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 En-
gineer to implement new systems with higher logic density, better cost-effectiveness, and improved product versa-
tility. 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 versa-
tility 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 grand-
father. 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
explained. 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 gen-
erate 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 UN-
IBUS 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 micropro-
gram, that enables the disc controller to completely emu-
late the RK11 disc controller (SSI TTL controller from
DEI). 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 micro-
processor 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 control-
er without the intervention of the host computer. In addi-
tion to supplying logical processing power, the micro-
processor also provides seventeen high-speed, 8-bit tem-
porary storage registers. Most of these registers are
assigned specific functions. In this application, twelve reg-
isters were used to build six 16-bit registers. These regis-
ters contain the disc address, memory address, transfer
word count, control and status information, error informa-
tion, 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 sub-sections of the disc controller. Four condition lines
(ZERO, MINUS, OVRFL, and CARRY) communicate the
results of logical 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 16-bit instruction
from the microprogram register. This 16-bit instruction con-
ists of a 8-bit microinstruction decode, an 8-bit register
select, and the carry-in, and carry-out. In this application, twelve regis-
ters are 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 sub-sections of the disc controller. Four condition lines
(ZERO, MINUS, OVRFL, and CARRY) communicate the
results of logical 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

<table>
<thead>
<tr>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESTINATION</td>
<td>ALU FUNCTION</td>
<td>ALU SOURCE</td>
<td>A REGISTER ADDRESS</td>
<td>B REGISTER ADDRESS</td>
<td>CARRY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTROL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MICROINSTRUCTION DECODE</th>
<th>REGISTER SELECT</th>
</tr>
</thead>
</table>

Figure 3.

The microprogram register is comprised of six 8-bit registers. The low-order register holds the data portion of each microinstruction. This register, an Am29LS374, 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 Am29LS273'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 (µPCLK). 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 µPCLK. 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 µPCLK, 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 interpret different types of disc drives, the microcodes 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.

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.
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 address register was 1600000, because the IN1 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.

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 (Am28LS161). In a DMA transfer, the starting address would be initially loaded into the Am28LS161'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 NRR REQ or NRR 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 NRR RDY or BRR 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 2914 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.
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 RDCLK 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, it will 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 a 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 16 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, 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 Am25LS181 provides the ability to select multiples of the basic crystal frequency as the output of the clock circuit (see Table I). To adjust the trim pot 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, 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 pCLCK 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
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 Am2961A'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 discon-
cremented 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 microproces-
sor. 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 con-
troller now has only to update its external device registers
from the internal set and it is back where it first started: in the
idle state.

Notice that the external device registers were updated only
at the successful completion of the transfer. Therefore, who-
never 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. In-
deed, 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 "leaks" 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 con-
sider what would be necessary to convert this DEC RK11/
RK05 compatible disc controller into a DEC TM11/TM10 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.

Volia!

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

---

### PARTS LIST

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
<th>Qty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Am2901A</td>
<td>Four-Bit Bipolar Microprocessor Slices</td>
<td>2</td>
</tr>
<tr>
<td>Am2907</td>
<td>Quad Bus Transceiver with</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Three-State Receiver and Parity</td>
<td>5</td>
</tr>
<tr>
<td>Am2911</td>
<td>Microprogram Sequencer</td>
<td>3</td>
</tr>
<tr>
<td>Am2761</td>
<td>Non-Inverting 64-Bit RAM with</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Three-State Outputs</td>
<td></td>
</tr>
<tr>
<td>Am29111</td>
<td>Next Address Control Unit</td>
<td>1</td>
</tr>
<tr>
<td>Am25LS138</td>
<td>One-of-Eight Decoder/Demultiplexer</td>
<td>2</td>
</tr>
<tr>
<td>Am25LS157</td>
<td>Quad 2-Input Multiplexer with</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>with Non-Inverting Outputs</td>
<td></td>
</tr>
<tr>
<td>Am25LS161</td>
<td>Synchronous 4-Bit Binary Counter</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>with Asynchronous Clear</td>
<td></td>
</tr>
<tr>
<td>Am25LS23</td>
<td>8-Bit Register with</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Common Clear</td>
<td></td>
</tr>
<tr>
<td>Am25LS374</td>
<td>8-Bit Register with Three-State Outputs</td>
<td>10</td>
</tr>
<tr>
<td>Am74123</td>
<td>Dual One-Shot Multivibrator</td>
<td>4</td>
</tr>
<tr>
<td>Am74LS251</td>
<td>8-Input Multiplexer with</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Three-State Outputs</td>
<td></td>
</tr>
<tr>
<td>Am74S174</td>
<td>Schottky 6-Bit High Speed Register</td>
<td>2</td>
</tr>
<tr>
<td>Am838</td>
<td>Quad Unified Bus Transceiver</td>
<td>8</td>
</tr>
<tr>
<td>Am9130E</td>
<td>1024 x 4 N-Channel Static RAM</td>
<td>12</td>
</tr>
<tr>
<td>7406</td>
<td>Hex Inverter</td>
<td>5</td>
</tr>
<tr>
<td>74S74</td>
<td>Dual D-Flip-Flop, Positive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Edge-Triggered</td>
<td>1</td>
</tr>
<tr>
<td>74LS00</td>
<td>Quad 2-Input NAND Gate</td>
<td>3</td>
</tr>
<tr>
<td>74LS02</td>
<td>Quad 2-Input NOR Gate</td>
<td>2</td>
</tr>
<tr>
<td>74LS04</td>
<td>Hex Inverter</td>
<td>5</td>
</tr>
<tr>
<td>74LS08</td>
<td>Quad 2-Input AND Gate</td>
<td>4</td>
</tr>
<tr>
<td>74LS11</td>
<td>Triple 3-Input AND Gate</td>
<td>1</td>
</tr>
<tr>
<td>74LS21</td>
<td>Dual 4-Input AND Gate</td>
<td>1</td>
</tr>
<tr>
<td>74LS32</td>
<td>Quad 2-Input OR Gate</td>
<td>1</td>
</tr>
<tr>
<td>74LS40</td>
<td>Dual 4-Input NAND Buffer</td>
<td>1</td>
</tr>
<tr>
<td>74LS55</td>
<td>2-Wide 4-Input AND-OR-Invert Gate</td>
<td>2</td>
</tr>
<tr>
<td>74LS74</td>
<td>Dual D-Flip-Flop, Positive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Edge-Triggered</td>
<td>7</td>
</tr>
<tr>
<td>74LS86</td>
<td>Quad 2-Input Exclusive OR Gate</td>
<td>1</td>
</tr>
<tr>
<td>74LS165</td>
<td>Dual 2-to-4 Decoder/Demultiplexer</td>
<td>3</td>
</tr>
</tbody>
</table>
| 8136       | 6-Bit Unified Bus Comparator, Open Collect | 2
| 8937       | Single Ended Line Receiver              | 2    |
| 8981       | Quad 2-Input NAND Gate                  | 3    |
| 9403       | First-In-First-Out (FIFO) Buffer Memory  | 2    |

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.
Disk Read Inner Loop

Enter

Has the FIFO assembled a byte?

Yes

M bus transfer: FIFO to UNIBUS low data register
Microprocessor: pop data on M bus with low checksum byte

No

Has the FIFO assembled a byte?

Yes

M bus transfer: FIFO to UNIBUS high data register
Microprocessor: pop data on M bus with high checksum byte

No

Assert MPR reg

Yes

MPR RDY?

No

Assert tran

Transfer done?

Yes

No

INC transfer or NORD CNT
NORD CNT = 0?

Yes

INC transfer or NORD CNT
NORD CNT = 0?

Yes

Exit

Disk Write Inner Loop

Enter

Is there room for another byte in the FIFO?

Yes

Assert MPR reg

No

MPR RDY?

Yes

Assert tran

Transfer done?

No

M bus transfer: UNIBUS low data register to FIFO
Microprocessor: pop data on M bus with low checksum byte

No

M bus transfer: UNIBUS high data register to FIFO
Microprocessor: pop data on M bus with high checksum byte

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

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

CONTROL RESET — RESET 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 POP-11.

000 00 AE 00 00 AC 80
001 00 AE 00 00 00 00
002 00 AE 00 00 00 00
003 00 AE 00 00 00 00
004 00 AE 00 00 00 00
005 00 AE 00 00 00 00
006 00 AE CD 00 00 00
007 00 AE 00 00 00 C2
008 00 AE EF 70 00 FF
009 00 AE FF 01 81 0A

RESET ALL INTERNAL REGS

SET UNIBUS MA TO 1774B2

CALL SUB FOR NEW WRITE

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

00A 75 AB AF 00 00 0C 80
00B 95 AB AF 00 0C FF

SET 'DONE' BIT IN INTERNAL REG
CLR 'GO' BIT IN INTERNAL REG

NOW IT'S TIME TO UPDATE THE EXTERNAL REGISTERS.

00C 00 00 00 AE 00 00 0C 80
00D 01 AE 0F 00 00 00 00
00E 01 AE FF 70 00 FF
00F 00 AB 00 01 81 0A
010 01 AC 00 00 00 00
011 01 AE 0F 01 81 07
012 01 AE 00 00 00 00
013 01 AF 0F 01 81 07
014 01 AE 0F 00 0C 04
015 01 AE 0F 01 81 0A

RESET ERROR RETRY COUNTER
SET UNIBUS MA TO 1774B6
CALL SUB TO UPDATE EXT RKRC
UPDATE EXTERNAL RK8A
UPDATE EXTERNAL RKDA
UPDATE EXTERNAL RKCS

IF INTERRUPTS ARE ENABLED, PERFORM INTERRUPT SEQUENCE.

016 94 AB AF 00 00 0C 40
017 00 AB 00 03 00 01
018 00 AB 00 03 00 00
019 00 AB 00 04 00 00
01A 00 AB 00 04 00 00
01B 00 AB 00 03 00 10
01C 00 AB 00 03 00 00
01D 00 AB 00 03 00 10
01E 00 AB 00 03 00 00

INTERRUPTS ENABLED?
NO, THEN GO TO 'IDLE'
INTERRUPT VECTOR TO UNIBUS DATA REG
REQUEST UNIBUS FOR INTERRUPT
FLOOR HERE TILL WE'VE GOT THE UNIBUS
PASS INTERRUPT
FLOOR TILL SSYN IS RECEIVED
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
RKCS IS UPDATED AT THIS TIME.

01F 00 AE 0F 60 00 0C 84
020 00 AE 0F 70 00 FF
021 00 AB 00 01 81 00
022 00 AD 00 01 81 00
023 00 AE 00 00 00 00
024 05 87 87 87 87 87
025 05 87 87 87 87 87
026 05 87 87 87 87 87
027 05 87 87 87 87 87
028 05 87 87 87 87 87

SET UNIBUS MA TO 1774B4
FLIP & WAIT FOR SECTOR PULSE
READ EXTERNAL RKCS
READ EXTERNAL RKDA
COPY TO INTERNAL RKDA

SELECT SPECIFIED DISK DRIVE

UPDATE EXTERNAL RKCS

029 10 AD 61 00 00 0C 80
02A 05 AD 61 00 00 0C 80
02B 05 AD 61 00 00 0C 80

LOAD SECTOR COUNTER
SET 'ACCESS READY'
CLR ALL BUT DRIVE SELECT

10
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.

I CALL TEMP ERR REG (NODO) & READ EXT RKER
I CONTROL RESET ?
I YES, DO DISPATCH
I ANY HARD ERRORS ?
I YES, THEN ABORT
I HARD ERRORS ?
I YES, THEN GO TO NODO
I UPDATE EXT RKER
I COPY INTERNAL RKCS TO LO & HI
I UPDATE EXT RKCS
I READ EXT RKER
I ... A COPY TO INTERNAL RKWC
I READ EXT RKBA
I ... A COPY TO INTERNAL RKBA & UNIBUS MA
I UPDATE INT RKBA TO TRANSFER END
I AGAIN, SINCE RKIC IS A WORD CNT
I DISPATCH

IFUNCTION IS LOW 3 BITS OF RP
I JMP TO * + RP*
I WRITE
I READ
I WRITE CHECK
I SEEK
I READ CHECK
I ORIVE RESET
I WRITE LOCK

I WRITE OPERATION
I MASK OUT ALL BIT NEW EXT BITS
I SET A17, A16, C1, C0
I READ SECTOR COUNTER
I SEEK TO SPECIFIED CYLINDER
I STORE FIFO REGISTERS
I LOAD FIFO WITH 2 "FE" BYTES
I ASSERT WRITE ENABLE & ERASE ENABLE
I CMTR FOR PREAMBLE BYTES
I LOOP TILL FIFO READY FOR MORE
I FEED FIFO ANOTHER BYTE
I DEC PREAMBLE BYTE CMTR
I ITERATE TILL CMTR GOES TO 0

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

I WAIT FOR FIFO
I DISPATCH SYNC BIT
I CALL ALL BUT CYLINDER RITS
I WAIT FOR FIFO
I DISPATCH 1ST HEADER BYTE
I REMOVE DRIVE 86L BITS
WAIT FOR FIFO
DISPATCH END HEADER BYTE

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

WRITE INNER LOOP

INITIATE UNIBUS TRANSFER, WAIT FOR FIFO & UBUS
COPY DATA TO FIFO & BUILD CHECKSUM
JUMP MP, CARRY INTO HI CHECKSUM BYTE ?
YES, SO INC CNK
WAIT FOR FIFO
COPY DATA & BUILD CHECKSUM
UNIBUS WC EXHAUSTED ?
IND, THEN WRITE ANOTHER WORD TO DISK
DISK WC ALSO EXHAUSTED ?
IND, THEN WRITE "O"S TILL IT IS

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

WAIT FOR FIFO
DISPATCH LO CHK BYTE

POSTAMBLE

WAIT FOR FIFO
DISPATCH A PIECE OF THE POSTAMBLE
MORE POSTAMBLE TO COME ?
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.

FIFO OUTPUT REG EMPTY ?
MAKE SURE WE DIDN'T SEE BETWEEN BYTE GLITCH

"NYTSEC" WILL DISABLE THE WRITE OR READ CURRENTS AND CHECK IF MORE
DATA IS TO BE READ OR WRITTEN BEYOND THE SECTOR THAT WAS 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.

TRUMP DA TO NEXT SECTOR, DISABLE CURRENTS
COPY JUST SECTOR TO RB
OVERFLOW TO NEXT TRACK ?
IND, THEN GO TO "MORE"
NEXT TRACK, SECTOR 0
JCARRY OUT OF RDAO ?
YES, SO RUMP RDA;
ANY MORE TO TRANSFER ?
IND, THEN WE'RE ALL DONE
JNEED SEEK TO NEXT CYLINDER ?
IND, PRETEND SEEK JUST COMPLETED

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

COPY SECTOR BITS TO RB
LEGAL SECTOR NUMBER ?
IND, THEN TAKE ERROR EXIT
COPY CYL BITS TO LO, HI
LOAD LOW CYL REG
THE SEEK HAS JUST BEEN INITIATED. NOW LOOP ON CHECKING FOR SEEK ERROR OR SEEK DONE.

INITIATE SEEK

GET ERROR BITS
DROP CYCLE
IF ERR, GO TO 'UNSAFE'
INCREASE SECTOR COUNTER
IS BUSY SEEKING?
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.

FORMAT READ?
YES, SO BYPASS VERIFICATION
WAIT FOR SECTOR PULSE
LOAD DELAY COUNTER
WAIT FOR PREAMBLE TO GET UNDER HEADS
WAIT LOOP
RESTORE FIFO REGISTERS
REINSERT READ ENABLE
WAIT FOR 1ST HEADER BYTE
LOW CYL ADDR OK?
NO, THEN SEEK ERROR
WAIT FOR 2ND HEADER BYTE
HIGH CYL ADDR OK?
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).

CLR CHECKSUM REGISTERS
1256, WORD DISK TRANSFER
ASSUME ALL OF SECTOR WANTED
FULL SECTOR TRANSFER?
SET UNIBUS WORD COUNT
CLR INTERNAL MKRC (LAST SECTOR)

SELECT HEAD

SELECT HEAD SPECIFIED IN INTERNAL MKRC.

COPY HEAD SEL BIT TO 'LD'
SELECT HEAD
RETURN

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

MARK PASSAGE OF ANOTHER WORD TO DISK
MARKDONE WHEN DONE
WAIT FOR FIFO
DISPATCH A ZERO
WAIT
A8E 00 A0 F9 0C 00
A8B 00 A0 F9 02 05

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

0BC 00 A0 F9 0C 32 ; SET A17, A14, C1, & C0
0BD 00 A0 F9 01 06
0BE 00 A0 F9 0C 08
0BF 0D A0 F9 0C 00
0C0 00 A0 F9 0C RF
0C1 00 A9 0A 0A 0C 00
0C2 00 A0 F9 0C 04
0C3 00 A0 F9 00 C3 C3
0C4 00 A0 F9 0C 00
0C5 00 A0 F9 0C 02
0C6 0D A0 F9 0C 03
0C7 00 A0 F9 00 0F
0C8 00 A0 F9 00 03 DF
0C9 00 A0 F9 00 02 DF
0CA 00 A0 F9 03 DF
0CB 25 E1 28 00 00

; DISK READ INNER LOOP

0CC 00 A0 F9 00 C3 00
0CD 00 A0 F9 03 CC
0CE 00 A0 F9 00 33 DO
0CF 00 E0 50 00 0C 00
0D0 00 A0 F9 00 03 E0
0D1 00 A0 F9 00 00 E0
0D2 00 A0 F9 00 00 E0
0D3 00 A0 F9 00 00 E0
0D4 00 A0 F9 00 00 E0
0D5 00 A0 F9 00 00 E0

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

0D6 00 A0 F9 00 C3 00
0D7 00 A0 F9 00 33 DA
0D8 00 A0 F9 03 DA
0D9 00 A0 F9 00 33 DA
0DA 00 A0 F9 00 00 E0
0DB 00 A0 F9 00 00 E0
0DC 00 A0 F9 00 00 E0
0DD 00 A0 F9 00 00 E0
0DE 00 A0 F9 00 00 E0

; READ A WRITE CHECK TRANSFER TO "RICKT", NOW TEST TO SEE WHICH IT IS AND BRANCH ACCORDINGLY.

0DF 00 A0 F9 00 0C 04
0E0 00 A0 F9 00 03 EA

; READ CHECK INNER LOOP

0E1 00 A0 F9 00 C3 E1
0E2 00 A0 F9 00 C3 E0
0E3 00 A0 F9 00 33 E0
0E4 00 A0 F9 00 C3 E0
0E5 00 A0 F9 00 33 E0
0E6 00 A0 F9 00 C3 E0
0E7 00 A0 F9 00 C3 E0
0E8 00 A0 F9 00 33 E0
0E9 00 A0 F9 00 33 E0

; WRITE CHECK INNER LOOP

0EA 00 A0 F9 00 0C 00
0EB 00 A0 F9 00 03 00
0EC 00 A0 F9 00 00 0C 00
0ED 00 A0 F9 00 00 0C 00
0EE 00 A0 F9 00 00 0C 00
0EF 00 A0 F9 00 00 0C 00

14
OF6 80 80 OF 00 LF 98
OF1 14 46 OF 00 OF 00
OF2 00 00 OF 01 03 OF
OF3 00 00 19 00 OF 00
OF4 00 00 OF 00 05 EA
OF5 00 00 OF 00 OF 02 FA

I WAIT FOR DISK DATA
{{{SUB BYTE FROM DISK
{{ERROR IF NOT $8
{{RESTART UNIBUS WORD COUNT
{{LOOP TILL DONE
{{JOIN FORMAT READ STREAM

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

0F6 11 E1 80 00 OF 00
0F7 00 00 OF 00 03 F9
0F8 2D 80 80 00 OF 40
0F9 00 00 OF 01 81 07
0FA 00 00 OF 00 81 70
0FB 00 00 OF 00 02 RE

I INTERNAL RKCH NEEDS UPDATE
I UNDO WHAT "SEEK" DID
I TRANSFER HEADER
I IF MORE, CONT TO NEXT SECTOR
I PREPARE READ STREAM

1 SEEK ROUTINE --- SINCE SEVERAL RKCH DRIVES ARE MAPPED INTO THE PERTEC
1 DRIVE, IT IS BEST TO SEEK ONLY BEFORE PERFORMING A DATA TRANSFER.

0FC 75 4F 00 00 OF 2A
0FD 00 80 OF 00 02 FA
I ISET "SEARCH COMPLETE" IN RKCS
I WE'RE DONE!

1 DRIVE RESET --- RECALIBRATE & BRANCH TO "SEEK"

0FE 80 80 OF 01 81 24
0FF 00 80 OF 00 02 FC
I RECALIBRATE
I JOIN UP WITH "SEEK"

1 UNIBUS DATA TRANSFER SUBROUTINES

100 00 80 OF 39 OF 36
101 00 80 00 04 OF 0A
102 OF 00 00 OF 01 63 08
103 10 4A 80 00 OF 00
104 10 80 74 00 OF 00
105 00 48 OF 00 8A 0B

I INITIAL DATA IN ENTRY POINT
I INITIALIZE TRANSFER
I WAIT FOR UNIBUS
I GET LOW DATA BYTE
I ... & HIGH DATA BYTE
I RETURN

106 80 40 OF 39 OF 36
107 10 86 80 00 OF 00
108 10 87 80 00 OF 00
109 80 80 00 04 OF 00
10A 80 80 OF 01 63 0A
10B 80 80 OF 00 8A 08

I INITIAL DATA OUT ENTRY POINT
I COPY LOW DATA
I ... & HIGH DATA
I INITIALIZE TRANSFER
I WAIT FOR "TRANSFER DONE"
I RETURN

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

10C 10 40 OF 00 OF OF 01
10D 80 80 OF 01 02 13
10E 10 40 OF 00 OF 02
10F 00 80 OF 01 03 13
110 10 40 OF 00 OF 20
111 00 80 OF 01 02 13
112 10 40 OF 00 OF 40
113 69 80 1F 01 02 1A
114 84 80 00 OF OF 02
115 00 80 OF 01 03 12
116 10 4F 80 OF 00 00
117 00 80 OF 01 02 19
I ISET "WE" IN RKER0
I ISET "CE" IN RKER0
I ISET "CSE" IN RKER0
I ISET "WS" IN RKER0
I ISET "NXS" IN RKER0
I ISET "XC" IN RKER0
I ISET "XCS" IN RKER0
I ISET "XCE" IN RKER1
I ISEEK INCOMPLETE?
I YES, THEN PRETEND "NXC"
I ISET "FAC" IN RKER1
116 1D A9 1F 00 00 00 00
117 00 A0 00 00 00 00 00
118 00 00 00 00 00 00 00 00
119 00 00 00 00 00 00 00 00
120 00 00 00 00 00 00 00 00
121 00 00 00 00 00 00 00 00
122 00 00 00 00 00 00 00 00
123 00 00 00 00 00 00 00 00
124 00 00 00 00 00 00 00 00
125 00 00 00 00 00 00 00 00
126 00 00 00 00 00 00 00 00
127 00 00 00 00 00 00 00 00
128 00 00 00 00 00 00 00 00
129 00 00 00 00 00 00 00 00
130 00 00 00 00 00 00 00 00
131 00 00 00 00 00 00 00 00
132 00 00 00 00 00 00 00 00
133 SET 'SKE' IN RKER
134 CLR RKER
135 STOP ON SOFT ERROR?
136 YES
137 DEC ERR CNTER, TIME TO GIVE UP?
138 NO, SO TRY AGAIN
139 YES
140 READ EXTERNAL RKER
141 UPDATE OLD RKER
142 REINIT RAKER TO 177402
143 WRITE UPDATED RKER
144 SET 'ERROR' IN RKC81
145 SOFT ERROR?
146 YES
147 NO, SO SET 'ME' IN RKC81
148 RETURN IN DISGRACE
149 RECALIBRATE
150 ... & TRY AGAIN
151 ASSERT 'RESTORE'
152 ASSERT 'STBCYL'
153 PAUSE
154 REMOVE 'STBCYL'
155 LOOP TILL NOT 'BUSY'
156 'BUSY'?
157 YES
158 EXIT