Hacking the Apple II+ ROM with ROMX+

One of the most useful features of the ROMX+ for the Apple II/II+ is its ability to have up to 32K of System ROM. Even if you only use the lower 16K of the ROM image, that's a whopping 25% improvement over the original machine's 12K. This is made possible by implementing the INTCXROM softswitch that is used by the Apple IIe and above for overlaying the standard expansion card ROM space with motherboard firmware.

You can use this extra ROM in one of two ways: by either 1) adding your code there and linking to it from the main ROM, or 2) moving parts of the main ROM (e.g. the cassette READ and WRITE routines) into $Cxxx space and using the freed up space in main ROM for your own code. This document will discuss option 1. For an example of option 2, see the MG ROM++ image at https://theromexchange.com/rom_images/romxce.

PART I – BASIC PROCESS

The basic workflow for making changes to the Apple II+ ROM is outlined below:

1. Identify the modification you wish to make and where in the original ROM the patch needs to be inserted. While modifications to the Applesoft interpreter are possible, we will focus only on changes to the Autostart F8 Monitor ROM.
2. After identifying the area of code to be modified, you need to determine the exact addresses to change and what, if any, bytes can be freed up or are available to allow linking to code that will be placed into the $C100-$CFFF block of ROM.
3. Next, you will need to write the actual code that goes into that $Cxxx space to accomplish what you want.
4. To access this code, the $Cxxx space needs to be activated prior to its use and then de-activated after its execution (to allow the slot ROMs to use this space again).

The best place to start is with a clean copy of the Apple II+ Applesoft (or other) image. This will occupy the $D000-$FFFF space of your image. The initial $1000 bytes ($C000-$CFFF) can initially contain any data, but it will be easier to work with if you set all $1000 bytes to $00 or $FF. You can easily create such a clean image with your own machine using these Monitor commands (assuming motherboard ROM is selected):

```
1. 3000:0                    ;initialize the blank area with zero
2. 3001<3000.3FFEM           ;fill the entire block
3. 4000<D000.FFFFM           ;copy existing motherboard ROM to upper part of image
4. BSAVE CLEAN.ROM, A$3000, L$4000       ; save image to disk
```

With this image saved, you can now start making your modifications. As you add more code, you can save the new image and try it out in ROMX+ to verify your progress.

NOTE: Remember that your code will reside in the $C100-$FFFF address range on the motherboard. But as an image in RAM, it will actually be located at $3100-$6FFF. Just mentally make a note that addresses starting with 3xxx=Cxxx, 4xxx=Dxxx, 5xxx=Exxx, and 6xxx=Fxxx. Relative addressing (e.g. branches) will always work correctly, even if the target address shows as the lower one when listing in the Monitor. Other addresses, especially JMPs and absolute addressing to your code will need to specify the real destinations in $C100-$FFFF space.

PART II – SOME SIMPLE EXAMPLES

Adding a cold boot reset command

One of the first hacks I tried using this process was to add a cold boot function to the Apple II+. The Apple IIe and later machines added open-apple and closed-apple keys to the keyboard. One of these was then used to allow cold booting with open-apple-ctrl-reset. Since the II+ does not have these keys (and using the game port buttons is really not practical), a different approach was needed. The simplest thing I could think of was to use the reset key itself. So I decided that a double reset in rapid succession would be used to trigger a coldstart. This turns out to be extremely easy to do.

Here are the steps I followed starting with the clean image as described in Part I. First, I examined the source code (or disassembly) of the Autostart ROM. The obvious patch point seemed to be at $FA82 where the code goes off to generate the boot beep just before it examines the SOFTEV locations to determine whether a cold or warm start will be executed:

```
fa62: d8           RESET      cld                      ;do this first this time
fa63: 20 84 fe                jsr      SETNORM
fa66: 20 2f fb                jsr      INIT
fa69: 20 93 fe                jsr      SETVID
fa6c: 20 89 fe                jsr      SETKBD
fa6f: ad 58 c0                lda      SETAN0          ;AN0 = TTL hi
fa72: ad 5a c0                lda      SETAN1          ;AN1 = TTL hi
fa75: ad 5d c0                lda      CLRAN2          ;AN2 = TTL lo
fa78: ad 5f c0                lda      CLRAN3          ;AN3 = TTL lo
fa7b: ad ff cf                lda      CLRROM          ;turn off extension ROM
fa7e: 2c 10 c0                bit      KBDSTRB         ;clear keyboard
fa81: d8                      cld
fa82: 20 3a ff                jsr      BELL            ;causes delay if key bounces
fa85: ad f3 03                lda      SOFTEV+1        ;is reset hi
fa88: 49 a5                   eor      #$a5            ;a funny complement of the
```

So we start by changing the JSR instruction at $FA82 to point to our modified code instead. What we need to do is execute something like this: activate INTCXROM, execute our code there, and then de-activate INTCXROM. This requires 10 bytes of code that must exist in the standard $D000-$FFFF code space. The original BELL routine only offers 5 bytes so we must look elsewhere. Fortunately, there is a small block of unused code at $FBB4 which is 13 bytes long. So we can place our first two patches here:

```
6A82-   20 B4 FB    JSR   $FBB4     ;remember, this is actually at FA82

6BB4-   2C 07 C0    BIT   $C007     ;enable INTCXROM
6BB7-   20 00 CE    JSR   $CE00     ;access our patch at $CE00
6BBA-   2C 06 C0    BIT   $C006     ;disable INTCXROM
6BBD-   60          RTS
```

Finally, we add our actual code at $CE00:

```
3E00-   EE F4 03    INC   $03F4   ;temporarily force cold start
3E03-   A9 87       LDA   #$87    ;go generate beep
3E05-   20 ED FD    JSR   $FDED
3E08-   CE F4 03    DEC   $03F4   ;restore warm start
3E0B-   60          RTS
```

The purpose of this code is to temporarily set up for a cold boot. Then we generate the beep sound which takes a small amount of time. After generating the beep, the warmstart function is restored. And it all works exactly like the original Autostart code. However, if another reset is seen while the beep is sounding, it will cause the reset routine to execute again, but this time the warmstart flag will no longer be valid. And therefore a cold boot will be performed instead.

While the beep routine is also used to de-bounce any multiple triggers from the reset switch, I did not see any issues with the (albeit shorter) delay of this code in eliminating any key bounce. If there had been an issue, a small delay routine could also have been added before going off to generate the beep.

From the Monitor, we can accomplish the above as follows:

1. BLOAD CLEAN.ROM
2. 6A82:20 B4 FB
3. 6BB4:2C 07 C0 20 00 CE 2C 06 C0 60
4. 3E00:EE F4 03 A9 87 20 ED FD CE F4 03 60
5. BSAVE COLDSTART.ROM, A$3000, L$4000

This file can then be used by ROMX+ to upload into any Bank.

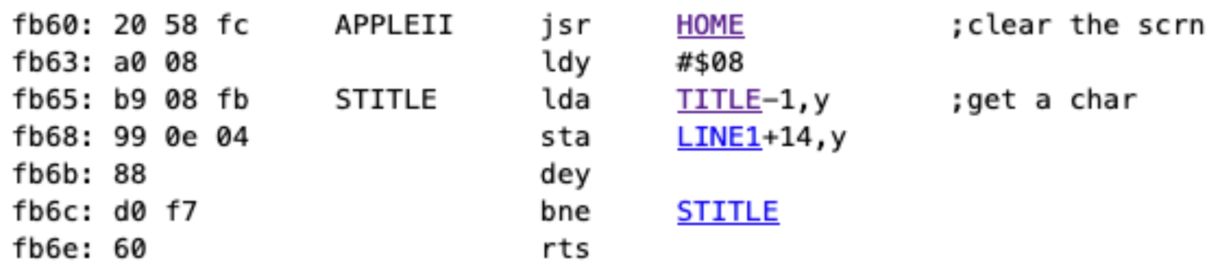
Altering the title screen

Another easy modification is to change the title message. When the Apple II does a cold boot, it clears the screen and then displays “Apple][“ at the top of the screen. One of the first hacks ever done to the Monitor was to alter this message. The simple method only involves changing the bytes starting at location $FB09. This is where the title string is stored in the ROM. More details of this approach can be found at https://www.applefritter.com/content/how-diy-changing-apple-title.

While many users (and clone manufacturers) often did this, there are two major downsides to this approach: 1) you are limited to only 8 characters and 2) early ProDOS versions checked for this exact string and would refuse to boot if not found (to stymie the clone makers). The patch described here will address these issues.

In order to extend the size of the title string we will need to move the entire “APPLEII” Monitor routine into our $Cxxx space along with the actual string data. This gives a huge amount of space to print anything we want on power up. In fact, we could display an entire screen – even graphics – as a splash screen on each cold boot. You could also add sound if so inclined.

All of this is possible using the following code to call our a “boot screen” routine in the new ROM expansion space. Similar to the cold boot reset previously described, we start by modifying the existing ROM code at $FB63:



```
fb60: 20 58 fc     APPLEII     jsr     HOME            ;clear the scrn
fb63: a0 08                    ldy     #$08
fb65: b9 08 fb     STITLE      lda     TITLE-1,y       ;get a char
fb68: 99 0e 04                 sta     LINE1+14,y
fb6b: 88                       dey
fb6c: d0 f7                    bne     STITLE
fb6e: 60                       rts
```

We replace this with a call to our title code, wrapped within BIT instructions to turn on and then off the INTCXROM space:

```
6B63-   2C 07 C0    BIT   $C007     ;enable INTCXROM
6BB6-   20 00 C1    JSR   $C100     ;access our patch at $C100
6BB9-   2C 06 C0    BIT   $C006     ;disable INTCXROM
6BBC-   60          RTS
```

Then we can put whatever we want at $C100 to draw our title or splash screen. A simple 40-character title could be done like this:

```
3100-   A0 28       LDY   #40       ;number of characters to draw
3102-   B9 0B C1    LDA   TITLE-1,Y ;start of string
3105-   99 FF 03    STA   TPLFT-1,Y ;first character position on screen
3108-   88          DEY
3109-   D0 F7       BNE   $3102     ;loop until done
310B-   60          RTS

TITLE
310C-   XX XX XX …  40 ASCII BYTES (HIGH MSB) of TITLE STRING
```

Making a title splash screen

You can extend this idea by creating a complete text screen in BASIC with whatever you want to display. Just add this line to your program to save the screen before exiting:

```
PRINT CHR$(4) “BSAVE SPLASHSCREEN, A$400, L$400”
```

For example, to display a simple bordered message you could use:

```
 10  HOME : PRINT
 20  PRINT " **************************************"
 30  FOR I = 1 TO 19
 40  PRINT " *                                    *"
 50  IF I = 09 THEN  PRINT " *       THIS IS MY APPLE II         *"
 60  NEXT
 70  PRINT " **************************************"
 100  PRINT  CHR$ (4)"BSAVE SPLASHSCREEN, A$400, L$400"
```

Then import this screen data at $3200-$35FF of your image using:

```
BLOAD SPLASHSCREEN, A$3200
```

Finally, change the display routine to:

```
3100-   A2 00       LDX   #$00
3102-   A0 03       LDY   #$03
3104-   BD 00 C2    LDA   $C200,X
3107-   9D 00 04    STA   $0400,X
310A-   BD 00 C3    LDA   $C300,X
310D-   9D 00 05    STA   $0500,X
3110-   BD 00 C4    LDA   $C400,X
3113-   9D 00 06    STA   $0600,X
3116-   BD 00 C5    LDA   $C500,X
3119-   9D 00 07    STA   $0700,X
311C-   E8          INX
311D-   D0 E5       BNE   $3104
311F-   88          DEY
3120-   10 E2       BPL   $3104
3122-   60          RTS
```

While this is a very quick and dirty way to display your splash screen, it also overwrites all of the screen holes. As part of a Reset routine this will be OK, but there are certainly more comprehensive ways to do this if need be.

PART III – ADDITIONAL NOTES

When creating images like this, it is much easier to work in ProDOS. You don't have to worry about patching DOS 3.3 to allow saving 32K binary files. And once you save a binary file using the A and L parameters, you don't need to type them again. Just a simple BLOAD <name> and BSAVE <name> will suffice; that's a great timesaver!

When using the CXROM space for your code, you need to be aware that some software might not be expecting the slot ROMs to be deactivated in an Apple II. This is especially true if the software uses interrupts that could possibly by triggered while you have the INTCXROM active. Although these situations are not very common, the easiest way to avoid such conflicts is to disable interrupts before activating the INTCXROM space. You can then restore the IRQ status when you return after turning off INTCXROM.

Also, while INTCXROM is active, there is the possibility that a Reset will be triggered and this needs to disable INTCXROM. The ROMX+ has logic to detect this and will automatically do so. However, there are some rare situations where this logic can be fooled. Therefore, it is good practice to make sure that the Reset code in your modified images also hit the $C006 softswitch to make sure that INTCXROM is disabled on reset.

Lastly, when changing the Reset handler like we have shown here, be aware of an issue that can arise when there is a 16K RAM or Language Card in the computer. While the IIe and above all disable the upper 16K on Reset, the original Apple II/II+ does not. So if a Reset is generated while a RAM card is active, it will use the reset vector stored in the card's RAM instead of the motherboard ROM. If DOS or ProDOS are loaded however, this is slightly modified using the warmstart vector and will usually work as desired. You can also make a small modification to the RAM card to have it switch off on Reset; this is similar to the modification required by a Language Card to disable its F8 ROM.

Now that you have the tools to create your own custom ROM images, use your imagination to make something interesting, fun, or even useful. The sky's the limit. Or rather, the $C100-$CFFF space is the limit! Once you've created your masterpiece, consider sharing with the ROMX community. Just remember to use the INFO.GEN.16 program to add metadata to your image and then send it to us for posting on our website, theRomExchange.com.