RomWBW ROM Applications

Version 3.1 Pre-release

Phillip Summers

Thu 10/21/2021

Contents

Summary

RomWBW includes a small selection of built in utilities and programming languages.

ROMWBW Monitor

The Monitor program is a low level utility that can be used for testing and programming. It allows programs to be entered, memory to be examined, and input/output devices to be read or written to.

It's key advantage is that is available at boot up.

Its key disadvantages are that code cannot be entered in assembly language and there is no ability to save to memory devices.

The available memory area for programming is `0200-EDFFh`. The following areas are reserved:

Memory Area	Function
`0000-00FFh`	Jump and restart (RST) vectors
`0100-01FFh`	HBIOS configuration block
`EE00-FDFFh`	MONITOR
`FE00-FFFFh`	HBIOS proxy

Commands can be entered at the command prompt > Automatic case conversion takes place on command entry and all arguments are expected to be in hex format.

The current memory bank in low memory is displayed before the prompt i.e.:

```
8E>
```

The Monitor allows access to all memory locations but ROM and Flash memory cannot be written to. Memory outside the normal address range can be accessed using the B command. The first 256 bytes `0000-01FF` is critical for the HBIOS operation. Changing banks may make this information inaccessible.

Refer to the RomWBW Architecture manual for details memory banking.

A quick guide to using the Monitor program follows:

? - Displays a summary of available commands.

```
Monitor Commands (all values in hex):
B                    - Boot system
D xxxx [yyyy]        - Dump memory from xxxx to yyyy
F xxxx yyyy zz       - Fill memory from xxxx to yyyy with zz
H                    - Halt system
I xxxx               - Input from port xxxx
K                    - Keyboard echo
L                    - Load Intel hex data
M xxxx yyyy zzzz     - Move memory block xxxx-yyyy to zzzz
O xxxx yy            - Output value yy to port xxxx
P xxxx               - Program RAM at address xxxx
R xxxx [[yy] [zzzz]] - Run code at address xxxx
                       Pass yy and zzzz to register A and BC
T xxxx               - X-modem transfer to memory location xxxx
S xx                 - Set bank to xx
X                    - Exit monitor
```

Cold Boot

B - Performs a cold boot of the ROMWBW system. A complete re-initialization of the system is performed and the system returns to the Boot Loader prompt.

Dump Memory

D xxxx [yyyy] - Dump memory from hex location xxxx to yyyy on the screen as lines of 16 hexadecimal bytes with their ASCII equivalents (if within a set range, else a '.' is printed). If the end address is omitted then 256 bytes is displayed.

A good tool to see where code is located, check for version id, obtain details for chip configurations and execution paths.

Examples: `D 100 1FF`

```
0100: 10 0B 01 5A 33 45 4E 56 01 00 00 2A 06 00 F9 11  ...Z3ENV...*..ù.
0110: DE 38 37 ED 52 4D 44 0B 6B 62 13 36 00 ED B0 21  Þ87íRMD.kb.6.í°!
0120: 7D 32 E5 21 80 00 4E 23 06 00 09 36 00 21 81 00  }2å!..N#...6.!..
0130: E5 CD 6C 1F C1 C1 E5 2A C9 8C E5 CD 45 05 E5 CD  åÍl.ÁÁå*É.åÍE.åÍ
0140: 59 1F C3 00 00 C3 AE 01 C3 51 04 C3 4C 02 C3 57  Y.Ã..î.ÃQ.ÃL.ÃW
0150: 02 C3 64 02 C3 75 02 C3 88 02 C3 B2 03 C3 0D 04  .Ãd.Ãu.Ã..ò.Ã..
0160: C3 19 04 C3 22 04 C3 2A 04 C3 35 04 C3 40 04 C3  Ã..Ã".Ã*.Ã5.Ã@.Ã
0170: 48 04 C3 50 04 C3 50 04 C3 50 04 C3 8F 02 C3 93  H.ÃP.ÃP.ÃP.Ã..Ã.
0180: 02 C3 94 02 C3 95 02 C3 85 04 C3 C7 04 C3 D1 01  .Ã..Ã..Ã..ÃÇ.ÃÑ.
0190: C3 48 02 C3 E7 04 C3 56 03 C3 D0 01 C3 D0 01 C3  ÃH.Ãç.ÃV.ÃÐ.ÃÐ.Ã
01A0: D0 01 C3 D0 01 C3 D0 01 C3 D0 01 01 02 01 CD 6B  Ð.ÃÐ.ÃÐ.ÃÐ....Ík
01B0: 04 54 68 69 73 20 66 75 6E 63 74 69 6F 6E 20 6E  .This function n
01C0: 6F 74 20 73 75 70 70 6F 72 74 65 64 2E 0D 0A 00  ot supported....
01D0: C9 3E FF 32 3C 00 3A 5D 00 FE 20 28 14 D6 30 32  É>ÿ2<.:].þ (.Ö02
01E0: AB 01 32 AD 01 3A 5E 00 FE 20 28 05 D6 30 32 AC  «.2-
.:^.þ (.Ö02¬
01F0: 01 C5 01 F0 F8 CF E5 26 00 0E 0A CD 39 02 7D 3C  .Å.ðøÏå&...Í9.}<
```

Fill Memory

F xxxx yyyy zz - Fill memory from hex xxxx to yyyy with a single value of zz over the full range. The Dump command can be used to confirm that the fill completed as expected. A good way to zero out memory areas before writing machine data for debug purposes.

Halt System

H - Halt system. A Z80 HALT instruction is executed. The system remains in the halt state until the system is physically rebooted. Interrupts will not restart the system. On systems that support a HALT status LED, the LED will be illuminated.

Input from port

I xxxx - Input data from port xxxx and display to the screen. This command is used to read values from hardware I/O ports and display the contents in hexadecimal.

Keyboard Echo

K - Echo any key-presses from the terminal. Press ‘ESC’ key to quit. This facility provides that any key stroke sent to the computer will be echoed back to the terminal. File down loads will be echoed as well while this facility is ‘on’.

Load Hex format file into memory

L - Load a Intel Hex format file via the terminal program. The load address is defined in the hex file of the assembled code.

The terminal emulator program should be configured to give a delay at the end of each line to allow the monitor enough time to parse the line and move the data to memory.

Keep in mind that this will be a transient unless the system support battery backed memory. Saving to memory drive is not supported.

Move memory

M xxxx yyyy zzzz - Move hex memory block xxxx to yyyy to memory starting at hex location zzzz. Care should be taken to insure that there is enough memory at the destination so that code does not get over-written or memory wrapped around.

Output to port

O xxxx yy - Output data byte xx to port xxxx. This command is used to send hexadecimal values to hardware I/O ports to verify their operation and is the companion to the I operation. Use clip leaded LEDs to confirm the data written.

Program memory location

P xxxx - Program memory location xxxx. This routine will allow you to program a hexadecimal value ‘into memory starting at location xxxx. Press ’Enter’ on a blank line to return to the Monitor prompt.

The limitation around programming memory is that it must be entered in hexadecimal. An alternative is to use the L command to load a program that has been assembled to a hex file on the remote computer.

An excellent online resource for looking up opcodes for entry can be found here: https://clrhome.org/table

Run program

R xxxx [[yy] [zzzz]] - Run program at location xxxx. If optional arguments yy and zzzz are entered they are loaded into the A and BC register respectively. The return address of the Monitor is saved on the stack so the program can return to the monitor. On return to the monitor, the contents of the A, HL, DE and BC registers are displayed.

Set bank

S xx - Change the bank in memory to xx. Memory addresses 0000-7FFF (i.e. bottom 32k) are affected. Because the interrupt vectors are stored in the bottom page of this range, this function is disable when interrupt mode 1 is

being used (IM1). Interrupt mode 2 is not affected as the associated jump vectors are stored in high memory.

Changing the bank also impacts the restart vectors (RST), so executing code that call the HBIOS using the RST 08 assembly code will not work.

The monitor stack resides in high memory and is not affected but any code that changes the stack to low memory will be affected.

Bank codes and descriptions

TYPE	DESCRIPTION	BANK	DETAILS
RAM	COMMON BANK	9F	1024K RAM SYSTEM
RAM	USER BANK	9E	1024K RAM SYSTEM
RAM	BIOS BANK	9D	1024K RAM SYSTEM
RAM	AUX BANK	9C	1024K RAM SYSTEM
RAM	OS BUFFERS END	9B	1024K RAM SYSTEM
RAM	OS BUFFERS START	98	1024K RAM SYSTEM
RAM	RAM DRIVE END	97	1024K RAM SYSTEM
RAM	COMMON BANK	8F	512K RAM SYSTEM
RAM	USER BANK	8E	512K RAM SYSTEM
RAM	BIOS BANK	8D	512K RAM SYSTEM
RAM	AUX BANK	8C	512K RAM SYSTEM
RAM	OS BUFFERS	8B	512K RAM SYSTEM
RAM	OS BUFFERS	8A	512K RAM SYSTEM
RAM	OS BUFFERS	89	512K RAM SYSTEM
RAM	OS BUFFERS	88	512K RAM SYSTEM
RAM	RAM DRIVE END	87	512K RAM SYSTEM
RAM	RAM DRIVE START	80	
ROM	BOOT BANK	00	COLD START & HBIOS
ROM	LOADER & IMAGES	01	MONITOR, FORTH
ROM	ROM IMAGES CONTD.	02	BASIC, ETC
ROM	FAT FILESYSTEM	03	UNA ONLY, ELSE UNUSED

TYPE	DESCRIPTION	BANK	DETAILS
ROM	ROM DRIVE START	04	
ROM	ROM DRIVE END	0F	512K ROM SYSTEM
ROM	ROM DRIVE END	1F	1024K ROM SYSTEM

X-modem transfer

T xxxx - Receive an X-modem file transfer and load it into memory starting at location xxxx.

128 byte blocks and checksum mode is the only supported protocol.

If the monitor is assembled with the DSKY functionality, this feature will be exclude due to space limitions.

NOTES:

The RTC utility on the CP/M ROM disk provides facilities to manipulate the Real Time Clock non-volatile Memory. Use the C or Z option from the Boot Loader to load CP/M and then run RTC to see the options list.

FORTH

CamelForth is the version of Forth included as part of the boot ROM in ROMWBW. It has been converted from the Z80 CP/M version published here www.camelforth.com/page.php?5. The author is Brad Rodriguez who is a prolific Forth enthusiast, whose work can be found here: www.bradrodriguez/papers/index.html

For those are who are not familiar with Forth, I recommend the wikipedia article en.wikipedia.org/wiki/Forth_(programming_language and the Forth Interest Group website www.forth.org

Important things to know

Forth is case sensitive.

To exit back to the boot loader type ***bye***

To get a list of available words type ***WORDS***

To reset Forth to its initial state type ***COLD***

Most of the code you find on the internet will not run unless modified or additional Forth words are added to the dictionary.

This implementation does not support loading or saving of programs. All programs need to be typed in. Additionally, screen editing and code blocks are not supported.

Structure of Forth source files

File	Description
camel80.azm	Code Primitives
camel80d.azm	CPU Dependencies
camel80h.azm	High Level words
camel80r.azm	ROMWBW additions
glosshi.txt	Glossary of high level words
glosslo.txt	Glossary of low level words
glossr.txt	Glossary of ROMWBW additions

ROMWBW Additions

Extensions and changes to this implementation compared to the original distribution are:

The source code has been converted from Z80mr assembler to Hector Peraza's zsm.

An additional file camel80r.azm has been added for including additional words to the dictionary at build time. However, as currently configured there is very little space allocated for addition words. Exceeding the allocated ROM space will generate an error message when building.

James Bowman's double precision words have been added from his RC2014 version: https://github.com/jamesbowman/camelforth-z80

Word	Syntax	Description
D+	d1 d2 – d1+d2	Add double numbers
2>R	d –	2 to R
2R>	d –	fetch 2 from R
M*/	d1 n2 u3 – d=(d1*n2)/u3	double precision mult. div

BASIC

For those who are not familiar with BASIC, it stands for Beginners All purpose Symbolic Instruction Code.

ROMWBW contains two versions of ROM BASIC, a full implementation and a "tiny" BASIC.

The full implementation is a version of Microsoft BASIC from the NASCOM Computer.

A comprehensive instruction manual is available in the Doc\Contrib directory.

ROMWBW specific features

- Sound
- Graphics
- Terminal Support

ROMWBW unsupported features

- Cassette loading
- Cassette saving

TastyBASIC

TastyBASIC offers a minimal implementation of BASIC that is only 2304 bytes in size. It originates from Li-Chen Wang's Palo Alto Tiny BASIC from around 1976. It's small size suited the tiny memory capacities of the time. This implementation is by Dimitri Theulings and his original source can be found here https://github.com/dimitrit/tastybasic

Features / Limitations

Integer arithmetic, numbers -32767 to 32767 Singles letter variables A-Z 1-dimensional array support Strings are not supported

Direct Commands

- `LIST`,`RUN`, `NEW`, `CLEAR`, `BYE`

Statements

- `LET`, `IF`, `GOTO`, `GOSUB RETURN`, `REM`, `FOR TO NEXT STEP`, `INPUT`, `PRINT`, `POKE`, `END`

Functions

- `PEEK`, `RND`, `ABS`, `USR`, `SIZE`

Operators

- >=, #, >, =, <=, <
- Operator precedence is supported.

Type ***BYE*** to return to the monitor.

Play a Game

2048

2048 is a puzzle game that can be both mindless and challenging. It appears deceptively simple but failure can creep up on you suddenly.

It requires an ANSI/VT-100 compatible colour terminal to play.

2048 is like a sliding puzzle game except the puzzle tiles are numbers instead of pictures. Instead of moving a single tile all tiles are moved simultaneously in the same direction. Where two tiles of the same number collide, they are reduced to one tile with the combined value. After every move a new tile is added with a starting value of 2.

The goal is to create a tile of 2048 before all tile locations are occupied. Reaching the highest points score, which is the sum of all the tiles is a secondary goal. The game will automatically end when there are no more possible moves.

Play consists of entering a direction to move. Directions can be entered using any of three different keyboard direction sets.

```
Direction | Keys
----------|----------
Up        | w ^E 8
Down      | s ^X 2
Left      | a ^S 4
```

```
Right     | d ^D 6
```

The puzzle board is a 4x4 grid. At start, the grid will be populated with two 2 tiles. An example game sequence is shown below with new tiles to the game shown in brackets.

```
Start              Move 1 - Up        Move 2 - Left      Move 3 - Left
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
|   |   |   |(2)| |   |   |   | 4 | | 4 |   |   |   | | 4 |   |   |   |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
|   |   |   |   | |   |   |   |   | |   |   |   |(4)| | 4 |   |   |   |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
|   |   |   |(2)| |   |   |   |   | |   |   |   |   | |   |   |   |   |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
|   |   |   |   | |   |   |(2)|   | | 2 |   |   |   | | 2 |   |(2)|   |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+

Move 4 - Left      Move 5 - Up        Move 6 - Right     Move 7 - Up
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
| 4 |   |   |   | | 8 |   |   | 4 | |   |   | 8 | 4 | |   |   | 8 | 8 |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
| 4 |   |   |(4)| | 4 |   |   |   | |   |   |   | 4 | |   |   |   | 2 |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
|   |   |   |   | |   |   |   |   | |   |   |   |   | |   |   |   |   |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
| 4 |   |   |   | |(2)|   |   |   | |(2)|   |   | 2 | |(2)|   |   |   |
+---+---+---+---+ +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
```

This is how I lost this game:

```
+---+---+---+---+
| 4 | 2 | 16| 4 |
+---+---+---+---+
| 32| 64| 8 | 2 |
```

```
+---+---+---+---+
| 4 | 8 |128| 32|
+---+---+---+---+
|(2)| 16| 8 | 4 |
+---+---+---+---+
```

Press Q at any time to bring up the option to Quit or Restart the game.

Network Boot

Xmodem Flash Updater

The ROMWBW Xmodem flash updater provides the capability to update ROMWBW from the boot loader using an x-modem file transfer. It offers similar capabilities to Will Sowerbutts FLASH4 utility except that the flashing process occurs during the file transfer.

These are the key differences between the two methods are:

Xmodem Flash Updater	FLASH4
Available from the boot loader	Well proven and tested
Xmodem transfer is integrated	Wider range of supported chips and hardware
Integrated checksum utilities	Wider range of supported platforms
Capability to copy a ROM image	Only reprograms sectors that have changed
More convenient one step process	Ability save and verify ROM images
No intermediate storage required	Progress display while flashing
.	Displays chip identification information
.	Faster file transfer

The major disadvantages of the Updater is that it is new and relatively untested. There is the risk that a failed transfer will result in a partially

flashed and unbootable ROM. There are some limitations on serial transfer speeds.

The updater utility was initially intended to support the Retrobrew SBC-V2-005 platform using Atmel 39SF040 flash chips but has now been extended to be more generic in operation.

Supported flash chips are 39SF040, 29F040, AT49F040, AT29C040, M29F040 , MX29F040, A29010B, A29040B

The Atmel 39SF040 chip is recommended as it can erase and write 4Kb sectors. Other chips require the whole chip to be erased.

Usage

In most cases, completing a ROM update is a simple as:

1. Booting to the boot loader prompt
2. Selecting option X - Xmodem Flash Updater
3. Selecting option U - Update
4. Initiating an X-modem transfer of your ROM image on your console device
5. Selecting option R - Reboot

If your console device is not able to transfer a ROM image i.e. your console is a VDU then you will have to use the console options to identify which character-input/output device is to be used as the serial device for transfer.

When your console is the serial device used for the transfer, no progress information is displayed as this would disrupt the x-modem file transfer. If you use an alternate character-input/output devices as the serial device for the transfer then progress information will be displayed on the console device.

Due to different platform processor speeds, serials speeds and flow control capabilities the default console or serial device speed may need to be reduced for a successful transfer and flash to occur. The **Set Console In-**

terface/Baud code option at the Boot Loader can be used to change the speed if required. Additionally, the Updater has options to set to and revert from a recommended speed.

Console Options Option (C) - Set Console Device

Option (S) - Set Serial Device

By default the updater assumes that the current console is a serial device and that the ROM file to be flashed will also be transferred across this device, so the Console and Serial device are both the same.

Either device can be can be change to another character-input/output device but the updater will always expect to receive the x-modem transfer on the **Serial Device**

The advantage of transferring on a different device to the console is that progress information can be displayed during the transfer.

Option (>) - Set Recommended Baud Rate

Option (<) - Revert to Original Baud Rate

Programming options

Option (U) - Begin Update

The will begin the update process. The updater will expect to start receiving an x-modem file on the serial device unit.

X-modem sends the file in packets of 128 bytes. The updater will cache 32 packets which is 1 flash sector and then write that sector to the flash device.

If using separate console, bank and sector progress information will shown

```
BANK 00 s00 s01 s02 s03 s04 s05 s06 s06 s07
BANK 01 s00 s01 s02 s03 s04 s05 s06 s06 s07
BANK 02 s00 s01 s02 s03 s04 s05 s06 s06 s07 etc
```

The x-modem file transfer protocol does not provide any filename or size information for the transfer so the updater does not perform any checks on the file suitability.

The updater expects the file size to be a multiple of 4 kilobytes and will write all data received to the flash device. A system update file (128kb .img) or complete ROM can be received and written (512kb or 1024kb .rom)

If the update fails it is recommended that you retry before rebooting or exiting to the Boot loader as your machine may not be bootable.

Option (D) - Duplicate flash #1 to flash #2

This option will make a copy of flash #1 onto flash #2. The purpose of this is to enable making a backup copy of the current flash. Intended for systems using 2x512Kb Flash devices.

Option (V) - Toggle Write Verify

By default each flash sector will be verified after being written. Slight performance improvements can be gained if turned off and could be used if you are experiencing reliable transfers and flashing.

Exit options

Option (R) - Reboot

Execute a cold reboot. This should be done after a successful update. If you perform a cold reboot after a failed update then it is likely that your system will be unusable and removing and reprogramming the flash will be required.

Option (Q) - Quit to boot loader.

The SBC Boot Loader is reloaded from ROM and executed. After a successful update a Reboot should be performed. However, in the case of a failed update this option could be used to attempt to load CP/M and perform the normal x-modem / flash process to recover.

CRC Utility options

Option (1) and (2) - Calculate and display CRC32 of 1st or 2nd 512k ROM.
Option (3) - Calculate and display CRC32 of a 1024k (2x512Kb) ROM.

Can be used to verify if a ROM image has been transferred and flashed correctly. Refer to the Teraterm section below for details on configuring the automatic display of a files CRC after it has been transferred.

In Windows, right clicking on a file should also give you a context menu option CRC SHA which will allow you to select a CRC32 calculation to be done on the selected file.

Teraterm macro configuration

Macros are a useful tool for automatic common tasks. There are a number of instances where using macros to facilitate the update process could be worthwhile if you are:

- Following the ROMWBW development builds.
- Doing lots of configuration changes.
- Doing development on ROMWBW drivers

Macros can be used to automate sending ROM updates or images and for my own purposed I have set up a separate macro for transferring each of the standard build ROM, my own custom configuration ROM and update ROM.

An example macro file to send an *.upd file, using checksum mode and display the crc32 value of the transmitted file:

```
Xmodem send, checksum, display crc32
xmodemsend '\\desktop\users\phillip\documents\github\romwbw\binary\sbc_std_cust.up
crc32file crc '\\desktop\users\phillip\documents\github\romwbw\binary\sbc_std_cust
sprintf '0x%08x' crc
messagebox inputstr 'crc32'
```

Serial speed guidelines

As identified in the introduction, there are limitations on serial speed depending on processor speed and flow control settings. Listed below are some of the results identified during testing.

Platform / Configuration	Processor Speed	Maximum Serial Speed
Sbc-v2 uart no flow control	2mhz	9600
sbc-v2 uart no flow control	4mhz	19200
sbc-v2 uart no flow control	5mhz	19200
sbc-v2 uart no flow control	8mhz	38400
sbc-v2 uart no flow control	10mhz	38400
sbc-v2 usb-fifo 2mhz+		n/a
sbc-mk4 asci no flow control	18.432mhz	9600
sbc-mk4 asci with flow control	18.432mhz	38400

The **Set Recommend Baud Rate** option in the Updater menu follows the following guidelines.

Processor Speed	Baud Rate
1Mhz	4800
2-3Mhz	9600
4-7Mhz	19200
8-20Mhz	38400

These can be customized in the updater.asm source code in the CLKTBL table if desired. Feedback to the ROMWBW developers on these guidelines would be appreciated.

Notes:

All testing was done with Teraterm x-modem, Forcing checksum mode using macros was found to give the most reliable transfer. Partial writes can be completed with 39SF040 chips. Other chips require entire flash to be erased before before being written. An SBC V2-005 MegaFlash or Z80 MBC required for 1mb flash support. The Updater assumes both chips are same type Failure handling has not been tested. Timing broadly calibrated on a Z80 SBC-v2 Unabios not supported