ROMifying a Program for Uploading to ROMX

Converting a program to run under ROMX (or ROMifying) can be a fairly simple task. There's a number of different steps you need to take and they depend upon how the program is written. For this tutorial we will only concern ourselves with machine language programs (those you can BRUN from a disk or otherwise create as direct 6502 code that resides in a specific location in RAM.

Additionally, to work in ROMX the code must fit into the 12K of space from $D000-$FFFF. Larger programs can be handled by splitting across multiple banks but we will not go into that here. If your program uses any F8 monitor ROM routines, then you need to either duplicate them inside your program or include some or all of the $F800-$FFFB space in your image (the RESET vector at $FFFC/D will definitely be changed).

There are two basic ways to ROMify code. The first involves rewriting the code so that it can run from ROM. This would include:

1. Translating and relocating all code that uses absolute addressing.
2. Separating the code and variable storage and finding a suitable location in RAM for the variables.
3. Removing any self-modifying code or references to any standard ROM addresses, e.g. to identify machine or ROM versions.
4. Modifying the exit process when the program is terminated.

While this may seem a bit daunting, there is a second method that is far simpler and fortunately will work for the majority of programs. It works by relocating the entire program to ROM space AS IS with little if any modification. A small loader program is then added that moves the code back into its original RAM location at launch and then jumps to the starting address of the program. The code then runs just as it would as if loaded from disk (except that the standard ROM and possibly DOS might not be there).

For the rest of this tutorial we will describe this second method. We will also make the following assumption: ***that the code is completely loaded into memory at one time and takes up less than $2700 bytes (roughly 10K or 40 sectors on disk)***. This will allow us to keep the entire F8 ROM code intact. This can also help with running and debugging your code since you have the full power of the monitor routines to inspect, break, and test your code. Another advantage of this approach is that you can even test your code in a Language Card or 16K RAM Card (LC) before committing to ROM. More on that later.

The steps to ROMify such code are then quite simple. We just need to relocate the code plus the F8 ROM and the loader program into a single file. The F8 image is modified so that the reset vector points to our loader code. And the loader code is configured with the source starting and ending addresses of the ROM data as well as the destination address in RAM. Finally, we tell the loader where to go after the code has been relocated into RAM.

The Image Tutorial.dsk file that accompanies this document has several files for converting a game to run under ROMX:

```
DISK VOLUME 254

 A 006 HELLO                     Standard Apple Language Card Loader
*B 050 INTBASIC                  Integer Basic to Load into Language Card
*B 034 BREAKOUT                  Original Breakout Game as downloaded from the INTERNET ARCHIVE
 B 050 BREAKOUT.IMG              ROMified image of Breakout
*B 026 CHIPOUT                   Original Chipout Game as downloaded from the INTERNET ARCHIVE
 B 050 CHIPOUT.IMG               ROMified image of Chipout
*B 050 IMAGE TEMPLATE 12K        Template Image for building 12K programs (no F8 Monitor ROM)
*B 050 IMAGE TEMPLATE WITH F8    Template Image for building 10K programs (with F8 Monitor ROM)

INTERNET ARCHIVE URL: https://archive.org/search.php?
query=Breakout_19xx____Breakout_19xx___Hires__Chipout_19xx_Twilight_Software
```

We will start with the IMAGE TEMPLATE WITH F8 file which has the modified F8 code, loader program, and a blank area for the remainder of the 12K image file. You can BLOAD this file, move your program into the blank area, modify the loader parameters, and then BSAVE it back to disk. Or use the Template source code to add into your Assembly programs.

Once you have a complete 12K image you can either Move it from RAM or BLOAD into a 16K LC for testing. Or you can Upload directly into a ROMX bank. We will now show some examples of how this is done. Since we will be testing on a LC first, you can even try this out using an Emulator such as Virtual II for the Mac. By the way, this is exactly how the ROMX firmware was developed. If you are working on real hardware and do not have a 16K RAM card, you can skip those sections and Upload directly into ROMX.

EXAMPLE #1.

Let’s say that we want to Upload to ROMX a Breakout program similar to the original one that came on cassette with the early Apple II machines. Several such programs can be found online and one is included here on the Image Tutorial.dsk as BREAKOUT. We can use a utility such as Copy][Plus to determine the load location and length of BREAKOUT; this turns out to be $800 and $2000 respectively. Since the total size of our program is only 8KB, we can also include the F8 ROM.

So we can now create a ROX image in three easy steps:

```
1. BLOAD IMAGE TEMPLATE WITH F8          (Loads at $1000-$3FFF)
2. BLOAD BREAKOUT, A$1000                (Loads into bottom of image)
3. BSAVE BREAKOUT.IMG, A$1000, L$3000
```

Since our template defaults to a starting address of $800, we don't need to make any modifications to the Loader. This image could now be loaded into ROMX using the Pick command or Manually (after BLOADing at $3000). But let's test it out first in the LC (make sure INTEGER or Applesoft has already been loaded there):

```
CALL -151             (Go into Monitor)
C083 C083             (Enable writing to Language Card)
D000<1000.3FFFM       (Move our image into Language Card)
F700G                 (Execute our Launch code)
```

If all went well, the Launch code would have moved the Breakout program into RAM at $800 and it would have started running.

EXAMPLE #2.

Let's see if we can do the same thing with the CHIPOUT program also found on the tutorial disk. This program loads at $07FD and is $1803 bytes long. So it will certainly fit into our ROM. But since the start address is not $800, we will have to make a couple of changes. After loading the Image Template, we need to BLOAD the program so that the start address is in the first page of the image. The Loader routine only copies full pages of memory at a time (just think about the first two digits of a hex address). So we will end up loading an extra 256-3 or 253 bytes that we will copy back but not really use. In this case, these bytes are in the screen RAM area so when the Loader restores them, we are likely to see some random characters written there just before the program launches. No harm, but watch for it when you launch the image either from the LC or ROM. So here are the steps to add the program to our template:

```
1. BLOAD IMAGE TEMPLATE WITH F8    (Loads at $1000-$3FFF)
2. BLOAD CHIPOUT, A$10FD           (Loads into bottom of image w/offset)
```

Before we save the image, we need to modify the Launch routine:

```
CALL -151             (Go into Monitor)
3706:07               (Change TARGET hi-byte to $07)
370B:FD 07            (Change JMP address to $07FD)
```

Now we can save the image to disk:

```
BSAVE CHIPOUT.ROM, A$1000, L$3000
```

And to test out on the LC:

```
C083 C083                  (Enable writing to Language Card)
BLOAD CHIPOUT.ROM, A$D000  (Move our image into Language Card)
F700G                      (Execute our Launch code)
```

Additional Notes.

When first trying to get a program to run from ROM, it is often helpful to set the RESET vector to $FF59 (the Monitor Entry point, assuming you are keeping the F8 ROM). When the image is loaded using ROMX, it will immediately enter the Monitor (* prompt). Then you can examine the ROM contents, execute the loader routine, and make changes or set breakpoints to the code in RAM.

Also note that the ROM images you create this way can be directly used with most emulators. First you need to extract the image from the Apple environment into a file on the host computer. CiderPress and AppleCommander are two great tools for doing this. Then you tell the Emulator to use this file as it's boot ROM. For example in Virtual II on the Mac, you launch a new machine, click the Setup button, and select ROM memory under the Components section. You can then pick "Use specific ROM" and select your image file (make sure it ends with ".ROM"). When you hit Restart the virtual machine will start up with your image. With Applewin, you can use the -rom option on the command line when you launch the program.

Here's another trick that might be helpful if your program just needs a little more space. If you examine the F8 monitor code, you will find that the first $70 bytes contain routines used to plot graphics. If your program does not use these, then the loader code could be moved here leaving the entire $D000-$F800 space available for your program. The IMAGE TEMPLATE 12K file goes even further giving you almost 12K of program space ($D000-$FEFF) at the expense of the entire Monitor ROM. So make sure your program does not attempt to make any calls there.

Finally, if you make a really cool image that you would like to share with others, please contribute to theRomExchange.com. Before sending however, please add some rmx metadata to describe your image:

```
1. BLOAD <your great image> A,$3000         (Loads at $3000-$5FFF)
2. RUN INFO GEN 3                           (from the ROMX Utilities disk)
3. Enter Description and Additional Info    (Adds metadata to image)
4. BSAVE <image name>.RMX, A$3000, L$3100   (Saves complete rmx image)
```

TEMPLATE LISTINGS.

IMAGE TEMPLATE WITH F8

```
                    ;   ROMX IMAGE TEMPLATE CODE
                    ;   FILE: IMAGE TEMPLATE WITH F8

                RAM_LOC  EQU   $0800
                INIT     EQU   $FB2F

                         ORG   $1000            ;Start of Image

1000: 00 00 00           DS    $3700-*          ;Room for our program

3700: 20 28 F7  Launch   JSR   Setup+$C000
3703: A2 D0              LDX   #/$D000          ;SOURCE hi-byte
3705: A9 08              LDA   #/RAM_LOC        ;TARGET hi-byte
3707: 20 0D F7           JSR   CopyLp+$C000

370A: 4C 00 08           JMP   RAM_LOC          ;Program entry point

370D: 85 03     CopyLp   STA   $03
370F: A9 00              LDA   #$00
3711: 85 00              STA   $00
3713: 85 02              STA   $02
3715: 86 01     CpyLp2   STX   $01
3717: A0 00              LDY   #$00

3719: B1 00     :2       LDA   ($00),Y          ;do 256 bytes
371B: 91 02              STA   ($02),Y
371D: C8                 INY
371E: D0 F9              BNE   :2
3720: E6 03              INC   $03
3722: E8                 INX
3723: E0 FF     SrcEnd   CPX   #/$FF00          ;END OF SOURCE  hi-byte
3725: D0 EE              BNE   CpyLp2           ;do xx pages
3727: 60                 RTS

3728: D8        Setup    CLD                    ;Do what we need to init computer
3729: 20 2F FB           JSR   INIT             ;Set up screen softswitches
372C: 60                 RTS

372D: 00 00 00           DS    $3FFC-*          ;F8 ROM moved into $3800-$3FFB

3FFC: 00 F7              DA    Launch+$C000     ;RESET vector ($FF00)
3FFE: 40 FA              DA    $FA40            ;IRQ vector (not used)
```

IMAGE TEMPLATE 12K

```
                   ;   ROMX IMAGE TEMPLATE CODE
                   ;   FILE: IMAGE TEMPLATE 12K

                RAM_LOC  EQU   $0800           ;Change per program
                INIT     EQU   $FB2F

                         ORG   $1000           ;Start of Image

1000: 00 00 00           DS    $3F00-*         ;Room for our program

3F00: 20 28 FF  Launch   JSR   Setup+$C000
3F03: A2 D0              LDX   #/$D000         ;SOURCE hi-byte
3F05: A9 08              LDA   #/RAM_LOC       ;TARGET hi-byte
3F07: 20 0D FF           JSR   CopyLp+$C000

3F0A: 4C 00 08           JMP   RAM_LOC         ;Program entry point

3F0D: 85 03     CopyLp   STA   $03
3F0F: A9 00              LDA   #$00
3F11: 85 00              STA   $00
3F13: 85 02              STA   $02
3F15: 86 01     CpyLp2   STX   $01
3F17: A0 00              LDY   #$00

3F19: B1 00     :2       LDA   ($00),Y         ;do 256 bytes
3F1B: 91 02              STA   ($02),Y
3F1D: C8                 INY
3F1E: D0 F9              BNE   :2
3F20: E6 03              INC   $03
3F22: E8                 INX
3F23: E0 FF     SrcEnd   CPX   #/$FF00         ;END OF SOURCE  hi-byte
3F25: D0 EE              BNE   CpyLp2          ;do xx pages
3F27: 60                 RTS

3F28: D8        Setup    CLD                   ;Do what we need to init computer
3F29: 20 2F FB           JSR   INIT            ;Set up screen softswitches
3F2C: 60                 RTS

3F2D: 00 00 00           DS    $3FFC-*         ;F8 ROM moved into $3800-$3FFB

3FFC: 00 FF              DA    Launch+$C000    ;RESET vector ($FF00)
3FFE: 40 FA              DA    $FA40           ;IRQ vector (not used)

```