intel

APPLICATION NOTE

AP-119

December 1982

# Microprocessor Interface for the BPK 72

**Paul Wells**
Application Engineer

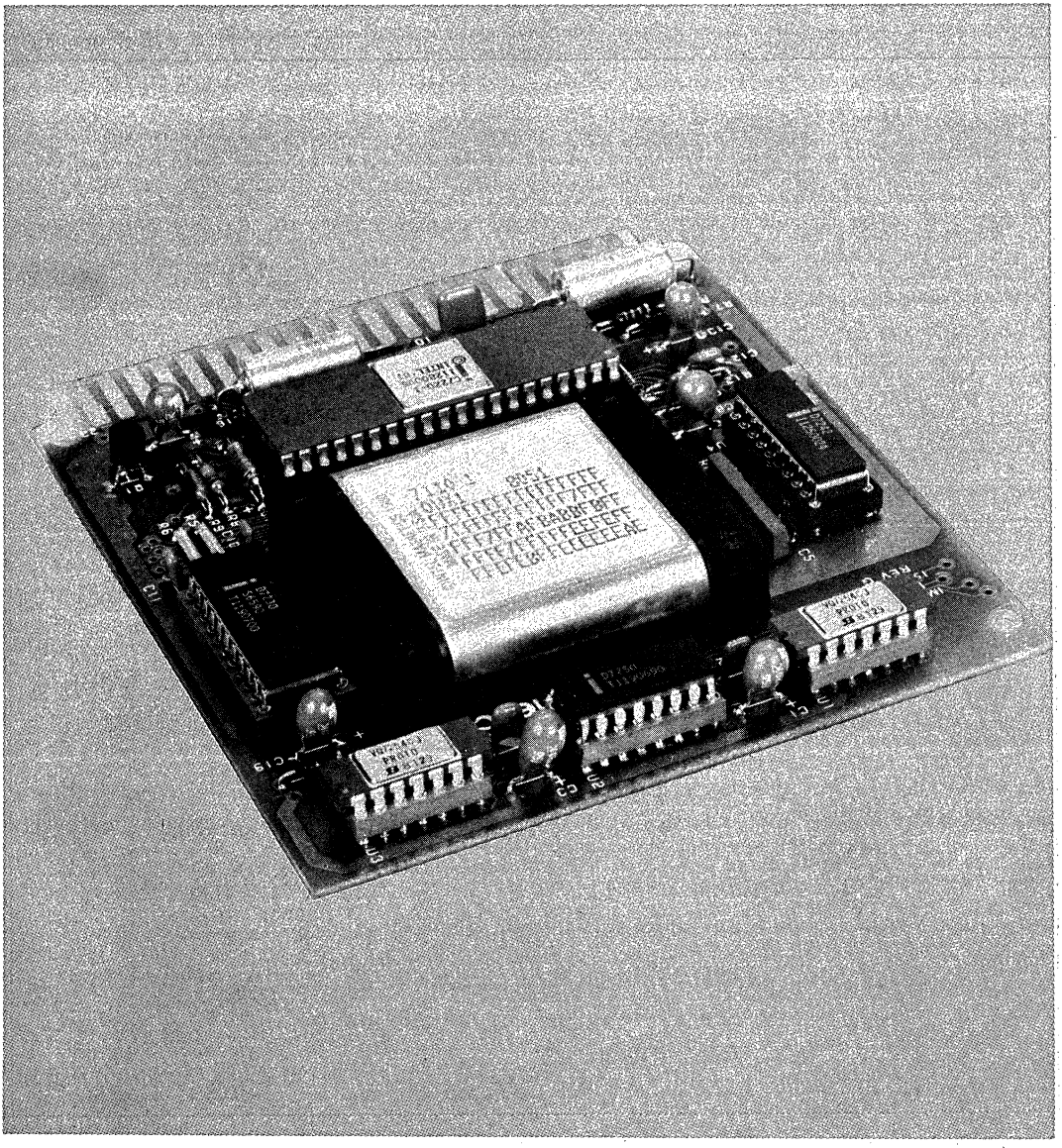
ORDER NUMBER: 210367-002

## INTRODUCTION

To date, a major obstacle in the implementation of bubble memories in systems has been the inherently complex control requirements imposed by the bubble memory devices themselves. With the advent of Intel's BPK 72 bubble memory prototype kit, a design engineer can immediately realize the benefits of non-volatility, form factor, density and reliability without the complex control concerns. This application note provides additional background on the operating characteristics of the BPK 72 and is intended to further ease the design effort required in the implementation of bubble memory systems.

## OVERVIEW

This application note provides an example of Bubble Memory system implementation using the BPK 72 and an Intel 8086 microprocessor. Before looking at this example, some explanation is necessary as to how this implementation was attained and how a user can take advantage of the principles involved.



As an introduction, the basic architecture of the BPK 72 is reviewed followed by an explanation of the operating characteristics of the BPK 72 kit as a whole and of the 7220 Bubble Memory Controller. Once the building blocks are in place, a detailed account of the implementation of a bubble memory kit is offered. The final section, which involves the actual implementation of the BPK 72 and an SDK-86, completes the application note.

## BUBBLE SYSTEM OVERVIEW

A block diagram of the Intel Magnetics 128K-byte system is shown in Figure 1. The support circuitry used with one 7110 magnetic bubble memory (MBM) in the BPK 72 kit consists of the following integrated circuit components: one 7250 Coil Predriver, two 7254 Quad VMOS Drive Transistor packs, one 7230 Current Pulse Generator, and one 7242 Formatter/Sense Amplifier. The 7220 Bubble Memory Controller (BMC) completes the basic system.

The 7250 and the two 7254s supply the drive currents for the in-plane rotating magnetic field (X and Y coils) that move the magnetic bubbles within the MBM. The 7230 supplies the current pulses that generate the magnetic bubbles and transfer the bubbles into and out of the storage loops of the MBM.

The 7242 accepts signals from the bubble detectors in the MBM during read operations, buffers the signals and performs data formatting tasks that include the transparent handling of bootloop information. During write operations, the 7242 enables the current pulses of the 7230 that cause the bubbles to be generated in the 7110 MBM. Automatic error detection and correction of the data can be performed by the 7242.

The 7220 provides the user interface, performs serial-to-parallel and parallel-to-serial data conversions, and generates all timing signals necessary for the proper operation of the MBM support circuitry.

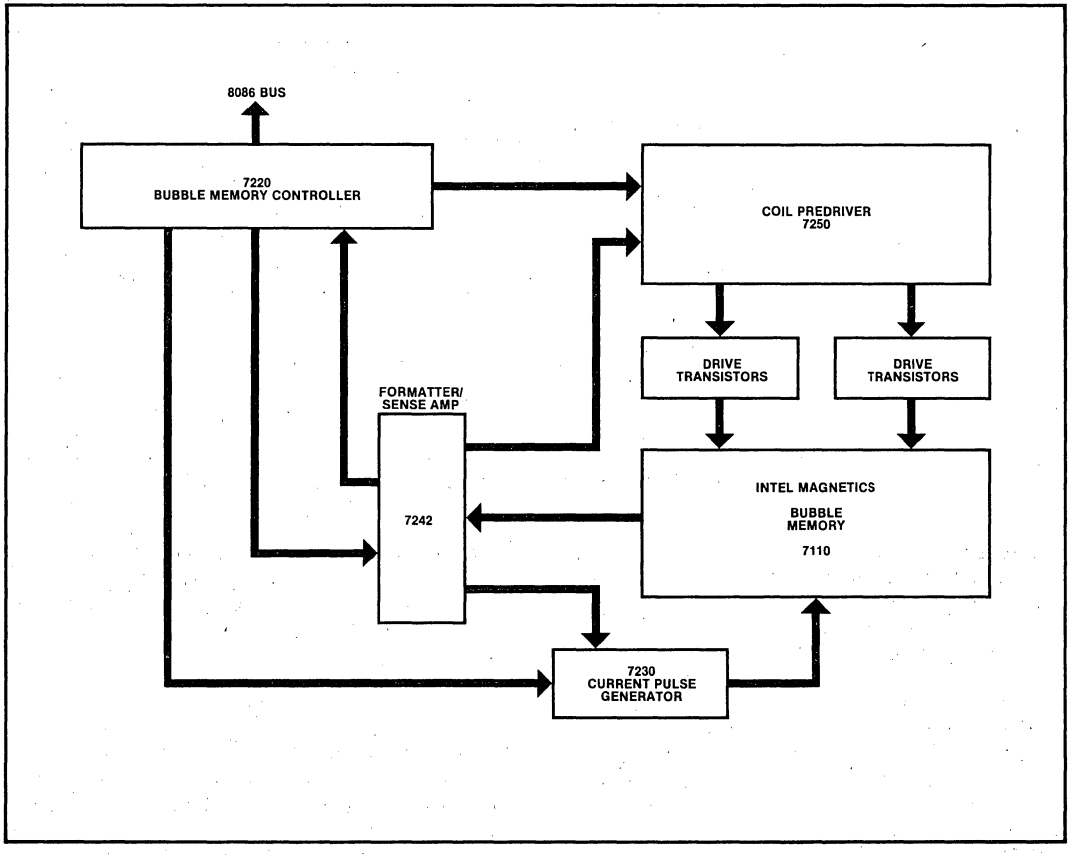



**Figure 1. Block Diagram of the 128K Byte Magnetic Bubble Memory System**

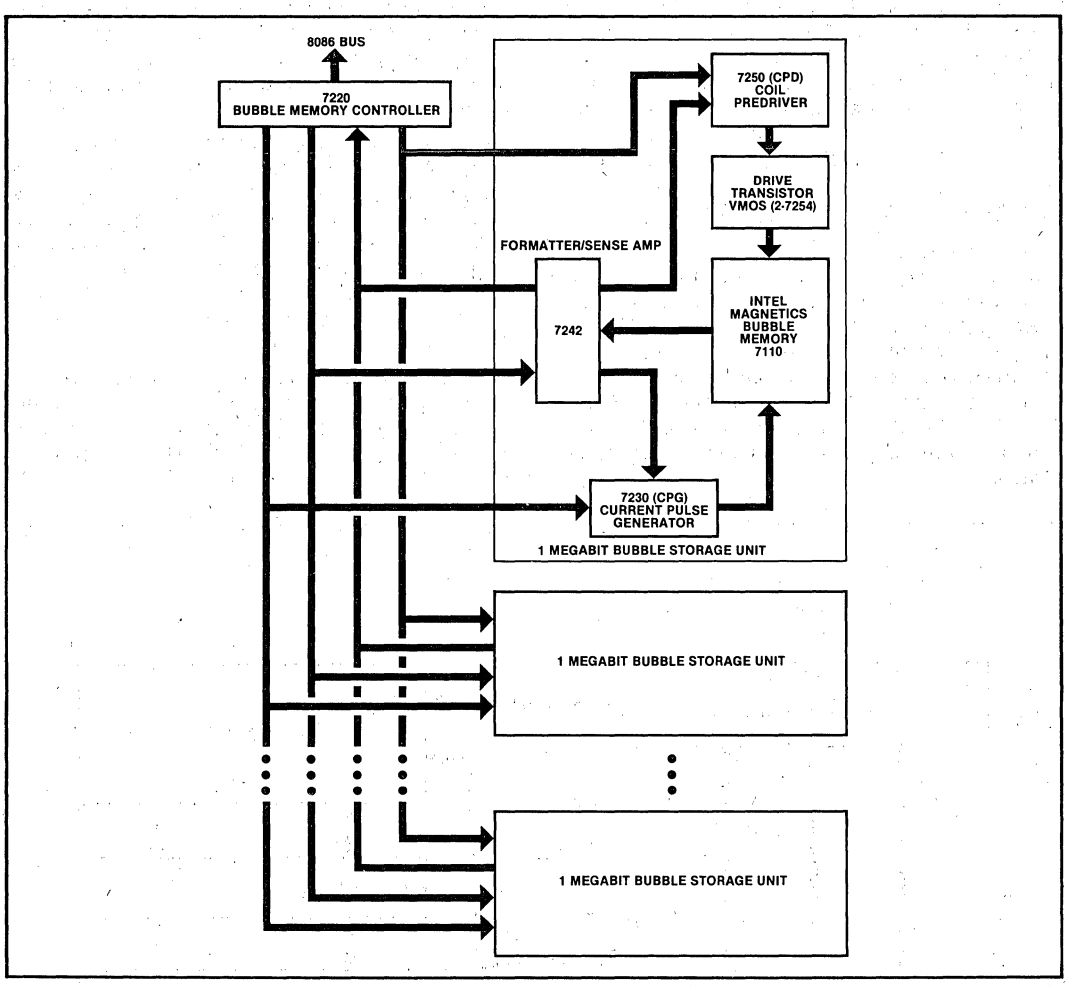



**Figure 2. Bubble Memory System Expansion up to One Megabyte**

Figure 2 shows how larger systems can be built from the basic components. A Bubble Storage Unit consists of one 128K-byte MBM and the five support chips shown. The components needed for one MBM cell are available as the BPK 70 kit. Larger systems can be constructed from the components supplied with one BPK 72 kit (which includes the 7220 controller) and one or more BPK 70 kits. For example, a one megabyte system can be assembled from one BPK 72 kit and seven BPK 70 kits. No additional TTL parts are required when building multibubble systems with up to eight MBMs.

One 7220 is capable of controlling up to eight Bubble Storage Units simultaneously. Larger systems can be configured with multiple 7220's and additional Bubble Storage Units.

## Functional Organization of the 7110 Bubble Memory

The Intel Magnetics 7110 Bubble Memory utilizes a "major track/minor loop" architecture. With this architecture, if a binary 1 is to be written, a "seed bubble," always present in the 7110, is split in two. One bubble remains at the generator as the

seed, and the other is propagated down the input (major) track. If a 0 is to be written, the seed bubble is not duplicated. The data generated is sent down the input track, in serial, until it is aligned with the "swap" gates at the minor loops of the device. The new data is then swapped into the minor loops in parallel at the same time the old data is swapped out to the major track.

To read data from the 7110, data is rotated in the minor loops until it is positioned at the "replicate" gates opposite the output track. On receipt of a replicate signal, the data in the minor loops is duplicated by splitting the bubbles. The original data remains in the minor loops, and the duplicate data is clocked down the output track where the detector elements of the bubble memory operate to transform the presence or absence of a bubble into small electrical signals that are converted into digital '1' and '0' signals in the 7242 FSA.

With the 7110, the process of reading data from the minor loops by simultaneously splitting all of the bubbles in a page is known as "block replicate." The advantage of the block replicate architecture is that the data currently stored in the minor loops is not compromised during a read operation; the data to be read never leaves the minor loops. This architecture can be contrasted with earlier architectures that required the data to leave the minor loops, be detected and then returned to the minor loops. In the event of a power failure, bubble systems not utilizing the block replicate architecture could suffer a loss of data during a read operation; the data being sensed would not be returned from the major loop to the minor loops.

With the 7110 MBM, there are 2048 positions for the data within a minor loop. To move the bubbles in the MBM, a magnetic field is induced and rotated in the plane of the 7110. As the field is rotated 360 degrees, every bubble is moved ahead one position, and all of the bubbles maintain the same position relative to one another. All of the bubbles in similar positions in the loops are referred to as a "page."

By way of illustration, suppose the bubble is made of five minor loops (a,b,c,d,e) capable of holding nine pages of data (Table 1). During four 360 degree "rotations" of the in-plane magnetic field, the nine pages of data shift four positions (1.1, 1.2, 1.3, 1.4).

**Table 1. 7110 Loop Operation**

| abcde | abcde | abcde | abcde |
|---|---|---|---|
| 00000 | 00011 | 00000 | 00000 |
| 00011 | 00000 | 00000 | 11111 |
| 00000 | 00000 | 11111 | 00000 |
| 00000 | 11111 | 00000 | 00000 |
| 11111 | 00000 | 00000 | 00000 |
| 00000 | 00000 | 00000 | 10110* |
| 00000 | 00000 | 10110* | 00000 |
| 00000 | 10110* | 00000 | 00011 |
| 10110* | 00000 | 00011 | 00000 |
| 1.1 | 1.2 | 1.3 | 1.4 |

* = page zero

The 7110 MBM actually contains 320 minor loops, of which 272 must be good. The additional 48 loops provide 15% redundancy. This redundancy factor allows some of the loops in the 7110 to be bad while maintaining a completely functional one megabit device. A map of the good and bad loops is placed on the label of the 7110 and is also

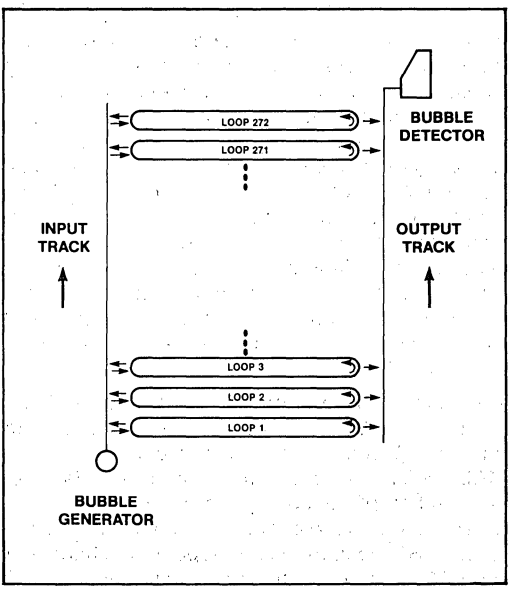



**Figure 3. Functional Organization of the 7110**

encoded and placed in the boot loop of the device as it is tested. This map, the bootloop, consists of forty bytes of data. Each good loop in the 7110 is represented by a one, each bad loop by a zero. When the system is initialized, the 7220 BMC reads the bootloop from the 7110 and decodes it. The bootloop is then automatically placed in the bootloop register of the 7242. The bootloop register serves as a working 'map' of the 7110 for read and write operations.

With the pages of data rotating around the minor loops, there must be a mechanism to orient the device and to assign a starting address to a page. The mechanism used to identify page zero involves the bootloop that resides on the 7110. Page zero (or address zero) is defined as the position of the 7110 after the bootloop has been read by the 7220 controller. Thus, each time the host CPU sends an "initialize" command, the bootloop is read by the 7220, and the 7110 is queued at page zero. From this point, any desired page in the bubble can be obtained by the controller.

## Data Flow Within the Bubble Memory System

To better understand the relationship between the 7110 MBM and its support circuitry, the data flow within the bubble system during a read operation is examined. During the read operation, bubbles from the storage loops are replicated onto an output track and then moved to a detector within the MBM. All movements within the MBM occur under the influence of a rotating magnetic field; the number of rotations and the rotation timing are under the control of the 7220 BMC. The detector outputs a differential voltage according to whether a bubble is present or absent in the detector at any given time. This voltage is fed to the detector input of the 7242 Formatter/Sense Amplifier (FSA).

The data path between the 7110 MBM and the 7242 FSA consists of two channels (channel A and channel B) connected to the two halves of the MBM. When data is written, the bit stream is divided with half of the data going to each side of the MBM. During a read operation, data from each half of the MBM goes to the corresponding channel of the FSA. In the FSA, the sense amplifier performs a sample-and-hold function on the detector input data, and produces a digital 0 or 1. The resulting data bit is then paired with the corresponding bit in the FSA bootloop register.

If an incoming data bit is found to be from a good loop (a corresponding "1" in the FSA bootloop register), it is stored in the FSA FIFO; otherwise, it is ignored. This process continues until both FSA FIFOs (channels A and B) are filled with 256 bits. Error detection and correction, if enabled by the user, is applied to each block of 256 bits at this point. If error correction is not enabled, 272 bits of data can be buffered in each FIFO.

As data leaves the 7242 FSA, the bit patterns buffered in each of the FSA FIFOs is interleaved and sent to the 7220 BMC in the form of a serial bit stream via a one-line bidirectional data bus (DIO line). In the 7220 BMC, the data undergoes a serial-to-parallel conversion and is assembled into bytes that are buffered in the 7220 FIFO. It is from this FIFO that the data is written onto the user interface.

## COMMUNICATING WITH THE 7220

The CPU views the 7220 BMC as two input/output ports on the bus. When the least-significant bit of the address line is active (A0 = 1), the command/status port is selected. When the least-significant bit of the address line is inactive (A0 = 0), the bidirectional data port is selected. In order to define the operations on these ports, it is necessary to understand something of the internal organization of the 7220 Bubble Memory Controller.

For simplicity, the user need only view the 7220 as containing a 40-byte FIFO and a collection of 8-bit registers. The FIFO is a buffer through which data passes on its way from the 7242 Formatter/Sense Amplifier (FSA) to the user, or from the user to the FSAs. The primary purpose of the FIFO is to reconcile differences in timing requirements between the user interface to the 7220 controller and the controller interface to the FSAs.

The six 8-bit registers internal to the 7220 are loaded by the user prior to any operation of the bubble system and contain information regarding the operating mode of the 7220. Loading the 7220 registers before any commands are sent is similar to passing parameters to a subroutine prior to invocation, hence, the registers are often referred to as "parametric registers."

Data transferred between the CPU and the 7220 FIFO and parametric registers takes place over an 8-bit data port. The choice as to whether the data is destined for the FIFO or the parametric registers, however, is made through the command/status port. In one case, the actual commands that cause some operation to take place, such as a read or write, consist of a 4-bit code sent by the CPU to select one of 16 possible commands. This 4-bit code occupies the low-order nibble (bits 0, 1, 2, and 3) of the command byte. The command byte must also have bit 4 set to indicate to the 7220 that a command is being sent. In the

second case, another 4-bit code on the command port (bits 0, 1, 2, and 3) is used to select either one of the parametric registers or the 7220 FIFO. As shown in Table 2, if bit 4 of the command byte is set to zero, the value of the low-order nibble is taken to be a pointer value that specifies a parametric register or the 7220 FIFO. This pointer is referred to as the "Register Address Counter" (RAC).

**Table 2. 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 |

RAC values that may be sent out on the command port and the corresponding register names are illustrated in Table 3. The RAC points to, or selects, six unique registers and the 7220 FIFO. Once a RAC value is sent by the CPU to the 7220 via the command port, the next read or write operation to the data port transmits data to or receives data from the register addressed. Notice that the six registers have values that are in ascending order starting at 0AH and that the FIFO has a value of 0.

The reason for this ordering is due to the auto-incrementing feature of the RAC; once the first register is selected, each subsequent byte of data on the data port causes the RAC to be automatically incremented and to point to the next register in the sequence. Once the most-significant byte of the Address Register has been loaded, the RAC value automatically rolls over from 0FH to 0 and points to the 7220 FIFO. The system is now in position to transfer data to or from the FIFO without the user code explicitly pointing to the FIFO.

**Table 3. 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 |

Once the FIFO has been selected, the RAC stops incrementing and continues to point to the FIFO until changed by the user software. This sequence minimizes the number of instructions necessary for a given transaction and aids in establishing a protocol to ensure that all of the necessary information is sent to the controller. The user, however, is not bound to follow this automatic sequence. Each parametric register may be selected and loaded in any order; specific registers may be updated where needed, but in each case, the host software must explicitly name the register to be loaded. Until a user is familiar with the bubble system, it is recommended that the auto-incrementing feature be used.

It is important to remember that once a command has been given to the 7220 BMC, the parametric registers must not be updated until the Status byte indicates that the operation is complete. The parametric registers are, in effect, working registers for the controller during the execution of a command. For example, during a Read or Write operation, the Block Length Register, which contains the terminal page count for the operation, is decremented by the 7220. Similarly, the Starting Address Register, which initially contains the starting page for an operation, is incremented by the controller as each page is transferred. Attempting to modify these registers during the operation of a command causes the block count and address to be incorrect.

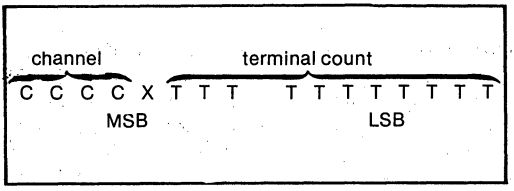
## Addressing the Bubble Memory System

One of the interesting aspects of the Intel Bubble Memory System is its inherent addressing flexibility. The user may treat a 7220 BMC with eight

bubbles as a collection of 16K pages of 64 bytes each (addressing each bubble in turn) or as collection of 2K pages of 512 bytes each (addressing eight bubbles in parallel). Of course, there are a variety of configurations in between these two extremes, each dictated by the user's need for speed, power consumption, address space, and cost. Control over the configuration is achieved at run time via two of the parametric registers: the Block Length Register and the Starting Address Register.

The Block Length Register (BLR) is a 16-bit value divided into two fields: the "terminal count" field and the "channel" field. The bit configuration for the BLR is as follows:

**Table 4. Block Length Register**




The "terminal count" field ranges over eleven bits and defines the total number of pages requested for a read or write operation. With eleven bits in the field, a user may request from one to 2048 pages be transferred (eleven bits of zero indicate a 2048-page transfer). The width of the page is effectively defined in the "channel" field. This field specifies the number of FSA channels that are to be addressed. Recalling that each 7242 FSA has two channels to communicate with one 7110 bubble memory, the legal combinations in this field address one channel (one half of a 7110), two, four, eight, or 16 channels. These combinations translate into page sizes of 32, 64, 128, 256, or 512 bytes, respectively. (The one-channel mode of operation is usually reserved for diagnostic purposes, and examples of its use will be illustrated later.)

Table 5 shows the relationship between the "channel" field and the number of FSA channels selected. Notice that the channel field bits are encoded. A value of "0001" binary selects two FSA channels: 0 and 1.

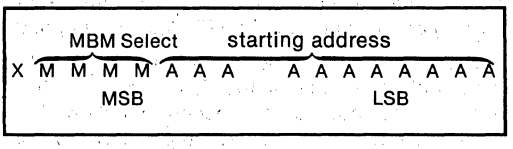
**Table 5. FSA Channel Select**

| | Channel field (BLR MSB bits 7, 6, 5,4) | | | | |
|---|---|---|---|---|---|
| | 0000 | 0001 | 0010 | 0100 | 1000 |
| Number of channels selected: | 0 | 0,1 | 0,1,2,3 | 0 to 7 | 0 to F |

Thus, a BLR value of "0001" in the high-order four bits selects one bubble through channels 0 and 1 Similarly, a BLR value of "0010" selects two bubbles in parallel with a page size of 128 bytes. This, however, is not the complete story. For example, a value of "0100" in the BLR selects four bubbles in parallel through channels 0 to 7. Suppose, that there are eight bubbles in the system and that the user desires to arrange the eight bubbles as two sets of four. The mechanism to communicate through channels 0 to 7 and channels 8 to F resides with the Address Register (AR).

The Address Register contains a 16-bit value divided into two fields: a "starting address" field of eleven bits and a "magnetic bubble memory (MBM) select" field of four bits.

**Table 6. Starting Address Register**




The eleven bits in the starting address field of the AR are set by the user to indicate to the 7220 BMC on which page of a bubble's 2048 pages the transfer is to start. For example, if a read operation is to start at page 1125 and is to continue for 16 pages, the starting address field contains 1125, and a value of 16 is placed in the terminal count field of the BLR. After each page is transferred, the starting address field is incremented and the terminal count is decremented by the controller.

Continuing with the example of two banks of four bubbles, notice in Table 7 that the MBM select field is needed to switch between the two banks. A value of "0000" in bits 3, 4, 5, and 6 of the high-order byte of the address register selects bank 0 or FSA channels 0 through 7; a value of "0001" selects bank 1 or FSA channels 8 through F. Each bank contains 2048 pages of 256 bytes.

To operate eight bubbles serially, a user needs only to specify a value of "0001" once in the channel field of the BLR and to begin with a value of "0000" in the MBM select field. As page 2048 is written in the first bubble, the AR, managed by the 7220 controller, rolls over to 0 and updates the MBM select field with no additional bit manipulation. In this case, the bubble system appears as 16K pages of 64 bytes each. Power consumption is one-eighth of that consumed by operating eight bubbles in parallel. However, the data rate is limited to the data rate of one bubble.

**Table 7. FSA Channel Select/MBM Select**

| MBM SELECT AR 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 | | | | |

## The Enable Register

The Enable register is the parametric register that defines the various modes of operation of the 7220 controller. The data transfer mode (polled, interrupt driven, or DMA operation) is selected by setting the appropriate bit in this register. Likewise, the type of error correction to be applied to the data is selected, based on the bits selected in this register.

While the function of each of the enable register fields is described in the BPK 72 manual, some of the finer points and implications are detailed here.

Note that it is possible to completely change the operating characteristics of the bubble system through software control. A system can go from the DMA mode with error correction enabled to a system operating in polled I/O with no error correction enabled by altering the value of the Enable register. Though most implementations will not take advantage of this degree of flexibility, there are cases where the Enable register is modified during system operation. For example, the normal interrupt and MFBTR bits can be modified between operations to change interrupt and read data rates, respectively. (If the error correction mode is changed, the CPU must issue an Initialize command to the 7220 controller).

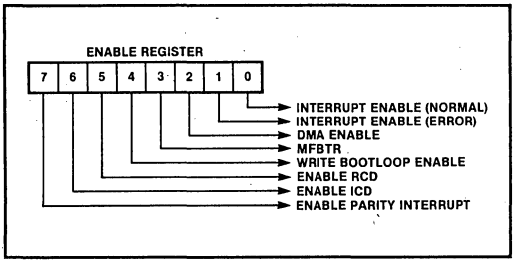



**Figure 4. Enable Register Definition**

The interrupt capabilties of the 7220 are reflected in the NORMAL, PARITY and ERROR INTERRUPT bits of the ENABLE register byte. The 7220 controller is capable of issuing interrupts to a CPU at the normal completion of an operation, if a parity error is encountered between the 7220 controller and the CPU, or if a data transfer error is found by the 7242 FSA. Any (or all) of these conditions are selected via the Enable register byte, and any resultant interrupts are sent to the CPU via a single INT line. At this point, the software must examine the status register to determine the cause of the interrupt. (An additional interrupt, the FIFO half-full interrupt, is issued on the DRQ pin and is not controlled by the Enable byte).

One of the more difficult aspects of the ENABLE register byte to understand is the operation of the ERROR INTERRUPT bit (bit 2). This bit normally is not used alone, but in conjunction with the ENABLE RCD and ENABLE ICD bits of this register. These three bits form combinations that gate selected 7242 error conditions to the CPU. For example, if, while operating under error correction, a user does not wish to be bothered by an interrupt that indicates an error has been corrected automatically by the system, a specific pattern of these three bits would be selected (100 or 010 from Table 8). If the user wishes to be notified of all errors, another pattern would be selected (011 or 101).

**Table 8. Error Correction Combinations**

| Enable ICD | Enable RCD | Interrupt Enable (ERROR) | Interrupt Action |
|---|---|---|---|
| 0 | 0 | 0 | No interrupts due to errors |
| 0 | 0 | 1 | Interrupt on TE only |
| 0 | 1 | 0 | Interrupt on UCE or TE |
| 0 | 1 | 1 | Interrupt on UCE, CE or TE |
| 1 | 0 | 0 | Interrupt on UCE or TE |
| 1 | 0 | 1 | Interrupt on UCE, CE or TE |
| 1 | 1 | 0 | Not used |
| 1 | 1 | 1 | Not used |

The purpose of the ERROR INTERRUPT bit is not to enable or disable error interrupts, but rather to aid in selecting the type of error interrupt received by the CPU. If any type of error correction is selected, interrupts are enabled automatically.

The ENABLE RCD (read corrected data) bit causes the error correction algorithm to be applied to the data being transferred from the 7110 MBM in an almost transparent manner. The RCD bit allows the 7220 controller to send its own commands to the 7242 FSA. These commands cause the FSA to automatically correct and transfer to the controller, any data that is found to be in error and that is considered correctable.

With only the RCD bit on, no interrupt is generated if a correctable error is found. However, the user is informed that a correctable error was encountered and corrected during the data transfer via the 7220 status byte at the end of the operation. Uncorrectable and timing errors cause an interrupt to which the CPU must respond. With both the RCD bit and ERROR INTERRUPT bit on, the CPU is notified via an interrupt whenever a correctable, uncorrectable or timing error is encountered.

The RCD mode of operation is suitable for transfers where a GO/NO GO termination is sufficient. For example, when loading executable code from the bubble to RAM, it is necessary to know that the transfer was good (with errors corrected) or aborted due to an uncorrectable error.

A retry of an uncorrectable page of data is accomplished by sending another Read command without modifying the parametric registers. It may be the case that the errors encountered were soft (read) errors that may not be present on a retry. Thus, what may have been detected as an uncorrectable error, may become a correctable error (or simply vanish ) on a subsequent read of the offending page. In this case, the error correction ability of the system corrects the errors automatically without additional user intervention.

The advantage of the RCD mode of operation is that error correction can be applied transparently to the CPU except for uncorrectable conditions. The disadvantage is that a page of uncorrectable data is passed to the controller before the interrupt is sent. The software must have the ability to clear the 7220 FIFO prior to rereading the offending page from the bubble.

If a given page continues to show up as having a correctable error after a number of retrys, it is up to the user's protocol to determine the action to be taken. One protocol suitable for handling errors involves "scrubbing" the data. Suppose a page appears with an error and, on retry, the error is still present. If the error is correctable, the data should be corrected and written back to the bubble and then read back into RAM. The probability of encountering an uncorrectable error after the first retry is 1 in $10^{16}$. Data scrubbing after one retry maintains this level of reliability.

The ENABLE ICD (internally correct data) bit also enables the error correction capability of the bubble system, but allows a slightly different interaction between the 7220 controller and the 7242 FSA than defined for the RCD mode. Error interrupt conditions are the same as defined for RCD operation. With the ICD bit on, correctable errors are handled automatically, but the operation halts for uncorrectable or timing errors. With both the ICD and ERROR INTERRUPT bits on, the operation halts for correctable, uncorrectable or timing errors. The ICD mode differs from the RCD mode in that when an operation halts due to an error, the offending page is held in the 7242 FSA and is not automatically transferred to the 7220 FIFO. Though the difference is subtle, the ICD mode of operation allows more flexibility in error logging and recovery. With data held in the 7242, the number of the bad page can be read for logging purposes, and the data can be recycled through the error correction network or reread from the bubble repeatedly. When the CPU is interrupted due to an error in the ICD mode, the user must look at the 7220 status byte to determine the type of error encountered. If the error is correctable, the user's software sends a Read Corrected Data command (0CH) to the controller. This command causes the controller to issue it's own commands to the 7242 to correct the error and to transfer the data to the 7220 FIFO. (Recall this action is done automatically when the RCD mode is selected; uncorrectable errors can be handled as described above).

As an example of how the ICD mode can be utilized, suppose that during a data transfer in the RCD mode, a correctable error consistently occurs. The

error, of course, is automatically handled by the 7242, and the only indication that an error had been corrected is through the status byte at the end of the transfer. There is no information as to how many or in what page the error or errors appear. One way to diagnose the problem is to reread the entire data block in the ICD mode with the ERROR INTERRUPT bit on. The transfer stops at the appearance of any error, and the data remains in the 7242. The page number of the error can be found by reading the Address Register since this register is incremented automatically after each page is read if no error is detected.

The user should then issue an RCD command to the 7220 to allow the page to be corrected and transferred to the 7220. Once the transfer is complete, the enable register again is changed to disable all error correction, and the 7220 is reinitialized. The entire block is read again and compared with the corrected version. (Error correction bits are appended to the data and can be ignored.) If a bad loop is suspected, the bad loop location could be calculated and the bootloop modified.

It is unlikely that repeated correctable errors are sufficient motivation to modify the bootloop. Repeated uncorrectable errors, however, at the same location, might be sufficient reason. Note that modifying the bootloop is an extreme measure and should only be performed as a last resort and only if justified by test data.

## The Status Register

The 7220's 8-bit Status register is accessed by reading the Command port (A0 = 1). This register provides information regarding error conditions, the termination of commands, and the readiness of the controller to transfer data or accept new commands.

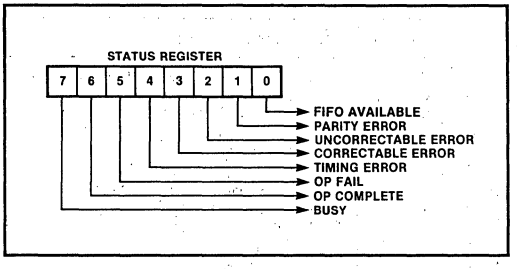



**Figure 5. Status Register Definition**

Values for the Uncorrectable Error and Correctable Error fields are generated when error correction is utilized as previously defined. The PARITY ERROR bit is set when a parity error is encountered on data sent to the controller on the $D_0$-$D_7$ lines. The TIMING ERROR bit is set for a number of conditions. The most frequent cause of a timing error is when the CPU fails to keep up with the rate at which the controller is filling or emptying the FIFO (an overflow or underflow condition). With one bubble in the system and the MFBTR bit of the Enable byte set to one, the controller moves data to or from the FIFO at a rate of about one byte every 80 microseconds. With eight bubbles operating in parallel, the rate is about one byte every 10 microseconds. (With the MFBTR bit set to 0, the data rate on a one page transfer or the last page of a multipage transfer is four times these rates.) Once a Read or Write command is issued, if the CPU cannot meet these transfer requirements, a timing error results.

Another way in which a timing error occurs is when the proper number of bits is not set in the bootloop register of the 7242 FSA. The 7242 must have 272 loops active to operate properly (270 with error correction enabled). If a mistake is made either when the bootloop of the 7110 is written or if the bootloop register is loaded incorrectly from RAM by the user, a timing error results. A timing error also occurs if the Write Bootloop command is issued to the 7220 controller and the WRITE BOOTLOOP ENABLE bit of the Enable byte is not on. Finally, a timing error is generated if the bootloop synch code is not found when a Read Bootloop or Initialize command is issued.

The OP FAIL and OP COMPLETE bits of the status register simply indicate the state of an operation after a command is executed. If an operation fails (OP FAIL = 1), the cause can be determined by looking at the other error bits of the status byte. When an operation (command) terminates successfully, the OP COMPLETE bit is set, and the status register shows a 40H.

The FIFO AVAILABLE bit of the status byte is more complex than the other bits since its meaning can change depending on the type of operation being performed as outlined below.

From an operational point of view, the FIFO AVAILABLE bit acts as a gate for the FIFO handling software. During a write operation, if the FIFO bit is set (1), there is room for more data; if the FIFO bit is clear (0), the FIFO is full. During a read operation, if the FIFO bit is set, data has been placed in the FIFO by the controller; if it is clear, the FIFO is empty.

**Table 9. FIFO Available Bit Semantics**

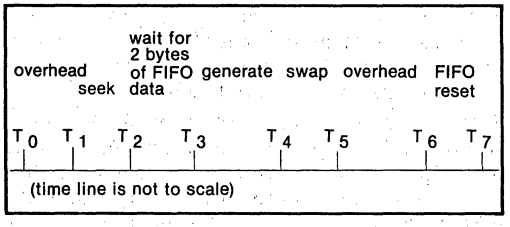
| FIFO AVAIL BIT | BUSY = 1 & writing | BUSY = 1 & reading | BUSY = 0 & reading |
|---|---|---|---|
| 1 | room for data | data avail. | data avail. |
| 0 | no room for data | no data | no data |

Note that it is possible to complete an operation with data still remaining in the FIFO (indicated by a 41H status value). This condition is quite legal; it is up to the software to remove the data or to issue a FIFO RESET command.

The BUSY bit indicates when the controller is in the process of executing a command. When a command is sent, the BUSY bit goes active within a few microseconds after the command is received and remains active until the operation either completes or fails. It is important to note that the BUSY bit remains active until all other bits in the status byte have been set. Thus it is possible to see logically-exclusive conditions such as BUSY and OP COMPLETE at the same time. The key to interpreting the status byte is to consider the status byte valid only after the BUSY bit returns to an inactive level. The single exception to this rule is the FIFO AVAILABLE bit.

The action of the controller during a write operation is one of the more complex sequences and serves as a good illustration of the behavior of the BUSY and FIFO AVAILABLE bits. Suppose a Write command is sent to transfer an arbitrary number of pages. Table 10 shows the activity of the controller at various steps in the sequence.

**Table 10. Stages of a Write Command**




Before the Write command is sent, the FIFO is in a general-purpose mode and remains in this mode until $T_2$. When the command is sent at $T_0$, the BUSY bit is low and, in fact, the BUSY bit must be low in order for the controller to accept a new command (except Abort). Sometime between $T_0$ and $T_1$, the BUSY bit goes high. Thus, between $T_1$ and $T_2$, the status byte will be 80H.

At $T_2$, the FIFO is internally placed in the "write mode," and FIFO AVAILABLE changes meaning from "FIFO has data" to "FIFO has room". For proper operation, the FIFO must be empty prior to issuing the WRITE command. This condition can be guaranteed by using the FIFO Reset command. Assuming the FIFO is empty, at $T_2$ the status byte changes from 80H to 81H. The status byte remains at 81H until $T_6$ (unless the CPU is able to fill the FIFO in which case, the FIFO AVAILABLE bit toggles between 0 and 1).

At $T_7$ (the completion of the command), the status byte should be 40H if the CPU did not load data between $T_6$ and $T_7$. If data was loaded during this interval, the status value is 41H.

Notice that if the FIFO contains data when the Write command is sent, the CPU can, by mistake, overflow the FIFO during the "seek" portion of the command. This condition results from the FIFO AVAILABLE bit being a "1" due to data present in the FIFO, not because there is room in the FIFO. While the following diagnostic routines take advantage of the "preloading" ability of the FIFO, the examples of operational software at the end of this application note do not preload the FIFO.

## 7220 Commands

The 7220 command set consists of 16 commands identified by a 4-bit command code. The function of most of the commands is obvious from the command name (e.g., Initialize, Abort, Read, Write). These commands are adequately described in the BPK 72 manual. There are, however, some commands and protocols that merit additional discussion (specific examples are covered later in this document).

**Table 11. 7220 Commands**

| 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 |

In general, all commands sent to the 7220 controller must be preceded by the setting of the parametric registers. While there are some exceptions as with the Abort command, it is usually necessary to supply operating information to the controller via the parametric registers prior to issuing any command. Since many initial problems stem from failing to load the registers prior to issuing commands, the user software should never assume that the regsiters contain valid data.

After the bubble system has been powered up, the 7220 controller inhibits (or ignores) all commands except an Initialize or Abort command. One of these commands must be sent prior to issuing any other command. Normally, the first command issued after loading the parametric registers is the Initialize command. This complex command reads and decodes the bootloop information from each bubble in the system and places this information in the bootloop register of the corresponding 7242 FSA. Pointers internal to the 7220 automatically are prepared for normal operation. As described later, the combination of the Abort, MBM Purge and Write Bootloop Register commands is functionally similar to the Initialize command. (The only time the MBM Purge command is used is in conjunction with the Abort command).

Once the system has been initialized, the remainder of the command set can be selected. Assuming, for example, that a Read command is to be executed, the user selects the page number and length of the transfer via the parametric regisiters and then issues the Read command. If the system uses the polled mode, the CPU reads the status register and waits for the BUSY bit to go active and then for the FIFO READY bit to indicate that data is being sent to the FIFO. Data can be taken from the FIFO until the FIFO READY bit goes inactive.

If the page selected for the read operation is not in position to be read (i.e., the page is not at the replicate gates), additional time is required to execute the Read command as the proper page is rotated into position. In systems where faster response is desired, the Read Seek command can be used to place the page into position in order to free the CPU to perform other tasks. Once the page is in position, approximately eight milliseconds are required before the data is available to the CPU. This latency only occurs on the first page of a multipage transfer. Similarly, when a page is not in a position to be written, Write Seek can be used to position the page at the swap gates.

If there is any doubt regarding the state of the FIFO prior to a read or write operation, the user should issue a FIFO Reset command in order to clear the 7220's FIFO counter before initiating the data transfer. If a prior transfer is stopped with data remaining in the FIFO or if the FIFO is partially filled, the 7220's internal FIFO counter is not zero, and there is a danger that the subsequent transfer count may be incorrect. If the FIFO is reset properly, execution of a FIFO Reset command is redundant.

Although the 7220 FIFO may be treated as a 40-byte RAM buffer, the temptation to "pre-load" the FIFO with 40 bytes of data and then to issue a Write command should be avoided due to the danger of overflowing the FIFO. Prior to issuing a Write command, a FIFO Reset command should be sent, and the parametric registers should be loaded. Following the Write command, the CPU should monitor the status byte and wait for the BUSY and FIFO AVAILABLE bits to go active. When this status condition occurs, the user software should then send the proper number of bytes to the 7220. The FIFO AVAILABLE bit of the status byte should be polled prior to sending each byte.

An exception to not preloading the FIFO is when a Write Bootloop, Write Bootloop Register, or Write Bootloop Register Masked command is used. Prior to issuing any of these commands, a FIFO Reset command must be sent before preloading the bootloop data into the FIFO. When one of the bootloop-related commands is issued, the 7220 controller immediately begins taking data from the FIFO. If the FIFO is not preloaded, incorrect data may be transferred. The operation of the normal Write command differs from the bootloop-related commands in that, after a Write command is issued, the 7220 waits for at least two bytes to be present in the FIFO before beginning to transfer data to the bubble.

If the FSA encounters an error condition during a read or write operation, the status of the FSA is reflected in the 7220 status byte. If the user system decodes the error and decides to continue, the error flags in the 7220 controller and FSA first must be cleared. To clear the status bytes, the software can issue an Initialize command. However, this command resets all of the current operating parameters in the 7220 controller. To continue processing without resetting the system, the software can use the Software Reset command. This command resets any error flags and clears the FIFO, but does not affect the parametric register fields that define the system configuration (e.g., number of FSA channels selected).

## INSTALLING THE BPK 72 BUBBLE MEMORY KIT

This section examines the individual components of the Bubble Memory System and how each component can be analyzed. All elements of the bubble system need not be working before any meaningful diagnostics can be effected. In general, a user first establishes communication between the host CPU and the 7220 controller. Next, communication with the 7242 formatter/sense-amplifier is verified via the 7220 controller. Finally, the operation of the 7110 Bubble Memory is checked. The software that exercises each of these phases of implementation should be small, well-defined device drivers that can be controlled through a system monitor.

The procedures that follow are applicable to most startup problems. The procedures are organized in chronological fashion and address each step of the installation process as it would normally occur. Software drivers in 8086 assembly language are provided to illustrate the basic functions supported by the device drivers.

### Powering Up for the First Time

With power removed from the IMB-72 board, insert all of the supporting integrated circuits with the exception of the 7110 Bubble Memory Module. Insert the "dummy module" included in the BPK 72 kit in place of the 7110. The dummy module is electrically equivalent to the 7110 module and allows the circuits of the BPK 72 kit to be tested without the possibility of damaging the bubble. With both the +5V and +12V power supplies turned off, insert the IMB 72 with the dummy module into the edge connector. As power is applied to the system, monitor the RESET.OUT/pin of the 7220 controller and verify that the signal goes from low to high after power is applied. The low-to-high transition indicates that the power-up sequence has been completed successfully.

### Communicating With the 7220 Bubble Memory Controller

The first step in communicating with the 7220 is to write initial values to the parametric registers using the code sequence in Table 15. When the registers have been set, the code shown in Table 12 can be used to examine the 7220 status byte.

The status value returned in Table 12 should be 40H. The user should not continue until the proper status value can be obtained repeatedly after performing the power-up sequence. Reading back the correct status indicates that the host CPU and the 7220 are communicating and that the power-up sequence is being performed by the 7220.

**Table 12. Reading 7220 Controller Status**

```
RDSTAT:
  ; THIS PROGRAM READS THE 7220
  ; STATUS BYTE
  ; TO READ STATUS, THE HOST CPU MUST
  ; READ FROM THE 7220 WITH A0 = 1.
    IN      AL, 49H       ; COMMANDS/STATUS
                          ; PORT ADDRESS OF
                          ; 7220
    MOV     STATUS, AL    ; MOVE AL REGISTER
                          ; TO STATUS
    RET
```

Once the power-up sequence is complete and the 7220 status register has been read, the 7220 FIFO can be accessed. The software drivers that write and read the FIFO are shown in Tables 13 and 14. Notice that these code sequences do not send commmands to the 7220; only data is transferred to and from the controller. The purpose here is to test the bus interface and timing between the CPU and the 7220 controller. In this case, the 7220 FIFO is used as a general purpose RAM. Any data can be written to the FIFO, but it is best to use an easily indentifiable sequence (e.g., an incrementing pattern) for easy recognition.

**Table 13. Writing the 7220 FIFO**

```
WTFIFO:
  ; THIS PROGRAM WRITES 40 BYTES FOR
  ; MEMORY TO THE 7220 FIFO.
  ; DATA IS ASSUMED TO BE ATBUFADR.
    MOVE    SI,BUFADR     ; LOAD BUFFER
                          ; POINTER
    MOV     CX,40         ; LOAD COUNT
WRT1:
    LODSB                 ; PUT BYTE AT SI
                          ; INTO AL, AUTO INCR
                          ; SI
    OUT     48H, AL       ; OUTPUT BYTE TO
                          ; DATA PORT
    LOOP    WRT1          ; DECREMENT COUNT,
                          ; LOOP IF NOT 0
    RET
```

Once forty bytes have been written to the FIFO, the 7220 status byte should be read. The status value should be "41H" (indicating that data is in the FIFO). Other status values such as "parity error" can be ignored. While status values give some indication of the CPU-7220 interaction, the integrity of the data is more important here. If the data read back is not the same as the data sent, a fundamental timing and/or interface problem between the CPU and the 7220 is indicated.

To verify that data is being transmitted to the 7220, the code sequence shown in Table 14 can be used to read back the FIFO data into user RAM space for direct comparison with the original pattern.

**Table 14. Reading the 7220 FIFO**

```
RDFIFO:
  ; THE PROGRAM READS 40 BYTES FROM
  ; THE 7220 FIFO INTO MEMORY.
    MOV    DI, BUFADR  ; LOAD BUFFER AD-
                       ; DRESS INTO DI
    MOV    CX,40       ; LOAD COUNT INTO
                       ; CX
RD1:
    IN     AL,48H      ; INPUT FROM DATA
                       ; PORT
    STOSB              ; STORE AL AT ADDR
                       ; IN DI, AUTO INCR. DI
    LOOP   RD1         ; DECREMENT COUNT
                       ; IN CX, LOOP IF NOT 0
    RET
```

After reading the FIFO, the status byte should be read (a value of "40H" or "42H," indicating that the FIFO has no data, should be obtained). The user should not proceed until the FIFO can be written and read correctly and until the FIFO status indicates the amount of data in the FIFO (not empty or empty). These steps verify that the CPU can communicate with the 7220. Note that no data has been transferred to or from the 7242 Formatter/Sense Amplifier or the 7110 bubble device (or dummy module).

## Communicating With the 7242 Formatter/Sense Amplifier

The next step in verifying the BPK 72 is to ensure that the 7220 is driving the 7242 Formatter/Sense Amplifier properly by first setting up the 7220 for interaction with the 7242 and then sending commands to the 7220 to exercise the 7242 functions that can be verified easily.

Under normal operating conditions an Initialize command is the second command sent to the system. However, the Initialize command assumes that the 7110 Bubble Memory is installed and attempts to read bootloop information. Since the dummy module is installed at this time, timing errors result from the attempted Initialize command. Although no harm results from using the Initialize command, an Abort command followed by an MBM-Purge command can be used in place of the Initialize command to eliminate timing errors. The Abort command is sent by executing the code sequence at label "CMND9" in Table 16. When Abort command execution is complete, the user should read the status byte and check for an op-complete indication (40H).

**Table 15. Write Register Sequence for Two FSA Channels**

```
WTREG2:;                        WRITE REGISTERS
; 2 FSA CHANNELS SELECTED.
; THIS IS USED FOR DEBUG TO WRITE/READ THE
; BOOTLOOP REGISTERS AND CHECK FOR MISSING SEEDS, ETC.
; THE FOLLOWING VALUES INTO THE 7220 REGISTERS
;       B = 01H     : 1 PAGE TRANSFER
;       C = 10H     : SELECT 2 CHANNELS (WHOLE BUBBLE)
;       D = 08H     : STANDARD TRANSFER RATE
;       E = 00H     : PAGE 0
;       F = 00H     : FIRST BUBBLE

    MOV     AL, 0BH         ; SELECT B REGISTER
    OUT     49H, AL
    MOV     AL, 01H         ; ONE PAGE TRANSFERS
    OUT     48H, AL
    MOV     AL, 10H         ; WHOLE BUBBLE (2 FSA CHANNELS)
    OUT     48H, AL
    MOV     AL, 08H         ; LOW FREQ
    OUT     48H, AL
    MOV     AL, 00H         ; START ADDRESS = 0000H
    OUT     48H, AL
    MOV     AL, 00H         ; FIRST BUBBLE
    OUT     48H, AL
    RET
```

Once the op-complete status is received, the MBM-Purge command is issued by executing the routine labeled "CMNDE" in Table 16. This command, as described in the BPK 72 manual, clears all of the controller registers, counters and address RAM (except the block length register), the NFC bits, the FSA present counter and the high-order four bits of the address register. After the command is complete, the user again should receive an operation complete indication on reading the status byte.

After the Abort and MBM-Purge commands are executed and is status verified, additional commands may be sent to the 7220 BMC. Since the purpose of this section is to verify the interaction of the 7242 and 7220, manually loading and reading the 7242 bootloop registers can be used for the verification. Two additional commands are required to load and read the bootloop registers: the Write Bootloop Register command and the Read Bootloop Register command. These commands transfer data between the 7242 bootloop registers and the 7220 FIFO. Since the ability to transfer data between user RAM and the 7220

**Table 16. 7220 Controller Commands**

```
CMNDS:          ; 7220 COMMANDS
    ; THESE 16 ROUTINES EACH SEND A SINGLE COMMAND TO THE 7220.
    ; FOR EXAMPLE, THE "INITIALIZE COMMAND" WILL WRITE 11H
    ; TO THE 7220 WITH A0 = 1. THESE ARE THE 7220 COMMANDS LISTED
    ; IN THE BPK-72 USERS MANUAL.
CMND0:
    MOV         AL, 10H         ; WRITE BOOTLOOP REGISTER MASKED COMMAND
    OUT         49H, AL
    RET
CMND1:
    MOV         AL, 11H         ; INITIALIZE COMMAND
    OUT         49H, AL
    RET
CMND2:
    MOV         AL, 12H         ; READ COMMAND
    OUT         49H, AL
    RET
CMND3:
    MOV         AL, 13H         ; WRITE COMMAND
    OUT         49H, AL
    RET
CMND4:
    MOV         AL, 14H         ; READ SEEK COMMAND.
    OUT         49H, AL
    RET
CMND5:
    MOV         AL, 15H         ; READ BOOTLOOP REGISTER COMMAND
    OUT         49H, AL
    RET
CMND6:
    MOV         AL, 16H         ; WRITE BOOTLOOP REGISTER COMMAND
    OUT         49H, AL
    RET
CMND7:
    MOV         AL, 17H         ; WRITE BOOTLOOP COMMAND
    OUT         49H, AL
    RET
CMND8:
    MOV         AL, 18H         ; READ FSA STATUS COMMAND
    OUT         49H, AL
    RET
CMND9:
    MOV         AL, 19H         ; ABORT COMMAND
    OUT         49H, AL
    RET
CMNDA:
    MOV         AL, 1AH         ; WRITE SEEK COMMAND.
    OUT         49H, AL
    RET
CMNDB:
```

**Table 16. 7220 Controller Commands (cont.)**

```
        MOV     AL, 1BH         ; READ BOOTLOOP COMMAND
        OUT     49H, AL
        RET
CMNDC:
        MOV     AL, 1CH         ; READ CORRECTED DATA COMMAND
        OUT     49H, AL
        RET
CMNDD:
        MOV     AL, 1DH         ; FIFO RESET COMMAND
        OUT     49H, AL
        RET
CMNDE:
        MOV     AL, 1EH         ; MBM PURGE COMMAND
        OUT     49H, AL
        RET
CMNDF:
        MOV     AL, 1FH         ; SOFTWARE RESET COMMAND
        OUT     49H, AL
        RET
```

FIFO has been verified previously, these two additional commands verify the system's ability to transfer between user RAM and the 7242 FSA.

The 7220 parametric registers must be loaded prior to sending the Write Bootloop Register command. The sequence of operations is important; loading the parametric registers destroys the first byte of data in the 7220 FIFO. If valid bootloop information is placed in the FIFO before the parametric registers are loaded, the first byte of bootloop register information is invalid. Accordingly, the sequence of operations must be as follows:

(1) load the 7220 parametric registers
(2) load bootloop data into the 7220 FIFO
(3) send the Write Bootloop Register command.

As a point of interest, if a user wishes to maintain the system bootloop in EPROM rather than to allow automatic handling by the system, the Initialize command would not be used and would be replaced by a sequence similar to the one described.

After the 7220 parametric registers are loaded, the CPU next must load the 7220 FIFO with 40 bytes of bootloop register data using the "write FIFO" sequence from Table 13. This sequence then is followed by the code sequence to issue the Write Bootloop Register command. The data pattern written to the bootloop register should be an easily identified sequence of bytes such as an incrementing pattern. Under operational conditions, the data written to the bootloop registers represents "loop map" information that is written on the label of the 7110 device. Under these test conditions, it only is necessary to ensure that the 40 bytes sent out are the same 40 bytes read back.

Once the Write Bootloop Register command has been sent, the status byte is read (when the BUSY bit goes low) and an operation-complete status is verified. Any parity error indication may be ignored. Valid status at this point indicates that communication with the 7242 has been established. To verify that the data has been transferred properly, the contents of the bootloop register are read into the 7220's FIFO. The CPU then must transfer the data to user RAM in order to compare the data with the original pattern. To read the bootloop register, it only is necessary to issue the Read Bootloop Register command. This command places the contents of the 7242's bootloop register into the 7220's FIFO. The user then must execute the "read FIFO" sequence from Table 14 in order to transfer the data from the 7220 FIFO to RAM. Comparing the loop map written into the bootloop register and the loop map read from the bootloop register should show the loop maps to be equal.

## Installing the 7110 MBM

Reading and writing the 7110 bubble memory requires the application of specific control signals at the appropriate times within the read or write cycles. These control signals originate from the 7254 and 7230 integrated circuits and are generated under the control of the 7220 BMC. Prior to installing the 7110, the presence of the control signals should be verified. While it is unlikely that the 7110 can be seriously damaged, it is possible for the "seeds" and bootloop established at the factory to be lost if there are problems with the 7254 or 7330 control signals and, if lost, would require additional steps on the part of the user to regenerate the seeds and bootloop data. With the dummy module installed, the required control signals can be verified directly on the bubble socket, and the possibility of damaging the bubble can be avoided.

The first control signal waveform to check is the coil drive on pins 9, 10, 11, and 12 of the 7110 socket. The drive current can be verified by ensuring that the voltage waveform on these pins (or on pins 1 and 7 of the 7254) conforms to Figure 6A when the drive field is being rotated. To rotate the drive field, the following code sequence can be used:

1. Write the parametric registers.
2. Send the Read command.

Next, the "cut and transfer" pulses generated during a read operation should be checked. The waveforms on pins 2 and 3 of the 7110 socket (REPLICATE.A and REPLICATE.B), should appear as shown in Figure 6B.

The cut and transfer pulses that occur during a write operation should now be verified. The waveforms on pins 7 and 8 of the 7110 socket (GENERATE. A and GENERATE. B) should appear as shown in Figure 6C. Since a write operation is required, a new code sequence must be used for this test:

1. Write the parametric registers.
2. Write data (any patten) to the FIFO.
3. Send the Write command.

bootloop register of the 7242 first must be loaded to allow data to be written. A Write Bootloop Register Masked command can be used to write a bootloop register pattern of all ones; it is only necessary to write the bootloop register once.

Finally, the SWAP pin is tested for proper operation during a write operation. The waveforms on pins 13 and 14 of the 7110 (SWAP.A and SWAP.B) should appear as shown in Figure 6D. The code sequence described for a write operation may be used.

One additional check of the system should be made prior to installing the 7110 device to determine if valid status values are received after a Read or Write command is issued to the 7220 BMC. Since the bubble is not yet installed, no data actually is transferred; the system should, however, execute the Read or Write command, and valid status should be received. Since a new command cannot be issued to the 7220 while a command is in progress, an Abort command is sent to cancel any command that may be pending from the last test performed. Next, a FIFO Reset command is sent to clear any data remaining in the FIFO. The status byte received should indicate an OP-COMPLETE and FIFO AVAILABLE status condition. The 7220 now is ready to execute a Read or Write command.

First, the 7220 parametric registers are loaded using the modified "diagnostic" driver shown in Table 17. This routine selects one FSA channel (half of a bubble) and, with ECC disabled, requires the loading of only 34 bytes in the 7220 FIFO. By limiting the FIFO to less than 40 bytes, FIFO underflow/overflow conditions are eliminated, and timing errors are avoided in the status byte. After, the 7220 FIFO is preloaded with 34 bytes of data (any pattern), a Write command is issued to the 7220 BMC. The 7220 status value received following command execution should reflect OP-COMPLETE since the 7220 transferred the data from its FIFO to the 7242 and executed the Write command as though the bubble were in place.

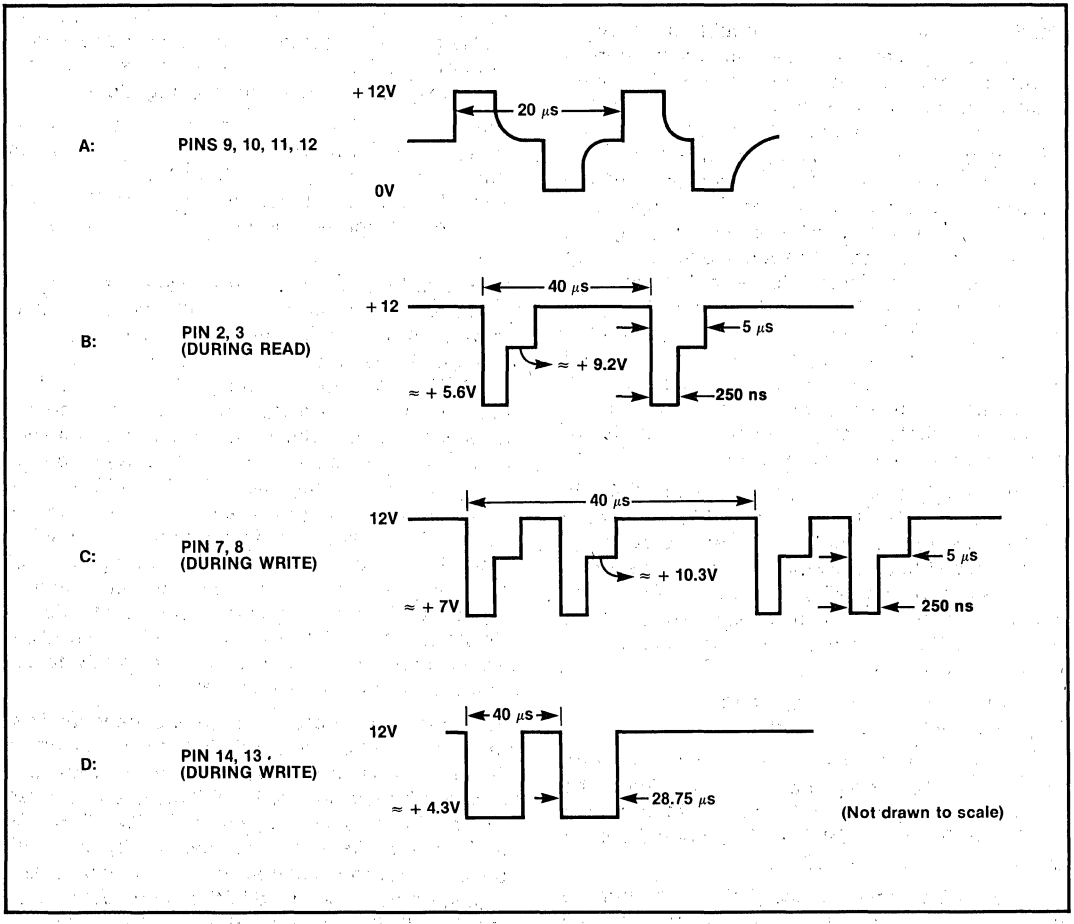



**Figure 6. Control Signal Waveforms**

To test the system in the read mode, the 7220 parametric registers are reloaded and a Read command is issued to the 7220. The user software must now read 34 bytes of "data" from the 7220's FIFO. Note that the data read will consist of all zeroes since no bubble is in place.

When the system completes all of the previous tests successfully, the 7110 bubble memory device may be inserted. Before proceeding, REMOVE POWER FROM THE SYSTEM.

Installing the 7110 is no different from installing any other device. Remove the dummy module in the 7110 socket and insert the 7110 Bubble Memory. Note that the 7110 is keyed to prevent the device from being inserted incorrectly. When power is applied, the system should execute its power-up sequence as described for the dummy module, and the 7220 status byte should return OP-COMPLETE after the parametric registers have been loaded.

**Table 17. Write Register Sequence for One FSA Channel**

```
WTREG1:;            WRITE REGISTERS (ONE HALF BUBBLE)
    ; THIS PROGRAM WRITES THE 7220 REGISTERS "B" THROUGH "F".
    ; DIAGNOSTIC ROUTINE WITH ONE FSA CHANNEL SELECTED
    ; THE FOLLOWING VALUES ARE WRITTEN TO THE 7220 REGISTERS.
    ;                B = 01H  :        1 PAGE TRANSFER
    ;                C = 00H  :        SELECT 1 CHANNEL (HALF BUBBLE)
    ;                D = 08H  :        LOW FREQ
    ;                E = 00H  :        PAGE 0
    ;                F = 00H  :        FIRST BUBBLE
      MOV     AL, 0BH       ; SET REGISTER ADDRESS COUNTER (RAC) TO B REGISTER
      OUT     49H, AL       ; PROT ADDRESS OF 7220 WITH A0 = 1
      MOV     AL, 01H       ; SET B REGISTER TO 01H (ONE PAGE TRANSFER)
      OUT     48H, AL       ; PORT ADDRESS OF 7220 WITH A0 = 0
      MOV     AL, 0H        ; SELECT HALF BUBBLE (1 FSA CHANNEL)
      OUT     48H, AL
      MOV     AL, 08H       ; SELECT LOW FREQ (NO ERROR CORRECTION)
      OUT     48H, AL
      MOV     AL, 0H        ; START ADDRESS = 000H
      OUT     48H, AL
      MOV     AL, 0H        ; SELECT THE FIRST BUBBLE
      OUT     48H, AL
      RET
```

## Normal Read and Write Operations

Under normal operating conditions, a user sends an Initialize command and then proceeds to access the bubble. The Initialize command automatically purges the RAM area of the 7220, reads and decodes the bootloop on the 7110, fills the 7242 bootloop registers, and places the 7110 at page 0. This very important command is the next command to be tested before reading and writing data.

To verify the Initialize command, load the 7220 parametric registers to select both FSA channels for one bubble and then send the Initialize command. Status following execution of this command should be 40H, OP-COMPLETE. Once the 7220 is initialized, data can be transferred to and from the bubble. For a first attempt, it is recommeded that the operations be kept simple. That is, avoid error correction, DMA, or interrupts and only attempt single page transactions until reasonably familiar with the basic operations.

Prior to issuing the Write command, a FIFO Reset command is sent and then the parametric registers are loaded to select the page address and number of FSA channels. After the Write command is sent, the data should be output to the 7220 FIFO. When the proper number of bytes have been transferred, the 7220 status byte should reflect OP-COMPLETE and FIFO AVAILABLE to indicate that the data has been written into the 7110 bubble memory and can now be read. To read back the data written, issue a FIFO Reset command and reload the parametric registers to select the same page address in which the data was written. Issue the Read command to move the data from the 7110 to the 7220 FIFO and then use the "read FIFO" routine to transfer the data to user RAM. As always, the 7220 status byte should be checked after the operation.

## AN IMPLEMENTATION EXAMPLE

To illustrate the ease with which Intel's bubble memory solution may be implemented, an MCS®-86 System Design Kit (SDK-86) is used as a vehicle to control a single BPK 72 bubble memory kit.

The bus interface between the 8086 CPU and the 7220 bubble memory controller requires seven integrated circuits and consists of four sections: address decode, data bus decode and buffering, a clock circuit, and miscellaneous control logic. The system requires power supply voltages of + 12V, + 5V, and, if a CRT is used, − 12V.

The 8086 bus is expanded through two 50-pin, wirewrap connectors, and the BPK 72 is connected to the SDK-86 by a flat cable into a 40-pin connector located on the SDK-86. The following interface diagram shows how the signals required by the bubble system are derived from the 8086. Detailed diagrams of the address, data, clock and control logic are in the appendix.

Either the SDK-86's Keypad or Serial monitor may be used to write and debug the necessary software drivers to control the BPK 72. There is, however, an EPROM-based monitor (BMDSDK) explicity designed for the BPK 72 and is available from the Intel Insite Library. Some of the bubble-specific portions of this monitor are discussed in the following text.

### Monitor Software

The BMDSDK Bubble Monitor is a highly-modular program that is written in 8086 assembly language and that resides in two 2716 EPROMs. This monitor implements, at the console level, most of the standard SDK-86 monitor functions (display/change memory, etc.) and all of the 7220 commands. The current version of the monitor utilizes only polled I/O protocol; implementing an interrupt-driven system on the SDK-86 is possible using the principles outlined in this application note. The DMA mode of operation is not available with the hardware described.

The BPK 72 driver routines are confined to one module; a listing of this module is included in the appendix. To provide some feeling for the elements of "operational" software as opposed to the test drivers discussed earlier, the write function implemented in BMDSDK monitor is examined. The flow chart in Figure 9 shows how the routine is constructed on a functional basis. Note that the subroutine reflects a very "safe" approach in that the FIFO Reset command always is sent prior to issuing the Write command. While the FIFO Reset command is not mandatory, if there is any a doubt regarding the state of the FIFO prior to a read or write operation, resetting the FIFO is a good idea. Note also that a running byte count is maintained and that the routine exits when the count goes to zero. Such a counter is not actually necessary; the FIFO AVAILABLE bit alone can be used to gate the data to the 7220.

The calling program supplies the BMWRIT routine with the total number of bytes to be transferred in the CX register. The total number of bytes written is sent to the console at the end of the operation as a monitor function. BMWRIT also returns the value of the status byte to the calling program.

Note that at label WRIT01, the routine does not progress after the Write command is sent unless both the BUSY and FIFO AVAILABLE bits are set by the controller. Once these values are set, the code issues a byte of data to the controller only if the FIFO AVAILABLE bit indicates there is room. The remainder of the code in BMWRIT is concerned with processing special write requests for the bootloop and bootloop register commands.

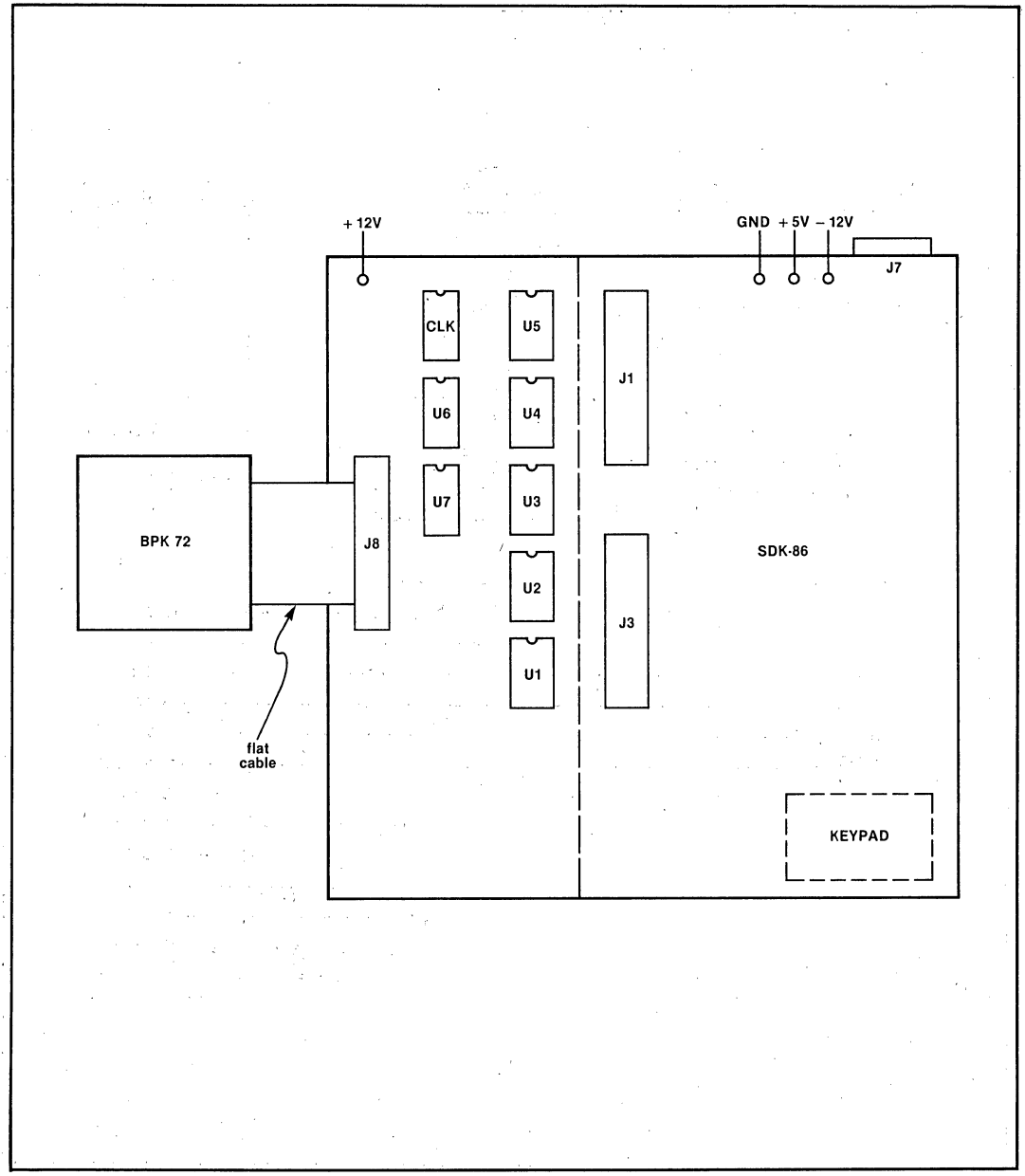



**Figure 7. SDK-86/BPK 72 Implementation**

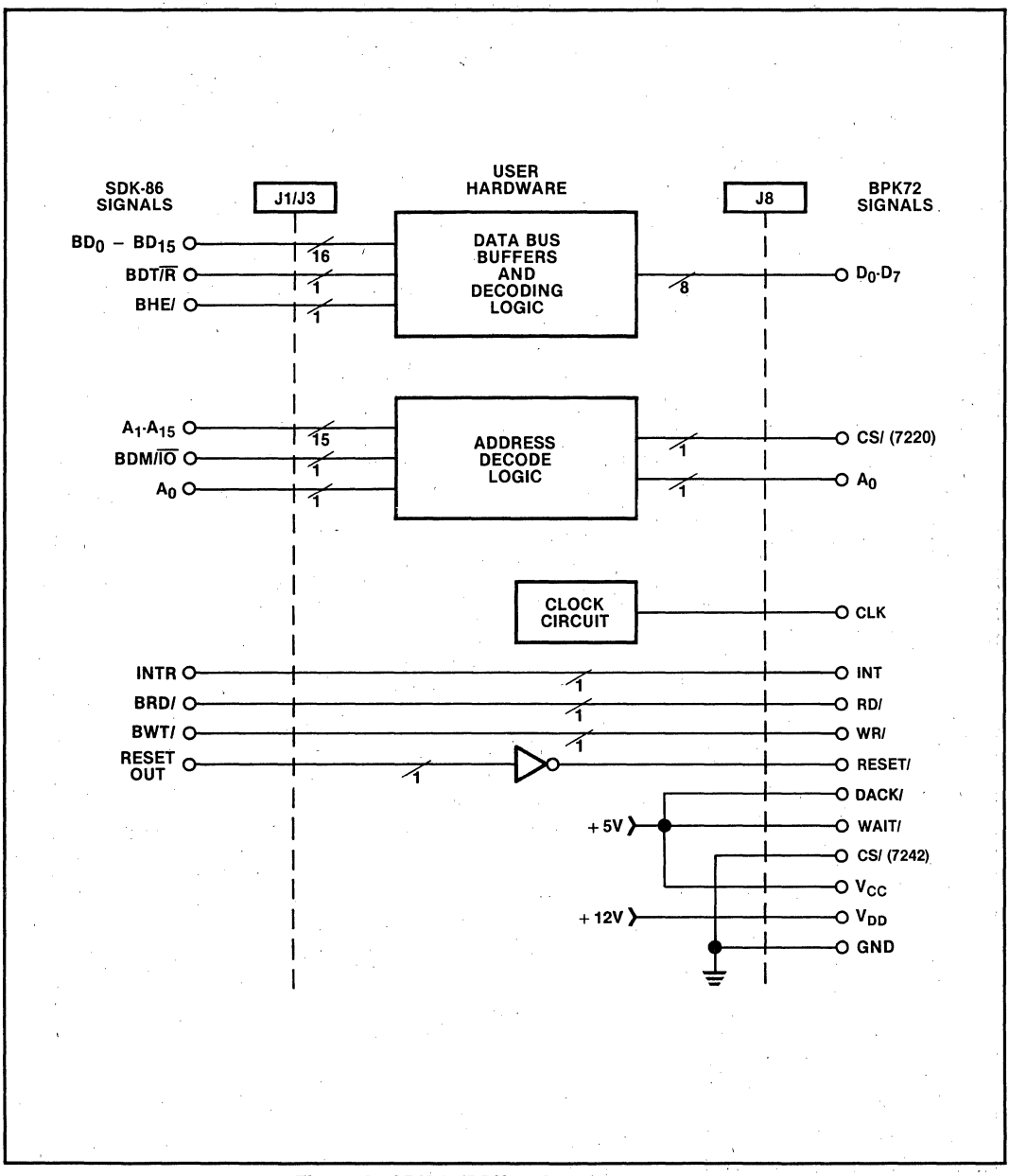



**Figure 8. SDK-86/BPK 72 Interface Diagram**

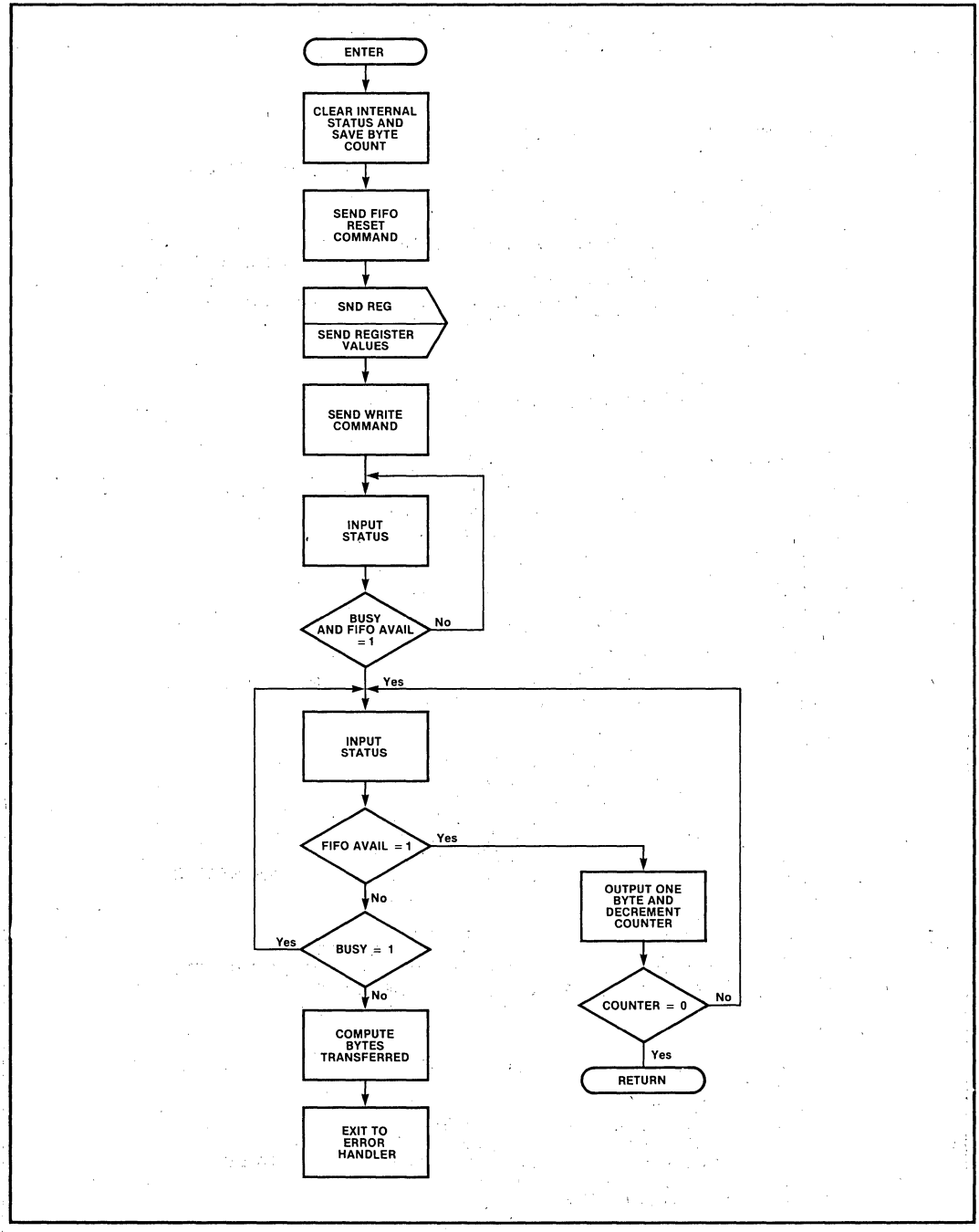



**Figure 9. BMWRIT Flowchart**

**Table 18. BMWRIT Procedure for the SDK-86**

```
;
;  FUNCTION: BMWRIT - WRITE BUBBLE MEMORY DATA.
;  INPUTS: CX = # OF BYTES TO WRITE.
;  OUTPUTS: A = STATUS: F/F(C = 1: ERROR OCCURED) BX = # OF BYTES WRITTEN.
;  CALLS: SNDREG, BMWAIT.
;  DESTROYS: ALL.
;  DESCRIPTION:   THIS PROCEDURE PERFORMS A BUBBLE MEMORY WRITE OPERATION.
;                 AN ERROR WILL OCCUR IF THE NUMBER OF BYTES GIVEN FOR THE
;                 WRITE OPERATION EXCEED THE NUMBER THAT THE BMC EXPECTS
;                 (DERIVED FROM COMMAND, BLOCK LENGTH AND NUMBER OF FSA
;                  CHANNELS), OR IF THE NUMBER OF BYTES IS LESS THAN THAT
;                  WHICH THE BMC EXPECTS.
;
BMWRIT:
        XOR     AL, AL              ; A = 0
        MOV     STATUS, AL          ; CLEAR STATUS
        MOV     BX, CX
        MOV     AL, CFR
        OUT     BMSTAT, AL          ; FIFO RESET
        CALL    SNDREG              ; SEND REGISTERS TO BMC.
        MOV     SI, BUFADR          ; SET UP SRC BFR PTR (IN DATA SEG)
        MOV     AL, BMCMD           ; GET COMMAND
        OUT     BMSTAT, AL          ; ISSUE IT.
WRIT01:
        IN      AL, BMSTAT
        TEST    AL, BUSYBT          ; WAIT FOR BUSY. . .
        JZ      WRIT01
        TEST    AL, FIFOBT          ; AND FIFO READY
        JZ      WRIT01
;
;  KEEP STUFFING DATA INTO FIFO UNTIL DONE OR AN ERROR OCCURS.
;  (NOTE: BMC GOING NOT BUSY IS AN ERROR).
;
WRIT03:
        IN      AL, BMSTAT          ; GET STATUS
        TEST    AL, FIFOBT          ; FIFO READY?
        JZ      WRIT04              ; NO, WAIT FOR IT
        LODSB                       ; YES, GET DATA FOR IT
        OUT     BMDATA, AL          ; GIVE IT TO BMC
        LOOP    WRIT03              ; LOOP UNTIL DONE.
        JMP     BMWAIT              ; XFER DONE, WAIT FOR A GOOD STATUS
WRIT04:
        TEST    AL, BUSYBT
        JNZ     WRIT03              ; OK IF STILL BUSY
        SUB     BX, CX              ; BX:# OF BYTES XFERED
        JMP     CTRL99              ; ERROR IF NOT BUSY AND CX NOT ZERO
;  SPECIAL WRITE FOR BOOTLOOP AND BOOTLOOP REG CMNDS
;
BMWRTB:
        XOR     AL, AL              ; A = 0
        MOV     STATUS, AL          ; CLEAR STATUS
        MOV     BX, CX
        MOV     AL, CFR
        OUT     BMSTAT, AL          ; FIFO RESET
        CALL    SNDREG              ; SEND REGISTERS TO BMC.
        MOV     SI, BUFADR          ; SET UP SRC BFR PTR (IN DATA SEG)

  ; FILL FIFO WITH 20/40/41 BYTES
```

**Table 19. BMWRIT Procedure for the SDK-86 (cont.)**

```
;
;
;
WRTB01:
        LODSB
        OUT     BMDATA, AL      ; STICK IN FIFO.
        LOOP    WRTB01          ; LOOP UNTIL FILL COUNT = 0.
        IN      AL, BMSTAT      ; GET BMC STATUS
        TEST    AL, BUSYBT      ; CHECK BUSY BIT.
        JZ      SHORT WAITEX    ; NOT BUSY, ALREADY DONE.
        MOV     CX, OFFFFH      ; JUST IN CASE. . .
WAITPO:                         ; POLLED WAIT MODE
        IN      AL, BMSTAT      ; GET STATUS
        TEST    AL, BUSYBT      ; CHECK BUSY BIT
        LOOPNZ  WAITPO          ; LOOP IF STILL BUSY
        JCXZ    CTRL99          ; PROBABLY AN ERROR IF CX = 0
WAITE:
        MOV     STATUS, AL      ; A = STATUS
        RET
```

## SUMMARY

The purpose of this application note is to provide a more clear understanding of the functions and characteristics of the BPK 72 one-megabit bubble memory kit. This kit has been designed specifically to relieve the user of the design effort that historically is associated with implementing a bubble memory system, and to provide a simple interface that is compatible with a broad range of microprocessor systems.

The BPK 72 is a subsystem in itself that should be viewed as simply one more component on the system bus. This component-level approach, plus the inherent flexibility of the kit, provides the user with maximum utility and functionality. By understanding how each of the subsystem parts fits together and by approaching the implementation of the kit in a methodical fashion as described in this note, the development of a working system is facilitated.

# APPENDIX A

# SDK-86/BPK 72
# HARDWARE INTERFACE

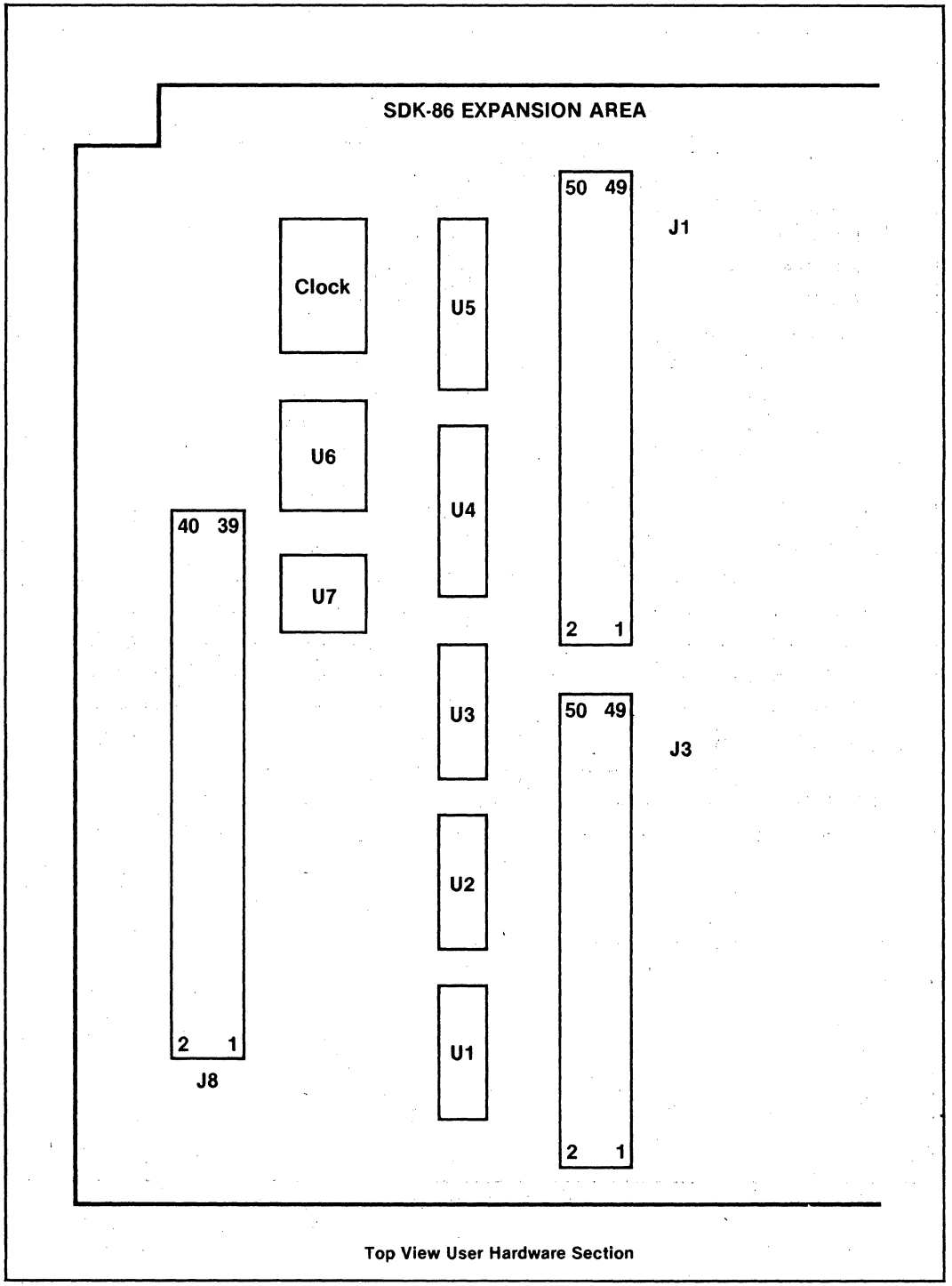
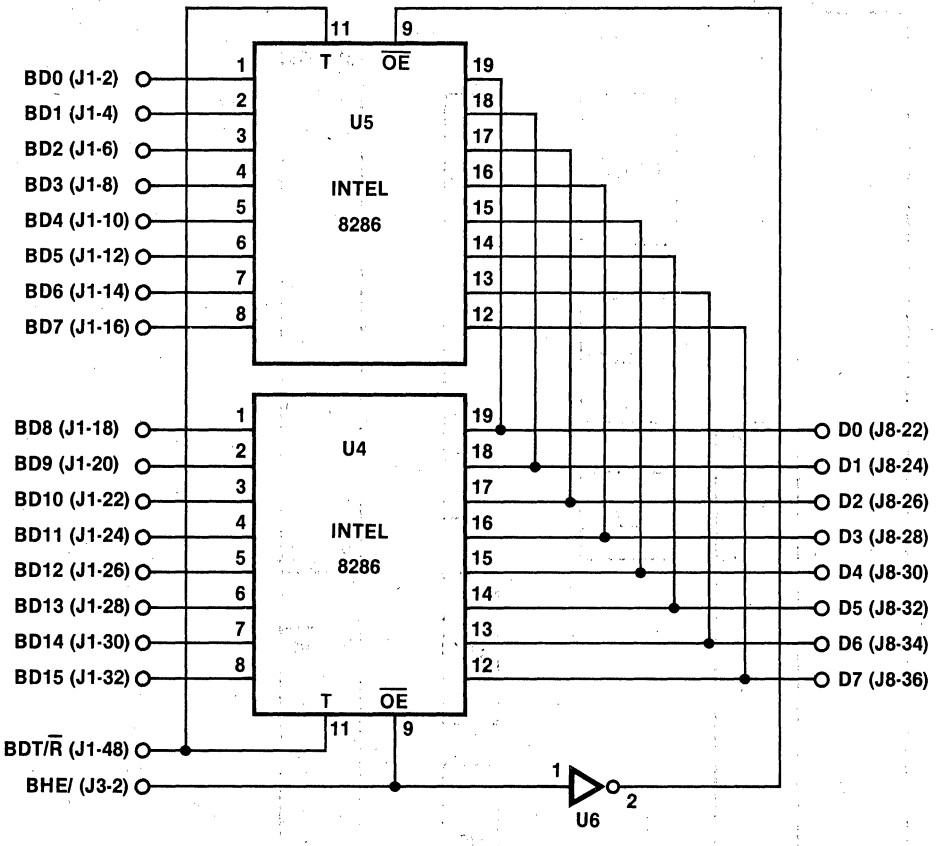



**Figure 10. Parts Layout**




8286 – VCC = 20
Gnd = 10

**Figure 11. Data Bus Buffer and Decoding Logic**

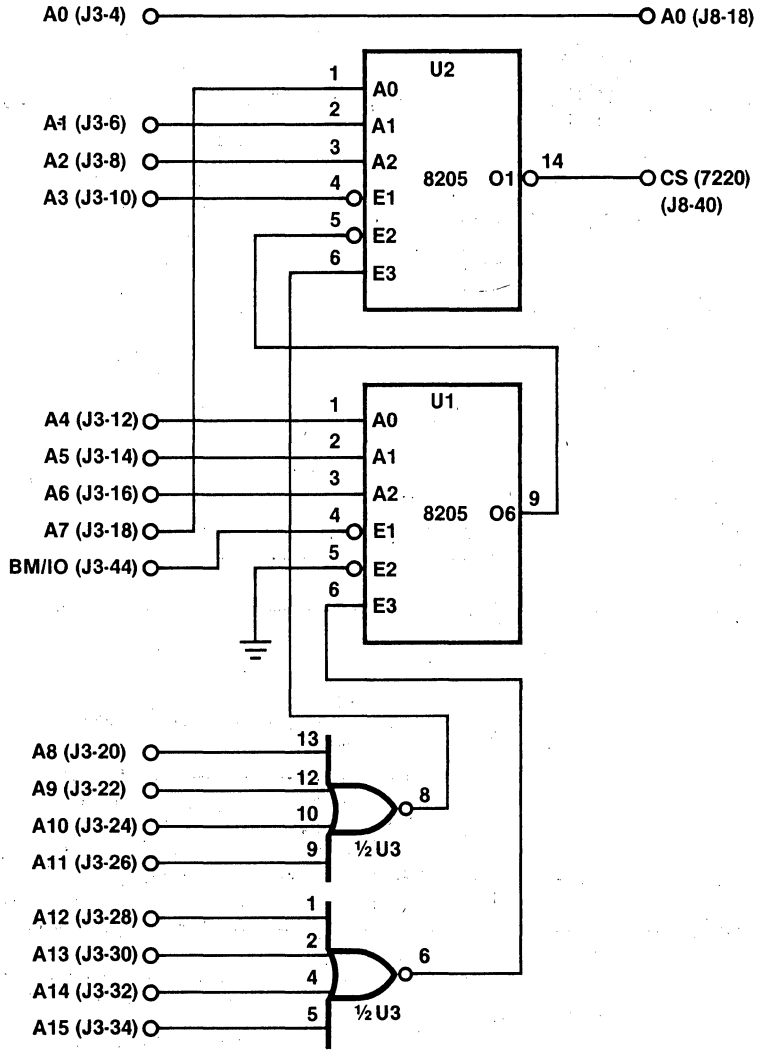



Figure 12. Address Decode Logic

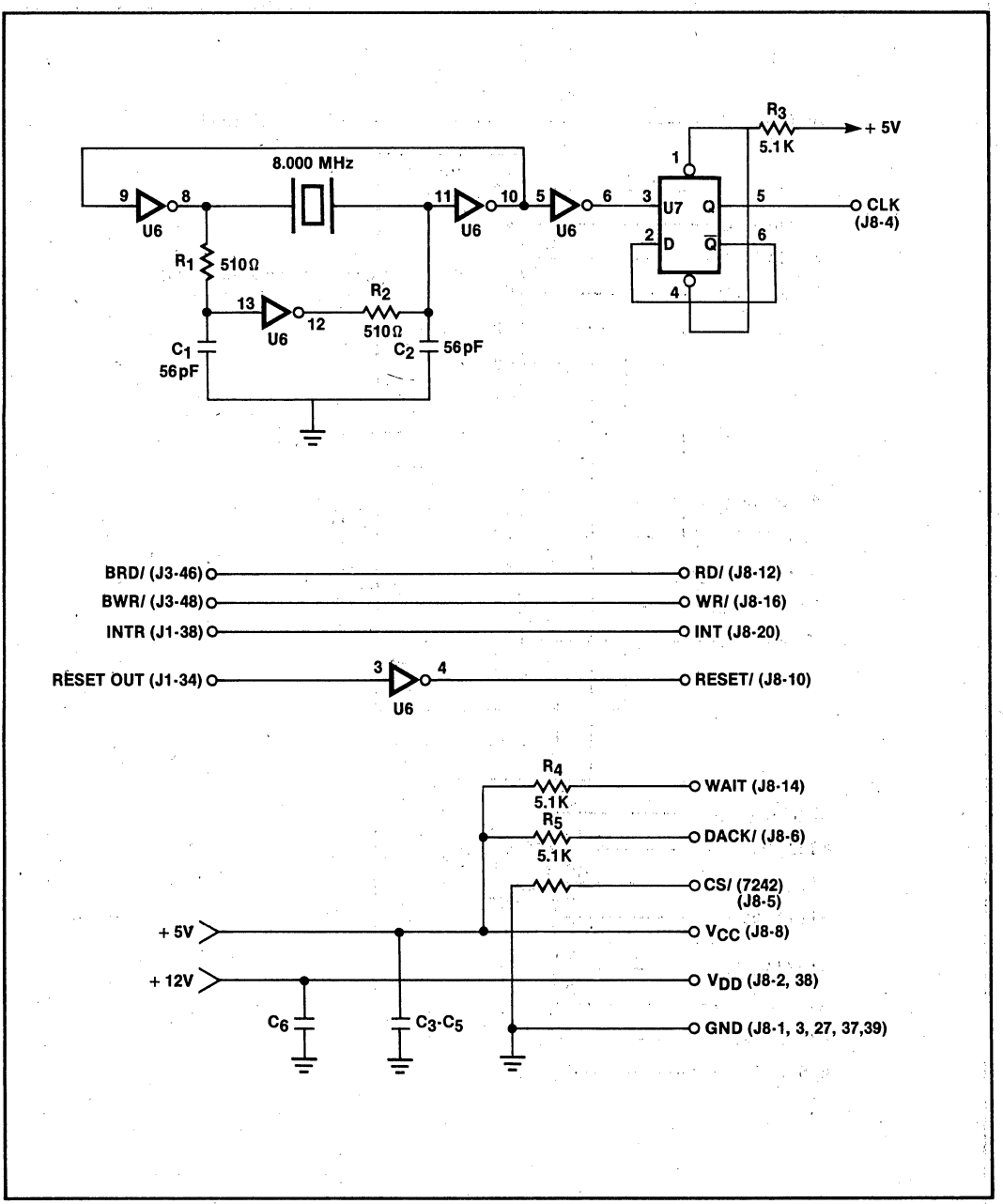



**Figure 13. Clock Circuit and Control Signals**

**Table 20. SDK-86 Pinout**

| Pin | J1/J2 | J3/J4 | J5 | J6 |
|---|---|---|---|---|
| 2 | BD0 | BHE/ | P2C1 | — |
| 4 | BD1 | A0 | P2C2 | P1B3 |
| 6 | BD2 | A1 | P2C3 | P1B4 |
| 8 | BD3 | A2 | P2B7 | P1B2 |
| 10 | BD4 | A3 | P2B0 | P1B5 |
| 12 | BD5 | A4 | P2B6 | P1B1 |
| 14 | BD6 | A5 | P2B3 | P1B6 |
| 16 | BD7 | A6 | P2B4 | P1B0 |
| 18 | BD8 | A7 | P2B2 | P1B7 |
| 20 | BD9 | A8 | P2B5 | P1C3 |
| 22 | BD10 | A9 | P2B1 | P1C2 |
| 24 | BD11 | A10 | P2C0 | P1C1 |
| 26 | BD12 | A11 | P2C4 | P1C0 |
| 28 | BD13 | A12 | P2C5 | P1C4 |
| 30 | BD14 | A13 | P2C6 | P1C5 |
| 32 | BD15 | A14 | P2C7 | P1C6 |
| 34 | RESET OUT | A15 | P2A0 | P1C7 |
| 36 | PCLK/ | A16 | P2A7 | P1A0 |
| 38 | INTR | A17 | P2A1 | P1A7 |
| 40 | TEST | A18 | P2A6 | P1A1 |
| 42 | HOLD | A19 | P2A2 | P1A6 |
| 44 | BHLDA | BM/IO/ | P2A5 | P1A2 |
| 46 | BDEN/ | BRD/ | P2A3 | P1A5 |
| 48 | BDT/R/ | BWR/ | P2A4 | P1A3 |
| 50 | BALE | BINTA/ | — | P1A4 |

All Odd Pins are Ground except as follows:

| | J2 |
|---|---|
| 41 | CSX/ (FD000-FDFFF) |
| 43 | CSY/ (FC000-FCFFF) |
| 45 | BS3 |
| 47 | BS4 |
| 49 | BS5 |

**Table 21. SDK-86/BPK 72 Cable Wiring**

| Signal | J8 | P1 |
|---|---|---|
| + 12v | 2, 38 | B, X |
| + 5v | 8 | F |
| Ground | 1, 3, 27, 37, 39 | 1, A, P, 22, Z |
| D0 | 22 | 11 |
| D1 | 24 | 12 |
| D2 | 26 | 13 |
| D3 | 28 | 14 |
| D4 | 30 | 15 |
| D5 | 32 | 16 |
| D6 | 34 | 17 |
| D7 | 36 | 18 |
| CS/ (7220) | 40 | Y |
| A0 | 18 | 10 |
| RD/ | 12 | J |
| WR/ | 16 | K |
| INT | 20 | N |
| RESET/ | 10 | H |
| CS/ (7242) | 5 | E |
| WAIT/ | 14 | 8 |
| CLK | 4 | 4 |
| DACK/ | 6 | L |

Cable is standard 40 conductor Flat Cable.
All Odd Conductors are grounded at J8.

**Table 22. SDK-86/BPK 72 Parts List**

| Item | Description | QT | Ref | |
|---|---|---|---|---|
| 1 | IC-8205 - Bindry Decoder | 2 | U1, U2 | Intel (TI-74LS13) |
| 2 | IC-8286 - Octal Bus Tranciever | 2 | U4, U5 | Intel |
| 3 | IC-746525 - Dual 4 Input M | 1 | U3 | Any |
| 4 | IC-74H04 - Inverter | 1 | U6 | Any |
| 5 | Resistor 510Q 1/4w | 2 | R1, R2 | Any |
| 6 | Capacitor, 56pF 25V | 2 | C1,C2 | Any |
| 7 | Capacitor, .1pF 25V | 4 | C3-C6 | Any |
| 8 | Crystal, 8.000MHz Serie Res. | 1 | Y1 | Any |
| 9 | Connector, 50 pin wirewrap | 2 | J1, J3 | 3M # 3433 |
| 10 | Connector, 40 pin wirewrap | 1 | J8 (M) | 3M # 3432 |
| 11 | Connector, 40 pin | 1 | J8 (F) | 3M # 3417 |
| 12 | Connector, 44 pin Edge w/w | 1 | P1 | Any |
| 13 | IC Socket, 20 pin w/w | 2 | | Any (Augat) |
| 14 | IC Socket, 16 pin w/w | 3 | | Any |
| 15 | IC Socket, 14 pin w/w | 3 | | Any |
| 16 | Adapter Plug Assembly, 16 pin | 1 | | Augat#616-CE1 |
| 17 | Flat Cable, 40 Conductor, 1 Ft. | 1 | | 3M # 3365 |
| 18 | IC-74LS74 - Dual D Flip-Flop | 1 | 07 | Any |
| 19 | Resistor 5.1K 1/4W ±5% | 3 | R3, R4, R5 R5 | Any |
| 20 | IC-74LS32 - OR Gate | 1 | U8 | Any |

# APPENDIX B

# SDK-86/BPK 72
# SOFTWARE DRIVER

```
ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE DRIVER
OBJECT MODULE PLACED IN :F1:DRIVER.OBJ
ASSEMBLER INVOKED BY:  asm86 :f1:DRIVER.a86 xref print(:f1:DRIVER.lst) debug WORKFILES(:F0:,:F0:)

LOC  OBJ                  LINE     SOURCE

                             1     $TITLE(                        BPK-72 DRIVER ROUTINES.)
                             2             NAME    DRIVER
                             3 +1  $INCLUDE(:F1:RAMDEF.EXT)
                         =1  4     ;
                         =1  5     ;       publics from module RAMDEF, file RAMDEF.A86
                         =1  6     ;
----                     =1  7     STACK   SEGMENT STACK
                         =1  8             EXTRN   BMSTAK:NEAR
----                     =1  9     STACK   ENDS
                         =1 10     ;
----                     =1 11     DATA    SEGMENT PUBLIC
                         =1 12             EXTRN   RAM:BYTE,SCRBUF:BYTE,MYBUF:BYTE
                         =1 13             EXTRN   DEFADR:WORD,DEFBUB:BYTE,DEFNFC:BYTE,DEFENA:BYTE
                         =1 14             EXTRN   DEFMOD:BYTE,DEFPAG:WORD,DEFBLK:WORD
                         =1 15             EXTRN   BUFADR:WORD,BLKLEN:WORD,ENABLE:BYTE,PAGENO:WORD
                         =1 16             EXTRN   BBLNUM:BYTE,NFC:BYTE,MODE:BYTE,STATUS:BYTE,BMCMD:BYTE
                         =1 17             EXTRN   INBUF:BYTE,INBUFP:WORD,INBUFC:BYTE
                         =1 18             EXTRN   INBUFA:WORD,INBUFL:BYTE
                         =1 19             EXTRN   OUTBUF:BYTE,OUTBFP:WORD,OUTBFC:BYTE
                         =1 20             EXTRN   OUTBFA:WORD,OUTBFL:BYTE
                         =1 21             EXTRN   RDLEN:WORD,WRLEN:WORD
                         =1 22             EXTRN   PROMPT:BYTE,LEVMSK:BYTE
                         =1 23             EXTRN   BPADR:WORD,USERRG:WORD
                         =1 24             EXTRN   POPREGS:WORD,PUSHREGS:WORD
                         =1 25             EXTRN   USERBX:WORD,USERDS:WORD,USERBP:WORD,USERSS:WORD
                         =1 26             EXTRN   USERSP:WORD,USERIP:WORD,USERCS:WORD,USERFL:WORD
                         =1 27             EXTRN   USERPC:WORD
----                     =1 28     DATA    ENDS
                         =1 29     ;
                            30 +1  $INCLUDE(:F1:BMC.EQU)
                         =1 31     ;
                         =1 32     ; THESE ARE THE COMMAND EQUATES FOR BMDS
                         =1 33     ;
  0010                   =1 34     CWBRM   EQU     10H                     ; WRITE BOOTLOOP WITH MASK.
  0011                   =1 35     CIZ     EQU     11H                     ;INITIALIZE
  0012                   =1 36     CRD     EQU     12H                     ;READ
  0013                   =1 37     CWD     EQU     13H                     ;WRITE
  0014                   =1 38     CRS     EQU     14H                     ;READ SEEK
  0015                   =1 39     CRBR    EQU     15H                     ;READ BOOTLOOP REGISTER
  0016                   =1 40     CWBR    EQU     16H                     ;WRITE BOOTLOOP REGISTER
  0017                   =1 41     CWB     EQU     17H                     ;WRITE BOOTLOOP
  0018                   =1 42     CRFS    EQU     18H                     ;READ FIFO STATUS
  0019                   =1 43     CAB     EQU     19H                     ;ABORT
  001A                   =1 44     CWRS    EQU     1AH                     ; WRITE SEEK.
  001B                   =1 45     CRB     EQU     1BH                     ;READ BOOTLOOP
  001C                   =1 46     CRCD    EQU     1CH                     ;READ CORRECTED DATA
  001D                   =1 47     CFR     EQU     1DH                     ;FIFO RESET
  001E                   =1 48     CPURG   EQU     1EH                     ; MBM PURGE COMMAND.
  001F                   =1 49     CSR     EQU     1FH                     ;SOFTWARE RESET
                         =1 50     ;
```

```
LOC  OBJ                  LINE     SOURCE

                    =1      51     ; I/O PORT ADDRESSES.
                    =1      52     ;
  00E1              =1      53     BMSTAT  EQU     0E1H            ; BUBBLE MEMORY DEVICE STATUS PORT.
  00E0              =1      54     BMDATA  EQU     0E0H            ; BUBBLE MEMORY DEVICE DATA PORT.
                    =1      55     ;
                    =1      56     ; STATUS WORD BITS
                    =1      57     ;
  0001              =1      58     FIFOBT  EQU     01H             ; FIRST BIT IS FIFO STATUS
  0002              =1      59     PARERR  EQU     02H             ; SECOND BIT IS PARITY ERROR.
  0004              =1      60     UNCERR  EQU     04H             ; THIRD BIT IS UNCORRECTABLE ERROR BIT.
  0008              =1      61     CORERR  EQU     08H             ; FOURTH BIT IS CORRECTABLE ERROR BIT.
  0010              =1      62     TIMERR  EQU     10H             ; FIFTH BIT IS TIMING ERROR BIT.
  0020              =1      63     OPFAIL  EQU     20H             ; OPERATION FAIL BIT.
  0040              =1      64     OPDONE  EQU     40H             ; OPERATION COMPLETE BIT.
  0080              =1      65     BUSYBT  EQU     80H             ; BUSY BIT.
                    =1      66     ;
                    =1      67     ; ENABLE REG BITS
                    =1      68     ;
  0001              =1      69     INTENA  EQU     01H             ; INTERRUPT NORMAL
  0002              =1      70     IERENA  EQU     02H             ; INTERRUPT ERROR
  0004              =1      71     DMAENA  EQU     04H             ; DMA
  0008              =1      72     RSVD1   EQU     08H
  0010              =1      73     WBLENA  EQU     10H             ; WRITE BOOTLOOP
  0020              =1      74     RCDENA  EQU     20H             ; READ CORRECTED DATA
  0040              =1      75     ICDENA  EQU     40H             ; INTERNALLY CORRECTED DATA
  0080              =1      76     RSVD2   EQU     80H
                            77 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                            78     CODE    SEGMENT PUBLIC
                            79             ASSUME  DS:DATA,CS:CODE,SS:STACK
                            80     ;*************************************************************
                            81     ;                                                           *
                            82     ;                 BPK72 DRIVER routines                     *
                            83     ;                 ===== ====== ========                     *
                            84     ;                                                           *
                            85     ;  The routines in this module constitute the routines      *
                            86     ;  needed to directly drive the BPK72 bubble memory         *
                            87     ;  development board.  This module is designed to be self   *
                            88     ;  contained, and may be called by ANY user procedures.     *
                            89     ;                                                           *
                            90     ;          The procedures in this module are                *
                            91     ;                                                           *
                            92     ;  BMCTRL - Perform non-data transfer BMC operations.      *
                            93     ;  BMREAD - Perform data read BMC operations.              *
                            94     ;  BMWRIT - Perform data write BMC operations.             *
                            95     ;                                                           *
                            96     ;  ZAPREG - Set internal registers to an acceptable value  *
                            97     ;                                                           *
                            98     ;          Parameter passing                                *
                            99     ;          ========= =======                                *
                           100     ;                                                           *
                           101     ;  All parameters are passed to the BMC driver routines via *
                           102     ;  common (PUBLIC) variables.  These variables are        *
                           103     ;                                                           *
                           104     ;  BUFADR - The memory address of the input/output buffer  *
                           105     ;            to be used for data transfer operations.       *
                           106     ;  ENABLE - The enable byte to be passed to the BMC before *
                           107     ;            every operation.                               *
                           108     ;  PAGENO - The starting block number to be passed to the  *
                           109     ;            BMC before every operation.  (NOTE: This field *
                           110     ;            has no meaning for control operations).        *
                           111     ;  BLKLEN - The number of pages to be transfered by the BMC.*
                           112     ;            (NOTE: This field has no meaning for control   *
                           113     ;            operations).                                   *
                           114     ;  BBLNUM - The bubble select to be transfered to the BMC  *
                           115     ;            before every operation.(NOTE: This field has   *
                           116     ;            no meaning for SOME control operations).       *
                           117     ;  NFC    - The number of FSA channels passed to the BMC    *
                           118     ;            before every operation. (NOTE: This field has  *
                           119     ;            no meaning for SOME of the control operatioNs).*
                           120     ;                                                           *
                           121     ;  For a detailed definition of the ENABLE,PAGENO.BLKLEN,  *
                           122     ;   BBLNUM. and NFC fields, refer to the BPK-72 USER MANUAL *
                           123     ;   or the Bubble Memory Design Handbook.                  *
                           124     ;                                                           *
                           125     ;*************************************************************
                           126 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           127     ;*******************************
                           128     ;
                           129     ; ENTRY POINTS
                           130     ;
                           131             PUBLIC  ZAPREG,BMCTRL,BMWAIT,BMREAD,BMWRIT,BMWRTB
                           132     ;
                           133     ;***********************
                           134     ;
                           135     ; MISC EQUATES
                           136     ;
  000B                     137     REG1    EQU     0BH                     ; FIRST BMC REGISTER TO USE IS BLOCK LENGTH
  003C                     138     STATER  EQU     3CH                     ; STATUS WORD ERROR MASK
                           139                                             ; IGNORE PARITY ERR, REV D OF BMC
                           140 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           141     ;*******************************
                           142     ;
                           143     ;    MODE BYTE DEFINITION
                           144     ;    ==== ==== ==========
                           145     ;
                           146     ; The bits in the MODE BYTE specify the type of the data transmission
                           147     ; TO USE, AND WHETHER TO PRINT STATUS AFTER EACH OPERATION.
                           148     ; If interrupts are enabled in the MODE BYTE, they must also be selected
                           149     ; in the ENABLE BYTE for desired operation to occur.
                           150     ;
  0001                     151     INTMOD  EQU     01H             ; FIRST BIT IN MODE WORD IT INTERRUPT SELECT.
  0002                     152     DMAMOD  EQU     02H             ; SECOND BIT IN MODE WORD IS DMA SELECT.
  0080                     153     DBGMOD  EQU     80H             ; DEBUG BIT OF MODE WORD
                           154 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          155      ;*******************************
                          156      ;
                          157      ; FUNCTION: BMCTRL - PERFORM BMC CONTROL OPERATIONS (NON-DATA TRANSFER).
                          158      ; INPUTS: NONE
                          159      ; OUTPUTS: A=STATUS;F/F(C=1: AN ERROR OCCURED).
                          160      ; CALLS: SNDREG,BMWAIT
                          161      ; DESTROYS: ALL
                          162      ; DESCRIPTION: THIS PROCEDURE IS USED TO PERFORM NON-DATA TRANSFER
                          163      ;              BMC OPERATIONS.
                          164      ;
0000                      165      BMCTRL:
0000 E8D700               166              CALL    SNDREG                  ; LOAD BMC REGISTERS.
0003 A00000         E     167              MOV     AL,BMCMD                ; GET COMMAND.
0006 E6E1                 168              OUT     BMSTAT,AL               ; INITIATE COMMAND.
0008 E80E00               169              CALL    BMWAIT                  ; WAIT FOR COMPLETION.
000B 243C                 170              AND     AL,STATER               ; DO WE HAVE AN ERROR?
000D A00000         E     171              MOV     AL,STATUS               ; LOAD STATUS INTO 'A' FOR EXIT
0010 7502                 172              JNZ     SHORT CTRL99            ; ERROR, RETURN WITH FLAG SET.
0012 F8                   173              CLC                             ; CLEAR CARRY(ERROR FLAG)
0013 C3                   174              RET                             ; AND RETURN
                          175      ;
                          176      ; WE HAD AN ERROR, RETURN WITH ERROR FLAG(CARRY FLAG) SET.
                          177      ;          THIS IS THE GENERAL ERROR EXIT
                          178      ;
0014                      179      CTRL99:
0014 A20000         E     180              MOV     STATUS,AL
0017 F9                   181              STC                             ; SET ERROR FLAG (CARRY FLAG)
0018 C3                   182              RET                             ; AND RETURN.
                          183      ;*******************************
                          184      ;
                          185      ; FUNCTION: BMWAIT
                          186      ; INPUTS: NONE
                          187      ; OUTPUTS: STATUS IN A
                          188      ; CALLS: NOTHING
                          189      ; DESTROYS: A,F/F
                          190      ; DESCRIPTION: THIS PROCEDURE WILL WAIT UNTIL THE CURRENT BMC
                          191      ;              OPERATION COMPLETES.
                          192      ;
0019                      193      BMWAIT:
                          194      ;
                          195      ; CHECK CURRENT STATUS (GOOD ONLY IF RAC=0 AND BSY=0)
                          196      ;
0019 E4E1                 197              IN      AL,BMSTAT               ; GET BMC STATUS
001B A880                 198              TEST    AL,BUSYBT               ; CHECK BUSY BIT.
001D 740B                 199              JZ      SHORT WAITEX            ; NOT BUSY, ALREADY DONE.
001F B9FFFF               200              MOV     CX,0FFFFH               ; JUST IN CASE...
0022                      201      WAITPO:                                 ; POLLED WAIT MODE
0022 E4E1                 202              IN      AL,BMSTAT               ; GET STATUS
0024 A880                 203              TEST    AL,BUSYBT               ; CHECK BUSY BIT
0026 E0FA                 204              LOOPNZ  WAITPO                  ; LOOP IF STILL BUSY
0028 E3EA                 205              JCXZ    CTRL99                  ; PROBABLY AN ERROR IF CX=0
002A                      206      WAITEX:                                 ; CORRECT STATUS AND RETURN.
002A A20000         E     207              MOV     STATUS,AL               ; A = STATUS
002D C3                   208              RET
                          209 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           210     ;*****************************
                           211     ;
                           212     ; FUNCTION: BMREAD
                           213     ; INPUTS: CX = NUMBER OF BYTES TO READ, ES SET TO DS
                           214     ; OUTPUTS: A = STATUS; F/F(C=1: ERROR OCCURED)
                           215     ;          BX = NUMBER OF BYTES READ
                           216     ; CALLS: SNDREG
                           217     ; DESTROYS: ALL
                           218     ; DESCRIPTION: ALL PARAMETERS ARE PASSED THROUGH COMMON(PUBLIC)
                           219     ;              VARIABLES( SEE MODULE HEADER).
                           220     ;
002E                       221     BMREAD:
002E 32C0                  222             XOR     AL,AL                   ; A = 0
0030 A20000          E     223             MOV     STATUS,AL               ; CLEAR STATUS.
0033 8BD9                  224             MOV     BX,CX                   ; SAVE BYTE COUNT FOR LOOP
0035 E8A200                225             CALL    SNDREG                  ; SEND REGISTERS TO BMC.
0038 8B3E0000        E     226             MOV     DI,BUFADR               ; SET UP DEST BFR PTR (IN EXTRA SEG)
003C 8CD8                  227             MOV     AX,DS
003E 8EC0                  228             MOV     ES,AX                   ; SET EXTRA SEG FOR BYTE MOVE DEST
0040 A00000          E     229             MOV     AL,BMCMD                ; GET COMMAND
0043 E6E1                  230             OUT     BMSTAT,AL               ; ISSUE IT.
                           231     ;
0045 B9FFFF                232             MOV     CX,0FFFFH
0048                       233     BMRD1:
0048 E4E1                  234             IN      AL,BMSTAT
004A A880                  235             TEST    AL,BUSYBT
004C E1FA                  236             LOOPZ   BMRD1                   ; WAIT FOR BUSY, BUT NOT FOREVER
004E E3C4                  237             JCXZ    CTRL99                  ; CX=0 PROBABLY AN ERROR
0050 8BCB                  238             MOV     CX,BX
                           239     ;
                           240     ;        READ LOOP
                           241     ;        ==== ====
                           242     ;
0052                       243     BMRD2:
0052 E4E1                  244             IN      AL,BMSTAT               ; GET STATUS
0054 A801                  245             TEST    AL,FIFOBT               ; FIFO EMPTY?
0056 7407                  246             JZ      SHORT BMRD3             ; YEP, GO CHECK FOR BUSY.
0058 E4E0                  247             IN      AL,BMDATA               ; NOPE, GET DATA
005A AA                    248             STOSB                           ; STORE IT
005B E2F5                  249             LOOP    BMRD2                   ; AND GO FOR MORE.
005D EBBA                  250             JMP     BMWAIT                  ; XFER DONE, WAIT FOR A GOOD STATUS
005F                       251     BMRD3:                                  ; NOTHING IN FIFO, IS OP COMPLETE?
005F A880                  252             TEST    AL,BUSYBT               ; CHECK BUSY BIT
0061 75EF                  253             JNZ     BMRD2                   ; STILL BUSY, WAIT.
0063 2BD9                  254             SUB     BX,CX                   ; BX <- # OF BYTES XFERED
0065 EBAD                  255             JMP     CTRL99
                           256 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          257      ;*****************************
                          258      ;
                          259      ; FUNCTION: BMWRIT - WRITE BUBBLE MEMORY DATA.
                          260      ; INPUTS: CX = # OF BYTES TO WRITE.
                          261      ; OUTPUTS: A = STATUS; F/F(C=1:ERROR OCCURED), BX=# OF BYTES WRITTEN.
                          262      ; CALLS: SNDREG,BMWAIT.
                          263      ; DESTROYS: ALL.
                          264      ; DESCRIPTION: THIS PROCEDURE PERFORMS A BUBBLE MEMORY WRITE OPERATION.
                          265      ;              AN ERROR WILL OCCUR IF THE NUMBER OF BYTES GIVEN FOR THE
                          266      ;              WRITE OPERATION EXCEED THE NUMBER THAT THE BMC EXPECTS
                          267      ;              (DERIVED FROM COMMAND, BLOCK LENGTH AND NUMBER OF FSA
                          268      ;               CHANNELS), OR IF THE NUMBER OF BYTES IS LESS THAN THAT
                          269      ;              WHICH THE BMC EXPECTS.
                          270      ;
0067                      271      BMWRIT:
0067 32C0                 272              XOR     AL,AL                   ; A = 0
0069 A20000          E    273              MOV     STATUS,AL               ; CLEAR STATUS
006C 8BD9                 274              MOV     BX,CX
006E B01D                 275              MOV     AL,CFR
0070 E6E1                 276              OUT     BMSTAT.AL               ; FIFO RESET
0072 E86500               277              CALL    SNDREG                  ; SEND REGISTERS TO BMC.
0075 8B360000        E    278              MOV     SI.BUFADR               ; SET UP SRC BFR PTR (IN DATA SEG)
0079 A00000          E    279              MOV     AL,BMCMD                ; GET COMMAND
007C E6E1                 280              OUT     BMSTAT,AL               ; ISSUE IT.
007E                      281      WRIT01:
007E E4E1                 282              IN      AL,BMSTAT
0080 A880                 283              TEST    AL,BUSYBT               ; WAIT FOR BUSY...
0082 74FA                 284              JZ      WRIT01
0084 A801                 285              TEST    AL,FIFOBT               ;   AND FIFO READY
0086 74F6                 286              JZ      WRIT01
                          287      ;
                          288      ; KEEP STUFFING DATA INTO FIFO UNTIL DONE OR AN ERROR OCCURS.
                          289      ; (NOTE: BMC GOING NOT BUSY IS AN ERROR).
                          290      ;
0088                      291      WRIT03:
0088 E4E1                 292              IN      AL,BMSTAT               ; GET STATUS
008A A801                 293              TEST    AL,FIFOBT               ; FIFO READY?
008C 7407                 294              JZ      WRIT04                  ; NO. WAIT FOR IT
008E AC                   295              LODSB                           ; YES, GET DATA FOR IT
008F E6E0                 296              OUT     BMDATA,AL               ; GIVE IT TO BMC
0091 E2F5                 297              LOOP    WRIT03                  ; LOOP UNTIL DONE.
0093 EB84                 298              JMP     BMWAIT                  ; XFER DONE, WAIT FOR A GOOD STATUS
0095                      299      WRIT04:
0095 A880                 300              TEST    AL,BUSYBT
0097 75EF                 301              JNZ     WRIT03                  ; OK IF STILL BUSY
0099 2BD9                 302              SUB     BX,CX                   ; BX <- # OF BYTES XFERED
009B E976FF               303              JMP     CTRL99                  ; ERROR IF NOT BUSY AND CX NOT ZERO
                          304      ;
                          305      ; SPECIAL WRITE FOR BOOTLOOP AND BOOTLOOP REG CMNDS
                          306      ;
009E                      307      BMWRTB:
009E 32C0                 308              XOR     AL,AL                   ; A = 0
00A0 A20000          E    309              MOV     STATUS,AL               ; CLEAR STATUS
00A3 8BD9                 310              MOV     BX,CX
00A5 B01D                 311              MOV     AL,CFR
```

```
LOC  OBJ                  LINE     SOURCE

00A7 E6E1                  312              OUT     BMSTAT.AL               ; FIFO RESET
00A9 E82E00                313              CALL    SNDREG                  ; SEND REGISTERS TO BMC.
00AC 8B360000        E     314              MOV     SI.BUFADR               ; SET UP SRC BFR PTR (IN DATA SEG)
                           315      ;
                           316      ; FILL FIFO WITH 20/40/41 BYTES
                           317      ;
00B0                       318      WRTB01:
00B0 AC                    319              LODSB
00B1 E6E0                  320              OUT     BMDATA,AL               ; STICK IN FIFO.
00B3 E2FB                  321              LOOP    WRTB01                  ; LOOP UNTIL FILL COUNT=0.
00B5 A00000          E     322              MOV     AL,BMCMD
00B8 E6E1                  323              OUT     BMSTAT,AL               ; SEND CMND
00BA E95CFF                324              JMP     BMWAIT
                           325 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          326      ;*******************************
                          327      ;
                          328      ; FUNCTION: ZAPREG - ZAP ALL INTERNAL REGISTERS.
                          329      ; INPUTS: NONE
                          330      ; OUTPUTS: NONE
                          331      ; CALLS: NOTHING
                          332      ; DESTROYS: NOTHING.
                          333      ; DESCRIPTION: SET ALL INTERNAL REGISTERS EXCEPT 'ENABLE' TO AN
                          334      ;              ACCEPTABLE VALUE.  NOTE:  AN ACCEPTABLE VALUE MAY
                          335      ;              OR MAY NOT BE THE ONE DESIRED AS A DEFAULT.
                          336      ;
00BD                      337      ZAPREG:
00BD 9C                   338              PUSHF                           ; SAVE FLAGS
00BE 50                   339              PUSH    AX                      ; SAVE REGISTERS
00BF 53                   340              PUSH    BX
00C0 BB0000               341              MOV     BX,0
00C3 891E0000         E   342              MOV     PAGENO,BX               ; STARTING PAGE NUMBER = 0
00C7 43                   343              INC     BX
00C8 891E0000         E   344              MOV     BLKLEN,BX               ; BLOCK LENGTH = 1
00CC 32C0                 345              XOR     AL,AL
00CE A20000           E   346              MOV     BBLNUM.AL               ; BUBBLE NUMBER = 0
00D1 FEC0                 347              INC     AL
00D3 A20000           E   348              MOV     NFC.AL                  ; # OF FSA CHANNELS = 1 (2 CHANNELS)
00D6 5B                   349              POP     BX                      ; RESTORE REGISTERS.
00D7 58                   350              POP     AX
00D8 9D                   351              POPF
00D9 C3                   352              RET
                          353 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           354     ;*******************************
                           355     ;
                           356     ; FUNCTION: SNDREG - FORMAT AND SEND INTERNAL REGISTERS TO BMC.
                           357     ; INPUTS: NONE
                           358     ; OUTPUTS: NONE
                           359     ; DESTROYS: NOTHING.
                           360     ; DESCRIPTION: FORMAT AND SEND ALL INTERNAL REGISTERS TO THE BMC.
                           361     ;
00DA                       362     SNDREG:
00DA 9C                    363             PUSHF
00DB 50                    364             PUSH    AX                      ; SAVE REGISTERS
00DC 53                    365             PUSH    BX
00DD 51                    366             PUSH    CX
00DE B00B                  367             MOV     AL,REG1                 ; GET FIRST REGISTER ADDRESS.
00E0 E6E1                  368             OUT     BMSTAT,AL               ; SELECT IT.
                           369     ;
                           370     ; CONSTRUCT AND SEND BLOCK LENGTH.
                           371     ;
00E2 8B1E0000        E     372             MOV     BX,BLKLEN               ; HL = BLOCK LENGTH
00E6 8AC3                  373             MOV     AL,BL                   ; A  = BLOCK LENGTH LSB
00E8 E6E0                  374             OUT     BMDATA,AL               ; GIVE IT TO BMC.
00EA A00000          E     375             MOV     AL,NFC                  ; A  = NUMBER OF FSA CHANNELS.
00ED B104                  376             MOV     CL,4
00EF D2E0                  377             SHL     AL,CL
00F1 0AC7                  378             OR      AL,BH                   ; MERGE INTO BLOCK MSB
00F3 E6E0                  379             OUT     BMDATA,AL               ; GIVE IT TO BMC.
                           380     ;
                           381     ; SEND ENABLE BYTE.
                           382     ;
00F5 A00000          E     383             MOV     AL,ENABLE               ; GET ENABLE BYTE
00F8 E6E0                  384             OUT     BMDATA,AL               ; GIVE IT TO BMC
                           385     ;
                           386     ; CONSTRUCT AND SEND ADDRESS REGISTER.
                           387     ;
00FA 8B1E0000        E     388             MOV     BX,PAGENO               ; HL = STARTING PAGE NUMBER
00FE 8AC3                  389             MOV     AL,BL                   ; A  = ADDRESS REGISTER LSB
0100 E6E0                  390             OUT     BMDATA,AL               ; GIVE IT TO BMC.
0102 A00000          E     391             MOV     AL,BBLNUM               ; A  = BUBBLE NUMBER
0105 B103                  392             MOV     CL,3
0107 D2E0                  393             SHL     AL.CL
0109 0AC7                  394             OR      AL,BH                   ; MERGE INTO PAGE NUMBER MSB.
010B E6E0                  395             OUT     BMDATA,AL               ; GIVE IT TO BMC.
                           396     ;
                           397     ; RESTORE REGISTERS AND RETURN.
                           398     ;
010D 59                    399             POP     CX
010E 5B                    400             POP     BX
010F 58                    401             POP     AX
0110 9D                    402             POPF
0111 C3                    403             RET
                           404 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           405     CODE    ENDS
                           406             END
```

XREF SYMBOL TABLE LISTING
---- ------ ----- -------

```
NAME      TYPE     VALUE  ATTRIBUTES, XREFS

??SEG . . SEGMENT        SIZE=0000H PARA PUBLIC
BBLNUM. . V BYTE   0000H  EXTRN  16# 346 391
BLKLEN. . V WORD   0000H  EXTRN  15# 344 372
BMCMD . . V BYTE   0000H  EXTRN  16# 167 229 279 322
BMCTRL. . L NEAR   0000H  CODE PUBLIC  131 165#
BMDATA. . NUMBER   00E0H    54# 247 296 320 374 379 384 390 395
BMRD1 . . L NEAR   0048H  CODE  233# 236
BMRD2 . . L NEAR   0052H  CODE  243# 249 253
BMRD3 . . L NEAR   005FH  CODE  246 251#
BMREAD. . L NEAR   002EH  CODE PUBLIC  131 221#
BMSTAK. . L NEAR   0000H  EXTRN  8#
BMSTAT. . NUMBER   00E1H    53# 168 197 202 230 234 244 276 280 282 292 312 323 368
BMWAIT. . L NEAR   0019H  CODE PUBLIC  131 169 193# 250 298 324
BMWRIT. . L NEAR   0067H  CODE PUBLIC  131 271#
BMWRTB. . L NEAR   009EH  CODE PUBLIC  131 307#
BPADR . . V WORD   0000H  EXTRN  23#
BUFADR. . V WORD   0000H  EXTRN  15# 226 278 314
BUSYBT. . NUMBER   0080H    65# 198 203 235 252 283 300
CAB . . . NUMBER   0019H    43#
CFR . . . NUMBER   001DH    47# 275 311
CIZ . . . NUMBER   0011H    35#
CODE. . . SEGMENT        SIZE=0112H PARA PUBLIC   78# 79 405
CORERR. . NUMBER   0008H    61#
CPURG . . NUMBER   001EH    48#
CRB . . . NUMBER   001BH    45#
CRBR. . . NUMBER   0015H    39#
CRCD. . . NUMBER   001CH    46#
CRD . . . NUMBER   0012H    36#
CRFS. . . NUMBER   0018H    42#
CRS . . . NUMBER   0014H    38#
CSR . . . NUMBER   001FH    49#
CTRL99. . L NEAR   0014H  CODE  172 179# 205 237 255 303
CWB . . . NUMBER   0017H    41#
CWBR. . . NUMBER   0016H    40#
CWBRM . . NUMBER   0010H    34#
CWD_. . . NUMBER   0013H    37#
CWRS. . . NUMBER   001AH    44#
DATA. . . SEGMENT        SIZE=0000H PARA PUBLIC   11# 28 79
DBGMOD. . NUMBER   0080H    153#
DEFADR. . V WORD   0000H  EXTRN  13#
DEFBLK. . V WORD   0000H  EXTRN  14#
DEFBUB. . V BYTE   0000H  EXTRN  13#
DEFENA. . V BYTE   0000H  EXTRN  13#
DEFMOD. . V BYTE   0000H  EXTRN  14#
DEFNFC. . V BYTE   0000H  EXTRN  13#
DEFPAG. . V WORD   0000H  EXTRN  14#
DMAENA. . NUMBER   0004H    71#
DMAMOD. . NUMBER   0002H    152#
ENABLE. . V BYTE   0000H  EXTRN  15# 383
FIFOBT. . NUMBER   0001H    58# 245 285 293
ICDENA. . NUMBER   0040H    75#
```

```
NAME      TYPE     VALUE  ATTRIBUTES, XREFS

IERENA. . NUMBER   0002H    70#
INBUF . . V BYTE   0000H  EXTRN  17#
INBUFA. . V WORD   0000H  EXTRN  18#
INBUFC. . V BYTE   0000H  EXTRN  17#
INBUFL. . V BYTE   0000H  EXTRN  18#
INBUFP. . V WORD   0000H  EXTRN  17#
INTENA. . NUMBER   0001H    69#
INTMOD. . NUMBER   0001H   151#
LEVMSK. . V BYTE   0000H  EXTRN  22#
MODE. . . V BYTE   0000H  EXTRN  16#
MYBUF . . V BYTE   0000H  EXTRN  12#
NFC . . . V BYTE   0000H  EXTRN  16# 348 375
OPDONE. . NUMBER   0040H    64#
OPFAIL. . NUMBER   0020H    63#
OUTBFA. . V WORD   0000H  EXTRN  20#
OUTBFC. . V BYTE   0000H  EXTRN  19#
OUTBFL. . V BYTE   0000H  EXTRN  20#
OUTBFP. . V WORD   0000H  EXTRN  19#
OUTBUF. . V BYTE   0000H  EXTRN  19#
PAGENO. . V WORD   0000H  EXTRN  15# 342 388
PARERR. . NUMBER   0002H    59#
POPREGS . V WORD   0000H  EXTRN  24#
PROMPT. . V BYTE   0000H  EXTRN  22#
PUSHREGS. V WORD   0000H  EXTRN  24#
RAM . . . V BYTE   0000H  EXTRN  12#
RCDENA. . NUMBER   0020H    74#
RDLEN . . V WORD   0000H  EXTRN  21#
REG1. . . NUMBER   000BH   137# 367
RSVD1 . . NUMBER   0008H    72#
RSVD2 . . NUMBER   0080H    76#
SCRBUF. . V BYTE   0000H  EXTRN  12#
SNDREG. . L NEAR   00DAH  CODE  166 225 277 313 362#
STACK . . SEGMENT         SIZE=0000H PARA STACK
STATER. . NUMBER   003CH   138# 170
STATUS. . V BYTE   0000H  EXTRN  16# 171 180 207 223 273 309
TIMERR. . NUMBER   0010H    62#
UNCERR. . NUMBER   0004H    60#
USERBP. . V WORD   0000H  EXTRN  25#
USERBX. . V WORD   0000H  EXTRN  25#
USERCS. . V WORD   0000H  EXTRN  26#
USERDS. . V WORD   0000H  EXTRN  25#
USERFL. . V WORD   0000H  EXTRN  26#
USERIP. . V WORD   0000H  EXTRN  26#
USERPC. . V WORD   0000H  EXTRN  27#
USERRG. . V WORD   0000H  EXTRN  23#
USERSP. . V WORD   0000H  EXTRN  26#
USERSS. . V WORD   0000H  EXTRN  25#
WAITEX. . L NEAR   002AH  CODE  199 206#
WAITPO. . L NEAR   0022H  CODE  201# 204
WBLENA. . NUMBER   0010H    73#
WRIT01. . L NEAR   007EH  CODE  281# 284 286
WRIT03. . L NEAR   0088H  CODE  291# 297 301
WRIT04. . L NEAR   0095H  CODE  294 299#
WRLEN . . V WORD   0000H  EXTRN  21#
WRTB01. . L NEAR   00B0H  CODE  318# 321
```

```
NAME        TYPE    VALUE   ATTRIBUTES, XREFS
ZAPREG. . . L NEAR  00BDH   CODE PUBLIC  131 337#

ASSEMBLY COMPLETE. NO ERRORS FOUND
```