CORSHAM
Corsham Technologies, LLC
www.corshamtech.com
617 Stokes Road, Suite 4-299
Medford, NJ 08055

xKIM Monitor

Introduction

The xKIM Monitor is an extension to the KIM-1's built-in TTY monitor that adds a few more useful commands and it meant to be easily extendable by users. It is included as an EPROM with the Corsham Technologies 60K RAM/ROM board, but can be used as a RAM based monitor as well.

During development, I had a very early version of xKIM in EPROM, then used the command to load hex files containing more extended versions for debugging. I.e., the monitor was used to debug itself!

Features

- Includes support for the Corsham Technologies SD Card system. Along with the low level driver functions, there are commands to do disk directories and load hex files from the SD card.
- Commands to examine/edit memory, perform a hex dump and run memory tests.
- All commands have help text. Brief, but better than nothing.
- Easy to add new code directly to the monitor.
- Monitor can be extended; load additional commands from the SD card.
- Vectors to many useful subroutines in the monitor, so making changes to the monitor does break existing code.
- Can run out of RAM or ROM.
- No additional RAM used in zero page or the standard memory on the KIM-1. The extended monitor re-uses existing KIM-1 memory.

- No heavy legal mumbo-jumbo. Feel free to use the code.

Starting the xKIM Monitor

Your KIM-1 needs to be running using a TTY (terminal) interface. Set the starting address of where xKIM begins and run:

Press RS on KIM-1
Press ENTER on console

```
KIM
135E 3E E000 space
E000 4C G

Extended KIM Monitor v0.3 by Corsham Technologies, LLC
www.corshamtech.com

>
```

At this point, hit a question mark to get available help:

```
>?
Available commands:

? ........... Show this help
B ........... Bob's Tiny BASIC
C ........... Show clock
D ........... Disk directory
E xxxx ...... Edit memory
H xxxx xxxx . Hex dump memory
J xxxx ...... Jump to address
K ........... Go to KIM monitor
L ........... Load HEX file
M xxxx xxxx . Memory test
O xxxx xxxx . Calculate branch offset
P ........... Ping disk controller
S xxxx xxxx . Save memory to file
T ........... Type disk file
! ........... Do a cold start
```

Some commands take additional arguments. Unless otherwise specified, the arguments are always hex numbers with exactly the number of digits show. So, for example, if you want to hex dump memory from 0000 to 00FF, you would enter four zeros, then two more zeros and two “F”s.

If you type a non-hex value then the command is immediately aborted and command returns to the prompt.

Some commands apply only when you have an SD Card System installed, such as D, P, S and T.

The L command can download a HEX format file from either the SD card or the terminal. When you press L it will ask for the filename to load. If you press RETURN then it will assume the file is being downloaded from your terminal program.

Cold versus Warm Start

Vectors to Internal Functions

To allow user written programs to use some of the handy subroutines present in the xKIM Monitor, a set of vectors sits at the start of the code. User programs, including extensions, should only call subroutines via the vectors and never directly jump to code inside or use data that is not defined as being public.

The addresses listed here assume xKIM is in EPROM starting at location E000, such as found on our KIM 60K RAM/ROM Board. If you are running xKIM from RAM then the addresses will be different.

RAM Locations

Address	Name	Description
DFFA	ColdFlag	Cold flag. The contents of these two locations are used to determine if the monitor is in a warm start or cold start state.
DFFC	ExtensionAddr	Address of any user extension. If a user extension is loaded, the starting address of the command table should be placed into these locations, LSB first.
DFFE	HighestAddress	Contains the highest address in RAM that user programs may occupy. Your programs are free to adjust this down to reserve space at the top of RAM.

Cold/Warm Start

Address	Name	Description
E000	extKim	Entry point. This handles both cold and warm entry. Any user written program or monitor extension should jump to this location when done. This uses the Cold Flag at DFFC to determine if this is a cold or warm start. This should be JMPed to, as it does not return.

Console Input/Output

Address	Name	Description
E003	OUTCH	Output the character in A to the console.
E006	GETCH	Wait for a character from the console and return it in A. Character is echoed.
E009	GETCHNE	Same as E006 for now, but was intended not to echo.
E00C	consolePoll	Currently unused but is meant to poll to see if a character is ready for reading.
E00F	putsil	Print string in-line. The address after the JSR to this function contains an ASCII string that is printed until a 0 byte is found.
E012	getHex	Gets a two digit hexadecimal number into A and carry clear. If a non-hex value is entered, returns the offending character in A and carry set.
E015	prtHex	Prints the contents of A as two hex digits.
E018	getStartAddr	Gets a four digit hex number and saves it in SAL/SAH (17F5/17F6) and carry clear. If a non-hex character is entered, return C set and the offending character in A.
E01B	getEndAddr	Gets a four digit hex number and saves it in EAL/EAH (17F7/17F8) and carry clear. If a non-hex character is entered, return C set and the offending character in A.
E01E	getAddrRange	Does a call to getStartAddr and then getEndAddr to set up a range of addresses.

Reserved for Future Use

It is highly likely that we'll add more useful subroutines to the monitor and make them available, so we're reserving some vectors for those future uses.

Address	Name	Description
E021		Reserved
E024		Reserved
E027		Reserved
E02A		Reserved
E02D		Reserved
E030		Reserved

Low Level SD Card

Address	Name	Description
E033	xParInit	Initializes the interface for the SD Card system.
E036	xParSetWrite	Sets SD card direction for writes.
E039	xParSetRead	Sets SD card direction for reads.
E03C	xParWriteByte	Writes A to the SD card system.
E03F	xParReadByte	Reads one byte from SD card system into A.

Higher Level SD Card

Address	Name	Description
E042	DiskPing	Does a sanity check to verify the SD controller system can be reached. Returns C if SD system is on-line, C set if not.
E045	DiskDir	Begins a directory read of the SD card. Takes no input parameters, but returns with C clear on success, or C set on error. Use DiskDirNext to read disk directory entries.
E048	DiskDirNext	On entry, X (MSB) and Y (LSB) point to a buffer area large enough to get the next directory entry. Returns C set if end of directory is reached (buffer has no valid contents) or C clear and X/Y point to null at end of filename. Currently, the buffer area should be 13 bytes.
E04B	DiskOpenRead	Open a file for reading. On entry, X (MSB) and Y (LSB) point to a null-terminated filename. On return C is clear if the file is open or set if not.
E04E	DiskRead	Reads the contents of an open disk file. On entry, X (MSB) and Y (LSB) point to the buffer area, and A contains the number of bytes to read. On return C is set on error, else C is clear and A contains the number of bytes actually read into the buffer.
E051	DiskClose	Closes an open disk file.

Making Changes Directly

If you've got some new commands and want to make them permanent, then the best way is to add them directly to the main source code. You can follow the existing logic and commands to see how to re-use existing functions to get addresses and perform other low-level functions.

Notice the label commandTable, as this command structure is used both for internal commands and also user written extensions. Each command consists of a five byte entry:

- ASCII character of command (1 byte)
- Pointer to the code that processes this command (2 bytes, LSB first)
- Pointer to brief command description (2 bytes, LSB first)

If you add your own vectors, please put them after the SD card vectors. While we might be adding new vectors there, no space was specifically allocated for them. Or, if you're willing to share the code, we can add it to the distributed versions.

Adding An Extension

Something I've wanted for long time is an easy way to add new commands to a monitor without making those extensions look like they were added later, so this was my chance to make that happen!

An extension is just a short program that hooks right into xKIM and adds additional commands seamlessly from the user's perspective. A very simplistic example is provided but here it is to show how simple it is.

```
;=====================================================
; A sample extension for the Extended KIM monitor.
; This is a very simple example of how to write an
; extension (adding a new command) for the
; Extended KIM monitor.
;
; How can you test this?  Easy.  First, use the "?"
; command in the extended monitor and verify the
; "Z" command is not listed, then load the binary
; version of this file.  Do "?" again and you'll see
; the new command has been added and can be used.
;
```

```
; 12/26/2015 - Bob Applegate, bob@corshamtech.com
;
; Consider buying a KIM-1 board from us:
; www.corshamtech.com
;=====================================================
;
; First, define some common ASCII characters
;
LF        equ  $0a
CR        equ  $0d
;
; Where the Extended KIM monitor starts in memory
;
EXTKIM         equ  $e000
;
; These are subroutines and addresses in the extended
; KIM monitor we can use.
;
          bss       ;uninitialized data segment
          org  EXTKIM-6
ColdFlag  ds   2    ;cold start flags
ExtensionAddr  ds   2    ;address of extension ptr
HighestRam     ds   2    ;highest available RAM
          org  EXTKIM
extKim         ds   3    ;extended monitor
outch          ds   3    ;output A to console
getch          ds   3    ;get a key and echo
getchne        ds   3    ;no echo - KIM can't do it
spare1         ds   3    ;future - console stat
putsil         ds   3    ;print string after JSR
getHex         ds   3    ;get hex value in A
PRTBYT         ds   3    ;print A as hex
getStartAddr   ds   3
getEndAddr     ds   3
getAddrRange   ds   3
;
; There are more vectors but I didn't need them
;
;=====================================================
; The actual sample
;
          code
          org  ExtensionAddr
;
; Set up the pointer to our sample extension...
;
          dw   Extension
```

```
;
; This is the table of commands being added.  Each
; entry has exactly five bytes:
;
;    Single character command
;    Address of code for this command
;    Descriptive text for this command
;
; After the last entry, the next byte must be zero
; to indicate the end of the table.
;
          org  $0400
Extension db   'Z'  ;adding the 'Z' command
          dw   zCode     ;pointer to code
          dw   zHelp     ;pointer to help
;
          db   0    ;END OF EXTENSIONS
;
; The descriptive text...
;
zHelp          db   "Z ........... Describe a zoo",0
;
; And the actual code...
;
zCode          jsr  putsil    ;call display function
          db   CR,LF
          db   "A Zoo is a place with "
          db   "lots of animals."
          db   CR,LF,0
          jmp  extKim    ;return to Extended KIM
```

Revision History

Version	Changes
A	Initial Beta.
1	Initial release

Errata

REV 1 Incorrect Chip

IC4 is incorrectly identified as a 74LS241 on both the schematic and PC board. The device is actually a 74LS244.

Updating Rev 1 PC Boards

Users who buy assembled boards do not need to do any of these steps, as they were applied when we built your board. For someone building from a bare board, these steps will ensure the extended memory works properly. Without the mods, the board functions except that writes to the DAT registers will result in memory corruption. I.e., if you only use the base 64K RAM then none of these mods are needed.

To perform the modifications, you will need a sharp hobby knife to cut some traces, some #30 wire, a stripper for the wire, and a soldering iron. It is recommended that all cuts be made prior to installing any components, as at least one trace is obscured once an IC socket is installed, necessitating several more cuts and jumpers.

Cuts

1. Cut trace on IC6 between pins 30 and 32 (bottom side) close to pin 30 (leave trace from pin 32 to IC7 pins 1 and 28.

2. Cut trace between IC4 pin 19 and IC20 pin 7 (bottom).

3. On top of board, cut short trace from IC4 pin 19 to the via immediately adjacent to it.

4. On bottom, cut trace from IC4 pin 13 to ground.

5. Locate IC10. Between pins 6 and 7, and 8 and 9 is a trace. Cut that trace. It does not matter if it is cut above or below IC10. Follow the trace down to a via right above SS-50 pin30. You'll need to know where this pad is for the next step.

Jumpers (install IC sockets before doing these)

1. On the bottom of the board, solder a wire from IC4 pin 7 to the pad located in the previous step. Verify continuity from IC4 pin 7 to SS-50 pin 41.

2. Install jumper on bottom side from IC6 pin 30 to IC14 pin 8.

3. Install jumper from IC4 pin 19 to IC15 pin 1.

4. Install jumper from IC4 pin 13 to pin 14 (adjacent pins).

5. On bottom, install jumper from IC4 pin 1 to IC20 pin 7.

New Part

1. Install a 6.8K resistor on bottom side of board on IC1 between pins 7 and 32.