Date: July 15, 1985

Author: Fern Bachman

Subject: Mouse ERS

Document Version Number: 00:10

Revision History

00:00 Initial Release
00:10 Soft switches to talk to the mouse have changed

General

The Columbia mouse uses front desk bus to communicate with the Columbia keyboard micro which in turn informs the monitor firmware of mouse actions. This is a departure from the AppleMouse card and the //c mouse interface which depends extensively on firmware to support the mouse. Columbia's mouse has a true passive mode like the AppleMouse but unlike the //c mouse which requires interrupts to do anything. The true passive mode is definitely an advantage. It allows mouse applications to operate while devices which have timing critical loops or which can't run if interrupted to operate. An interrupting communications card is a good example of a device which must be the only interrupting device in a system to operate. The true passive mode also prevents the 65816 from being overburdened with interrupts from the mouse such as can occur in the //c if someone is moving the mouse rapidly while an application program is running.

Columbia's mouse can cause an interrupt to the 65816 only if an interrupt mode was selected, the mouse is on, and the interrupt condition has occurred. Unlike the //c which interrupts whenever the mouse is moved, Columbia's mouse interrupts in sync with the Vertical Blanking signal (VBL) of Columbia only. The maximum times that Columbia's mouse can interrupt the 65816 is 60 times per second. This cuts down on the overhead the mouse puts on the 65816. If an interrupt condition (determined by the mode byte setting) occurs an interrupt will be passed to the 65816 when the next VBL occurs. This is consistent with the operation of the AppleMouse card. This function is accomplished by sending a command to the KeyGlu chip to tell it to interrupt the 65816 in sync with VBL only if a predefined condition exits.

The FDB mouse (Columbia's mouse) when giving position data returns a delta position of up to +/- 63 counts (approximately 0.8 inches of travel). It is up to Columbia's firmware to convert this relative position (delta) to absolute position. The FDB mouse also provides the following features;

1. Current button0 and button1 data (1 if down, 0 if up)
2. Previous button0 and button1 data (1 if was down, 0 if was up)
3. Interrupt data (whether VBL, button0/1, or movement int)

At power up the FDB mouse, via the KeyGlu chip defaults to a mouse off, non interrupt state for safety. Reset will also cause the KeyGlu chip to turn the FDB mouse interrupt off and enter a non interrupt state.

Addresses Used

Address	Function
$C027	KeyGlu status register defined as follows bit 0 = d Must NOT be altered by mouse bit 1 = 0 'X' - position available (read only) = 1 'Y' - position available (read only) bit 2 = k Must NOT be altered by mouse bit 3 = k Must NOT be altered by mouse bit 4 = d Must NOT be altered by mouse bit 5 = d Must NOT be altered by mouse bit 6 = 1 Mouse interrupt enable (read/write) bit 7 = 1 Mouse register full (read only) k = used by keyboard handlers d = used by FDB handlers
$C024	Mouse data register 1st read yields 'X' position data and button 1 data 2nd read yields 'Y' position data and button 0 data

Program Restrictions/Requirements

1. To enable mouse interrupts set bit 6 of $C027 to 1.
2. To determine if an interrupt came from the mouse or not the user reads bit 7 of $C027 and bit 6 of $C027. If both are a 1 then an interrupt is pending from the mouse.
3. To read the mouse position the following conditions must occur or the data is contaminated and corrective measures must be taken;
   1. Read bit 7 of $C027 -
      If bit 7 = 0 then 'X' and 'Y' data is not available yet
      If bit 7 = 1 then data is available but may be contaminated
   2. Read bit 1 of $C027 only if bit 7 was a 1
      If bit 1 = 0 then 'X' and 'Y' data is not contaminated and may be read. The first read of $C024 returns 'X' data and button 1 data and the second read of $C024 returns 'Y' data and button 0 data. Be careful of indexed instructions. The false reads and writes that some indexed instructions do can cause the 'X' data to be lost and the 'Y' data to appear when 'X' data was wanted.
      If bit 1 is 1 and you have not read $C024 yet then the data in $C024 is contaminated and must be considered useless. If condition occurs do the following

1. Read $C024 one time only
2. Ignore the byte read in
3. Exit your mouse read routine without updating the 'X' and 'Y' or button data. This will not hurt any program and will guarantee that the next time you read the mouse positions everything should be accurate.

3. To data read in is encoded as follows

'X' data byte -

bit 7 = 0 then mouse button 1 is up
1 then mouse button 1 is down
bits 0-6 delta mouse move
bit 6=0 then a positive move up to $3F (63) ticks
bit 6=1 then negative move in two's complement up to $40 (64)

'Y' data byte -

bit 7 = 0 then mouse button 0 is up
1 then mouse button 0 is down
bits 0-6 delta mouse move
bit 6=0 then a positive move up to $3F (63) ticks
bit 6=1 then negative move in two's complement up to $40 (64)

4. The main screen holes can be in either bank $00 or bank $E0 depending on whether shadowing is on or off. If it is on the screen holes used will be in bank $00. If shadowing is off the screen holes used will be in bank $E0.

The mouse is resident in Columbia in internal slot 4. When the mouse is in use the main memory screen holes for slot 4 are used by the mouse to hold X and Y absolute position data, current mode, button0/1 status, and interrupt status. Eight other bytes are needed for mouse information storage. They will hold the maximum and minimum clamps for the mouse's absolute position. Following is a break down of screen hole usage by the mouse when Columbia firmware is used ONLY.

Main Memory Screen Holes Used

Address	Use
$47C	Low byte of absolute X position
$4FC	Low byte of absolute Y position
$57C	High byte of absolute X position
$5FC	High byte of absolute Y position
$67C	Reserved and used by firmware
$6FC	Reserved and used by firmware
$77C	Button0/1 / interrupt status
	Bit 7 = Currently button0 is up/down (0/1)
	Bit 6 = Previously button0 was up/down (0/1)
	Bit 5 = X/Y moved since last READMOUSE
New --->>	Bit 4 = Currently button1 is up/down (0/1)
	Bit 3 = VBL interrupt
	Bit 2 = Button0/1 interrupt
	Bit 1 = Movement interrupt
New --->>	Bit 0 = Previously button1 was up/down (0/1)

Button0/1-Interrupt Status Byte

7	6	5	4	3	2	1	0
Current Button0 Status	Previous Button0 Status	X/Y move since last read	Current Button1 Status	VBL int occurred	Button int occurred	Movement int occurred	Previous Button1 Status

$7FC Mode byte
Bit 7 = Reserved
Bit 6 = Reserved
Bit 5 = Reserved
Bit 4 = Reserved
Bit 3 = Interrupt on VBL
Bit 2 = Interrupt on next VBL if button pressed
Bit 1 = Interrupt on next VBL if mouse moved
Bit 0 = Mouse off/on (0/1)

Mode Byte

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	VBL int mode	Button int mode	Movement int mode	Mouse off/on

Firmware RAM Used

The mouse clamps are kept in the auxiliary screen holes in these locations.

Address	Use
$E0xxxx	Low byte of low X clamp
$E0xxxx	High byte of low X clamp
$E0xxxx	Low byte of high X clamp
$E0xxxx	High byte of high X clamp
$E0xxxx	Low byte of low Y clamp
$E0xxxx	High byte of low Y clamp
$E0xxxx	Low byte of high Y clamp
$E0xxxx	High byte of high Y clamp

These locations are NEVER to be changed by the user directly. They are to be changed only via CLAMPMOUSE.

Firmware calls

To use the mouse firmware the user must enter via the user interface provided below. It conforms to the PASCAL 1.1 protocol for peripheral cards.

Location	Routine	Definition
$C40D	PINIT	PASCAL init device (Not implemented)
$C40E	PREAD	PASCAL read character (Not implemented)
$C40F	PWRITE	PASCAL write character (Not implemented)
$C410	PSTATUS	PASCAL get device status (Not implemented)
$C411 = $00		Indicates that more routines follow

Standard routines implemented on Columbia (same as //c and AppleMouse card)

$C412	SETMOUSE	Set mouse mode
$C413	SERVEMOUSE	Service mouse interrupt
$C414	READMOUSE	Read mouse position
$C415	CLEARMOUSE	Clear mouse position to 0 (For delta mode)
$C416	POSMOUSE	Set mouse position to user defined position
$C417	CLAMPMOUSE	Set mouse bounds in a window
$C418	HOMEMOUSE	Set mouse to upper left corner of clamping window
$C419	INITMOUSE	Reset mouse clamps to defaults, positions to 0,0

In addition to the above the following locations are significant.

$C400	BINITENTRY	Initial entry point when coming from BASIC
$C405 = $38	BASICINPUT	(opcode SEC) PASCAL I.D. byte BASIC input entry point
$C407 = $18	BASICOUTPUT	(opcode CLC) PASCAL I.D. byte

	BASIC output entry point
$C40B = $01	PASCAL generic signature byte
$C40C = $20	Apple Tech Support I.D. byte
$C4FB = $D6	Additional I.D. byte

PASCAL firmware calls

Enough of the firmware has been set up so that PASCAL recognizes the mouse as a valid device, however PASCAL is not directly supported by the firmware. A PASCAL driver for the mouse is available from Apple to interface programs with the mouse. The standard PASCAL calls PINIT, PREAD, PWRITE and PSTATUS if called will return with the X register set to 3 (PASCAL illegal operation error) and carry set.

PINIT

Function:	Not implemented (just an entry point in case user calls it by mistake.)
INPUT :	All registers and status bits don't care
OUTPUT :	X = $03 -- Error 3 = Bad mode: Illegal operation
	C = 1 -- Always
	Screen holes: Unchanged

PREAD

Function:	Not implemented (just an entry point in case user calls it by mistake.)
INPUT :	All registers and status bits don't care
OUTPUT :	X = $03 -- Error 3 = Bad mode: Illegal operation
	C = 1 -- Always
	Screen holes: Unchanged

PWRITE

Function:	Not implemented (just an entry point in case user calls it by mistake.)
INPUT :	All registers and status bits don't care
OUTPUT :	X = $03 -- Error 3 = Bad mode: Illegal operation
	C = 1 -- Always
	Screen holes: Unchanged

PSTATUS

Function:	Not implemented (just an entry point in case user calls it by mistake.)
INPUT :	All registers and status bits don't care
OUTPUT :	X = $03 -- Error 3 = Bad mode: Illegal operation
	C = 1 -- Always
	Screen holes: Unchanged

Assembly language firmware calls

The Apple // mouse standard interface as pointed out earlier is taken from the PASCAL firmware interface protocol. To use a mouse routine from assembly language the user must first read the location corresponding to the routine he wants to call. The value he reads is the offset to the actual entry point of the routine to be called. This interface guarantees that programs written for the Apple // family of mice will be able to get to the required firmware routine the same way in all machines. It allows Apple to update the firmware routines without obsoleting any software applications. It also means that programs written for one machine will work on the entire family of Apple // computers assuming the program does not make use of any machine specific hardware.

The firmware calls available on Columbia, the AppleMouse card and the //c are:

NOTES:

1. n = slot number mouse is in --- n = 4 for Columbia and //c mice
2. These bits are not changed by mouse firmware: e,m,I,x, direct register, data bank register, program bank register.
3. The user MUST NOT change any mouse screen hole at any time except during POSMOUSE when the user is required to put new mouse coordinates directly in the screen holes. No other mouse screen hole can be changed without seriously affecting the mouse.
4. Interrupts MUST be disabled before each mouse call.
5. If shadowing is on the screen holes in bank $00 are used. If shadowing is off the screen holes in bank $E0 are used.

SETMOUSE

Function: Set mouse operation mode
Input: A = mode ($00 to $0F only valid modes)
X = Cn for std interface (Columbia mouse doesn't care)
Y = n0 for std interface (Columbia mouse doesn't care)
Output: A = mode if illegal mode entered - else A is scrambled
X,Y,V,N,Z = scrambled
C = 0 if legal mode entered (mode is <= $0F)
C = 1 if illegal mode entered (mode is > $0F)
Screen holes: Mode byte updated only

SERVEMOUSE

Function: Tests for interrupt from mouse and resets mouses interrupt line.
Input: A,X,Y = don't care
Output: A,X,Y,V,N,Z = scrambled
C = 0 if it was a mouse interrupt
C = 1 if it was not a mouse interrupt
Screen holes: Interrupt status bits updated to show current status

READMOUSE

Function: Reads delta X/Y positions, updates absolute X/Y positions, and reads button statuses from FDB mouse.

Input:
A = don't care
X = Cn for std interface (Columbia mouse doesn't care)
Y = n0 for std interface (Columbia mouse doesn't care)

Output:
A,X,Y,V,N,Z = scrambled
C = 0 -- Always
Screen holes: XLO, XHI, YLO, YHI, Buttons and movement status bits updated -- Interrupt status bits are cleared

CLEARMOUSE

Function: Resets to 0, X and Y, the buttons, movement and interrupt status. This mode is intended for use when the user wants to do delta mouse positioning instead of absolute positioning which is normal.

Input:
A = don't care
X = Cn for std interface (Columbia mouse doesn't care)
Y = n0 for std interface (Columbia mouse doesn't care)

Output:
A,X,Y,V,N,Z = scrambled
C = 0 -- Always
Screen holes: XLO, XHI, YLO, YHI, Buttons / movement / interrupt status set to 0

POSMOUSE

Function: Allows user to change current mouse position.

Input: User places new absolute X/Y positions directly in appropriate screen holes.
A = don't care
X = Cn for std interface (Columbia mouse doesn't care)
Y = n0 for std interface (Columbia mouse doesn't care)

Output:
A,X,Y,V,N,Z = scrambled
C = 0 -- Always
Screen holes: User changed X and Y absolute positions only bytes changed

CLAMPMOUSE

Function: Set up clamping window the mouse should use. Power up defaults are 0 to 1023 ($0000 - $03FF).

Input:
A = 0 if entering X clamps
A = 1 if entering Y clamps
Clamps are entered in slot 0 screen holes by the user as follows:

$478 = low byte of low clamp
$4F8 = low byte of high clamp
$578 = high byte of low clamp
$5F8 = high byte of high clamp

X = Cn for std interface (Columbia mouse doesn't care)

Y = n0 for std interface (Columbia mouse doesn't care)

Output: A,X,Y,V,N,Z = scrambled
C = 0 -- Always
Screen holes: X / Y absolute position set to upper left hand corner of clamping window. Clamping RAM values in bank $E0 are updated.

TAKE NOTE:

This means that the Columbia mouse does an automatic HOMEMOUSE after a CLAMPMOUSE. This is not done by either the AppleMouse card or the //c. It is highly recommended that a HOMEMOUSE follow a CLAMPMOUSE when dealing with the //c or the AppleMouse. The execution of an automatic HOMEMOUSE is required due to the fact that deltas are being fed to the firmware instead of +/-1's as is the case for the //c and for the 6805 microprocessor on the AppleMouse card. The delta from Columbia's FDB mouse can alter the absolute position to a point where the clamping techniques used by the other 2 mice are useless for Columbia.

HOMEMOUSE

Function: Sets X / Y absolute position to upper left hand corner of clamping window.

Input: A = don't care
X = Cn for std interface (Columbia mouse doesn't care)
Y = n0 for std interface (Columbia mouse doesn't care)

Output: A,X,Y,V,N,Z = scrambled
C = 0 -- Always
Screen holes: X / Y absolute position changed

INITMOUSE

Function: Sets screen holes to defaults, and sets clamping window to default of 0000-1023 ($0000,$03FF) in both the X and Y directions. Resets KeyGlu mouse interrupt capabilities.

Input: A = don't care
X = Cn for std interface (Columbia mouse doesn't care)
Y = n0 for std interface (Columbia mouse doesn't care)

Output: A,X,Y,V,N,Z = scrambled
C = 0 -- Always
Screen holes: X/Y positions, button statuses, interrupt status set to 0

As with the other Apple // mice the X and Y positions, button statuses and movement status are guaranteed to be valid only after a READMOUSE and, if the application program is to be compatible with the //c, only until 65816 interrupts are enabled again. Interrupt status bits are guaranteed to be valid only after a

SERVEMOUSE and, if the application program is to be compatible with the //c, only until 65816 interrupts are enabled again. Interrupt status bits are reset after a READMOUSE. The user should, read and use, or read and save, the appropriate mouse screen hole data before enabling or reenabling 65816 interrupts.

Standard Firmware Call Example

Note: Interrupts must be disabled on every call to the mouse firmware.

```
SETMOUSEOFF EQU $Cn12        ;Offset to SETMOUSE offset ($C412 for Columbia)

            LDA SETMOUSEOFF  ;Get offset into code
            STA TOMOUSE+1    ;Modify operand
            LDX Cn           ;Where Cn = C4 in Columbia
            LDY n0           ;Where n0 = 40 in Columbia
            PHP              ;Save interrupt status
            SEI              ;Guarantee no interrupts during call
            LDA #$01         ;Turn mouse passive mode on
            JSR TOMOUSE      ;JSR to a modified JMP instruction
            BCS ERROR        ;C = 1 if illegal mode entered error
            PLP              ;Restore interrupt status
            RTS              ;Exit

ERROR       PLP              ;Restore interrupt status
          JMP ERRORMESSGE ;Exit to error routine

TOMOUSE     JMP $Cn00        ;Modified operand for correct entry point
                              $C400 for Columbia
```

BASIC Firmware Entry

The mouse and BASIC have the following interface. To turn the mouse on do the following.

```
1. PRINT CHR$(4);"PR#4" :REM Ready mouse for output
2. PRINT CHR (1) :REM Send the mouse a 1 to turn it on from BASIC
3. PRINT CHR$(4);"PR#0" :REM restore screen output. NOTE use
   PRINT CHR$(4);"PR#3" to return to 80 columns
```

Whenever the mouse is turned on from BASIC to accept outputs, firmware changes the output hooks at $36 and $37 to point to $C407 and does an INITMOUSE which is described above.

To turn the mouse off do the following.

```
1. PRINT CHR$(4);"PR#4" :REM Ready mouse for output
2. PRINT CHR (0) :REM Send the mouse a 0 to turn it off from BASIC
3. PRINT CHR$(4);"PR#0" :REM restore screen output. NOTE use
   PRINT CHR$(4);"PR#3" to return to 80 columns
```

To read mouse position and button statuses from BASIC do the following.

```
1. PRINT CHR$(4)"IN#4" :REM Ready mouse for input
2. INPUT X,Y,B :REM Input mouse position
3. PRINT CHR$(4)"IN#0" :REM Return keyboard as input device when
done reading mouse positions
```

Whenever the mouse is turned on from BASIC to do inputs, the firmware changes the input hooks at $38 and $39 to point to $C405. When an INPUT statement is invoked while talking to the mouse the firmware does a READMOUSE before converting the screen hole data to decimal ASCII and placing it in the input buffer beginning at $200.

In BASIC the mouse runs in passive mode (that is a non interrupt mode), its clamps are set automatically to 0000-1023 ($0000-$03FF) in both the X and Y directions and position data in the screen holes is set to 0. During a BASIC INPUT statement the firmware reads the deltas from the FTD mouse adds them to the absolute position in the screen holes (clamping the positions if necessary) and then converts the absolute positions in the screen holes to ASCII and places that data with the button0 status data into the input buffer followed by a carriage return and returns to BASIC. Button1 status cannot be returned to BASIC since doing so would add another input variable to the intput buffer which will result in an ?EXTRA IGNORED error being printed in existing mouse BASIC programs. A BASIC program wanting to read button1 status can PEEK the screen hole containing that data. The data returned in the input buffer is in the following form

```
s  x1  x2  x3  x4  x5  ,  s  y1  y2  y3  y4  y5  ,  sb  B0  b1  cr

        s = sign of absolute position
  x1...x5 = 5 ASCII characters giving the decimal value of X
  y1...y5 = 5 ASCII characters giving the decimal value of Y.
       sb = - if key on keyboard was pressed during input statement
            + if none was pressed
       B0 = ASCII space character
       b5 = 1 if button0 is pressed now and was pressed on last INPUT
            statement
          = 2 if button0 is pressed now but was not pressed on last INPUT
            statement
          = 3 if button0 is not pressed now but was pressed on last INPUT
            statement
          = 4 if button0 is not pressed now and was not pressed on last INPUT
            statement
       cr = Carriage Return - Required as terminator before passing
            control form firmware back to BASIC
```

It is up to the BASIC program to reset the key strobe at $C010 if sb comes back negative. A POKE 49168,0 will reset the strobe.