G. L. Dix
M. D. Brown

Common Chip for Use in Disk and Diskette Controllers

The advent of LSI technology makes common microprogrammable controllers very cost-effective today. This paper focuses on the application of microcontrollers for disk and diskette control functions and describes a custom-designed FET chip which is being developed for use in these types of controllers. The architecture, the functional organization, and the physical design of this chip are presented, and the requirement of matching a microcontroller to the application is discussed.

Introduction

Many existing IBM systems have utilized unique disk and diskette attachments; an example is the attachment of the 72MD diskette magazine drive to the Series/1 [1], System/34 [2], and System/38 [3]. These attachments were optimized to the internal structure of each system in order to minimize product cost.

With the advent of LSI, product cost per circuit has decreased significantly, while both design time and development cost per circuit have increased. These factors make it cost-effective today to produce common controller designs for attaching disk and diskette drives to a system. An instance of such a common design was used in the attachment of the 62PC disk drive to the Series/1 [4] and System/34 [5].

Our objective was to provide common controllers for newly developed disk and diskette drives which would be available to each system using those drives. Product cost and development resource constraints, along with the need to present a common appearance for differing drives, led to the development of a common chip which serves as a building block central to several controllers.

In the following sections of this paper, the design considerations leading to the development of this chip are first discussed. Each functional area of the chip is then described, including the microcontroller, the Random-Access Memory (RAM), the system port, and the device port. The physical design of the chip is then discussed briefly, followed by some conclusions concerning the design effort.

Design considerations

A common approach to designing controllers is complicated by unique drive characteristics. One example is that disk data rates are an order of magnitude higher than diskette data rates. Sector sizes and data organization vary from one drive to another. Access control requirements also vary. Another factor is that diskette drive attachments are more cost-sensitive because a diskette drive costs much less than a disk drive.

To allow common attachments, it is necessary to provide an interface which improves device independence. This makes the initial attachment of a unique device simpler and facilitates migration from device to device. Some of the features of this interface are as follows:

- Sectors are addressed logically by a consecutive linear number. The system need not be aware of physical disk characteristics (*e.g.*, number of heads) because the controller performs the logical-to-physical translation.
- Data are transferred using simple read and write commands, with data accessing occurring automatically. Separate seek commands are unnecessary.
- Error recovery procedures and diagnostics are provided by the controller. These are typically unique to each device and in the past have been written uniquely by each system development group.

- Multisector data buffering is provided which is designed to prevent channel overrun conditions and to handle a range of system channel performance.

It was apparent that a microcontroller would be a cost-effective way of providing these higher-level functions. It would also provide flexibility, since many of the unique drive characteristics could be handled with microcode. A properly architected microcontroller would also displace logic by taking over control of such functions as access control, spindle speed control, and processing between sectors.

Note that we have referred to a requirement for a microcontroller (*i.e.*, optimized to device control) rather than to a microprocessor (*i.e.*, optimized to data processing). Although the distinction is somewhat nebulous, the following features were desired.

- *Fast execution of simple functions.* A two-operand instruction architecture usually performs better than a single-operand, accumulator-based architecture because fewer instructions are required. Since many operations will involve I/O control registers, these control registers should not require special I/O instructions, but should instead be directly addressable as operands.
- *Low cost.* This can be accomplished by using an eight-bit data flow with a relatively simple instruction set. Simple arithmetic capability is necessary, but operations like multiply are infrequently used and can be provided by subroutine.
- *Fast interrupt response.* Significant I/O events can be communicated by an interrupt mechanism rather than by being sensed by a poll loop. Response time includes any instructions necessary to save internal registers; automatic hardware saving is desirable.

The original intent was to use an existing microcontroller; therefore, several were evaluated. Some of those evaluated are described in [6–10]. Performance was evaluated by coding representative kernels to determine execution time and the number of bytes of microcode required. Cost was also evaluated and included all components necessary to complete a disk/diskette controller. The result was that none of the microcontrollers evaluated satisfied both the cost and performance objectives.

One of the microcontrollers evaluated [9] seemed to be a good fit to the application and incorporated many of the features previously described. The problem was that it had been implemented in a high-performance bipolar technology [11] and it exceeded our cost objectives. Since it also exceeded our performance objectives, a possible solution was to implement the same function in a lower-cost FET technology. Initial estimates indicated that a redesign using FET technology could meet our performance objectives and that it could be packaged on a portion of a chip. Several packaging alternatives were then examined, resulting in the common chip and overall controller structure shown in Fig. 1.

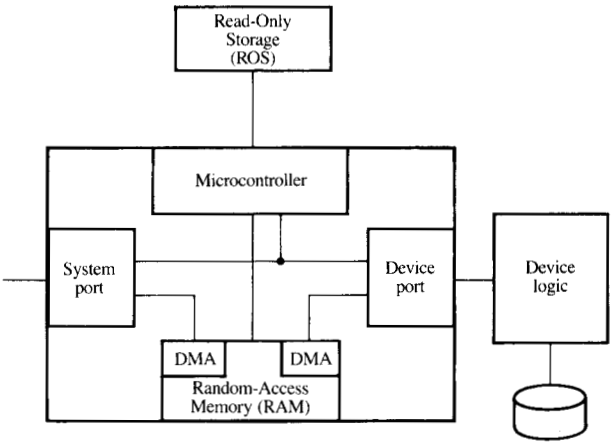


Figure 1 Controller structure. (Note: DMA = Direct Memory Access.)

In addition to the microcontroller, the common chip contains a RAM, a system port, and a device port. In a more conventional partitioning, the microcontroller and RAM would be on separate chips, while the RAM controls, system port, and device port would require one or more chips. In addition, off-the-shelf chips generally do not interface directly together, but require some "glue" logic; combining functions on a single chip eliminates this glue. The end result is that the chip configuration shown has a significant cost leverage over a comparable multiple-chip design.

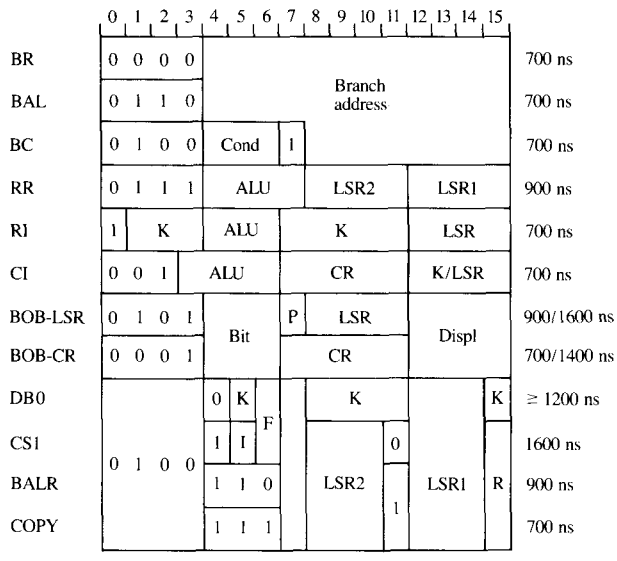
Functions included within the common chip allow it to be used in several diskette controllers. The ROS personality and the device logic are unique to a specific controller and are partitioned onto separate chips.

Separate buses are used for the ROS, control registers, and RAM interfaces. This isolates the instruction path from the data path and allows the microcontroller to operate concurrently with the high-speed data transfer required by disk files.

Microcontroller

As previously mentioned, the instruction set of an existing microcontroller [9] formed the starting point of the design. As the design progressed, some changes were made to better suit the application; significant changes are as follows:

- The address space was partitioned to allow addressing of either the on-chip RAM or the off-chip ROS.



ALU functions

And
Or
Exclusive Or
Test Under Mask
Compare
Bit Clear
Move
Add
Add with Carry
Subtract
Subtract with Borrow

Abbreviations

LSR = Local Store Register
CR = Control Register
K = Immediate Operand
P = Polarity
I = Increment Address
F = Fetch
R = Reserved

Figure 2 Instruction set.

- An indexing capability was added to the addressing of the on-chip RAM.
- A subtract function was added to the ALU.

The microcontroller uses a 16-bit microinstruction with up to 16 sub-operations (sub-ops) in a single instruction. There are six branch-type instructions, four register instructions, and two storage instructions. The register instructions address either local store registers (LSRs) or control registers (CRs). The storage instructions address either external control store or the on-chip dynamic RAM.

A summary of the microinstruction set follows (Note: an instruction = 16 bits, a halfword = 16 bits, and K = 1024):

BR: Branch within the 4K block of instructions, with the address specified by the immediate data.

BAL: 4K branch, with the branch address specified by the immediate data and return address saved in the LSRs.

BALR: 64K branch, with both branch and return address in the LSRs.

BC: 256-instruction branch on condition code.

BOB-LSR: 16-instruction branch based on the state of a bit in a specified LSR.

BOB-CR: 16-instruction branch based on the state of a bit in the specified CR.

RR: LSR-to-LSR ALU operation—has 12 sub-ops.

RI: LSR immediate ALU operation with 8 sub-ops.

CI: Control register immediate operation with 16 sub-ops.

COPY: Copies one LSR halfword to another.

DB0: Moves a halfword between an LSR and the RAM. LSR1 contains a base RAM address which is ORed with a 6-bit immediate field to perform an indexing function.

CS1: Moves a halfword between an LSR and the ROS/RAM. LSR1 contains the ROS/RAM address and LSR2 is the data source or destination. The ROS/RAM address is optionally incremented.

The instruction set code points and the execution times of the instructions are summarized in Fig. 2.

The local store registers are implemented as a 64 × 16 static RAM which can be read or written in either byte (8 bits) or halfword (16 bits) mode. These LSRs are partitioned into 16 pages, with each page containing 8 bytes. A local store page register (LSPR) provides for independent selection of a primary and secondary page. The LSR addressing field within the instruction can select a byte or halfword within either the primary or secondary page.

The instruction set can address up to 32 control registers. These 8-bit hardware registers are used to control the system port, the device port, the dynamic RAM, and the interrupts. A 5-bit field in the instruction directly addresses the register to be used.

The data flow (shown in Fig. 3) was modified from the bipolar design to allow operations in parallel and to obtain more performance with the slower FET technology. For example, the bipolar design utilized the ALU to assist in incrementing the instruction address, while the FET design has a separate incrementer. Another difference is that the LSRs and the interrupt controls have been integrated into the microcontroller. Programmable logic arrays (PLAs) were used in selected areas in order to simplify the design

process and facilitate change. The ALU is an 8-bit design providing arithmetic and logical functions. The LSR has an 8-bit path to and from the ALU, but the rest of the LSR paths are 16 bits wide. A path between the LSRs and the Memory Address Register/Back-Up Register (MAR/BUR) provides branching capability. Another path between the LSRs and dynamic RAM gives the microcontroller access to the RAM. There is a third path between the LSRs and control store. Finally, a 16-bit path connects the output of the LSRs to the input to the LSRs to allow copying one LSR pair to another in one instruction time.

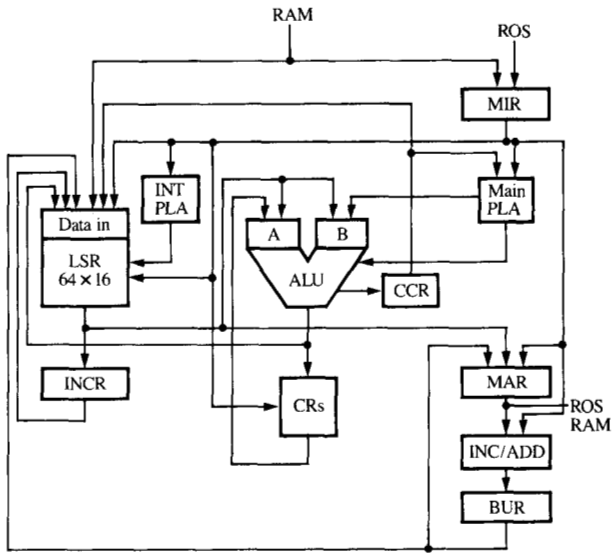
There are eight interrupt levels in the microcontroller, Levels 0 through 7: Level 0 has the highest priority while Level 7 is the default level. There are two interrupt registers; CR29 is the interrupt-request register and CR30 is the interrupt-enable register. The interrupt request is normally set by a hardware event, but it is also settable by the microcode. When an interrupt request becomes active and has the highest priority of the enabled levels, a Program Status Word (PSW) swap will occur. The instruction address, the condition code, and the LSPR are automatically exchanged and the microcontroller starts executing at the new level. The delay from receipt of the interrupt request until execution begins in the new level is a maximum of two instruction times. To exit from an interrupt level, the microcontroller resets the level bit in the request register. This causes a PSW swap and execution begins on the next highest active level.

The chip can address up to 32K halfwords of external ROS. The memory address register (MAR) is 16 bits long. Setting the high-order bit (bit 0) on allows the on-chip 3K-byte RAM to be used for instruction storage. The RAM can thus be used for transient (*e.g.*, diagnostic) programs.

Random-access memory

A 3K-byte dynamic RAM is provided which is organized as 1.5K × 18 bits and has a cycle time of 400 ns. The overhead for refresh is approximately 3% with all refresh controls contained on the chip, yielding a useful RAM data rate of 4.85 megabytes per second. This allows the controller to support disk data rates in excess of 2 megabytes per second, since on consecutive sector operations this data rate can be supported simultaneously on both the system and device ports.

RAM accesses are allocated on a priority basis, with the microcontroller, system port, device port, and refresh controls requesting RAM cycles as required. RAM-address registers and length-count registers are provided for the system and device ports; they are accessible to the microcontroller as control registers.



Abbreviations

MIR = Memory Instruction Register
INT = Interrupt
PLA = Programmable Logic Array
INCR = Increment
CCR = Condition-Code Register
MAR = Memory Address Register
BUR = Back-Up Register

Figure 3 Data flow.

System port

The system port contains registers and controls which allow the controller to function as a slave to a using system. The bus structure supports both DPC (Direct Program Control) and DMA (Direct Memory Access) operations. DPC cycles are used to transfer commands from the system and to return status to the system, while DMA cycles are used to transfer data between the system and the controller. For flexibility, both one-byte and two-byte transfer options are provided on the interface.

Commands and parameters received from the system during DPC cycles are buffered internally and then automatically stored in preassigned RAM locations. After an entire command block has been received, the microcontroller is interrupted to notify it of the command receipt.

DMA data transfers are initiated after the microcontroller loads the address and length count for the system port. Data are transferred between the system and the RAM, utilizing a buffer register and controls contained in the system port. The data transfer is paced using "request" and "acknowledge"

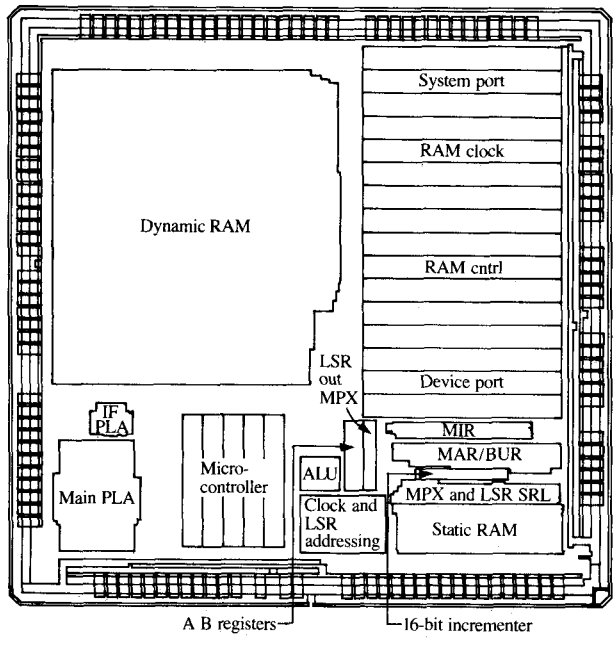


Figure 4 Physical layout. (Note: MPX = multiplexer, SRL = Shift Register Latch.)

lines. Because the data are buffered, the transfer rate need not match the instantaneous file data rate, and data may be transferred at either a faster or slower rate. Data transfer continues, under control of the system port, until the length count is exhausted. Data transfer then terminates and an ending interrupt is presented to the microcontroller.

After an entire command sequence has been completed, the microcontroller loads the ending status into a control register and creates an interrupt to the using system. The system then performs a DPC operation to read the ending status.

Device port

The device port contains registers and control circuitry which allow the common chip to interface with device logic. The microcontroller can read or write device logic registers by utilizing a DPC path. Two control registers are provided: a DPC address register, which is used to select a particular register in the device logic, and a DPC data register, which is used to buffer data being read from or written to the device logic.

A DMA path is provided for transferring data between the RAM and the device logic. DMA transfers are enabled after the microcontroller loads the address and length count, but the actual data transfer is paced by the device logic utilizing a "request DMA" line. These transfers may occur in either one-byte or two-byte mode.

DMA and DPC cycles are time-multiplexed on a single bus. Bus cycle time options are provided for flexibility in attaching either bipolar or FET logic.

An on-chip interval timer facilitates real-time device control. It is programmable in 50-microsecond increments to a maximum of 12.8 milliseconds.

Physical design

A physical layout is shown in Fig. 4. The microcontroller is contained on the lower portion of the chip and is partitioned into several macros. Each macro is a functional unit; *e.g.*, PLA, ALU, register, and static RAM. Manual design and wiring were utilized in order to maximize density.

The dynamic RAM occupies the upper left portion of the chip. It utilizes a single-device-per-bit technology and includes a refresh address counter.

The RAM control, system port, and device port are contained on the upper right portion of the chip. This area was designed using logic books, (*i.e.*, circuits) which fit into a regular structure and allow the use of automated placement and wiring programs. Examples of book types are AND, OR, latch, register, and parity functions.

The design philosophy was to achieve the highest densities on the most design-stable portion (the microcontroller) of the design. On the less stable portions, density was sacrificed somewhat in order to use automated design aids, to shorten design time, and to facilitate engineering design changes. The result is that alternative chip designs which modify the system and/or device ports but do not modify the microcontroller or RAM can be produced with limited development effort.

Conclusions

The common chip described in this paper is capable of supporting controllers for a variety of disk and diskette drives. The additional chip functions which are required for common use, and which may not be required in a particular controller, are more than offset by an aggressive design which integrates functions previously contained on multiple chips into a single chip.

Although this chip was designed specifically for disk and diskette controllers, there is potential for application to other types of devices: *e.g.*, magnetic tape, printers, and communication lines. If necessary, alternative versions of the chip could be developed to fit a particular application. Since the device and system ports are designed using regular structures, they are amenable to change, while changes to the microcontroller and RAM would be more difficult.

The current status of the development effort is that the chip design is complete and functional parts have been built.

Acknowledgments

The authors thank John Guest and John Burchfiel for their support and encouragement. We are also grateful to Mike Sheehan and Bernie Manning for their contributions to the physical design.

References

1. *IBM Series/1 4966 Diskette Magazine Unit Description,* Order No. GA34-0052, available through IBM branch offices.
2. *IBM System/34 Functions Reference Manual,* Chapter 8, Order No. SA21-9243, available through IBM branch offices.
3. J. W. Froemke, N. N. Heise, and J. J. Pertzborn, "System/38 Magnetic Media Controller," *IBM System/38 Technical Developments,* 41–43 (1978); Order No. G580-0237, available through IBM branch offices.
4. *IBM Series/1 4963 Disk Subsystem Description,* Order No. GA34-0051, available through IBM branch offices.
5. *IBM System/34 Functions Reference Manual,* Chapter 11, Order No. SA21-9243, available through IBM branch offices.
6. *M6800 Microcomputer System Design Data,* ®Motorola Semiconductor Products Inc., Phoenix, AZ (1976).
7. *MCS-85 User's Manual,* ®Intel Corporation, Santa Clara, CA (1977).
8. *MCS-48 User's Manual,* ®Intel Corporation, Santa Clara, CA (1979).
9. John I. Norris, "A High-Performance Microprocessor," *IBM Disk Storage Technology,* 27–30 (1980); Order No. GA26-1665-0, available through IBM branch offices.
10. E. F. Dumstorff, "Application of a Microprocessor for I/O Control," *IBM System/38 Technical Developments,* 28–31 (1978); Order No. G580-0237, available through IBM branch offices.
11. R. J. Blumberg and S. Brenner, "A 1500 Gate, Random Logic, LSI Masterslice," *IEEE J. Solid-State Circuits* **SC-14,** 818–822 (1979).

Received July 28, 1981; revised January 29, 1982

The authors are located at the IBM System Products Division laboratory, 3605 Highway 52, 37th Street N., Rochester, Minnesota 55901.