SOFTWARE CONTROL OF THE DISK II OR IWM CONTROLLER

PREPARED BY

N I

APRIL 26, 1984

REVISION 1, MAY 10, 1984

SOFTWARE CONTROL OF THE DISK II OR IWM CONTROLLER

Each of the eight expansion slots of the Apple II computer has the exclusive use of sixteen memory locations for I/O control. These memory locations can be used as software switches. A software routine can instruct the interface card in a particular slot to perform a predefined hardware task by toggling a software switch. Whenever the Apple II addresses one of the sixteen I/O locations allocated to a particular slot, the signal on pin 41 of that slot, called DEVICE SELECT/, switches to the active low state. This signal, in conjunction with the four low-order address lines (A0 - A3), can be used to enable logic in the peripheral card to perform a particular task. The following table illustrates the memory space for the sixteen I/O locations for each expansion slot.

TABLE 1: PERIPHERAL CARD I/O SPACE

SLOT	LOCATIONS
0	$C080 - $C08F
1	$C090 - $C09F
2	$C0A0 - $C0AF
3	$C0B0 - $C0BF
4	$C0C0 - $C0CF
5	$C0D0 - $C0DF
6	$C0E0 - $C0EF
7	$C0F0 - $C0FF

One common method of accessing peripheral card I/O control softswitches through software is to use the Apple's (6502) indexed addressing mode. For example, a LDA $C080, X instruction can be used to access softswitch 0 in any slot, if the index register X is loaded with a value equal to the slot number times sixteen.

Sixteen peripheral I/O addresses are used to control the functions of the disk II controller. The following table illustrates the functions of the sixteen software switches.

TABLE 2: DISK II CONTROLLER SOFTSWITCHES

ADDRESS	FUNCTION
$C080, X	PHASE 0 OFF
$C081, X	PHASE 0 ON
$C082, X	PHASE 1 OFF
$C083, X	PHASE 1 ON
$C084, X	PHASE 2 OFF
$C085, X	PHASE 2 ON
$C086, X	PHASE 3 OFF
$C087, X	PHASE 3 ON
$C088, X	TURN MOTOR OFF
$C089, X	TURN MOTOR ON
$C08A, X	SELECT DRIVE 1
$C08B, X	SELECT DRIVE 2
$C08C, X	Q6L
$C08D, X	Q6H
$C08E, X	Q7L
$C08F, X	Q7H

The index register X has the value of slot number times 16. The last four addresses have the following functions.

Q6	Q7	FUNCTION
L	L	READ
H	L	SENSE WRITE PROTECT OR PREWRITE STATE
L	H	WRITE
H	H	WRITE LOAD

In general, any valid 6502 instruction can be used to access the above soft-switch address, except a load instruction is used to read a byte of encoded data from the controller and a store instruction is used to write a byte of encoded data to the controller. Below are typical examples demonstrating the use of the disk II controller softswitches. It is assumed that both Q6 and Q7 are low at the beginning of the read/write examples.

SELECT DRIVE

```
LDA   $C08A, X          SELECT DRIVE 1
LDA   $C08B, X          SELECT DRIVE 2
```

The hardware design of the controller allows only one drive to be selected at one time. A LDA $C08A, X instruction will select drive 1 and deselect drive 2. A LDA $C08B, X instruction will select drive 2 and deselect drive 1.

MOTOR ON

```
LDA   $C089, X          TURN MOTOR ON
LDA   $C088, X          TURN MOTOR OFF
```

It should be noted that there is only one interface signal (ENABLE/), going from the controller to each floppy disk drive, which is used to enable the drive's read/write function and to turn on the motor. Both the select drive and the motor on instructions must be executed in order to activate the ENABLE/ signal of a particular drive. A typical program will select the drive first and then turn on the motor at a later time. After the completion of the motor on instruction, the program should wait at least 1 second for the motor to come up to speed, before read/write functions can be performed reliably.

The disk II controller hardware will keep the ENABLE/ signal to its active low state for approximately one second after the execution of the motor off instruction, therefore read/write can be performed reliably within this period. To be on the safe side, the program should verify that the motor is spinning by monitoring the change in data pattern read from the drive. This delay in turning off the motor facilitate rapid and repeat access to the same drive.

SENSE WRITE PROTECT

```
LDA   Q6H, X      WRITE PROTECT SENSE MODE
LDA   Q7L, X      READ CONTROLLER STATUS REG.
BMI   WRPROT      BRANCH IF BIT 7 OF STATUS REG. IS HIGH
```

The above instruction will load the content of the controller's status register into the accumulator. Bit 7 of the status register is the write protect flag. A "one" in bit 7 of the status register indicates that a write protected diskette is inserted into the drive. A BMI instruction will check the write protect flag. The program will branch to the WRPROT address label if the write protect flag is set.

READ A DATA BYTE

```
        LDA   Q7L, X     MAKE SURE IN READ MODE
LOOP    LDA   Q6L, X     READ THE BYTE
        BPL   LOOP       STAY IN THE LOOP IF THE M.S. BIT IS LOW
          .
          .
          .
          .
LP      LDA   Q6L, X     READ ANOTHER BYTE
        BPL   LP         SAY IN THE LOOP IF THE M.S. BIT IS LOW
          .
          .
```

NOTE: THERE SHOULD BE NO PAGE CROSSING FOR THE BPL INSTRUCTIONS.

The LDA Q7L, X instruction makes sure that the controller is in read mode. The LDA Q6L, X instruction loads the contents of the controller's data shift register into the accumulator. Since the Apple GCR code requires that the most significant bit of every encoded data byte is high, the BPL instruction will force the program to stay in a two instruction loop until the M.S. bit of the controller data shift register is high. At the beginning of every byte time, the controller internal logic will clear the data shift register. As data bits are shifted into the data shift register, the M.S. bit of the register is high, if and only if a full byte of data is assembled in the register. The data byte will stay in the register for a little more than 7 us. Therefore, it is important to make sure that the BPL instruction does not cross the page boundary. This is necessary to ensure that the execution time of the two instruction loop (LDA, BPL) is no more than 7 us.

WRITE A DATA BYTE

```
        LDA   Q6H, X     GO TO
        LDA   Q7L, X     PREWRITE STATE
        LDA   DATA
        STA   Q7H, X     PARALLEL LOAD DATA INTO CONTROLLER
        LDA   Q6L, X     CONTROLLER SHIFT DATA OUT SERIALLY
   EXECUTION TIME         .
OF THESE INSTRUCTIONS     .
   MUST BE EXACTLY        .
   32 CLOCK CYCLES        .
        STA   Q6H, X     PARALLEL LOAD ANOTHER BYTE
        LDA   Q6L, X     SHIFT OUT DATA
          .
          .
          .
        LDA   Q7L, X     OUT OF WRITE MODE
        LDA   Q6L, X     TO READ MODE
```

NOTE: It is important to write a garbage byte (HEX FF) before turning off the write mode, so that the drive electronics has enough time to write the last valid data byte.

The first two instructions force the controller into the prewrite state. These are the same instructions to sense the write protect flag. It is important to execute these two instructions even the programmer does not want to sense the write protect flag. The STA instruction loads the contents of the accumulator into the controller's data shift register. The next instruction [LDA Q6L,X] causes the data in the register to shift out serially. Q6H and Q7H are the conditions required for parallel loading the data into the controller. Shifting out the data serially to the disk drive requires Q6L and Q7H. The first STA instruction sets Q7 high, because the conditions are Q6H and Q7L before executing this instruction. The conditions are Q6L and Q7H before the second STA instruction, therefore the second STA instruction sets Q6H.

The execution time of the instructions between the end of two consecutive parallel load instructions [STA] has to be exactly 32 clock cycles, otherwise invalid data will be written on the diskette. In order to calculate the execution time, it is important to note that the 6502 processor requires one additional execution cycle for branching or indexing operations crossing the page boundary. The program should switch the controller back to the read mode after all the data has been written.

WRITE SELF SYNC BYTE

```
                    LDA   Q6H, X
                    LDA   Q7L, X
                    LDA   #$FF
                    STA   Q7H, X     PARALLEL LOAD AUTO SYNC BYTE
                    ORA   Q6L, X     START TO SHIFT OUT AUTO SYNC BYTE
  EXECUTION TIME      .              ORA IS USED SO THAT LDA  #$FF IS NOT
OF THESE INSTRUCTIONS .              NEEDED TO WRITE THE NEXT SYNC CYCLE
  MUST BE EXACTLY     .
  40 CLOCK CYCLES
                    STA   Q6H, X     PARALLEL LOAD ANOTHER AUTO SYNC BYTE
                    ORA   Q6L, X     SHIFT OUT SYNC BYTE
  SELF SYNC BYTE      .
  40 CLOCK CYCLES     .
                      .
                    LDA   DATA
                    STA   Q6H, X     LOAD FIRST DATA BYTE
                    LDA   Q6L, X     SHIFT OUT DATA BYTE
    DATA BYTE         .
  32 CLOCK CYCLES     .
                      .
                    STA   Q6H, X
                    LDA   Q6L, X
                      .
                      .
                      .
                    LDA   Q7L, X     OUT OF WRITE MODE
                    LDA   Q6L, X     TO READ MODE
```

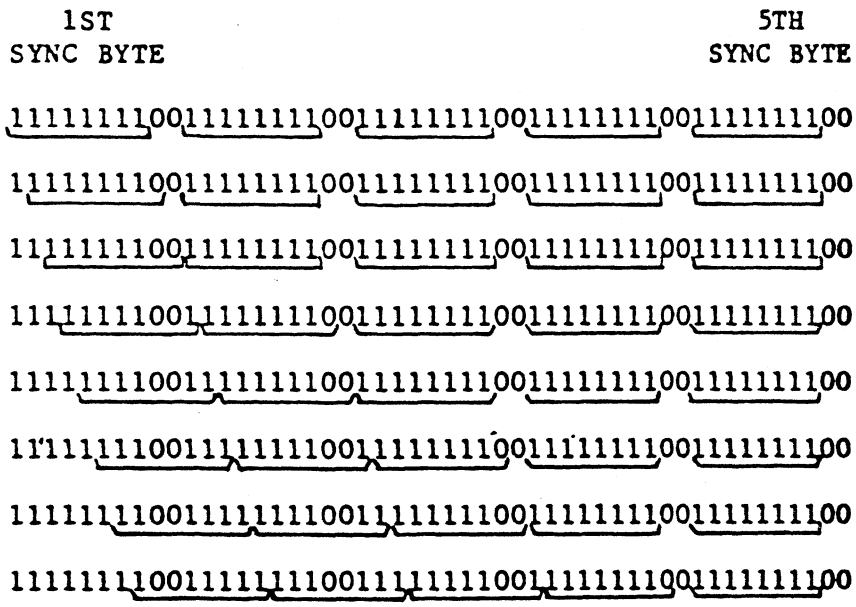
NOTE: Write a garbage byte (HEX FF) before turning off write mode.

The Apple GCR code uses a unique synchronization technique to determine the byte boundary. A self sync byte consists of eight bits of "1" and two bits of "0." The procedure to write a sync byte is the same as to write a data byte, except that the execution time of the instructions between the end of two consecutive parallel load sync byte instructions has to be exactly 40 us. During the write sync byte time, the processor loads eight bits of "1" into the controller. After shifting out 8 bits of "1", the controller hardware will shift out "0" until the next parallel load instruction. Since there are 40 us between two consecutive parallel load instructions and a 4 us bit time, 8 bits of "1" from the processor and 2 bits of "0" appended by the controller hardware are shifted out to the drive. It is necessary to write at least five self sync bytes at the beginning of both the address and data field.

READ SELF SYNC BYTE

Due to the Apple GCR code's unique synchronization technique, the controller hardware will determine the byte boundary automatically. The following is a brief description of the Apple synchronization technique.

FIGURE 1: SYNCHRONIZATION PROCESS



In the above diagram, each row of brackets represent what the controller will send out to the Apple II should the controller start reading at any given bit in the first self sync byte. The controller groups the self sync read data stream into 8-bit byte with a "1" in the most significant bit of each byte. Any "0" bit between bytes are dropped out. From the above diagram, it is shown that the controller is able to group the data at the correct byte boundary within five byte time after the beginning of the read. This is always true for any bit position to start the reading. Therefore, a minimum of five self sync bytes are required for the controller to sync on the read data. After the fifth self sync byte, the controller has established the byte boundary and is able to read the data following the sync bytes correctly. The "D5 AA 96" and "D5 AA AD" address mark sequences follows the self sync bytes in the address and data field respectively. It is not necessary to read and verify the sync byte. In order to read/write a sector, the program should look for the "D5 AA 96" sequence which are the address mark bytes for the address field.

The D5 and AA patterns are reserved for address mark. These patterns are not used to encode data. Therefore, byte syncronization for the address field is achieved by searching for the "D5 AA 96" sequence. Byte synchronization for the data field is done by looking for the "D5 AA AD" sequence.

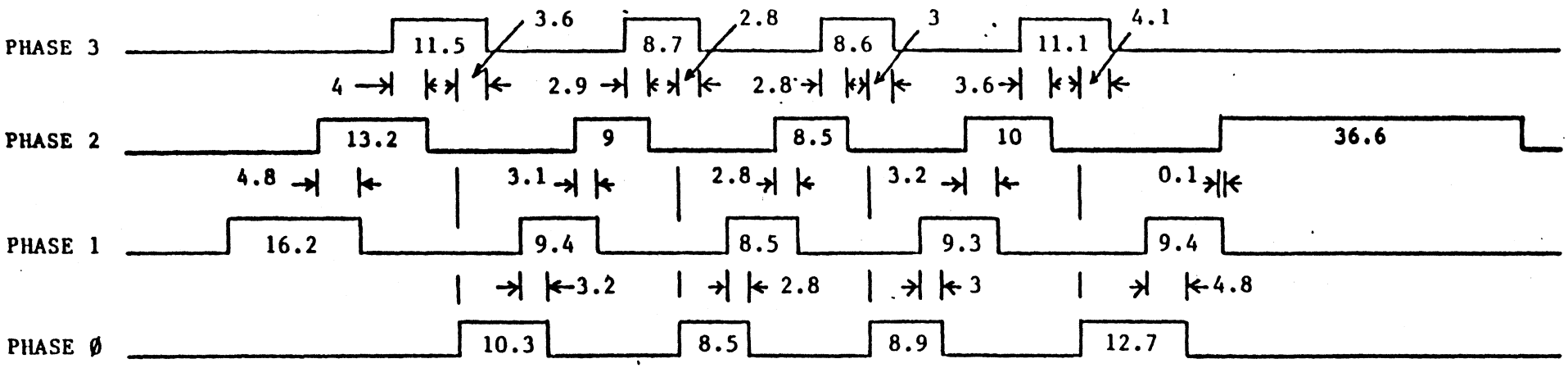
SEEK TO ANOTHER TRACK

The stepper motor in the Disk II is a four phase stepper motor. Eight I/O control softswitches are used to toggle the four phase on and off as shown in table 2. Two adjacent phases have to be activated in sequence in order to move the R/W head to the adjacent track. Activating the phases in ascending order (0, 1, 2, 3, 0, 1, ...) moves the head towards (inward) the center of the diskette. The head moves away (outward) from the center of the diskette when the phases are activated in descending order (3, 2, 1, 0, 3, 2, ...). All even numbered tracks are positioned under phase 0 and all odd numbered tracks are under phase 2. In order to step in a track, the phase 1 and then phase 2 have to be activated in sequence from an even numbered track, while the phase 3 and then phase 0 is activated in sequence from an odd numbered track. The phase 3 and then phase 2 sequence is used to step out a track from an even numbered track. For stepping out a track from an odd numbered track, the phase 1 and then phase 0 sequence is used. The spindle motor should be on for 150 ms before starting the seek operation. The following is an example to step in a track from an even numbered track.

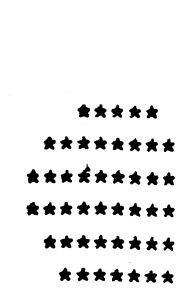
```
LDA   $C083, X   TURN ON PHASE 1
  .
  .
11.5 msec delay loop
  .
  .
  .
LDA   $C085, X   TURN ON PHASE 2
  .
  .
 0.1 msec delay loop
  .
  .
  .
LDA   $C084, X   TURN  OFF PHASE 1
  .
  .
36.6 msec delay loop
  .
  .
  .
LDA   $C086, X   TURN OFF PHASE 2
```

The above programming example is used to illustrate the timing required to step in a track from an even numbered track. The user may use a indexed look up table for the parameter required for different delay loops. No matter how many tracks to step, the user has to allow the last phase to be on for 36.6 msec, because this timing includes the head settling time requirement (25 ms) of the drive. For long seek (step a number of tracks), two adjacent phases can overlap the phase on time in order to increase the torque of the stepper motor and to reduce the seek time. Since the timing between the phase ON/OFF time is critical, it is recommended that the user calls upon the SEEK routine in the Apple DOS for seeking. Figure 2 shows the waveforms of the phases to seek from track 0 to track 9.

FIGURE 2: PHASE WAVEFORMS TO SEEK FROM TRACK 0 TO TRACK 9



NOTE: ALL THE NUMBERS SHOWN IN THE DIAGRAM ARE IN MILLISECONDS.



DATE: May 29, 1984

TO: Distribution

FROM: N L N.L.

SUBJECT: Software Routine to Determine A State Machine or IWM Controller

The attached software routine will determine whether a state machine disk II controller or an IWM controller is installed in the system. This routine may be useful for programmer writing copy protection codes for Apple II.

Upon exiting from the routine, the IWM controller will restore back to synchronous mode (line no. 46 and 47), since all our current applications of IWM for Apple II is in that mode. Future application of the IWM may require the asynchronous mode, the user should set up the mode register (line 46) accordingly.

NL:npk
Attachment

DISTRIBUTION

M A
P B
C C
B G
J S
R W

```
    CE   FILE #01 =>IWM
0000:                1 *
0000:                2 *
0000:                3 * THIS ROUTINE WILL DETERMINE WHETHER A STATE MACHINE DISK II CONTROLLI
0000:                4 * OR AN IWM CONTROLLER IS INSTALLED IN THE SYSTEM.
0000:                5 * UPON EXIT FROM THE ROUTINE, Y=1 MEANS IWM CONTROLLER AND Y=0 MEANS
0000:                6 * STATE MACHINE DISK II CONTROLLER.
0000:                7 *
0000:                8 * ASSUME A MOTOR OFF INSTRUCTION (LDA $C088,X) HAS BEEN EXECUTED FOR
0000:                9 * TWO SECONDS BEFORE THE USER CALLS ON THIS ROUTINE.  OTHERWISE, A
0000:               10 * TWO SECOND DELAY LOOP MUST BE ADDED AFTER THE FIRST MOTOR OFF
0000:               11 * INSTRUCTION (LDA $C088,X)  AT THE BEGINNING OF THIS ROUTINE.
0000:               12 *
0000:               13 * THE ENABLED DISK DRIVE WILL CONTINUE TO BE ON FOR 1 SEC
0000:               14 * AFTER EXIT FROM THIS ROUTINE.
0000:               15 *
0000:               16 *
----- NEXT OBJECT FILE NAME IS IWM.0
1000:       1000    17          ORG   $1000
1000:AE 41 10       18          LDX   SLOTX16       ;X REG=SLOT NO. X 16
1003:BD 88 C0       19          LDA   $C088,X       ;MOTOR OFF
1006:A0 00          20          LDY   #00           ;CLEAR REG. Y
1008:BD 8D C0       21          LDA   $C08D,X       ;Q6H
100B:BD 8F C0       22          LDA   $C08F,X       ;Q7H, ADDRESS MODE REG.
1[illegible]:A9 04  23          LDA   #$04
     .9D 8F C0      24          STA   $C08F,X       ;DISABLE TIMER BIT IN MODE REG.
1013:BD 8E C0       25          LDA   $C08E,X       ;Q7L, OUT OF WRITE MODE
1016:BD 89 C0       26          LDA   $C089,X       ;MOTOR ON
1019:48             27 LOOP     PHA                  ;10 MSEC DELAY LOOP
101A:68             28          PLA                  ;WAIT FOR THE MOTOR ON
101B:48             29          PHA                  ;SIGNAL TO BE ACTIVE IN
101C:68             30          PLA                  ;THE CONTROLLER CARD
101D:48             31          PHA
101E:68             32          PLA
101F:48             33          PHA
1020:68             34          PLA
1021:48             35          PHA
1022:68             36          PLA
1023:C8             37          INY
1024:D0 F3   1019   38          BNE   LOOP          ;END OF DELAY LOOP
1026:BD 8E C0       39          LDA   $C08E,X       ;Q7L, READ STATUS REG.
1029:9D 88 C0       40          STA   $C088,X       ;MOTOR OFF
102C:29 1F          41          AND   #$1F          ;MASK 5 L.S. BITS
102E:C9 04          42          CMP   #$04          ;CHECK TIMER BIT
1030:D0 09   103B   43          BNE   DISKII        ;DISK II CONTROLLER IF NOT EQ
1032:C8             44 IWM      INY                  ;INCREMENT Y REG.
1033:BD 8F C0       45          LDA   $C08F,X       ;Q7H, ADDRESS MODE REG.
1036:A9 00          46          LDA   #$00
1038:9D 8F C0       47          STA   $C08F,X       ;Q7H, RESTORE MODE REG.
1[illegible] BD 8E C0       48 DISKII   LDA   $C08E,X       ;Q7L,
1[illegible] .BD 8C C0      49          LDA   $C08C,X       ;Q6L, RESTORE TO READ MODE
1041:       1041    50 FIN      EQU   *
1041:60             51 SLOTX16  DFB   $60
```

```
1800 DISKII          ?1841 FIN          ?1832 IWM          1819 LOOP
1841 SLOTX16
** SUCCESSFUL ASSEMBLY := NO ERRORS
*  ASSEMBLER CREATED ON 15-JAN-84 21:28
    TAL LINES ASSEMBLED    51
   REE SPACE PAGE COUNT    89
```