intel

APPLICATION NOTE

AP-157

October 1983

Software Design and Implementation Details for Bubble Memory Systems

Richard Pierce
Applications Engineer
Intel Corporation

ORDER NUMBER: 230707-002

INTRODUCTION

The 7220-1 is a single-chip LSI Bubble Memory Controller (BMC) that implements a bubble memory storage subsystem (with up to eight bubble storage units (BSUs) per BMC). Each bubble storage unit consists of five support circuits in addition to the bubble memory chip and provides one megabit (128 kbytes) of non-volatile read/write memory. This application note examines the programmatic interface to the 7220-1 BMC and how communications between a host processor and the bubble subsystem (i.e., 7220-1) govern all bubble system operations.

The BMC provides all the control and timing signals for the bubble and support circuits. All data synchronization and error checking is automatically performed by the bubble subsystem to ensure reliable data storage. The BMC easily interfaces to microprocessor systems and communicates via a set of high-level commands. This application note explains the operations and functions performed by these instructions and provides the basic programmatic interface descriptions and program guidelines to allow you to design and implement a program module or "driver" to control all bubble system operations. In addition, several possible software interface levels are defined. While the design guidelines presented are not targeted for integration with any particular operating system, they do, however, provide conceptual information on design requirements and serve as a foundation on which to develop a modular and flexible software driver aligned with your specific application requirements.

Product Line Overview

Intel offers a complete line of bubble memory components, development kits and assembled boards.

The BPK 72 (Bubble Memory Prototype Kit) serves primarily as a means to evaluate the potential of bubble storage. The BPK 72 comes complete with all the hardware and documentation necessary to prototype a one-megabit bubble memory system. After the kit is assembled, the designer is left with the simple task of interfacing to a host processor.

The BPK 70 one-megabit bubble storage subsystem is a fully interchangable component bubble memory system. Each kit contains all the components in a Bubble Storage Unit (BSU). A single 7220-1 Bubble Memory Controller (purchased separately) can operate up to eight BPK 70 subsystems at one time. The BPK 70 is available for the production of custom systems where high volume is required.

The iSBX™ 251 Magnetic Bubble MULTIMODULE™ board is a one-megabit bubble memory mounted on a standard dual width Intel MULTIMODULE memory expansion card. Completely assembled and tested, the iSBX-251 board is fully plug compatible with all Intel iSBC Single Board Computers that have iSBX connectors.

The iSBC® 254 Bubble Memory Board is a completely assembled Intel MULTIBUS® memory board. The board can be configured with one bubble memory (128 kbytes), two bubble memories (256 kbytes), or four bubble memories (512 kbytes).

The 7220-1 BMC acts as the interface to the host processor in each of the aforementioned products, thus simplifying system programming. The basic programming techniques discussed in this application note will provide the system programmer with a complete understanding of programming requirements and shorten the software development time.

SOFTWARE INTERFACE

Basic Driver Operation

As will become evident, a basic compromise or "tradeoff" exists between the design of your application software and the capabilities of the bubble memory controller. While you will be responsible for the development of a driver to integrate the bubble into your system, the level of driver interaction and degree of flexibility will vary according to the needs of your application. If an application program is small and simple, a basic bubble driver simply may be called from the main program. At the next level of driver sophistication, the bubble system is viewed by the application program as a logical device. At this level, the key to driver design is the mapping of the "logical bubble interface" (as viewed by the application program) into the "physical bubble interface" (as implemented by the bubble drivers). This logical-to-physical mapping serves to isolate application programs and system software from the idiosyncracies of the bubble memory controller. At the

highest level of driver sophistication, the application program treats the bubble system as a collection of named data areas or files similar to the way in which data is stored and retrieved in disk operating systems. At the file system level, an application program can ignore the mechanics of bubble storage and access and merely present a "file name" to the driver to open, read or write, and then close the desired "bubble file."

At the subsystem level (i.e., "physical bubble interface"), the bubble driver is responsible for all system interaction with the bubble and is intrinsic to the efficient and reliable operation of the bubble system. The driver accepts bubble memory commands and command execution parameters from the application program, controls and monitors command execution, and returns operational status information to the application program at command completion. To perform all of these operations, the bubble driver must support the bit/byte level of the bubble memory controller's command and status registers and the "parametric" registers that define the operating mode, system configuration, and extent of the transfer. Depending on the interface level of the application software, the driver itself may be made up of a set of subroutines that are called individually by the host to perform specific bubble system operations.

SOFTWARE-SELECTIVE CONFIGURATIONS

Before you can begin to design a software driver, you must consider all of the selective configurations available within the bubble system and which of these configurations you want to support. As will be explained, the type of data transfer (i.e., direct memory access, interrupt driven, or polled), the data transfer rate, and the selective implementation of the bubble system's error correction feature all are software controlled. The complexity of the driver design depends on the system flexibility desired. For example, a driver may be designed to support only one transfer mode and may not make use of error correction. Conversely, a more generalized driver can be designed to support various transfer modes, data rates, and levels of error correction. The ensuing paragraphs define the driver responsibilities associated with the available software configurations and will aid you in designing a driver that satisfies your specific application.

Data Organization

Probably the most important aspect of the bubble memory system interface is the organization of data. From a software viewpoint, data logically is organized into blocks of bytes called "pages." During data transfer operations, one or more of these pages are transferred between the bubble(s) and the host microprocessor. A page is the smallest increment of data that can be transferred; single bytes cannot be transferred. Conceptually, the data organization within a bubble memory is analogous to a disk system. Just as disk sector sizes are fixed when a disk is formatted, bubble page sizes are established, under software control, when the bubble system is initialized. For a single bubble system, the page size is fixed at either 64 bytes when error correction is implemented or 68 bytes without error correction, and the total number of pages available is 2048. In systems with multiple bubbles, page size can vary from 64 bytes (68 bytes without error correction) to 512 bytes (544 bytes without error correction) depending on the number of bubble devices in the system. Page size is directly proportional to the system data rate and also determines the total number of available pages (address field size). As an example, consider a system consisting of two bubbles (using error correction). With two bubbles, there are two possible ways to configure the system; paralleling the two bubbles for a page size of 128 bytes and a total number of 2048 pages or treating the bubbles serially for a page size of 64 bytes and a total number of 4096 pages. The average data rate for the 128-byte page is 17.0 kbytes per second, and the average data rate for the 64-byte page is 18.5 kbytes per second.

The selection of the appropriate page size depends primarily on the data rate supported by the system. For file system implementation, an additional consideration in page size selection is bubble transfer efficiency versus bubble storage efficiency. Essentially, the following system factors must be weighed:

- Data Rate. The higher the data rate, the faster the microprocessor must respond to the demands of the bubble memory controller. Depending on the data transfer mode selected, some data rates may exceed the data transfer rate of the host microprocessor.
- Bubble Transfer Efficiency. A file consisting of a few large pages can be transferred more efficiently (faster) than a file consisting of a number of small pages.
- Bubble Storage Efficiency. For a typical file system, space must be allocated within each page to link the pages of each file together (individual pages of a file may not necessarily be contiguous). Too large or too small a page size can waste

bubble storage space. If most files are comprised of many small pages (e.g., 64 bytes), a large percentage of bubble storage would be required for establishing the fore/back pointer linkage. Conversely, if most files are smaller than a single page, a large amount of space would remain unused at the end of each page.

Buffering

Buffer operation is an extremely important factor in the reliable transfer of data between the bubble and system memory and is a major consideration in software driver design. A buffer, ideally speaking, is a memory storage area that contains the same amount of data storage as the data block to be transferred. The bubble system's bubble memory controller includes a first-in, first-out (FIFO) 40 byte data buffer that reconciles timing differences between the parallel data transfer to or from the host microprocessor and the serial data transfer to or from the bubble memory. Accordingly, when an application program requests data from a bubble, the software driver is responsible for keeping up with the FIFO for the duration of the data transfer in order to prevent the FIFO from overflowing or underflowing. The specific software driver requirements are dependent on the method of data transfer selected.

Data Transfer Interface Modes

Three distinct software interface techniques can be used to interface host system memory with the bubble system for page data transfer: DMA: interrupt driven, and polled.

In the DMA transfer mode, the BMC operates in conjunction with a DMA controller (e.g., Intel's 8257 or 8237) and uses the DRQ (data request) and DACK/ (data acknowledge) signal lines for establishing the handshake protocol. Assisted by the DMA controller, the BMC transfers data to or from the host system memory. Once the transfer begins, further program intervention is not required until the entire transfer has been completed.

In the interrupt-driven data transfer mode, the DRQ line is connected to an interrupt controller (e.g., Intel's 8259A). During non-DMA data transfers the DRQ line indicates when the BMC's FIFO is half-full (bubble read operations) or half-empty (bubble write operations). This method of interrupting results in a data block transfer arrangement in which the software is responsible for performing the appropriate transfer of data (typically 22 bytes) to or from the FIFO when the interrupt occurs. Using this technique, the software driver only processes the FIFO buffer as needed, and program waits during I/O transfers (polled I/O) are eliminated.

The polled I/O mode is the most simple to implement since no special or external hardware is required to perform data transfers. In the polled I/O mode, the software must determine when to transfer data to or from the FIFO by continually polling a status bit in the BMC's Status Register. This status bit indicates the presence or absence of data in the FIFO on a byte-by-byte basis. The polled I/O mode places significant demand on the host system's processing time since the software continuously must monitor the Status Register to ensure that FIFO overflow or underflow does not occur.

ECC Highlights

The last software controlled option to be considered in the design of your bubble driver is if the built-in error correction circuitry (ECC) within the 7242 Formatter/Sense Amplifier is to be implemented and, if so, what level of error correction is best suited to your application.

Although the inherent data integrity of your bubble memory is extremely high, the incorporation of error correction improves the overall integrity of your system by several orders of magnitude. While boosting the data integrity, the implementation of error correction adds increased overhead to both driver design and host interaction. Since the associated error handling routines can range from simple to complex, you must carefully weigh the software requirements before considering the level of error correction to be supported. In balancing driver and host responsibilities, you must understand the following factors pertaining to ECC:

- The type and nature of errors associated with bubble memories.
- The way in which ECC operates.
- The various levels of error correction available.

All of these factors are explained in detail in a later section.

COMMUNICATING WITH THE BMC

All communications between the host and the bubble memory actually are peformed through the 7220-1 BMC. The BMC has two input/output (I/O) ports; an 8-bit bidirectional data port and an 8-bit command/status port (Figure 1). The port addressed is determined by the least-significant bit of the port address byte. Conceptually, the BMC can be thought of as a disk system controller in that data in the bubble memory is organized into blocks called "pages" that are similar to disk sectors. Information such as starting page location, direction of transfer, and the number of pages to be transferred is passed to the BMC before the desired read or write operation is initiated.

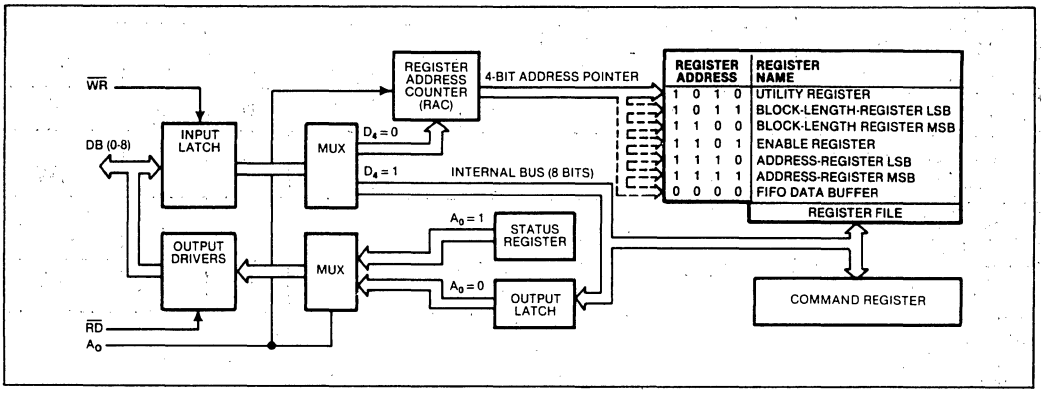


Figure 1. BMC Block Diagram

For simplicity, you can think of the BMC as a 40-byte FIFO (first-in first-out) buffer and a series of six user-accessible 8-bit registers. The FIFO passes data between the outside world and the bubble system's 7242 Formatter/Sense Amplifier and compensates for speed variations. The six registers are loaded prior to most operations and contain information regarding the upcoming transfer and the operating mode of the BMC. Since these registers always are loaded before a command is sent similar to passing parameters to a subroutine before it is invoked, this set of registers is referred to as the "parametric registers." The transfer of data between the host and the BMC's FIFO and parametric registers takes place over the 8-bit data port. The destination (FIFO or parametric register) of the data port transfer is determined by the value in another register called the RAC. As shown in Table 1, bit 4 of the command/status port byte is used to distinguish between a BMC command and either the address of one of the parametric registers or the FIFO.

Table 1. Command Port Function

Function	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Command	0	0	0	1	C	C	C	C
RAC	0	0	0	0	R	R	R	R

When bit 4 is a "one," the low-order four bits are decoded as a BMC command, and when bit 4 is a "zero," the low-order four bits are interpreted as a pointer to the parametric registers/FIFO. This 4-bit register pointer is referred to as the "Register Address Counter" or simply the "RAC."

RAC values that may be written to the command/status port and the registers selected are outlined in Table 2. The RAC points to, or selects, one of the six registers or the FIFO. Once a RAC value is written to the command status port, the next data port read or write operation transfers data between the host interface and the register addressed.

Table 2. Register Address Counter Assignments*

Register Name	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read/ Write
Utility Register	0	0	0	0	1	0	1	0	R/W
Block Length Register (LSB)	0	0	0	0	1	0	1	1	W
Block Length Register (MSB)	0	0	0	0	1	1	0	0	W
Enable Register	0	0	0	0	1	1	0	1	W
Address Register (LSB)	0	0	0	0	1	1	1	0	R/W
Address Register (MSB)	0**	0	0	0	1	1	1	1	R/W
7220 FIFO	0	0	0	0	0	0	0	0	R/W

NOTE(S):
* With A0 = 1
** Write-only bit

Referring now to the table, notice that the register addresses are in hexadecimal order from "A" to "F" and that the FIFO has an address of zero. This arrangement of addresses is due to the RAC's auto-incrementing feature. Once a register is selected, each subsequent data port I/O read or write causes the RAC to advance and to point to the next register in the sequence. After the most significant byte of the Address Register is addressed, the RAC advances from F to 0 to point to the FIFO. When it reaches this point it no longer increments. The system now is ready to transfer data into or out of the FIFO without further instructions from the host.

After the FIFO is selected, the RAC stops incrementing and continues to point to the FIFO until the RAC again is accessed through the command/status port. The auto-incrementing feature minimizes the number of instructions required for a given command sequence and ensures that all of the required parametric information is sent to the BMC.

As a user, you are not required to utilize the auto-incrementing feature; each parametric register can be selected and loaded in any order, and specific registers may be updated as required. When individual registers are not accessed in order, each register must be specifically addressed and loaded. Until you become more familiar with the bubble system, the auto-incrementing feature is recommended.

A point to remember is that once a command has been issued to the BMC, the parametric registers must not be updated until the operation is complete. The parametric registers essentially are working registers for the BMC during command execution. When a bubble read or write operation is in progress, the Block Length Register, as explained later in this chapter, contains the terminal page count and is decremented with each page transferred. Attempting to modify this register during command execution would cause the final page count to be incorrect.

The Parametric Registers

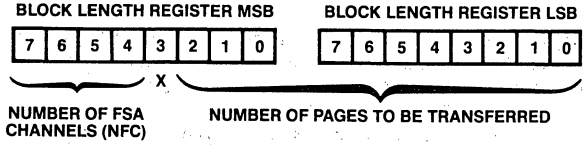
Now that you have been introduced to the Register Address Counter and its operation, let's look at the individual parametric registers addressed by the RAC in more detail.

Utility Register

The Utility Register is a user-defined register that can be both written and read. This register is not incremented or decremented by the BMC. Since the Utility Register is the first register of the RAC sequence, if the register is not used, the least-significant byte of the Block Length Register initially can be addressed to eliminate the Utility Register from the sequence. Note that the 4 Mbit bubble memory controller (7224) does not contain a utility register.

Block Length Register

The Block Length Register is made up of two 8-bit registers, a low-order byte register and a high-order byte register. The contents of the block length register determine the system page size and also the number of pages to be transferred in response to a single bubble data read or write command. The Block Length Register or "BLR" is a write-only register that is divided into a terminal count field and a channel field as follows:



The terminal count field is eleven bits in length and is loaded with the total number of pages to be transferred in the ensuing bubble read or write operation. With a field length of eleven bits, from 1 to 2048 pages can be transferred (all zeroes in the field indicates a 2048-page transfer).

The page width (size) is defined by the 4-bit channel field. This field actually specifies the number of formatter/sense amplifier channels available. Note that each 7242 formatter/sense amplifier has two channels to communicate with each bubble memory, therefore the acceptable values in this field select one channel (one half of a bubble memory), two, four, eight, or 16 channels. These field values correspond to page sizes of 32, 64, 128, 256, and 512 bytes (assuming error correction), respectively, when the bubble memories are operated in parallel. (The one-channel mode usually is reserved for diagnostic operations.) Table 3 shows the relationship among page size, channel field value, and formatter/sense amplifier channel selection for parallel bubble operation.

As shown in the table, the channel field bits are encoded and only one bit ever is set in the field.

For example, a channel field value of "0001" selects one bubble memory through channels 0 and 1.

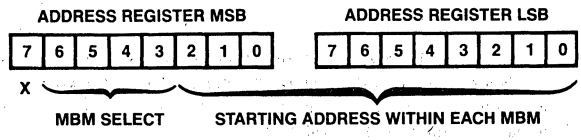
Table 3. FSA Channel Select/MBM Select

MBM Select AP, MSB Bits (6, 5, 4, 3)	"Channel Field" (BLR MSB Bits 7, 6, 5, 4)				
	0000*	0001	0010	0100	1000
0 0 0 0	0	0, 1	0, 1, 2, 3	0 to 7	0 to F
0 0 0 1	1	2, 3	4, 5, 6, 7	8 to F	
0 0 1 0	2	4, 5	8, 9, A, B		
0 0 1 1	3	6, 7	C, D, E, F		
0 1 0 0	4	8, 9			
0 1 0 1	5	A, B			
0 1 1 0	6	C, D			
0 1 1 1	7	E, F			
1 0 0 0	8				
1 0 0 1	9				
1 0 1 0	A				
1 0 1 1	B				
1 1 0 0	C				
1 1 0 1	D				
1 1 1 0	E				
1 1 1 1	F				

NOTE(S):
*Normally reserved for diagnostic operations.

Address Register

The Address Register, like the Block Length Register, is made up of two 8-bit registers; a low-order byte register and a high-order byte register. The Address Register is divided into a starting address field and an MBM (Magnetic Bubble Memory) select field show as follows:



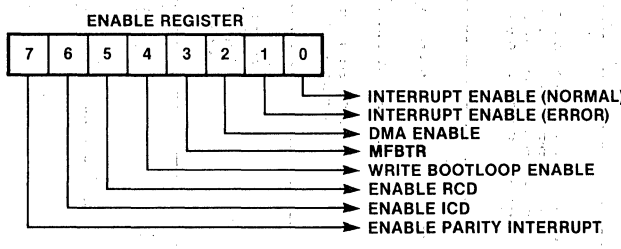
The Address Register's starting address field is eleven bits in length and is used to define on which page of a bubble's 2048 pages that the transfer is to start. The starting address field is incremented with each page transferred during multipage transfers, automatically selecting the next sequential page.

The Address Register's MBM select field is used in conjunction with the Block Length Register's channel field to control the serial selection of bubble memories or groups of bubble memories operated in parallel. To better understand the function of the MBM select field, consider a system consisting of four bubble memories operated as two banks of two bubble memories each.

Referring back to table 3, the channel field in the Block Length Register would be set to "0010" to select two bubbles in parallel and a corresponding page size of 128 bytes. To select between the two banks, the Address Register's MBM select field would be set to "0000" to select the first bank (FSA channels 0 through 3). As page 2048 is transferred to or from the first bank, the Address Register's starting address field rolls over to "0000" and increments the MBM select field to "0001" to select the second bank (FSA channels 4 through 7).

Enable Register

While the Address and Block Length Registers define the system configuration and data transfer, the Enable Register defines the various modes of operation under which the data transfer is performed and defines the conditions under which interrupts can be generated. Several of the Enable Register bits are used individually while other bits are used in combination. Figure 3 shows the individual Enable Register bit definitions.



Interrupt Enable (Normal), when set ("1"), enables the BMC to interrupt the host processor on the successful completion of a command. Conversely, if this bit is not set, an interrupt is not generated on command completion and the host processor must poll the BMC's Status Register to determine when command execution is complete.

Interrupt Enable (Error) is used in conjunction with the Enable RCD and Enable ICD bits to select various error conditions under which the BMC will terminate command execution and interrupt the host processor. The following table outlines the bits combination and corresponding error conditions recognized.

Table 4. Error Correction Options

Enable ICD (bit 6)	Enable RCD (bit 5)	Interrupt Enable (Error) (bit 1)	Interrupt Condition
0	0	0	No interrupt on error
0	0	1	Interrupt only on timing error
0	1	0	Interrupt on uncorrectable or timing error
0	1	1	*Interrupt on uncorrectable, correctable or timing error
1	0	0	Interrupt on uncorrectable or timing error
1	0	1	Interrupt on uncorrectable, correctable or timing error
1	1	0	Illegal
1	1	1	Illegal

NOTE(S):
*Normally not used.

DMA Enable, when set, enables the BMC to operate in the DMA data transfer mode. In the DMA mode, a DMA controller is interfaced to the BMC and DRQ-DACK/protocol is used to perform byte transfers. Note that in the DMA mode, the BMC activates its DRQ output *each time* it places a byte in the FIFO (bubble read operation) or each time there is room for *at least one byte* in the FIFO (bubble write operation). When the DMA Enable bit is not set, the BMC operates in the interrupt driven or polled mode. In either of these modes, the BMC's DRQ output goes active when 22 or more bytes are present in the FIFO during a bubble read operation or when there is space for 22 bytes during a bubble write operation. Note that if the DRQ is not used (i.e., polled mode), the host processor must examine the Status Register's FIFO Ready bit to determine when data should be taken from or written to the FIFO.

MFBTR, (Maximum FSA to BMC Transfer Rate), determines the maximum burst transfer rate from the FSA(s) to the BMC FIFO. This bit only applies to bubble read operations and is effective only during single-page transfers or during the transfer of the last page of a multipage transfer. Table 3 shows the effect of MFTBR bit on the transfer rate for parallel bubble operation.

Write Bootloop Enable, when set, enables the bootloop to be rewritten. Conversely, if this bit is not set and a Write Bootloop command is received, the command is aborted immediately and the Timing Error is set in the Status Register. Since writing the bootloop only is performed as a diagnostic test or to recover from a system failure, this bit provides protection against accidental rewrite of the bootloop data.

Enable RCD (Read Corrected Data), when set, causes the BMC to issue a Read Corrected Data instruction to all the FSAs in the system in response to one or more FSAs reporting an error. On receipt of the instruction, each FSA cycles the erroneous page through its error correction logic and then transfers the page to the BMC. For any FSA not reporting an error, the data harmlessly cycles through the error correction logic. When the page transfer is complete, the BMC interrogates the FSA to determine if the error was correctable (if the error was uncorrectable, erroneous data would have been transferred to the BMC).

Enable ICD (Internally Correct Data), when set, causes the BMC to issue an Internally Correct Data instruction. Any FSA reporting an error causes the instruction to be issued to all FSAs in the system.

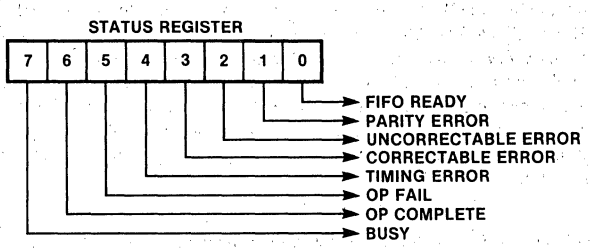
On receipt of the instruction, each FSA cycles the data through its error correction logic (regardless of whether it contained an error), but does not transfer the page to the BMC. After cycling the page, each FSA reports its error Status (correctable or uncorrectable) to the BMC. The FSA not reporting an error harmlessly cycles through the error correction logic and reports a correctable error to the BMC. Note that both the Enable RCD and Enable ICD bits cannot be set at the same time.

Enable Parity Interrupt, when set, enables the BMC to interrupt the host processor when it detects a parity error on a data byte sent from the host processor (the BMC automatically checks for odd parity on each data byte received from the host processor and implements odd parity on each byte sent to the host processor). When the Enable Parity Interrupt bit is not set and parity error is detected, no interrupt is generated (following the transfer, the Parity Error bit in the Status Register will be set).

Status Register

As stated at the beginning of this chapter, the host processor executes an I/O read instruction with the BMC's A0 address input equal to "1" to access the Status Register. As will become evident in the individual Status Register bit descriptions, the Status Register is read in response to interrupts from the BMC and, during polled data transfers, is read continually to determine when data is to be written to or read from the BMC's FIFO.

The Status Register also provides information regarding error conditions, the completion or termination of commands, and the BMC's readiness to accept new commands. The individual Status Register bits are shown as follows.



Note that bits 1 through 6 are valid only when the Busy bit (bit 7) is not set and that these bits are cleared whenever a new command is received. The Status Register also can be cleared by writing to the register address counter (RAC) with bit 5 set to "1" and any valid parametric register address (0AH through 00H).

Busy. Bit 7, the Busy bit, is set ("1") when the BMC is in the process of executing a command. The BMC sets its Busy bit shortly after a command is received (the Busy bit must be clear in order for the BMC to accept any command other than an Abort command) and keeps Busy set until the operation is complete (or until an operation is halted because an error has occurred.)

It is important to note that the Busy bit remains set until all other status bits have been updated and that it therefore is possible to see illogical bit combinations such as Busy and Op Complete at the same time. With the exception of the FIFO Available bit, all other Status Register bits should be considered valid only after the Busy bit returns to an inactive ("0") level.

OP Complete, Bit 6, Op Complete, is set when the BMC successfully completes the execution of a command.

OP Fail, Bit 5, Op Fail, is set when the BMC is unable to complete execution of a command. In general, this bit is valid only after the Busy bit returns to an inactive level; other error bits in the Status Register will be set to indicate why the operation failed.

Timing Error, Bit 4, Timing Error, is set for several error conditions. The most frequent cause of timing errors is when the host processor cannot keep up with the rate at which the BMC fills or empties its FIFO (referred to as an overflow or underflow condition). Timing errors also occur if the correct number of bits is not set in an FSA's bootloop register (to function properly, an FSA must have either 272 loops active when error correction is not implemented or 270 loops active when error correction is implemented). This condition occurs during a Read command if a mistake is made either when the bubble's bootloop is written or if the bootloop register is loaded incorrectly from the user's system. Another source of timing error is if no bootloop sync code is found during an Initialize or Read Bootloop command. The last source of a timing error is when a Write Bootloop command is received by the BMC and the Write Bootloop Enable bit is not set in the Enable Register. Of the preceding sources, regular or periodic occurrences of timing errors usually indicate an inherent inability of the host processor to meet the bubble's data transfer requirements and will be most apparent during bubble read operations, especially if the MFBTR bit is set (i.e., MFBTR=0 in the Enable Register).

Correctable Error, Bit 3, Correctable Error, is set when an FSA reports that a correctable error was detected during the last bubble read operation. Depending on the level of error correction selected, the setting of this bit indicates a correctable error in the current page held by the FSA, in the last page read, or in one of the pages read during a multipage transfer.

Uncorrectable Error, Bit 2, Uncorrectable Error, is set when an FSA reports an uncorrectable error during the last bubble read operation. Like the Correctable Error bit, the setting of this bit depends on the level of error correction selected and indicates an uncorrectable error in either the last page transferred or in the page currently held by the FSA.

Parity Error, Bit 1, Parity Error, is set when the BMC detects a parity error on the data byte sent from the host during a bubble write operation. If the Enable Parity Interrupt bit is set in the Enable Register, the BMC will terminate the write operation and interrupt the host processor when the parity error is detected.

FIFO Available, Bit 0, FIFO Available, is unique in that it is the only bit the Status Register that is valid when the Busy bit is set. During data transfer operations (i.e., when the Busy bit is set), the FIFO Available bit acts as a gate for the host microprocessor's data handling software. During bubble write operations, if the FIFO Available bit is set there is room for more data in the FIFO and the host microprocessor can proceed with the transfer. Conversely, if the FIFO Available bit is not set, the FIFO is full and the host must wait for the BMC to "catch up" before sending more data. During bubble read operations, if the FIFO Available bit is set, data has been placed in the FIFO by the BMC and can be taken by the host microprocessor; if the FIFO Available bit is not set, the data is not yet available.

FIFO

The BMC's first-in first-out (FIFO) buffer passes data between the user interface and the FSA. The primary purpose for the FIFO is to reconcile timing differences between both the user interface and the BMC and between the BMC and the FSAs.

The FIFO itself is 40 bytes wide and, including the FIFO's input and output latches and the BMC's input latch, can store up to 43 bytes. The FIFO is "dual ported" in that data can be written into one port while simultaneously being read from the other port.

When the BMC is busy executing a command, the FIFO functions as a data buffer, and when the BMC is not busy, the FIFO is available for use as a general-purpose FIFO. For this reason, the BMC is said to be in the general-purpose FIFO mode when it is not busy. During execution of commands that involve the transfer of data between the user interface and the FSAs, the data passes through the FIFO, and the status of the FIFO is indicated by the FIFO Ready bit in the BMC's Status Register. When the FIFO Ready bit is set ("one") during a write operation, there is space in the FIFO for more data, and when this bit is set during a read operation, data is present in the FIFO.

The BMC's DRQ output also indicates FIFO status. In the DMA data transfer mode, DRQ is used in conjunction with the DACK/ input from a DMA controller (e.g., in Intel 8257 or 8237) on the user interface to provide a standard DMA data transfer protocol. In the non-DMA transfer mode (polled or interrupt-driven data transfers), the DRQ output indicates that the FIFO is either half full or half empty according to the direction of the data transfer; during a write operation, DRQ is set when there is space for 22 more bytes, and during read operation, DRQ is set when there are 22 bytes present in the FIFO. Accordingly, when performing non-DMA data transfers, blocks of 22 bytes should be transferred to or from the FIFO so that the host processor does not spend a disproportionate amount of time servicing the DRQ-initiated interrupt or polling the BMC's Status Register.

The FIFO automatically is addressed after the last parametric register is written due to the self-incrementing nature of the Register Address Counter. Alternatively, the FIFO can be addressed explicitly by writing to address 0 of the Register Address Counter. Note that since byte 0 of the FIFO is cleared whenever the Register Address Counter is addressed, writing the parametric registers (or to the FIFO itself) must be avoided while the FIFO contains valid data. Also, when using the FIFO in the general-purpose mode, the host system is responsible for resetting the FIFO prior to any data transfer.

During read and write operations, the host system is responsible for keeping up with the data transfer in order to prevent a FIFO overflow or underflow condition. If the FIFO overflows or underflows during the data transfer, the operation is aborted and the Op Fail and Timing Error bits are set in the BMC's Status Register.

The FIFO AVAILABLE bit is also set whenever the RAC is not pointing to the BMC FIFO (RAC address = 00H). When writing the parametric registers, for example, if the user reads the status between say the Enable Register and the Address Register, the status byte would indicate FIFO AVAILABLE. This status value is forced by internal BMC logic.

COMMANDS

The BMC's command set consists of 16 commands that are selected by a 4-bit command code. As previously described, commands are passed to the BMC by writing to the BMC's command port with bit 4 of the command byte set to "1" (the Register Address Counter is addressed when bit 4 is "0"). Table 5 lists the 4-bit command codes and the corresponding commands.

For most commands sent to the BMC, the parametric registers must be loaded with operating information before the command is sent. Also, with the exception of the Abort command, commands cannot be issued to the BMC while another command is being executed (i.e., when the Busy bit in the Status Register is set). The remainder of this section offers brief descriptions of the BMC's command set; descriptions of command usage are presented in succeeding sections. Table 6 of this section presents a summary of command execution times.

The 16 commands can be grouped into categories according to their frequency of use. The most commonly used commands are:

- Abort
- Initialize
- Read Bubble Data
- Write Bubble Data

Other commands used during normal bubble system operation include:

- Read Seek
- Write Seek
- MBM Purge
- Read Corrected Data
- Reset FIFO
- Software Reset
- Read FSA Status
- Read Bootloop

The remaining commands are related to bootloop operation and are used only for diagnostic purposes:

- Read Bootloop Register
- Write Bootloop Register
- Write Bootloop Register Masked
- Write Bootloop

Abort

The Abort command (unlike any other command), when issued while the BMC is busy executing another command, causes the termination of the command being executed. On receipt of this command, the MBMs are stopped in an orderly manner to prevent the loss of bubble data. Since the Abort command is the only command recognized by the BMC while it is busy, this command is issued following power-up or whenever the BMC is in an unknown state. The Abort command requires no prior loading of the parametric registers.

When an Abort command is issued while the BMC is not busy, the command functions as an FIFO Reset to clear any data present in the FIFO. An Abort command issued when the MBM drive coils are active (i.e., data transfer command is executing indicated by the Busy bit in the Status Register) must be followed by an Initialize command.

Table 5. BMC Command Set

D3	D2	D2	D1	Command Name
0	0	0	0	Write Bootloop Register Masked
0	0	0	1	Initialize
0	0	1	0	Read Bubble Data
0	0	1	1	Write Bubble Data
0	1	0	0	Read Seek
0	1	0	1	Read Bootloop Register
0	1	1	0	Write Bootloop Register
0	1	1	1	Write Bootloop
1	0	0	0	Read FSA Status
1	0	0	1	Abort
1	0	1	0	Write Seek
1	0	1	1	Read Bootloop
1	1	0	0	Read Corrected Data
1	1	0	1	Reset FIFO
1	1	1	0	MBM Purge
1	1	1	1	Software Reset

Table 6. 7220-1 Command Execution Times

Four-Bit Command Code (Hex)	Command	Description	Performance
0	LRBLRMSK	1 FSA channel selected 2 FSA channel selected	900 μs 900 μs
1	Initialize	Best case (N=#MBM) Worst case	350+(85,200) N μs 350+(164,740) N μs
2	Read	Single page (MFBTR=0) Single page (MFBTR=1) N page transfer (MFBTR=0) N page transfer (MFBTR=1)	t_{SEEK}+8690 μs t_{SEEK}+12,770 μs t_{SEEK}+8690+(7500) (N−1) μs t_{SEEK}+12,770+(7500) (N−1) μs
3	Write	1 page transfer N page transfer	t_{SEEK}+7450 μs t_{SEEK}+7450+(7500) (N−1) μs
4	Read Seek	Best case Worst case	7350 μs 89,250 μs
5	Read BLR	Any number of FSA channels	900 μs
6	Write BLR	Any number of FSA channels	900 μs
7	Write BL	Single bubble selected	82,850 μs
8	Read FSA Status	1 bubble in system N bubbles in system	75 μs 75+40 (N−1) μs
9	Abort	1 bubble in system N bubbles in system	100 μs 100+40 (N−1) μs
A	Write Seek	Best case Worst case	7350 μs 89,250 μs
B	Read BL	Best case Worst case	86,000 μs 165,000 μs
C	Read Corrected Data	Any number of FSAs selected	1400 μs
D	FIFO Reset	N/A	50 μs
E	MBM Purge	N/A	150 μs
F	Software Reset	N/A	50 μs

Initialize

The Initialize command prepares the bubble system for subsequent operations and is used when the bubble system is powered up (following the Abort command) or whenever the system's error correction implementation is changed. The Initialize command effectively performs the following BMC commands:

- Abort
- MBM Purge
- FIFO Reset
- Read Bootloop
- Write Bootloop Register Masked

When an Initialize command is received, all internal registers within the BMC are cleared and the FIFO is reset. The BMC then reads the bootloop from each bubble and writes the corresponding bootloop information into the bootloop registers of each FSA. The bubble is left positioned at page zero (logically) corresponding to the values set in the BMC page address counters. Before an Initialize command can be issued, the following information must be loaded into the parametric registers:

- The channel field in the Block Length Register must be set to "0001" to arrange all bubbles in the system in a serial configuration to allow the individual bootloops to be read from each bubble and subsequently written to the bootloop registers of the corresponding FSA channels.
- The MBM select field in the Address Register must select the last (highest numbered) bubble in the system to inform the BMC of the number of bubbles present within the system (for a one bubble system the value would be "0000").
- The bits selecting error correction in the Enable Register must be set according to how subsequent read and write operations are to be performed (with or without error correction) since the number of 1's written to the FSA bootloop registers is not the same with error correction implemented as when error correction is not implemented. Note that simply switching between ECC modes (level 1, 2, or 3) does not require re-initialization.

Read Bubble Data

The Read Bubble Data command causes data to be read from the MBMs and into the BMC's FIFO. Immediately before the Read Bubble Data command is issued, the host computer must load, with the exception of the Utility Register, all of the parametric registers. Specifically, the following parametric information must be loaded prior to command execution:

- The channel and "number of pages to be transferred" in the Block Length Register must be set to define the page size (number of FSA channels) and number of pages to be transferred.
- The appropriate bits must be set in the Enable Register to select the transfer mode (DMA or non-DMA) and interrupt sources; if error correction is to be used (i.e., if the FSA bootloop registers have been initialized for error correction), the level of error correction must be selected.
- The MBM select and starting address fields in the Address Register must be set to define the (first) MBM and page within the MBM where the transfer is to occur.

Write Bubble Data

The Write Bubble Data command causes data read from the BMC's FIFO to be written into the MBMs. Immediately prior to issuing the Write Bubble Data command, the host computer must load the Block Length, Enable, and Address Registers as described for the Read Bubble Data command. Data should not be loaded into the BMC FIFO until after the command has been issued. Note that a write data transfer does not begin until at least two bytes of data have been loaded into the BMC FIFO.

Read Seek

The Read Seek command is used to reduce system access time of a subsequent Read Bubble Data command by positioning the page to be read at the selected bubble's output track. Note that although the Read Seek command does not cause any data to be transferred, the following information must be loaded into the parametric registers before the command is issued:

- The channel field in the Block Length Register must specify the page size (number of FSA channels).
- The error correction bits in the Enable Register must be set identical to the values used when the system was initialized.
- The MBM select field in the Address Register must be set to select the bubble containing the page to be read and the address field must specify a page number that is one less than the (first) page to be read by the subsequent Read Bubble Data command.

Write Seek

The Write Seek command is used to reduce the system access time for a subsequent Write Bubble command by positioning the page to be written at the selected bubble's input track. Note that like the Read Seek command, the Write Seek command does not cause any data to be transferred. Similarly, the following information is issued:

- The channel field in the Block Length Register must specify the page size (number of FSA channels).
- The error correction bits in the Enable Register must be set identical to the values when the system was initialized.
- The MBM select field in the Address Register must be set to select the bubble containing the page to be written and the address field must specify a page number that is one less than the (first) page to be written by the subsequent Write Bubble Data command.

MBM Purge

The MBM Purge command is used in place of the Initialize command when the bootloop register is to be loaded from an external source (once a bootloop has been identified, the bootloop register pattern can be maintained by the host and loaded directly from external memory with a Write Bootloop Register or Write Bootloop Register Masked command to conserve power and increase speed during an initialization sequence). The MBM Purge command clears the BMC's internal registers and the address field of the Address Register. The MBM select field in the Address Register and the other parametric registers are not cleared. Like the Abort command, the MBM Purge command does not require the loading of the parametric registers prior to command execution.

Read Corrected Data

The Read Corrected Data command is used only when error correction is implemented and only is applicable when error correction level 2 or 3 is selected (the Read Corrected Data command is issued automatically by the BMC when error correction level 1 is selected). When an FSA reports a correctable error, the Read Corrected Data command is issued to cause the corrected page in the FSA(s) to be read into the BMC's FIFO. Note that since the parametric registers previously were loaded for the read operation in which the error was detected and since the address field is not incremented (i.e., the address of the page in error is in the address field), it is not necessary to load the parametric registers prior to issuing the Read Corrected Data command.

Reset FIFO

The Reset FIFO command causes the BMC's FIFO (and its input and output latches) to be cleared. Normally, this command is issued prior to issuing the Write Bootloop, Write Bootloop Register, or Write Bootloop Register Masked commands to ensure that the FIFO will accept 40 bytes; the Reset FIFO command does not require loading of the parametric registers.

Software Reset

The Software Reset command clears the BMC's internal registers, and the terminal count field in the Block Length Register and starting address field in the Address Register. Additionally, the Software Reset command causes the BMC to clear the status register of every FSA channel in the system. Since the Software Reset command does not clear the FSA channel and MBM select fields in the Block Length and Address Registers or any of the Enable Register bits, it is not necessary to reinitialize the parametric registers following a Software Reset command, and no loading of the parametric registers is required prior to command execution.

Read FSA Status

The Read FSA Status command causes the BMC to read the 8-bit status register of each FSA channel in the system (the number of FSA channels specified in the channel field of the Block Length Register). This command is issued to determine the number of FSAs in a system. Following command execution, the individual status register bytes will be in the BMC's FIFO in ascending order; one byte per FSA channel. All values returned to the host will be 20H or 28H if error correction is enabled. This occurs because the FSA status is cleared when read. The BMC always reads the FSA status prior to interrupting (or informing) the host of an error. Therefore, the Read FSA status command can only read the "cleared" FSA status values (20H or 28H). No loading of the parametric registers is necessary prior to command execution.

Read Bootloop Register

The Read Bootloop Register command causes the BMC to read the bootloop register of the selected FSA channel into its FIFO. This command is used initially to ensure that the bubble system is communicating properly (bubble-to-FSA and FSA-to-BMC communication established) and is used to transfer bootloop information to the host system for subsequent bootloop initialization from an external source (e.g., user EPROM). Prior to command execution, the channel field in the Block Length Register and the MBM select field in the Address Register must be loaded with the number of FSA channels (normally two) and the corresponding bubble (FSA channel pair) to be selected. Note that since each individual FSA channel's bootloop register contains 20 bytes, reading a pair of FSA channels fills the BMC's 40-byte FIFO; reading the bootloop registers of more than two channels is possible but not recommended since data must be taken from the FIFO to avoid an overflow condition. Also, since the bootloop register data from each FSA channel pair actually is interleaved on a bit-by-bit basis before it is assembled into bytes, reading the bootloop register for a single channel likewise is not recommended. Remember that a unique BMC status value (C1H or C3H) is expected with this command (see "Considerations for Polled Command Execution").

Write Bootloop Register

The Write Bootloop Register command causes the BMC to write the contents of its FIFO into the bootloop register of the selected FSA channel(s). This command (and the Write Bootloop Register Masked command) is used during bubble system initialization when the bootloop is written from an external source rather than from the bubble itself. In order to use the Write Bootloop Register command, the bootloop register data must be loaded into the FIFO prior to command execution and the channel and MBM select fields of the Block Length and Address Registers must be loaded with the number of FSA channels (normally two) and the corresponding bubble (FSA channel pair) to be selected. Recalling that each individual FSA channel's bootloop register contains 20 bytes, 40 bytes normally are written into the FIFO to initialize the two FSA bootloop registers associated with each bubble. However, a 41st byte of zeroes *must* be written to ensure a successful command execution status. Note that the parametric registers should be loaded prior to loading the FIFO.

Proper operation of the bubble system requires that each FSA bootloop register contains either 135 (error correction selected) or 136 (error correction disabled) logic "1" bits that correspond to the 135 or 136 valid data storage loops.

Write Bootloop Register Masked

The Write Bootloop Register Masked command is identical to the Write Bootloop Register command previously described with the exception that the number of "1" bits written automatically is stopped when the required 135 or 136 logic "1" bits have been written for each channel. The parametric registers are loaded as described for the Write Bootloop Register command; the number of logic "1" bits written (135 or 136) is determined by the setting of the error correction bits in the Enable Register.

Write Bootloop

The Write Bootloop command causes the existing contents of the selected bubble's bootloop to be replaced by the 40 bytes of information currently contained in the BMC's FIFO. This command is used only after it has been determined that the existing bootloop is invalid (i.e., a storage loop previously identified as "good" has become defective) and typically is not required for the life of the bubble system. Remember that the parametric registers must be loaded prior to pre-loading the FIFO with the 40 bytes of information to be written followed by a 41st byte of zeroes.

If it should become necessary to use the Write Bootloop command, the channel field in the Block Length Register and the MBM select field in the Address Register must be loaded with the number of FSA channels and the corresponding bubble to be selected. As an additional safeguard, the Write Bootloop Enable bit in the Enable Register additionally must be set (if this bit is not set, command execution immediately will be aborted and the Timing Error bit will be set in the Status Register).

Read Bootloop

The Read Bootloop command causes the BMC to read the bootloop of the selected bubble into its FIFO. This command is used to determine if the existing bootloop is valid by comparing the bubble bootloop information to that on the label of the device. Remember that the parametric registers must be loaded prior to command execution. Also note that a BMC status value of C1H or C3H is expected (see "Considerations for Polled Command Execution").

ERROR CORRECTION

As mentioned earlier in this application note, several factors pertaining to error correction must be understood and weighed according to your specific application before implementing error correction. From a software perspective, the selection of the appropriate level of error correction is based on the host system's requirements relative to error logging and data recovery. Before describing the individual levels of error correction and the associated software requirements, the types of errors that can occur within bubble memories are described as a preface to error correction.

Bubble Errors

To understand the function and implementation of error correction, the differences between bubble errors and semiconductor device errors must be distinguished. In conventional semiconductor memory, errors are classified as either hard (non-recoverable) or soft (recoverable). Usually, hard errors are the result of physical or irreversible damage within the device itself (e.g., oxide breakdown or junction burnout), and soft errors are the result of transient conditions and generally do not reappear when the data is rewritten.

Unlike semiconductor devices, bubble memory devices have no active elements and, as such, rarely experience hard (non-recoverable) errors. Accordingly, all bubble memory errors can be considered as soft errors (for information on irreversible failure mechanisms in bubbles, refer to the Intel 7110 Bubble Memory Reliability Report, RR-36 Order Number 210632). In order to further define the nature of soft errors in bubbles, errors are classified either as "data" errors or "read" errors. A data error occurs when a data inversion occurs within a storage loop; the data is lost, but since no physical damage has occurred, the data can be rewritten and the bubble can remain operational. Data errors typically occur when the bubble is operated beyond its safe operating region and, as such, are seldom encountered during normal operation. The most common type of soft error is a "read" error that occurs during a bubble read operation, usually as a result of noise in the detection/sense circuitry. Since the data in the storage loop is unaltered during a bubble read, the data can be recovered by simply repeating the read operation.

Error Detection/Correction Capability

In respect to the Formatter/Sense Amplifier (FSA), errors either are correctable (the FSA is able to reconstruct the data using an error correction algorithm before the data is transferred to the BMC) or uncorrectable, irrespective of the type of error (data error or read error). The error correction code used by the FSA is a 14-bit Fire code that is appended to each 256-bit block of data. This code is capable of correcting all single error bursts up to, and including, five bits in length and has proven to be well suited to the error model for the 7110 MBM. Table 7 outlines the FSA's error correction capability and probability of page errors; the bit error rate can be obtained by dividing the "probability of page in error" by the number of bits per page.

ECC Options

The FSA's error correction circuitry (ECC), in conjunction with the BMC, provides three levels of error correction. Each level places unique demands on the host system that range from simple data recovery to the logging of specific pages in which the error occurs. The desired level of error correction is selected by the setting of the appropriate bit or bits within the BMC's Enable Register. Table 8 defines the relevant Enable Register bits for the three levels of error correction available.

Table 7. FSA Error Detection/Correction Capability

Type of Error	Approximate Probability of Page in Error	% Correction	% Detection
Single Read Error	10^{-6}	100	—
Single Data Error	10^{-7}	100	—
Random Double Read Error	10^{-12}	—	100
Read Error Burst Length 2	—	100	—
Data Error Burst Length 2	—	100	—
Read Error Burst Length 3/4	10^{9}	100	—
Random Double Data Error	10^{14}	—	100
Read Error Burst Length 5	—	100	—
Random Triple Read Error	10^{17}	—	100
Single Soft + Read Burst 2	—	—	100
Undetected/Uncorrected Error Escape Rate	10^{13}	—	—

NOTES:

1. Read errors are recoverable by retry or error correction.
2. Data errors are recoverable by error correction methods only.

Table 8. Error Correction Level Selection

Error Correction Level	Enable Register Bit		
	Bit 6 ICD	Bit 5 RCD	Bit 1 Interrupt Enable (Error)
Level 1	0	1	0
Level 2	1	0	0
Level 3	1	0	1

In typical bubble memory systems, the prevention of the erroneous transfer of data to the host is primary concern, and the actual location of the error (error logging by page) is secondary. This fact is especially true if the error is correctable. Both the prevention of transferring erroneous data and error logging of the page in error, however, can be satisfied by error correction, and you must select the error correction level that best fits your specific application.

ECC Operation

When error correction is implemented, each page of data written to the MBM contains additional bits for the ECC code. For example, in a single-bubble system like the BPK 72, the page size decreases from 68 bytes to 64 bytes per page when error correction is implemented. As each page of data is read from the MBM and assembled into the FSA, the FSA's error correction circuitry checks the integrity of the data. When an error is detected (in a completely assembled page), the FSA notifies the BMC (by activating its ERR.FLG/ line) and sets the appropriate bits within its internal status register according to whether or not the error is correctable. The BMC, depending on the level of error correction selected and the nature of the error (correctable or uncorrectable), either issues a command to the FSA to continue the transfer (transparent to the host) or interrupts the host and waits for additional instructions before proceeding.

The point at which the data is transferred between the FSA and the BMC's FIFO is the key to understanding the difference among the three levels of error correction. Once this timing is understood, it will become easier to see how the levels differ in terms of the interrupts generated and the intervention required by the host (software driver) during the transfer. Note that while the BMC generates an interrupt when an error is detected, it is not mandatory for the host to support interrupts; the host can read the BMC's Status Register at the completion of command execution to determine the nature of the error condition.

During a bubble read operation with error correction enabled, the FSA senses and formats the data and places the data in its two 270-bit serial FIFOs (one FIFO for each channel). The data is read in the fully buffered mode where a full page (512 bits) plus the additional 28 ECC bits are read into the two FIFOs before any data is transferred to the BMC's FIFO. By fully buffering the data, the FSA can detect an error and notify the BMC before any data is transferred. If no error is detected by the FSA, the contents of the FSA's FIFOs are written to the BMC's FIFO while the next page from the MBM is being read into the FSA (note that the 28-bits of ECC code are never transferred to the BMC's FIFO). In order to correct an error once it has been detected, the FSA cycles the data through its error correction network. Once the data is cycled, the "corrected" data is either automatically transferred to the BMC (transparent to the host) or the data is held in the FSA FIFOs awaiting further instruction from the host. As an alternative to using the "corrected" data (error correction level dependent), the page in error could simply be reread when an error was reported since a majority of the errors encountered are "read" errors. The following paragraphs describe ECC operation during each of the three levels of error correction.

Level 1

Level 1 is the minimum level of error correction and is used only when the host system is concerned with maintaining bubble data integrity. As an example of this type of application, consider a bubble-based operating system that is downloaded into user RAM for execution. With this application, the primary concern is that the entire operating system is transferred correctly rather than where the error occurred. As will be seen, Level 1 is well suited to applications where go/no-go types of data transfers are required.

If an error is detected during Level 1 operation, the FSA activates its ERR.FLG/ line to the BMC. The BMC, in response, automatically issues a Read Corrected Data (RCD) command to the FSA which causes the FSA to cycle the data through its ECC network and update its status register, and then to transfer the data to the BMC. If the soft error was correctable, valid data would have been transferred to the BMC; the BMC would increment its Address Register to the next page to allow the operation to continue, and an interrupt would not be generated (i.e., the entire operation would be transparent to the host system). However, if the soft error was beyond the capability of the FSA's error correction circuitry (i.e., an uncorrectable error), invalid data would have been transferred to the BMC. The BMC, after reading the "uncorrectable error" status from the FSA, stops command execution and interrupts the host, and does not increment its Address Register. Since the host is aware that an uncorrectable error has occurred, the proper response is to repeat the entire read operation (by reloading the parametric register and reissuing the read command) since the erroneous page already has been transferred to the host.

Note that since the BMC does not increment its Address Register after reading the "uncorrectable error" status from the FSA, it is possible to perform page-specific logging of uncorrectable errors.

Level 2

Level 2 differs from Level 1 in that the transfer of erroneous (uncorrectable) data to the BMC is prevented and successive retries of the page in error are possible since the Address Register is not incremented. As with Level 1, correctable errors are transparent to the host system.

When a soft error is detected during Level 2 operation, the BMC automatically issues an Internally Corrected Data (ICD) command to the FSA.

In response to this command, the FSA cycles the data through its error correction network and updates its status register to indicate if the error is correctable or uncorrectable, but does not transfer the data to the BMC. The BMC, in turn, reads the FSA's status register and updates its own Status Register. If the error is correctable, the BMC automatically issues an RCD command to the FSA (to transfer the corrected data) and increments its Address Register to the next page address to allow the read operation to continue. Like Level 1 operation, the transfer of the corrected page is transparent to the host; the subtle difference between Level 1 and Level 2 is the time required to execute since the data is cycled through the error correction network twice (first by the ICD command and again by the RCD command). While the second correction cycle does not change the data, it does, however add approximately 350 milliseconds to the transfer operation.

If the soft error is uncorrectable, command execution is halted on the page in error and the BMC interrupts the host system. When interrupted, the host system can read the BMC's Address Register to determine the page in error and can issue a subsequent Read command (without reloading the parametric register) to retry the page. If, when the page is reread, the error does not recur (i.e., if the uncorrectable error is a "read" error), command execution continues with the next page. If successive retrys are unsuccessful (i.e., if the uncorrectable error is a "data" error), the page most likely will have to be rewritten. The host system, however, can examine the erroneous page by issuing an RCD command to the BMC to transfer the uncorrectable data from the FSA. Following the transfer of the erroneous page, the BMC again will interrupt the host and will not increment its Address Register. Note that the uncorrectable data transferred to the BMC will not necessarily match the erroneous data originally read from the MBM since the FSA's error correction circuitry attempts to correct the data.

Level 3

Level 3 offers the most complete means of error handling and, at the same time, is the most software intensive level since the host system is interrupted when either a correctable or uncorrectable error is encountered. Accordingly, error logging may be performed on pages containing correctable as well as uncorrectable errors, and an unlimited number of retrys can be attempted on the erroneous page. As with Level 2, the transfer of erroneous data to the BMC can be prevented.

When a soft error is detected during Level 3 operation, the BMC automatically issues an ICD command to the FSA. Like Level 2 operation, the FSA cycles the data through its error correction network and updates its status register, but does not transfer the data to the BMC. The BMC, in turn, reads the FSA status register, updates its own Status Register, and interrupts the host system even if the error is correctable. When interrupted, the host system must examine the BMC's Status Register to determine if the error is correctable or uncorrectable and can log the address of the page in error by reading the BMC's address register (the Address Register is not incremented). Note that it is not necessary for the host to support interrupts as the host can continuously poll the BMC's Status Register for an error interrupt.

If the error is correctable, the host system can either issue an RCD command (to transfer the corrected data from the FSA to the BMC) or can retry the page by issuing subsequent Read commands. Note that by retrying pages with correctable errors, the host system can distinguish between "read" and "data" errors since consistently correctable errors on a page indicate a "data" error (a "read" error would not recur when the page was reread).

If the error is uncorrectable, the host system only would retry the erroneous page by issuing additional Read commands and, if successive retrys were unsuccessful, would have to rewrite the page. As with Level 2 operation, the host system can issue an RCD command to transfer the uncorrectable data from the FSA.

Status Register

When using error correction, the host system can read the BMC's Status Register at the time of the interrupt or at the completion of command execution to ascertain additional information relating to the error condition. The relevant bits of the STR are bits 6, 5, 3 and 2; the remaining bits are irrelevant to error correction.

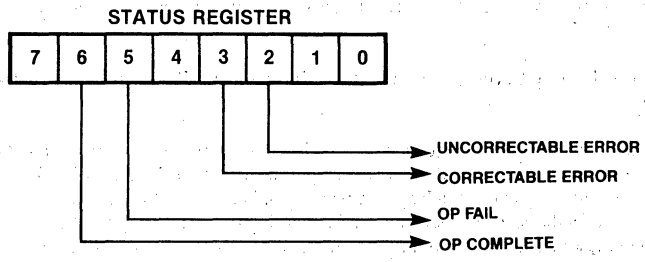


Table 9 lists the possible status indications that could occur when using error correction.

Table 9. Error Correction Status Indications

STR Bit Set	Error Correction Level	Indication
3	3	Correctable error detected, address of page containing the error is in the AR.
5 and 2	All	Uncorrectable error detected, address of page containing the error is in the AR.
6	All	Operation complete, no errors detected.
6 and 3	1 and 2	Operation complete, one or more correctable errors detected and corrected.
6 and 2	1	Operation complete, uncorrectable error detected on last page of multipage transfer.
5, 3 and 2*	1 and 2**	Very Rare. During a multipage transfer, one or more correctable errors detected and corrected on one or more pages, and a subsequent uncorrectable error detected on a page other than the last page of the specified transfer.
6, 3 and 2*	1	Extremely rare. During a multipage transfer, one or more correctable errors detected and corrected on one or more pages, and a subsequent uncorrectable error detected on the last page of the specified transfer.

*These status indications may occur on single-page transfers in Multi-MBM systems when one MBM has a correctable error and another MBM has an uncorrectable error.

**This status indication may occur in level 3 in Multi-MBM systems when more than one MBM has an error on the same page and at least one of the errors is correctable and one of the errors is uncorrectable.

DRIVER DESIGN

This section contains specific software interface examples that outline important considerations associated with the various ways in which a bubble system can be operated. As will be explained, command execution can be performed either in an interrupt driven mode or in a polled mode irrespective of the data transfer mode (polled, interrupt-driven, or DMA) selected to effectively provide the six unique operating modes shown in Figure 2. Since both polled and interrupt driven command execution are common to all three data transfer modes, details concerning command execution are discussed prior to examining specific examples of each data transfer mode. Before you begin to design your software driver, it is recommended that you thoroughly understand the information presented in this section.

Command Execution

To better understand the software driver's responsibilities, it is helpful to separate command execution into two phases; a command phase and a result phase. During the command phase, the driver generally (command dependent) loads the parametric registers, issues the desired command, then verifies that the command has been accepted; during the result phase, the driver determines the success or failure of the command issued either by polling the BMC's Status Register (polled mode) or through an interrupt service routine (interrupt-driven).

- Polled Mode. After loading the parametric registers (if required), the software driver issues the desired command, checks to see if the command was accepted, and then continuously polls the BMC's Status Register for an Op Complete indication.
- Interrupt-Driven Mode. After loading the parametric registers (if required), the software issues the desired command, checks to see if the command was accepted, and then waits for the interrupt to occur at successful command completion. Note that the interrupt Enable (Normal) bit must be set in the Enable Register to allow the BMC to generate the interrupt at successful command completion. Also, since an interrupt is not generated if the intended operation fails, additional provisions must be included to avoid "hanging" the system while waiting for the command completion interrupt. Error interrupts are reported according to the settings of the three "error correction" bits in the Enable Register.

Another important fact to note is that the BMC does not support the typical two-way handshaking common to most interrupt-driven peripheral devices and that interrupts from the BMC are cleared (or acknowledged) either when a subsequent command is issued or when the Register Address Counter (RAC) is written with the modifier bit (bit D5) set to "one" and with bits D3 through D0 set to "zero"; interrupts are not cleared by reading the BMC's Status Register.

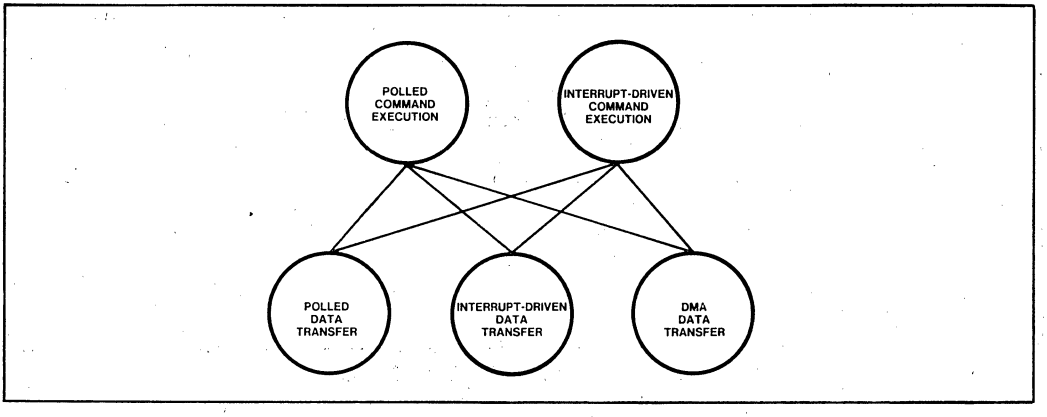


Figure 2. Bubble System Operating Modes

Considerations For Polled Command Execution

The operation of the BMC (and in particular, the BMC's Status Register) during polled command execution is the key to software driver development. Figure 3 shows the activity of the BMC at various stages during a typical command execution sequence. The disussion of the BMC's Status Register is centered around the status bits associated with command execution (i.e., the BUSY, OP COMPLETE, and OP FAIL status bits) although as will be explained, additional Status Register bits can have an effect on the state of the BUSY bit.

Before the command is sent at time T0, the BUSY bit is clear and, in fact, must be clear in order for the BMC to accept a new command (other than an Abort command). Sometime between T0 and T1, the BUSY bit is set to indicate that the command has been accepted and that command execution has begun. Note that all software examples in the remainder of this chapter include a check to ensure that the BUSY bit has been set after a command has been issued.

If the Status Register is examined between T1 and T2, the status byte value will be 80H (BUSY bit set). Depending on the command issued, the FIFO AVAILABLE bit may set at T2 if it is necessary for the host to initiate the transfer of data (see "polled data transfers" later in this section). Whether or not the FIFO AVAILABLE bit is set, the only normal status byte values between T2 and Tn-1 are 80H or 81H (BUSY and FIFO AVAILABLE).

At time Tn (completion of the command), the Status Register indicates the success or failure of the command. Generally, the BUSY bit is cleared at this time (indicating that command execution has terminated); an exception occurs if the FIFO still contains 22 or more bytes of data at command completion (i.e., when the BMC completes its internal microcode routine). During execution of read commands, it is possible for the host to leave data within the FIFO, and if 22 or more bytes are remaining, the BUSY bit will remain set even though command execution has been completed.

If the command is executed successfully and if the FIFO is empty, the status byte value will be either 40H (OP COMPLETE) or 42H (OP COMPLETE and PARITY ERROR). If data was loaded into the FIFO during the interval between T2 and Tn-2, the possible status byte values will be:

41H or 43H if the FIFO contains less than 22 bytes
C1H or C3H if the FIFO contains 22 or more bytes

In the case of unsuccessful command execution, several additional status byte values again are possible depending on the command being executed. Generally, the OP FAIL bit is set in the Status Register along with additional bits that indicate the nature of the failure (e.g., TIMING ERROR or PARITY ERROR). As with successful command execution, the BUSY bit is cleared when command execution is completed unless the FIFO still contains 22 or more bytes.

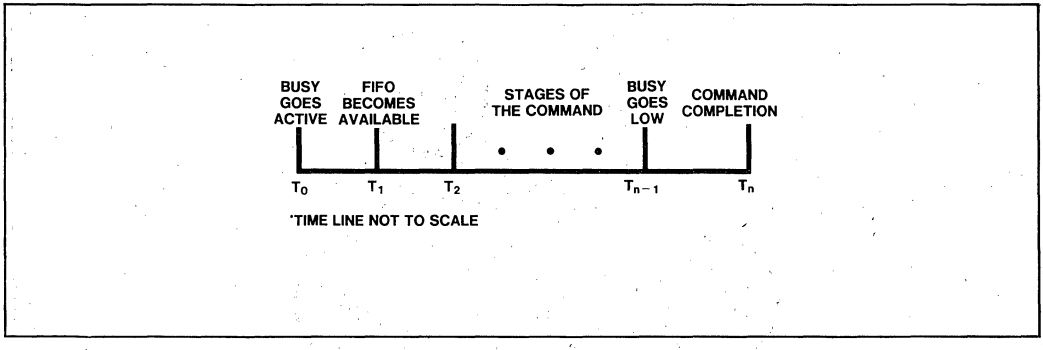


Figure 3. Command Execution Stages

Considerations For Interrupt-Driven Command Execution

As previously mentioned, when the Interrupt Enable (Normal) bit is set, an interrupt occurs only when a command is executed successfully as indicated by the setting of the Op Complete bit in the Status Register. If the Interrupt Enable (Error) bit is not set in the Enable Register and an error occurs that causes the operation to fail, an interrupt will not be generated. The user system must take this fact into account and provide either a fail-safe timer in hardware or a timeout counter in system software to guard against the possibility of "hanging" the system while waiting for an interrupt.

The following possible conditions are applicable to interrupt-driven operation:

1. After issuing a command, the host processor halts and waits for the interrupt to occur. Under this condition, the system will hang if the expected interrupt never occurs. A fail-safe timer connected to a low-level input of an interrupt controller (e.g., an Intel 8259A) would eliminate this problem.
2. In a "true" interrupt driven system where the host processor performs other tasks while the bubble command is being executed, the fail-safe timer in hardware again would be used although it is possible to use a software timeout counter. When operating in the non-DMA data transfer mode (polled or interrupt-driven data transfers), an interrupt service routine would be used to complete the data transfer and then either would wait for the command interrupt or would continuously poll the Status Register until command execution was successfully completed or until a software counter timed out (indicating unsuccessful command execution). However, devoting an excessive amount of time to polling by the host defeats the intention of an interrupt-driven system and reaffirms the need to provide a "watchdog" timer in hardware. In the DMA data transfer mode, the logical approach to detecting when execution of a command has failed is to provide a fail-safe timer in hardware that generates a timeout interrupt to the host.

An additional consideration common to all three data transfer modes is the fact that when an interrupt occurs in response to the successful execution of a command, data still may be present in the BMC's FIFO. If the interrupt service routine immediately acknowledges receipt of the command completion interrupt at the source (i.e., at the BMC), any data remaining in the BMC's FIFO when the interrupt is cleared would be destroyed. To eliminate this potential problem, an intermediate level of interrupt control would be required (again an Intel 8259A) to allow the interrupt routine to complete the transfer. The details described in the "Interrupt-Driven Data Transfer Mode" section will help to clarify the interrupt service routine requirements necessary to avoid leaving any data in the FIFO at command completion.

Data Transfer Modes

As mentioned at the beginning of this section, both polled and interrupt driven command execution can support any of the three data transfer modes. Regardless of the data transfer mode used, the basic operation of the BMC is the same for each data transfer mode. During all data transfers (i.e., during execution of a Read Bubble Data or Write Bubble Data command), the BMC continues to transfer pages of data until the page count in the Block Length Register is satisfied. Since the BMC's FIFO cannot hold a complete page of data, the host processor is required to "keep up" with the BMC as it continues to read data from, or to write data to, the MBM until all data has been transferred. At the beginning of read/write command execution, a "seek" operation is performed to locate the designated page. Once the addressed page is located within the MBM, the read data transfer begins (a write data transfer does not begin until at least two bytes of data have been loaded into the FIFO).

Polled Data Transfers

The polled data transfer mode is the most time-consuming in terms of processor overhead while at the same time is the easiest mode to implement. To support the polled data transfer mode, it is essential that the Status Register bit definitions be clearly understood and, in particular, the function of the FIFO AVAILABLE bit, since the interpretation of the bit changes according to the direction of the transfer.

From an operational viewpoint, the FIFO AVAILABLE bit acts as a gate for the FIFO-handling software. During a bubble write operation, if the FIFO AVAILABLE bit is set, there is room for additional data, and if the FIFO AVAILABLE bit is clear, the FIFO is full. During a bubble read operation, if the FIFO AVAILABLE bit is set, data has been placed in the FIFO by the BMC, and if the bit is clear, the FIFO is empty.

The BUSY bit indicates when the controller is in the process of executing a command. As previously described, the BUSY bit is set within a few microseconds following receipt of the command and remains set until the operation is completed (successfully or unsuccessfully). Remember that since the BUSY bit is internally gated with the BMC's DRQ output signal, it is possible during a read operation to obtain such logically exclusive status indications as BUSY and OP COMPLETE if the host fails to empty the FIFO following comand execution (during non-DMA data transfers, the DRQ signal acts as a "half full/half empty" flag for the BMC's FIFO).

Later in this section, a basic polled data transfer mode read/write routine is flowcharted. Note that to allow the status byte to be considered valid only after the BUSY bit is cleared and to avoid concurrent setting of the BUSY and OP COMPLETE bits at command completion, a running byte counter is implemented in software to ensure that all bytes have been removed from the FIFO. While the FIFO AVAILABLE bit alone can be used to gate the data to or from the FIFO, a byte counter is required to determine when the specified number of bytes has been transferred. Also note that a FIFO Reset command is sent prior to issuing the Write Bubble Data command as a "safety measure." Although issuing the FIFO Reset command is not mandatory, resetting the FIFO is recommended if there is any doubt regarding the state (emptiness) of the FIFO prior to initiating command execution.

Interrupt-Driven Data Transfers

The interrupt-driven data transfer mode requires less processor overhead than the polled data transfer mode (the polling wait time is eliminated) and also allows the data to be transferred in blocks rather than in individual bytes. The actual data transfer rate is fixed; the benefits of this mode only can be realized if the interrupt service routine is efficient (fast) enough to allow additional time for processing other tasks and is based on the host processor's execution speed. Accordingly, if the interrupt-driven data transfer mode is selected, make sure that the additional hardware and software required provide the desired performance increase.

The interrupt-driven data transfer mode is based on the fact that the BMC's DRQ line doubles as a "FIFO half full/FIFO half empty" indicator when the BMC is not in the DMA Mode. Physically, the DRQ line is connected either directly to the host's interrupt input or to an interrupt controller as the interrupt source for the data transfer. When an interrupt occurs, the host processor transfers a block of data to or from the FIFO in a single burst.

The recommended size of the data block transferred is 22 bytes. This block size was selected because transferring 22 bytes, and not 20 bytes (half of the FIFO), accounts for the two additional bytes held in the BMC's internal FIFO input and output latches while also optimizing the buffering capability of the BMC's FIFO. Since 22-byte block transfers rarely are exact multiples of the total number of bytes transferred, an efficient method must be devised to transfer the last remaining bytes of the transfer. While several methods are possible, the following method ensures that an interrupt is issued to transfer all of the "remainder" bytes (i.e., less than 22 bytes never will be "left" in the FIFO to prevent the DRQ line from going active and initiating the last block transfer).

As an example, assume that a two-page (128 byte) read or write data transfer is to be performed. The initial DRQ interrupt indicates that at least 22 bytes should be transferred. However, the first block transferred would be 18 bytes that correspond to the "remainder" bytes (128 mod 22 is 18). By first transferring 18 bytes, five subsequent interrupts occur (each initiating a 22-byte block transfer) to guarantee that no additional polling of the Status Register is required to transfer any "leftover" bytes. The calculation of the remainder count used for the first execution of the interrupt service routine is the responsibility of user software.

DMA Data Transfer Mode

In terms of a bubble system, direct memory access or DMA is the transfer of data between system memory and the BMC without host processor assistance or intervention. Since host processor involvement with the actual data transfer is not required, the overhead associated with software controlled (non-DMA) data transfers is eliminated.

Overall system performance is the primary factor for considering the DMA data transfer mode. That is, if the overhead associated with the available software-controlled data transfer modes degrades system performance to an unacceptable level, the implementation of DMA may prove to be the ideal solution. However, to support the DMA data transfer mode, additional hardware is required (e.g., a programmable DMA controller).

From a system perspective, the DMA hardware definition will dictate the software requirements. In a fixed system, discrete devices may be configured to perform a majority of the data transfer operations, including memory addressing, transfer direction, and terminal count (i.e., the number of bytes to be transferred).

The implementation of a programmable DMA controller such as an Intel 8257 or 8237 provides additional flexibility by allowing DMA to be performed under program control. Since it is not possible to describe all possible configurations, the BMC's operation during DMA transfers is described followed by a specific software example using an Intel 8257 DMA Controller.

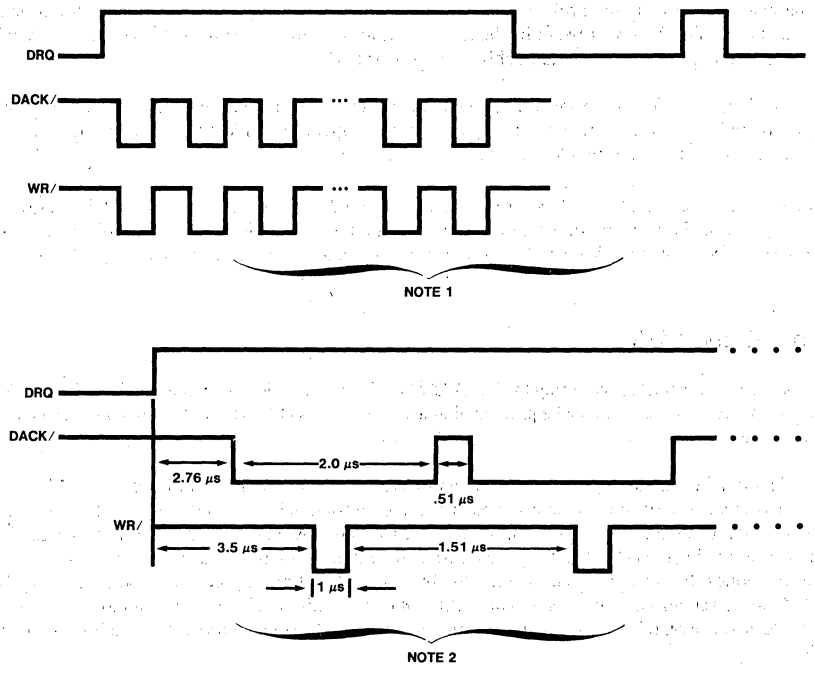
BMC Operation During DMA

The DMA data transfer mode is selected by setting the DMA ENABLE bit in the BMC's Enable Register. In the DMA mode, the BMC supports a standard two-way handshake protocol that uses the BMC's DRQ (Data Request) output signal to interact with the DMA hardware.

To initiate a DMA data transfer (assuming that the DMA hardware has been properly initialized), the BMC's parametric registers are loaded and a read or write command is issued to start the transfer. When the BMC is ready to transfer a data byte to or from system memory, it activates its DRQ output signal. The DMA hardware responds to DRQ by first gaining control of the system bus, then activating the DACK/ signal to the BMC, and finally making a read or write request to the BMC. The BMC responds to the request by placing a data byte on the bus (read operation) or accepting the byte on the bus (write operation). The DRQ signal remains active as long as the BMC either has additional bytes to transfer to system memory (and the FIFO contains at least one byte of data) or expects additional bytes from system memory (and the FIFO contains at least one empty location).

Referring to figure 4, when a Write Bubble Data command is issued, the BMC continuously requests (holds DRQ active) bytes from system memory until the FIFO is filled (i.e., data initially is transferred in a "burst" mode). Once the FIFO is filled, the BMC issues subsequent DRQ requests on a byte-by-byte basis until the last bytes have been transferred. When command execution is complete, the BUSY bit is cleared and the OP COMPLETE bit is set in the Status Register. If the INTERRUPT ENABLE (NORMAL) bit is set in the Enable Register, an interrupt is generated.

When a Read Bubble Data command is issued, the BMC activates DRQ only after the first data byte has been assembled and placed in the FIFO. While the DMA hardware is responding to the DRQ, additional bytes enter the FIFO at a minimum rate of 80 microseconds per byte (the single-MBM system transfer rate; the transfer rate increases according to the number of MBMs operated in parallel). Typical DMA response times usually are fast enough to take each byte before the next byte arrives, and data is transferred on a byte-to-byte basis.



NOTES:
1. THE DATA TRANSFER SEQUENCE INITIALLY OPERATES IN A "BURST" MODE (TYPICALLY 44 TO 46 BYTES ARE TRANSFERRED).
2. THE TIMING SHOWN IS ONE EXAMPLE OF VALUES OBTAINED DURING NORMAL OPERATION. THE VALUES MAY VARY IN YOUR SYSTEM PROVIDED THEY ARE WITHIN SPECIFICATIONS.

Figure 4. DMA Handshake Timing During Read/Write Command Execution

START-UP PROCEDURES

Whenever power is applied to the bubble memory system, a powerfail reset circuit is activated to satisfy specific power/timing sequences required by the BMC. All bubble system designs must include a powerfail reset circuit to ensure proper initialization of the bubble system.

The primary function of the powerfail reset circuit is to ensure that the bubble memory system powers up correctly. A built-in hardware time delay allows the BMC to complete its internal power-up sequence prior to enabling the additional support components; a software time delay is needed before any commands can be issued to the BMC. Following the delay, the first user communication with the BMC must be an Abort command whether the power-up is a cold start (application of power) or a warm restart routine (a RESET/ pulse applied to the 7220-1 BMC). Note that the status register should not be considered valid until the Abort command has been issued. A complete power-up flowchart, including initialization, is shown in figure 5.

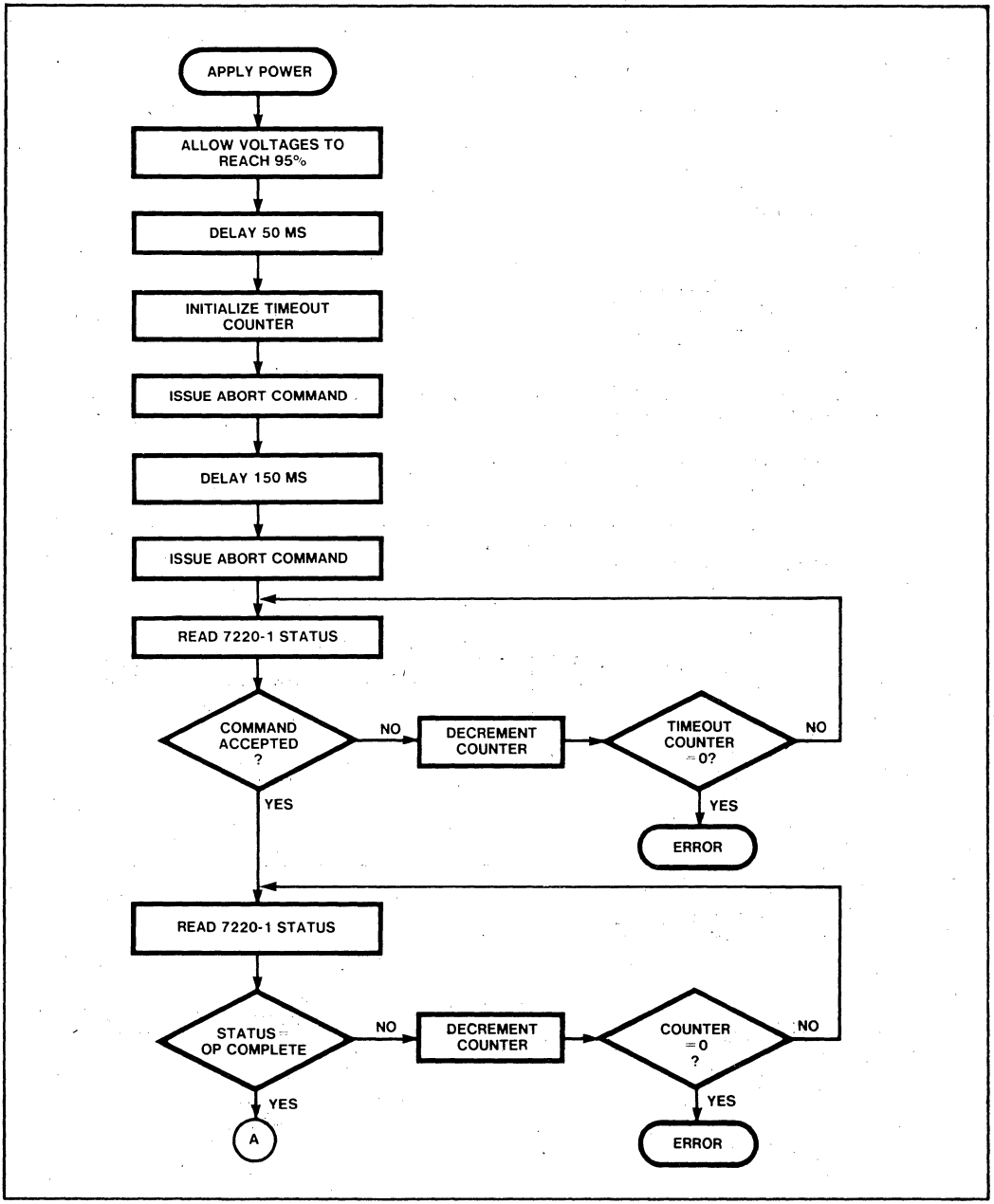


Figure 5. Power-Up Sequence (Polled Command Execution)

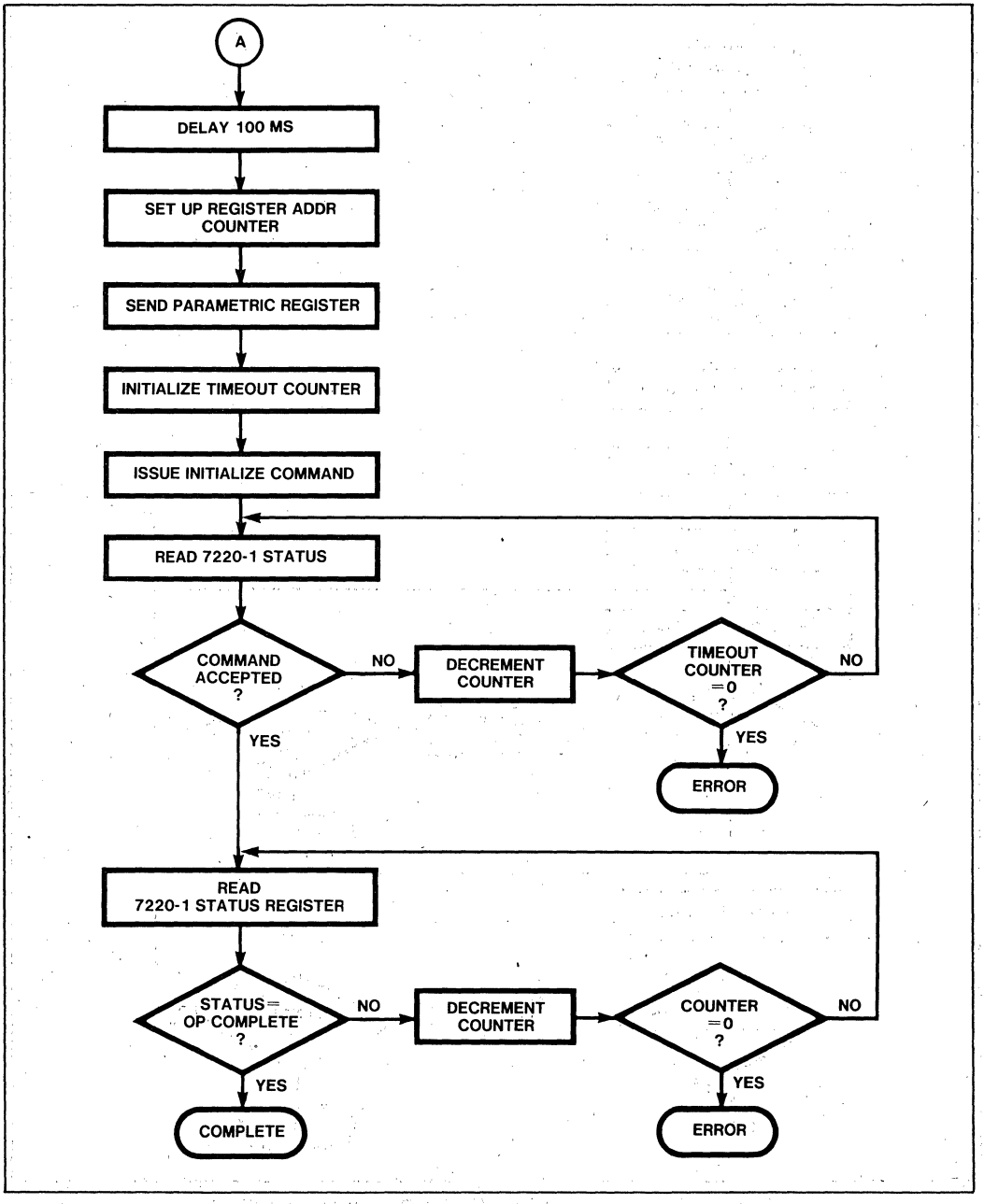


Figure 5. Power-Up Sequence (Polled Command Execution) (Cont.)

Initialization

Following successful execution of an Abort command, the user must initialize the bubble memory system using the Initialize command. The Initialize command requires that the parametric registers first are loaded with specific values. The software flowchart (figure 5) shows one example for polled command execution without error correction for a one-bubble system (i.e., the BPK 72 or iSBX 251). The Block Length Register always must be loaded with the values shown while the MBM Group Select bits in the Address Register always must select the last MBM in the system. For your particular application, set the appropriate bits in the Enable Register prior to issuing the Initialize command. Note that error execution mode changes require re-initialization of the bubble system.

The flowchart example polls the Status Register to see whether or not the initialization was successful. The OP COMPLETE bit indicates success. If an initialization problem occurs, the TIMING ERROR and OP FAIL bits will be set in the Status Register.

In an interrupt-driven system, the interpretation of the INT (interrupt) line depends on the setting of specific bits in the Enable Register. If INTERRUPT ENABLE (NORMAL) is set in the register, on the *successful* completion of command execution, the INT line will activate on the user interface. If INTERRUPT ENABLE (ERROR) is set in the register, the INT line will activate when an error condition occurs (in this case a TIMING ERROR). The user should devise interrupt service routines that can differentiate between the two interrupt sources by polling the Status Register. It is apparent from this discussion that a system that relies solely on interrupts may find it necessary to incorporate a watchdog timer (timeout counter interrupt) depending on the selection of the interrupt enable bits. A timeout counter avoids the possibility of never receiving an interrupt if a command is *unsuccessful* (this condition is possible if only the INTERRUPT ENABLE (NORMAL) bit is set).

If a system uses the BMC's DRQ signal for data transfers (DMA or interrupt mode), special precautions are necessary during Intialization. First, it is recommended that the DMA Enable bit in the Enable Register be disabled (i.e., set to 0) for the Initialize command. Once the Initialize command is successfully completed, the DMA Enable bit is enabled for the subsequent DMA data transfer.

When the BMC is in the non-DMA mode, the DRQ pin acts as a half full/half empty flag for the BMC's FIFO. Since the FIFO is used to temporarily store bootloop data during execution of the Initialize command, the DRQ pin is toggled. User-written software should therefore take appropriate precautions with respect to the DRQ pin during initialization (i.e., mask all interrupts, disable DMA controllers, etc.).

The Initialize command establishes a "working subset" of the bootloop information from the bubble and places this information into the bootloop registers of the FSA. A masking algorithm within the BMC reduces the total of 320 possible good storage loops to a working subset of 270 or 272 good loops (error correction dependent).

If the bootloop information in the MBM is incorrect, the system can inadvertently appear to be initialized correctly. As a one-time check to ensure that the system has been initialized correctly, simply read the bootloop registers from the FSA and count the number of "1's."

The count should yield:

- 270 "1's" out of 320 with error correction implemented.
- 272 "1's" out of 320 without error correction.

NORMAL OPERATION EXAMPLES

The following software flowcharts outline the important considerations and provide examples of each mode of operation previously discussed. First, figures 6 and 7 show the two possible command execution techniques. Note that in each case, the appropriate additional software must be inserted according to the data transfer mode selected. Figures 8 through 10 detail each data transfer mode. For any commands that do not involve the transfer of data (i.e., Abort, FIFO Reset, etc.), no additional software is required to be inserted in the command execution examples.

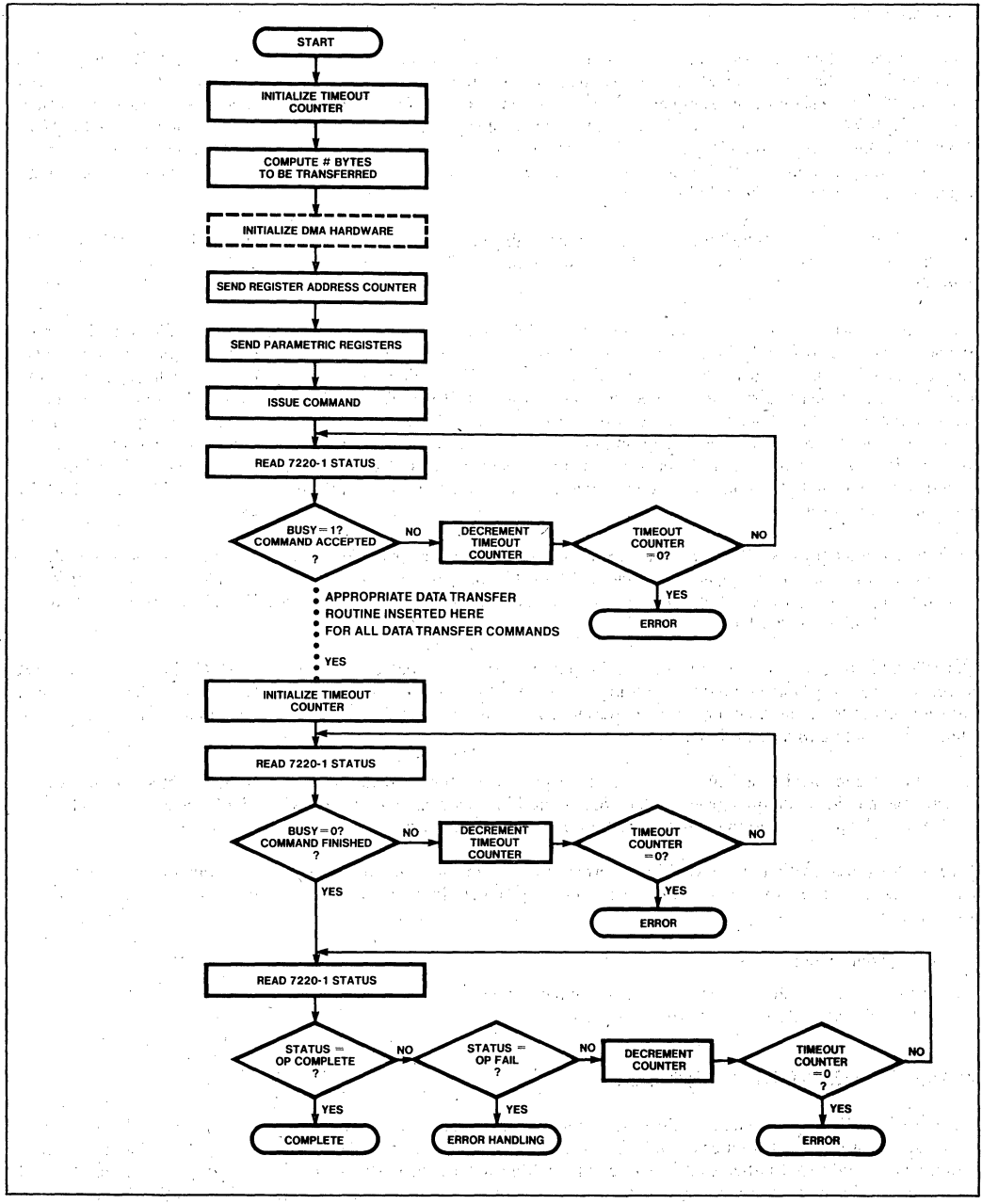


Figure 6. Polled Command Execution Routine

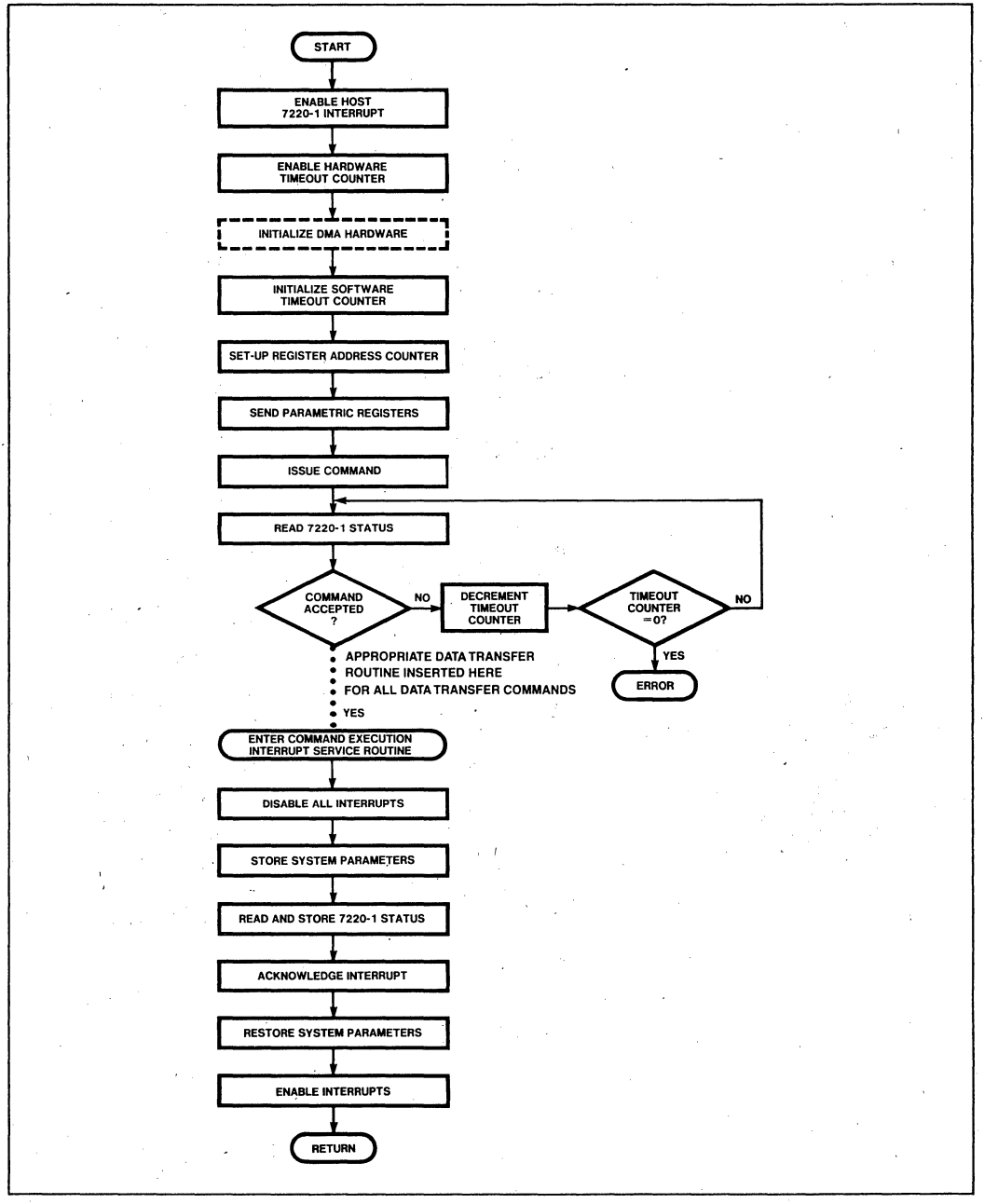


Figure 7. Interrupt-Driven Command Execution Routine

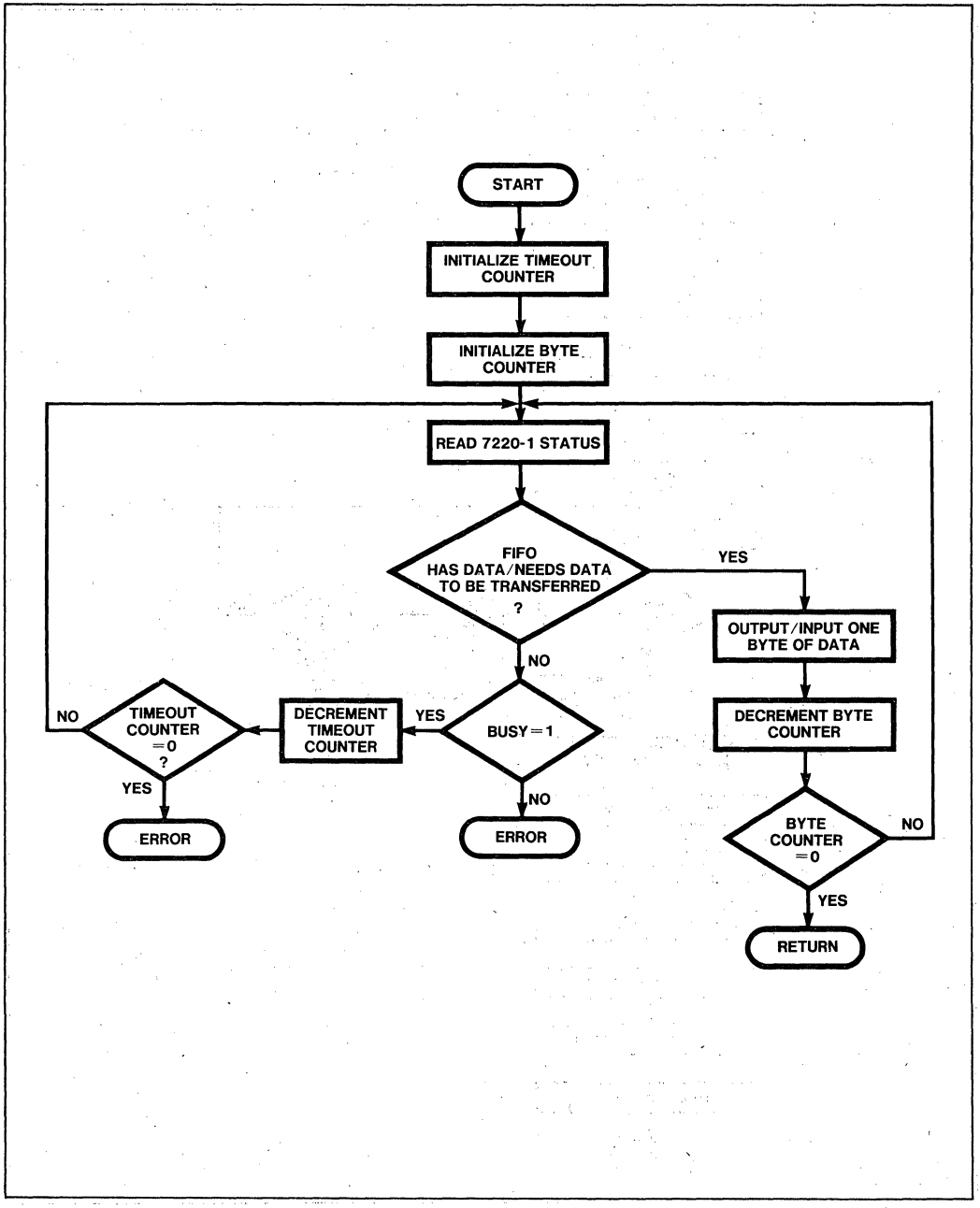


Figure 8. Polled Data Transfer Routine

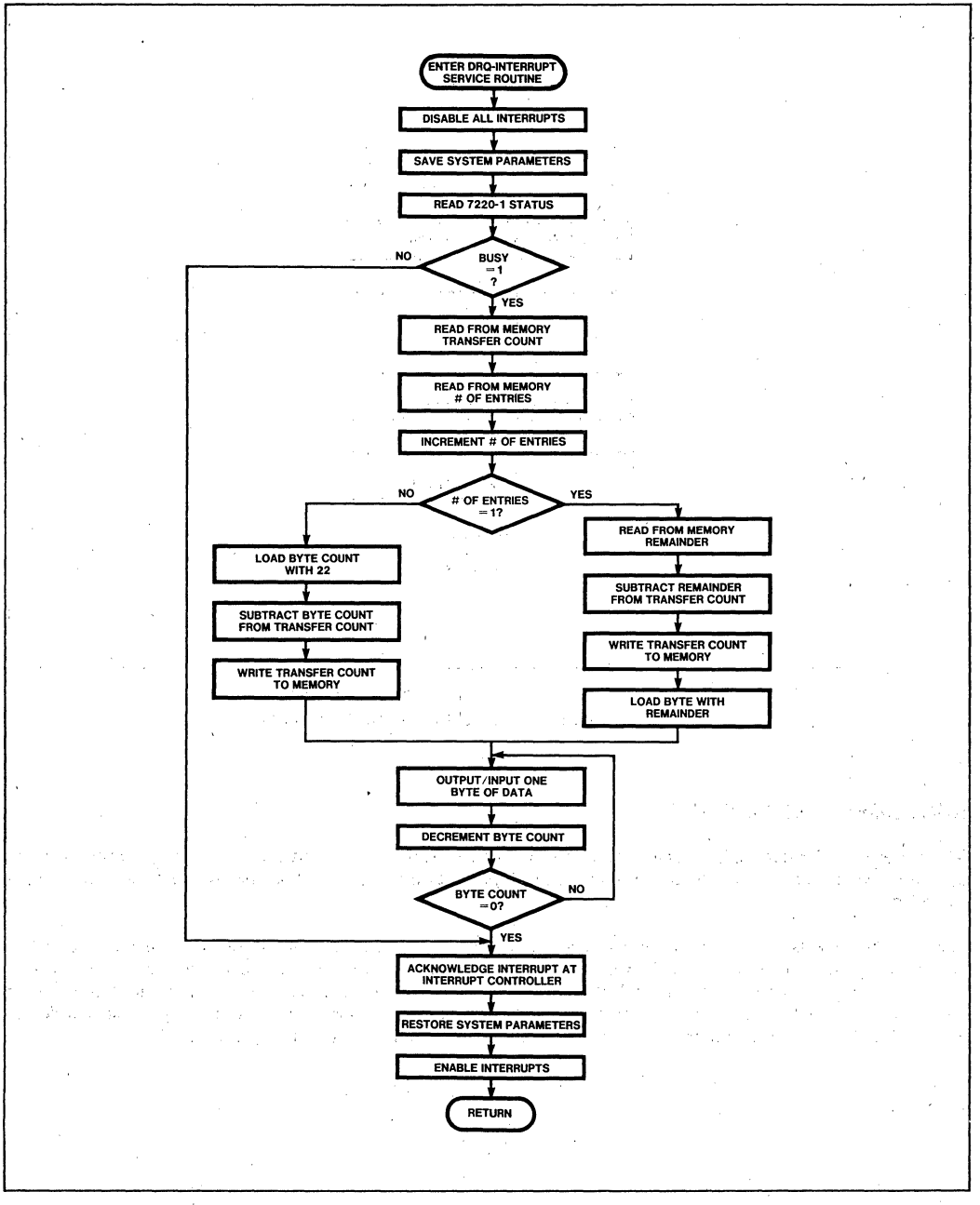


Figure 9. Interrupt-Driven Data Transfer Routine

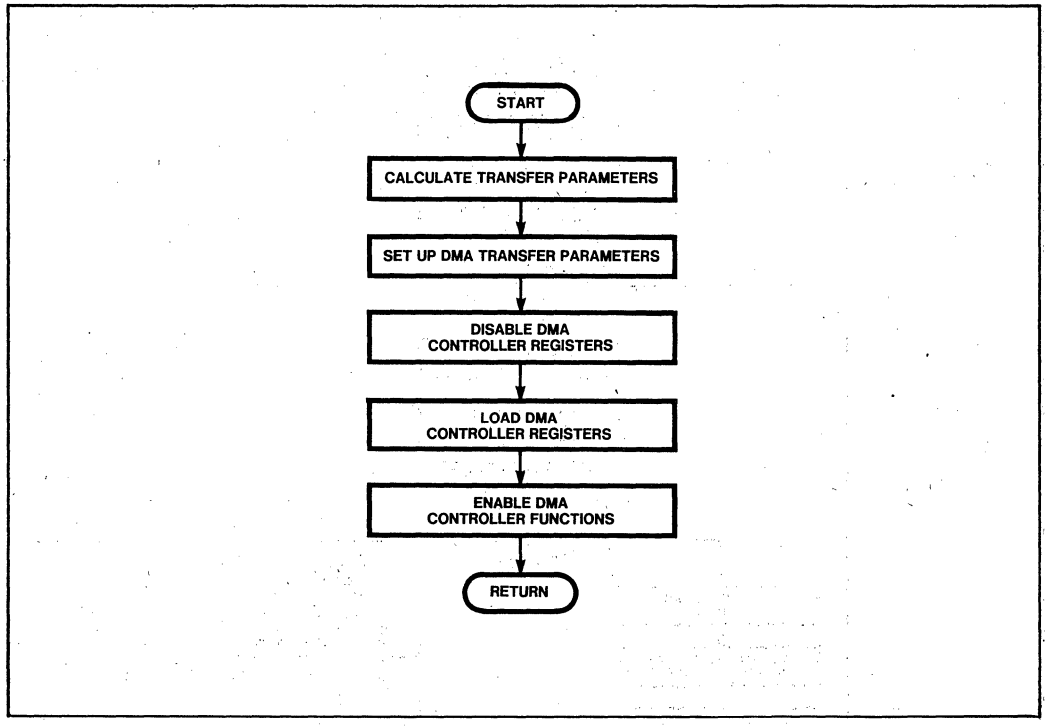


Figure 10. DMA Hardware Initialization Routine (Assumes an Intel 8257 DMA Controller)

SHUT-DOWN PROCEDURES

The power down procedure is the same regardless of whether the user voluntarily powers down the bubble memory system or power is inadvertently lost. The only special precaution is to ensure that the voltage decay rates are not exceeded.

The 7230 Current Pulse Generator contains a special powerfail detection circuit. The purpose of this circuit is to detect when the power supplies fall below threshold levels. Such an event automatically initiates an orderly shutdown of the rotating magnetic field and control signals for the 7110 function generators, in such a manner that no MBM data will be lost or invalidated, provided the voltage decay rates are met. When power is restored, the software implementation details described in the Start-Up procedures should be observed to ensure correct powerfail circuit operation.