Now You See It, Now You Don't

Michael R. Mossman
Quispamsis, NB

The secret of "transparent" cartridges on the C64

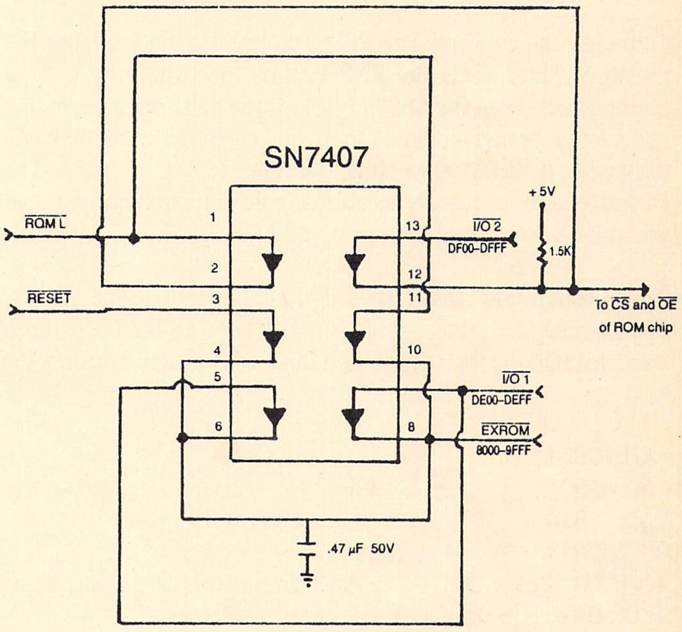
Have you ever wondered why those marvellous cartridges can do things to your computer, but never show up anywhere in memory? The first time I saw Busscard II (an IEEE interface for the C64 that adds basic 4.0 disk commands) at a friends house, it made me curious. I asked to see the machine language code, but we could not find it in memory. This curiosity stayed on the back burner until I bought my Fast Load cartridge. Again, the program could not be found in memory. Overwhelmed, I proceeded to dismantle the cartridge. Inside, I expected to find a maze of modern electronics. Much to my disappointment, there sat two lowly ICs. One was the expected EPROM and the other a 7407. I traced some of the lines but it didn't make much sense. Disappointed, I closed it up and it remained in the back of my computer for over a year.

A few months back, I bought a "Promenade" EPROM programmer to burn a few of my own custom chips. Every now and then I would cast an eye at that Fast Load cartridge, wishing that I could make my cartridges invisible in memory. I revived my attack on that despicable cartridge with renewed vigor. I removed the EPROM from the board and read the program out with my Promenade. I looked at the code and figured that the program ran at $8000 but I knew that the program could not be seen at $8000. This time I sat down and traced out every line on the board and drew a diagram as I went. Low and behold, the secrets were revealed to me.

I would like to point out that this article is not to show you how to copy the Fast Load cartridge. The cartridge is such good value for the money that building one yourself costs more than buying it outright. The code itself is of no use because it will not run by just loading and running it at $8000 – it is much more involved than that. The value lies in being able to put wedges in BASIC and set vectors that are completely transparent to other programs. All this and your program occupies no memory. The memory area at $C000 – $CFFF is fought over by so many programs. There are times when I want the DOS wedge and another program in memory at the same time. This is impossible because they conflict at $C000.

To make your own invisible program, it is necessary to understand the normal control line operation of the expansion port. These are the lines available:

EXROM – This line is normally high (1). To tell the PLA that you want the CPU to read the external rom at $8000, this line is set low.


SN7407
$\overline{\text{ROM L}}$
$\overline{\text{RESET}}$
1
2
3
4
5
6
8
10
11
12
13
$\overline{\text{I/O 2}}$
DF00–DFFF
+5V
1.5K
To $\overline{\text{CS}}$ and $\overline{\text{OE}}$ of ROM chip
$\overline{\text{I/O 1}}$
DE00–DEFF
$\overline{\text{EXROM}}$
8000–9FFF
.47 µF 50V

1) Pin 7 is Ground
2) Pin 14 goes to +5V supply
3) All address and data lines on the ROM or EPROM go to their equivalents on the expansion port.

Figure 1

ROML – This line is a type of decoded address line. When the CPU wants to read the external ROM at $8000, this line is pulled low or 0. ROML will never go low if the EXROM line is not low.

RESET – This line is usually high when the computer is running. Its purpose is to prevent the CPU from trying to execute ML instructions when the computer is cold started. This allows the other chips to reach their "normal" states before the CPU addresses them. RESET is low during reset time. The computer would act flaky without a RESET line. The RESET line goes low in only two normal operations:

1) When the computer is turned on.
2) When the reset button is pressed on the computer.

I/O 1, I/O 2 – These lines are intended for selecting an external I/O device. e.g. Adding an ACIA or a CIA chip. Selection is done by pulling the line low. This is done when you do a read or write to $DE00 – $DFFF. I/O 1 is the area from $DE00 – $DEFF and I/O 2 is $DF00 – $DFFF.

Now, let's look at the invisible cartridge – see Figure 1. The chip is a 7407 hex buffer and so if a low or a 1 is put in, a low or 1 comes out. When the computer is turned on, the RESET line is low. This causes the EXROM line to be low. The line is held low for a period of time, after reset, by the capacitor. When the computer reads the $8000 area it will see the EXROM line in a low state and use ROML to address the external cartridge. If it finds the autostart sequence, it then passes control over to the cartridge code. All this time, the EXROM line has been held low because ROML line is low.

To review the concept: The RESET line starts the sequence but the ROML line holds the EXROM line low while the CPU is reading code from the $8000 block. If the CPU stops executing code for a period of time, then the cartridge at $8000 will disappear (EXROM stays high because ROML is high). The cartridge code sequence is: normal cold start initialization, set your own vectors, and then pass control to BASIC.

The question now arises "How do I get the code to reappear at $8000, now that the cartridge is invisible?". If a read or write is done to $DE00, then the I/O 1 line will go low causing the EXROM line to go low. The capacitor will hold the line low for period of time. Just enough, so that when a read or write is done to $8000, the ROML will be pulled low by the CPU because EXROM is still low. In this case, I/O 1 line starts the sequence but the ROML line, again, holds it.

One of the vectors that you set in the cold start code could point to code in the cassette buffer, the $02A7 – $02FF area, or $C000 block. This code is necessary because it will make the $8000 code visible again. The drawback is that using the above areas is dangerous because other programs like to use these same spots. The answer is in using the I/O 2 line. You will notice from Fig.1 that I/O 2 is connected to the CS (selected by a low) through a buffer. When you do a read of the area from $DF00 – $DFFF you will see code. The magic thing about this code is that it is really located in the rom chip at $9F00 – $9FFF. You appear to see it at I/O 2 area because of the way the chip is selected.

Let's look at how this type of cartridge can be used in your own code. Suppose you would like to implement a wedge in basic. When the machine is turned on, the RESET line is pulled low causing the EPROM at $8000 to appear. The cartridge stays visible because of the capacitor on the EXROM line. The code at $8000 is executed because the key code exists. You make a jump to the $DFF0 area to initialize I/O devices, perform the RAM test, set up page zero kernal locations and then the I/O vectors are set. Chrget and various zero page BASIC pointers and finally the vectors at $0300 – $030B are set. It is here that you can now set the BASIC error vector at $0300 to point to your code at $DF00. When the BASIC interpreter errors out because it does not recognize a command, the error vector will point to your code at $DF00. The code at $DF00 will do a read or write to $DE00. This will cause the EXROM line to go low and the eprom to appear at $8000. You can now jump to your code in the EPROM to check the chrget routine for your command. If it is your wedge then the command is carried out, if not then you jump to the normal error handling routine.

I can see many uses for this type of programing and I think that many of you will also. Included here is a little machine language program that will make the code from the Fast-Load cartridge appear and then store it to normal ram at $8000.

```
 40: 0801              .opt p4
 50: 0801       store  =    $fb        ;address for loop storage
 60: 0814              .bas ml
 90: 0814       ml     =    *
100: 0814 a9 00        lda  #$00       ;set up for read write loop
110: 0816 85 fb        sta  store
120: 0818 a9 80        lda  #$80
130: 081a 85 fc        sta  store+1
140: 081c a0 00        ldy  #$00
150: 081e a2 00        ldx  #$00       ;completely discharge ca-
                                       pacitor for reading eprom
160: 0820       loop   =    *
170: 0820 8e 00 de     stx  $de00
180: 0823 ca           dex
190: 0824 d0 fa        bne  loop
200: 0826       loop1  =    *          ;loop for reading eprom
210: 0826 8c 00 de     sty  $de00
220: 0829 b1 fb        lda  (store),y  ;read eprom
230: 082b 91 fb        sta  (store),y  ;store to ram at same
                                       memory location
240: 082d c8           iny
250: 082e f0 03        beq  add
260: 0830 4c 26 08     jmp  loop1
270: 0833       add    =    *          ;if low byte is zero then
                                       increase high byte by one
280: 0833 e6 fc        inc  store+1
290: 0835 a5 fc        lda  store+1
300: 0837 c9 a0        cmp  #$a0       ;if high byte is equal to
                                       $a0 then end
310: 0839 f0 03        beq  end
320: 083b 4c 26 08     jmp  loop1
330: 083e       end    =    *
340: 083e 60           rts             ;return to basic.program
                                       can now be read with a
                                       monitor from $8000-$9fff
```