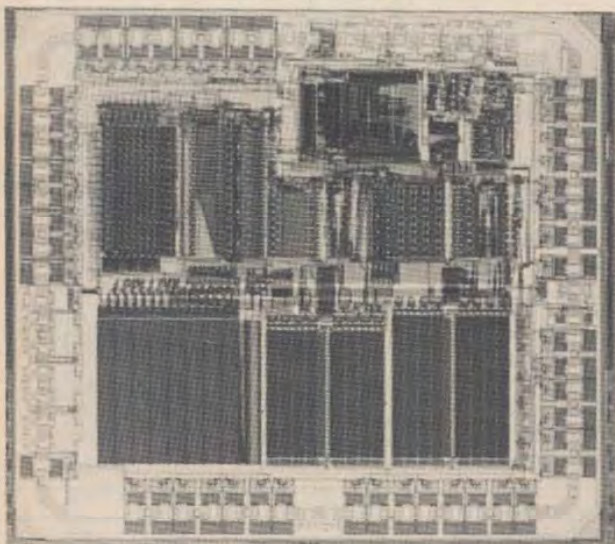


The Texas Instruments TMS320C25 Digital Signal Microcomputer

Gene A. Frantz, Kun-Shan Lin,
Jay B. Reimer, and Jon Bradley
Texas Instruments Incorporated



Capable of 10 million operations per second, the newest member of the TMS320 family can serve as an inexpensive alternative to bit-slice processors or custom ICs in digital signal processing applications.

Digital signal processing encompasses a variety of applications, including digital filtering, speech vocoding, image processing, fast Fourier transforms, and digital audio.¹⁻⁵ All DSP applications have several characteristics in common. First, they employ algorithms that are mathematically intensive. An example is the finite-duration impulse response, or FIR, filter, which in the time domain takes the form

$$y(n) = \sum_{i=1}^N a(i) \cdot x(n-i), \quad (1)$$

where $y(n)$ is the output sample at time n , $a(i)$ is the i th coefficient or weighting factor, and $x(n-i)$ is the $(n-i)$ th input sample. From this equation, we can see that the FIR filter contains an abundance of multiplications and additions (that is, sums of products). This equation is the general form of an FIR filter⁶ as well as the convolution of two sequences of numbers $a(i)$ and $x(i)$.⁷ Both operations are fundamental to digital signal processing.

Second, DSP algorithms must be performed in real time; i.e., they must not produce a delay noticeable to the user. In a speech recognition system, for example, the algorithms must not produce a noticeable delay between a word being spoken and that word being recognized. In an image processing system, processing needs to be completed within a frame update period.

Third, all DSP applications involve the sampling of a signal. Referring to Equation 1, we can see that the output $y(n)$ is calculated to be the weighted sum of the previous N inputs. In other words, the input signal is sampled at periodic intervals, and the samples are multiplied by a weighting factor $a(i)$ and then added together to give the output result $y(n)$. In a typical DSP application, the processor must be able to perform arithmetic computations and effectively handle sampled data in large quantities.

Last, DSP systems must be flexible enough to incorporate improvements in the state of the art. Many DSP techniques are still developing, and therefore their algorithms tend to change. Speech recognition, for example, is presently an inexact technique still undergoing algorithmic modification. This implies that DSP systems need to be programmable so that they can easily accommodate revised algorithms.

Over the past several decades, digital signal processing machines have taken several forms in response to application need and available technology. Array processors have long been the accepted solution for the research laboratory and have been extended to end applications in some instances. However, as integrated circuit technology has matured, digital signal processing has migrated from the array processor to the bit-slice processor to the single-chip processor. This has brought the cost of DSP solutions down to a point that allows pervasive use of the technology.

The members of the TMS320 family of devices are examples of the single-chip digital signal processor. The first member of the family, the TMS32010, was introduced to the market in 1983.^{8,9} It can perform five million DSP

operations per second, including the add and multiply functions¹⁰ required in Equation 1. The newest member of the family, the TMS320C25, can perform 10 million DSP operations per second,¹¹ and it combines the multiply/accumulate functions into one single-cycle operation.

Basic TMS320 architecture

The fundamental attribute of a digital signal processor is fast arithmetic operations. The members of the TMS320 family,¹⁰⁻¹² like many other digital signal processors, achieve fast arithmetic operations by employing

- a Harvard architecture,
- a dedicated hardware multiplier,
- special DSP instructions, and
- extensive pipelining.

Use of these concepts allows a digital signal processor to handle a vast amount of data and execute most DSP operations in a one-cycle instruction.

The TMS320 family utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture,^{13,14} the program memory and data memory lie in two separate spaces, permitting a full overlap of the instruction fetch and execution. The TMS320 family's modification of the Harvard architecture allows transfers between the program space and data space, thereby increasing the flexibility of the devices in the family. This architectural modification eliminates the need for a separate coefficient ROM and also maximizes processing power by maintaining two separate bus structures (program and data) for full-speed execution.

The TMS320 family's dedicated hardware multiplier employs a 16×16 -bit organization, which yields a 32-bit result and allows multiplication to take place in a single cycle. The special DSP instructions include DMOV (data move) and RPT (repeat), which speed up DSP operations. The extensive pipelining ensures maximum throughput for real-time applications.

The TMS320C25 architecture

The TMS320C25 digital signal processor is a microcomputer with a 32-bit internal Harvard architecture and a 16-bit external interface. It is a pin-compatible CMOS version of the TMS32020 microprocessor but has an instruction execution rate twice as fast and includes additional hardware and software features. The TMS320C25's instruction set is a superset of that of the TMS32010 and that of the TMS32020, and it maintains source-code compatibility with them. In addition, it is completely object-code-compatible with the TMS32020 so that TMS32020 programs can run unmodified on the TMS320C25. Some of the major features of the TMS320C25 are

- a 32-bit ALU and accumulator,
- an instruction cycle time of 100 ns,

- a single-cycle multiply/accumulate,
- use of low-power CMOS technology with a power-down mode,
- 4K 16-bit words of on-chip masked ROM,
- 544 words of on-chip data RAM,
- 128K words of data/program memory space,
- eight auxiliary registers with a dedicated arithmetic unit,
- an eight-level hardware stack,
- a fully static double-buffered serial port,
- concurrent DMA that uses an extended hold operation,
- bit-reversed addressing modes for fast Fourier transforms,
- extended-precision arithmetic and adaptive filtering support,
- full-speed operation of data move instructions from external memory,
- an accumulator carry bit and related instructions, and
- fabrication in 1.8- μ m CMOS and packaging in a 68-pin PLCC.

The 100-ns instruction cycle time provides a significant throughput advantage for many applications. Since most of the TMS320C25's instructions can execute in a single cycle, it can execute 10 million instructions per second. Most of the other features listed above also contribute to the TMS320C25's high throughput.

The TMS320C25 includes instructions to perform the data transfers between program space and memory space discussed earlier. Externally, the program and data memory spaces are multiplexed over the same bus so as to maximize the address range for both spaces and minimize the pin count of the device. Internally, the TMS320C25 architecture maximizes processing power by maintaining two separate bus structures, program and data, for full-speed execution.

Program execution in the device takes the form of a three-level instruction fetch-decode-execute pipeline. This pipeline is invisible to the user except in cases in which it must be broken, such as for branch instructions. In this case, the instruction timing takes into account the fact that the pipeline must be emptied and refilled.

Two large, on-chip data RAM blocks (a total of 544 words), one of which is configurable either as program or data memory, are provided. An off-chip, 64K-word, directly addressable data memory address space is included to facilitate implementations of DSP algorithms with large data memory requirements. Four-K words of on-chip program ROM and 64K words of off-chip program address space are available. Large programs can execute at full speed from this memory space. Programs can also be

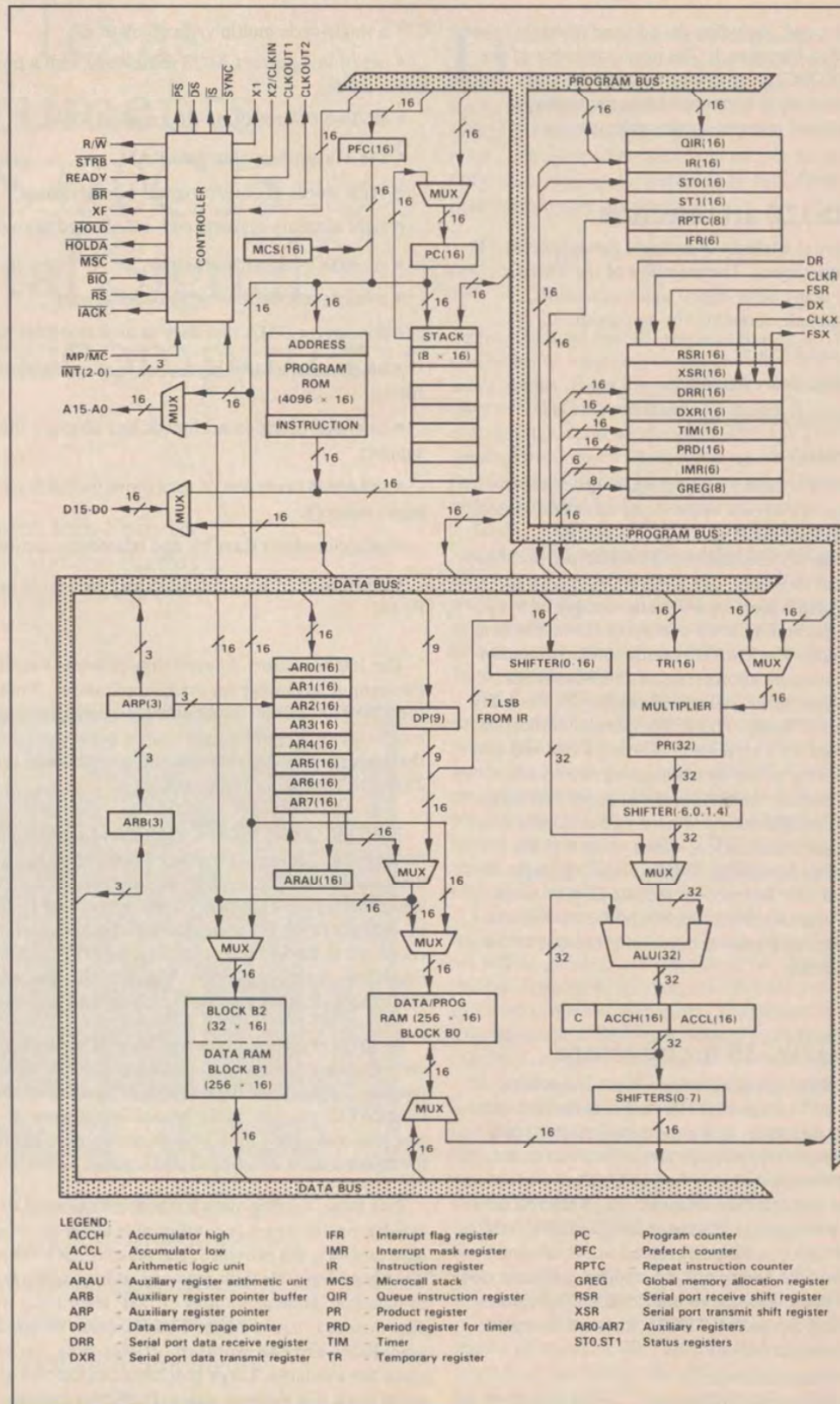


Figure 1.
TMS320C25
block diagram.

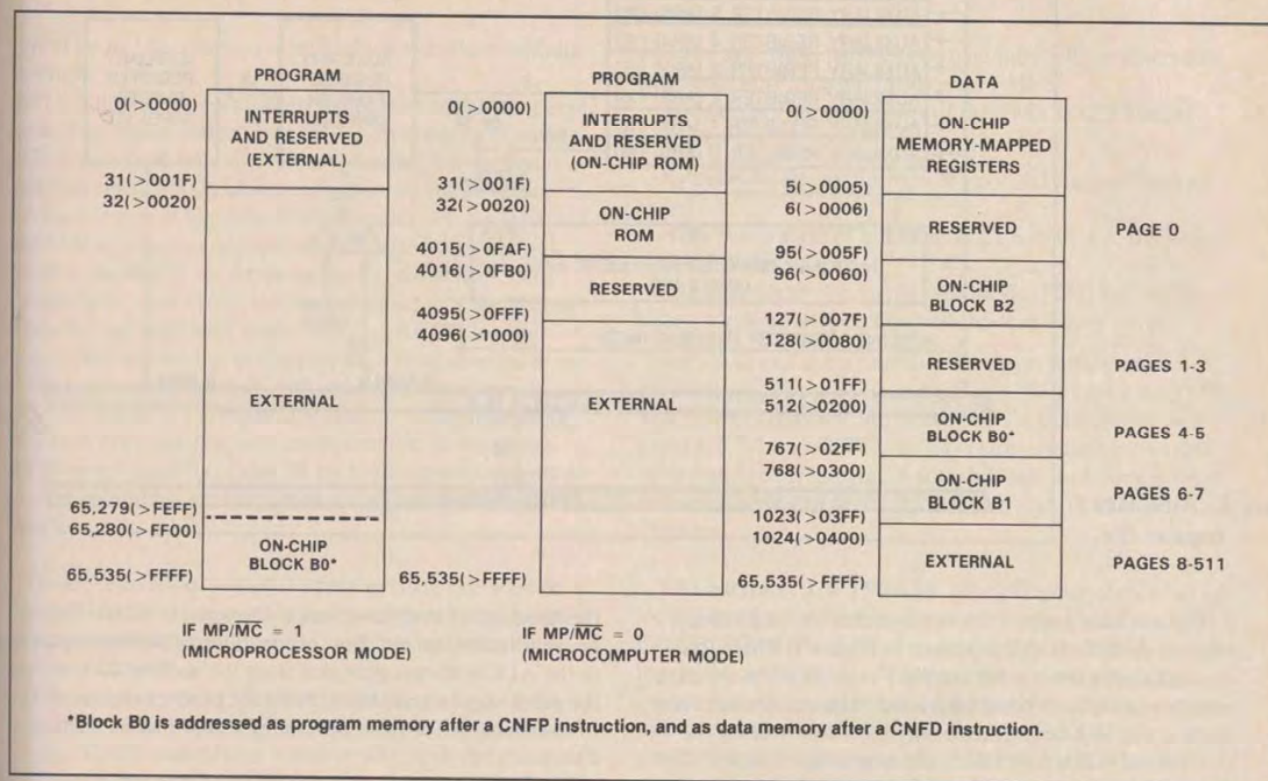


Figure 2. TMS320C25 memory maps.

downloaded from slow external memory to on-chip RAM for full-speed operation.

The TMS320C25 also incorporates a hardware timer and a block data transfer capability.

The diagram of the TMS320C25 in Figure 1 shows the principal blocks and data paths within the processor. It also shows all of the TMS320C25's interface pins.

The TMS320C25's architecture is built around the program and data buses. The program bus carries the instruction code and immediate operands from program memory. The data bus interconnects elements such as the central arithmetic logic unit (CALU) and the auxiliary register file to the data RAM. Together, the program and data buses can carry data from on-chip data RAM and internal or external program memory to the multiplier in a single cycle for multiply/accumulate operations.

A high degree of parallelism exists in the device—for example, while data are being operated on by the CALU, arithmetic operations can be implemented in the auxiliary register arithmetic unit (ARAU). Such parallelism results in a powerful set of arithmetic, logical, and bit-manipulation operations that can be performed in a single machine cycle.

Memory allocation. As mentioned above, the TMS320C25 provides 4K 16-bit words of on-chip program ROM and 544

16-bit words of on-chip data RAM. The RAM is divided into three blocks, B0, B1, and B2. Of the 544 words, 256 words (block B0) are configurable as either data memory or program memory; 288 words (blocks B1 and B2) are always data memory. A data memory size of 544 words allows the TMS320C25 to handle a data array of 512 words but still leaves 32 locations for intermediate storage.

The TMS320C25 maintains separate address spaces for program memory, data memory, and I/O. In addition to blocks B0, B1, and B2, the on-chip data memory map (see Figure 2) includes memory-mapped registers. Six peripheral registers, the serial-port registers (DRR and DXR), timer register (TIM), period register (PRD), interrupt mask register (IMR), and global memory allocation register (GREG), have been mapped into the data memory space so they can be easily modified.

The TMS320C25 has a register file containing eight auxiliary registers that can be used for indirect addressing of data memory or for temporary storage. These registers, AR0-AR7, can be either directly addressed by an instruction or indirectly addressed by a three-bit auxiliary register pointer (ARP). The auxiliary registers and the ARP can be loaded either from data memory or by an immediate operand defined in the instruction. The contents of the registers can also be stored in data memory.

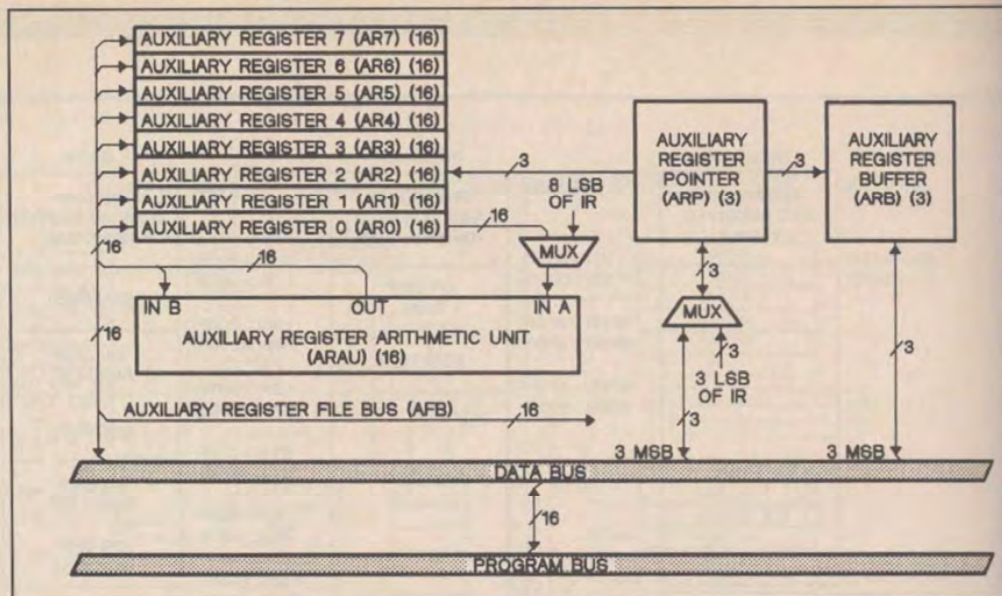


Figure 3. Auxiliary register file.

The auxiliary register file is connected to the auxiliary register arithmetic unit as shown in Figure 3. The ARAU can autoindex the current auxiliary register while the data memory location is being addressed. The current auxiliary register can also be indexed either by $+1/-1$ or by the contents of AR0. As a result, the accessing of tables of information does not require the CALU for address manipulation, thereby freeing it for other operations.

Although the ARAU was designed to support address manipulation in parallel with other operations, it can also serve as an additional general-purpose arithmetic unit since the auxiliary register file can communicate directly with data memory. The ARAU implements 16-bit unsigned arithmetic, whereas the CALU implements 32-bit two's-complement arithmetic. The ARAU also provides branches dependent on the comparison of AR0 to the auxiliary register pointed to by the ARP.

Central arithmetic logic unit. The CALU contains a 16-bit scaling shifter, a 16×16 -bit parallel multiplier, a 32-bit ALU, and a 32-bit accumulator. The scaling shifter has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU. This shifter produces a left shift of 0 to 16 bits on the input data, as programmed in the instruction. The least significant bits of the output are filled with zeroes, and the most significant bits are either filled with zeroes or sign-extended, depending upon the state of the sign-extension mode bit of status register ST1. Additional shifters at the outputs of both the accumulator and the multiplier are suitable for numerical scaling, bit extraction, extended-precision arithmetic, and overflow prevention. Due to the pipelining in the TMS320C25, shifting is accomplished as part of an instruction and thus does not require additional cycles for execution.

The 32-bit ALU and accumulator perform a wide range of arithmetic and logical instructions. An overflow saturation mode permits the accumulator to be loaded with the most positive or negative number (the choice depending on

the direction of overflow), and it allows an overflow flag to be set whenever an overflow occurs. One of the two inputs to the ALU is always provided from the accumulator, and the other may be transferred from the product register (PR) of the multiplier or from the scaling shifter loaded from data memory.

The implementation of a typical ALU instruction requires these steps:

- data are fetched from the Rcarrrn the data bus;
- data are passed through the scaling shifter and through the ALU, where the arithmetic is performed; and
- the result is moved into the accumulator.

The 32-bit accumulator is split into two 16-bit segments for storage in data memory: ACCH (accumulator high) and ACCL (accumulator low). Shifters at the output of the accumulator provide a shift of 0 to 7 places to the left. This shift is performed while the data are being transferred to the data bus for storage. The contents of the accumulator remain unchanged. The accumulator also has an in-place one-bit shift to the left or right (SFL or SFR instruction) and a rotate through carry (ROL or ROR instruction) for shifting its contents.

A carry bit is provided to the accumulator, allowing more efficient extended-precision computation. ADDC (add with carry) and SUBB (subtract with borrow) are two instructions using the carry bit. Branch instructions that use the carry bit are also provided.

Hardware multiplier. The TMS320C25 uses a 16×16 -bit hardware multiplier that can compute a 32-bit product during every machine cycle. Two registers are associated with the multiplier: a 16-bit temporary register (TR) that holds one of the operands for the multiplier, and a 32-bit product register (PR) that holds the product.

The output of the product register can be left-shifted one or four bits. This is useful for implementing fractional arithmetic or justifying fractional products. The output of the PR can also be right-shifted six bits to enable the execu-

tion of up to 128 consecutive multiply/accumulates without overflow.

The multiplier performs both signed and unsigned operations. Two signed instructions, MAC (multiply/accumulate) and MACD (multiply/accumulate and data move), can process both operands simultaneously, thereby fully utilizing the computational bandwidth of the multiplier. For MAC and MACD, the two operands are transferred to the multiplier at each cycle via the program and data buses. This enables MAC and MACD to be performed in a single cycle when they are used with repeat (RPT or RPTK) instructions. The program bus can supply data from internal or external memory (RAM or ROM) and still maintain single-cycle operation. An unsigned multiply (MPYU) instruction facilitates extended-precision multiplication. It multiplies the unsigned contents of the TR by the unsigned contents of the addressed data memory location, and places the result in the PR.

Control operations. Control operations are provided on the TMS320C25 by an on-chip timer, a repeat counter, three external maskable user interrupts, and internal interrupts generated by serial-port operations or by the timer.

A memory-mapped 16-bit timer (TIM) register (a down counter) is continuously clocked by CLKOUT1. A timer interrupt (TINT) is generated whenever the timer decrements to zero. The timer is reloaded with the value contained in the period (PRD) register within the first cycle after it reaches zero so that interrupts may be programmed to occur at regular intervals of $(PRD + 1) \cdot CLKOUT1$ cycles. This feature is useful for control operations and for synchronous sampling of or writing to peripherals.

The repeat counter (RPTC) is loaded with either a data memory value (in the case of the RPT instruction) or an immediate value (in the case of the RPTK instruction). The repeat feature enables a single instruction to be executed up to 256 times. It can be used with instructions such as multiply/accumulates, block moves, I/O transfers, and table read/writes. Those instructions that are normally multicyle are pipelined when the repeat feature is used and effectively become single-cycle instructions. For example, the table read (TBLR) instruction ordinarily takes three or more cycles, but when it is repeated, it becomes a single-cycle instruction.

The three external maskable user interrupts, $\overline{INT}2$ to $\overline{INT}0$, enable external devices to interrupt the processor. Internal interrupts are generated by either the serial port, the timer, or the software interrupt instruction. Interrupts are prioritized, with reset having the highest priority and the serial-port transmit interrupt the lowest.

Serial port. An on-chip serial port provides direct communication with serial devices such as codecs and serial A/D and D/A converters. The serial port's interface requires a minimum of external hardware. The port has two memory-mapped registers—a data transmit register and a data receive register—which can be operated in either an eight-bit byte mode or a 16-bit word mode. The transmit

framing sync pulse can be generated internally or externally. The serial port's maximum speed is 5 MHz.

The primary enhancements of the TMS320C25's serial port are

- double buffering for both receive and transmit operations,
- the elimination of a minimum CLKR/CLKX frequency ($f_{min} = 0$ Hz), and
- the provision of a frame sync mode (FSM) bit, which allows continuous operation with no frame sync pulses.

The FSM is useful for communicating on pulse-code-modulated telephone system highways. As a result the TMS320C25 can communicate directly on PCM highways such as AT&T T-1 and CCITT G.711/712 by counting the transmitted and received bytes in software and performing the instructions needed to set (SFSM) and reset (RFSM) the FSM bit.

I/O interface. The TMS320C25's I/O space consists of 16 input and 16 output ports. These ports provide a full 16-bit parallel I/O interface via the processor's data bus. A single input (IN) or output (OUT) operation typically takes two cycles; however, when executed in the repeat mode, such an operation becomes single-cycle. The TMS320C25 supports a range of system interfacing requirements. As previously mentioned, three separate address spaces—program, data, and I/O—provide interfacing to memory and I/O, thereby maximizing system throughput. The TMS320C25 simplifies I/O design by treating I/O the same way it treats memory. It maps I/O devices into the I/O address space using its external address and data buses in the same way as it uses them for mapping memory devices into memory address space.

The local memory interface consists of a 16-bit parallel data bus (D15-D0), a 16-bit address bus (A15-A0), three pins for data memory, program memory, and I/O space select ($\overline{DS}$, $\overline{PS}$, and $\overline{IS}$, respectively), and various system control signals. The R/W signal controls the direction of a data transfer, and STRB provides a timing signal to control the transfer. When using on-chip program RAM, ROM, or high-speed external program memory, the TMS320C25 runs at full speed without wait states. By using the READY signal, it can generate wait states so it can communicate with slower off-chip memories.

The TMS320C25 supports direct memory access to external program and data memory. Another processor can take complete control of the TMS320C25's external memory by asserting $\overline{HOLD}$ low, causing the TMS320C25 to place its address, data, and control lines in the high-impedance state. Two modes are available on the device. In the first mode, execution is suspended during assertion of $\overline{HOLD}$. In the second mode—the “concurrent DMA mode”—the TMS320C25 continues to execute its program while operating from internal RAM or ROM, thereby greatly increasing throughput in data-intensive applications. Signaling between the external processor and the TMS320C25 can be performed through interrupts.

Table 1.
TMS320C25 instructions.

ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO. WORDS	OPERATION
ABS	Absolute value of accumulator	1	$ ACC \rightarrow ACC$
ADD	Add to accumulator with shift	1	$ACC + [(dma) \times 2^{shift}] \rightarrow ACC$
ADDC [†]	Add to accumulator with carry	1	$ACC + (dma) + (C) \rightarrow ACC$
ADDH	Add to high accumulator	1	$ACC + [(dma) \times 2^{16}] \rightarrow ACC$
ADDK [‡]	Add to accumulator short immediate	1	$ACC + 8\text{-bit constant} \rightarrow ACC$
ADDS	Add to low accumulator with sign extension suppressed	1	$ACC + (dma) \rightarrow ACC$
ADDT [†]	Add to accumulator with shift specified by T register	1	$ACC + [(dma) \times 2^{Treg}] \rightarrow ACC$
ADLK [†]	Add to accumulator long immediate with shift	2	$ACC + [16\text{-bit constant} \times 2^{shift}] \rightarrow ACC$
AND	AND with accumulator	1	$ACC(15:0).AND.(dma) \rightarrow ACC(15:0),$ $0 \rightarrow ACC(31:16)$
ANDK [†]	AND immediate with accumulator with shift	2	$ACC(30:0).AND.[16\text{-bit constant} \times 2^{shift}] \rightarrow$ $ACC(30:0), 0 \rightarrow ACC(30:0)$
CMPL [†]	Complement accumulator	1	$\overline{ACC} \rightarrow ACC$
LAC	Load accumulator with shift	1	$(dma) \times 2^{shift} \rightarrow ACC$
LACK	Load accumulator immediate short	1	8-bit constant $\rightarrow ACC$
LACT [†]	Load accumulator with shift specified by T register	1	$(dma) \times 2^{Treg} \rightarrow ACC$
LALK [†]	Load accumulator long immediate with shift	2	$(16\text{-bit constant}) \times 2^{16} \rightarrow ACC$
NEG [†]	Negate accumulator	1	$(ACC) \rightarrow ACC$
NORM [†]	Normalize contents of accumulator	1	
OR	OR with accumulator	1	$ACC(15:0).OR.(dma) \rightarrow ACC(15:0)$
ORK [†]	OR immediate with accumulator with shift	2	$ACC(30:0).OR.[16\text{-bit constant} \times 2^{shift}] \rightarrow$ $ACC(30:0)$
ROL [‡]	Rotate accumulator left	1	$ACC(30:0) \rightarrow ACC(31:1), (C) \rightarrow ACC(0),$ $ACC(31) \rightarrow C$
ROR [‡]	Rotate accumulator right	1	$ACC(31:1) \rightarrow ACC(30:0), (C) \rightarrow ACC(31),$ $ACC(0) \rightarrow C$
SACH	Store high accumulator with shift	1	$[(ACC) \times 2^{shift}] \rightarrow dma$
SACL	Store low accumulator with shift	1	$[(ACCL) \times 2^{shift}] \rightarrow dma$
SBLK [†]	Subtract from accumulator long immediate with shift	2	$ACC - [16\text{-bit constant} \times 2^{shift}] \rightarrow ACC$
SFL [†]	Shift accumulator left	1	$ACC(30:0) \rightarrow ACC(31:1), 0 \rightarrow ACC(0)$
SFR [†]	Shift accumulator right	1	$ACC(31:1) \rightarrow ACC(30:0), ACC(31) \rightarrow ACC(31)$
SUB	Subtract from accumulator with shift	1	$ACC - [(dma) \times 2^{shift}] \rightarrow ACC$
SUBB [‡]	Subtract from accumulator with borrow	1	$ACC - (dma) - (\overline{C}) \rightarrow ACC$
SUBC	Conditional subtract	1	
SUBH	Subtract from high accumulator	1	$ACC - [(dma) \times 2^{16}] \rightarrow ACC$
SUBK [‡]	Subtract from accumulator short immediate	1	$ACC - 8\text{-bit constant} \rightarrow ACC$
SUBS	Subtract from low accumulator with sign extension suppressed	1	$ACC - (dma) \rightarrow ACC$
SUBT [†]	Subtract from accumulator with shift specified by T register	1	$ACC - [(dma) \times 2^{Treg}] \rightarrow ACC$
XOR	Exclusive-OR with accumulator	1	$ACC(15:0).XOR.(dma) \rightarrow ACC(15:0)$
XORK [†]	Exclusive-OR immediate with accumulator with shift	2	$ACC(30:0).XOR.[16\text{-bit constant} \times 2^{shift}] \rightarrow$ $ACC(30:0)$
ZAC	Zero accumulator	1	$0 \rightarrow ACC$
ZALH	Zero low accumulator and load high accumulator	1	$(dma) \times 2^{16} \rightarrow ACC$
ZALR [‡]	Zero low accumulator and load high accumulator with rounding	1	$(dma) \times 2^{16} + >8000 \rightarrow ACC$
ZALS	Zero accumulator and load low accumulator with sign extension suppressed	1	$(dma) \rightarrow ACCL, 0 \rightarrow ACCH$

[†]These instructions are not included in the TMS32010 instruction set.

[‡]These instructions are not included in the TMS32020 instruction set.

AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	OPERATION
ADRK [‡]	Add to auxiliary register short immediate	1	(ARn) + 8-bit constant → ARn
CMPR [†]	Compare auxiliary register with auxiliary register ARO	1	If ARn CM ARO, then 1 → TC; else 0 → TC
LAR	Load auxiliary register	1	(dma) → (ARn)
LARK	Load auxiliary register short immediate	1	8-bit constant → ARn
LARP	Load auxiliary register pointer	1	3-bit constant → ARP, (ARP) → ARB
LDP	Load data memory page pointer	1	(dma) → DP
LDPK	Load data memory page pointer immediate	1	9-bit constant → DP
LRLK [†]	Load auxiliary register long immediate	2	16-bit constant → ARn
MAR	Modify auxiliary register	1	
SAR	Store auxiliary register	1	(ARn) → dma
SBRK [‡]	Subtract from auxiliary register short immediate	1	(ARn) - 8-bit constant → ARn

T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	OPERATION
APAC	Add P register to accumulator	1	(ACC) + (shift Preg) → ACC
LPH [†]	Load high P register	1	(dma) → Preg (31-16)
LT	Load T register	1	(dma) → Treg
LTA	Load T register and accumulate previous product	1	(dma) → Treg, (ACC) + (shifted Preg) → ACC
LTD	Load T register, accumulate previous product, and move data	1	(dma) → Treg, (dma) → dma + 1, (ACC) + (shifted Preg) → ACC
LTP [†]	Load T register and store P register in accumulator	1	(dma) → Treg, (shifted Preg) → ACC
LTS [†]	Load T register and subtract previous product	1	(dma) → Treg, (ACC) - (shifted Preg) → ACC
MAC [†]	Multiply and accumulate	2	(ACC) + (shifted Preg) → ACC, (pma) × (dma) → Preg
MACD [†]	Multiply and accumulate with data move	2	(ACC) + (shifted Preg) → ACC, (pma) × (dma) → Preg, (dma) → dma + 1
MPY	Multiply (with T register, store product in P register)	1	(Treg) × (dma) → Preg
MPYA [‡]	Multiply and accumulate previous product	1	(ACC) + (shifted Preg) → ACC, (Treg) × (dma) → Preg
MPYK	Multiply immediate	1	(Treg) × 13-bit constant → Preg
MPYS [‡]	Multiply and subtract previous product	1	(ACC) - (shifted Preg) → ACC, (Treg) × (dma) → Preg
MPYU [‡]	Multiply unsigned	1	Usn (Treg) × Usn (dma) → Preg
PAC	Load accumulator with P register	1	(shifted Preg) → ACC
SPAC	Subtract P register from accumulator	1	(ACC) - (shifted Preg) → ACC
SPH [‡]	Store high P register	1	(shifted Preg (31-16)) → dma
SPL [‡]	Store low P register	1	(shifted Preg (15-0)) → dma
SPM [†]	Set P register output shift mode	1	2-bit constant → PM
SQRA [†]	Square and accumulate	1	(ACC) + (shifted Preg) → ACC, (dma) × (dma) → Preg
SQRS [†]	Square and subtract previous product	1	(ACC) - (shifted Preg) → ACC, (dma) × (dma) → Preg

SYMBOL	MEANING
ACC	Accumulator
ARB	Auxiliary register pointer buffer
ARn	Auxiliary Register n (AR0 through AR7 are predefined assembler symbols equal to 0 through 7, respectively)
ARP	Auxiliary register pointer
BIO	Branch control input
C	Carry bit
CM	2-bit field specifying compare mode
CNF	On-chip RAM configuration control bit
dma	Data memory address
DP	Data page pointer
FO	Format status bit
FSM	Frame synchronization mode bit
HM	Hold mode bit
INTM	Interrupt mode flag bit
>nn	Indicates nn is a hexadecimal number. (All others are assumed to be decimal values.)
OV	Overflow flag bit
OVm	Overflow mode bit
P	Product register

SYMBOL	MEANING
PA	Port address. (PA0 through PA15 are predefined assembler symbols equal to 0 through 15, respectively.)
PC	Program counter
PM	2-bit field specifying P register output shift code
pma	Program memory address
Preg	Product register
RPTC	Repeat counter
STn	Status Register n (ST0 or ST1)
SXM	Sign extension mode bit
T	Temporary register
TC	Test control bit
TOS	Top of stack
Treg	Temporary register
TXM	Transmit mode bit
Usn	Unsigned value
XF	XF pin status bit
→	Is assigned to
	An absolute value
[]	Optional items
()	Contents of

Table 1 cont'd

BRANCH/CALL INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO. WORDS	OPERATION
B	Branch unconditionally	2	pma → PC
BACC [†]	Branch to address specified by accumulator	1	(ACC(15-0)) → PC
BANZ	Branch on auxiliary register not zero	2	If (AR(ARP)) ≠ 0, then pma → PC; else (PC) + 2 → PC
BBNZ [†]	Branch if TC bit ≠ 0	2	If (TC) = 1, then pma → PC; else (PC) + 2 → PC
BBZ [†]	Branch if TC bit = 0	2	If (TC) = 0, then pma → PC; else (PC) + 2 → PC
BC [‡]	Branch on carry	2	If (C) = 1, then pma → PC; else (PC) + 2 → PC
BGEZ	Branch if accumulator ≥ 0	2	If (ACC) ≥ 0, then pma → PC; else (PC) + 2 → PC
BGZ	Branch if accumulator > 0	2	If (ACC) > 0, then pma → PC; else (PC) + 2 → PC
BIOZ	Branch on I/O status = 0	2	If (BIO) = 0, then pma → PC; else (PC) + 2 → PC
BLEZ	Branch if accumulator ≤ 0	2	If (ACC) ≤ 0, then pma → PC; else (PC) + 2 → PC
BLZ	Branch if accumulator < 0	2	If (ACC) < 0, then pma → PC; else (PC) + 2 → PC
BNC [‡]	Branch on no carry	2	If (C) = 0, then pma → PC; else (PC) + 2 → PC
BNV [†]	Branch if no overflow	2	If (OV) ≠ 0, then pma → PC; else (PC) + 2 → PC
BNZ	Branch if accumulator ≠ 0	2	If (ACC) ≠ 0, then pma → PC; else (PC) + 2 → PC
BV	Branch on overflow	2	If (OV) = 0, then pma → PC; else (PC) + 2 → PC
BZ	Branch if accumulator = 0	2	If (ACC) = 0, then pma → PC; else (PC) + 2 → PC
CALA	Call subroutine indirect	1	(ACC(15-0)) → PC, (PC) + 1 → TOS
CALL	Call subroutine	2	(PC) + 2 → TOS, pma → PC
RET	Return from subroutine	1	(TOS) → PC
I/O AND DATA MEMORY OPERATIONS			
MNEMONIC	DESCRIPTION	NO. WORDS	OPERATION
BLKD [†]	Block move from data memory to data memory	2	(dma1, addressed by PC) → dma2
BLKP [†]	Block move from program memory to data memory	2	(pma, addressed by PC) → dma
DMOV	Data move in data memory	1	(dma) → dma + 1
FORT [†]	Format serial port registers	1	1-bit constant → FO
IN	Input data from port	1	(data bus, addressed by PA) → dma
OUT	Output data to port	1	(dma) → data bus, addressed by PA
RFSM [‡]	Reset serial port frame synchronization mode	1	0 → FSM
RTXM [†]	Reset serial port transmit mode	1	0 → TXM
RXF [†]	Reset external flag	1	0 → XF
SFSM [‡]	Set serial port frame synchronization mode	1	1 → FSM
STXM [†]	Set serial port transmit mode	1	1 → TXM
SXF [†]	Set external flag	1	1 → XF
TBLR	Table read	1	(pma, addressed by ACC (15-0)) → dma
TBLW	Table write	1	(dma) → pma, addressed by ACC (15-0)
CONTROL INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO. WORDS	OPERATIONS
BIT [†]	Test bit	1	(dma bit at (15-bit code)) → TC
BITT [†]	Test bit specified by T register	1	(dma bit at (15-Treg)) → TC
CNFD [†]	Configure block as data memory	1	0 → CNF
CNFP [†]	Configure block as program memory	1	1 → CNF
DINT	Disable interrupt	1	1 → INTM
EINT	Enable interrupt	1	0 → INTM
IDLE [†]	Idle until interrupt	1	(PC) + 1 → PC, powerdown
LST	Load status register ST0	1	(dma) → ST0
LST1 [†]	Load status register ST1	1	(dma) → ST1

[†]These instructions are not included in the TMS32010 instruction set.

[‡]These instructions are not included in the TMS32020 instruction set.

CONTROL INSTRUCTIONS cont'd

NOP	No operation	1	(PC) + 1 → PC
POP	Pop top of stack to low accumulator	1	(TOS) → ACC
POPD [†]	Pop top of stack to data memory	1	(TOS) → dma
PSHD [†]	Push data memory value onto stack	1	(dma) → TOS
PUSH	Push low accumulator onto stack	1	(ACCL) → TOS
RC [‡]	Reset carry bit	1	0 → C
RHM [‡]	Reset hold mode	1	0 → HM
ROVM	Reset overflow mode	1	0 → OVM
RPT [†]	Repeat instruction as specified by data memory value	1	(dma) → RPTC
RPTK [†]	Repeat instruction as specified by immediate value	1	8-bit constant → RPTC
RSXM [†]	Reset sign-extension mode	1	0 → SXM
RTC [‡]	Reset test/control flag	1	0 → TC
SC [‡]	Set carry bit	1	1 → C
SHM [‡]	Set hold mode	1	1 → HM
SOVM	Set overflow mode	1	1 → OVM
SST	Store status register ST0	1	ST0 → dma
SST1 [†]	Store status register ST1	1	ST1 → dma
SSXM [†]	Set sign-extension mode	1	1 → SXM
STC [‡]	Set test/control flag	1	1 → TC
TRAP [†]	Software interrupt	1	(PC) + 1 → TOS, 30 → PC

The TMS320C25's conditions and modes are stored in two status registers, ST0 and ST1. Instructions are provided to allow these registers to be stored in or loaded from data memory. This capability allows the current status of the device to be saved during interrupts and subroutine calls.

TMS320C25 software

Earlier, we characterized digital signal processing as the real-time processing of mathematically intensive algorithms. This characterization equates to a requirement for high-speed, multiply/accumulate capability in a processor. The performance of a signal processor is therefore measured in terms appropriate to this requirement—that is, it is measured in terms of the speed of execution of individual instructions, the power of the instruction set, and the I/O capabilities. The speed is given as the basic instruction cycle time and the number of cycles required to complete any instruction.

As we noted earlier, pipelining of instruction fetching, decoding, and execution provides an instruction cycle time of only 100 ns. The overwhelming majority of the TMS320C25's instructions (97 out of 133) are executed in a single instruction cycle. Of the 36 instructions requiring additional cycles for execution, 21 involve branches, calls, and returns that result in a reload of the program counter and a break in the execution pipeline. Another seven of

the instructions are two-word, long immediate instructions. The remaining eight—IN, OUT, BLKD, BLKP, TBLR, TBLW, MAC, and MACD—support I/O and transfers of data between memory spaces, or provide for additional parallel operation in the processor. Furthermore, these eight instructions become single-cycle when used in conjunction with the repeat counter. The instruction set of the TMS320C25 exploits the parallelism of the processor, allowing complex or numerically intensive computations to be implemented in relatively few instructions. Table 1 lists the TMS320C25's instructions.

Addressing modes. Most TMS320C25 instructions are coded in a single 16-bit word—the reason most can be executed in a single cycle. The 16-bit word comprises an eight-bit opcode and an eight-bit address. Three memory addressing modes are available: direct, indirect, and immediate (Table 2). Both direct and indirect addressing are used to access data memory. Immediate addressing uses the contents of the memory addressed by the program counter. Figure 4 illustrates operand addressing in the direct, indirect, and immediate modes.

In direct addressing, seven bits of the instruction word are concatenated with the nine-bit data memory page pointer (DP) to form the 16-bit data memory address. The DP register points to one of 512 possible data memory pages, each 128 word in length, to obtain a 64K total data memory space. The seven-bit address in the instruction

Table 2.
Addressing modes.

ADDRESSING MODE	OPERATION
OP A	Direct addressing
OP * (.NARP)	Indirect; no change to AR
OP *+ (.NARP)	Indirect; current AR is incremented.
OP *- (.NARP)	Indirect; current AR is decremented.
OP *O+ (.NARP)	Indirect; AR0 is added to current AR.
OP *O- (.NARP)	Indirect; AR0 is subtracted from current AR.
OP *BR0+ (.NARP)	Indirect; AR0 is added to current AR (with reverse carry propagation)
OP *BR0- (.NARP)	Indirect; AR0 is subtracted from current AR (with reverse carry propagation)

NOTE: The optional NARP field specifies a new value of the ARP.

points to the specific location within the data memory page.

Indirect addressing is provided by the eight auxiliary registers AR0-AR7. These registers can be used to indirectly address data memory, as loop counters, or for temporary data storage. Indirect auxiliary register addressing (Figure 5) allows placement of the data memory address of an instruction operand into one of the eight auxiliary registers. These registers are pointed to by a three-bit auxiliary register pointer (ARP) that is loaded with a value from 0 through 7 designating AR0 through AR7, respectively. The auxiliary registers and the ARP may be loaded either from data memory or by an immediate operand defined in the instruction. Furthermore, the contents of the auxiliary registers may be stored in data memory.

There are seven types of indirect addressing (see Table 2 again):

- indexing with increment,
- indexing with decrement,
- indexing by adding the contents of AR0,
- indexing by subtracting the contents of AR0,
- indexing by adding the contents of AR0 with the carry propagation reversed (for bit-reversing an FFT),
- indexing by subtracting the contents of AR0 with the carry propagation reversed (also for bit-reversing an FFT), and
- no indexing.

All indexing operations are performed on the current auxiliary register in the same cycle as the original instruction, with loading of a new ARP value available as an option. The operations performed in the ARAU can even be performed during branch instruction execution, allowing efficient control with conditional looping.

Bit-reversed indexed addressing modes allow efficient I/O to be performed for the resequencing of data points in a radix-2 FFT program. The direction of carry propagation in the ARAU is reversed when this mode is selected, and AR0 is added to or subtracted from the current auxiliary register.

In immediate addressing, the instruction word contains the value of the immediate operand. Both single-word (8-bit and 13-bit constant) short immediate instructions and two-word (16-bit constant) long immediate instructions are included in the instruction set. In the case of long immediate

instructions, the word following the instruction opcode is used as the immediate operand. MPYK is an example of an immediate instruction; it multiplies the contents of the T register by a signed 13-bit constant. Seventeen immediate operand instructions are included in the instruction set (see Table 1 again).

Instruction set parallelism—an example. The MACD (multiply/accumulate and data move) instruction serves as an informative example of the parallelism designed into the TMS320C25 instruction set as well as into the TMS320C25 architecture. As shown in Equation 1, the requirement for parallelism exists in common DSP operations such as convolution and filtering.^{6,7}

Parallelism in the execution of instructions enables a complete multiply/accumulate/data move operation to be completed in a single 100-ns instruction cycle. The execution of the MACD involves the following steps:

- 1) The contents of the 32-bit P register are shifted (scaled) by an output shifter.
- 2) The 32-bit ALU accumulates the shifted result of the 32-bit P register with the current contents of the 32-bit accumulator.
- 3) The 16-bit contents of a data memory location (usually addressed indirectly via one of the auxiliary registers) are loaded into the T register.
- 4) The 16-bit contents of a program memory location (addressed via the prefetch counter PFC) are introduced to the multiplier and a 16×16 -bit multiply is executed, resulting in a new 32-bit product. The product is placed in the P register to be accumulated during the next cycle.
- 5) The 16-bit contents of the data memory location are copied to the next higher data memory address.
- 6) The carry and overflow status bits are set, as appropriate, in the status registers.
- 7) The 16-bit contents of the auxiliary register pointed to by the ARP are modified (typically decremented) in preparation for the use of the data memory address on the next cycle.
- 8) The 16-bit contents of the PFC are incremented in preparation for the use of the program memory address on the next cycle.
- 9) The repeat counter is decremented.

As can be seen from the above, one of the data values is taken from data memory while the other is taken from program memory. A single-cycle execution and data move is accomplished when the data memory being addressed is the on-chip data memory. The program memory location can be either on or off chip and, if on chip, can come from either ROM or the reconfigurable memory block B0.

Parallel operation of certain subsets of TMS320C25 functions is also available. These subsets include loading the T register in combination with addition (LTA), subtraction (LTS), or a move of the P register's contents to the accumulator (LTP). The accumulation can be supplemented by the data move function (LTD). Another combination (MPYA/MPYS) provides the accumulation of the previous

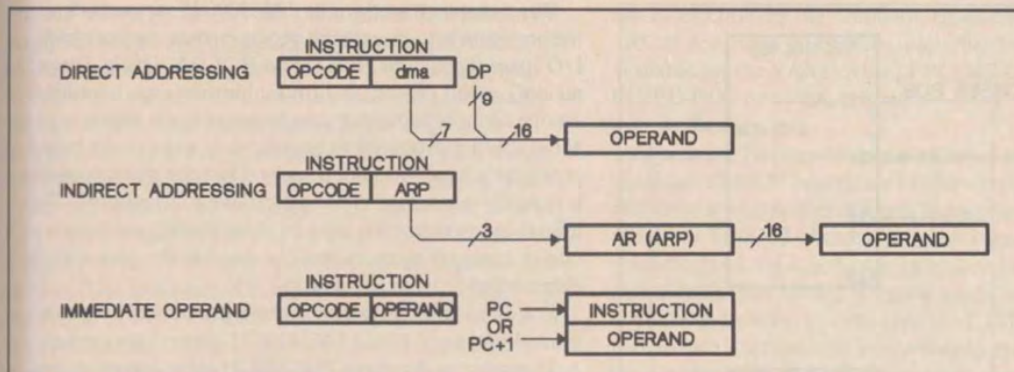


Figure 4. Methods of addressing the instruction operand.

product along with the execution of the multiplier to generate a new product. This combination is particularly useful in adaptive filtering techniques such as those embodied in the least-mean-square (LMS) algorithm.^{4,15} The implementation of an adaptive filter by means of these instructions will be described in detail in the section on applications.

Block moves. The TMS320C25 provides six instructions for data and program block moves and transfers of data via the I/O ports. When these instructions are pipelined by means of the repeat instruction, significantly higher throughput is achieved—the pipelining results in a transfer rate of 160 million bits per second.

The BLKD instruction moves a block within data memory, and the BLKP instruction moves a block from program memory to data memory. Block transfers between program and data memory spaces can also be implemented with the TBLR and TBLW (table read and table write) instructions. The advantages of TBLR and TBLW are that they allow the source address as well as the destination address to be determined during programming and that they permit the data to be transferred from data memory to program memory. The IN and OUT instructions permit data to be transferred between the I/O and data memory spaces. While the source address is determined by the prefetch counter, which is incremented on every cycle, the destination address is determined by an auxiliary register whose contents can be modified in any of the previously specified ways. This permits sequential and contiguous data placement (*+, *−), sequential but noncontiguous data placement (*0+, *0−), or scrambled data placement (*BR0+, *BR0−). The value of these address modifications during block data transfers becomes particularly apparent in the use of indexing with reverse-carry propagation to set up the data block in an FFT. The result is not only a savings in execution time but a savings in program memory space as well.

Floating-point support. The TMS320C25 supports floating-point operations for applications requiring a large dynamic range. The NORM (normalization) instruction normalizes fixed-point numbers contained in the accumulator

or by performing left shifts. The LACT (load accumulator with shift specified by the T register) instruction denormalizes a floating-point number by arithmetically left-shifting the mantissa through the input scaling shifter. The shift count, in this case, is the value of the exponent specified by the four low-order bits of the T register. ADDT and SUBT instructions (add to/subtract from accumulator with shift specified by the T register) have been provided to allow additional arithmetic operations.

TMS320C25 hardware

The most important task for a hardware designer is interfacing the DSP device to the rest of the system as inexpensively as possible. Here, we will discuss the TMS320C25's interfacing capabilities.

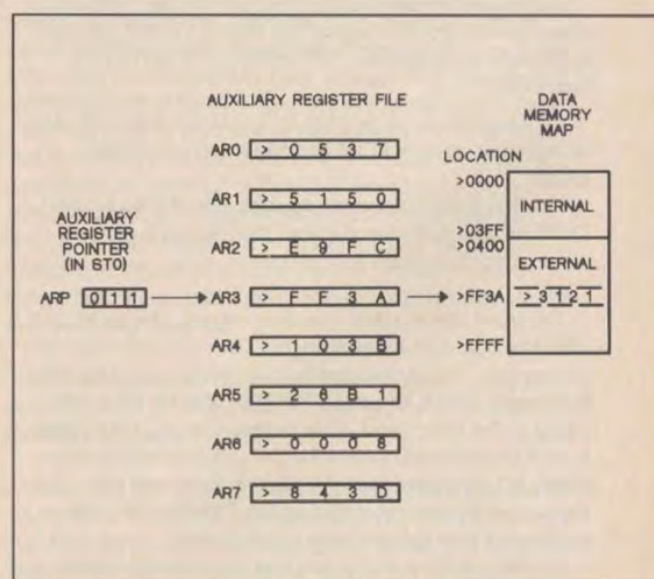


Figure 5. Example of indirect auxiliary register addressing.

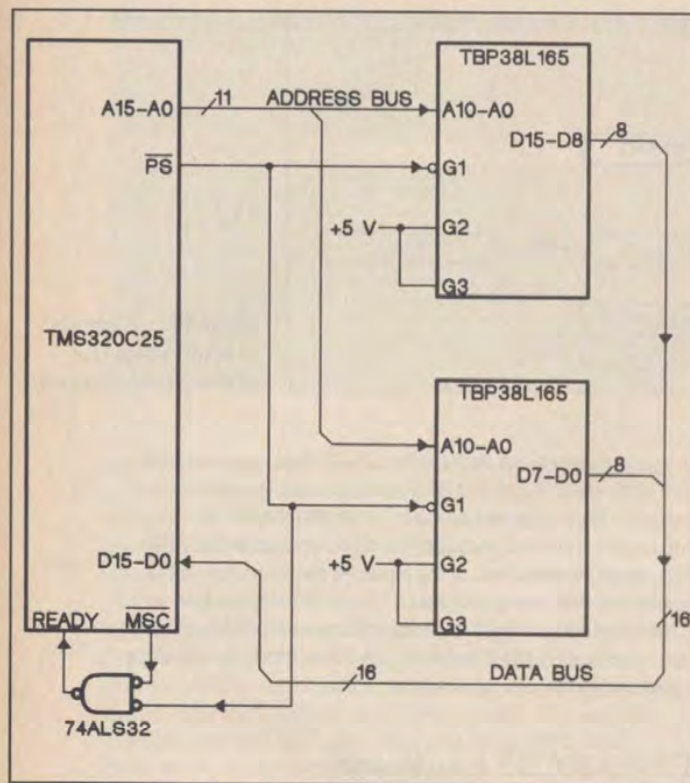


Figure 6. Minimal configuration for external program memory.

System configurations. The flexibility of the TMS320C25 allows systems configurations that satisfy a broad range of application requirements. The TMS320C25 can be configured as

- a stand-alone system (that is, as a single processor using 4K words of on-chip ROM and 544 words of on-chip RAM),
- part of a parallel multiprocessing system (two or more TMS320C25s) with shared global data memory, or
- a coprocessor for a host processor.

The stand-alone system interface consists of a 16-bit parallel data bus, a 16-bit address bus, three pins for memory space select, and various system control signals. In Figure 6, an external data RAM and a PROM/EPROM have been added to the basic stand-alone system. The READY signal is used for wait-state generation for communicating with slower off-chip memories. All the memories and I/O devices are directly controlled by the TMS320C25, thus minimizing external hardware requirements.

Parallel multiprocessing and host/coprocessor systems take advantage of the TMS320C25's direct memory access and global memory configuration capabilities.

Direct memory access. The TMS320C25 supports direct memory access to its external program/data memory and I/O space through its $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$ signals. Direct memory access can be used for multiprocessing: Execution on one or more processors can be temporarily halted to allow another processor to read from or write to the halted processor's local off-chip memory. Here the multiprocessing is typically performed in a master/slave configuration. The master can initialize the slave by downloading a program into its program memory space or provide the slave with the data needed to complete a task.

In a direct memory access scheme, the master may be a general-purpose CPU, a TMS320C25, or perhaps even an A/D converter. A master TMS320C25 takes complete control of the slave's external memory by asserting $\overline{\text{HOLD}}$ low through its external flag (XF). This causes the slave to place its address, data, and control lines in a high-impedance state. By asserting $\overline{\text{RS}}$ in conjunction with $\overline{\text{HOLD}}$, the master processor can load the slave's local program memory with the necessary initialization code on reset or power-up. The two processors can be synchronized through use of the SYNC pin to make the transfer over the memory bus faster and more efficient.

After control of the slave's buses is given to the master processor, the slave alerts the master by asserting $\overline{\text{HOLDA}}$. This signal can be tied to the master's $\overline{\text{BIO}}$ pin. The slave's XF pin can be used to indicate to the master when the slave has finished performing its task and needs to be reprogrammed or given additional data to continue processing. In a multiple-slave configuration, the priority of each slave's task can be determined by tying the slave's XF signals to the appropriate $\overline{\text{INT}}$ pin on the master.

A PC environment provides an example of a direct memory access scheme in which the system bus is used for data transfer. In this configuration, either the master CPU or a disk controller may place data on the system bus for downloading into the local memory of the TMS320C25. Here the TMS320C25 acts like a peripheral processor with multifunction capability. In a speech application, for example, the master can load the TMS320C25's program memory with algorithms to perform tasks such as speech analysis, synthesis, or recognition, and its data memory with the required speech templates. In a graphics application, the TMS320C25 can serve as a dedicated graphics engine. Programs can be stored in ROM or downloaded via the system bus into program RAM. Again, data can come from PC disk storage or be provided directly by the master CPU. In this configuration, decode and arbitration logic is used to control the direct memory access. When the address on the system bus resides in the local memory of the peripheral TMS320C25, this logic asserts the $\overline{\text{HOLD}}$ signal while sending the master a not-ready indication to allow wait states. After the TMS320C25 acknowledges the direct memory access by asserting $\overline{\text{HOLDA}}$, READY is asserted and the information is transferred.

Global memory. In some digital signal processing tasks, the algorithm being implemented can be divided into sections and a processor dedicated to each. In this case, the

first and second processors can share global data memory, as can the second and third, the third and fourth, and so on. Arbitration logic may be required to determine which section of the algorithm will execute and which processor will have access to the global memory. The dedication of each processor to a distinct section of the algorithm makes pipelined execution—and thus higher throughput—possible.

External memory can be divided into global and local sections. Special registers and pins on the TMS320C25 allow multiple processors to share up to 32K words of global data memory. This facilitates efficient "shared data" multiprocessing, in which data are transferred between two or more processors. Unlike a direct memory access scheme, reading or writing global memory does not require one of the processors to be halted.

TMS320C25 development tools and support

A digital signal processor is essentially an application-specific microprocessor or microcomputer. Like any microprocessor, it needs good development tools and technical support—no matter how impressive its performance or how easy its interfacing to other devices, it cannot be easily designed into systems without such tools and support. In developing an application, a designer encounters problems

can be executed by the simulator, emulator, or the TMS320C25 processor. The macro assembler/linker is currently available for the VAX/VMS, TI PC/MS-DOS, and IBM PC/PC-DOS operating systems.

Simulator. The simulator is a software program that simulates TMS320 operations to allow program verification. Its debug mode enables the user to monitor the state of the simulated TMS320 while his program is executing. The simulator uses the object code produced by the macro assembler/linker. During program execution, the internal registers and memory of the simulated TMS320 are modified as each instruction is interpreted by the host computer. Once program execution is suspended, the internal registers and the program and data memories can be inspected and modified. The simulator is currently available for the VAX/VMS, TI PC/MS-DOS, and IBM PC/PC-DOS operating systems.

Hardware tools. Tools are provided for in-circuit emulation and hardware program debugging such as breakpointing and tracing so that DSP algorithms can be developed and tested in a real-product environment.

Evaluation module. The evaluation module, or EVM, is a stand-alone board that contains all the hardware tools

No matter how impressive its performance or how easy its interfacing to other devices, a digital signal processor cannot be designed into systems without good development tools and vendor support.

and needs to ask questions. Often the tools and vendor support given him are the difference between the success and failure of his project.

The TMS320C25 is supported by many development tools.¹⁶ These tools range from inexpensive modules for application evaluation and benchmarking to an assembler/linker and software simulator to a full-capability hardware emulator.

Software tools. An assembler/linker and software simulator that enable users to develop and debug TMS320 DSP algorithms are available for the TI PC, IBM PC, and VAX.

Assembler/linker. The macro assembler translates assembly language source code into executable object code. It allows the programmer to work with mnemonics rather than hexadecimal machine instructions and to reference memory locations with symbolic addresses. It supports macro calls and definitions along with conditional assembly. The linker permits a program to be designed and implemented in separate modules that are later linked to form the complete program. The linker resolves external definitions and references for relocatable code, creating an object file that

needed to evaluate the TMS320C25 and that provides in-circuit emulation of it. The EVM's firmware package contains a debug monitor, an editor, an assembler, a reverse assembler, and software communication to two EIA ports. These ports allow the EVM to be connected to a terminal and to either a host computer or a line printer. The EVM accepts either source or object code downloaded from the host computer. Its resident assembler converts incoming source text into executable code in just one pass by automatically resolving labels after the first assembly pass is completed. When a session is finished, code is saved via the host computer interface.

Software development system. The SWDS is a plug-in card for the TI PC and IBM PC that provides the same functionality as the EVM.

Emulator. The XDS (Extended Development System) is an emulator providing full-speed in-circuit emulation with real-time hardware breakpointing and tracing and program execution capability from target memory. The XDS allows integration of hardware and software modules in the debug mode. By setting breakpoints based on internal conditions

or external events, the XDS user can suspend execution of the program and give control to the debug mode. In the debug mode, he can inspect and modify all registers and memory locations. Single-step execution is available. Full-trace capabilities at full speed and a reverse assembler that translates machine code back into assembly instructions also increase debugging productivity. The XDS system is designed to interface with either a terminal or a host computer. Object code generated by the assembler/linker can be downloaded to the XDS and then controlled through a terminal.

Analog interface board. The AIB is an analog-to-digital (A/D) and digital-to-analog (D/A) conversion board that can be used in conjunction with the EVM or XDS. It can also be used in an educational environment to help familiarize the user with real-world digital signal processing techniques. The AIB includes A/D and D/A converters with 12-bit resolution as well as antialiasing and smoothing filters that have a cut-off frequency programmable from 4.7 kHz to 20 kHz.

In addition to the above design tools, development support includes

- the Digital Filter Design Package, which runs on both TI and IBM PCs and which allows the user to design digital filters (low-pass, high-pass, band-pass, and band-stop types) using a menu-driven approach,
- TI Regional Technology Centers staffed with qualified engineers who provide technical support and design services,
- access to third parties with DSP expertise in various application areas,
- a series of DSP books covering DSP theory, algorithms, and applications and TMS320 implementations,^{4,5,7}
- documentation such as user's guides,¹⁰⁻¹² data sheets, a development support reference guide,¹⁶ and comprehensive application reports,⁴ and
- a technical support hotline and a bulletin board service.

TMS320C25 applications

The TMS320C25 is designed for real-time DSP and other computation-intensive tasks in telecommunications, graphics, image processing, high-speed control, speech processing, instrumentation, and numeric processing. In these applications, the TMS320C25 provides an excellent means for executing signal processing algorithms such as fast Fourier transforms (FFTs), digital filters, frequency synthesizers, correlators, and convolution routines. It can also execute general-purpose functions since it includes bit-manipulation instructions, block data move capabilities, large program and data memory address spaces, and flexible memory mapping.

Since digital filters are used in so many DSP applications, let us examine them as a prelude to our discussion of TMS320C25 applications.

Digital filtering. Filters are often implemented in digital signal processing systems. Such filters fall into two categories: finite impulse response (FIR) filters and infinite impulse response (IIR) filters.^{4,6} For both types of filter, the coefficients of the filter (weighting factors) may be fixed or adapted during the course of the signal processing. The TMS320C25 reduces the execution time of all filters by virtue of its 100-ns instruction cycle time and optimized instructions for filter operations.

As we stated earlier, the FIR filter is simply the sum of products in a sampled data system (see Equation 1 again). A simple implementation of the FIR filter uses the MACD instruction (multiply/accumulate and data move) for each filter tap and the RPT/RPTK instruction to repeat the MACD for each tap. Thus, a 256-tap FIR filter can be implemented as

```
RPTK 255
MACD *- ,COEFFP
```

Here, the coefficients can be stored anywhere in program memory (in the reconfigurable on-chip RAM, in the on-chip ROM, or in external memories). When the coefficients are stored in on-chip ROM or externally, the entire on-chip data RAM can be used to store the sample sequence. This allows filters of up to 512 taps to be implemented. Execution of the filter will be at full speed, or 100 ns per tap, as long as the memory (either on-chip RAM or high-speed external RAM) supports full-speed execution.

Up to this point, we have assumed that the filter coefficients are fixed from sample to sample. If the coefficients are adapted or updated with time, as they are in adaptive filters for echo cancellation,^{4,15} the DSP algorithm requires a greater computational capacity from the processor. To adapt or update the coefficients, usually with each sample, the TMS320C25 uses three instructions—multiply and add/subtract previous product to/from accumulator (MPYA/MPYS), zero-out low-order accumulator bits and load high-order accumulator bits with data (ZALR), and store high-order bits of accumulator to data memory (SACH). The method it uses to adapt the coefficients is the least-mean-square, or LMS, algorithm, which can be expressed as

$$b_k(i+1) = b_k(i) + 2B [e(i) \cdot x(i-k)], \quad (2)$$

where $b_k(i+1)$ is the weighting coefficient for the next sample period, $b_k(i)$ is the weighting coefficient for the present sample period, B is the gain factor or adaptation step size, $e(i)$ is the error function, and $x(i-k)$ is the input of the filter.

In an adaptive filter, the coefficients $b_k(i)$ must be updated to minimize the error function $e(i)$, which is the difference between the output of the filter and a reference signal. Quantization errors arising during coefficient updating can strongly affect the performance of the filter, but these errors can be minimized if the updated values are obtained by rounding rather than truncating. For each coefficient in the filter at a given point in time, the factor

$2*B*e(i)$ is a constant. This factor can be computed once and stored in the T register for each of the updates. This reduces the computational requirement to one multiply/accumulate plus rounding. Without the new instructions, the adaptation of each coefficient would take five instructions corresponding to five clock cycles, as the following instruction sequence shows:

```

LRLK AR2,COEFFD ; LOAD ADDRESS OF COEFFICIENTS.
LRLK AR3,LASTAP ; LOAD ADDRESS OF DATA SAMPLES.
LARP AR2
LT ERRF ; errf = 2*B*e(i)
*
*
ZALH *,AR3 ; ACC = bk(i)*2**16
ADD ONE,15 ; ACC = bk(i)*2**16 + 2**15
MPY *,AR2
APAC ; ACC = bk(i)*2**16
+ errf*x(i-k) + 2**15
SACH *+ ; SAVE bk(i+1).
*
*

```

When the MPYA and ZALR instructions are used, the adaptation reduces to three instructions corresponding to three clock cycles, as shown below:

```

LRLK AR2,COEFFD ; LOAD ADDRESS OF COEFFICIENTS.
LRLK AR3,LASTAP ; LOAD ADDRESS OF DATA SAMPLES.
LARP AR2
LT ERRF ; errf = 2*B*e(i)
*
*
ZALR *,AR3 ; ACC = bk(i)*2**16 + 2**15
MPYA *,AR2 ; ACC = bk(i)*2**16
+ errf*x(i-k) + 2**15
; PREG = errf*x(i-k+1)
SACH *+ ; SAVE bk(i+1).
*
*

```

Note that the processing order has been slightly changed to incorporate the use of the MPYA instruction. This is due to the fact that the accumulation performed by the MPYA is the accumulation of the previous product.

We have now seen the basic code for a FIR filter tap and a coefficient update. Figure 7 shows a routine to filter a signal and update the coefficients for a 256-tap adaptive FIR filter. Note that for each tap one instruction cycle is needed to perform the FIR filter (i.e., to execute a MACD), three instruction cycles are needed to update the filter coefficients, and 33 instruction cycles are needed for overhead. Therefore, the total number of execution cycles needed for the routine is $33 + 4n$, where n is the filter length. Also, note that data memory and program memory requirements are $5 + 2n$ and $30 + 3n$ words, respectively. For adaptive filters, the filter length is restricted by both execution time and memory. There is obviously more processing to be completed per sample due to the adaptation, and the adaptation

```

TITL 'ADAPTIVE FILTER'
DEF ADPFIR
DEF X,Y
*
* THIS 256-TAP ADAPTIVE FIR FILTER USES ON-CHIP MEMORY BLOCK
* B0 FOR COEFFICIENTS AND BLOCK B1 FOR DATA SAMPLES. THE
* NEWEST INPUT SHOULD BE IN MEMORY LOCATION X WHEN CALLED.
* THE OUTPUT WILL BE IN MEMORY LOCATION Y WHEN RETURNED.
* ASSUME THAT THE DATA PAGE IS 0 WHEN THE ROUTINE IS CALLED.
*
COEFFP EQU >FF00 ; B0 PROGRAM MEMORY ADDRESS
COEFFD EQU >0200 ; B0 DATA MEMORY ADDRESS
*
ONE EQU >7A ; CONSTANT ONE (DP=0)
BETA EQU >7B ; ADAPTATION CONSTANT (DP=0)
ERR EQU >7C ; SIGNAL ERROR (DP=0)
ERRF EQU >7D ; ERROR FUNCTION (DP=0)
Y EQU >7E ; FILTER OUTPUT (DP=0)
X EQU >7F ; NEWEST DATA SAMPLE (DP=0)
FRSTAP EQU >0300 ; NEXT NEWEST DATA SAMPLE
LASTAP EQU >03FF ; OLDEST DATA SAMPLE
*
* FINITE IMPULSE RESPONSE (FIR) FILTER.
*
ADPFIR CNFP ; CONFIGURE B0 AS PROGRAM:
MPYK 0 ; Clear the P register.
LAC ONE,14 ; Load output rounding bit.
LARP AR3
LRLK AR3,LASTAP ; Point to the oldest sample.
FIR RPTK 255 ; 256-tap FIR filter.
MACD COEFFP,*- ; CONFIGURE B0 AS DATA:
CNFD
APAC
SACH Y,1 ; Store the filter output.
NEG
ADD X,15 ; Add the newest input.
SACH ERR,1 ; err(i) = x(i) - y(i)
*
* LMS ADAPTATION OF FILTER COEFFICIENTS.
*
LT ERR
MPY BETA
PAC ; errf(i) = beta * err(i)
ADD ONE,14 ; ROUND THE RESULT.
SACH ERRF,1
*
MAR *+
LAC X ; INCLUDE NEWEST SAMPLE.
SACL *
*
LRLK AR2,COEFFD ; POINT TO THE COEFFICIENTS.
LRLK AR3,LASTAP ; POINT TO THE DATA SAMPLES.
LT ERRF
MPY *,AR2 ; P = 2*beta*err(i)*x(i-255)
*
ADAPT ZALR *,AR3 ; LOAD ACCH WITH b255(i) & ROUND.
MPYA *,AR2 ; b255(i+1) = b255(i) + P
; P = 2*beta*err(i)*x(i-254)
SACH *+ ; STORE b255(i+1).
*
ZALR *,AR3 ; LOAD ACCH WITH b254(i) & ROUND.
MPYA *,AR2 ; b254(i+1) = b254(i) + P
; P = 2*beta*err(i)*x(i-253)
SACH *+ ; STORE b254(i+1).
*
ZALR *,AR3 ; LOAD ACCH WITH b253(i) & ROUND.
MPYA *,AR2 ; b253(i+1) = b253(i) + P
; P = 2*beta*err(i)*x(i-252)
SACH *+ ; STORE b253(i+1).
*
*
ZALR *,AR3 ; LOAD ACCH WITH b1(i) & ROUND.
MPYA *,AR2 ; b1(i+1) = b1(i) + P
; P = 2*beta*err(i)*x(i-0)
SACH *+ ; STORE b1(i+1).
*
ZALR *,AR3 ; LOAD ACCH WITH b0(i) & ROUND.
APAC ; b0(i+1) = b0(i) + P
SACH *+ ; STORE b0(i+1).
*
RET ; RETURN TO CALLING ROUTINE.

```

Figure 7. 256-tap adaptive FIR filter routine.

itself dictates that the coefficients be stored in the reconfigurable block of on-chip RAM. Thus, an adaptive filter with no external data memory is limited to 256 taps.

Telecommunications applications. Digital signal processing will be more extensively used in telecommunications as it evolves toward all-digital networks.¹⁷ Below, we discuss several typical uses of the TMS320C25 in telecommunications applications.

Echo cancellation. In echo cancellation, an adaptive FIR filter performs the modeling routine and signal modifications needed to adaptively cancel the echo caused by impedance mismatches in telephone transmission lines. The TMS320C25's large on-chip RAM of 544 words and on-chip ROM of 4K words allow it to execute a 256-tap adaptive filter (32-ms echo cancellation) without external data or program memory.

High-speed modems. For high-speed modems, the TMS320C25 can perform functions such as modulation and demodulation, adaptive equalization, and echo cancellation.^{18,19}

Voice coding. Voice-coding techniques such as full-duplex, 32,000-bit-per-second adaptive differential pulse-code modulation (CCITT G.721), CVSD, 16,000-bit-per-second subband coding, and linear predictive coding are frequently used in voice transmission and storage. The speed of the TMS320C25 in performing arithmetic and its normalization and bit-manipulation capabilities enable it to implement these functions, usually within itself (i.e., with no external devices).

Graphics and image processing applications. In these applications, a signal processor's ability to interface with a host processor is important. The TMS320C25 multiprocessor interface enables it to be used in a variety of host/coprocessor configurations. Graphics and image processing applications can use the TMS320C25's large directly addressable external data space and global memory capability to allow graphical images in memory to be shared with a host processor, thus minimizing data transfers. The TMS320C25's indexed indirect addressing modes allow matrices to be processed row by row when matrix multiplication is performed for 3-D image rotation, translation, and scaling.

High-speed control applications. These applications use the TMS320C25's general-purpose features for bit-test and logical operations, timing synchronization, and fast data transfers (10 million 16-bit words per second). They use the TMS320C25 in closed-loop systems for control signal conditioning, filtering, high-speed computing, and multichannel multiplexing. The following examples demonstrate typical control applications.

Disk control. In disk drives, a closed-loop actuation mechanism positions the read/write heads over the disk surface. Accurate positioning requires various signal conditioning tasks to be performed. The TMS320C25 can replace costly bit-slice, custom, and analog solutions in performing such tasks as compensation, filtering, and fine/coarse tuning.

Robotics. The TMS320C25's digital signal processing and bit-manipulation power, coupled with its host interface, allow it to be useful in robotics control. The TMS320C25 can replace both the digital controllers and the analog signal processing hardware a robot needs to communicate to a central host processor, and it can perform the numerically intensive control functions typical of robotic applications.

Instrumentation. Instruments such as spectrum analyzers often require a large data memory space and a processor capable of performing long-length FFTs and generating high-precision functions with minimal external hardware. The TMS320C25 fulfills these requirements.

Numeric processing applications. Numeric and array processing applications benefit from the TMS320C25's performance. The device's high throughput and its multiprocessing and data memory expansion capabilities make it a low-cost, easy-to-use replacement for a typical bit-slice array processor.

Benchmarks. The TMS320C25 has demonstrated impressive performance of benchmarks representing common DSP routines and applications. Table 3 shows this performance.

The TMS320C25 digital signal processor is the newest member of the TMS320 family. It is a pin-compatible, CMOS version of the TMS32020 but offers several enhancements of that device—a 100-ns instruction cycle time, 4K words of on-chip masked ROM, eight auxiliary registers, an eight-level hardware stack, and a double-buffered serial port. It also enhances the TMS32020 instruction set to support adaptive filtering, extended-precision arithmetic, bit-reversed addressing, and faster I/O.

The TMS320C25's multiprocessor capability, large memory spaces, and general-purpose features allow it to be used in a variety of systems, including ones currently employing costly bit-slice processors or custom ICs.

Table 3.
TMS320C25 benchmarks.

DSP ROUTINES/APPLICATIONS	PERFORMANCE
FIR filter tap	100 ns per tap
256-tap FIR filter sample rate	37 kHz
LMS adaptive FIR filter tap	400 ns per tap
256-tap adaptive FIR filter sample rate	9.5 kHz
Biquad filter element	1 μ s
Echo canceller	32 ms per single chip (with internal memory)
32,000-bit/s CCITT ADPCM	1 channel full-duplex, single-chip (with internal memory)
16,000-bit/s subband coding	2 channels full-duplex, single-chip (with 0.5K external data memory)
2400-bit/s LPC-10 coding	2 channels full-duplex, single-chip (with 2K external data memory)

References

1. L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
2. A. V. Oppenheim, ed., *Applications of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1978.
3. L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, Englewood Cliffs, N.J., 1978.
4. *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments Inc., 1986.
5. R. Morris, *Digital Signal Processing Software*, DSPS Inc., Ottawa, Ont., 1983.
6. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
7. C. Burrus and T. Parks, *DFT/FFT and Convolution Algorithms*, John Wiley & Sons, New York, 1985.
8. K. McDonough, E. Caudel, S. Magar, and A. Leigh, "Microcomputer with 32-bit Arithmetic Does High-Precision Number Crunching," *Electronics*, Feb. 24, 1982, pp. 105-110.
9. S. Magar, E. Caudel, and A. Leigh, "A Microcomputer with Digital Signal Processing Capability," *Digest of Tech. Papers—1982 IEEE Int'l Solid-State Circuits Conf.*, pp. 32-33 and 284-285.
10. *TMS32010 User's Guide*, Texas Instruments Inc., 1983.
11. *TMS320C25 User's Guide*, Texas Instruments Inc., 1986.
12. *TMS32020 User's Guide*, Texas Instruments Inc., 1985.
13. H. G. Cragon, "The Elements of Single-Chip Microcomputer Architecture," *Computer*, Vol. 13, No. 10, Oct. 1980, pp. 27-41.
14. S. Rosen, "Electronic Computers: A Historical Survey," *Computing Surveys*, Vol. 1, No. 1, Mar. 1969.
15. M. Honig and D. Messerschmitt, *Adaptive Filters*, Kluwer Academic Publishers, Hingham, Mass., 1984.
16. *TMS320 Family Development Support Reference Guide*, Texas Instruments Inc., 1986.
17. M. Bellanger, "New Applications of Digital Signal Processing in Communications," *IEEE ASSP Magazine*, July 1986, pp. 6-11.
18. R. Lucky et al., *Principles of Data Communication*, McGraw-Hill, New York, 1965.
19. P. Van Gerwen et al., "Microprocessor Implementation of High Speed Data Modems," *IEEE Trans. Communications*, Vol. COM-25, 1977, pp. 238-249.



Gene A. Frantz has been Texas Instruments' applications manager for digital signal processing products since 1984. He is also a senior member of the Technical Staff at TI. He joined TI in 1974 as a system design engineer and worked on calculators in TI's Consumer Products Division. In 1976 he was assigned to the Li'l Professor design team. He was next assigned to the Speak & Spell project, where he served as program manager. Since then, he has been involved with every speech-related consumer product developed at TI.

Frantz received a BSEE from the University of Central Florida in 1971, an MSEE from Southern Methodist University, and an MBA from Texas Tech University.



Kun-Shan Lin has been involved in digital signal processing applications in the TI Semiconductor Group since 1984. He is a senior member of the TI Technical Staff. He joined Texas Instruments in 1979 and was assigned to the Consumer Products Division, where he developed speech techniques for learning aids. Prior to joining TI, he was an assistant professor of electrical engineering at Tennessee State University and an adjunct assistant professor of EE at the University of New Mexico. Lin received his PhD from the University of New Mexico in 1976.



Jay Reimer, a member of the TI Technical Staff, handles DSP applications engineering for the TMS320 family of products. He joined TI in 1979 to work with speech products in the company's Consumer Products Division. In 1984, he transferred to the Semiconductor Group to work with digital signal processors. His responsibilities include software development for the TMS320 family and applications assistance for customers using the processors. Reimer received a BS in physics from Fort Hays State University, Kansas, in 1975 and an MS in physics from the University of Kansas in 1977.



Jon Bradley is an applications engineer for the TMS320 family. He joined Texas Instruments in 1976 and has been an applications engineer for most of TI's microprocessor and peripheral products, starting with the TMS9900 family. His responsibilities have included microprocessor system design, digital and analog circuit design, integrated circuit design, test engineering, and programming. Bradley received a BSEE from Worcester Polytechnic Institute, Massachusetts, in 1976.

Questions about the products discussed in this article can be directed to Texas Instruments, Digital Signal Processing Dept.—Mail Station 6400, PO Box 1443, Houston, TX 77001; (713) 879-2320.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 150 Medium 151 Low 152