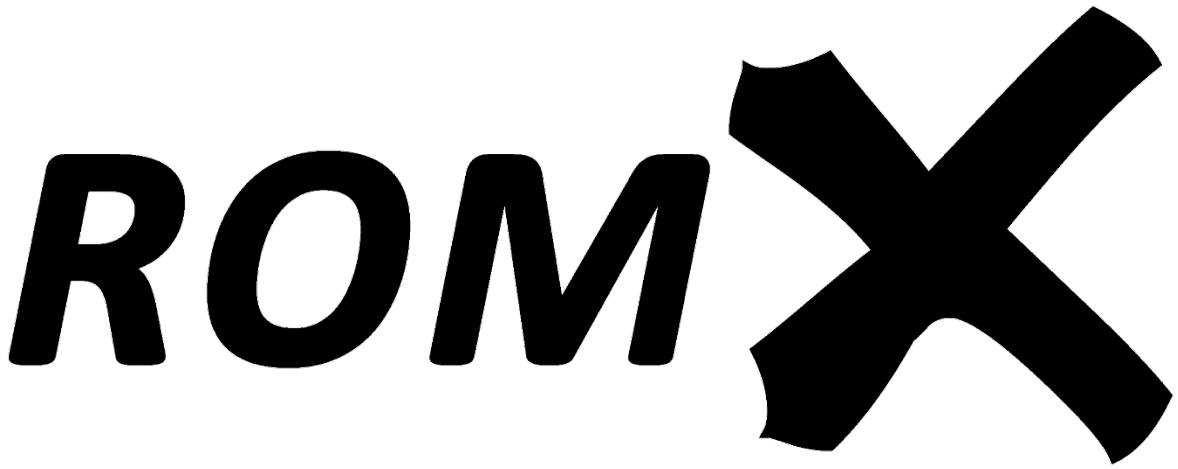

Apple II, II+, //c, //e, //c+

API Reference

Revision 8 – December 2024

Jeff Mazur, Dean Claxton

Contents

ROMX API Overview

This guide will allow the programmer to take advantage of many ROMX features to enhance or produce their own custom solutions. Unless otherwise noted, all of the addresses listed below can be accessed with either READ or WRITE operations, with the BIT instruction being the recommended method. The term ROMX in this document will refer to the ROMX+, ROMXe, ROMXc, and ROMXc+ versions only, not the original ROMX for the Apple][and Apple][+ computers. IMPORTANT: There is a separate API Reference document that pertains to the original ROMX product for the Apple II/II+.

Activating the ROMX Firmware Bank

Assuming motherboard ROM is enabled, at any time and from any bank the following address sequence will instantly activate the Firmware Bank 0 image of the ROMX:

$FACA $FACA $FAFE (immediately switches to Bank 0)

Upon entering the firmware in this manner, several other softswitches will be set: TempBanks ($F850), TempRomBank0 ($F830), and TempLower ($F824). See details below. NOTE: If interrupts are enabled, you should disable them before entering the firmware and then re-enable afterwards.

Manually, from the monitor you can type FACA FACA FAFE and press Return to activate ROM bank 0. Current versions of the ROMX firmware duplicate most of the Autostart Monitor at $F800-$FFFF so you can easily use the standard commands and seamlessly switch between your System ROM and the ROMX Firmware bank. Note however that not all of the F8 ROM is guaranteed to match in future versions.

The following ROMX functions are only available after activating Bank 0:

- Activate ROMX Menu
- Switch Bank Selects
- Switch System ROM bank
- Switch Text ROM bank
- Read the Default System ROM bank
- Read the Default Text ROM bank
- Set and Read the Clock/NVRAM

Displaying the ROMX Menu

While in Bank 0, you can bring up the ROMX menu with or without the launch countdown.

```
$DFD0 (JMP or monitor G command; starts countdown)
```

To bring up the menu without the countdown, execute:

```
JSR $DFD9     (moves ROMX code to RAM)
JSR $1012     (init ROMX code in RAM)
JMP $103C     (launch ROMX Menu in RAM)
```

Switching Bank Selects

ROMX has two System ROM bank pointers; one holds a Temporary bank while the firmware is active and the other stores the Main bank, which is selected when the firmware exits. While in Bank 0, accessing these addresses will switch to the selected Bank:

```
$F850   (selects Temporary Bank)   default when ROMX activated
$F851   (selects Main Bank)        default when ROMX exits
```

If you switch out of bank 0 you will need to use FACA FACA FAFE to return to bank 0 again.

Manually, from the monitor after activating ROM Bank 0, type F851 and press Return to exit to the active Main bank.

Switching Temporary ROM bank

While in Bank 0, accessing any address in the range **$F83x** will switch to the selected Bank x (where x is 1-9,A-F).

```
$F835   switches to Bank 5
```

If you switch out of bank 0 you will need to use FACA FACA FAFE to return to bank 0 to switch again.

Presetting System ROM bank

While in Bank 0, accessing any address in the range **$F80x** will preset the Current System ROM bank to the selected Bank. That bank will be activated when exiting the firmware Bank 0 via the $F851 switch.

```
$F805   presets the Main System ROM to Bank 5
```

Manually, from the monitor, after activating ROM Bank 0, type F805 and press enter to preset the Main System ROM to bank 5.

Presetting the Text ROM bank

If you have an optional ROMX Text ROM board installed, while in Bank 0 accessing any address in the range **$F81x** will preset the Main Text ROM Bank to x (where x is 0-9,A-F). The font will change when exiting Bank 0. If no Text board is installed, this will have no effect.

`$F815  presets the Main Text ROM to Bank 5`

Manually, from the monitor, after activating ROM Bank 0, type F815 and press Return to preset the Text ROM to bank 5.

Reading the Default System ROM bank

While in Bank 0, reading this location will return the current default ROM bank. The value will be in the range of $01-$0F.

`$D034` `(reads current System default Bank)`

Reading the Default Text ROM bank

While in Bank 0, reading this location will return the current default Text ROM bank. The value will be in the range of $10 -$1F.

`$D02E` `(reads current Text default Bank)`

The following code (from the TEXT.DEMO) program will illustrate the use of these commands:

```
0300 - 48        PHA
0301 - 2C CA FA  BIT $FACA  ;Select ROMX Bank 0
0304 - 2C CA FA  BIT $FACA
0307 - 2C FE FA  BIT $FAFE
030A - AD 34 D0  LDA $D02E  ;Retrieve Default Text ROM bank
030D - 8D 1B 03  STA $0318  ;So we can read it later
0310 - 2C 10 F8  LDA $F81x  ;Select Text Bank ($0311 modified externally)
0313 - 68        PLA
0314 - 2C 51 F8  BIT $F851  ;Exit firmware and restore original ROM bank
0317 - 60        RTS
```

Reading the current BootDelay value

While in Bank 0, reading this location will return the current BootDelay value. The value will be in the range of $00 -$0F. The actual delay in seconds is approximately this value divided by 3.

NOTE: Firmware version 1.1.4 and above treat a BootDelay of $0F as Forever or Infinite Delay.

```
$DECA   (reads Delay Value)
```

Accessing the Real-Time Clock (RTC)

ROMX uses an MCP7940 Battery-Backed Clock/Calendar chip to provide time and date information. This chip also contains 64 bytes of battery-backed SRAM (Non-Volatile Memory). The first 16 bytes are reserved for ROMX, but the remaining area is available for user applications. There will be an official procedure for allocating and tracking this space to avoid conflicts. See theRomExchange.com for the latest details.

The MCP7940 has various registers that hold clock, control, and other information. The full data sheet can be found at https://www.microchip.com/wwwproducts/en/MCP7940N .

There are two routines in the ROMX firmware to facilitate setting and reading the RTC. Both use a 7-byte buffer at location $2B0 to temporarily store data for writing to or reading from the clock. After disabling interrupts (if necessary), enabling INTCXROM, and activating ROMX Bank 0:

```
RTC_BUF     EQU $2B0        ;use keyboard buffer for temp storage

Set_Clock   JSR $C803       ;set the RTC using data in RTC_BUF
Read_Clock  JSR $C806       ;read the RTC storing data in RTC_BUF
```

Also see Clock Driver routine on page 9. Note that both routines transfer raw data; it is up to the calling routine to set and handle various bits that are used for control or formatting. A full description of these bits can be found in the MCP7940 datasheet. But the clock data can roughly be obtained using the following table of registers:

```
REG_RTCSEC   = 0x00;  // Register Address: Time Second
REG_RTCMIN   = 0x01;  // Register Address: Time Minute
REG_RTCHOUR  = 0x02;  // Register Address: Time Hour
REG_RTCWKDAY = 0x03;  // Register Address: Date Day of Week
REG_RTCDATE  = 0x04;  // Register Address: Date Day
REG_RTCMTH   = 0x05;  // Register Address: Date Month
REG_RTCYEAR  = 0x06;  // Register Address: Date Year
```

To read the clock from the Monitor:

```
FACA FACA FAFE      ;activate bank 0
C007                ;turn on INTCXROM
C806G               ;read clock
F851                ;exit bank 0
2B0.2B6             ;show results (typical values below)

2B0- D5  Seconds = 55 + $80 ST Oscillator enabled bit
2B1- 50  Minutes = 50
2B2- 07  Hour    = 07 + $00 24-hour format
2B3- 2E  WeekDay = 06 + $28 VBATEN battery enable, OSCRUN
2B4- 17  Date    = 17
2B5- 04  Month   = 04 + $00 Not Leap Year
2B6- 21  Year    = 21
```

While we strongly suggest using the built-in firmware routines, it is also possible to access the MCP7940 directly over its I2C bus using the following addresses.

```
SCL_LO   EQU $F860       ;set SCL low
SCL_HI   EQU $F861       ;set SCL high
SDA_LO   EQU $F862       ;set SDA low
SDA_HI   EQU $F863       ;set SDA high
RTC_Read EQU $F86C       ;disable SDA so we can read RTC
RTC_Send EQU $F86D       ;enable SDA so we can write RTC
RTC_Out  EQU $F86F       ;output D7 to data bus
```

Linking to a second Bank on launch

When launching an image whose Description has an associated &D command (Dual Image, or Double Bank), a second (and conceivably more) image bank(s) can be run before actually launching the selected Bank. If the Description contains the characters &D in positions 30-31 of the string, then the next character will be interpreted as a link to another bank that will be loaded when the original bank is selected.

Normally, when a Bank is selected using a bank key (1-9, A-F) or when the Menu exits after the X)it command or times out, the default Text ROM is enabled followed by a check for either an associated Text Bank and/or Dual bank. If neither is present, then the ROMX activates the selected image and Jumps to the RESET vector pointed to by $FFFC/D in that image.

If there is an associated Text Bank, it will first select that Bank overriding the default Text bank. Then it will continue launching the System Image using the RESET vector.

If there is a Dual bank, then code will be executed to select that bank and Jump to its entry code pointed to by the $FFFE/F vector in its image. The current bank (i.e. the one selected in the Menu) will be passed in location $02A6 (in the range $01-$0F) so that after the secondary code has executed it can resume selecting and launching the desired image.

In a similar fashion, any bank that appends the &Sn command will be treated as an application that first loads a System ROM in bank n before running its own code. The desired System ROM is passed in location $02A6 and the application's own bank can be found at $0287 when it is launched.

Special Note for ZIP CHIP users

If your computer is equipped with a ZIP CHIP accelerator, there are a couple of programming changes you need to consider when accessing the ROMX. Because the current ROMX design cannot run at accelerated speeds or with a cache enabled, you must programmatically disable the ZIP CHIP before accessing the ROMX through its API. This can be done either with a full disable (see ZIP CHIP manual) or for simple tasks by temporarily hitting an I/O address for a slot configured in the ZIP to run as NORMAL. For example, the default speed for Slot 6 is NORMAL so you could access the ROMX in this way :

```
BIT $C0E0  ;Temporarily disable ZIP CHIP
BIT $FACA  ;Select ROMX Bank 0
BIT $FACA
BIT $FAFE
```

This method will work if your call to ROMX takes less than the 50 mS provided by the slot delay. Thus accessing the clock or changing a text font can be done this way. You can also do the same thing from Applesoft with:

```
X = PEEK(49376)
```

In fact, the CLOCK.SET and TEXT.DEMO programs on the ROMX Utilities disk do exactly that. There should be little risk in adding these extra steps to your programs since machines without a ZIP CHIP will effectively ignore them. NOTE: Most likely you will not be able to execute commands this way directly from the Monitor. They must be run from a program.

Other Hardware Addresses

Since the ROMX banks can hold 32K but only map to the roughly 16K ROM space at $C1xx-$FFFF, each bank is sub-divided into a lower and upper bank. This is similar to how the Apple //c System ROMs (except ROM 255) are mapped. When in the Main operating mode, selection of the sub-banks is handled in the usual fashion by the RA14 signal from the IOU chip. In the ROMX+ and ROMXe – and while in the Bank 0 firmware – sub-bank selection is handled by the following softswitches:

```
$F820   (Selects Main ROM Lower Bank)
$F821   (Selects Main ROM Upper Bank)
$F824   (Selects Temporary ROM Lower Bank)
$F825   (Selects Temporary ROM Upper Bank)
```

ROMX+ Only

The ROMX+ also implements a IIe style CXROM switch to allow access to the $C100-$CFFF range within the ROM. This uses the standard softswitch addresses (which work in all Banks, not just Bank 0 firmware):

```
$C006   (Selects ROM on cards)
$C007   (Selects ROM on motherboard)
```

There is also a “master switch” to enable this CXROM switching (these only work from Bank 0 firmware):

```
$F82C   (Disables the CXROM switching)
$F82D   (Enables the CXROM switching)
```

The firmware leaves the CXROM switching enabled upon exit. This allows other banks and your own programs access to the $C100-$CFFF area of the ROM. While Apple II(+) programs will not normally use this space (and indeed, the standard Apple II ROM images will be blank there), it should be safe to leave this switching enabled. You are free to add your own code in this space to enhance the existing 12K ROM images. If you find a conflict however with some software, it is always possible to disable the switching with the $F82C switch.

You can also determine the current status of the CXROM switch:

```
$C014   (Returns with MSB=1 if CXROM switching Enabled; MSB=0 if Disabled)
$C015   (Returns with MSB=1 if internal CXROM selected; MSB=0 if slot ROM)
```

Reserved addresses

The following addresses are Reserved for ROMX firmware use and SHOULD NEVER BE ACCESSED while Bank 0 is active:

```
$C7xx     (Reserved)
$F82x     (Reserved)
$F87x     (Reserved)
```

ROMX detection routine

The following code can be used to detect if a ROMX is present (assumes ROM is currently active). As always, if interrupts are enabled, you should disable before running this and then re-enable afterwards.

```
  BIT $C0E0  ;Temporarily disable ZIP CHIP
  BIT $FACA  ;Select ROMX Bank 0
  BIT $FACA
  BIT $FAFE
  LDA $DFFE  ;Will return $4A if ROMX present
  CMP #$4A   ;  or $AA if ROMX in Recovery mode
  BNE Exit
  LDA $DFFF  ;Will return $CD if ROMX present
  CMP #$CD   ;  or $55 if ROMX in Recovery mode
  BNE Exit
. . . . .  ;ROMX present (do what you want, then fall through to...)

Exit
  BIT $F851  ;Return to Main Bank (MUST DO, EVEN IF ROMX NOT FOUND!)
  RTS
```

While Bank 0 is active, you can get the ROMX Model from locations $D905-$D906 and the firmware version from locations $D923-$D928.

Clock Driver routine

There is a firmware routine at $D8F0 used by the ProDOS clock driver that can help when accessing the clock. It handles the INTCXROM softswitch used in the Apple II/II+ (with ROMX+) and Apple //e.

```
*-------------------------------
*** PRODOS CLOCK DRIVER ROUTINE *** ;MUST BE AT $D8F0
*-------------------------------
DrvrEnt
      LDA INTCXROM      ;Get INTCXROM status
      PHA               ;and save it
      STA INTCXROMON    ;turn our Cxxx on
      JSR ReadClock
Exit  PLA               ;Recover INTCXROM status
      BMI PDR_RTS       ;leave our Cxxx active
      STA INTCXROMOFF   ;turn our Cxxx back off
PDR_RTS
      RTS
```

After enabling the ROMX firmware bank and calling this routine, the clock data will be stored in the 7-byte buffer at location $2B0. See “Accessing the Real-Time Clock (RTC)” for a breakdown of this data.