Apple Slot Finder

by Steven Weyhrich
2419 N. 92nd Ave. Apt. 19
Omaha, NE 68134

INTRODUCTION

This program is a modification of one I found in the only issue I've ever seen of CONTACT, the Apple User's Group Newsletter. In issue #6 of October 1979 there was a program entitled "CONFIG", which checked the Apple's peripheral slots to determine what was plugged into them. At that time I had no real application for the routine used in that program. Recently, however, I have been working on a set of programs that I wanted to be portable between Apples with different printer and slot configurations. Specifically, I wanted the program to automatically select the slot the printer was plugged into, so the user wouldn't need to know what a "slot" is; when he wanted to print, the program would print. SLOT FINDER is a general application of the routine altered for my own use. **As it appears in Listing 1, it will identify the Apple Silentype printer card, the Apple Serial Interface card, the Apple Disk II Controller card, the Hayes Micromodem II card**, and for those with an Apple /// running under Apple II Emulation mode, the Emulation Communications card and the Emulation Serial Interface card.

USING SLOT-FINDER

Enter the program from Listing 1 and run it. If your Apple has any of the above mentioned peripheral cards plugged into it, the program will identify the card as it scans each slot. **If you have a card that the program does not allow for, the program will declare that slot empty, even if there really is a card plugged in there. Continue reading, and you will learn how to use the BYTE FINDER subroutine for the specific configuration of your Apple.**

```
]RUN
SLOT 1 IS EMPTY
SLOT 2 IS EMPTY
SLOT 3 IS EMPTY
SLOT 4 IS EMPTY
SLOT 5 IS EMPTY
SLOT 6 HAS A DISK CONTROLLER CARD
SLOT 7 IS EMPTY

]
```

THEORY OF OPERATION

On an Apple with empty slots, a **PR#s** (where s = 1 to 7, corresponding to slots 1 to 7) to an empty slot causes the computer to "hang" until the RESET key is pressed. An IN#s to an empty slot causes you to get dumped into the Monitor. This is because the Apple tries to redirect its output hooks (with a PR#s) or its input hooks (with a IN#s) to the program that starts at memory address **$Cs00.** If there is no peripheral card plugged in, there is no ROM or RAM at that memory location, and 'crash' goes the computer. To avoid this in real life programming applications, it is useful to know which slots are empty and which are not. One way to do this is to have the program execute some lines like this:

```
100 INPUT "WHICH SLOT FOR PRINTER? ";SL
110 PRINT CHR$(4);"PR#"SL
```

This requires the user to know which slot has a printer in it. Alternately, the program can have on disk a file which contains the number of the printer slot, and the computer will appear to automatically select the right slot. The problem with this is if the file was set up for a differently configured Apple, the program will crash when it tries to execute a PR# to an empty slot. Then there is the SLOT FINDER solution. During each run the Apple will learn which slot contains the printer card and will always PR# to the right slot.

The SLOT FINDER subroutine first examines each page of slot memory to determine the presence or absence of a card. To do this, it jumps across the memory of each 256 byte slot and PEEKS at the value of every 64th byte, summing these four bytes into the array CS (checksum). This is done three times. On the Apple II, memory locations without RAM or ROM return pseudo-random numbers when PEEKed. To view this for yourself, enter the Monitor and list part of the memory of an empty slot:

```
CALL -151
*C700.C71FF          (for slot 7)
```

Now do it again. Notice that **one four line group is different from the other.**

Do the same for a slot that does contain a card. This time the two groups of lines should match, byte for byte at each address. If this is done on an Apple /// in its Apple II emulation mode, the numbers returned for an empty slot are all $FF (255 decimal). In the SLOT FINDER subroutine, if the three check sum values match, the slot is occupied; if they don't match, or if the sum reveals each checked byte to be $FF, the slot is empty.

IDENTIFYING THE CARD

The next part of the subroutine, once it finds a filled slot, is to identify the peripheral card. **Since each card has its own unique assembly language routine, checking the same two of three relative bytes in each different card and comparing them to known values makes it possible to uniquely identify the card.**

The accompanying program in **Listing 2, BYTE FINDER,** is designed to aid in locating these unique bytes. The addresses used in the original CONFIG program were the 5th and 7th bytes of each card, but that will not allow differentiation between an Apple Serial Interface card and a Silentype printer card, so I chose to use the 10th and 15th bytes of each slot. The subroutine uses a seven element array called SLOT, and after RETURNing, each element of SLOT holds a number representing which card was found in that slot. If no match was found, a zero is returned. So, (in this example) if the variable SLOT (4) = 5, slot 4 of your Apple contains a Hayes Micromodem II, since that was the 5th device defined in the initialization part of the driver program.

CONFIGURING YOUR OWN SYSTEM

In making your own program to drive the SLOT FINDER subroutine, several variables must be set before doing a GOSUB to it. The variables **C000, C100,** and **C700** contain the decimal values for the hex numbers **$C000, $C100,** and **$C700** respectively.

N represents the number of cards you will define for your program to identify.

R1 and R2 represent the relative bytes being checked in each slot.

Four arrays are used by the subroutine: B1 and B2, each DIMensioned to size N, holding the known bytes at relative addresses R1 and R2 respectively for each card; CS, the checksum array; and SLOT, the array which on returning from the subroutine holds the information gathered by it.

The string array NAME$ is not used by the subroutine; it is used in this example to list the cards identified by the subroutine.

To facilitate the identification of bytes and relative addresses unique to your Apple peripheral cards, Listing 2 is a program called **BYTE FINDER**. This program will ask for which relative byte to display, and then do a continuously updated listing of that relative byte for all seven slots. When running this on an Apple II, **the empty slots will have different bytes each time the slots are scanned.** If you run it on an Apple /// in Apple II Emulator mode, the empty slots will show with a 255.

By examining the two screens which follow, you can see that between the two timeframes in which the slots were scanned, the values changed in all except Slots 1, 2, 4 and 6. These values, which in a different system represent different printers, a clock card, and the disk controller, can then be used in the Slot Finder to identify the specific peripheral cards.

```
TYPE ■ TO QUIT
BYTE # (0 - 255) 1
SLOT   ADDRESS   BYTE #   VALUE
----   -------   ------   -----
  1      49408        1     176
  2      49664        1      48
  3      49920        1     123
  4      50176        1     160
  5      50432        1     249
  6      50688        1      32
  7      50944        1     173

HIT ANY KEY TO CHANGE BYTE #
```

```
TYPE ■ TO QUIT
BYTE # (0 - 255) 1
SLOT   ADDRESS   BYTE #   VALUE
----   -------   ------   -----
  1      49408        1     176
  2      49664        1      48
  3      49920        1     160
  4      50176        1     160
  5      50432        1     160
  6      50688        1      32
  7      50944        1     165

HIT ANY KEY TO CHANGE BYTE #
```

MODIFICATIONS

The SLOT FINDER subroutine is completely relocatable, containing no GOTOs or GOSUBs. If you are sure that you won't be using it on an Apple /// emulating Applesoft, the last three comparisons in the IF statement in line 350 may be deleted. Line 290 is the FOR . . . NEXT loop that skips over the slot memory. As it stands, every 64th byte is PEEKed. If you want more certainty that the routine is really identifying empty slots, the STEP value in this line can be decreased; however, this will increase the execution time the subroutine takes to check all seven slots, since it does more PEEKing. Conversely, if you need more speed, you can try increasing the STEP value to a number greater then 64. Just be sure that it is still working in this altered mode before you use it in your prize program.

Another alteration which can be made is the way in which the subroutine saves what it learns about the slot configuration. In its present form, the subroutine identifies slots by which cards (if any) are plugged into them. By making the following changes to Listing 1, the subroutine will instead identify peripheral cards by the slot they are plugged into.

Modify these lines:

```
80  Replace SLOT(7) with CARD(N)
380 Replace SLOT(SLOT) = I with CARD(I)
      = SLOT
```

Replace these lines:

```
170 FOR I = 1 TO N
180 PRINT "THE "NAME$(I)" IS ";
190 IF CARD(I) = 0 THEN PRINT "NOT
      PRESENT"
200 IF CARD(I) < > 0 THEN PRINT "IN
      SLOT "CARD(I)
240 FOR I = 1 TO N:CARD(I) = 0: NEXT I
```

I hope you'll find SLOT FINDER a useful addition to your library.

LISTING 1

```
]LIST
10  REM      **********************
11  REM      *     SLOT FINDER     *
12  REM      * BY STEVEN WEYHRICH  *
13  REM      * COPYRIGHT (C) 1983  *
14  REM      * BY MICROSPARC, INC  *
15  REM      * LINCOLN, MA. 01773  *
16  REM      **********************
30  REM   ADAPTED FROM PROGRAM "CONFIG" IN CONTACT #6,
      APPLE USER'S GROUP NEWSLETTER OF OCTOBER 1979
40  REM   IDENTIFIES SLOTS BY WHICH CARDS ARE PLUGGED
      INTO THEM
50 C000 = 49152:C100 = 49408:C700 = 50944: REM MEMORY
      ADDRESSES
60 N = 6: REM NUMBER OF CARDS DEFINED
70 R1 = 10:R2 = 15: REM RELATIVE BYTE IN EACH SLOT
80  DIM B1(N),B2(N),NAME$(N),CS(2),SLOT(7)
90 B1(1) = 130:B2(1) = 120:NAME$(1) = "SILENTYPE PRINT
      ER CARD"
100 B1(2) = 120:B2(2) = 072:NAME$(2) = "SERIAL PRINTER
      CARD"
110 B1(3) = 036:B2(3) = 060:NAME$(3) = "DISK CONTROLLE
      R CARD"
120 B1(4) = 038:B2(4) = 072:NAME$(4) = "COMMUNICATIONS
      CARD"
130 B1(5) = 255:B2(5) = 007:NAME$(5) = "HAYES MICROMOD
      EM II"
140 B1(6) = 038:B2(6) = 197:NAME$(6) = "EMULATION SERI
      AL CARD"
150  GOSUB 240: REM CHECK THE SLOTS
160  REM REPORT ON RESULTS OF SEARCH
170  FOR I = 1 TO 7
180  PRINT "SLOT "I;
190  IF SLOT(I) = 0 THEN  PRINT " IS EMPTY"
200  IF SLOT(I) > 0 THEN  PRINT " HAS A "NAME$(SLOT(I
     )
210  PRINT : NEXT I
220  END
230  REM   ********************* SLOT FINDER SUBROUTI
     NE *********************
240  FOR I = 1 TO 7:SLOT(I) = 0: NEXT I
250  FOR S = C100 TO C700 STEP 256
260 SLOT = (S - C000) / 256: REM IDENTIFY THE SLOT #
270  REM MAKE 3 PASSES OVER SLOT MEMORY
280  FOR K = 0 TO 2:CS(K) = 0
290  FOR I = 0 TO 255 STEP 64
300 CS(K) = CS(K) +  PEEK (S + I)
310  NEXT I: NEXT K
320  REM NOW CHECK IF SUM FROM EACH PASS
330  REM IS THE SAME; IF NOT, OR IF ALL
340  REM BYTES ARE $FF, THEN SLOT IS EMPTY
350  IF CS(0) ( ) CS(1) OR CS(0) ( ) CS(2) OR CS(1) (
     ) CS(2) OR CS(0) = 1020 OR CS(1) = 1020 OR CS(2)
     = 1020 THEN 400: REM EMPTY SLOT
360  REM IDENTIFY THE CARD
370  FOR I = 1 TO N
380  IF  PEEK (S + R1) = B1(I) AND  PEEK (S + R2) = B2
     (I) THEN SLOT(SLOT) = I:I = N: REM A MATCH; TERMI
     NATE LOOP
390  NEXT I
400  NEXT S: REM CHECK THE NEXT SLOT
410  RETURN
```

LISTING 2

```
]LIST
10  REM      **********************
11  REM      *     BYTE FINDER     *
12  REM      * BY STEVEN WEYHRICH  *
13  REM      * COPYRIGHT (C) 1983  *
14  REM      * BY MICROSPARC, INC  *
15  REM      * LINCOLN, MA. 01773  *
16  REM      **********************
40 C000 = 49152:C100 = 49408:C700 = 50944
50 KBD =  - 16384:STR =  - 16368
60  TEXT : HOME
70  FOR K = 0 TO 1:K = 0
80  VTAB 3: PRINT "TYPE ";: INVERSE : PRINT "Q";: NORMAL
     : PRINT " TO QUIT": PRINT
90  INPUT "BYTE # (0 - 255) ";BYTE$
100  IF BYTE$ = "Q" THEN  VTAB 22: END
110 BYTE =  VAL (BYTE$)
120  IF BYTE < 0 OR BYTE > 255 THEN 90
130  VTAB 7
140  PRINT "SLOT  ADDRESS  BYTE #  VALUE"
150  PRINT "----  -------  ------  -----"
160  FOR J = 0 TO 1:J = 0
170  VTAB 10
180  FOR I = C100 TO C700 STEP 256
190 SLOT = (I - C100) / 256 + 1
200 BTE$ =  RIGHT$ ("    " +  STR$ (BYTE),3)
210 VLUE$ =  RIGHT$ ("    " +  STR$ ( PEEK (I + BYTE)),
     3)
220  PRINT "  "SLOT"     "I"     "BTE$ TAB( 25)VLUE$
230  NEXT I
240  PRINT : PRINT : PRINT "HIT ANY KEY TO CHANGE BYTE
     #"
250 X =  PEEK (KBD): POKE STR,0: IF X > 127 THEN J = 1

260  NEXT J
270  NEXT K
```