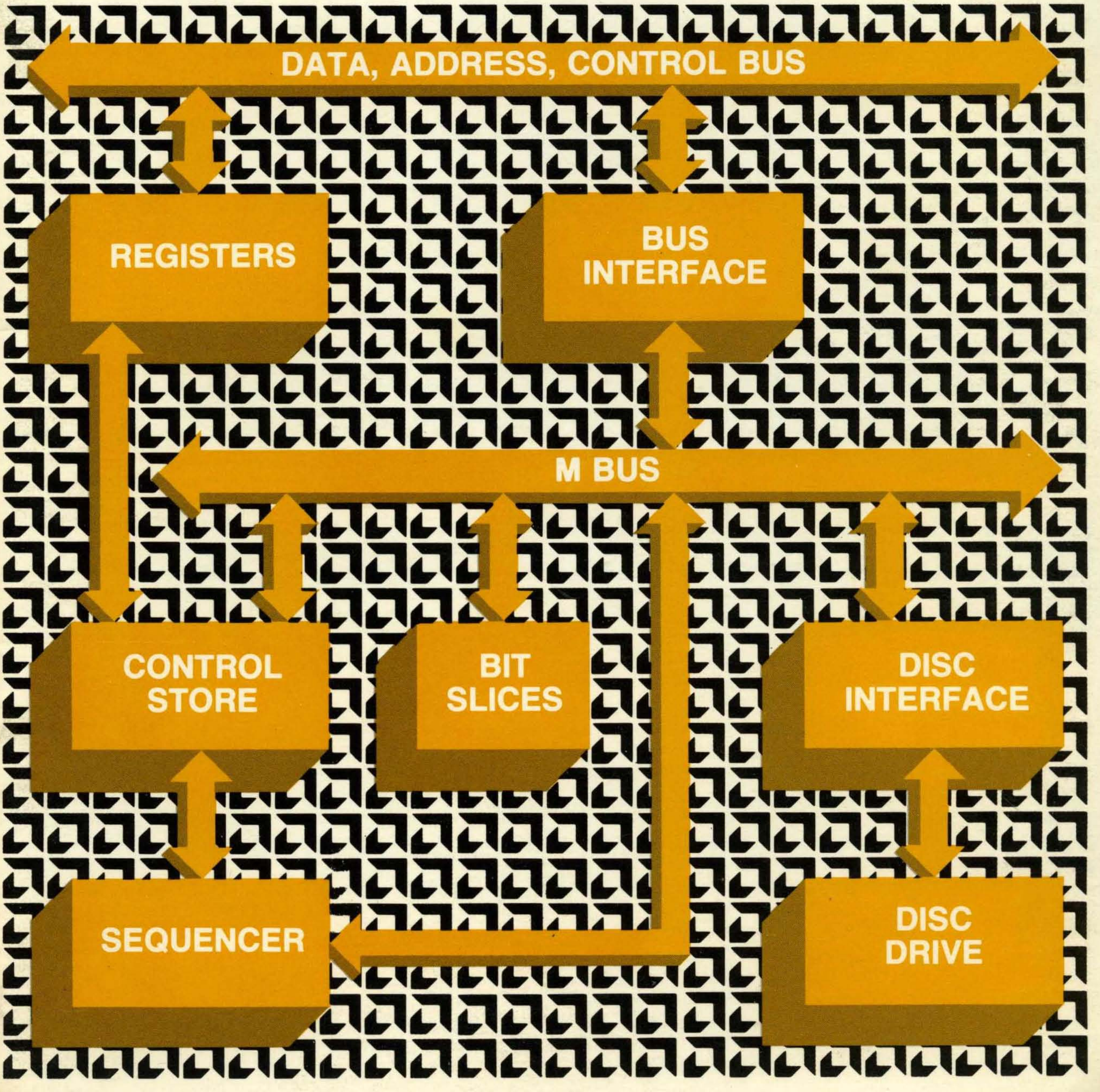
Advanced Micro Devices

# A High Performance Disc Controller

$5.00



# A HIGH PERFORMANCE DISC CONTROLLER

## INTRODUCTION

The Am2901A Four-Bit Bipolar Microprocessor Slice, a significant advance in the state-of-the-art technology in Low-Power Schottky Integrated Circuits, enables the Design Engineer to implement new systems with higher logic density, better cost-effectiveness, and improved product versatility. The higher logic density and better cost-effectiveness of microprocessor-based designs is well-known and will not be discussed here. This application note, describing a Pertec D3441 Disc Controller for the Digital Equipment Corporation (DEC) PDP-11, will demonstrate how improved product versatility can be achieved by employing the Am2901A in the design of a peripheral controller.

This disc controller design is not intended to be an example of a minimal logic, cost-effective controller only one step away from the marketplace. Instead, think of it as the grandfather. Its large, writeable microprogram control store and its generalized disc and UNIBUS interface make it suitable to be the prototype for a family of disc controllers. Individual controllers would use ROM's of the appropriate size for the control store, and the disc interface would be tailored to a particular disc drive.

## THE DISC CONTROLLER

A major advantage of designing with microprocessors is that the designs tend to be highly structured and therefore much easier to comprehend. Referring to Figure 1, notice that the disc controller is composed of a small number of well-defined sub-sections. Each sub-section will be discussed in detail and then the interaction between sub-sections will be described. The reader will find that the individual sub-sections are easy to understand because each one has a limited but well-defined role in the disc controller.

## THE MICROPROCESSOR

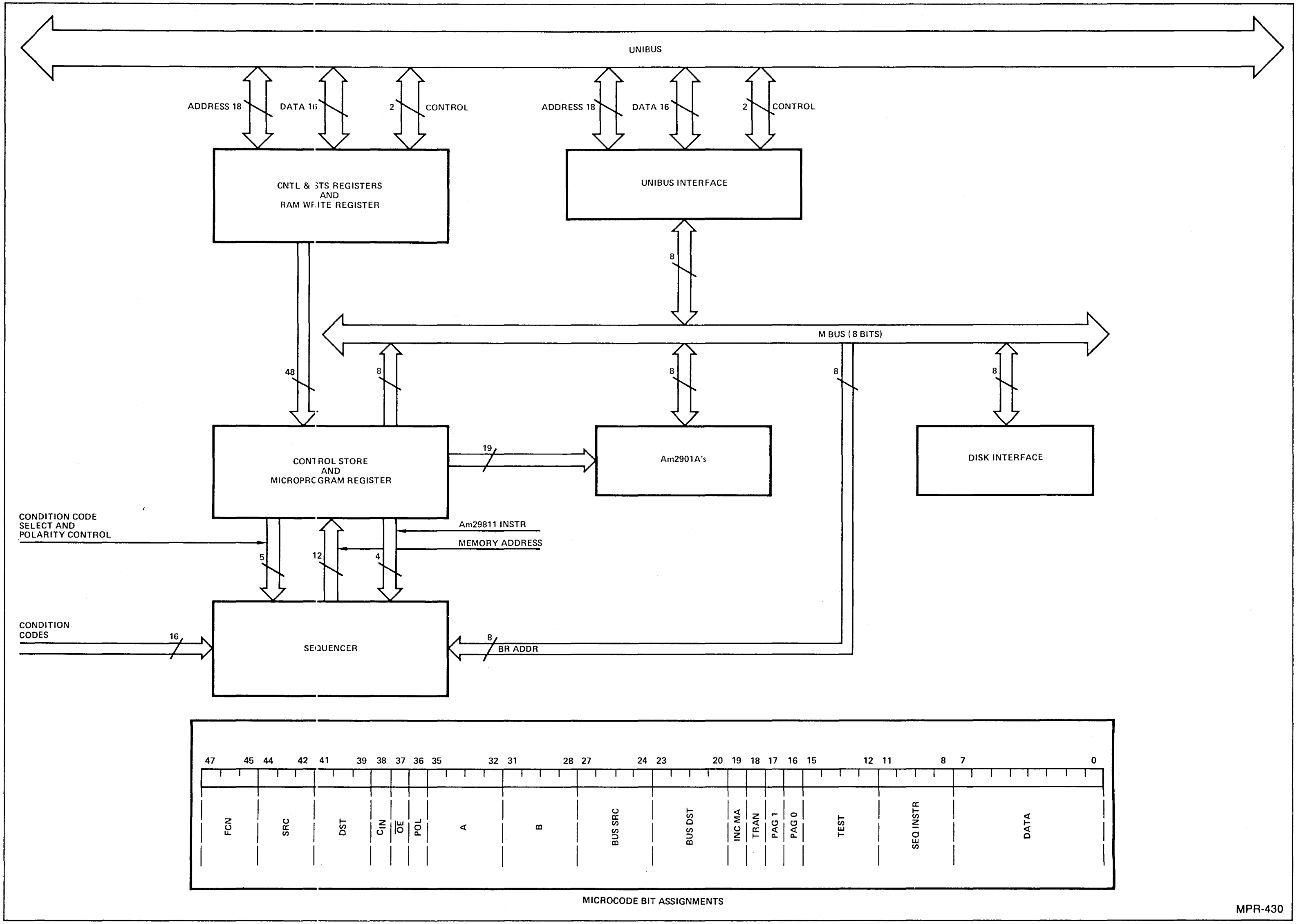
The microprocessor, 8 bits wide using two Am2901A's, provides the disc controller with an arithmetic and logic capability. In this application, the arithmetic capabilities of the Am2901A are not taxed. Mainly, they are used to generate checksums on disc reads. The principal role of the microprocessor in this design is that of a logic processor. As the reader will discover further on, both the DEC UNIBUS interface and the disc interface are very general purpose. It is the logic processing power of the Am2901A, coupled with the control information of the microprogram, that enables the disc controller to completely emulate the RK11 disc controller (SSI TTL controller from DEC). If the disc controller is considered as a state machine, at any given instance, the current state of the machine is to a large degree defined by the contents of the microprogram register. When an unexpected state is encountered, the logic processing power of the microprocessor enables it to exercise more control over the selection of the next state to enter. In the disc controller, this is evidenced more through error recovery procedures. All recoverable errors can be handled by the disc controller without the intervention of the host computer. In addition to supplying logic processing power, the microprocessor also provides seventeen high-speed, 8-bit temporary storage registers. Most of these registers are assigned specific functions. In this application, twelve registers were used to build six 16-bit registers. These registers contain the disc address, memory address, transfer word count, control and status information, error information, and the checksum. Of the remaining five registers, four are utility registers that are employed as needed, and the fifth is the Q register which can be used to store and retrieve 8-bit values.

Figure 2, depicting the two Am2901A's, shows that the microprocessor interface to the other sub-sections is very simple. The 8-bit bidirectional M bus (microprocessor bus) enables the microprocessor to input/output data from/to the other subsections of the disc controller. Four condition lines (ZERO, MINUS, OVRFL, and CARRY) communicate the results of logic and arithmetic operations to the sequencer, which may select one of these lines to determine the address of the next microinstruction. Notice that since the condition lines are latched, the sequencer is always looking at the conditions of the previous microinstruction. On each clock cycle, the Am2901A's are presented with a 19-bit instruction from the microprogram register. This 19-bit instruction consists of a 9-bit microinstruction decode, an 8-bit register select, the carry-in, and the output enable (see Figure 3). By the end of the clock cycle, the specified arithmetic or logic operation has been performed, the result has been stored, and the condition codes have been latched. The microprocessor is now ready to perform the next instruction.

## THE SEQUENCER

A microinstruction usually has two primary parts. These are: (1) the definition and control of all elemental micro-operations to be performed, and (2) the definition of the address of the next microinstruction to be executed. Referring back to the consideration of the disc controller as a state machine, it is evident that the controller's ability to perform any useful function is dependent on its ability to progress from state to state in a controller manner. It is the task of the sequencer to provide control over the transitions from state to state.

In order to provide this control, some feedback from various system components is necessary. For example, when reading a word from PDP-11 main memory, the controller must first request the UNIBUS by asserting NPR (non-processor request). The controller then enters a waiting state and the sequencer will keep the controller in this state until the signal NPR RDY informs the sequencer that the UNIBUS is now available for the transfer. At this time, the sequencer will transition the controller into the next state which would start driving the address onto the UNIBUS and assert MSYN (master sync). The sequencer designed for this controller (see Figure 2) provides for up to sixteen different input condi-



UNIBUS
ADDRESS 18
DATA 16
2
CONTROL
ADDRESS 18
DATA 16
2
CONTROL
CNTL & STS REGISTERS AND RAM WRITE REGISTER
UNIBUS INTERFACE
8
M BUS (8 BITS)
48
8
8
8
8
CONTROL STORE AND MICROPROGRAM REGISTER
19
Am2901A's
DISK INTERFACE
CONDITION CODE SELECT AND POLARITY CONTROL
Am29811 INSTR
MEMORY ADDRESS
5
12
4
CONDITION CODES
16
SEQUENCER
8
BR ADDR
47 45 44 42 41 39 38 37 36 35 32 31 28 27 24 23 20 19 18 17 16 15 12 11 8 7 0
FCN
SRC
DST
$C_{IN}$
$\overline{OE}$
POL
A
B
BUS SRC
BUS DST
INC MA
TRAN
PAG 1
PAG 0
TEST
SEQ INSTR
DATA
MICROCODE BIT ASSIGNMENTS
MPR-430


**Figure 1.**

| 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DESTINATION CONTROL | | | ALU FUNCTION | | | ALU SOURCE | | | A REGISTER ADDRESS | | | | B REGISTER ADDRESS | | | | CARRY IN | OUTPUT ENABLE |
| MICROINSTRUCTION DECODE | | | | | | | | | REGISTER SELECT | | | | | | | | | |

Figure 3.

tions. On each microcycle, four bits from the microprogram register will select one of the sixteen input conditions. The selected condition is XOR'ed with another bit from the microprogram register to provide polarity control over the selected conditions as it is inputted to the Am29811.

The Am29811, the next address control unit, can execute sixteen different next address control functions, most of which are conditional. Thus, the device requires four instruction inputs as well as the condition code test input. The four instruction inputs come from a multiplexer that normally selects the Am29811 instruction specified in the microprogram register. However, when the writeable control store is being loaded, the multiplexer selects the other input, which forces the Am29811 to execute JUMP ZERO on the first write cycle and CONTINUE on all following write cycles.

The outputs of the Am29811 are used to control the stack pointer and the next address multiplexer of the three Am2911's. These three Am2911's are cascaded to form the 12-bit microprogram sequencer.[1] The Am2911's can select an address from any of three sources. They are: (1) external data from the D inputs, stored in an internal register; (2) a four-word deep push/pop stack; or (3) a program counter register (which usually contains the last address plus one). The push/pop stack includes certain control lines so that it can efficiently execute nested subroutine linkages. The internal register that is loaded from the M bus appears to the rest of the system as just another M bus destination. At the end of a bus cycle, if the two low-order Am2911's or the high-order Am2911 has been selected as the M bus destination, the selected Am2911's register enable will be strobed to clock in the data on the M bus. Once the internal register is loaded, it can be selected on any following microinstruction as the source of the next address.

**THE CONTROL STORE**

The output of the microprogram sequencer is a 12-bit address that selects the next microinstruction to be fetched from the control store. At the beginning of each microcycle, the output of the control store is strobed into the microprogram register. Since this register holds the microinstruction while it is being executed, the memory is free to fetch the next microinstruction as soon as the sequencer can determine the address of the next instruction. This technique, referred to as pipelining, allows the fetching of the next microinstruction to be overlapped with the execution of the current microinstruction.

The disc controller's control store, 48 bits wide by 1K deep, is comprised of twelve Am9130's (see Figure 4). The Am9130 is a high-performance, low-power, 4096-bit, static, read/write memory organized as 1024 words by 4 bits per word. The data input and output signals are bussed together and share common I/O pins.

The microprogram register is comprised of six 8-bit registers. The low-order register holds the data portion of each microinstruction. This register, an Am25LS374, has three-state outputs and when selected as a bus source, it will drive the data onto the M bus. The other five registers are Am25LS273's, which consist of D-type flip-flops with a common clock and a common clear.

Normally, the control store is clocked by the microprocessor clock ($\mu$PCLK). However, when the control store is being loaded by the PDP-11, it is clocked every time a 48-bit word, assembled in the RAM Write Register, is ready to be written into the control store. When a millisecond has passed without a RAM write cycle, a one-shot times out (the signal LD MCODE is no longer asserted), and the control store is once again clocked by $\mu$PCLK. While LD MCODE was asserted, the clear input to the microprogram register was also asserted and the output of the Am9130's was disabled.

**THE CONTROL AND STATUS REGISTERS**

To provide for communication between the PDP-11 CPU and the disc controller, sixteen 16-bit registers have been interfaced to the UNIBUS (see Figure 5). Except for the fact that the last two registers play a special role in loading the control store (determining the address of these registers on the UNIBUS) and in selecting the frequency of the $\mu$PCLK, these registers are just memory locations. Indeed, core memory locations could be used for the control and status registers. The only disadvantage to doing this would be that the controller would not be compatible with existing software.

The disc controller uses the same procedure for reading or writing the control and status registers as it does when reading or writing in main memory. This approach has the advantage of using the UNIBUS arbitration logic to solve the problem of both the CPU and the controller accessing the same control and status register at the same time.

Since the control and status registers are just memory locations, the definition of what each group of bits means is totally determined by the microprogram. As the same controller is used to interface different types of disc drives, the microcode can define the control and status registers to be compatible with whatever PDP-11 disc system is to be emulated.

As was mentioned earlier, the last two registers are special. When data is written into the last register, it is also stored in one of the RAM WRITE REGISTERS. Which register is selected is determined by a 2-bit counter that is incremented after each write. Every fourth write is a signal that 48 bits have been accumulated in the RAM WRITE REGISTER and it is time for the control store to perform a write cycle. Example 1 is a listing of PDP-11 code that would load the control store from a 3K word buffer in main memory.

```
;LOADCS is entered with R0 a pointer to the buffer
;and R1 a pointer to the last device register (160016).
LOADCS:   RESET              ;initialize 2-bit counter and
                             ;cause LDMCODE to be asserted
LOOP:     MOV (R0)+, (R1)    ;load 48-bit RAM
          MOV (R0)+, (R1)    ;WRITE REGISTER
          MOV (R0)+, (R1)
          CLR (R1)           ;phony write to cause control store write
          CMP R0, #BUFEND    ;condition : has all of the buffer been copied?
          BLO LOOP           ;if no, then branch
          RTS PC             ;if yes, then return
```

Example 1. PDP-11 Code to Load Control Store.

Whenever the second to last register is written, the data is also stored in a 16-bit internal register. The high-order byte is used to set the UNIBUS address of the control and status registers. Initially, the base device register address was $160000_8$, because the INIT pulse on the UNIBUS (caused by power-up or the RESET instruction) cleared the 16-bit internal register. It is up to the PDP-11 to keep track of the current address of the control and status registers as they are moved about. Also, the PDP-11 must somehow let the controller know where its registers are. Usually, this information is contained in the microcode. This ability to change the address of the device registers allows the controller to attempt to emulate just about whatever it wants to emulate.

The low-order four bits of this internal register can be set by the PDP-11 to select 1-of-16 microprocessor clock rates. It is not clear that this is very useful, but in a general purpose prototyping design, why not?

### THE UNIBUS INTERFACE

The UNIBUS interface consists of two main parts: (1) the transceivers for the address, data, and control lines; and (2) the handshaking logic required to control UNIBUS transactions.[2]

Figure 6, depicting the address, data, and control line transceivers, illustrates that the microprocessor communicates with the transceivers via registers which can act as either sources or destinations for the M bus. The registers for the address line transceivers (in this case used only as line drivers) are synchronous 4-bit counters (Am25LS161). In a DMA transfer, the starting address would be initially loaded into the Am25LS161's in two M bus cycles. On the first cycle, the low-order byte of the address register would be loaded. The second cycle would load the high byte. Once the memory address register has been initialized to the transfer starting address, it can be incremented to successive memory locations at the end of each transfer by the assertion of INC MA. The output of the address register is shifted one bit position as it is fed into the UNIBUS drivers to compensate for the fact that each byte has a unique address in the PDP-11, and the controller only addresses word locations.

Am2907's are used as the transceivers for the UNIBUS data lines. Internal to the Am2907's are the data input and the data output registers. On a UNIBUS read cycle, data is strobed into the data input register from the UNIBUS when SSYN (Slave Sync) is received. The data is then available to the microprocessor via the M bus. On a UNIBUS write cycle, data is first loaded into the data output register via the M bus, and then the UNIBUS write transaction is initiated.

Another Am2907 is used for the control lines and the two high-order address lines. These control and address lines are initialized before the start of a DMA transfer. The control lines never need to be changed during a DMA burst. However, if the memory address register should overflow, the two high-order address bits will need to be updated before the next UNIBUS read or write transaction.

In addition to the address, data, and control lines, the UNIBUS has additional signals which provide synchronization for data transfers, allow control of the UNIBUS to be passed to any DMA controller, and provide an interrupt capability.

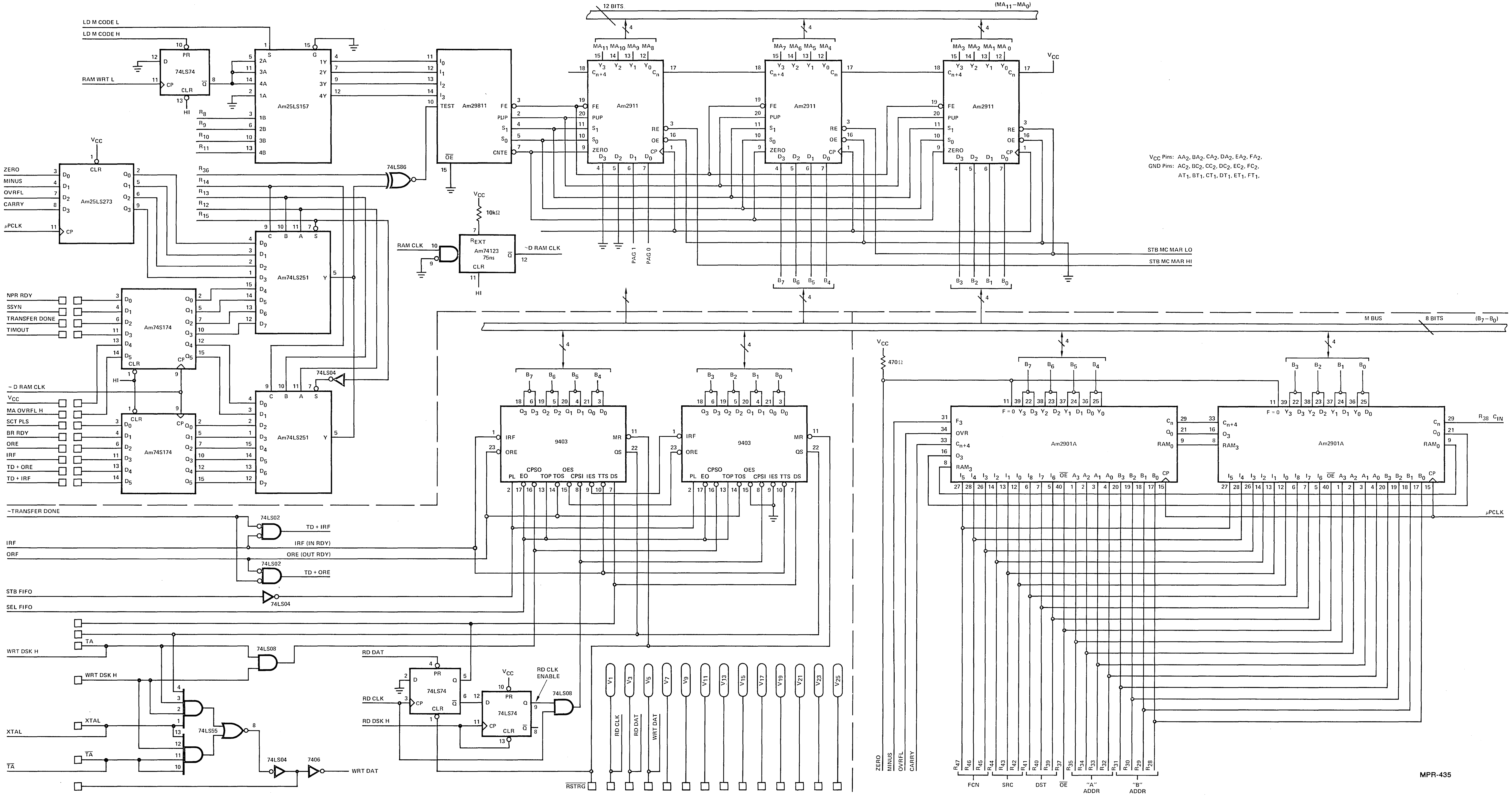
Figure 7 is the diagram of the UNIBUS handshaking logic. The microprocessor may request the UNIBUS by asserting NPR REQ or BR REQ, depending on whether the bus is being requested for a DMA transfer or an interrupt transaction. When the handshaking logic has gained control of the UNIBUS, the microprocessor will be informed by the assertion of either NPR RDY or BR RDY. For a read or write transaction, TRAN is asserted to initiate the data transfer. Coming to the microprocessor's aid once again, the handshaking logic will sequence through UNIBUS protocols and inform the microprocessor of the completion of the transfer by asserting TRANSFER DONE.

### THE DISC INTERFACE

The disc interface is comprised of a 24-bit parallel input port and a 24-bit parallel output port (Figure 6), and an 8-bit wide, 16-word deep FIFO (Figure 2). The input and output ports are "soft", in that the function of the individual bits are defined in the microcode. Since both ports are quite wide, almost any disc based on 2314 technology can be accommodated by the controller.

The input port receives status information and control signals from the disc drive. Status information generally includes the sector counter, the index and sector pulses, error conditions, and unit attention. Any control signals from the drive that are used to strobe data into registers should be received on a line with a wire-wrap pin. This allows for simple gating of the control signals to generate data strobes.

The output port transmits control information, such as cylinder address, head select, read and write enable, and unit select, to the disc drives.



LD M CODE L
LD M CODE H
RAM WRT L
74LS74
Am25LS157
Am29811
Am2911
Am2911
Am2911
12 BITS
(MA11–MA0)
VCC Pins: AA2, BA2, CA2, DA2, EA2, FA2.
GND Pins: AC2, BC2, CC2, DC2, EC2, FC2, AT1, BT1, CT1, DT1, ET1, FT1.
ZERO
MINUS
OVRFL
CARRY
μPCLK
Am25LS273
74LS86
Am74LS251
Am74LS251
10kΩ
RAM CLK
Am74123 75ns
~D RAM CLK
PAG 1
PAG 0
STB MC MAR LO
STB MC MAR HI
NPR RDY
SSYN
TRANSFER DONE
TIMOUT
Am74S174
Am74S174
~D RAM CLK
VCC
MA OVRFL H
SCT PLS
BR RDY
ORE
IRF
TD + ORE
TD + IRF
74LS04
M BUS
8 BITS
(B7–B0)
9403
9403
470Ω
Am2901A
Am2901A
~TRANSFER DONE
74LS02
TD + IRF
IRF (IN RDY)
ORE (OUT RDY)
TD + ORE
STB FIFO
SEL FIFO
74LS04
TA
WRT DSK H
WRT DSK H
XTAL
XTAL
TA
TA
74LS08
74LS55
74LS04
7406
WRT DAT
RD DAT
RD CLK
RD DSK H
74LS74
74LS74
RD CLK ENABLE
74LS08
RSTRG
RD CLK
RD DAT
WRT DAT
V1 V3 V5 V7 V9 V11 V13 V15 V17 V19 V21 V23 V25
ZERO MINUS OVRFL CARRY
R47 R46 R45 R44 R43 R42 R41 R40 R39 R37 R35 R34 R33 R32 R31 R30 R29 R28
FCN
SRC
DST
OE
"A" ADDR
"B" ADDR
R38 CIN
μPCLK
MPR-435


Figure 2.

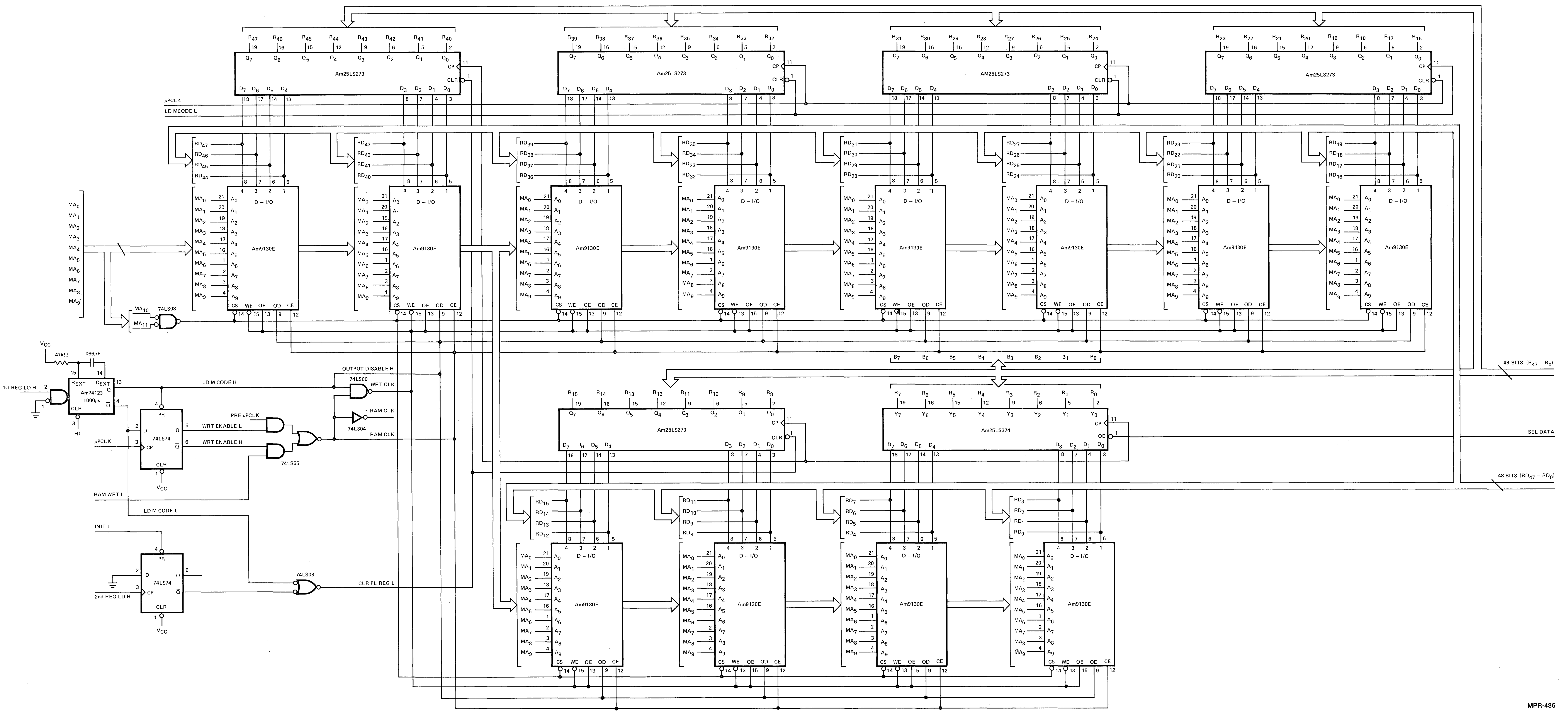
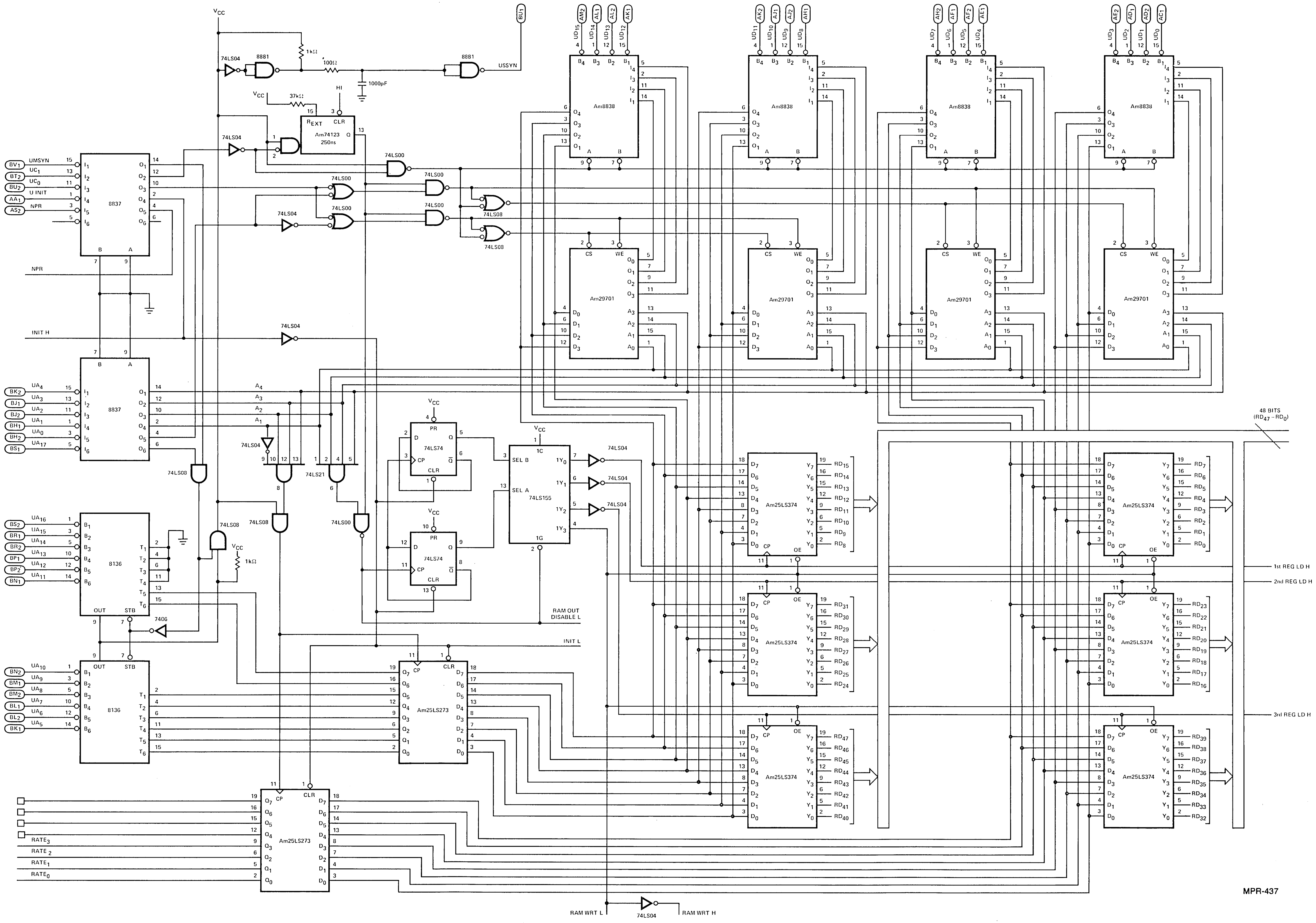


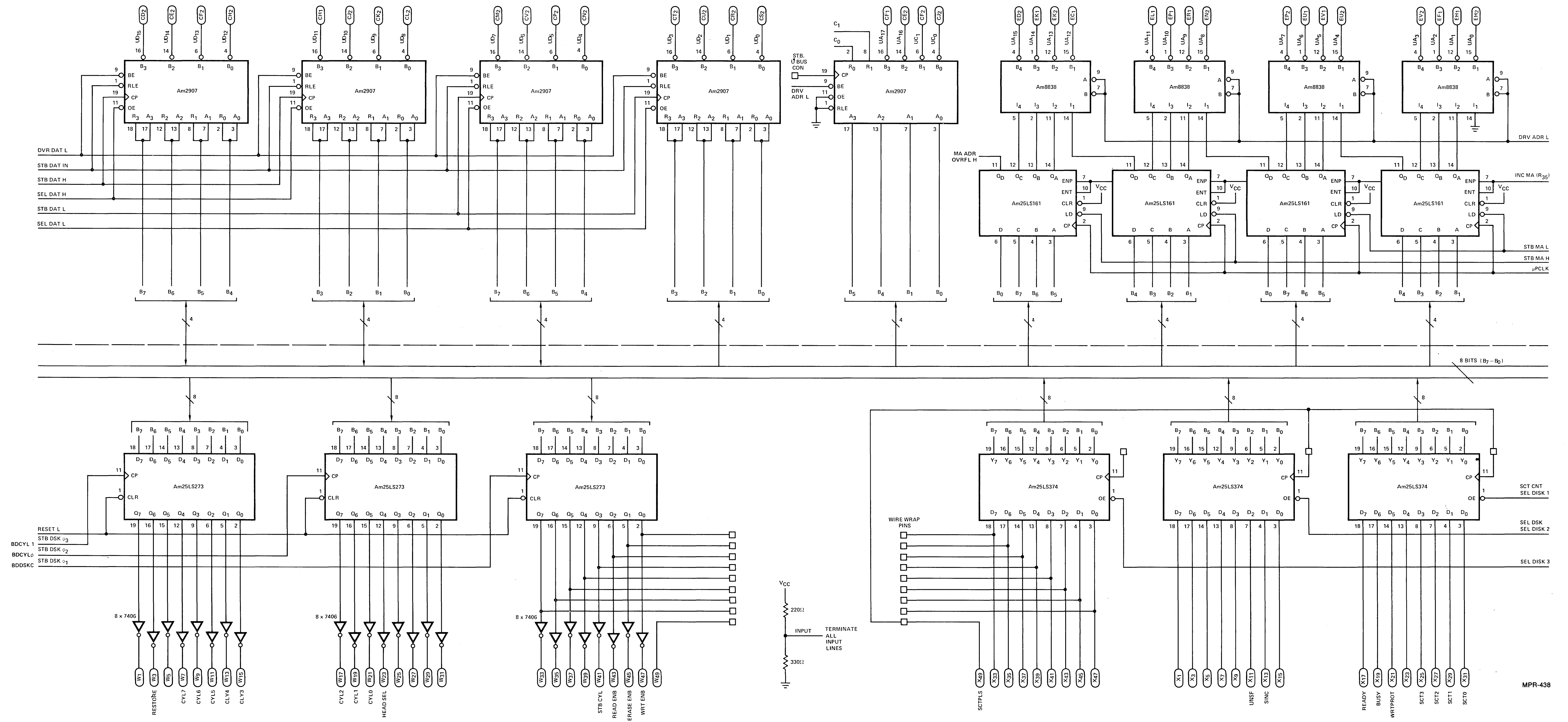

Figure 4.

MPR-436



MPR-437

Figure 5.



MPR-438

Figure 6.

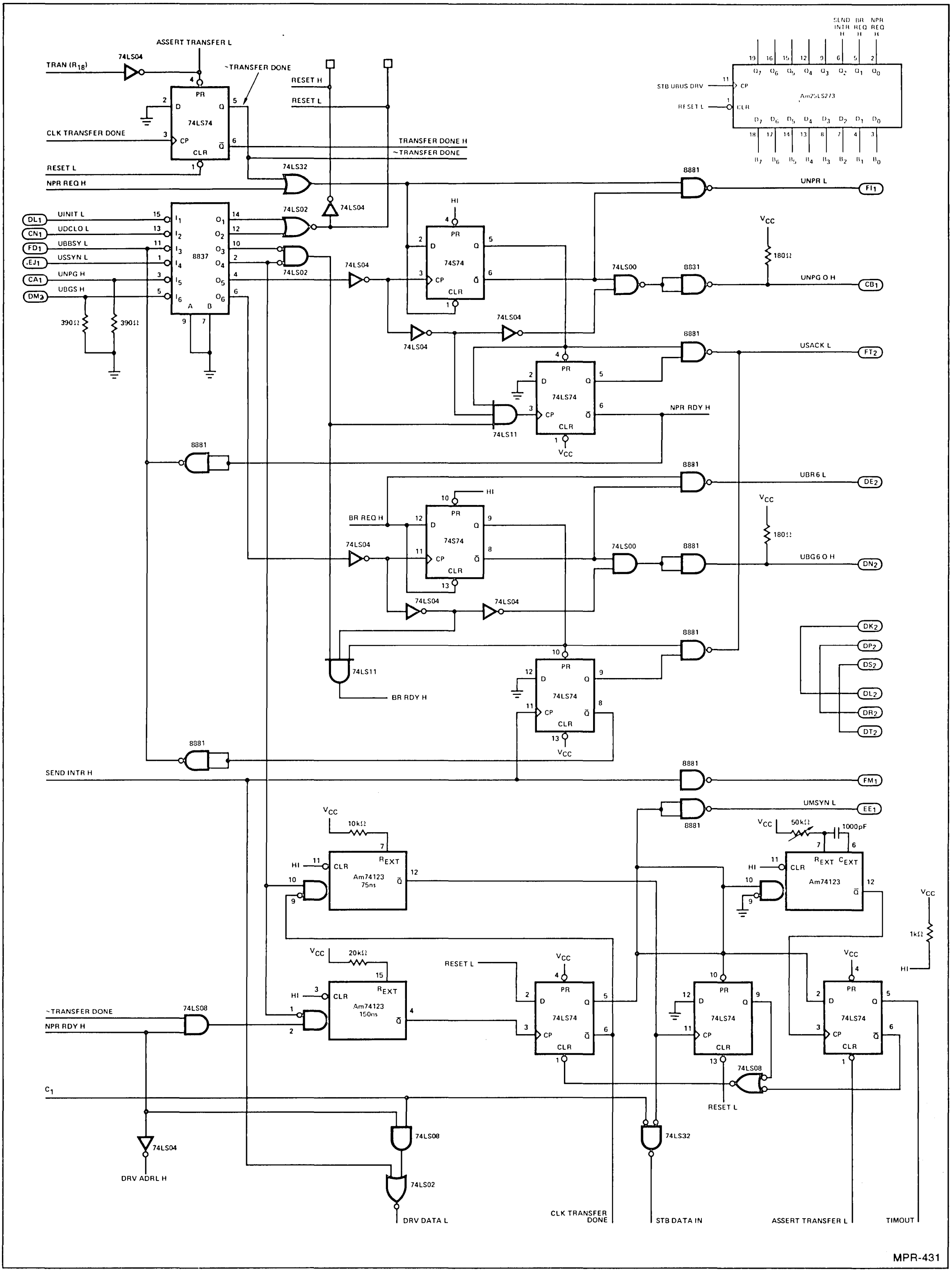

ASSERT TRANSFER L
TRAN (R18)
74LS04
~TRANSFER DONE
RESET H
RESET L
CLK TRANSFER DONE
74LS74
TRANSFER DONE H
~TRANSFER DONE
RESET L
NPR REQ H
74LS32
8881
UNPR L
FI1
UINIT L
UDCLO L
UBBSY L
USSYN L
UNPG H
UBGS H
DL1
CN1
FD1
EJ1
CA1
DM2
8837
74LS02
74S74
HI
VCC
180Ω
74LS00
8831
UNPG O H
CB1
390Ω
USACK L
FT2
74LS11
NPR RDY H
UBR6 L
DE2
BR REQ H
UBG6 O H
DN2
BR RDY H
DK2
DP2
DS2
DL2
DR2
DT2
SEND INTR H
FM1
UMSYN L
EE1
STB UBUS DRV
Am25LS273
SEND INTR H
BR REQ H
NPR REQ H
10kΩ
REXT
CEXT
CLR
Am74123
75ns
50kΩ
1000pF
20kΩ
150ns
1kΩ
~TRANSFER DONE
NPR RDY H
74LS08
C1
DRV ADRL H
DRV DATA L
CLK TRANSFER DONE
STB DATA IN
ASSERT TRANSFER L
TIMOUT
MPR-431

Figure 7.

The FIFO performs parallel-to-serial conversion on data that is being written on the disc and serial-to-parallel conversion on data that is read from the disc. When writing, the FIFO is clocked by a crystal oscillator at whatever frequency is required by the disc drive. However, when reading data from the disc, the FIFO is clocked by the RD CLK signal from the disc drive.

In addition to converting from parallel-to-serial and vice versa, the FIFO provides buffering between the controller and the disc drive. For example, before a disc write is initiated, the 16-word deep FIFO will have been filled. Each time a byte is dispatched to the disc, the contents of the FIFO will schuffle down and the microprocessor will be signalled that there is room for another byte in the FIFO. If the controller experiences a delay in gaining control of the UNIBUS to fetch the next word, the 16-byte buffer within the FIFO will enable it to keep sending serial data to the disc in sync with the write clock. Once the controller gains control of the UNIBUS, it should not release it until enough data has been read from main memory to refill the FIFO.

### THE M BUS

The microprocessor bus is the main communication path that links the various subsections of the disc controller together. On each microcycle, the M bus can perform one 8-bit data transfer between a bus source and a bus destination. At the beginning of the microcycle, the selected bus source begins driving data onto the M bus. After a short propagation delay, the data is available to all destinations on the M bus. At the very end of the microcycle, the data on the M bus will be strobed into the selected destinations.

The M bus sources and destinations are selected by 4-bit fields in each microinstruction (refer to Figure 8). Therefore, the M bus can have up to 15 sources and 15 destinations. In addition, the microprocessor can be either a source or a destination. Notice that if the microprocessor is not using the M bus during a microcycle, the M bus is free to perform a data transfer in parallel with whatever the microprocessor is doing. Also, it is sometimes useful for the microprocessor to be a second M bus destination. For example, when the controller is reading data from the disc, as each byte is transferred from the FIFO to either the high- or low-order UNIBUS data register, the microprocessor also receives the data on the M bus and adds it to the partially formed checksum. Thus, the microprocessor is kept busy building the checksum, while the M bus is being used as the data path between the disc interface and the UNIBUS interface. This parallel operation ability of the controller becomes important when the data rate of the disc drive approaches the transfer capacity of the controller because the controller's capacity is directly related to the number of microinstructions that must be executed on each pass through the inner loop of the disc write or disc read code.

### THE CLOCK

Figure 9 is a logic diagram of the disc controller clock which is the main source of synchronization signals within the controller. The Am25LS161 provides the ability to select multiples of the basic crystal frequency as the output of the clock circuit (see Table I). The duty cycle of the clock can be varied by adjusting the trimpot on the Am74123 One-Shot.

The crystal is selected to provide the proper frequency for the disc drive to be interfaced. Disc drives based on 2314 technology use the double frequency recording method,

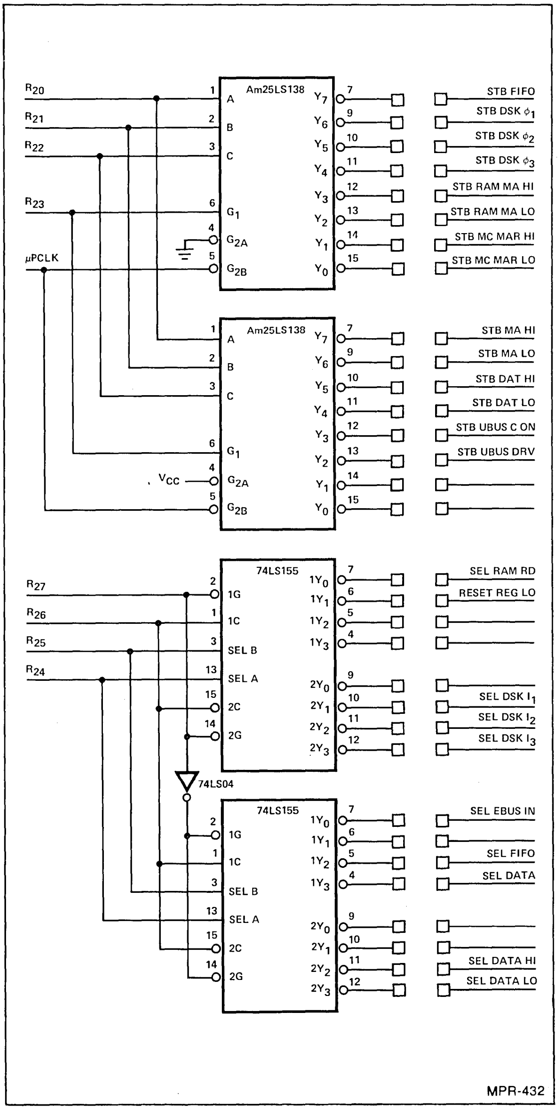



Figure 8.

which means that every other pulse is a clock pulse and the presence or absence of pulses between the clock pulses defines "ones" and "zeros". The crystal frequency must be the same as the double frequency when writing all "ones". If RATE is set to 17, then the frequency of $\mu$PCLK will be one-half the crystal frequency (see Table I), and the microprocessor will cycle once for every data bit received from the disc. This implies that for a 16-bit computer, 16 is the maximum number of microinstructions that can be executed on each pass through the inner loop of the disc read or disc write microcode. (Refer to Appendix I to find examples of the inner loop for reading and writing.) Any more and the controller will gradually fall behind until either the FIFO overflows (disc read) or runs out of data (disc write). It might be possible to clock the microprocessor as fast as it will run, and clock only the FIFO in sync with the disc drive (thus allowing

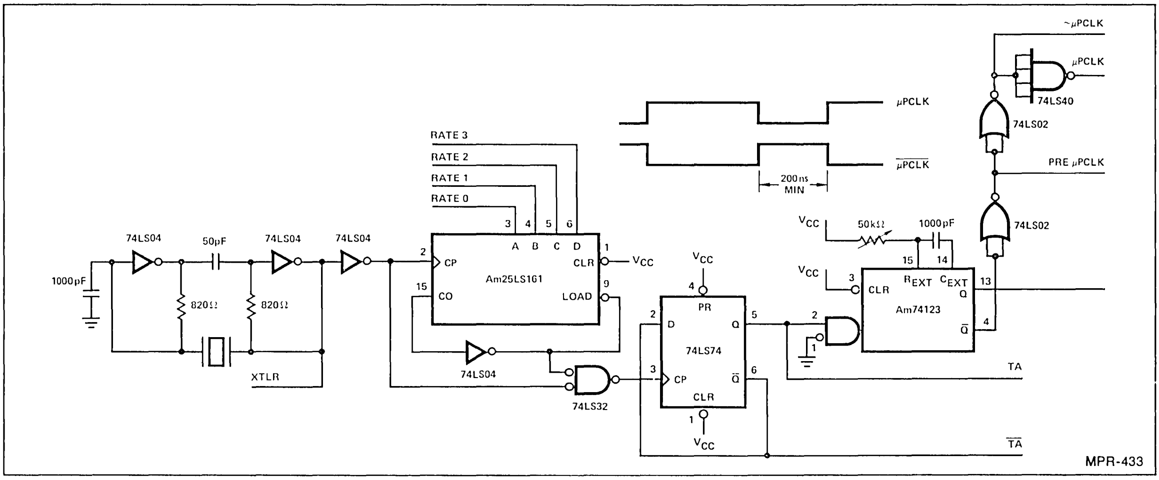



Figure 9.

| Rate Input | µPCLK Frequency |
|---|---|
| 17 | XTLR/2 |
| 16 | XTLR/4 |
| 15 | XTLR/6 |
| 14 | XTLR/8 |
| 13 | XTLR/10 |
| 12 | XTLR/12 |
| 11 | XTLR/14 |
| 10 | XTLR/16 |
| 7 | XTLR/18 |
| 6 | XTLR/20 |
| 5 | XTLR/22 |
| 4 | XTLR/24 |
| 3 | XTLR/26 |
| 2 | XTLR/28 |
| 1 | XTLR/30 |
| 0 | XTLR/32 |

Table I. Selecting µPCLK Frequency.

more microinstructions per word transferred), but strange problems with roots based in the beat frequency between the microprocessor clock and the FIFO clock would be likely to occur.

### INTERACTION OF CONTROLLER SUB-SECTIONS

Now that each sub-section has been described, it should be instructive to step through a disc transfer operation and observe the interaction of the controller sub-sections. Initially, the controller, in its idle state with no error conditions present, is looping on the SECTOR PULSE condition line. When SECTOR PULSE, a signal from the disc drive, goes "true", the controller loads the address of its control and status register into the UNIBUS address register, sets the control lines for a read operation, and then requests the UNIBUS by asserting NPR REQ. When control has been granted (signalled by the assertion of NPR RDY), the controller asserts TRAN to start the UNIBUS read cycle. The assertion of TRANSFER DONE signals that the control and status register has been read and the data is in the UNIBUS data register.

If the low-order bit of the control and status register is not set, then no operation has been requested. The controller will fall back into its idle loop as soon as it updates the disc status register, which contains the sector number of the sector currently under the heads.

If the low-order bit was set, then the next low-order three bits define the function to be performed. However, before dispatching to the appropriate routine for whatever function is to be performed, the controller reads the memory address, word count, and disc address for the upcoming transfer from its device registers and copies the data into its internal registers (these are the registers within the Am2901A's). Assuming, for this example, that the function is a disc read, the controller dispatches to the read microcode.

The first microinstruction of the read routine is a subroutine call to the SEEK routine. This routine loads the cylinder address, derived from the disc address, into the output port of the disc interface. The following microinstruction asserts the CYLINDER ADDRESS STROBE on another line in the output port. CYLINDER ADDRESS STROBE is then removed and the controller loops until the drive indicates that the seek has been completed. The SEEK subroutine then selects the proper head (again derived from the disc address) and finally starts looping on SECTOR PULSE. Each time a sector pulse is detected, the controller checks if this is the sector specified in the disc address. If it is, SEEK returns control to the microinstruction following the one that made the call on SEEK. Notice that SEEK doesn't just seek to the desired cylinder, it seeks the sector specified in the disc address.

When control returns to the disc read microcode, the controller waits about 100µs and then asserts READ ENABLE, one of the lines in the output port of the disc interface. At this time, the preamble should be under the enabled head. The preamble is a string of "zeros" terminated by a "one" bit. The "one" bit signals that the data record follows immediately. The first "one" bit will set a flip-flop and assert RD CLK ENABLE (see Figure 2), which will enable the RD CLK from the drive to start clocking data into the FIFO. Control now falls into the "disc read inner loop" microcode (flowcharted in Appendix I). In this loop, each time a byte is assembled in the FIFO it is copied alternately to the low-order UNIBUS data register and then to the high-order data register. As the data is copied from the FIFO to the data register, the checksum is built by the microprocessor. Every time the high data register is loaded, it is time to transfer another word into PDP-11 main memory. At the end of each UNIBUS transfer, INC MA is asserted to advance the UNIBUS memory address register to the next word address. The transfer word count is then decremented and if not zero another iteration through the "inner loop" is required. When the transfer word count reaches zero, the entire sector has been transferred, and the next word read from the disc is the checksum. This is compared with the checksum that has been built by the microprocessor. If they are not equal, the controller may attempt a retry, or it may just set the checksum error bit in the disc error register and continue as if there were no error. Assuming there wasn't any checksum error, the controller now drops READ ENABLE and the read has been completed. The controller now has only to update its external device registers from the internal set and it is back where first started: in the idle state.

Notice that the external device registers were updated only at the successful completion of the transfer. Therefore, whenever any error condition is encountered, the controller always has the complete information necessary to perform as many retries as the microcode dictates.

**SUMMARY**

Greater product versatility can be achieved by employing the Am2901A in the design of peripheral controllers. Indeed, there is nothing in the design discussed in this app note that says it has to be a disc controller. The FIFO is the only hardware that "leans" in the direction of a disc controller, and it does so only by virtue of the way it is clocked. But don't forget that the FIFO is just a general purpose, buffered parallel-to-serial and serial-to-parallel converter.

To stress this point of product versatility, let us briefly consider what would be necessary to convert this DEC RK11/RK05 compatible disc controller into a DEC TM11/TU10 mag tape controller. First, remove the FIFO. Next, re-label Figure 6 to read "Mag Tape Interface". Then connect a cable from the mag tape interface to whatever mag tape drive has been selected. Finally, write the microcode that will enable this hardware to emulate the TM11/TM10.

Voila!

NOTE: Advanced Micro Devices wishes to acknowledge the contributions of William Pitts in the design and implementation of this application note.

[1]Microprogramming Handbook; Mick, John R. and Jim Brick, Advanced Micro Devices, 1976.

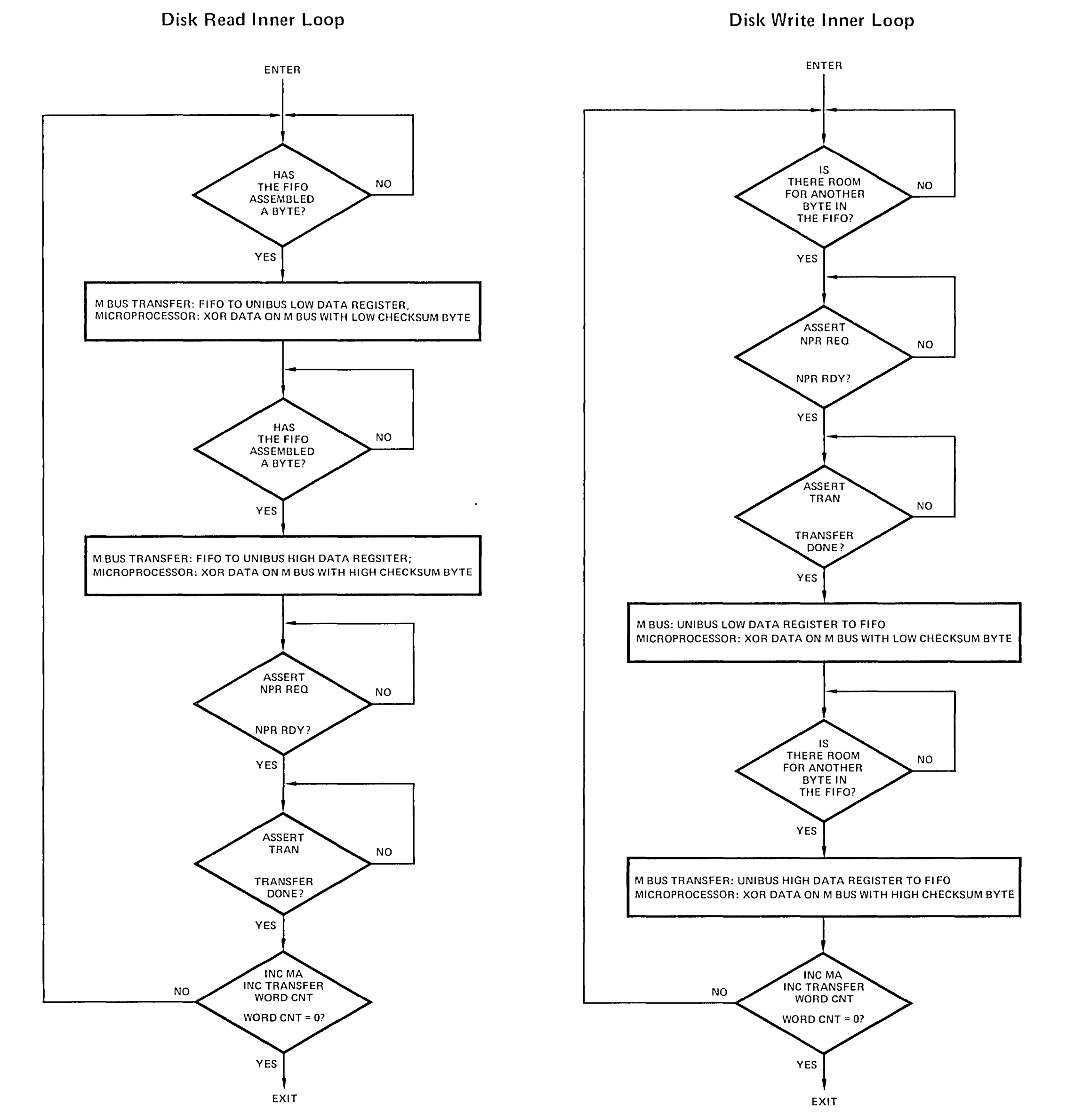
[2]PDP-11 Peripherals Handbook, Digital Equipment Corporation, 1975.

**PARTS LIST**

| Device | Description | Qty. |
|---|---|---|
| Am2901A | Four-Bit Bipolar Microprocessor Slice | 2 |
| Am2907 | Quad Bus Transceiver with Three-State Receiver and Parity | 5 |
| Am2911 | Microprogram Sequencer | 3 |
| Am29701 | Non-Inverting 64-Bit RAM with Three-State Outputs | 4 |
| Am29811 | Next Address Control Unit | 1 |
| Am25LS138 | One-of-Eight Decoder/Demultiplexer | 2 |
| Am25LS157 | Quad 2-Input Multiplexer with with Non-Inverting Outputs | 1 |
| Am25LS161 | Synchronous 4-Bit Binary Counter with Asynchronous Clear | 5 |
| Am25LS273 | 8-Bit Register with Common Clear | 12 |
| Am25LS374 | 8-Bit Register with Three-State Outputs | 10 |
| Am74123 | Dual One-Shot Multivibrator | 4 |
| Am74LS251 | 8-Input Multiplexer with Three-State Outputs | 2 |
| Am74S174 | Schottky 6-Bit High Speed Register | 2 |
| Am8838 | Quad Unified Bus Transceiver | 8 |
| Am9130E | 1024 x 4 N-Channel Static RAM | 12 |
| 7406 | Hex Inverter | 5 |

| Device | Description | Qty. |
|---|---|---|
| 74S74 | Dual D-Flip-Flop, Positive Edge-Triggered | 1 |
| 74LS00 | Quad 2-Input NAND Gate | 3 |
| 74LS02 | Quad 2-Input NOR Gate | 2 |
| 74LS04 | Hex Inverter | 5 |
| 74LS08 | Quad 2-Input AND Gate | 4 |
| 74LS11 | Triple 3-Input AND Gate | 1 |
| 74LS21 | Dual 4-Input AND Gate | 1 |
| 74LS32 | Quad 2-Input OR Gate | 1 |
| 74LS40 | Dual 4-Input NAND Buffer | 1 |
| 74LS55 | 2-Wide 4-Input AND-OR-Invert Gate | 2 |
| 74LS74 | Dual D-Flip-Flop, Positive Edge-Triggered | 7 |
| 74LS86 | Quad 2-Input Exclusive OR Gate | 1 |
| 74LS155 | Dual 2-to-4 Decoder/Demultiplexer | 3 |
| 8136 | 6-Bit Unified Bus Comparator, Open Collector | 2 |
| 8837 | Single Ended Line Receiver | 3 |
| 8881 | Quad 2-Input NAND Gate | 3 |
| 9403 | First-In-First-Out (FIFO) Buffer Memory | 2 |
| | TOTAL | 120 |

NOTE: The crystal used in this particular design oscillated at 3.125MHz, and was chosen so that this disc controller would be compatible with the DEC RK11/RK05. Those desiring a different data transmission rate may choose a different crystal to suit their application.




NOTE: Each box represents one microinstruction, but some instructions may be executed more than once.

MPR-434

Appendix I

# APPENDIX II

```
        ;  MICROCODE FOR RK11 SOFTWARE COMPATIBLE DISK CONTROLLER
        ;  WILLIAM M. PITTS
        ;  26 APR 77


        ; CONTROL RESET -- RESETS THE DISK CONTROLLER.  THIS ROUTINE IS ENTERED
        ;   WHENEVER 'INIT' IS ASSERTED ON THE UNIBUS OR WHEN
        ;   THE FUNCTION 'CONTROL RESET' HAS BEEN SPECIFIED BY
        ;   THE PDP-11.

000  89 A0 60 00 0C 00                  ;RESET ALL INTERNAL REGS
001  89 A0 70 00 0C 00
002  89 A0 80 00 0C 00
003  89 A0 90 00 0C 00
004  89 A0 A0 00 0C 00
005  89 A0 B0 00 0C 00
006  89 A0 C0 00 0C 00
007  89 A0 DF 60 0C 02                  ;SET UNIBUS MA TO 177402
008  89 A0 EF 70 0C FF
009  89 A0 FF 01 81 06                  ;CALL SUB FOR MEM WRITE

        ;  A DISK OPERATION HAS JUST BEEN COMPLETED, SO SET THE 'DONE' BIT &
        ;  CLEAR THE 'GO' BIT IN THE INTERNAL RKCS.

00A  75 A8 8F 00 0C 80                  ;SET 'DONE' BIT IN INTERNAL REG
00B  95 A8 8F 00 0C FE                  ;CLR 'GO' BIT IN INTERNAL REG

        ;  NOW IT'S TIME TO UPDATE THE EXTERNAL REGISTERS.

00C  1D A0 3F 00 0C 10                  ;RESET ERROR RETRY COUNTER
00D  11 AA 6F 60 0C 06                  ;SET UNIBUS MA TO 177406
00E  11 AB 7F 70 0C FF
00F  00 A0 0F 01 81 06                  ;CALL SUB TO UPDATE EXT RKWC
010  11 AC 60 00 0C 00
011  11 AD 7F 01 81 07                  ;UPDATE EXTERNAL RKBA
012  11 AE 60 00 0C 00
013  11 AF 7F 01 81 07                  ;UPDATE EXTERNAL RKDA
014  11 A8 6F 60 0C 04                  ;RESET UNIBUS MA TO 177404
015  11 A9 7F 01 81 06                  ;UPDATE EXTERNAL RKCS

        ;  IF INTERRUPTS ARE ENABLED, PERFORM INTERRUPT SEQUENCE.

016  94 A8 0F 00 0C 40                  ;INTERRUPTS ENABLED ?
017  00 A0 0F 00 03 1F                  ;NO, THEN GO TO 'IDLE'
018  00 A0 0F 40 0C 90                  ;INTERRUPT VECTOR TO UNIBUS DATA REG
019  00 A0 0F 50 0C 00
01A  00 A0 0F 20 0C 40                  ;REQUEST UNIBUS FOR INTERRUPT
01B  00 B0 0F 00 B3 1B                  ;LOOP HERE TILL WE'VE GOT THE UNIBUS
01C  00 A0 0F 20 0C 60                  ;ASSERT INTERRUPT
01D  00 B0 0F 00 53 1D                  ;LOOP TILL SSYN IS RECEIVED
01E  00 A0 0F 20 0C 00                  ;RELEASE UNIBUS

        ;  THE CONTROLLER WAITS FOR SOMETHING TO DO HERE IN THE 'IDLE' LOOP.
        ;  EVERY TIME A SECTOR PULSE IS SEEN, THE CONTROLLER READS THE EXTERNAL
        ;  RKCS TO SEE IF A DISK OPERATION HAS BEEN REQUESTED.  ALSO, THE EXTERNAL
        ;  RKDS IS UPDATED AT THIS TIME.

01F  00 A0 0F 60 0C 04                  ;SET UNIBUS MA TO 177404
020  00 A0 0F 70 0C FF
021  00 B0 0F 00 A3 1F                  ;LOOP & WAIT FOR SECTOR PULSE
022  00 A0 0F 01 81 00                  ;READ EXTERNAL RKCS
023  95 A6 8F 08 0C 7F                  ;MASK & COPY TO INTERNAL REG
024  95 A7 9F 08 0C 0F
025  00 A0 0F 01 81 01                  ;READ EXTERNAL RKDA
026  11 A6 E0 00 0C 00                  ;COPY TO INTERNAL RKDA
027  11 A7 FF 60 0C 00
028  00 A0 0F 00 81 B2                  ;SELECT SPECIFIED DISK DRIVE

        ;  UPDATE EXTERNAL RKDS

029  1D A0 61 00 0C 00                  ;LOAD SECTOR COUNTER
02A  D5 A6 6F 00 0C 40                  ;SET 'ACCESS READY'
02B  95 A7 7F 00 0C E0                  ;CLR ALL BUT DRIVE SELECT
```

```
02C   D5 A7 7F 00 0C 08            ;SET 'RK05'
02D   00 A0 0F 01 81 06            ;UPDATE EXTERNAL RKDS
02E   94 A8 0F 00 0C 01            ;'GO' BIT SET IN RKCS ?
02F   00 A0 0F 00 03 1F            ;NO, THEN LOOP & WAIT

        ;  A DISK OPERATION HAS BEEN REQUESTED.  IF REQUESTED FUNCTION IS
        ;  CONTROL RESET, GO DO IT.  ELSE CHECK IF ANY HARD ERRORS ARE PRESENT.
        ;  IF NO HARD ERRORS, UPDATE ALL INTERNAL REGISTERS AND THEN DECODE
        ;  THE REQUESTED FUNCTION AND DISPATCH TO THE APPROPRIATE ROUTINE.

030   89 A0 1F 01 81 00            ;CLR TEMP ERR REG (NOGO) & READ EXT RKER
031   94 A8 0F 00 0C 0E            ;CONTROL RESET ?
032   00 A0 0F 00 03 40            ;YES, SO DISPATCH
033   95 A6 6F 00 0C FC            ;ANY HARD ERRORS ?
034   00 B0 0F 01 03 22            ;YES, THEN ABORT
035   95 A7 7F 00 0C FF            ;HARD ERRORS ?
036   00 B0 0F 01 03 22            ;YES, THEN GO TO NOGO
037   00 A0 0F 60 0C 02            ;UPDATE EXT RKER
038   00 A0 0F 01 81 06
039   11 A8 60 00 0C 00            ;COPY INTERNAL RKCS TO LO & HI
03A   11 A9 7F 01 81 07            ;UPDATE EXT RKCS
03B   00 A0 0F 01 81 00            ;READ EXT RKWC
03C   11 A6 A0 00 0C 00            ; ... & COPY TO INTERNAL RKWC
03D   11 A7 BF 01 81 01            ;READ EXT RKBA
03E   11 06 C0 60 0C 00            ; ... & COPY TO INTERNAL RKBA & UNIBUS MA
03F   11 07 D0 70 0C 00
040   25 EA C0 00 0C 00            ;UPDATE INT RKBA TO TRANSFER END
041   25 FB DF 00 33 43            .+2
042   2D A0 D0 00 0C 00
043   25 EA C0 00 0C 00            ;AGAIN, SINCE RKWC IS A WORD CNT
044   25 FB DF 00 33 46            .+2
045   2D A0 D0 00 0C 00

        ;  DISPATCH

046   96 A8 0F 00 0C 0E            ;FUNCTION IS LOW 3 BITS OF R0
047   15 A0 0F 00 0C 48
048   10 80 0F 00 02 00            ;JMP TO '. + R0'
049   00 A0 0F 00 02 50            ;WRITE
04A   00 A0 0F 00 02 BC            ;READ
04B   00 A0 0F 00 02 BC            ;WRITE CHECK
04C   00 A0 0F 00 02 FC            ;SEEK
04D   00 A0 0F 00 02 BC            ;READ CHECK
04E   00 A0 0F 00 02 FE            ;DRIVE RESET
04F   00 A0 0F 00 02 0A            ;WRITE LOCK

        ;  WRITE OPERATION

050   95 A8 0F 00 0C 30            ;MASK OUT ALL BUT MEM EXT BITS
051   11 00 00 30 0C 00            ;SET A17, A16, C1, & C0
052   1D A0 11 00 0C 00            ;READ SECTOR COUNTER
053   95 A1 1F 00 0C 20            ;DRIVE WRITE LOCKED ?
054   00 B0 0F 01 03 19            ;YES, SO SET ERR BIT & ABORT
055   00 A0 0F 00 81 88            ;SEEK TO SPECIFIED CYLINDER
056   00 A0 00 50 0C 00            ;RESTORE FIFO REGISTERS
057   00 A0 0F F0 0C 00            ;LOAD FIFO WITH 2 '0' BYTES
058   00 A0 0F F0 0C 00
059   00 A0 0F E0 0C 03            ;ASSERT WRITE ENABLE & ERASE ENABLE
05A   1D A0 0F 00 0C 2E            ;CNTR FOR PREAMBLE BYTES
05B   00 A0 0F 00 D3 5B            ;LOOP TILL FIFO READY FOR MORE
05C   00 A0 0F F0 0C 00            ;FEED FIFO ANOTHER BYTE
05D   2D A0 00 00 0C 00            ;DEC PREAMBLE BYTE CNTR
05E   00 B0 0F 00 03 5B            ;ITERATE TILL CNTR GOES TO 0

        ;  THE PREAMBLE IS NOW ON ITS WAY TO THE DISK (SOME OF IT IS STILL IN THE
        ;  FIFO).  NEXT WILL BE THE SYNC BIT FOLLOWED BY 2 HEADER BYTES.

05F   00 A0 0F 00 D3 5F            ;WAIT FOR FIFO
060   00 A0 0F F0 0C 80            ;DISPATCH SYNC BIT
061   95 AE 0F 00 0C E0            ;CLR ALL BUT CYLINDER BITS
062   00 A0 0F 00 D3 62            ;WAIT FOR FIFO
063   11 00 00 F0 0C 00            ;DISPATCH 1ST HEADER BYTE
064   95 AF 0F 00 0C 1F            ;REMOVE DRIVE SEL BITS
```

```
065   00 A0 0F 00 D3 65                  ;WAIT FOR FIFO
066   11 00 00 F0 0C 00                  ;DISPATCH 2ND HEADER BYTE

                ;  THE SYNC BIT & THE 2 BYTE HEADER ARE NOW ON THE WAY TO THE DISK.
                ;  NEXT COMES 512 BYTES OF DATA.  BUT 1ST THE DISK WORD COUNT (DWC)
                ;  IS UPDATED BY SUBTRACTING THE UNIBUS WORD COUNT (UWC).

067   25 E1 20 00 0C 00

                ;  WRITE INNER LOOP

068   00 B0 0F 04 F3 68                  ;INITIATE UNIBUS TRANSFER, WAIT FOR FIFO & UBUS
069   15 A4 4B F0 0C 00                  ;COPY DATA TO FIFO & BUILD CHECKSUM
06A   00 B0 0F 08 33 6C                  ;BUMP MA.  CARRY INTO HI CHECKSUM BYTE ?
06B   0D E0 50 00 0C 00                  ;YES, SO INC CHK1
06C   00 A0 0F 00 03 0D                  ;WAIT FOR FIFO
06D   15 A5 5A F0 0C 00                  ;COPY DATA & BUILD CHECKSUM
06E   0D E0 10 00 0C 00                  ;UNIBUS WC EXHAUSTED ?
06F   00 B0 0F 00 03 68                  ;NO, THEN WRITE ANOTHER WORD TO DISK
070   10 A2 00 00 0C 00                  ;DISK WC ALSO EXHAUSTED ?
071   00 B0 0F 00 03 B7                  ;NO, THEN WRITE '0'S TILL IT IS

                ;  512 DATA BYTES ARE ON THE WAY TO THE DISK.  NEXT COMES 2 CHECKSUM BYTES
                ;  AND THEN THE POSTAMBLE.

072   00 A0 0F 00 D3 72                  ;WAIT FOR FIFO
073   11 04 40 F0 0C 00                  ;DISPATCH LO CHK BYTE
074   00 A0 0F 00 D3 74
075   11 05 50 F0 0C 00                  ; ... & NOW HI CHK BYTE

                ;  POSTAMBLE

076   1D A0 0F 00 0C 06
077   00 A0 0F 00 D3 77                  ;WAIT FOR FIFO
078   00 A0 0F F0 0C 00                  ;DISPATCH A PIECE OF THE POSTAMBLE
079   2D A0 00 00 0C 00                  ;MORE POSTAMBLE TO COME ?
07A   00 A0 0F 00 03 00                  ;YES, SO ITERATE

                ;  POSTAMBLE IS ON ITS WAY TO THE DISK.  NOW WE MUST WAIT FOR FIFO TO
                ;  EMPTY BEFORE THE WRITE CURRENTS ARE DISABLED.

07B   00 B0 0F 00 C3 7B                  ;FIFO OUTPUT REG EMPTY ?
07C   00 B0 0F 00 C3 7B                  ;MAKE SURE WE DIDN'T SEE BETWEEN BYTE GLITCH

                ;  'NXTSEC' WILL DISABLE THE WRITE OR READ CURRENTS AND CHECK IF MORE
                ;  DATA IS TO BE READ OR WRITTEN BEYOND THE SECTOR THAT HAS JUST BEEN
                ;  COMPLETED.  IF MORE IS CALLED FOR, 'DOSEEK' WILL BE CALLED TO
                ;  POSITION THE HEADS FOR THE NEXT SECTOR & THEN CONTROL WILL BE RETURNED
                ;  TO THE READ OR WRITE ROUTINE.

07D   0D E0 EF E0 0C 00                  ;BUMP DA TO NEXT SECTOR, DISABLE CURRENTS
07E   95 AE 0F 00 0C 0F                  ;COPY JUST SECTOR TO R0
07F   34 E0 0F 00 0C 0C                  ;OVERFLOW TO NEXT TRACK ?
080   00 B0 0F 00 03 84                  ;NO, THEN GO TO 'MORE'
081   15 AE EF 00 0C 04                  ;NEXT TRACK, SECTOR 0
082   00 B0 0F 00 33 84                  ;CARRY OUT OF RKDA0 ?
083   0D E0 F0 00 0C 00                  ;YES, SO BUMP RKDA1
084   10 AB 00 00 0C 00                  ;ANY MORE TO TRANSFER ?
085   00 A0 0F 00 03 0A                  ;NO, THEN WE'RE ALL DONE
086   94 AE 0F 00 0C 1F                  ;NEED SEEK TO NEW CYLINDER ?
087   00 B0 0F 00 03 A4                  ;NO, PRETEND SEEK JUST COMPLETED

                ;  'DOSEEK' IS THE ROUTINE THAT SEEKS TO THE CYLINDER ADDRESSED IN THE
                ;  INTERNAL RKDA.  AFTER THE SEEK HAS BEEN COMPLETED, 'OKSEEK' WILL
                ;  WAIT UNTIL THE SPECIFIED SECTOR IS JUST BEFORE THE HEADS (SECTOR PULSE
                ;  IS SEEN) AND THEN RETURN TO THE CALLING ROUTINE.

088   95 AE 0F 00 0C 0F                  ;COPY SECTOR BITS TO R0
089   34 E0 0F 00 0C 0C                  ;LEGAL SECTOR NUMBER ?
08A   00 B0 0F 01 13 10                  ;NO, THEN TAKE ERROR EXIT
08B   95 AE 6F 00 0C E0                  ;COPY CYL BITS TO LO, HI
08C   95 AF 7F 00 0C 1F
08D   10 86 00 D0 0C 00                  ;LOAD LOW CYL REG
```

```
08E   10 87 00 C0 0C 00                    ; ... & HIGH CYL REG
08F   00 A0 0F E0 0C 08                    ;INITIATE SEEK

                ;  THE SEEK HAS JUST BEEN INITIATED, NOW LOOP ON CHECKING FOR SEEK ERROR
                ;  OR SEEK DONE.

090   1D A0 02 00 0C 00                    ;GET ERROR BITS
091   00 A0 0F E0 0C 00                    ;DROP 'STBCYL'
092   00 B0 0F 01 03 14                    ;IF ERR, GO TO 'UNSAFE'
093   1D A0 01 00 0C 00                    ;READ SECTOR COUNTER
094   95 A0 0F 00 0C 40                    ;BUSY SEEKING ?
095   00 B0 0F 00 03 90                    ;YES, SO LOOP

                ;  SEEK COMPLETE.  NOW READ HEADER OF NEXT SECTOR THAT COMES BY AND
                ;  VERIFY THAT THIS IS THE CORRECT CYLINDER, UNLESS THIS IS A
                ;  FORMAT READ IN WHICH CASE THERE IS NO CYLINDER VERIFICATION.

096   94 A9 0F 00 0C 04                    ;FORMAT READ ?
097   00 B0 0F 00 03 A4                    ;YES, SO BYPASS VERIFICATION
098   00 B0 0F 00 A3 98                    ;WAIT FOR SECTOR PULSE
099   1D A0 0F 00 0C 80                    ;LOAD DELAY COUNTER
09A   2D A0 00 00 0C 00                    ;WAIT FOR PREAMBLE TO GET UNDER HEADS
09B   00 B0 0F 00 03 9A                    ;WAIT LOOP
09C   00 A0 00 50 0C 00                    ;RESTORE FIFO REGISTERS
09D   00 A0 0F E0 0C 04                    ;ASSERT READ ENABLE
09E   00 A0 0F 00 C3 9E                    ;WAIT FOR 1ST HEADER BYTE
09F   34 E6 0E 00 0C 00                    ;LOW CYL ADDR OK ?
0A0   00 B0 0F 01 03 18                    ;NO, THEN SEEK ERROR
0A1   00 A0 0F 00 C3 A1                    ;WAIT FOR 2ND HEADER BYTE
0A2   34 E7 0E 00 0C 00                    ;HIGH CYL ADDR OK ?
0A3   00 B0 0F 01 03 18                    ;NO, THEN SEEK ERROR

                ;  GOOD SEEK.  NOW INITIALIZE CHECKSUM REGISTERS TO ZERO, SET UNIBUS
                ;  WORD COUNT TO WHATEVER IT SHOULD BE, & SET DISK WORD COUNT FOR ONE
                ;  SECTOR (256 WORDS).

0A4   89 A0 40 00 0C 00                    ;CLR CHECKSUM REGISTERS
0A5   89 A0 50 00 0C 00
0A6   89 A0 20 00 0C 00                    ;256. WORD DISK TRANSFER
0A7   89 A0 10 00 0C 00                    ;ASSUME ALL OF SECTOR WANTED
0A8   0D E0 B0 00 0C 00                    ;FULL SECTOR TRANSFER ?
0A9   00 B0 0F 00 03 AD                    ;'OKSEEK' IF MORE THAN FULL SECTOR
0AA   11 AA 10 00 0C 00                    ;SET UNIBUS WORD COUNT
0AB   89 A0 A0 00 0C 00                    ;CLR INTERNAL RKWC (LAST SECTOR)

                ;  SELECT HEAD

0AC   00 A0 0F 00 81 B2

                ;  AT 'OKSEEK', EVERYTHING IS SET UP FOR THE UPCOMING TRANSFER.  NOW
                ;  THE CONTROLLER WILL WAIT UNTIL THE SPECIFIED SECTOR IS JUST REACHING
                ;  THE HEADS BEFORE RETURNING TO THE CALLER.

0AD   00 B0 0F 00 A3 AD                    ;WAIT FOR SECTOR PULSE
0AE   D4 AE 01 00 0C 00                    ;SPECIFIED SECTOR ?
0AF   94 A0 0F 00 0C 0F                    ;DON'T KNOW TILL WE CLEAN IT UP
0B0   00 B0 0F 00 03 AD                    ;NOT SECTOR WE WANT
0B1   00 A0 0F 00 8A 00                    ;RETURN

                ;  'SELECT' SELECTS THE HEAD SPECIFIED IN THE INTERNAL RKDA.

0B2   95 AE 6F 00 0C 10                    ;COPY HEAD SEL BIT TO 'LO'
0B3   10 86 00 D0 0C 00                    ;SELECT HEAD
0B4   00 A0 0F 00 8A 00                    ;RETURN

                ;  'WRITEZ' APPENDS ZEROS TO SHORT RECORDS AS THEY ARE WRITTEN ON
                ;  THE DISK.

0B5   0D E0 20 00 0C 00                    ;MARK PASSAGE OF ANOTHER WORD TO DISK
0B6   00 A0 0F 00 03 72                    ;'WRTDON' WHEN DONE
0B7   00 A0 0F 00 D3 B7                    ;WAIT FOR FIFO
0B8   00 A0 0F F0 0C 00                    ;DISPATCH A ZERO
0B9   00 A0 0F 00 D3 B9                    ;WAIT
```

```
0BA   00 A0 0F F0 0C 00                 ;PAD
0BB   00 A0 0F 00 02 B5                 ;LOOP

            ;  READ OPERATIONS -- READ, READ CHECK, & WRITE CHECK ALL TRANSFER HERE.

0BC   94 80 0F 30 0C 32                 ;SET A17, A16, C1, & C0
0BD   00 A0 0F 00 81 88                 ;SEEK
0BE   1D A0 0F 00 0C 80                 ;LOAD DELAY COUNTER
0BF   2D A0 00 00 0C 00                 ;WAIT FOR PREAMBLE TO GET UNDER HEADS
0C0   00 B0 0F 00 03 BF                 ;WAIT LOOP
0C1   00 A0 00 50 0C 00                 ;RESTORE FIFO REGISTERS
0C2   00 A0 0F E0 0C 04                 ;ASSERT READ ENABLE
0C3   00 A0 0F 00 C3 C3                 ;WAIT FOR 1ST HEADER BYTE
0C4   1D A0 6E 00 0C 00                 ;GET 1ST HEADER BYTE
0C5   00 A0 0F 00 C3 C5                 ;WAIT FOR 2ND HEADER BYTE
0C6   1D A0 7E 00 0C 00                 ; ... & STICK IT IN 'HI'
0C7   94 A9 0F 00 0C 04                 ;FORMAT READ ?
0C8   00 80 0F 00 03 F6                 ;YES, SO TRANSFER JUST HEADER BYTES
0C9   94 A8 0F 00 0C 02                 ;READ OR WRITE CHECK ?
0CA   00 80 0F 00 03 DF                 ;YES
0CB   25 E1 20 00 0C 00                 ;UPDATE DISK WORD COUNT

            ;  DISK READ INNER LOOP

0CC   00 A0 0F 00 C3 CC                 ;WAIT FOR DATA
0CD   15 A4 4E 40 0C 00                 ;COPY DATA & BUILD CHECKSUM
0CE   00 B0 0F 08 33 D0                 ;BUMP MA, CARRY INTO CHK1 ?
0CF   0D E0 50 00 0C 00                 ;YES, SO SEE THAT IT GETS THERE
0D0   00 B0 0F 00 E3 D0                 ;WAIT FOR DATA & UNIBUS
0D1   15 A5 5E 50 0C 00                 ;COPY & BUILD
0D2   0D E0 10 04 0C 00                 ;START UNIBUS TRANSFER, THIS LAST WORD ?
0D3   00 B0 0F 00 03 CC                 ;NO, THEN GO READ NEXT WORD
0D4   10 A2 00 00 0C 00                 ;MORE DATA STILL IN SECTOR ?
0D5   00 B0 0F 00 03 E1                 ;YES, SO CONT TO BUILD CHECKSUM

            ;  DATA HAS JUST BEEN READ.  NOW READ & VERIFY CHECKSUM.

0D6   00 A0 0F 00 C3 D6                 ;WAIT FOR 1ST CHECKSUM BYTE
0D7   35 E4 4E 00 0C 00                 ;SUB LOW CHECKSUM BYTES
0D8   00 B0 0F 00 33 DA                 ;DON'T FORGET THE CARRY
0D9   2D A0 50 00 0C 00                 ;OK
0DA   00 A0 0F 00 C3 DA                 ;WAIT FOR 2ND CHECKSUM BYTE
0DB   35 E5 5E 00 0C 00                 ;SUB HIGH CHECKSUM BYTES
0DC   64 A4 5F E0 0C 00                 ;CHECKSUM ERROR ?  DISABLE READ CURRENT
0DD   00 B0 0F 01 03 0E                 ;NO, THEN ERROR
0DE   00 A0 0F 00 02 7D                 ;IF MORE, CONT TO NEXT SECTOR

            ;  READ & WRITE CHECK TRANSFER TO 'RDCK0'.  NOW TEST TO SEE WHICH IT
            ;  IS AND BRANCH ACCORDINGLY.

0DF   94 A8 0F 00 0C 04                 ;READ CHECK ?
0E0   00 B0 0F 00 03 EA                 ;NO, SO MUST BE WRITE CHECK

            ;  READ CHECK INNER LOOP

0E1   00 A0 0F 00 C3 E1                 ;WAIT FOR DATA
0E2   15 A4 4E 00 0C 00                 ;BUILD CHECKSUM
0E3   00 B0 0F 00 33 E5                 ;CARRY ?
0E4   0D E0 50 00 0C 00                 ;YES
0E5   00 A0 0F 00 C3 E5                 ;WAIT FOR DATA
0E6   15 A5 5E 00 0C 00                 ;BUILD CHECKSUM
0E7   0D E0 20 00 0C 00                 ;BUMP DISK WORD COUNT
0E8   00 B0 0F 00 03 E1                 ;LOOP THRU ALL OF SECTOR
0E9   00 A0 0F 00 02 D6                 ; ... & THEN GO TO 'RDDONE'

            ;  WRITE CHECK INNER LOOP

0EA   00 A0 00 04 0C 00                 ;START UNIBUS READ
0EB   00 A0 0F 00 03 00                 ;WAIT FOR DATUM FROM BOTH SOURCES
0EC   1D A0 6E 00 0C 00                 ;GET DISK DATA BYTE
0ED   34 E6 0B 00 0C 00                 ;SUB BYTE FROM MEMORY
0EE   00 B0 0F 01 03 0C                 ;ERROR IF NOT 0
0EF   1D A0 6A 00 0C 00                 ;GET BYTE FROM MEMORY
```

```
0F0   00 A0 0F 00 C3 F0                    ;WAIT FOR DISK DATA
0F1   34 E6 0E 00 0C 00                    ;SUB BYTE FROM DISK
0F2   00 B0 0F 01 03 0C                    ;ERROR IF NOT 0
0F3   0D E0 10 00 0C 00                    ;BUMP UNIBUS WORD CNT
0F4   00 B0 0F 00 03 EA                    ;LOOP TILL DONE
0F5   00 A0 0F 00 02 FA                    ;JOIN FORMAT READ STREAM

                ;  FORMAT READ -- THE 2 HEADER BYTES ARE IN 'LO' & 'HI'.  TRANSFER THEM
                ;  TO MAIN MEMORY & ITERATE TILL UNIBUS WORD COUNT IS 0.

0F6   11 E1 A0 00 0C 00                    ;INTERNAL RKWC NEEDS UPDATE
0F7   00 A0 0F 00 03 F9
0F8   2D A0 80 00 0C 00                    ;UNDO WHAT 'SEEKOK' DID
0F9   00 A0 0F 01 81 07                    ;TRANSFER HEADER
0FA   00 A0 0F 00 81 7D                    ;IF MORE, CONT TO NEXT SECTOR
0FB   00 A0 0F 00 02 BE                    ;REENTER READ STREAM

                ;  SEEK ROUTINE -- SINCE SEVERAL RK05 DRIVES ARE MAPPED ONTO THE PERTEC
                ;  DRIVE, IT IS BEST TO SEEK ONLY BEFORE PERFORMING A DATA TRANSFER.

0FC   75 A9 9F 00 0C 20                    ;SET 'SEARCH COMPLETE' IN RKCS
0FD   00 A0 0F 00 02 0A                    ;WE'RE DONE !

                ;  DRIVE RESET -- RECALIBRATE & BRANCH TO 'SEEK'

0FE   00 A0 0F 01 81 28                    ;RECALIBRATE
0FF   00 A0 0F 00 02 FC                    ;JOIN UP WITH 'SEEK'

                ;  UNIBUS DATA TRANSFER SUBROUTINES

100   00 A0 0F 30 0C 30                    ;INITIAL DATA IN ENTRY POINT
101   00 A0 00 04 0C 00                    ;INITIATE TRANSFER
102   00 B0 0F 01 63 02                    ;WAIT FOR UNIBUS
103   1D A0 6B 00 0C 00                    ;GET LOW DATA BYTE
104   1D A0 7A 00 0C 00                    ; ... & HIGH DATA BYTE
105   00 A0 0F 08 8A 00                    ;RETURN

106   00 A0 0F 30 0C 32                    ;INITIAL DATA OUT ENTRY POINT
107   10 86 00 40 0C 00                    ;COPY LOW DATA
108   10 87 00 50 0C 00                    ; ... & HIGH DATA
109   00 A0 00 04 0C 00                    ;INITIATE TRANSFER
10A   00 B0 0F 01 63 0A                    ;WAIT FOR 'TRANSFER DONE'
10B   00 A0 0F 08 8A 00                    ;RETURN

                ;  ERROR ROUTINES -- ALL ERRORS ARE HANDLED IN THE SAME MANNER.
                ;  1ST THE APPROPRIATE ERROR BIT IS SET IN THE INTERNAL ERROR REGISTER
                ;  (R0, UWC), AND THEN THE INTERNAL RKCS IS CHECKED TO SEE IF
                ;  'STOP ON SOFT ERROR' IS SET.  IF NOT, THEN 'RETRY' WILL BE CALLED
                ;  TO ATTEMPT THE COMPLETE TRANSFER ONCE AGAIN.  UP TO 16 RETRIES WILL
                ;  BE ATTEMPTED AUTOMATICALLY.  IF THE ERROR CONDITION PERSISTS, OR
                ;  'STOP ON SOFT ERROR' IS SET, THE EXTERNAL RKER WILL BE READ AND ORED
                ;  WITH THE INTERNAL ERROR REGISTER & THEN THE EXTERNAL RKER WILL BE
                ;  UPDATED WITH THIS NEW ERROR DATA.  FINALLY, THE EXTERNAL RKCS WILL
                ;  BE UPDATED WITH THE APPROPRIATE ERROR BITS & CONTROL WILL TRANSFER
                ;  TO 'DONE'.

10C   1D A0 0F 00 0C 01                    ;SET 'WCE' IN RKER0
10D   00 A0 0F 01 02 13

10E   1D A0 0F 00 0C 02                    ;SET 'CSE' IN RKER0
10F   00 A0 0F 01 02 13

110   1D A0 0F 00 0C 20                    ;SET 'NXS' IN RKER0
111   00 A0 0F 01 02 13

112   1D A0 0F 00 0C 40                    ;SET 'NXC' IN RKER0
113   89 A0 1F 01 02 1A                    ;CLR RKER1, GO TO 'ERROR'

114   94 A0 0F 00 0C 02                    ;SEEK INCOMPLETE ?
115   00 B0 0F 01 03 12                    ;YES, THEN PRETEND 'NXC'
116   1D A0 1F 00 0C 80                    ;SET 'DRE' IN RKER1
117   00 A0 0F 01 02 19
```

```
118   1D A0 1F 00 0C 10        ;SET 'SKE' IN RKER1
119   89 A0 00 00 0C 00        ;CLR RKER0
11A   94 A9 0F 00 0C 01        ;STOP ON SOFT ERROR ?
11B   00 80 0F 01 03 1E        ;YES
11C   2D A0 30 00 0C 00        ;DEC ERR CNTR, TIME TO GIVE UP ?
11D   00 80 0F 01 03 29        ;NO, SO TRY AGAIN
11E   00 A0 0F 60 0C 02        ;SET MA = 177402 (RKER)
11F   00 A0 0F 70 0C FF
120   00 A0 0F 01 81 00        ;READ EXTERNAL RKER
121   65 A0 60 00 0C 00        ;UPDATE OLD RKER
122   65 A1 7F 60 0C 02        ;RESET MA TO 177402
123   00 A0 0F 01 81 06        ;WRITE UPDATED RKER
124   75 A9 9F 00 0C 80        ;SET 'ERROR' IN RKCS1
125   94 A0 0F 00 0C 03        ;SOFT ERROR ?
126   00 80 0F 00 03 0A        ;YES
127   75 A9 9F 00 0C 40        ;NO, SO SET 'HE' IN RKCS1
128   00 A0 0F 00 02 0A        ;RETURN IN DISGRACE

129   00 A0 0F 01 81 2B        ;RECALIBRATE
12A   00 A0 0F 00 02 1F        ; ... & TRY AGAIN

12B   00 A0 0F C0 0C 40        ;ASSERT 'RESTORE'
12C   00 A0 0F E0 0C 08        ;ASSERT 'STBCYL'
12D   00 A0 00 00 0C 00        ;PAUSE
12E   00 A0 0F E0 0C 00        ;REMOVE 'STBCYL'
12F   1D A0 01 00 0C 00        ;LOOP TILL NOT 'BUSY'
130   94 A0 0F 00 0C 40        ;BUSY ?
131   00 A0 0F 01 03 2F        ;YES
132   00 A0 0F 00 8A 00        ;EXIT
```

**ADVANCED**
**MICRO**
**DEVICES, INC.**
*901 Thompson Place*
*Sunnyvale*
*California 94086*
*(408) 732-2400*
*TWX: 910-339-9280*
*TELEX: 34-6306*
*TOLL FREE*
*(800) 538-8450*

AM-PUB065 2-78