DOS/65 V3 ROM SUPPLEMENT

VERSION 3.0 ROM

NOTE

V3 ROM software uses some 65C02-only opcodes and as written is not compatible with NMOS processors. You are free to make changes to overcome this restriction as long as the resulting source code is clearly documented.

NOTE

V3 ROM software and some recent utility software, e. g., SD.ASM, are compatible with the WDC tool set rather than the DOS/65 native assembler or TASM. They can be changed to use other tools however that is a major undertaking with significant associated risk. The WDC tools, especially the assembler and linker, are used with .BAT files that are included with the software distribution.

2693 Haven Hill Drive
Grand Junction, CO 81506

richardaleary@gmail.com

Table of Contents

SECTION 1 - INTRODUCTION

This brief supplement to the DOS/65 V2.1 and V3.0 documentation describes the changes associated with the DOS/65 V3.0 ROM version. While the ROM version of DOS 65 can be adapted to almost any ROM environment the first environment used was the Western Design Center (WDC) W65C02SXB. This small single board computer uses the PLCC versions of the WDC devices. The devices are rated for 14 MHz clock speed but the actual clock speed used on the board is 8 MHz. Like many single board computers that have both ROM and RAM, this specific one has 32 kB of RAM and at any one time 32 kB of ROM. The ROM device used is a flash technology device having a total capacity of 128 kB.

Because the WDC product was the basis for the initial development much of this document will refer to specifics associated with that product.

Key characteristics of the ROM version of DOS/65 are:

- The standard interfaces in Page 1 RAM are the same as in prior RAM based versions of DOS/65. The jump to SIM for a WARM BOOT at $100 and the jump to PEM at $103 function exactly the same as they did in prior versions and the target page can be used to synthesize other addresses especially in the SIM jump table.
- The external interfaces to CCM, PEM, and SIM are implemented in a small segment of RAM at the top of the RAM base. Each interface is aligned on a page boundary just as is the case in the RAM based version. This enables application software that captures the PEM location or the SIM jump table location from RAM Page 1 to operate just as they did before.
- Since the standard CCM and PEM lengths are no longer valid, calculations that use the RAM based system parameters will be overly conservative. For example if the RAM resident portion CCM is characterized by the RAM parameter as 2560 bytes when in fact it is actually 256 bytes a significant portion of memory that could be used by an application would not be available. The solution is simple. Edit the source code for any application that makes an assumption about CCM length to change to the 256 bytes applicable to the ROM-based version. Without doing that the application should work fine with a slightly restricted memory.
- The variables needed by each module are contained within the RAM based structure established by the software at startup.
- Cold boot becomes nothing more than copying the RAM structure from a save location in ROM and then jumping to an appropriate starting point in SIM.
- Because the W65C02SXB depends on the PC-based WDC debugger and has no autonomous functionality, I decided to base the system on Daryl Rictor's SBC2 monitor with whatever adaptations are needed because of IO device addressing and functionality differences. Initial development used the WDC as-delivered monitor but as soon as the SBC2 release was baselined the W65C02SXB release was revised to include the SBC2 monitor as the post-reset environment.

- One key utility is provided to initialize and test the microSD card interface. This utility is loaded and executed from the Rictor monitor using the “U” command followed by the “200G” command.
- Because the system is initialized with blank “disks” a compatible version of XMODEM is included. This is also loaded using the “U” command just before booting DOS/65 and then saved using the DOS/65 built-in save command and then executed to load other software onto the system.
- The baseline W65c02SXB-based system uses an SD card interface. Basic system consists of four 8 MB drives allocated out of the microSD card on the expansion board. This interface is based on Andre Fachat's v1.1 design using a VIA and three eternal ICs. That board is a 4-layer PCB that can be purchased by the user.

As has been my policy for several years, complete source code is provided for all of the software.

SECTION 2 – DESIGN CONCEPT

2.1 OVERVIEW

The first design requirement for the ROM version of DOS/65 is that it must look like a standard RAM based system to application software. For example, this means that the user can replace CCM or overwrite CCM and count upon a warm boot to restore the normal CCM. The fact that the RAM based portion of CCM is only one page long is not important to the user or application programmer. The fact that the jump to PEM at $103 is a constant from the very early days of the DOS/65 and contains the information needed to determine where PEM is located in RAM is important.

<u>NOTE</u>

The term MONITOR refers to the general-purpose monitor derived from the V5.1 SBC2 monitor. MON refers to a section of the DOS/65 SIM that is unique to DOS/65.

2.2 STRUCTURE

The ROM based system is structured as shown in the following conceptual sketch (The addresses shown are those relevant for the W65C02SXB version and may not always be valid for other versions. Note that any address less than $8000 is in RAM or is an IO device and conversely any address from $8000 and up is in ROM.):

```
$7000 to $70FF RAM for CCM
        jmp ccm
        CCM variable and buffer space
$7100 to $71FF RAM for PEM
        jmp pem (NB: $7100 is the destination of the jump at $103)
        PEM variable and buffer space
$7200 to $7CFF RAM for SIM
        SIM jmp table (NB: $7203 is the destination of the jump at $100)
        console definition block
        SIM variable and buffer space
$7D00 to $7DFF RAM for MON
        MON variable space
$7E00 to $7EFF RAM for MONITOR
        MONITOR variable space
$7F00 to $7FFF I/O devices
        e. g., VIA for SPI/SD interface @ $7FC0
$8000 to $9FFF ROM for MONITOR
        code and fixed parameters
$A000 to $A8FF ROM for CCM
        code and fixed parameters
$A900 to $B4FF ROM for PEM
        code and fixed parameters
$B500 to $B8FF ROM for SIM
        code and fixed parameters
$B900 to $C0FF ROM for MON
```

code and fixed parameters

$D000 to $DDFF ROM for RAM image

code and data that gets moved to $7000 up

$FFFA

hard IRQ, reset, and NMI vectors

This structure satisfies the primary requirement while at the same time minimizing the complexity associated with the cold boot and warm boot processes. The first step in the cold boot process is to transfer the entire RAM image resident in the ROM to RAM. The warm boot process is equally simple and only the CCM and PEM portions of the RAM image, a total of 512 bytes need be moved – a very small amount of movement.

SECTION 3 – NEW PLATFORM

3.1 OVERVIEW

The first steps in implementing the ROM version of DOS/65 on a new platform are:

First, determine functions to be provided by the OEM monitor or other software that can support the overall DOS/65 processing needs. For example, in the case of the W65C02SXB the only functions that would make sense to use for DOS/65 are the character I/O functions, i.e., those that support console I/O. The W65C0SXB has no storage I/O capability built into it so there was nothing to exploit in designing the system.

Second determine what resources are essential to performing the functions previously assessed as required. Then either allocate existing resources or acquire additional resources. That is exactly what was done for the SPI/SD interface. It did not exist but there was a VIA and a VIA was what was used by Andre Fachat in his elegant SPI/SD interface.

Third, test the resources assigned to ensure that the intended functions are provided correctly. For this step a standalone software package is recommended.

Finally, build the operational software around the resources and functions tested and conduct further testing as the software is built.

This is precisely the process that was followed in the development of the SPI/SD interface and the ROM version of DOS/65. One key advantage of such a process is that problems are found when the complexity of the software is limited thus allowing easier debugging. It does not mean that problems will not be found at the top level but discovery should be at a sufficiently low level to allow clear resolution.

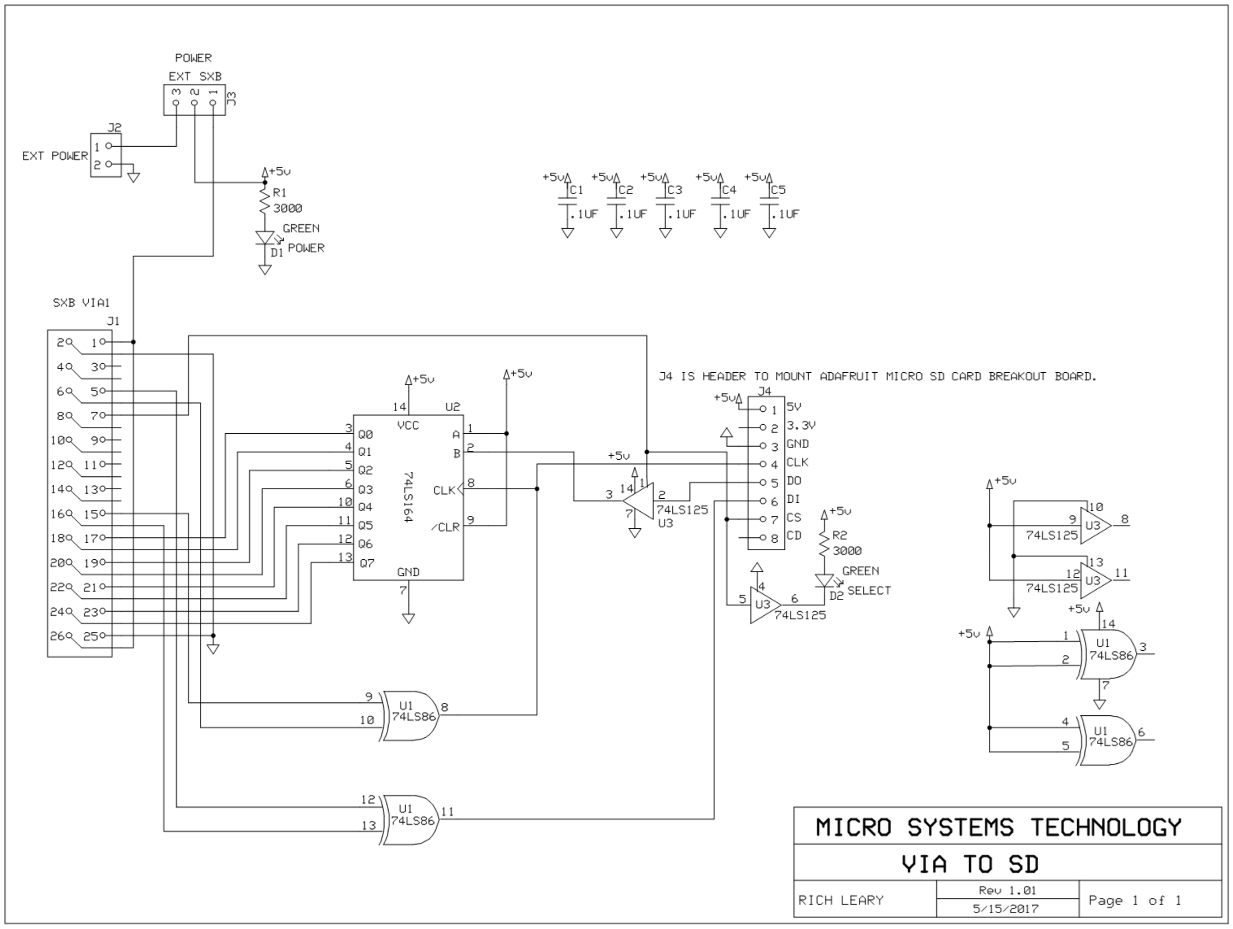
3.2 IMPLEMENTATION

Implementation is straightforward. The basic DOS/65 software structure need not change. However, the specific addresses of functions may have to change as a function of the platform monitor or other software elements. The process of implementing D0S/65 for the SBC2 drew heavily upon the experience involved in producing the version for the WDC SXB but was actually simpler because of the capabilities provided by the SBC2 monitor.

SECTION 4 – SD INTERFACE

Figure 4-1 is the schematic of the VIA to SPI interface for the W65C02SXB.

The SBC2 interface is similar and ExpressPCB schematic and PCB data are provided



for both.

SECTION 5 – ALTERNATIVES

Alternative designs are available and will be explored as time is available. These include:

- Use of Daryl Rictor's 65SPI device that provides the 65xx to SPI interface using a CPLD to provide most of the interface related processing rather than CPU software as is the case in the baseline. This approach promises higher overall processing speed with less hardware than the baseline VIA centric approach. I have all the resources needed to pursue such an approach and may do that in the coming months.
- Use of an IDE/CF interface rather than the SPI/SD interface. The general IDE interface also provides for interfacing multiple different devices to the system. The storage device could be an IDE hard disk drive or a CF card that while significantly different than an SD card is similar in concept of having a solid-state storage device that is removable. Daryl Rictor has done some interesting work in this area and I may capitalize on that in conjunction with use of his 65SPI device. I would also use some of my own experience in providing an IDE/CF interface on my S100 system including its 6502 CPU.
- Use of a different file structure than the relatively simple file structure that is common between CP/M and DOS/65. I have no current plans to pursue such an approach as I think it is counter to the engineering design philosophy evident in the existing file structure. If an IDE/CF interface is used it might make sense to have a FAT file structure underlying what DOS/65 sees. A single FAT file then would become an 8 MB or whatever other size desired partition for DOS/65. Such an approach has been used by others.

APPENDIX A – INSTALLING DOS/65 IN SBC2

Tools required: The first general need is for the soldering tools required to assemble the interface board. Other than those soldering tools the primary need is to have some means of modifying the contents of the EPROM or EEPROM on the board. Daryl Rictor provides no software or firmware to accomplish that task. I decided to use an external programmer. My device programmer is a Xeltek 610P that can handle the 27C256 or 29C256 as it comes. No adapter is needed. Second tool needed is a chip extractor tool to reduce the risk of damage to both the chip and socket. I actually decided to use an Aries low-profile ZIF socket on my SBC2. It is piggybacked on the non-ZIF socket originally installed. If I were just building a new SBC2 I would of course solder the low-profile ZIF socket directly to the board. Finally, the source code and the build scripts are based upon use of the WDC software tool set. Ensure that the WDC tools are installed in the C:\WDC directory as specified by WDC. This installation process as described below also makes the assumption that it is being run using a modern Windows machine. The last software tool is Notepad++, a powerful text editor.

Material required: A few of the EPROM or EEPROM chips are advisable. Adhesive backed colored dots are a useful way of labeling ROM chips. Write down the scheme, e. g., red dot means do not erase or do not change this chip as it is the original ROM chip with the original contents as supplied by Daryl Rictor.

The steps required are as follows:

First, assemble the interface board and construct the cable that will connect the interface board to the SBC2. While not absolutely necessary I recommend using flux remover to clean the board after soldering as that helps reveal any possible soldering errors or splashes or other faults that could hinder progress.

Second, with power off and none of the three integrated circuits on the interface board installed, connect the interface board to the SBC2. Power up the SBC2 and verify that it operates as before and the proper voltages appear at the appropriate pins on the sockets on the interface board.

Third, power off the SBC2 and then insert the three integrated circuits in the interface board and insert a micro SD card into the adapter. Apply power to the computer and the connected interface board and verify that it still operates as before and the proper voltages appear on all the appropriate pins on the sockets on the interface board.

Fourth, remove the EPROM/EEPROM chip. Save that chip and program a new chip using the SBC2ROM.BIN file provided. Insert that chip in the SBC2 and power the system up. Before taking that step make sure you connect a communication software package configured for the correct port. The communications software package should be configured for 9600 bits per second, 8N1, and hardware handshaking.

Fifth, unless you have arranged for me to provide a formatted micro SD card, you'll have to use the monitor U command to load the test SD software (TEST_SBC_SD.HEX with possibility of version numbers on the end) on the computer. It will load at $200 and you should tell the monitor to go at $200.

Sixth, you should see data output from the SD card after a brief turn on of the SELECT LED on the interface board. If the SELECT LED stays on, your micro SD card is probably bad in which case you need to power down and change the card. If you see nonsensical character strings on the screen those are error reports – error reporting is very crude! If all goes well you will then be in the menu portion of the test software. It will start out at block number 00000000. Fill the buffer with the default fill character $E5. Press 7 to clear the DOS/65 directory area on drive A. Repeat that three times after setting the block number to the following values:

```
00800000
01000000
01800000
```

Now reset the SBC2 and at the monitor prompt enter the “@” command to boot DOS/65. You should see the same OCR and CID data that the test software showed followed by a banner message from DOS/65 and then the prompt from CCM of “A0>” meaning you are connected to User Area 0 of drive A. There should be no error messages. A “DIR” command should result in a “NOT FOUND” response.

Reset the SBC2 and at the monitor prompt enter the “U” command and send the XMODEM.HEX file (file name will be XSBCnnn.HEX where nnn will be replaced with a version number, e. g., 124. Once that is complete enter the “@” command to boot DOS/65. As soon as you have the cursor after the “A0>” prompt type:

```
save 22 xmodem.com
```

followed by ENTER. You should see a brief flash of the activity LED and the “A0>” prompt should return. A “DIR” command should show the XMODEM.COM file on drive A.

Now for the tedious part...

Execute a command similar to this to load a file on the drive:

```
xmodem rq <ufn>
```

where <ufn> is an unambiguous file name, e. g., ALLOC.COM. The files in the distribution ZIP file usually have a version number in the file name, e. g., ALLOC204.COM. I recommend retaining the version numbers in the .ASM, .PRN, .KIM, .HEX, and .LST files. For the .COM files use the “normal” name, i. e., the name used in the manual.

The process used for the SXB is the same as that for the SBC2 except for obvious file name changes and the changes driven by the ROM hardware changes.

Tools required: The first general need is for the soldering tools required to assemble the interface board. Other than those soldering tools the primary need is to have some means of modifying the contents of the flash ROM on the board. WDC provides no software or firmware to accomplish that task. The flash bank (0 to 3) can be selected using two VIA output lines and at least one person has written software that enables modification of the flash contents. I decided to use an external flash programmer. My device programmer is a Xeltek 610P that with the proper PLCC adapter (SA015A1T), available from Digi-Key and others, can handle the SST 39SF010A-70-4C-NHE flash memory chip. Second tool needed is a PLCC extractor tool to reduce the risk of damage to both the chip and socket. I use and recommend the Jonard EX-5 also available from Digi-Key. Finally, the source code and the build scripts are based upon use of the WDC software tool set. Ensure that the WDC tools are installed in the C:\WDC directory as specified by WDC. This installation process makes the assumption that it is being run using a modern Windows machine. The last software tool is Notepad++, a powerful text editor.

Material required: A few of the flash chips are advisable. At a little over a $1 each from Digi-Key they are a bargain. Adhesive backed colored dots are a useful way of labeling flash chips. Write down the scheme, e. g., red dot means do not erase or do not change this chip as it is the original ROM chip supplied by WDC.

The steps required are as follows:

First, assemble the interface board and construct the cable that will connect the interface board to the SXB. While not absolutely necessary I recommend using flux remover to clean the board after soldering as that helps reveal any possible soldering errors or splashes or other faults that could hinder progress.

Second, with power off and none of the three integrated circuits on the interface board installed, connect the interface board to the SXB. Power up the SXB and verify that it operates as before and the proper voltages appear at the appropriate pins on the sockets on the interface board.

Third, power off the SXB and then insert the three integrated circuits in the interface board and insert a micro SD card into the adapter. Apply power to the computer and the connected interface board and verify that it still operates as before and the proper voltages appear on all the appropriate pins on the sockets on the interface board.

Fourth, remove the flash memory chip. Save that chip and program a new chip using the SXBROM.BIN file provided. Insert that chip in the SXB and power the system up. Before taking that step make sure you connect a communication software package configured for the correct port. The communications software package should be configured for 19200 bits per second, 8N1, and hardware handshaking.

Fifth, unless you have arranged for me to provide a formatted micro SD card, you'll have to use the monitor U command to load the test SD software (TEST_SXB_SD.HEX with possibility of version numbers on the end) on the computer. It will load at $200 and you should tell the monitor to go at $200.

Sixth, you should see data output from the SD card after a brief turn on of the SELECT LED on the interface board. If the SELECT LED stays on, your micro SD card is probably bad in which case you need to power down and change the card. If you see nonsensical character strings on the screen those are error reports – error reporting is very crude! If all goes well you will then be in the menu portion of the test software. It will start out at block number 00000000. Fill the buffer with the default fill character $E5. Press 7 to clear the DOS/65 directory area on drive A. Repeat that three times after setting the block number to the following values:

```
00800000
01000000
01800000
```

Now reset the SXB and at the monitor prompt enter the “@” command to boot DOS/65. You should see the same OCR and CID data that the test software showed followed by a banner message from DOS/65 and then the prompt from CCM of “A0>” meaning you are connected to User Area 0 of drive A. There should be no error messages. A “DIR” command should result in a “NOT FOUND” response.

Reset the SXB and at the monitor prompt enter the “U” command and send the XMODEM.HEX file (file name will be XSXBnnn.HEX where nnn will be replaced with a version number, e. g., 123. Once that is complete enter the “@” command to boot DOS/65. As soon as you have the cursor after the “A0>” prompt type:

```
save 22 xmodem.com
```

followed by ENTER. You should see a brief flash of the activity LED and the “A0>” prompt should return. A “DIR” command should show the XMODEM.COM file on drive A.

Now for the tedious part...

Execute a command similar to this to load a file on the drive:

```
xmodem rq <ufn>
```

where <ufn> is an unambiguous file name, e. g., ALLOC.COM. The files in the distribution ZIP file usually have a version number in the file name, e. g., ALLOC204.COM. I recommend retaining the version numbers in the .ASM, .PRN, .KIM, .HEX, and .LST files. For the .COM files use the “normal” name, i. e., the name used in the manual.