status during the interrupt routine. Preventing the

stacking process in the interrupt routine is sometimes necessary to gain immediate control of a system in which the stack pointer value may have been corrupted.

The host interface can also be used as an independent debug/test channel into the local memory space. Debugging programs can use this port to access state variables supplied by a local monitor program giving information about the processor's internal machine state. Similarly, the port can be used for communicating system state variables at the end of prescribed system tests at bootup. These can be either controlled locally by the 34010 or command-driven over the host port.

Display controller. Although designed for CRT control, the display controller is a very flexible counter that can be used for a wide range of timing or event-counting functions. Functionally, it has two cascaded 16-bit counters (for a total of 32 bits of dynamic range) with four programmable comparators on each counter. The comparators are used to generate three output signals (nominally used as vertical sync, horizontal sync, and blanking). A programmable interrupt can be set on the basis of any vertical count.

The input clock for the counter can run from 0 (stopped) to 7.5 MHz and can be totally asynchronous with the processor clock. The counters support an external video mode, which allows external events to reset the vertical and horizontal counters independently.

Internal clocks. The clock timing logic converts the input clock frequency into the various internal timing clocks needed to operate the processor. In addition, it generates the local bus clock signals used by external devices to operate synchronously with the processor's local bus. Current devices operate with a divide-by-eight from the input clock to generate 130-ns cycle times for a 60-MHz input clock.

Model of operation. The large register file, instruction cache, and independent memory controller are designed to work together for high performance. The processor's model of operation is that instructions controlling the algorithm are automatically loaded into the cache, data is stored in the large register file, and the memory controller and its bandwidth into the off-chip memory are used only in manipulating external data. Other system operations such as host accesses are requested and scheduled by the memory controller as a background task, and interrupts are used for communications, handshaking, and error conditions.

Feature set and definition decisions

In designing a processor, one must evaluate package size, pin count, target memory type, and a wide variety of features that can be incorporated. The 34010 was designed to bring the high performance and ease of programming associated with 32-bit microprocessors to low-cost systems. Contrary to common belief, the difficult part of product definition is not identifying good features to add (which are infinite) but making the tough choices between what can be included and what must be left out for cost reasons.

To achieve the best system cost-performance ratio, the feature set and functions of a processor must be balanced. *Balanced* means that the features complement each other in a practical way. For example, the 34010 was targeted at low-cost memory systems. Features such as the on-chip instruction cache and large register file provide faster execution by reducing the need for access to the DRAM, while direct DRAM control and multiplexed addressing reduce system cost and complexity.

Large linear address space with bit-field processing. Graphics display systems need large amounts of memory, leading to several basic design decisions. A clean architecture to support a large linear address reach is needed, and this in turn requires a 32-bit internal data path to manipulate the large addresses quickly.

Unique among microprocessors, the 32-bit address of the 34010 points to the exact bit location in memory rather than to the byte, word, or long-word of other processors. The whole of memory is viewed as a series of bits ordered from 0 to $2^{32}-1$. All the memory addressing modes directly support bit-field processing. Field lengths from 1 to 32 bits are directly supported in all general-purpose move operations. The autodecrement and autoincrement addressing modes also use the field size to adjust address registers. In addition, any size array of fields can be moved with the PIXBLT instruction (pixel block transfer).

The byte, word, and long-word are artifacts of older processor architectures in which there was only one basic data size—the byte. In processors with limited address bits, byte addressing served as a good compromise between data granularity and address reach. Also, earlier architectures did not have the hardware, such as barrel shifters and mask/merge multiplexers, to quickly handle problems bit-aligned addressing can create. But the 34010's architecture started with a 32-bit address reach, and its hardware manipulates bit fields with equal ease as bytes or words, so there was no need to impose a distinction.

Bit-field processing is directly supported by special hardware on the device. This extends the bit control and manipulation facilities found in controller systems, adding memory bit manipulations to those available in registers. Processors without this facility must perform many operations to achieve the same effect. To transfer a nonaligned byte field from one memory location to another, the typical processor must read the source word into a register, mask out uninvolved bits, align the source word with the destination location,

Contrary to common belief, the difficult part of product definition is not identifying good features to add.

read in the destination word, mask out the affected field (byte), logically merge the source and destination words, and then write the result to the destination. The 34010 CPU can direct this operation within a single instruction and then continue to execute operations within its register file while the destination memory accesses are being performed.

Large register file. The decision to go to thirty-one 32-bit registers rather than the 16 or fewer found on most machines was driven by the desire to make time-critical functions run faster and to ease assembly-level programming. Register-to-register operations occur in a single cycle when running out of cache and can occur in parallel with the memory controller's completion of previously started write cycles. This parallelism naturally occurs in routines in which the CPU is computing functions written to a series of memory locations. The example used as a model during the 34010's definition was an ellipse-drawing routine, in which the address computations and data values are held in the register file and the pixels to be written are sent to the memory controller. A large register file means that all the parameters for most time-critical functions can be kept inside the processor, thus preventing the thrashing of parameters between the register file and memory. By preventing thrashing, the register file frees the memory bandwidth for other functions such as memory write cycles, host accesses, and DRAM refresh. In many programs the large register file can be the single most important feature for improving performance.

There are several ways to take advantage of the larger register file. It is helpful to the compiler to have several working registers for storing intermediate results of computations such as evaluating expression trees. Intelligent compilers (made popular by RISC architectures) can take even greater advantage of more registers by tracking the contents of registers and performing efficient constant generation. This also means that the programmer trying to optimize his or her own code in high-level language will not compete with the compiler for access to a small number of register variables to hold critical values.

Important, time-critical routines are usually written in assembly code to optimize performance. It is in writing such routines that the benefit of the large register file is most obvious. If there are too few registers, most of a programmer's algorithmic effort can be expended on register management. With 31 registers, a large number of critical values can be kept in the register file during execution. This is sufficient for most control algorithms and has a direct impact on the ease of design of the algorithm implementation. In addition, the processor's efficient register-stacking and -unstacking instructions make register allocation and management trivial.

The 34010's initial target applications area clearly indicated that a large number of 32-bit registers were very desirable. Many graphics algorithms require a large number of data variables and address pointers. Without the target application to measure against, it would have been much easier to define an architecture with fewer registers.

The on-board registers are organized as two 15-register files named the A and B files. The distinction between the files is that instructions requiring two registers must have both registers in the same file. The move register to register instruction is an exception to this rule so that data can be moved between register files. The other exception is the stack pointer that is accessible as the 16th register of either file.

The memory move, arithmetic, Boolean, shift, and other register-based instructions and addressing modes can use any of the 31 registers. This is particularly important in optimizing languages such as C, where any operation that can be performed on data can be done on address pointers. Therefore, data/address placement is not constrained, giving the compiler great latitude in organizing register usage.

An important part of the processing task for embedded-control applications is the processor's interrupt support. An added benefit of the large register file is the ability to *dedicate* registers to time-critical functions such as interrupt routines. Embedded applications often have parameters that must be dealt with quickly when an interrupt occurs. Dedicating part of the register file to these values can save considerable time swapping data in and out.

Instruction cache. The four-way set associative approach was designed to support two loops, each straddling set boundaries without thrashing. During the 34010's early definition, relatively little time was spent defining the cache compared with time spent justifying its hypothetical performance on pathological cases, cache architecture being part philosophy and part science.

Since graphics was the focus of the 34010, circle, line, and ellipse algorithms were used as model test cases for the cache. The nature of these algorithms is that they have a single loop, but a decision based on incremental calculations is made between two sets of code on each loop. The goal was to have these algorithms fit in the cache without thrashing and without requiring the programmer to worry about the position of code in memory. The four-way set associative method worked well on the graphics test cases.

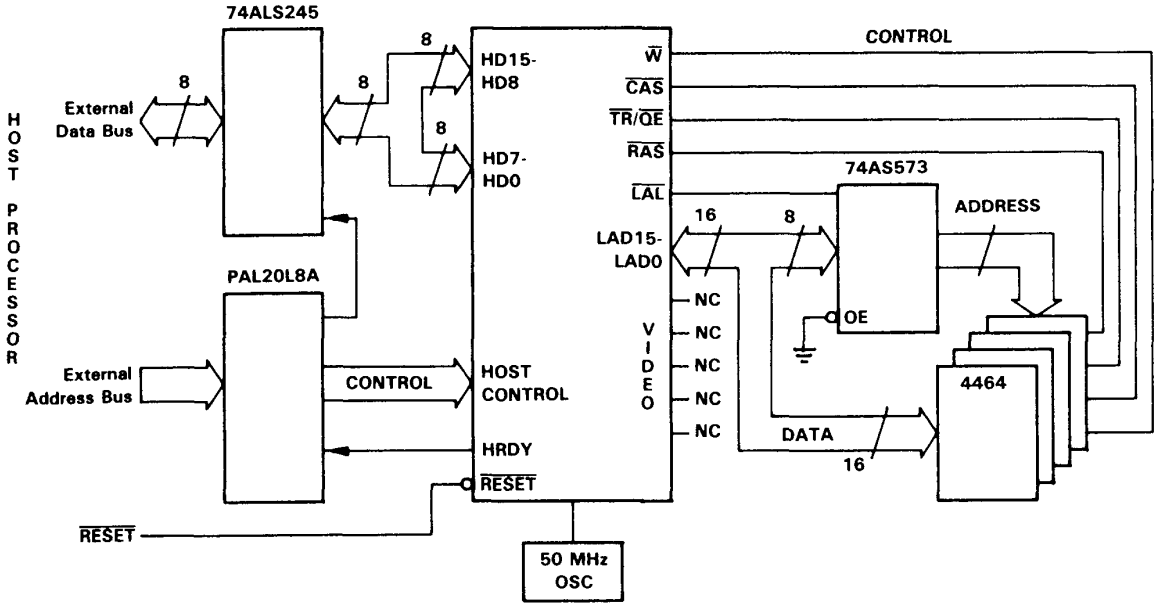


Figure 6. TMS34010 DRAM system.

Unlike general data caches, relatively small instruction caches can have very good hit rates. This is due to the locality of programs and the fact that many time-critical functions are done in small loops. Additionally, instruction caches are less expensive than general data caches since they do not have to deal with data writing back to main memory (unless self-modifying code is allowed).

Making DRAM accessible to low-chip-count systems. A key difference between the 34010 and other microprocessors is its direct interface to DRAM. This interface changes the point at which DRAM becomes cost effective in a system.

Figure 6 shows the minimum number of devices required to implement an embedded system in a low-cost PC platform. A PAL (programmable array logic) and an octal transceiver provide the host processor access to the 34010's five-chip, 128K-byte DRAM system. The host processor provides nonvolatile program store and bootstrap capability for the system. This provides two benefits. First, the host has random access into the processor's memory space for communications, eliminating the need for a separate I/O channel. Second, the operating software of the embedded processor can be updated or entirely changed remotely from the host system. This eliminates the need for exchanging parts or otherwise "touching" the embedded system hardware in order to perform field upgrades.

Although DRAMs are the most cost-effective memory device per bit, they have not generally been used in low-cost, low-chip-count systems. DRAM devices require complex timing and address multiplexing, which, if not built into the microprocessor, require external control. Because of this overhead, applications needing relatively few memory chips have not previously used DRAM.

The capacity of DRAM chips allows the designer fairly large amounts of memory in very few chips. With 256K-bit DRAMs organized as 64K deep by 4 bits wide per chip, only four chips give a 16-bit data width and 128K bytes of memory. With 256K by 4 (one megabit) DRAM devices, four chips result in 512K bytes. Thus, a relatively small number of DRAM devices can provide all the RAM required for most embedded applications.

Having the DRAM interface built into the processor can provide performance benefits in systems using these lower cost memories. Typically, considerable time is lost in interfacing a microprocessor to the DRAM because of the inherent losses in going through another controller chip and because of timing mismatches between the microprocessor's signals and the DRAM. The processor uses a high-frequency (40 MHz to 60 MHz) input clock so that it can precisely place the large number of timing edges required by a DRAM.

Since DRAM is generally slower than other memory types, considerable attention was given to improving

performance in a DRAM-based system. While "big system" host models may use multitiered memory hierarchies with external instruction and data caches and very fast buses to achieve very high performance, these features come at a cost. The instruction cache and the large register file were designed to reduce accesses to the DRAM.

Packaging for low cost. The processor requires only 68 pins and power dissipation of about one-half watt, allowing it to use a very low cost plastic package. Packaging dominates the cost of making most high-volume components, so fitting a low-cost package was an important design consideration. One key to keeping the pin count small lay in the DRAM timing. Time multiplexing the row address, column address, and data on the same pins fit the DRAM timing requirements well and saved a number of pins.

While the 34010 has a 32-bit internal data path, the external data path is 16 bits to reduce overall system cost. This design decision kept the pin count down and eliminated the overhead associated with a wider data bus. As part of the original strategy, wider data bus versions of the device are under development.

Opcode formats and design simplicity. The 34010 follows the RISC philosophy of having very few fixed opcode formats and has a fixed-length 16-bit instruction. As other researchers have indicated, a 32-bit opcode, as in most RISC machines, is not efficient in terms of the number of bits required by a function.[6] The same philosophy of architectural efficiency has resulted many times in inefficient instruction encoding. Opcodes (not including immediate data) on the 34010 were kept to 16 bits in length to reduce instruction bandwidth requirements and to obtain better utilization of the instruction cache.

Another objective was to make the instruction formats easy to decode. There are only four opcode formats: one-register, two-register, short-constant, and jump opcodes, as shown in Figure 7. These four formats are organized into two groups for decoding purposes: single-register and everything else. For the one-register format a special nonbinary address decoder was used to avoid an extra level of decoding. Thus, one less level of instruction pipelining was required, resulting in simpler logic and faster jumps and branch execution. For the other formats, the upper 8 bits (bits 0 to 15) of the opcode specify the instruction. Redundant states in the microcode preclude the need for detailed decoding beyond the upper 8 bits.

The two-register-file organization was devised to provide a large register file and at the same time to maintain a compact, easy-to-decode, 16-bit instruction word. This organization requires only 9 bits (since the register file select bit is shared) as opposed to 10 bits to specify two of 31 different registers, but it also limits operations between files. Other approaches were considered, such as split address/data files as was done on the 68000.[7] But that organization might have restricted the operations that could be done on addresses, limited the flexibility of register usage for address or data, and made instruction decoding more complicated.

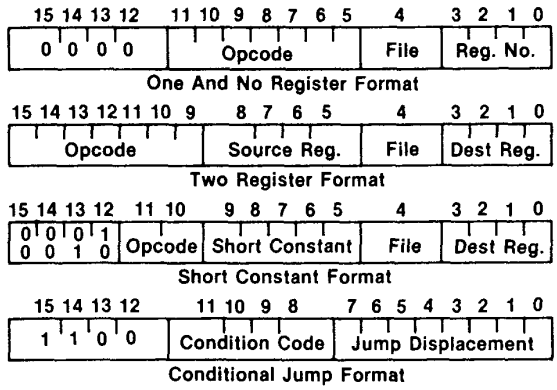


Figure 7. TMS34010 opcode formats.

RISC. What discussion of microprocessor architecture decisions would be complete today without commenting on RISCs, a much overused and abused term?[8] The 34010 design, influenced by the Berkeley RISC philosophy, has an RISC base instruction set to which were added special graphics instructions. Independent of the RISC influence, our experience designing the 9900 family of microprocessors had demonstrated that decoding complex instructions wastes hardware and performance.

Statistical data on general applications running on our 990 minicomputers agreed with other studies indicating that move, jump, increment, decrement, and add operations dominate the mix of instructions executed.[9,10] In contrast, embedded applications can stress a specific function and thus may have instruction mixes that look very dissimilar to the general-purpose cases.

The 34010 supports variable bit-field sizes, not found in many RISCs, but it does adopt the Berkeley RISC concept of sign or zero extending to 32 bits for smaller data types when moving them into a register.[11] Like the Berkeley RISC, it performs 32-bit register-to-register arithmetic.

Although it does not use the Berkeley register window concept, the 34010 does offer a larger register file than most 32-bit microprocessors. Some consideration was given to the register window concept since it is similar to the older workspace pointer concept of the TMS9900 family, which used a pointer to its register file in external RAM. The 9995 microprocessor had 256 bytes of internal memory, typically used to hold a large number of workspace "registers," but this was still in slower RAM rather than faster, dual-ported reg-

ister file storage. The 34010's designers decided that the very large windowed file of the Berkeley RISC would add too much to the chip's size and would create speed paths that might limit fast operation.

In keeping with the RISC philosophy, the 34010 executes most basic instructions in a single cycle. By using a single-level, very wide (166-bit) control word, the processor can execute single-cycle instructions without extra pipelining. The opcodes were kept very simple and were constructed to act directly as addresses into the microcoded ROM. This eliminated the extra decode logic normally associated with microcoded microprocessors. In effect, whereas a RISC machine uses a PLA for instruction decoding, a single-level ROM lookup is sufficient on the 34010.

Departing from the strict RISC philosophy, the designers recognized that applications would benefit greatly from more sophisticated instructions that do not fit the single-cycle model of pure RISC machines. The complex PIXBLT instructions can be pipelined much more efficiently as single instructions than if broken into multiple single-cycle instructions. With internal microcoding, the address manipulations, field extractions, merging, and multiple memory cycles can be more efficiently coordinated. The wide control supports many parallel operations for faster execution.

Real-time software development. The importance of assembly and HLL support was the basis for TI's decision to support the 34010 family with source and object management tools for assembly and C. In addition, both real-time and software-only emulation tools provide debugging capability for hardware and algorithm prototyping. Application libraries provide source code algorithms for specific design tasks such as CCITT (International Consultative Committee for Telegraphy and Telephony) Group 3/Group 4 compression and decompression, as well as for various graphics standards from the Massachusetts Institute of Technology, the American National Standards Institute, Graphic Software Systems, Microsoft, and others.[12] Embedded-processor applications also usually require real-time operating systems. These have been developed specifically for embedded processing for the 34010 and are available from external parties.[12,13]

Applications in embedded control

The general-purpose processing power and low system cost of the 34010 make it useful for a wide number of applications. Because of its graphics hardware and instructions, it naturally found wider initial use in graphics and graphics-related applications, but over time more designers are finding its larger applicability. The bit-field processing is an important feature that makes it attractive in control and data compression applications.

Graphics terminal and display systems. The 34010 has been widely used in both graphics add-in boards and display terminals.[12] The hardware support for video display terminals, DRAM, VRAM, and host interface greatly reduces the cost of these systems. The support for graphics drawing, X-Y addressing, pixel block transfers, and general-purpose processing provides high performance and easier programming.

The 34010 can also be used to offload nongraphic processing from the host to improve overall system performance. As a result, entire applications have been written to run directly on the embedded processor, using the host only for keyboard interface, disk drive, and other I/O functions.

Consumer electronics. Because it is both a powerful microprocessor and a graphics chip integrated in low-cost packaging, the 34010 has potential applications in new consumer graphics designs. To date, most video game systems have used a combination of an 8-bit microprocessor and a video controller chip to support graphics functions such as sprites (two-dimensional X-Y positional characters). The microprocessor and the controller chip can be replaced by a single 34010. The processing power of the resultant system opens up whole new areas of education as well as entertainment. Electronic building blocks, home 3-D CAD, flight simulators, driving trainers, and 3-D adventure games are just a few of the possibilities.

Image compression for facsimile and CD-ROM. Graphics and images require large amounts of data to be stored and/or transmitted, making data compression essential both for performance and cost. The need to globally transmit images has resulted in the CCITT facsimile standards for data compression.[14,15] Inexpensive stand-alone systems can be constructed incorporating CCITT Group 3, a 9600-bps modem, and paper-handling control. The acceptance of CCITT Group 3 compression has led to its wider use in other applications. A laser printer with the addition of a 9600-bps modem can double as a facsimile printer. The CCITT standard is also being used as a compression method in applications such as CD-ROM (compact disks) and scanners.

Fax-modem PC add-in boards provide facsimile transmission directly from a PC. They have a considerable quality advantage over printing out the image and then having it scanned by a stand-alone fax machine because they eliminate the distortion introduced during the scanning process. Since CCITT encoding is a generally accepted standard, it is also a convenient way to communicate images between machines without an intermediate step to paper.

The CCITT Group 3 and Group 4 standards are based on bit manipulations and variable field length encoding. This bit-field processing is applied to edge detection, run length encoding, and data movement. A graphics processor has the added benefit of being able

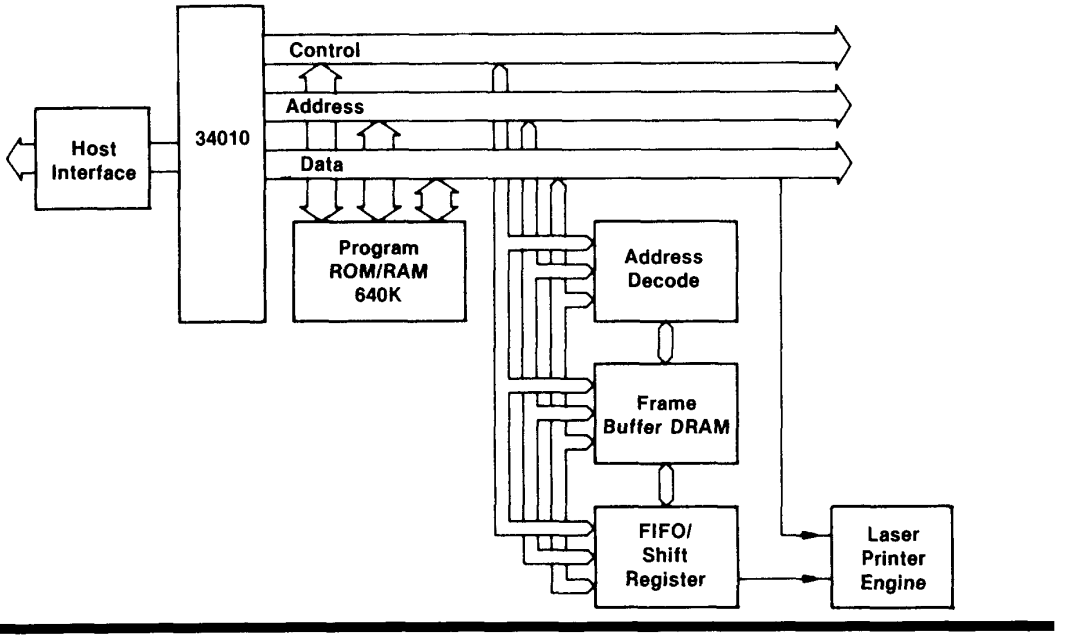


Figure 8. Laser printer system.

to generate, display, and manipulate the resulting images. In the fax-modem application, although most PC displays cannot display the high-resolution fax image, the image can be translated into a gray-scale image that can be displayed for preview.

Data compression techniques are being applied to increase the storage capabilities of such high-density media as CD-ROMs and other optical laser disks. As the storage requirements for these media increase, a higher degree of processing capability is required of the embedded controller to compress and decompress the data.

The CCITT standard is primarily focused on black-and-white document transmission and is not ideal for every application. There are denser compression methods, particularly for handling color and gray-scale images. Because of the 34010's programmability, it can be adapted to handle many of these unique compression methods.

Page printers. Laser and other printing technology, such as thermal dye transfer (used in many color printers), is an obvious application for an embedded microprocessor with special graphics capabilities. Figure 8 shows a 34010-based laser printer system. In addition to generating the print image, the processor can perform other functions such as controlling the print engine and the page feeder.

Page printers require the ability to move and manipulate (to the bit level) large bitmaps of data, calculate outline fonts, and emulate dot matrix printers. Many printers also support page description languages such as Postscript or compatible products. For economy the printers typically have used 16-bit microprocessors, but these processors can often be the limiting factor in print speed, particularly when a page description language is used. Even for black-and-white laser printers, the amount of memory required is large due to the relatively high-resolution (typically over 2500 by 3300 dots or one megabit) images they generate. This has led to the almost exclusive use of dynamic memory for the image buffer.

For 6- to 10-page-per-minute laser printers, the 34010 works alone. The on-chip DRAM controller reduces the external logic requirement, and the host interface is used as a general communication port. For performance of 15 to 60 ppm with page description requirements, multiprocessor configurations can be applied to improve throughput. The embedded system's master processor can be either a 16- or 32-bit microprocessor. The master processor handles the page description language interpretation and then passes the graphics processing to the slave processors. The graphics processing performance of the 34010 and the resulting low system cost makes this multiprocessor configuration feasible.

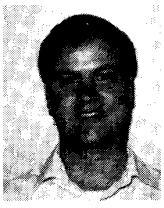
Additional embedded application areas for the 34010 include laboratory and factory data acquisition, optical character recognition, dashboard and cockpit instrumentation, cardiac monitors, radar control systems, process control, robotics, and medical imaging systems.

The 34010 is the first of a family of 32-bit embedded processors blending performance and system integration to support the growth of advanced applications. New family members currently in design will extend the design philosophy of this processor and improve the performance of the next generation of embedded systems.

References

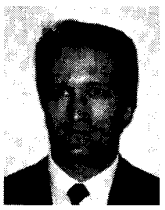
1. R. Noyce and M. Hoff, "A History of Microprocessor Development at Intel," *IEEE Micro,* Feb. 1981, pp. 8-21.
2. C.R. Killebrew Jr., "The TMS34010 Graphics System Processor," *Byte,* Dec. 1986, pp. 193-204.
3. M. Asal et al., "The Texas Instruments 34010 Graphics System Processor," *IEEE Computer Graphics and Applications,* Oct. 1986, pp. 24-39.
4. *TMS34010 User's Guide,* Graphics Products, Texas Instruments Inc., Jan. 1987.
5. R. Pinkham, M. Novak, and K. Guttag, "Video RAM Excels at Fast Graphics," *Electronic Design,* Aug. 18, 1983, pp. 161-168.
6. M. Johnson, "System Considerations in the Design of the Am29000," *IEEE Micro,* Aug. 1987, pp. 28-41.
7. E. Stritter and T. Gunter, "A Microprocessor Architecture for a Changing World: The Motorola 68000," *Computer,* Feb. 1979, pp. 43-52.
8. R.P. Colwell et al., "Computers, Complexity, and Controversy," *Computer,* Sept. 1985, pp. 8-19.
9. R.P. Blake, "Exploring a Stack Architecture," *Computer,* May 1977, pp. 30-38.
10. D. Fairclough, "A Unique Microprocessor Instruction Set," *IEEE Micro,* May 1982, pp. 8-18.
11. D.A. Patterson and C.H. Sequin, "A VLSI RISC," *Computer,* Sept. 1982, pp. 8-20.
12. *TMS34010 3rd Party Guide,* 2nd ed., Texas Instruments Inc., Oct. 1987.
13. *C Executive User's Manual,* JMI Software Consultants, 1986.
14. "Standardization of Group 3 Facsimile Apparatus for Document Transmission," Recommendation T.4, CCITT, 1984.
15. "Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus," recommendation T.6, CCITT, 1984.

Karl M. Guttag works in the Microprocessor and Microcontroller Division of Texas Instruments. Since 1982 he has been responsible for graphics product definition, including graphics processor architecture and the multiport video RAM. Previously, he was the IC architect of two 16-bit microprocessors and a design engineer of the TMS9918 video display processor. Guttag received his BSEE from Bradley University and his MSEE from the University of Michigan. He was elected a Texas Instruments fellow in 1988. He holds 25 patents in microprocessors and computer graphics hardware and is a member of the IEEE and the ACM.



Thomas M. Albers is responsible for software support of the 340 family of processors. He joined TI in 1980 and has worked with 4-, 8-, and 32-bit microprocessors. He has contributed to the microcode tools, software simulator, and operating environment for the TMS34010. Currently, he is working on strategic software development and TI's next generation of graphics devices. Albers received a BS in electrical engineering from Texas A&M University. He is a member of Tau Beta Pi and the ACM.

Michael D. Asal is a systems engineer in the Microprocessor and Microcomputer Products Division of Texas Instruments. He is currently working on the design of TI's next-generation graphics processor in Bedford, England. Since joining TI in 1982, he has been involved with the specification, architectural development, and design of the TMS34010 and higher performance follow-on devices. He was elected a member of the Semiconductor Group technical staff in 1987. Asal received the BSEE and MSEE from Bradley University.



Kevin G. Rose is a strategic marketing engineer with Texas Instruments. He provides application and marketing support to laser printer, terminal, and facsimile designers. He also manages business development for these markets. Rose received his BS from Brigham Young University and his MBA from the University of Utah.

Questions about this article can be addressed to Tom Albers, Texas Instruments Inc., PO Box 1443, M/S 712, Houston, TX 77001.