MLDL2000 User Manual

Introduction

This document serves as the User Manual for the M2KM: MLDL2000 Manager software. All references to MLDL2000 refer to version 1 and 2 unless indicated. This document should be used in combination with the MLDL2000 Specifications. It is assumed that the user is familiar with the operation of the HP41, especially with regards to the operation of plug-in modules. The software may be updated regularly while the manual is updated less frequently.
The MLDL2000 is designed to make the most out of your 30+ years old device, have fun with it!

Meindert Kuipers *Email:* meindert@kuiprs.nl
December 2010 *Website: www.kuiprs.nl*

References

- MLDL2000 Specifications and all references and credits in it www.kuiprs.nl
- HEPAX and MLDL2000 How-to (written by Howard Owen) www.kuiprs.nl
- SDK41 by Warren Furlow www.hp41.org

Conventions

$00FF	Hexadecimal numbers are used to indicate addresses, leading zero's are typically used to indicate the total possible range, e.g. $00FFFF. The 'x' character is used for a "don't care" situation
$040	The '$' character is used to indicate hexadecimal values when context is not clear.
17o	Octal values are indicated with the character 'o'
HP41	HP41 indicates all versions of the HP41 calculator, including HP41C, CV, CX, etc
TBD	To Be Defined: information is not fully specified yet

Copyrights and Disclaimer

Version History

1.65	November 2010	Split software manual for easier maintenance	MK, done
1.70	December 2010	Version for release	MK, done

IMPORTANT INFORMATION

The MLDL2000 is a powerful means of expanding the HP41 and using those old modules that you always wanted to have or make your own. The nature of the MLDL2000 requires the user to think before connecting the MLDL2000 to an HP41. Make certain that you are aware of potential conflicts between modules and peripherals at various locations, both physical, in the MLDL2000 and in the HP41 memory map. Check the website at www.kuiprs.nl for possible updates or answers to your questions. The M2kM Manager is originally written for Windows XP and tested on Windows Vista and Windows 7. Linux and MacOS versions are being prepared.
The actual software is update more frequently than this manual, so there may be differences between this manual and the software you are using. This document is written for version 1.70 of the M2kM software, and Firmware version 1.70 or higher.

Table of Contents

MLDL2000 Features

Current features of the MLDL2000 are:

- 255 ROM Banks of FLASH EPROM memory, each 4k * 10 bits, can be relocated to any HP41 ROM location at any bank, including the HP41 System ROMS
- Up to 255 banks of MLDL SRAM memory, each 4k * 10 bits, can be relocated to any HP41 ROM location at any bank, including the HP41 System ROMS
- USB interface for I/O and programming FLASH and SRAM
- Each ROM/RAM block can be individually enabled or disabled
- In-system Programmable CPLD as control interface to allow upgrades and/or functional changes
- M2kM: MLDL2000 Manager: software for managing the MLDL2000 contents, up- and downloading ROM images
- The MLDL2000 V2 to come ….

Technical Data

Memory	1M * 16 bit FLASH memory (>100.000 erase/programming cycles, >10yrs data retention)
	512K *16 bit SRAM memory (some units have 1M * 16 bit SRAM, serial numbers below 2050)
ROM Images	255 possible ROM images in FLASH memory
	Up to 255 possible ROM images in SRAM
Bank Switching	4 banks per page for pages $0-$F, odd and even page share bankswitching with HEPAX support
Logic	Xilinx XPLA3 CPLD with 384 macrocells (XCR3384)
	>100.000 erase/programming cycles, >10yrs data retention
Interface	FTDI-based single-chip USB 2.0 Controller (FT2232C)
Power	MLDL2000 is powered by the HP41 through 3.3V regulator
	SRAM can be battery backed up
	The USB controller is powered through the USB cable
	Power consumption: 6-7 mA additional to the HP41 when running.
Speed	MLDL2000 supports all known speeds of the HP41

MLDL2000 Software Installation (V1 and V2)

First download and install FTDI USB drivers before connecting the USB interface to your PC. These can be downloaded form www.ftdichip.com. Double check if you are using the drivers for the FT2232C controller with D2XX functionality for your operating system. The combined driver (VCP and D2XX) should work just fine. Install according to the instructions that come with it.
After the drivers are installed connect the MLDL2000 (plugged in your HP41) to a USB of your computer. A USB hub in between should not make any difference. Automatic installation is usually fine, just follow the instructions of your PC. Since the USB controller has two channels it will do the installation twice.
The M2kM software does not require specific installation instructions or registry modifications. Just unpack the downloaded zipfile and install it in a suitable directory and launch it *after* the FTDI drivers have been installed. On Windows Vista and Windows 7 it should be installed in a user directory!
Another file, BORLNDMM.DLL or fpcmemdll.dll is in the same zipfile as M2kM, this must be in the same directory as the M2kM executable. M2kM will generate a file MLDL2K.INI with settings if this file does not already exist. This file must be in the same directory as the executable.

The package consists of the following files:

- MLDL2K.exe — the main executable
- MLDL2K.ini — preferences and settings
- borlndmm.dll — dll with memory manager (up to V1.61B)
- fpcmemdll.dll — dll with memory manager (V1.70 and up)
- SYSTEMLABELS.TXT — textfile with mcode labels for the disassembler
- XROM.TXT — textfile with XROM numbers and names for the disassembler
- README.TXT — textfile with latest info

Upon execution of M2kM.exe, the software should recognize the MLDL2000 (if connected) with the correct serial number and you are ready to go!

ROM Images

ROM Images are soft-copies of existing HP plug-in modules (Application Pacs) like the MATH module, custom modules (PPC module, HEPAX) or modules that have been privately developed (SANDBOX or M2K ROM). Some modules contain specific hardware or functions that may or may not be successfully implemented in the MLDL2000. A list of specific modules that is known to be supported by the MLDL2000 can be found in the appendix. Not supported are modules that have specific I/O, such as the IL Module, Time Module, Printer, Extended Functions/Memory, Card Reader or Wand (Barcode Reader), although the ROM images of these modules may be loaded in the MLDL2000 and some functions may work.The primary means for loading ROM images in the MLDL2000 is by using the USB connection and the M2kM program. Alternatively a physical module may be copied to the MLDL2000 by using one of the ROM images for mcode development to copy a complete ROM to MLDL2000 SRAM and then saving it over USB to the PC. It is not possible to copy a ROM directly to FLASH memory from the HP41.

Standard ROMs are 4K * 10 bits, or multiples thereof. Currently M2kM supports various formats to load and save ROM Images. In this format a 10-bit ‘word’ is saved in a 16-bit word, with the 6 most significant bits set to zero. M2kM takes care of the proper conversion.

Please observe copyrights of ROMs and ROM Images!

I/O Functions (V1 & V2)

The I/O Interface is implemented on MLDL2000 V1 from firmware version 1.60. MLDL2000 V1 does not support the SPI registers.

The External Interface is described in detail in the Specifications (V 1.51). This paragraph explains the use of the I/O functions from the user perspective when coding in HP41 Mcode. The IO Interface is memory mapped in a ROM bank (see description of the Status Register) to allow for an easy and flexible IO Interface. Although multiple IO Banks could be mapped, these all alias to the same physical IO Interface. The IO Bank can be used as any normal ROM, and it can be put in both SRAM and FLASH under special conditions. The real I/O is enabled by registers which are embedded in the ROM memory map, and only these locations cannot be used for any HP41 code.
Within a ROM page that is mapped to an IO Bank the address x800 to x80F are reserved for I/O registers. Currently the following registers are defined:

Address	Read Operation (FETCH S&X)	Write Operation (WROM, $040)
$x800		I/O Register 0 write
$x801		I/O Register 1 write ALPHA data
$x802		I/O Register 2 write
$x803	I/O Register read	I/O Register 3 write
$x804		I/O Register 4 write
$x805		I/O Register 5 write
$x806		I/O Register 6 write
$x807	I/O Register read inverted (for test)	I/O Register 7 write
$x808	MLDL Status register	MLDL Control Register
$x809	Reserved (SPI Data read)	Reserved (SPI Data write)
$x90A	Reserved (SPI Status read)	Reserved (SPI Status write)
$x80B-x80F	Reserved for future use, read values may vary	Reserved for future use, write are ignored

The remaining parts of the block can be used for regular ROM code. The registers are mapped to the I/O even when the ROM block is mapped in FLASH (using a special setting in the Settings Registers). The SPI registers are intended to be used for the MLDL2000 V2 and are reserved in V1 of the MLDL2000. Using the reserved registers may lead to unexpected results.

Note that the I/O registers can be read under other circumstances, for example when doing a ROM checksum. Since the register contents may change at any time, ROMS with embedded I/O will typically not have a valid ROM checksum.

There is only one I/O register for read, this is aliased throughout the addresses $x800..$x807. The internal MLDL2000 I/O register is 16 bits wide, and when writing the 3 least significant address bits are transferred to the MLDL I/O register according to the following table:

C-register	C15	C14	C13	C12	C11	C10	C09	C08	C07	C06	C05	C04	C03	C02	C01	C00
	0	io14	io13	io12	io11	io10	io09	io08	io07	io06	io05	io04	io03	io02	io01	io00
	LSB part of C Mantissa (IO register address)				C-register S&X											

The I/O Registers can only be used by ROM's which have the correct Settings Register set. It is recommended to program NOP's in the I/O locations x800-x80F when creating an image of the ROM for example by SDK41.

When reading the I/O Register, the meaning of the bits are the following, for example after reading with FETCH S&X:

	C-register	C09	C08	C07	C06	C05	C04	C03	C02	C01	C00
		H_BSY	H_DAV	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

Only 8-bit data read transfers are possible with the 8 least significant bits to and from nibble 0 and 1 of the C-register. The 2 most significant bits are used to create a basic handshaking mechanism, which are connected to X_BSY (eXternal Interface Busy) and X_DAV (eXternal Interface Data Available) on the external MLDL2000 interface. The bits are called H_BSY (HP41 Busy) and H_DAV (HP41 Data Available).
Writes to the I/O register can be done with all 12 bits of the C register, S&X part. As long as the external interface did not read the I/O, reading the I/O register from the HP41 will generally return the 8 least significant bits of the written data. Bit 8 and 9 are always the H_BSY and H_DAV status bits, bits 10 and 11 read back as 0 (like in any normal FETCH S&X).
X_DAV is set immediately after a write to the I/O Register. This indicates to the external interface that data is available from the HP41. The signal will be reset upon a read by the external interface. The HP41 can use the H_DAV status bit to monitor when the external interface has read the data and new data can be written.

H_BSY is set by the external interface when writing to it. When the HP41 reads from the I/O Register the H_BSY bit is read with the data, and then immediately reset. This indicates to the external interface that the HP41 has read data. When the HP41 expects data from the external interface, it can simply read the I/O register until H_BSY is set. The data that is read with the H_BSY high signal is the new valid data. Be aware that when H_BSY is set, reading the I/O register will reset this bit.
Write and read operations should not be mixed, otherwise the handshaking mechanism may fail.
Data should only be written by the HP41if H_DAV is low, meaning that no data was be previously written to the IO Register by the HP41 or that the previous data was effectively read by the external interface. When data is written with H_DAV high the previous data will be overwritten. A read of the I/O register after a write will read the data back, with H_DAV set. Reading the I/O register when the data has been read by the external interface will return zero's in most cases, with H_DAV low.
It is important to note that the I/O register is affected by bank switching.

M2kM Software Overview

NOTE: M2kM is constantly being changed. Please check regularly on www.kuiprs.nl/hp41 for updates. Some menu items mentioned may be disabled and the screenshots in this manual may differ from the most recent version. The description in this version of the manual is valid for version 1.70, although the generics apply to previous versions as well. Of course I assume no responsibility for damage, crashes, blue screens and other disasters.

There is only one version of M2kM and it supports both V1 and V2 of the MLDL2000. For testing and production I have exactly the same version and the final release therefore contains functions that are not intended for casual use. This allows me to support you in case of problems by taking over your computer over the internet and use some of the special functions that you should not normally use, or to instruct you to use these. The test functions are potentially dangerous, although I have no proof that they can cause any physical harm to the MLDL2000, the HP41 or your PC. So you have these functions available, but please keep away from them, unless specifically instructed. The functions are described in this manual to a certain extent and the descriptions contain warnings.

M2kM V1.61B and older are written in Borland Delphi and requires a specific dll: BORLNDMM.DLL. This is available from my website. M2kM V1.70 and higher are written in Free Pascal with the Lazarus IDE, and uses fpcmemdll.dll as memory manger.

NOTE: The current version of M2kM does not support auto detection of plugging and unplugging the MLDL2000 USB connection. It is therefore advised to connect the MLDL2000 before starting M2kM and not to disconnect it while M2kM is running. Attempting to communicate with an unconnected MLDL2000 may result in crashing the M2kM application or in extreme cases may require a reset of the host PC.

Before using M2kM please install and test the FTDI USB driversM2kM (MLDL2000 Manager) software is used to communicate between the MLDL2000 and a PC using a USB connection. Only the Windows XP operating system (SP2) is tested. The basic functions of M2kM are:

- Uploading ROM images to FLASH and/or SRAM
- Downloading ROM images from FLASH and/or SRAM
- Erasing blocks of FLASH memory
- Managing the Settings Registers
- Communication between the HP41 and the PC
- Testing of the MLDL2000
- Modifying the MLDL2000 operation by loading a new firmware version
- Managing MOD files
- Disassembly of ROM images

Standard ROMs are 4K * 10 bits, or multiples thereof. M2kM supports the .ROM and .MOD format to load and save ROM Images. M2kM takes care of the proper conversion. The ROM and MOD formats are supported by various other HP41 programs, like Warren Furlows V41 PC simulator (see www.hp41.org). Be aware that the internal memory of the MLDL2000 is actually 16 bits, and more than 10 bits are used with firmware version 1.70 and up.

M2kM will work also when the MLDL2000 is not connected, and has a number of functions that may be very useful for non-owners of the MLDL2000.

M2KM software structure

M2kM is an application with 4 main tabbed windows:

- ROM Handler operations on a 4K or 8K ROM image
- Settings Registers organizing the contents of the MLDL2000
- Flash Handler operation son the non-volatile FLASH memory of the MLDL2000
- MOD File Handler manipulations of ROM images with the MOD structure

The menu and button bar offer a number of generic functions that can almost always be executed, independent of the tab state. A Status Line shows details of the connected MLDL2000, the currently open file and mode and a progress bar for up- and download operations.

Special functions can be executed with the ***Tools*** menu. Settings are stored in the M2KM.ini file and can be manipulated using the ***Preferences*** menu function.

Some functions or menus may be disabled, depending on having an MLDL2000 connected.

After launching the M2kM application you will see the main ROM Handler window with menu and status bar. The status bar will show if the MLDL2000 is connected and initialized and also shows the serial number, firmware version and memory type of your MLDL2000.

Through the menu the following choices are possible (when a choice is not possible, it is greyed out)

File	- Open	Opens and/or import a ROM, MOD or SR file
	- Save As	Saves a ROM or SR file
	- New MODFile	Creates a new .MOD file
	- Exit	Leave the M2kM application
MLDL2000	- Download	Load a ROM or SR file from the MLDL2000
	- Upload	Saves a ROM or SR file to the MLDL2000
	- Verify	Verify the contents of the MLDL2000
	- Connect	Connect and initialize MLDL2000 connection after plugging in
	- Disconnect	Disconnect MLDL2000 before unplugging
Tools	- Memory Editor	Random access to MLDL2000 Memory
	- CPLD Upgrade	Upgrade contents of CPLD
	- MLDL Test	Test of MLDL2000 (handle with care!)
	- BitBang	Individual control of the MLDL2000 I/O signals (handle with care!)
	- I/O Handler	Communication with the HP41
Preferences		Change the default settings of M2kM
Help		Shows the version of the M2kM application

Starting M2kM and connecting the MLDL2000

Connecting and Disconnecting the MLDL2000

When launching the M2kM application it will automatically check if an MLDL2000 is connected and will show the serial number, firmware version and memory configuration if it finds one. If not, all functions that communicate with the MLDL2000 will be disabled. You may connect the MLDL2000 later and use ***Connection*** - ***Connect*** for initialization. If you want to keep M2kM running and want to unplug the MLDL2000 first use ***Connection*** - ***Disconnect*** before pulling the plug. Automatic detection of plugging/unplugging of the MLDL2000 is not supported. Only one MLDL2000 can be connected.

The position and size of the various windows of M2kM are saved in the M2kM.ini file, and used when starting the program again.

Communication Speed

When transfer of data between the PC and the MLDL2000 is unreliable or your connected MLDL2000 is not recognized by M2kM, please decrease (or sometimes increase!) the Communication Speed with the Preferences dialog. The default speed is high enough to be comfortable, but is some cases there could be problems. The quality or length of the USB cable, presence of other USB devices or hubs on the system may influence the communication speed.

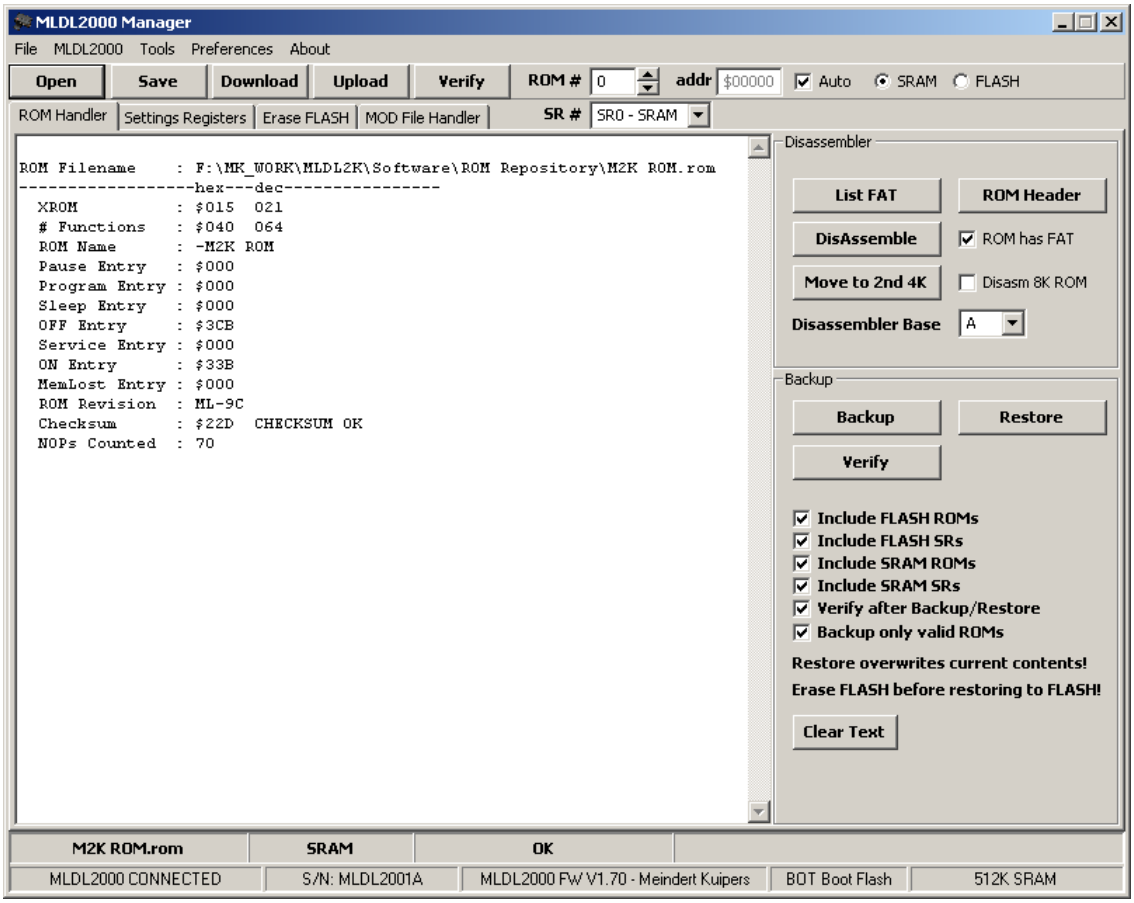
Firmware upgrading should normally be done at lower speed, therefore the JTAG speed is usually lower than the normal communication speed. You may also try to increase the communication speed. Maximum speed is 6 mbit/s. See the section on Preferences for more information.

Handling MLDL2000 ROM Images

The ROM Handler will typically be the most used tool for communicating with the MLDL2000. It contains the current ROM image for uploading to the MLDL2000, and a downloaded image or opened (or imported image) becomes the current ROM image. This ROM image can be uploaded to the MLDL2000, saved on disk, disassembled or copied to a MOD file. In addition, the ROM Handler offers tools for creating and restoring backups of the MLDL2000.

The text window shows the summary of the ROM contents (**ROM Contents** button), the FAT will be shown when clicking the **List FAT** button. This will also work with 8K ROM images, see the instructions for the disassembler.

Opening and importing ROM images



The ***Open*** button, or the ***File*** – ***Open*** is used to open a file. This can be any of the following type of files (extensions):

- .**ROM** file: traditional ROM format, direct image of the ROM without packing. Every 10-bit word is stored in a 16-bit word. After opening a ROM file this ROM image becomes the active ROM, and the header of the rom is listed.
- **.SR** file: MLDL2000 format for Settings Registers. This is a text file with the contents of the Settings Registers in hex characters, optionally followed by comments. After opening an SR file, the active tab will change to Settings Registers
- **.MOD** file: MOD format as introduced in the V41 emulator. The file will be opened, and the tab will change to the MOD File Handler. Since a MOD file may contain multiple ROM images, there will be no active ROM. The active ROM must be selected through the MOD File Handler
- **.HEX** file: imports a HEX file from a Clonix or NoVRAM device. The ROM image will become the current ROM
- **i41CX mailmod** files: these are textfiles, with .txt, .htm or .html extension. This function can be used to import MOD files that are mailed from the i41CX for iPhone or iPod with the MAILMOD function. These emailed files must be saved (as a complete email since there is no attachment visible) from the email client as html or text. The embedded MOD file will be extracted and converted to a standalone MOD file, the user will be asked to rename and save this file. Subsequently the MOD File Handler will become the active tab.
  (experimental) This function will also import saved emails that were sent with the MAILHEP functions. These are HEPAX RAM images and are not part of a MOD container, but rather .ROM images.
- **.ASC** file: this is used for testing only and imports a test file with a hex listing of a ROM image
- **.RAW** file: this is used for testing only, please do not use

Saving a ROM image

The Save button, or the File – Save As is used to save the current ROM file, a dialog will ask for a name. When the ROM Handler tab is active, only the current ROM can be saved as a .ROM file.

Copying a ROM image from the MLDL2000 unit

The **Download** button, or the **MLDL2000** – **Download** menu is used to copy a ROM image from the MLDL2000 unit to the ROM Handler. This will download the ROM image indicated with the **ROM #** selector and type (**SRAM**/**FLASH** radio button). The ROM characteristics will be listed. The Auto checkbox must be unchecked, and the relevant ROM must

be selected. An easier way to select and download ROM images is by using the SR Handler, where a ROM can be picked from the Contents list.

Saving a ROM image to the MLDL2000 unit

The **Upload** button, or the **MLDL2000** – **Upload** menu is used to save the current ROM file to the MLDL2000. The ROM number must be selected and SRAM or FLASH must be chosen.

There is no check for overwriting existing content to SRAM. Overwriting FLASH is generally not possible, unless the page is erased first. The Auto Checkbox may be used to automatically find the first free ROM page in the selected memory type. Since it will communicate with the MLDL2000 more frequent it may disrupt any ongoing operations in the HP41. This also happens when M2kM is started. The algorithm behind it checks the first 16 words of a ROM page. When it finds any bit set to 1 in the 6 most significant bits of any of the words (these bits are not used by the 10-bit ROM words) it will decide that the page is not is use. For SRAM this means that there was random data, for FLASH is means that it was erased. If not all of FLASH was erased this will be discovered when writing to the page.

NOTE: When uploading to FLASH Memory the target page will be checked. If it is not erased (all words $FFFF), programming is not possible since FLASH memory bits can only be programmed from '1' to '0'. In that case the complete sector containing the ROM needs to be erased first. First save any other ROM images in that sector. This check is not done when uploading to SR's. A time-out will occur when attempting to program a bit from 0 to 1. Programming FLASH takes more power than normal operation. The HP41 should be connected and powered.

Verifying a ROM image in the MLDL2000 unit

The **Verify** button, or the **MLDL2000** – **Verify** menu is used to compare the current ROM image against the selected ROM image in the MLDL2000. The location of the first word that differs is reported in the status line.

Backup and Restore

Backups are created in MOD files. The created MOD file has a special attribute to indicate that it is an MLDL2000 backup. MLDL2000 serial number, firmware version and date/time are written in the comments. Every module image has the Custom Header set to indicate memory type and address. A special attribute has been created for SR backups. It is possible to have more than 255 pages images in one MOD file, but this is not compatible with V41!

SRAM, FLASH and SR's can be individually saved and restored, this is indicated the checkboxes. When backing up FLASH or SRAM, the user can choose to save all ROM pages, including those which are not used (are not shown in the list in the SR Handler). This may be useful when there is custom information stored in FLASH or SRAM. Note that these pages are not automatically restored. MOD files only save a 10-bit word, while FLASH and SRAM have 16-bit words. For normal HP41 code this is not an issue.

A backup MOD file may be manipulated like any MOD file, ROM images can be individually extracted, uploaded, etc. When there are more than 255 images in one MOD file it is not possible to add pages but deleting pages is possible.

The restore functions will restore the contents of a backup MOD file to the MLDL2000. Note that any previous contents will be overwritten, and that all FLASH will be erased first! Use the MOD File Handler for restoring individual ROM images. The SR checkbox indicates if the SR's are restored as well. Pages which were marked as empty or unused will not be automatically restored.

Restoring SR's is a bit more complicated, since the complete sector must be erased first if these were in FLASH, and this might erase ROM images. Therefore SR's should be saved to SRAM first. Using the SR Handler the SR's can be individually downloaded from SRAM, edited, and uploaded to FLASH. Of course, if the SR's in SRAM were in use, these should be backed up first.

Changes to the MOD file format:

- Additional Category: MLDL2000_backup = 8
- Additional Page: SR Backup = $8F
- NumPages is $FF when there are more than 255 images in the MOD file
- HeaderCustom definition: Byte 0 $A5, identifier for MLDL2000 backup
  Byte 1 Number of images DIV $100
  Byte 2 Number of images MOD $100
- PageCustom definition: Byte 0 $A5, identifier for MLDL2000 backup
  Byte 1 $01 - SRAM (bit 0, lsb)
  $02 - FLASH (bit 1), bit 0 or 1 is always set
  $04 - ROM (bit 2)
  $08 – SR (bit 3), bit 2 or 3 is always set
  Byte 2 MLDL2000 ROM number, $00 for SR, $FF for empty page (FLASH only)

Disassembler

The Disassembler will allow disassembly of the current open ROM image. The disassembly listing will be shown in a separate window as text, from where it can be selected and copied to another application, for example to be saved or edited.
The checkbox **ROM has FAT** is checked by default, but should be unchecked when disassembling ROM images which do not have a FAT, for example the HP41 system ROMs. The **Disassembler Base Page** should be used when a ROM image has a known page in the HP41 address map.
The Disassembler does not do a lot of error detection, and assumes 'normal' ROM images. It is very well possible that M2kM crashes or shows error messages when attempting to disassemble a ROM image with random data (for example after downloading uninitialized SRAM).

- ***List FAT*** button: lists the FAT of the current ROM. When a 2nd ROM is loaded and **Disasm 8K** is checked it will list both FAT's. Do not use base pages 0 to 2 when there are labels in the FAT area as some XROM numbers might not show correctly.
- ***ROM Header*** button: lists the header of the current ROM file or of both if Disasm 8K is sleceted
- ***Disassemble*** button: starts the disassembler
- ***Move to 2nd 4K*** button: copies the current Rom to the 2nd 4K page for disassembly
- ***ROM has FAT*** checkbox: use this to indicate if a FAT has to be disassembled (uncheck when disassembling the system ROMs for example)
- ***Disasm 8K ROM*** checkbox: enables disassembling of an 8K ROM
- ***Disassembler Base*** pulldown: choose the base page for the disassembler.

The Disassembler uses 2 external files: XROM.TXT and SYSTEMLABELS.TXT. These are text files that supply standard XROM numbers and the names and addresses of the system entry points (Mainframe Labels). An explanation of the format is in the files. These files may be edited with other XROM numbers, labels or comments as required.

To disassemble an 8K ROM, use the following steps:

1. Open the 2nd 4K ROM image
2. Click **Move to 2nd 4K**
3. Open the first 4K ROM image
4. Check the box **DisAsm 8K ROM**
5. Click **DisAssemble**

When disassembling 8K ROMS, the Base Page must always be an even page (the 2nd block will be in the odd page).

Currently, the Disassembler has the following features:

- System entry points are read from SYSTEMLABELS.TXT (this may be switched off in Preferences)
- When a ROM with a base Page from $0 to $7 is disassembled, the local labels are read from SYSTEMLABELS.TXT (this may be switched off in Preferences). Use this when disassembling the system ROMs. As an additional feature, disassembly with base page $8 en $9 will also take the labels from SYSTEMLABELS.TXT. This allows non-system ROMs with a know label table to be disassembled with the local labels. In this case the labels have to be edited in the SYSTEMLABELS.TXT file with base address $8000 or $9000.
- XROM numbers are read from XROM.TXT (this may be switched off in Preferences)
- Automatic comments are generated for local GOSUB, and XROM's, comments for system labels or XROM's are read from SYSTEMLABELS.TXT or XROM.txt (this may be switched off in Preferences). Local XROM numbers are generated from the disassembled ROMs own FAT.
- User code is recognized and disassembled, line numbers are tracked and the 2 words preceding the user code are explained. Distance to the previous label is shown, key assignments are not translated (this is not supported in ROMs anyway).
- Synthetic characters in User Code and special HP41 characters in function names (Sigma etc) are not displayed correctly, but shown with the '\x' character followed by the hexcode. The append character is also not printed, but the '>' symbol is shown before the string (this is compatible with the User Code Compiler from Leo Duran)
- Labels for jump address are automatically generated (this may be switched off in Preferences)
- ROM checksum is verified
- Mnemonics type may be chosen in the Preferences Dialog (Jacobs/De Arras, HP or Zencode)
- SDK41Mode may be chosen to generate a listing that allows re-assembly with SDK41. Due to several limitations it cannot be guaranteed that re-assembly will create an identical ROM image
- Multiple consecutive (more than 3) NOPs may be skipped (see Preferences). Only NOPs that do not have a label will be skipped. In some cases labels may be generated by disassembling data.
- Cross Reference table for the used labels may be generated

There are also some limitations:

- Disassembly is never perfect. It is not always possible to distinguish data from instructions, in many cases data is incorrectly disassembled as instructions. This may also generate labels that are never jumped to.
- Function names and User Code which have their FAT in another Page (which is the case for many 8K ROMS) may have their function names and User Code incorrectly disassembled as mcode. Resolve this by disassembling both ROMs at the same time as an 8K ROM.
- When disassembling .MOD files, the ROM images must be moved to the ROM Handler first.
- When disassembling random data, the disassembler may be stuck in an endless loop or show system error messages

Various options for the Disassembler can be set in the Preferences dialog, including the layout of the listing. The user is encouraged to experiment with this. Note that the layout for SDK41 compatible listings is fixed and cannot be changed. The fields that can be used for modifying the layout are:

LABEL	Label at this address
ADDRESS	Address
HEX CODES	up to 3 words of hexcode
MNEMONIC	mnemonic of the instruction, or User Code instruction
MNEMONIC ARGUMENT 1	argument, for example S&X or the label of a jump, also User Code instruction
MNEMONIC ARGUMENT 2	usually the address of a jump
REFERRED ADDRESS	address where the jump goes to
REFERRED LABEL	label where the jump goes to
USER CODE LINE NUMBER	User Code line number
HEX CODE	one hex code at this address
COMMENTS	generated comments

When a line position is 0, the parameter will not be listed at all. When disassembling User Code the instruction is printed at the position of Mnemonic Argument 1, the line number can be at any other position. Take care not to overlap parameters, in general the order is which the parameters are listed also define the order in which the listing is generated, this means that Referred Address will overwrite Mnem Arg 1 when these overlap. The line positions are not used when generating SDK41 compatible disassembly.

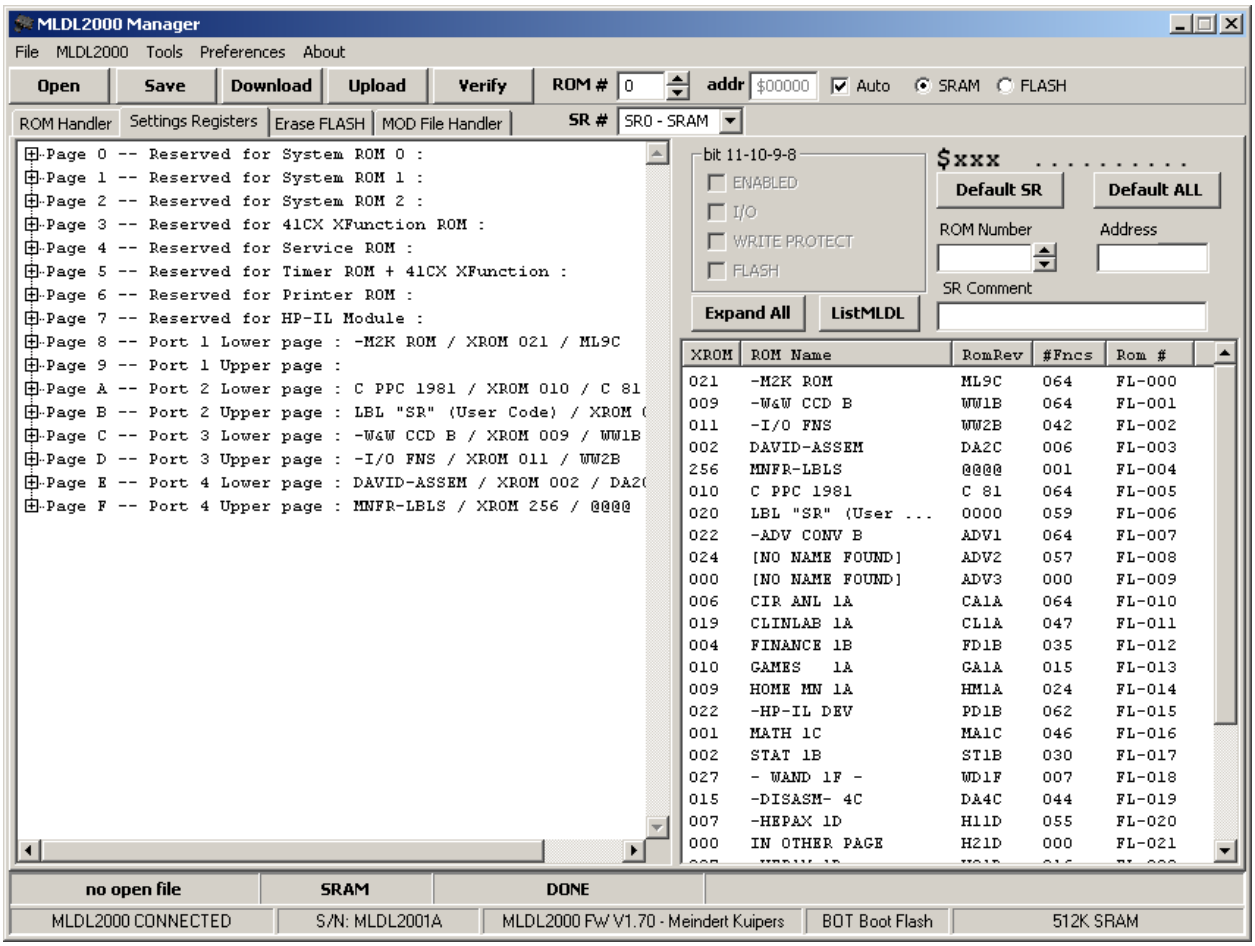
The disassembler listing can also be modified with the other Preferences settings:

- ***Auto Comments***: enable the automatic generation of comments that are read for the XROM.TXT or SYSTEMLABELS.TXT file and internally generated comments. When unchecked, no comments are generated at all
- ***Auto Label Generation***: the disassembler generates labels when it finds a jump or execute to an address, for the function entries, FAT and at many other points. A label is shown in the disassembly listing for these locations in the form of LB_1234, where 1234 is the address. When unchecked, no labels are generated at all
- ***Auto Label Fixed Addresses***: generate labels for the fixed ROM addresses, such as the entry points, FAT entries, etc.
- ***Use MainFrame Label File***: enable use of the SYSTEMLABELS.TXT file for translation of address into text for jumps/executes, and also for generating labels when diassembling system pages.
- ***Use XROM file***: enable use of the file XROM.TXT when disassembling user code for translating XROM numbers in function names with possible comments.
- ***Skip Multiple NOPS***: causes the disassembler to skip NOPS if there are more than 3.
- ***Arg1 = Label***: for jumps the (generated) lable is the first argument, then the address is listed. Deselecting this option prints the address first, and then the label
- ***MESL argument on next line***: a second line is generated with .MESSL “text” under the ?NC XQ MESSL, otherwise the text is printed on the same line as the ?NC XQ
- ***One disassembly line per word***: a listing is generated with one line per ROM address
- ***Generate label cross reference***: generates a cross reference listing at the end of the disassembly listing
- ***Do not print Address and HexCode***: generates a ‘clean’ listing with labels and mnemonics only
- ***Mnemonics Type***: sets the preferred mnemonics type: JDA (Jacobs/De Arras), HP, ZenCode or SKD41 compatible JDA (as that is slightly different from ‘standard’ JDA)
- ***SDK41 disassembly***: generates a listing that follows the SDK41 syntax and format is ready to be re-assembled with SDK41. It can not be guaranteed that re-assembly generates an identical ROM image, the listing must be reviewed and where needed edited. For best results select SDK41 compatible JDA mnemonics.

Settings Register Handler

The Settings Register Handler allows easy editing of the Settings Registers, it shows all possible ROM pages of the HP41. The Listing window allows expansion of the Page in the 4 possible banks. Please refer to the paragraph about the Settings Registers in the specifications for more details. The file format allows for comments. For a better overview the splitter between the SR tree and MLDL2000 contents list can be moved.

The file format for the .SR files is based on a text file for easy off-line editing. See the example SR file that comes with M2kM.



The Status Bar will always indicate the current open file, selected memory type and status. It also contains a progress bar to monitor up- and downloads.

- ***Open*** button: will open a .sr file, and will list the contents.
- ***Save*** button: will save the open .sr file.
- ***Upload*** button: will copy the open SR's to the MLDL2000.
- ***Download*** button: copies SR's from the MLDL2000.
- ***Verify*** button: checks the contents of MLDL2000 against the current open ROM image.
- ***SR Number*** edit/up-down control: indicates the SR number for up- or download. The actual SR address will be shown below it. SR Number and Address will be recalculated according to choice of FLASH/SRAM.
- ***SRAM/FLASH*** radio button: indicates memory type for up-or download
- ***Default SR*** button: Reset the current SR to $FFF, which indicates a disabled ROM ($3FF for < V1.61B)
- ***Default ALL*** button: Reset all SR's to $FFF ($3FF for version up to V1.61B).
- ***Collapse/Expand All*** button: Collapses or Expands the treeview.
- ***List MLDL*** button: lists the contents of the connected MLDL2000.

The ROM Number and SR attributes (the 4 checkboxes) allow easy editing of the SR contents. The SR that is selected in the treeview will be updated after every change, but not automatically uploaded to the MLDL2000. Right clicking in the SR tree will give various options for collapsing or expanding part of the tree.

When the MLDL List is present and an SR configuration is available, the ROM names will be shown in the HP41 Page Overview (if they are part of the current configuration). Selecting a ROM Page and Bank on the left hand side, and double clicking a ROM in the right window, will add that ROM to the SR overview to build a complete configuration. The individual SR settings may then be modified. When finished the SR configuration may be saved. Do not forget to upload the configuration to the MLDL2000, in the correct SR, to activate it and set the dipswitches to the matching position. Clicking the titles on the list header will sort the list.

The SR pulldown will immediately download the selected set from the connected MLDL2000. Any edits may be lost if not previously saved and/or uploaded.

The List MLDL function only finds ROM images that have the 6 most significant bits set to 0 in the first 16 words of the image, otherwise it will assume that there is no valid image. This means that images with all zeroes will be found. It is very unlikely that uninitialized SRAM (with random contents) will be detected as a valid ROM.
Right-clicking on a ROM image gives the following options:

- ***Refresh List***: Re-reads the list from the MLDL2000, same as the *List MLDL* button
- ***Download***: download the selected ROM image to the current ROM
- ***Save As***: download and save the current ROM.
- ***Wipe SRAM***: only works on SRAM: will fill the first 16 words of the selected ROM images with $FFFF and effectively removes the ROM image from SRAM. It will not show again in the list. Select *Refresh List* to verify this. Multiple ROM images may be selected by holding the SHIFT or CONTROL key while selecting the ROMS, FLASH pages will not be erased.
- ***Append to MOD***: append the selected ROM images to the current open MOD file. Will not work if there is no MOD file open (see MOD File Handler)

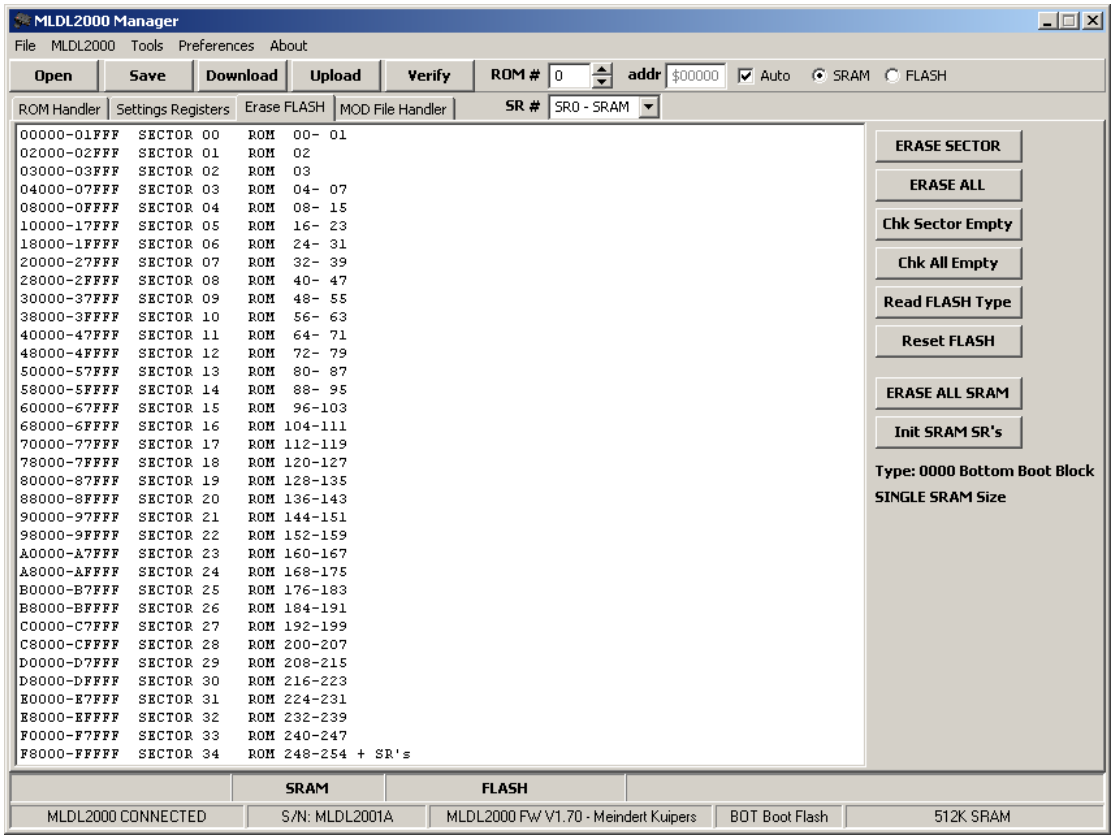
Remember that double clicking in the list will immediately add that ROM to the currently selected SR entry.

NOTE: When uploading an SR configuration in FLASH Memory, there is no check if the data can actually be programmed. FLASH memory bits can only be programmed from '1' to '0'. A time-out error will occur when attempting to program a bit from '0' to '1'. In that case the complete sector containing the SR must be erased first.

When experimenting with the Settings Registers, always do this in SRAM, since the individual bits may be changed if a mistake is made. This is much more difficult in FLASH, because the complete sector containing the SR's must normally be erased, wiping out all other SR's in the process. There is no functional difference between SR's coming from FLASH or SRAM. Once a certain SR configuration is final, just download it from SRAM, select FLASH in the FLASH/SRAM radio button and choose 1 of 4 SR sets, and then program to FLASH.

FLASH Erase Handler

The buttons for Open, Save, are not relevant here. The list window allows selection of one of the FLASH Memory sectors. It is not possible to select multiple sectors. Depending on the FLASH type used in your specific MLDL2000 version, the sector layout may differ.



- ***ERASE SECTOR*** button: Erases the selected sector.

- ***ERASE ALL*** button: Erases the complete FLASH memory. This may take up to one minute and no progress bar is shown.
- ***Chk Sector Empty*** button: verifies if the selected sector is indeed empty. When it is not empty, the address of the first non-$FFFF word will be shown in the statusbar. The check stops at the first non-$FFFF word is finds.
- ***Chk ALL Empty*** button: verifies all of FLASH memory (may take a longer time depending on the communication speed and block size settings) and shows empty sectors. If a sector is not empty, the address of the first non-$FFFF word will be shown in the statusbar. The check stops within a sector after the first non-$FFFF word it finds.
- ***Read FLASH Type*** button: re-reads the FLASH memory type. This is done automatically after connecting the MLDL2000.
- ***Reset FLASH*** button: Under certain circumstances the FLASH memory may 'hang' in a programming mode, for example after a time-out. This can be verified by reading an alternating pattern from FLASH. Use this button to recover from that situation.
- ***ERASE ALL SRAM*** button: initializes all SRAM to the value $A5A5 and deletes any ROM images, initializes SR's to their default value of $0FFF.
- ***INIT SRAM SR's*** button: initializes SR's in SRAM to their default value of $0FFF.

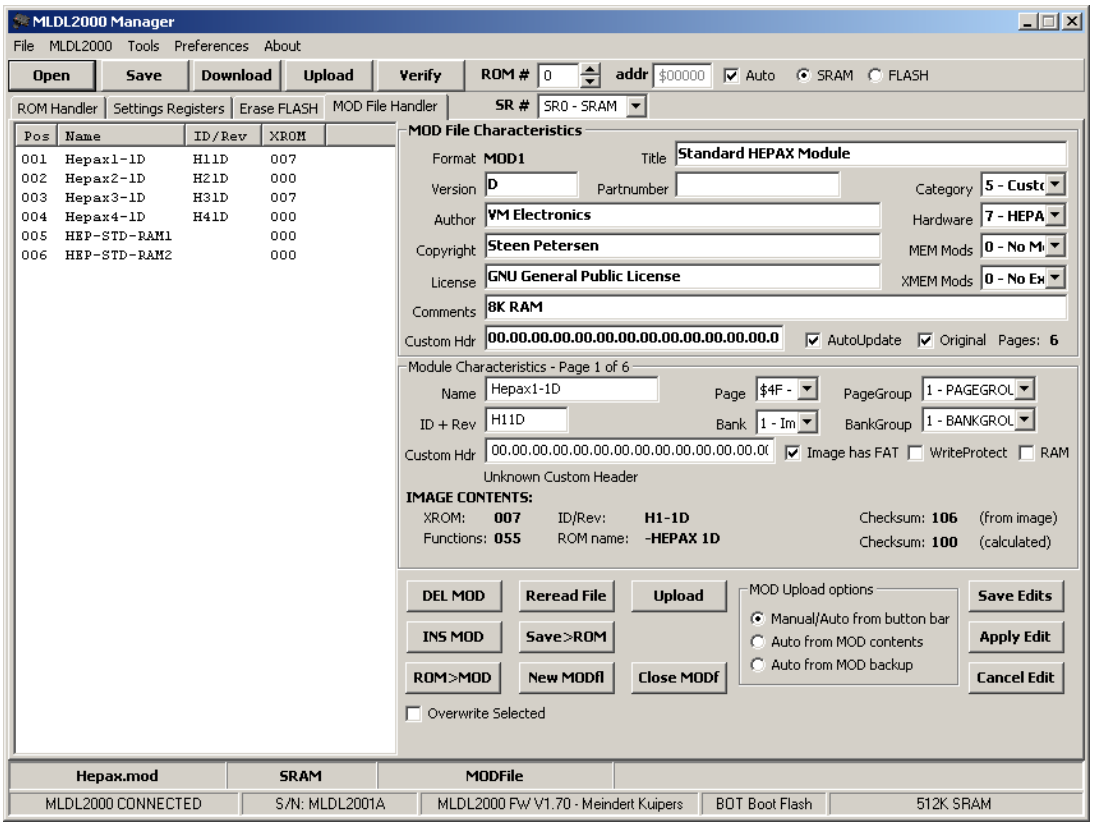
The time-out values may be changed in the Preferences dialog. This is not recommended, but may be required in rare circumstances, for example when the FLASH cells have been programmed many times and become more difficult to reprogram or erase. The FLASH cells are normally guaranteed at least 1 million write cycles per sector.

Erasing and programming FLASH takes more power than normal operation. The HP41 must be connected to USB and preferably powered by an external adapter or batteries.

MOD File Handler

MOD Files are container files that may hold up to 256 ROM images. The advantage is that, apart from the ROM image itself, a MOD file also contains information about the type of ROM, comments, manufacturer and how it should be used. MOD files are introduced by Warren Furlow for his V41 emulator and supported by the i41CX emulator for iPhone and iPod/iPad. The exact format is described in the V41 sources that can be downloaded from www.hp41.org. The MLDL2000 uses some minor additions to some of the field definitions.

MOD files may be created, opened, edited, saved and up- and downloaded from the MOD File Handler in M2kM. Automatic configuration like in V41 is not supported by M2kM. For disassembly the ROM image must be copied to the ROM Handler. The MLDL2000 backups are in MOD file format.



When opening a MOD file, the ROM images are shown in the list on the left. The MOD file header is shown on the top right, the Module Characteristics of the first ROM on the lower right. Clicking one of the ROMs will show its contents on the lower right. Note that the ROM image itself cannot be edited, only the parameters in the headers. Some information from the image is shown. A backup copy of the MOD file is made immediately after opening the file. It has the same name as the MOD file with the extension .BAK. Clicking the headers in the list will sort the list. When opening a MOD file, some checks are done to prevent opening a file with non-compatible contents.

Manipulation of the MOD files can be disk intensive. Only one ROM image is kept in memory, the other images are read whenever these are requested. Edits must be specifically saved before selecting a different ROM page in the MOD file. The various fields may be edited at any time. In some cases changes are immediately written to disk.

Backup files from the MLDL2000 are stored as MOD file, and can be handled as such without problems. Care should be taken when the backup file contains the Settings Registers. These can be recognized by the Custom Header, and the value of the Page (this will be $8F). It is possible to handle the SR backup just as a normal ROM image, but the results will be unpredictable. Be aware that the format in which the SR's are saved in a backup file has changed from version V1.70, and automatic conversion is not available.

To Save, Apply or Cancel edits made in the MOD file:

- ***Save Edits*** button: Will write the changes to disk for both the MOD file header and the ROM.
- ***Apply Edit*** button: Activates the changes, the various fields will be updated and refreshed. Edits are not yet written to disk.
- ***Cancel Edit*** button: Restores the original contents of the MOD file fields. Must be done before clicking *Save* or *Apply*.

Manipulation of the MOD file (some functions are available only in the popup menu in the list).

- ***Reread File*** button: Re-reads the file from disk and updates the list with the changes that may have been done.
- ***New MODf*** button: Creates a new MOD file with zero ROM images and an empty header, the Save As dialog will be started.
- ***Close MODf*** button: Saves and closes the current MOD file.
- ***Save>ROM*** button: Saves all selected pages to .ROM files. The name of the ROM (Module name in Header) will be used as the filename, it there is no name or the file exists, the Save As dialog will be shown.
- ***ROM>MOD*** button: Appends the current open ROM from the Rom Handler (this may be downloaded from the MLDL2000 or opened from disk) to the current MOD file. The ROM page parameters are cleared and may be edited. If the checkbox *Image has FAT* was selected before clicking *ROM>MOD* the name field will be filled. The *ID+Rev* field will always be filled from the image contents. Click *Save Edits* and *ReRead* File. Multiple ROM images may be added to the MOD file in this way. A maximum of 255 ROM images can be in any single MOD file. When the *Overwrite* checkbox is checked the first selected ROM image will be overwritten.
- ***Overwrite*** checkbox: when checked the *ROM>MOD* function will not append, but overwrite the current selected ROM image. To prevent unintended overwrites the checkbox is cleared after every *ROM>MOD* action.
- ***INS MOD***: this button inserts an empty page before the selected page. This is done in the order the pages are in the MOD file, the actual sort order in the list is ignored.
- ***DEL MOD***: removes all selected pages from the MOD file. Confirmation is asked.
  Upload ROM: Uploads the selected ROM images to the MLDL2000. See explanation below.

When uploading ROM images to the MLDL2000 the MOD Upload options (radio buttons) are active in combination with the ROM # (and address) and Auto checkbox. When a MOD file contains Settings Registers (for example from a backup) these will not be uploaded. Multiple selected pages may be uploaded and the free pages will be automatically found with the algorithm used to find ROMs in the SR Handler, modules occurring in that list will not be overwritten. FLASH will always be check if it is erased. The following options can be used:

- When the Auto checkbox (in the button bar) is not checked, only manual loading is supported, the MOD Upload Options are ignored. The ROM number and Memory Type as indicated in the button bar will be used, and only the first selected image will be uploaded. Uploading will be just like loading a single image from the ROM Handler.
- When the Auto checkbox is checked, multiple selected images may be uploaded. This is always done in the order as the images appear in the list. When errors occur, the complete upload will be aborted. The radio buttons in MOD Upload Options are used as follows:
  - *Auto from Button Bar*: will automatically find free ROM space in the memory type indicated by the SRAM/FLASH radio button. All selected pages will be uploaded if there is enough free space. FLASH memory will be checked first if it is erased.
  - *Auto from MOD contents*: same as option above, with the exception that pages which have the RAM attribute set (in the MOD file header) will always be uploaded to SRAM, other pages will be uploaded to FLASH or SRAM as indicated in the button bar.
  - *Auto from MOD backup*: the settings in the button bar will be ignored, the custom header field will be used to individually restore pages from a backup file. Note that in this case an existing ROM in SRAM may be overwritten.

In the list multiple ROM images may be selected by holding the SHIFT or CONTROL keys while selecting a ROM. Right clicking in the list activates a popup menu with the following choices (some of the functions above are also available in the popup menu):

- ***Delete Selected Pages***: Deletes the selected ROM images from the MOD file. Confirmation is asked.
- ***Insert Before Selected***: inserts an empty page before the selected page. This is done in the order the pages are in the MOD file, the actual sort order in the list is ignored. It is not possible to add pages to a backup MOD file.
- ***ROM>MOD***: (append or overwrite): appends (or overwrites) the current open ROM from the Rom Handler, see ROM>MOD button.
- ***Reread File***: Re-reads the file from disk and updates the list with the changes that may have been done.
- ***Extract to ROM Handler*** (1st 4K or 2nd 4K): Copies the ROM contents to the ROM Handler for saving as .ROM, disassembly, up- or downloading. When multiple pages are selected, the second one is copied to the 2nd 4K block..
- ***Extract to SR Handler*** (0..3): Copies one of the SR's from the MOD file to the SR Handler for verification or uploading to the MLDL2000. It is not possible to immediately upload an SR to the MLDL2000. The contents must come from an SR backup, and the SR page must be selected, otherwise the result in the SR handler will be undefined.
- ***Extract selected to .ROM file***: Saves all selected pages to .ROM files. The name of the ROM (Module name in Header) will be used as the filename, it there is no name or the file exists, the Save As dialog will be shown.
- ***Extract to SR file***: Extracts all 4 SR backups to an .sr file. The name is taken from the Page Name field. If this name is undefined, a new filename will be prompted for. 4 files will be created, with the name ending in '_srx', x is 0..3. The individual files may be opened with the SR Handler for editing and/or uploading.
- ***Upload to MLDL2000***: See Upload button.

To add a ROM image which is in the MLDL2000, it must first be downloaded to the ROM Handler, or it can be appended to the current MOD file from the list in the SR tab. The last option supports adding multiple (or all) ROM images. When adding multiple ROM images, the file is immediately written to disk and it is assumed that the ROMs have a FAT to update the ROM name and ID/Rev. Also the Custom Header of the ROM page will be filled with the MLDL2000 address and memory type in the same way as in the MLDL2000 backup files.

NOTES

- Uploading (multiple) ROM images from MOD files to the MLDL2000 still require the user to create the correct SR settings. For example the Write Protect attribute from the MOD file must be set in the correct way in the SR's, otherwise there will be no write protection. MOD files created with M2kM may not always work in the V41 emulator, depending on the settings of the attributes. After the upload, refresh the module list in the SR handler to check where the images have ended up.
- M2kM uses the Custom Header sections. This may conflict with other applications using these fields.
- M2kM defines extra attributes for storing SR's and backups. MOD files created with these attributes should not be loaded in V41.
- Be aware that the format in which the SR's are saved in a MOD backup file has changed from version V1.70, and automatic conversion is not available. When restoring pre V1.70 MOD backups the restore of the SR's must be disabled.
- Future changes in V41 may results in different MOD file definitions and interoperability cannot be always guaranteed.
- The MOD file will remain open for reading and writing in M2kM until specifically closed, until another MOD file is opened or until M2kM is closed. When the file is open, other applications like V41 may have no access to the file.

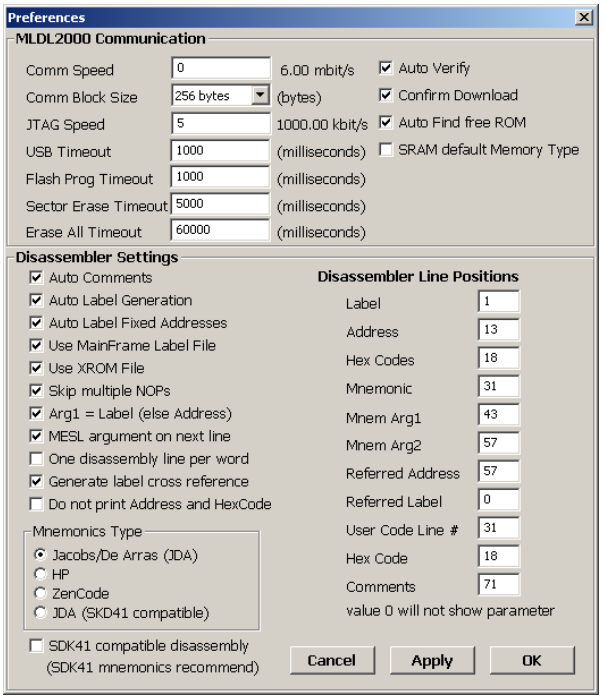
Preferences

The Preferences dialog is used to set the default settings of M2kM. There are two sections, one for the Disassembler, and one for Communication. Apart from the settings in the preferences screen, some other options are automatically saved in the M2kM.ini file:

- Screen positions and sizes of all windows
- Position of splitter in SR tab
- Memory type (SRAM or FLASH)

The effects of the Preferences settings are as follows (diassembler option are explained in the disassembler section):

- ***Comm Speed***: Defines the default communication speed between the USB controller and the MLDL2000. A value of 0 indicates the highest speed (6 mbit/s), a value of 65535 the lowest speed. Increase the value when communication is unreliable. Normally the speed should be as high as possible, and 6 mbit/s works in most cases. The communication speed may depend on the USB cabling, use of USB hubs, performance of your computer and or diskdrive and possible other factors.

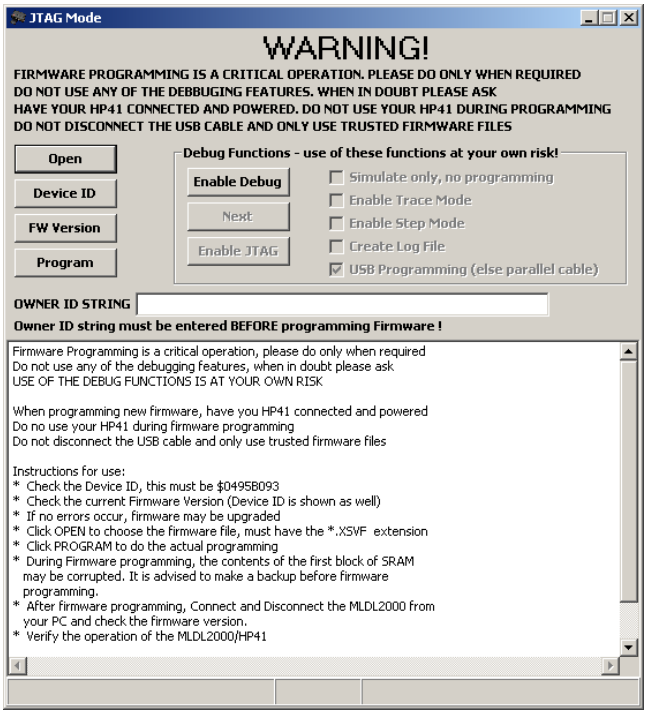


- ***Comm Block Size***: Defines the default USB communication block size. The larger the block, the faster the communication. Change the value when communication is unreliable.
- ***JTAG Speed***: Defines the communication speed between the USB controller and the MLDL2000 when upgrading firmware. This should be about 300 kbit/s. A value of 0 indicates the highest speed (6 mbit/s), a value of 65535 the lowest speed. Increase the value when Firmware upgrading is unreliable.
- ***USB Timeout***: Defines the default USB timeout value (in milliseconds) when M2kM is communicating with the MLDL2000. Failures may occur, for example when unplugging the USB cable. This value should not normally be changed.
- ***Flash Prog Timeout***: Defines the FLASH timeout value (in milliseconds) when writing a single word to FLASH. This timeout occurs when attempting to program a '1' to a '0', or when the FLASH is damaged.
- ***Sector Erase Timeout***: Defines the FLASH timeout value (in milliseconds) when erasing a sector. Errors may occur when there is not enough power (low BAT). Sectors are normally erased in about 300 milliseconds.
- ***Erase All Timeout***: Defines the FLASH timeout value (in milliseconds) when erasing the complete FLASH memory. Normally erasing takes about 30 seconds. Errors may occur when there is not enough power (low BAT).

The time-out values for programming or erasing should normally not be changed, but may be required in rare circumstances, for example when the FLASH cells have been programmed many times and become more difficult to reprogram or erase. The FLASH cells are normally guaranteed at least 1 million write cycles per sector. When errors occur, first reset FLASH and verify if the HP41 has enough battery power.

- ***Auto Verify***: Enables automatic verify after downloading a ROM or SR's. Verify is always done after uploading.
- ***Confirm Download:*** Requires confirmation before downloading a ROM or SR. Confirmation is always required when uploading a ROM.
- ***Auto Find Free ROM***: Enables automatic search for free ROM space in the MLDL2000. This will enable more frequent and automatic communication with the MLDL2000, which could disrupt operation of the HP41.
- ***SRAM default Memory Type***: Will set SRAM to the default memory type for up- and downloads.

CPLD Upgrade (JTAG Mode)



The CPLD contains the MLDL2000 Firmware. The firmware may be upgraded when new functionality becomes available or when bugs are resolved. Firmware Programming is a critical operation, please do only when required. Do not use any of the debugging features, when in doubt please ask. The debugging functions are there to be able to support you in case of problems. Firmware upgrading takes more power than normal operation. When programming new firmware, have you HP41 connected and powered. Do no use your HP41 during firmware programming and use fresh batteries or an external power adapter. Do not disconnect the USB cable and only use trusted firmware files.

Instructions for use:

- Check the Device ID, this must be $0495B093
- Check the current Firmware Version (Device ID is shown as well)
- If no errors occur, firmware may be upgraded
- Click OPEN to choose the firmware file, must have the .XSVF extension
- Click PROGRAM to do the actual programming. The progress bar is shown. The actual programming consists of 4 steps, although these steps are not individually shown:

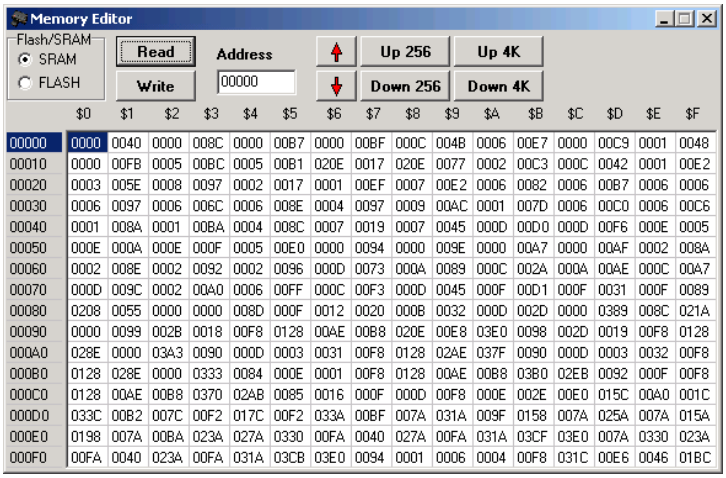
1. Erasing current firmware
2. Programming new firmware
3. Verify new firmware. Any programming problems will be visible during this step
4. Program Firmware version string (only after successful programming)

- During Firmware programming, the contents of the first block of SRAM may become corrupted. It is strongly advised to make a backup of the entire MLDL2000 SRAM and restore it after programming.
- Should Firmware programming fail, decrease the JTAG Speed (in the Preferences dialog) and try again. The default JTAG Speed is already lower than the regular communication speed.
- After successful programming, verify the operation of the MLDL2000 by doing a CAT 2 of some ROMs that are in the MLDL2000, and download a ROM image. A full test is not required.

BitBang

Please do not use this function. It is available for production and test/debug purposes only.

Memory Editor



The Memory Editor is useful for direct inspection and modification of SRAM and /or FLASH contents without the limitations of the ROM and SR Handler. It should be used with care.

- ***FLASH/SRAM*** radio button: Selects between SRAM and FLASH memory
- ***Address*** edit: start address of the listing.
- ***Read*** button: reads 256 16-bit words starting at *Address*.
- ***Write*** button: Writes all 16-bits words to the selected address.
- ***Up Arrow*** button: increases the *Address* by $10 and read.
- ***Down Arrow*** button: decreases the *Address* by $10 and read.
- ***Up 256*** button: increases the *Address* by $100 and read.
- ***Down 256*** button: decreases the *Address* by $100 and read.
- ***Up 4K*** button: increases the *Address* by $1000 and read.
- ***Down 4K*** button: decreases the *Address* by $1000 and read.

Data may be changed by selecting a cell and editing the information. There is no specific error checking of the edited data. The *Write* button writes all data in the edit windows back, it does not check if data is actually changed. After the write operation the block is read again.

NOTE: When changing words in FLASH Memory, there is no check if the data can actually be programmed. FLASH memory bits can only be programmed from '1' to '0'. A time-out error will occur when attempting to program a bit from '0' to '1'.

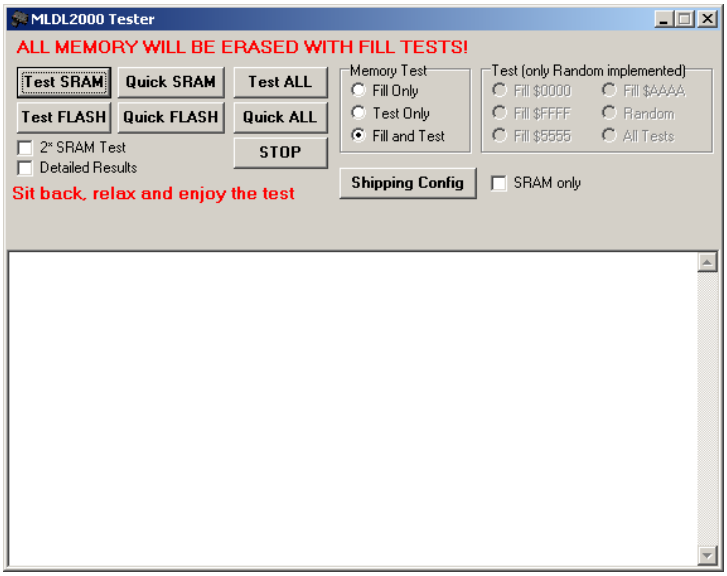
NOTE: SRAM addresses above $7FFFF do not exist, but are aliased with the lower part of SRAM except on the units with 2* SRAM.

MLDL TEST

The TEST window is used for 'production' test of the MLDL2000. Not all tests are currently implemented. Only the memory tests can be used. The button Shipping Config relies on absolute file paths and should not be used. The memory tests will fill the FLASH and/or SRAM with a random pattern, starting with a fixed seed. The random pattern is regenerated again while reading the memory contents. The random pattern virtually guarantees that any shorts or bad connections between traces on the PCB are detected. It may miss defects in individual memory cells.
Testing memory may take a long time, so please be patient. The tests can be interrupted with the STOP button (erasing FLASH cannot be interrupted).

- ***TEST SRAM*** button: Will test SRAM, or the 2* SRAM configuration is that checkbox is checked
- ***TEST FLASH*** button: Will test FLASH. FLASH memory will be erased!
- ***QUICK SRAM*** button: Will do a test of the first 4K SRAM block only
- ***QUICK FLASH*** button: Will do a test of the first 4K FLASH sector only. The sector containing the block will be erased first.
- ***TEST ALL*** button: First do a full SRAM test, then a full FLASH test
- ***QUICK ALL*** button: First do a QUICK SRAM test, then a QUICK FLASH test (see above)

- ***2* SRAM*** checkbox: for double SRAM configurations
- ***Detailed Results*** checkbox: when checked, all errors with the offending address and data will be shown, otherwise only one error message per block will be shown.
- ***MEMORY TEST*** radio buttons: selects the type of test. *Fill Only* will one write the memory, not test, *Test Only* will only test the memory, and not fill it before. *Fill and Test* will do both.
- ***Shipping Config*** button: Prepares the MLDL2000 in the default configuration. This depends on absolute file paths and should not be used.
- ***SRAM only*** checkbox: Prepares the MLDL2000 in the default configuration for SRAM only, FLASH is not programmed. This depends on absolute file paths and should not be used.



The test progress and results will be shown.

Limitations of M2kM

- Only one MLDL2000 can be connected to a PC.
- There is no monitoring of plugging/unplugging MLDL2000, this may lead to a crash of the application and/or the PC. It is best to connect the MLDL2000 before starting M2kM, and to remove it after closing M2kM or use the Connection menu.
- While communicating (meaning actively reading or programming) the MLDL2000 must be connected to the HP41. Do not run programs on the HP41 that use any of the MLDL2000 functions (like ROM images) while communicating. The MLDL2000 will be disabled while communicating, and no ROM images will be visible.
- M2kM is tested only on Windows XP (SP2) and Windows 7.
- There is a very small (theoretical) possibility that other USB devices with the FT2232C controller are recognized as an MLDL2000
- The error checking and recovery of M2kM may not catch all possible errors. Reset FLASH if you suspect problems, for example after a FLASH programming time-out
- FTDI offers a program for erasing and reprogramming of the EEPROM on the USB print. DO NOT USE THIS, or the MLDL2000 might not be able to work with M2kM again
- In case of problems, check if you have the latest version of the FTDI drivers, the M2kM software and if all software and drivers are installed correctly
- Verify only tests until the first faulty address, and will show the error address in the status bar
- In the current version there is a possibility to test the MLDL2000, but the functionality is very limited.

Supported ROMS

A list of modules that are known or expected to work with the MLDL2000 is maintained on www.kuiprs.nl. Feedback regarding success or failure of a certain modules will be appreciated. ROM Images are soft-copies of existing HP plug-in modules (Application Pacs) like the MATH module, custom modules (PPC module, HEPAX) or modules that have been privately developed (SANDBOX or ML ROM). Please observe the copyrights of these modules. Some modules contain specific hardware or functions that may or may not be successfully implemented in the MLDL2000. Not supported are modules that have specific I/O, such as the IL Module, Time Module, Printer, Extended Functions/Memory, Card Reader or Wand (Barcode Reader), although the ROM images of these modules may be loaded in the MLDL2000 and some functions may work.

HEPAX is working in a specific configuration. A paper written by Howard Owen (can be downloaded from www.kuiprs.nl) describes this in detail.

It may be interesting to note an experiment that I have done with a faulty Wand (barcode reader). When plugging in an HP41 it was not visible in CAT 2, nor could the ROM be seen with Mcode tools. The LED in the tip would light up however when the key was pressed on the Wand. I then obtained a copy of the Wand ROM, programmed it into the MLDL2000 and attached the (faulty) Wand to the HP41. It then turned out that the Wand was working perfectly with the ROM contents coming from another port!

The primary means for loading ROM images in the MLDL2000 is by using the USB connection and the M2kM program. Alternatively a physical module may be copied to the MLDL2000 by using one of the ROM images for mcode development to

copy a complete ROM Image to MLDL2000 SRAM and then saving it over USB to the PC. It is NOT possible to copy a ROM image directly to FLASH memory.

Warranty

All MLDL2000 products and services are supplied as is and without any warranty. Please understand that the whole project is a hobby project and definitely NOT for profit.

APPENDIX A: I/O MCode examples

The I/O Registers communicate with the USB interface and M2kM. In the current version, only register 1 is specifically reserved for ALPHA data. M2kM interprets any data as ASCII characters. This is more for demonstration than practical use. Actual examples are shown in the appendix. In all cases it is assumed that the ROM containing the code also contains the I/O registers. The basic routines are X2OUT and OUT2X, respectively for sending X to I/O Register 0 and reading the I/O Register. No handshaking is implemented here. M2kM now has an I/O handler added that can be used for transferring data.

```
; M2K ROM V2, contains I/O functions to support I/
; version for SDK41, ready for assembly with A41/L41
.JDA

; XROM X2OUT
; send the X-register to I/O register 0, no handshaking is used.
; X is converted with BCDBIN in the system ROM
; X must be <1000
.NAME       "X2OUT"                 ; 018 032 00F 015 094
[X2OUT]     CLRF    9
            READ    3(X)
            ?NCXQ   [BCDBIN]        ; convert to BIN, result in C.X

; subroutine to send C S&X to output, IOREG0
[SUBOUT0]   ?NCXQ   00D7            ; get own address PCTOC
            R=      5               ; and create I/O register address
            LD@R    8
            LD@R    0
            LD@R    0
            WROM                    ; write word
            RTN

; subroutine to read IOREG0
[SUBIN]     ?NCXQ   00D7            ; determine ROM page, PC in C[6:3], PCTOC
            R=      5
            LD@R    8               ;
            LD@R    0               ; send to output
            LD@R    0               ; I/O register location
            FETCHS&X                ; and read register
            RTN


; XROM OUT2X
; reads I/O register 1 to X. No stack lift, just very basic
.NAME       "OUT2X"                 ; 00F 015 014 032 098
[OUT2X]     ?NCXQREL [SUBIN]        ; read the data
            A<>C    S&X             ; save in A
            ?NCXQREL [BIN2BCD]      ; convert to a decimal
            WRIT    3(X)
            RTN

; below is a simple BIN->BCD subroutine
; LB_A567 from -M2K ROM
[BIN2BCD]   C=0     ALL
            ?A#0    S&X
            ?NCRTN
            LDIS&X  04B
            RCR     1
            A<>C    S&X
[BIN2BCD10] SETDEC
            C=C+C   M
            SETHEX
            C=C+C   S&X
            JNC     [BIN2BCD20]
            SETDEC
            C=C+1   M
            SETHEX
[BIN2BCD20] C=C-1   MS
            JNC     [BIN2BCD10]
            C=0     MS
            C=0     S&X
            R=      12
            RCR     9
[BIN2BCD30] RCR     13
```

```
                A=A-1   S&X
                ?C#0    @R
                JNC     [BIN2BCD30]
                A<>C    S&X
                RTN
```

The function below is an example which uses handshaking. The function A2OUT sends the APLHA register to I/O Register 1, character by character and waits until the handshaking is cleared before sending the next character. It ends with sending a <carriage return>. This is all handled by M2kM. The routine is not very code efficient. A time out counter is used to prevent a lock-up.

```
        ; subroutine to send one alpha character to the I/O register 1 (ALPHA output)
        ; input:  I/O address       in A.M
        ;         character to send in C.S&X
        ; use entry SENDAHSB (character to send in B.S&X) when calling
        ; with GOTO or GOSUB due to use of C-register
        ; use entry SENDAHS0 (character to send in M) to skip if char is NULL

        [SENDAHS0]   C=M                        ; get character from M
                     R=      1
                     ?C#0    R<                 ; character NULL?
                     ?NCRTN                     ; yes, return

        [SENDAHS]    B<>C    S&X                ; save char to write in B.S&X
        [SENDAHSB]   LDIS&X  080                ; time-out value
                     A=C     S&X                ; time-out value saved in A.S&X
                     A<>C    M                  ; get I/O address
                     A=C     M                  ; save back in A.M
        [SENDAHS10]  FETCHS&X                   ; read I/O register
                                                ; C.XS contains bits "0.0.X_BSY.X_DAV"
                     C=C+C   XS
                     C=C+C   XS
                     C=C+C   XS
                     C=C+C   XS                 ; sets carry if X_DAV set
                     JNC     [SENDAHS20]        ; carry not set, so X_DAv low, ready to write
                     A=A-1   S&X                ; decrement time-out counter
                     JNC     [SENDAHS10]        ; no time-out yet, try again
                     ?NCGO   [ERROF]            ; time-out reached, generate OVERFLOW error
        [SENDAHS20]  B<>C    S&X                ; character to write
                     WROM                       ; write to I/O and return
                     RTN

        ; entry to send <CR> with handshake to I/O register 1
        [SENDA_CR]   LDIS&X  00D                ; send <CR>, $0D, decimal 13
                     JNC     [SENDAHS]

        ; XROM A2OUT
        ; sends the ALPHA register to I/O register 1 with handshake. ALPHA is not disturbed.
        ; A NULL character marks the end of ALPHA. When done, a <CR> is send as last character.
        ; Assumes that the I/O registers are in this ROM!
        .NAME        "A2OUT"                    ; 001 032 00F 015 094
        [A2OUT]      ?NCXQ  00D7                ; get address of current ROM, PCTOC is not in A41/L41
                     R=      5                  ; and create I/O register 1 address
                     LD@R    8
                     LD@R    0
                     LD@R    1
                     A<>C    M                  ; save address in A.M
                     C=0     ALL
                     RAMSLCT                    ; select chip 0 with status registers
                     READ    8(P)               ; read 1st ALPHA register
                     RCR     6
                     M=C
                     ?NCXQREL [SENDREGP]        ; only 3 chards to send from P-register
                     READ    7(O)
                     M=C
                     ?NCXQREL [SENDREGA]        ; send out register
                     READ    6(N)
                     M=C
                     ?NCXQREL [SENDREGA]        ; send out register
                     READ    5(M)
                     M=C
                     ?NCXQREL [SENDREGA]        ; send out register
                     JNC      [SENDA_CR]        ; almost done, now send <CR>

        ; subroutine to send the M-register as 7 characters to I/O register 1 with handshake
        [SENDREGA]   C=M
                     RCR     12
                     M=C
                     ?NCXQREL [SENDAHS0]        ; send 1st character
                     C=M
                     RCR     12
                     M=C
                     ?NCXQREL [SENDAHS0]        ; send 2nd character
                     C=M
                     RCR     12
                     M=C
```

```
                ?NCXQREL [SENDAHS0]       ; send 3rd character
                C=M
                RCR     12
                M=C
                ?NCXQREL [SENDAHS0]       ; send 4th character
[SENDREGP]      C=M                       ; entry to send only 3 chards from P
                RCR     12
                M=C
                ?NCXQREL [SENDAHS0]       ; send 5th character
                C=M
                RCR     12
                M=C
                ?NCXQREL [SENDAHS0]       ; send 6th character
                C=M
                RCR     12
                M=C
                ?NCXQREL [SENDAHS0]       ; send 7th character
                RTN
```