Speak software and carry a strip chip

by MICHAEL SHAPIRO
Texas Instruments
Houston, Texas

ABSTRACT

A short description of TI's innovative Strip Chip Architectural Topology is given. The key features of the TMS7000 8-bit Microlanguage Processor are listed, and each of the current family members is discussed briefly. The architecture of the 7000 family is reviewed with emphasis placed on those aspects which enhance its programming power. Addressing modes and other software highlights are discussed in some detail, followed by an overview of microprogramming.

INTRODUCTION

In the 1970's the Texas Instruments team hit high and low, scoring points with both the budget-cutting TMS1000 4-bit microcomputer family and the cerebral TMS9900 16-bit microprocessor. While churning out yards of silicon in 4-bit slices (more than 70 million chips), we also introduced the industry's first 16-bit single-chip microcomputer—the TMS9940. Now, to center our offensive line, we have plunged into the 1980's with the innovative TMS7000 Microlanguage Processor family, our new 8-bit star.

TI had no intention of being a look-alike in a marketplace which already accepted several 8-bit architectures. Rather, by using a unique design approach to lower chip costs, and by implementing a rich instruction set to raise programming efficiency, we embarked on a third-generation design which is expanding into a powerful line of microcomputer products. This paper will touch first on the design concept and hardware features, concentrating later attention on the instruction set highlights and other software considerations.

SCAT—STRIP CHIP ARCHITECTURE TOPOLOGY

SCAT is TI's term for the design philosophy that incorporates the nonmemory elements of the microcomputer (the CPU registers, the ALU, the control logic) into a strip of vertical blocks in the logic design. Traditional design schemes have attacked the individual functional blocks first, leaving the problem of interconnect for last. Unfortunately, in the final layout, the interconnect often squanders the real estate prudently conserved in the early stages of design. To combat this profligate process, TI planned both architecture and layout from the beginning.

Figure 1 shows the layout of the TMS7020, the 2K ROM version of the TMS7000 family. By placing most of the random logic in the "strip," we were able to use control and data paths that interconnect the active elements but take up almost no additional silicon area. The logic of the elements in the strip is implemented on a low level of the silicon bar, whereas the data and address busses are constructed in metal over the silicon. This avoids the wasteful dedication of bar area to interconnect alone.

An additional space-saving feature of the SCAT design is the use of transistor arrays and ROM elements to replace random logic. Not only are these structures more compact, but the use of the micro-control ROM in place of the commonly used programmable logic array for the instruction decode allows the necessary control signals to be fed horizontally out of the control ROM right across to the strip. Torturous routing problems are avoided, and no additional combinatorial logic is required. A valuable by-product of this approach is microprogrammability, which will be discussed later in this paper.

TMS 7000 MICROLANGUAGE PROCESSOR FAMILY
TMS 7000/7020 MICROCOMPUTER DEVICE BAR PLAN

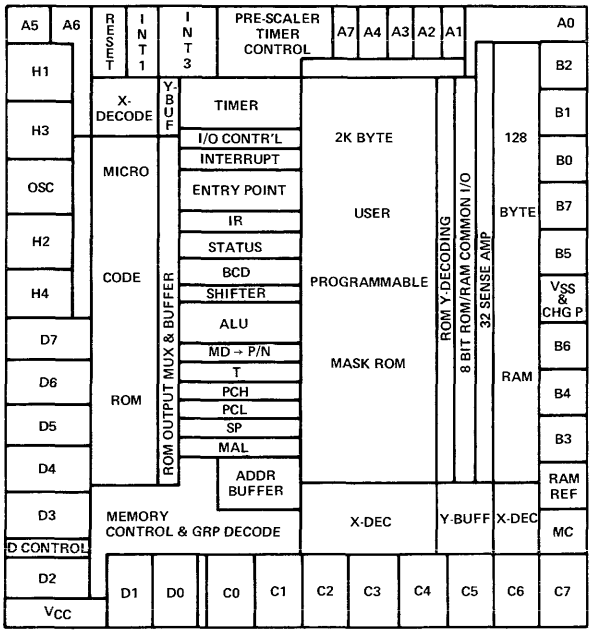

Figure 1—TMS7000/7020 microcomputer device bar plan

KEY ELEMENTS OF THE TMS 7000 FAMILY

The most attractive components of the TMS7000 family include the microprogrammed 8-bit CPU, addressing capability for up to 64K bytes of onboard and offboard memory, 32 individual I/O lines, multiple operating modes, unrestricted stack for control and data storage, 8-bit timer with presettable 5-bit prescaler, and four levels of vectored interrupt. The first family members have been implemented in high-density NMOS technology. CMOS and LMOS versions will follow in the months to come.

Family Overview

The TMS7000 family offers a variety of on-chip RAM and ROM configurations plus packaging and technology options to support the full scope of application requirements. The current family members include the TMS7000, 7020, 7040, 70L22, and the soon to be released 70E40.

The TMS7000 is a ROM-less device with 128 bytes of RAM. It functions as a powerful 8-bit microprocessor with on-chip RAM, interfacing to as much as 64K bytes of external memory on an 8-bit data system bus. The TMS 7000 provides eight input and four output I/O pins on the chip, each of which may be set, reset, and tested individually. Utilizing the 8-bit data bus, any of the common 8-bit I/O peripherals can be easily interfaced to the TMS7000 in order to expand its I/O capability.

The TMS7020 and 7040 are similar to the TMS7000 and contain the same CPU, RAM, and on-chip I/O when operating in the Microprocessor Mode. Moreover, these devices contain 2K and 4K respectively of on-chip ROM for application programming. The 7020 and 7040 may be configured in several memory expansion modes where memory interface pins are traded off for I/O pins. Besides the Microprocessor Mode, the other choices are as follows:

1. Single-Chip Mode providing 32 I/O lines
2. Peripheral Expansion Mode for interfacing to 8-bit peripherals
3. Full Expansion Mode to address 64K bytes of memory
4. System Emulator Mode for aiding program development

The most pertinent features of the TMS7020 and 7040 microcomputers are as follows:

1. Microprogrammed 8-bit CPU
2. 2048 bytes of on-chip ROM—TMS7020
3. 4096 bytes of on-chip ROM—TMS7040
4. 128 Memory-mapped registers (register file)
5. Multilevel program/data stack
6. 32 bits of general purpose I/O
7. On-chip 13-bit timer/event counter with interrupt and capture latch
8. Three maskable interrupts

The TMS70E40 is functionally identical to the 7040 except that the System Emulator Mode has been deleted and the on-chip mask ROM has been replaced by a programmable EPROM. One change has also been made in the instruction set to allow the 70E40 to program its own internal EPROM. This device is ideally suited for prototype fabrication or initial field testing of a new application prior to masked ROM volume production.

The TMS70L22 is a lower-cost alternative to the 7020, which retains most essential features, but gives up nine I/O pins to accommodate the smaller (and cheaper) 28-pin package. Processed in our power-saving LMOS technology, the 70L22 also works a trade on the clock frequency, operating at 1 MHz versus 5 MHz, achieving a tenfold reduction in power consumption. A new feature on the 70L22 is a slowdown mode that allows the user to further reduce current to accommodate applications in which power must be conserved.

Architecture

All members of the TMS7000 family incorporate features that take the best from both memory- and register-based architectures. The first byte in the RAM register file, Register A (R0), functions just like a dedicated accumulator to allow for faster access times and the 1-byte instructions that are inherent in a register type of machine. Similarly, the second byte, Register B (R1), can perform the task of a dedicated index register. However, the flexibility of the 7000 enables any one of the on-chip RAM bytes to assume the accumulator function by the addition of one byte to the instruction. True register-to-register operations can be accomplished throughout the 128-byte register space when a third byte is used in the instruction to specify the second operand.

Registers

The 7000 family has three hard-wired CPU registers accessible to the user. The 16-bit program counter (PC) contains the address of the next instruction to be executed. The status register (ST) contains three status bits that are used for conditional jump instructions. Also present in this register is the interrupt enable bit (I). The 8-bit stack pointer (SP) points to the top (last) entry in the data stack, and it facilitates multilevel subroutining and interrupts. The register file (RF) consists of 128 bytes of on-chip RAM.

Peripheral File

Beyond the memory address space devoted to the register file, there is a 256-byte region for memory-mapped peripheral input/output control, called the peripheral file (PF). The 32 bits of general purpose I/O, available in the Single-Chip Mode, are broken out into four 8-bit ports (see Figure 2) that can be manipulated via six dedicated peripheral instructions. Any of these bits may be individually set or cleared, or tested in conjunction with an appropriate bit-test-jump instruction.

Not only can the dedicated input (A Port) and output (B Port) ports be read from and output to, but the individual bits of the bi-directional ports (C Port and D Port) can be configured selectively as input or output by accessing their data direction registers (DDR), which also reside in the peripheral file.

To simplify use of the peripheral file, a special peripheral file-addressing mode was established to reference all 256 locations. Inputs and outputs on the I/O lines are accomplished by reading or writing to the appropriate port. For example, the B Port is implemented as port P6 in the periph-

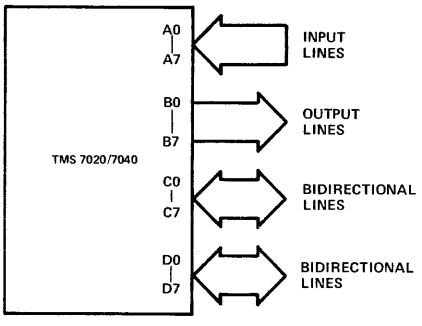

Figure 2—I/O ports in single-chip mode

eral file. Thus, writing to this port is handled by the instruction

```
MOVP    A,P6
```

which takes the value in the Register A (R0) and stores it on the B Port outputs.

In the Peripheral Expansion Mode, the peripheral instructions can be used to communicate with off-chip devices. When a memory address not corresponding to an on-chip port is used, the 7000 family device performs an external memory reference enabling an 8-bit peripheral chip to respond.

Timer/Event Counter

The 7000 family is equipped to handle real-time control applications by using a programmable 8-bit timer with a presettable prescaler value of from 1 to 32. As shown in Figure 3, the timer may use an internal clock source divided down or an external signal. On each positive edge transition of the clock input, the prescaler register is decremented. When the prescaler reaches zero, the decrement is performed on the 8-bit timer, and the prescaler is reloaded from the control latch.

As with the prescaler, the timer register will decrement until it reaches zero. The succeeding decrement will generate an interrupt (INT2), and the timer register will be reloaded from the timer latch. Since these registers reside in the peripheral file, the prescale latch value and the timer latch value may be written to, and the current timer value may be read using peripheral file instructions. Likewise, the timer on/off and the clock source bit are under program control in the peripheral file.

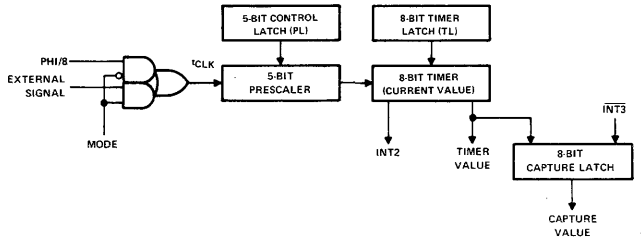

Figure 3—Programmable timer/event counter

In the event counter mode, the counter will function as described above, but the decrementer clock source will now be line A7 of the A Port. This timer mode can also serve the purpose of a real-time clock when an appropriate source is fed to A7. The A7 input can also be used as a positive edge-triggered interrupt by loading the prescaler and timer latches with 0.

A unique feature of the 7000 timer is the 8-bit capture latch, which saves the current value of the timer when an external (INT3) interrupt occurs. This allows the processor to determine precisely when the external event took place by comparing the captured value to the value that is now current. This capability can be essential if the external interrupt occurs while the processor is servicing a higher-order interrupt.

Interrupts

There are three levels of maskable interrupts: the INT2 associated with the timer and INT1 and INT3, which are externally triggered. The system reset cannot be masked, but the other three interrupts can each be enabled separately by bits in the I/O control register, and as a group by the interrupt enable bit (I) in the status register. When an interrupt is recognized, the contents of the status register and the program counter are pushed onto the stack. The processor then branches to the location stored in the corresponding interrupt vector location and starts execution of the interrupt routine.

Interrupts may be tested without actually recognizing them, allowing for greater user flexibility. Interrupts may be edge- or level-triggered, and no external synchronization is required. The signals are latched internally to catch short interrupt pulses.

The TRAP instruction can be used to create a "software" interrupt. There are 24 TRAP opcodes corresponding to 24 trap vector locations in the highest addresses of memory. As in an interrupt, the trap vector will provide a branch address at which a subroutine begins execution. Limitation on nesting in subroutines or interrupts is only a function of the overall stack capacity.

PROGRAMMING THE TMS7000

From the outset, the TMS7000 family was designed to optimize programming efficiency by virtue of its architecture and instruction set. The ease of access to the RAM, ROM, and I/O is achieved by mapping all of these into a single address space. Figure 4 illustrates the memory address scheme for the 7020/7040. This structure can be fully exploited by means of

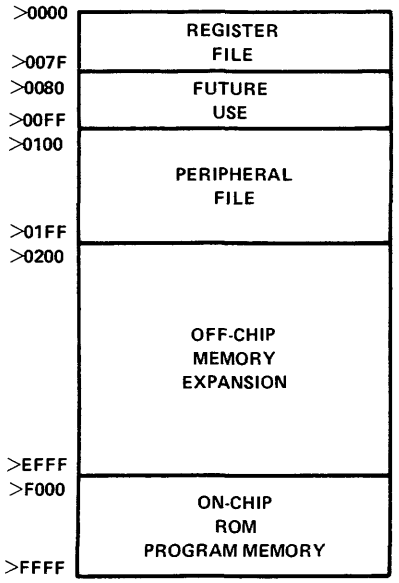

Figure 4—TMS7040 memory map

nine separate addressing modes. Add to this a full complement of standard instructions (the usual byte-oriented instructions plus multiplication, single- and multiple-bit tests, double precision arithmetic), and the design engineer has the upper hand in dealing with almost every application.

Addressing Modes

The nine different addressing modes for the TMS7000 family are listed below. The terms Register A and Register B are synonymous with the first two bytes in the register file, R0 and R1, respectively.

1. Register File—The byte(s) following the opcode specify any byte in the register file as the operand location(s). This includes single operand instructions such as

```
INC   R56     Increment the contents of R56
CLR   R99     Clear R99
```

   and dual operand instructions such as

```
ADD   R68,R45   Add R68 to R45 and store in R45
```

2. Register A—The operand location is implied, and R0 is fetched from the register file. This is a special case of register-file addressing, since Register A can be referenced implicitly as A or explicitly as R0; however, the implied mode saves a byte in the instruction. For example, the instruction

```
MOV   R20, R30   Move R20 to R30
```

   is three bytes versus two for the instruction

```
MOV   R20,A
```

3. Register B—The operand location is implied, and R1 is fetched from the register file. This is identical to Register A addressing except now B is the implied register.
4. Peripheral File—The byte following the opcode specifies a port in the peripheral file which contains the operand. These instruction mnemonics are identified by a P suffix. Each is a dual operand instruction with a peripheral file as the second or destination operand. Examples of these are

```
XORP  A,P3      Exclusive OR A with P3 and
                place the result in P3 (the timer
                control register)

MOVP  %>60,     Setup bits 1,2 of D PORT as
      DDDR      inputs
```

5. Direct—The two bytes after the opcode contain the address of the byte in memory that contains the operand. The notation for the direct memory address is the expression preceded by the @ sign. For example

```
LDA   @>E34D    Copy the contents of memory lo-
                cation >E34D to Register A
```

6. Indirect—The byte following the opcode specifies the second of a RAM register pair which contains the address of the operand. This addressing mode is indicated by the * before the register as in the following instruction:

```
STA   *R19      Copy the contents of A into the
                memory location specified by R18
                and R19
```

7. Indexed—The 16 bits following the opcode are added to the B register contents to form the effective address of the operand. The format for this instruction is given below.

```
BR    @HERE(B)  Branch to the address specified
                by the contents of B and the val-
                ue of the symbol HERE
```

8. PC Relative—The byte following the opcode is used as a signed offset to the current PC to produce the effective address. This is the addressing mode used for all jump instructions, and it eliminates the designer's concern about where in ROM his program is jumping to, since the offset may lie anywhere in ROM.
9. Immediate—The byte following the instruction is the operand. For example, the instruction

```
ANDP  %COUNT,   Logically AND the value of
      P10       COUNT and the contents of P10
                and copy results to P10.
```

   illustrates the use of immediate addressing.

Because of the memory-mapped architecture, many modes can apply universally to any 16-bit address in the TMS7000 memory space. Thus ROM, RAM, or peripherals can be referenced with similar instructions possibly using common routines. The need for dedicated instructions in each category is now eliminated.

A very flexible feature of the 7000 is the capability of freely specifying two operands, the source and destination, within the dual operand addressing modes. While most microcomputers would restrict one of the operands to a particular register, the 7000 allows any RAM location to be named the source or the destination.

Instruction Set Highlights

As mentioned before, the TMS7000 family provides the full range of standard instructions. Rather than list the entire set, we will discuss some of the more unique members.

The MPY (Multiply) instruction takes the product of a general source and destination operand and places the 16-bit result in either A or B. The 7000 can perform this 8-by-8-bit

unsigned multiply in just 17.2 microseconds, assuming a 5 MHz clock.

The MOVD (Move Double) instruction is used to move a 16-bit value to a specified register pair destination. The source for this move can be an immediate constant, another register pair, or an indexed address.

The DAC (Decimal Add with Carry) and DSB (Decimal Subtract with Borrow) instructions provide the unique feature of performing fully corrected decimal addition or subtraction on two packed binary coded decimal (BCD) bytes.

The DECD (Decrement Double) instruction allows a 16-bit address to be easily decremented. This instruction can be especially useful for referencing tabular information in memory.

There are several jump instructions with especially useful test conditions to dictate transfer of program control. The BTJO (Bit Test and Jump if One) instruction looks at those bits which are 1's in the source operand and compares the corresponding bits in the destination operand. If any of these bits are also 1's, the relative jump is taken. There is a similar instruction BTJZ which does the comparison on bits which are 0's. These instructions allow for single- or multiple-bit tests.

Instructions as powerful as these are usually only available on more expensive high-end microcomputers (if at all). However, in the case where the designer has underscoped the task or runs up against a particular application intricacy, microprogrammability provides a possible out.

Microprogrammability

When TI implemented the TMS7000 instructions using a control ROM rather than random logic, it opened up the possibilities for user-defined "personalized" instruction sets, because the control ROM can be altered and then mask-programmed for production. Although the standard instruction set is very efficient for most applications, the user may find a repetitive program sequence of several instructions that could be reduced to a single command through microcoding. This would both increase throughput and reduce memory usage. Approximately 75% of the standard instructions are designated as core instructions and must be maintained. The remainder may be swapped out for user-created instructions which are customized to best serve that particular application. Software will soon be available to aid users in the design of microcode for a custom instruction set.

SUMMARY

This paper has attempted to give a broad overview of the TMS7000 family. We have given the reader only a brief taste (with software seasoning) of the capabilities available in the 7000 larder. In addition to the stock of products now available, we will soon be introducing a CMOS implementation and enhanced feature versions. To the hungry design engineer in search of a satisfying microcomputer—bon appetit!