POWER_CL Module

41CL Extensions ROM

User's Manual and Quick Reference Guide

©Photo By Jürgen Keller, 2011.

Programmed by Ángel M. Martin

June 2012

This compilation revision 2.W.01

Screen captures taken from V41, Windows-based emulator developed by Warren Furlow.
See www.hp41.org

CLWRITE Source Code written by Raymond Wiker.

Cover photo © Juergen Keller, 2011.
Inside photos © Geoff Quickfall, 2011

Acknowledgment.- This manual and the POWERCL module would obviously not exists without the 41CL. Many thanks to Monte Dalrymple for the development of the amazing CL board.

POWER_CL Module
Extension Functions for the 41CL

Table of Contents.


SYSTEMYDE INTERNATIONAL
41CL

1. Introduction.

Without a doubt the 41CL can be considered in many ways to be the pinnacle of the HP-41 system. It comes with a well thought-out function set to manage its capabilities, from the basic to the more adventurous ones – which have inspired the writing of yet further extensions to that capable toolset.

This module is designed to enhance and complement the YFNS function set, providing easier access to the many powerful capabilities of the 41CL platform. Some are function launchers, grouping several functions by their area of functionality into a single, prompt-driven one – like it's the case for the Plug/Unplug functions, the Baud rate, TURBO and MMU settings functions. A "launcher of launchers" sits atop these, providing quick access to 35 YFNS and other functions from a single key assignment.

Some others extend the functionality by providing new features and more convenient alternative to manual tasks. Examples of these are:

- A fully-featured ROM library catalog system, allowing direct plugging into the port of choice
- The *Page* Plug functions (alternative to the *Port* ones), including routines to handle page #4.
- HEPAX configuration and set-up, making the HEPAX integration a simple and reliable affair.
- Security functions to password-protect your machine from prying hands.
- Two powerful RAM and ROM editors, a hacker's real delight.
- An extended implementation of the Unit Management System, with Electrical units support and featuring an all-new Constants Library.

Other housekeeping functions roundup the set, making for a total of 64 functions tightly packed into a bank-switched 8k ROM. This was a design criterion, as the small footprint of the module (just one page, or effectively 4k) makes it ideal to combine with other utility packs, most notoriously the CCD OS/X (or its alter-ego AMC OS/X) for the ultimate control - so save some small exceptions there is no duplication between the two.

Page#4 Library and Bank-Switching.

The first thing to say about the POWER_CL module is that - being based on the CL_UTILS revision 4H - it extensively uses routines and functions from the Page#4 Library. Make sure the Library#4 it's installed properly on your system or things can go south. Refer to the Page#4 Library documentation to properly configure the Library#4 before you start using it.

This module is, to the author's knowledge, one of the very few ones using bank switching. The idea of using an auxiliary bank originated from the usage of the very long ROM image table required by **CLLIB** and **ROMLIB**. Even after completing the page#4-aware version of CL_UTILS, it was clear that a bank-switching version would be the ultimate solution, which has also enabled an enhanced implementation of the Unit Conversion functionality to be included.

A word of caution.

As wise men remind us, "*with power comes responsibility*". Indiscriminate usage of some of these functions can have unpleasant consequences, ranging from unexpected results and easy-to-recover machine lock-ups to more serious ones, losing HEPAX data. Functions have some built-in protection to ensure that they're used properly, but they are not absolutely foolproof in that such protection can always be circumvented. So beware, and as general rule "*if you don't understand something, don't use it".*

To help you with this, the more dangerous functions are marked with **WARNING** signs all throughout this manual. Avoid them if you're not absolutely sure that you know what they are for, and fully understand their operation. Note that the flash back-up functions that were available in the CL_UTILS module have been removed, as they used hard-coded addresses incompatible with V3 revision of the CL board - and because they were deemed to be too risky for casual users; so this follows the " better safe than sorry" rule.

It had to be said – so now that we got it out of the way we're ready to dive into the CL UTILS description and usage example. May you have a nice ride!

Function index at a glance.

Without further ado, here are all POWERCL functions:

#	Function	Description	Inputs	Output
1	***-POWERCL BS***	*Module Header / SPLASH*	*n/a*	*Checks for Library#4*
2	**ΣCL _**	Main CL Global Launcher	Prompts "B:C:H:M:P:T:U"	Launches selected Launcher
3	**ADRID**	ROM id# from address	Flash addr in Alpha	ROM ID in Alpha
4	**BAUD _**	Baud functions launcher	Prompts "1:2:4:9:I"	Launches selected function
5	**CHECKF**	Check ROM Configuration	none	OK/BAD message
6	**CLLIB _**	ROM Library Alphabetical	Prompts "A-Z"	Starts listing at selected letter
7	**HEPINI _"_**	HEPAX FileSys Init	Prompts for values	Iinitializes HEPAX File System
8	**HEPX _**	HEPAX Launcher	Prompts "4:8:6:I:D"	Launches selected function
9	**HEPYX**	HEPAX FileSys Init	# pages in Y, first page in X	Iinitializes HEPAX File System
10	**HPX4 _**	HEPAX Config 4k - CL	prompts for page#	Configures 4k HEPAX on CL
11	**HPX8 _**	HEPAX Config 8k - CL	prompts for page range	Configures 8k HEPAX on CL
12	**HPX16 _**	HEPAX Config 16k - CL	prompts for page range	Configures 16k HEPAX on CL
13	**MMU _**	MMU functions launcher	Prompts "C:D:E:G:T:?"	Launches selected function
14	**MMUCAT**	MMU Catalog	none	Sequential list of MMU Entries
15	**PLUG _**	PLUG functions launcher	Prompts for location	ROM with ID in ALPHA is plugged
16	**PLUGG _**	Plug Page	Prompts for page	Plugs ROM in page
17	**PLUGG? _**	Page Location MMU	Prompts for page	content of MMU entry for page
18	**PLUGGX**	Plug Page by X	page# in X	Plugs ROM in page
19	**PPG#4 _**	Page#4 Plug	Prompts "F:L:S"	Selected ROM plugged
20	**ROMLIB _**	ROM Library by type	Prompts "E:F:G:M:U:X"	Sequential list of ROMs by type
21	**ROMLST**	Shows all XROMs plugged	none	Strintg in Alpha
22	**SECURE**	enable password lock	none	Sets SECURE mode ON
23	**TURBO _**	TURBO functions launcher	Prompts "X:2:5:1:0:,:?"	Launches selected function
24	**UNLOCK**	disable password lock	asks for password	Sets Secure mode OFF
25	**UPPG4**	Clears MMU entry for page #4	none	MMU entry cleared
26	**UPLGG _**	UPLUG page	Prompts for page "6-F"	Unplugged if present
27	**UPLGGX**	UPLUG Page by X	Page# in X	Unplugged if present
28	**UPLUG _**	UPLUG functions Launcher	Prompts for location	Location is removed from MMU
29	**UPLUGA**	UPLUG by Alpha	Page Revision in Alpha	Unplugged if present
30	**XCAT _**	Xtended CATalogs	Prompts: "B:F:H:K:L:M:U"	Launches selected function
31	**XPASS**	change password	asks old/new passwords	Password is changed
32	**YBSP**	ALPHA back Space	string in ALPHA	Deletes rightmost character
33	**YCL>**	Clears string from ">"	string in ALPHA	Clears from ">" char to the right
34	**YCL-**	Clears string from hyphen	string in ALPHA	Clears from "-" char to the right
35	**YFNZ**	Page location of YNFS	none	page location in X
36	**YINPT _**	Y Input	prompts for characters	HEX entry plus control chrs
37	**YSWAP>**	swaps both sides of ">"	string in ALPHA	Alpha swapped around ">"
38	**YSWAP-**	swaps both sides of hyphen	string in ALPHA	Alpha swapped around "-"

39	*-SYS/EXT*	*Section Header / DTOA*	*Text in Display*	*Writes text to ALPHA*
40	**ASG _**	Multi-byte ASN	prompts for mnemonic	Assignment is made
41	**BFCAT**	Buffer Catalog	none	Shows present buffers
42	**BLCAT**	Block Catalog	none	Lists block contents
43	**CPYPG _ _**	Copy Page	FROM: in X, TO: in prompt	Copies page from/ to
44	**DCD**	Decode	NNN in X	HexCode in Alpha
45	**DTST**	Display Test	none	Shows display all lit up
46	**HEXKB _**	HEX Keyboard	prompts for entry	Codes NNN in X, string in Alpha
47	**PGSIG _ _**	Gets Signature for page	Prompts for page, 1-14	data string in Alpha
48	**RAMED _**	RAM Editor	address in X	Memory edited as per inputs
49	**READPG**	Reads page from HPIL disk	page# in X, FName in Alpha	Page read from disk
50	**ROMCAT _ _**	ROM id# CAT	prompts ROM id#, 1-32	Stats CAT-2 at this point
51	**ROMED _ _ _ _**	ROM Editor	prompts for address	ROM edited as per inputs
52	**SIGPG?**	Gets page# of matching sig	Signature in Alpha	Page# in X
53	**SPEED**	Recalls CPU cycles data	none	CPU cycles/sec in X
54	**T>BS _ _**	Base Ten to Base	number in X, base in prompt	converted number in ALPHA
55	**WRTPG**	Writes page to HPIL disk	page# in X, FName in Alpha	Page copied to Disk
56	**XCF _ _**	Xtended CF	prompts for flag#	Flag is cleared.
57	**XROM _ _,_ _**	ROM function Launcher	Prompts for values	Launches function
58	**XSF _ _**	Xtended SF	prompts for flag#	Flag is set
59	**-SI**	***Direct Unit conversion***	**value in X, string in Alpha**	**converted unit in X**
60	**APPEND _**	Appends text to Alpha	text in prompt	text appended to Alpha
61	**KLIB _**	Constants Library	4 displays of 5x each, SST	constant value to X, unit to Alpha
62	**SI-**	Reverse Unit conversion	value in X, string in Alpha	converted unit in X
63	**UCAT _**	Unit Catalog	section in prompt (1-5)	Lists units starting at section#
64	**Y/N?**	Yes/No input	none	only takes "Y", or "N"

Legend:

BLACK fns.	FOCAL code w/ MCODE entry
BLUE fns.	MCODE Functions
RED fns.	New/Enhanced in POWER_CL
Blue Background	Uses Bank-2

Plugging the POWER_CL module.

Being a bank-switched module, there are two 4k-blocks involved: both banks need to be loaded using the "-16K" control string in the **PLUGxx** function input in ALPHA, Whether you use a copy in sRAM or it's already burned in FLASH, make sure that both banks occupy contiguous blocks, or the **PLUGxx** command will not work properly.

For example, assume the banks are copied into locations 0x840 and 0x841 on the V3 CL board (any other pair of contiguous addresses will also work, even in V2 systems). Then this will be the syntax required to plug it in page #9:

"840-16K" in ALPHA
PLUG1U

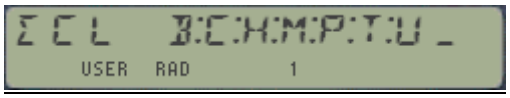
You can also use **PLUGH** or **PLUGP** in V3 systems, (obviously without Printer or HP-IL) as the module is designed to be compatible with those locations. The functions listed above with a blue background all use bank-2, thus can be used as an effective way to verify the proper installation.

WARNING: Be aware of the dependency with the Library#4, which requires it to also be configured for the proper operation of the functions. Failure to do this will cause surely unexpected and likely unpleasant results. Refer to the Library#4 documentation for details.

WARNING: POWER_CL is designed to work paired with the XROM #15 version of YFNZ.

2. The functions in detail.

The following sections of this document describe the usage and utilization of the functions included in the CL-UTILS module. While some are very intuitive to use, others require a little elaboration as to their input parameters or control options, which should be covered here.

2.1.- LAUNCHERS	

2.1 Function Launchers.

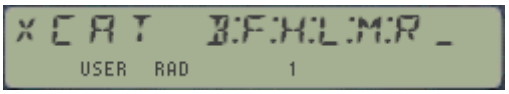
The table below lists the launchers by function groups:

Index	Function	Warnings	Description
1	**BAUD**	None	Calls BAUD12, BAUD24, BAUD48, or BAUD96
2	**MMU**	None	Calls MMUDIS, MMUEN, or MMU?
3	**TURBO**	None	Calls the corresponding TURBO*xx* function
4	**PLUG**	**Light**	Prompts for port location. Enter L/U first (when needed)
5	**UPLUG**	**Light**	Prompts for port location. Enter L/U first (when needed)
6	**HEPX**	**Medium**	Configuration loading for HEPAX set-ups
7	**XCAT**	None	Extended CATalogs: Buffers, Blocks, CLLIB, etc.
8	**ΣCL**	None	Launcher of Launchers -> invokes any of the seven above

Once you assign **ΣCL** to any key, that alone will give you access to more than 56 functions (!) from that single key – an effective way to make it compatible with other existing key-assignments, saving memory (KA registers) and time. So go ahead and get comfortable with that arrangement as your standard baseline.

Prompting functions use a technique called partial key entry, dividing the data entry in two (or more) parts. The keyboard is also re-defined, in that just those keys corresponding to the appropriate options are active. The cues in the prompt will offer you indication of which keys are active on the keyboard, and typically are intuitive enough to figure out in each case.

See below a few examples of the prompts, where the choices are to a large extent self-explanatory:

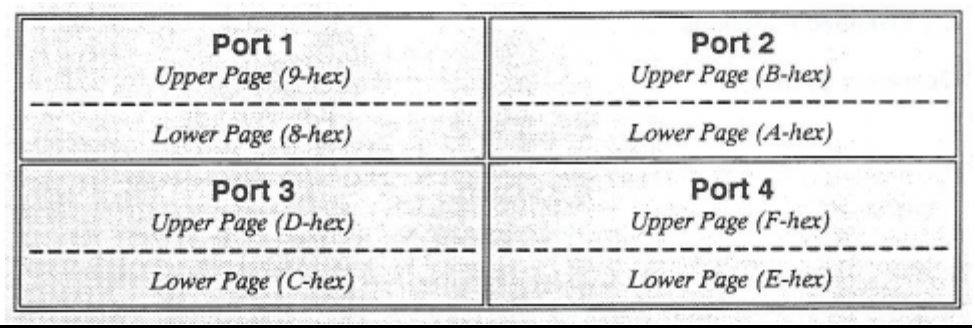


The TURBO options are as follows: none, 2x, 5x, 10x, 20x, 50x, and ?. Use “0” for 20x, Radix for “50” - as 2 and 5 are already taken for 2x and 5x speeds.

In general all launchers behave in a similar manner.

- The Back Arrow key will either cancel out entirely or remove partial entries;
- Non-active keys will blink the display and maintain the prompt
- Holding down the last key briefly shows the invoked function name – visual feedback.
- This will be followed by NULL if kept depressed long enough – last chance to bail out.
- Launchers are not programmable per-se – but:
- *They can be used in PRGM mode* to enter the called-upon function as a program line.

The **PLUG** and **UPLUG** launchers don't offer any cues in the prompt – and therefore deserve special consideration. The picture below shows the convention for the external pages of the 41:



Valid entries for the prompt are:

[SHIFT**],** toggles between **PLUG** and **UPLUG** launchers
[L], to flag a LOWER half-port condition, followed by the port number
[U], to flag an UPPER half-port condition, followed by the port number
[1], for port 1 – comprising pages 8 and 9
[2], for port 2 – comprising pages A and B
[3], for port 3 – comprising pages C and C

For the (**U**)**PLUG** cases the prompt completes either when the number 1-4 or the letter {A,G,P,H} is entered, and the corresponding function is launched.

For half-port (or 4k) modules *use the **L/U** keys first* in the (un)plugging prompts, then the port number. These keys act as toggles when pressed sequentially, replacing each other in the display upon repeat usage. Also during these events pressing BackArrow removes the half-port condition and returns to the main prompt.

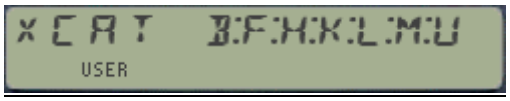
Remember that plugging a module into the "wrong" port location can create minor issues (or major havoc) if you're overwriting some/part of the machine's configuration. A good example is overwriting YFNS itself, or a HEPAX RAM block. Always make sure the destination is safe – using **BLCAT**, the standard CAT2 or better yet the CCD CAT'2.

Also valid entries are :

[H], for page #7 – the HP-IL reserved page
[P], for page #6 -- the Printer reserved page
[G], for page prompt 6-F – effectively calling the **PLUGG**(**?**) functions
[A], to invoke functions **PLUGG?** and **UPLUGA** – see later description.

Remarks.- Both **PLUG**/**UPLUG** offer all the 14 available choices in **YFNS**, including (**U**)**PLUGP** and (**U**)**PLUGH**. Exercise extra caution with those two locations, as they may be used by system extensions like Printer or HP-IL. Page #6 in particular has more strict demands on the ROM layout that makes it non-suitable for the majority of ROMS. *Also because pages #6 and #7 on V2 version of the CL don't support bank-switching, they unfortunately aren't a good place for the HEPAX ROM for V2 systems.*

Note: PLUGG will also allow "4" as valid input, which invokes the **PGG#4** function settings. More about this one will follow later.

2.2.- CATALOGS	

2.2. CATALOGS, CATALOGUES…

The additional CATalogs are accessed by **XCAT**, and include the following:

Index	Function	Warnings	Description
1	**BLCAT**	None	Borrowed from the HEPAX ROM – shows the 4k-blocks contents.
2	**BFCAT**	**Light**	Lists those elusive buffers present in the system.
3	**MMUCAT**	None	Lists the MMU mappings into each block.
4	**HEPDIR**	None	Calls the HEPAX' HEPDIR function
5	**KILB _**	**None**	Constants Library Selections
6	**CLLIB _**	**Light**	Cl ROM Image Library - alphabetical or by type.
7	**UCAT _**	None	Lists all available units for UMS conversion.

If you're like me you'll like to have good visibility into your machine's configuration. With its ROM Library and MMU settings the CL adds a few dimensions to the already rich 41CX system – and the goal is to have equivalent catalogue functions to review the status and options available.

Each CATalog has its own idiosyncrasies, but in general they feature single-step modes, and have "hot keys" to allow for specific actions – like deletion of buffers, navigation shortcuts, and direct plugging of ROMs into a port. This makes chores like searching for the correct syntax and plugging a module from the library a trivial task.

Some (**BLCAT**, **BFCAT, KLIB, UCAT**) are not strictly related to the CL, and will also work on a standard 41. Obviously **MMUCAT** is only meaningful for a CL machine, and will return zeroes if the CL board is not installed.

CATalog functions are notoriously complex and take up a significant amount of space – yet you'd hopefully agree with me that the usability enhancements they provide make them worthwhile the admission price.

2.2.1. Block CATALOG

BLCAT	**Block Catalog**	Author: VM Electronics	Source: HEPAX Module

Lists the first function of every non-empty ROM block (i.e. Page), starting with Page 3 in the 41 CX or Page 5 in the other models (C/CV). The listing will be printed if a printer is connected and user flag 15 is enabled.

- Non-empty pages will show the first function in the FAT, or *"NO FAT"* if such is the case
- Empty pages will show the *"NO ROM"* message next to their number.
- Blank RAM pages will also show *"NO FAT"*, indicating their RAM-in-ROM character.

No input values are necessary. This function doesn't have a "manual mode" (using **R/S**) but the displaying sequence will be halted while any key (other than **R/S** or **ON**) is being depressed, resuming its normal speed when it's released again.

2.2.2. Buffer CATALOG

BFCAT	**Buffer CATalog**	Hot keys: R/S, SST, SHIFT, D, H	
[D]	Deletes Buffer	*In manual mode*	
[H]	Decodes Header register	*In manual mode*	

This function is very close to my heart, both because it was a bear to put together and because the final result is very useful and informative. It doesn't require any input parameter, and runs sequentially through all buffers present in the calculator, providing information with buffer id# and its size.

41 buffers are an elusive construct that is mainly used for I/O purposes. Some modules reserve a memory area right above the KA registers for their own use, not part of the data registers or program memory either. The OS will recognize those buffers and allow them to exist and be managed by the "owner" module – which is responsible to claim for it every time the calculator is switched on.

A good example is the Time module, which uses it to store the alarms data.

Each buffer has an id# number, ranging from 1 to 14. Only one buffer with a given id# can exist, thus the maximum number present at a given time is 14 buffers – assuming such hoarding modules would exit – which thankfully they don't.

The table below lists the well-known buffers that are possibly to be found on the system:

Buffer id#	Module/Eprom	Reason
1	David Assembler	MCODE Labels already existing
2	David Assembler	MCODE Labels referred to
3	Eramco RSU-1B	ASCII file pointers
4	Eramco RSU-1A	Data File Pointers
5	CCD Module, Advantage	Seed, Word Size, Matrix Name
6	Extended IL (Skwid)	Accessory ID of current device
7	Extended IL (Skwid)	Print Cols, number & width
8	Complex Stack	Ángel Martin's 41Z ROM
10	Time Module	Alarms information
11	Plotter Module	Data and barcode parameters
12	IL Development, CMT-200	IL buffer and monitoring
13	CMT-300	Status Info
14	Advantage	INTEG & SOLVE scratch
15*	Mainframe	Key Assignments
*) KA area isn't really a buffer.		

For instance, plug the AOSX module into any available port. Then type **PI**, **SEED**, followed by **BFCAT** to see that a 2-register buffer now exists in the 41 I/O area – created by the **SEED** function.

id#=5, buffer at address 194, size=2, properly allocated.

Suppose you also change the default word size to 12 bits, by typing: 12, **WSIZE**. This has the effect of increasing the buffer size in one more register, thus repeating **BFCAT** will show:

id#=5, buffer at address 194, size=3, properly allocated.

Say now that you also plug the 41Z module into a full port of your CL. Just doing that won't create the buffer, but switching the calculator OFF and ON will – or alternatively execute the **-HP 41Z** function. After doing that execute **BFCAT** again, then immediately hit **R/S** to stop the listing of the buffers and move your way up and down the list using **SST** and **BST**. You should also see the line for the 41Z buffer, as follows:

id#=8, buffer at address 197, size=12, properly allocated.

If the module is not present during the CALC_ON event (that's to say it won't re-brand the buffer id#) the 41 OS will mark the buffer space as "reclaimable", which will occur at the moment that PACKING or PACK is performed. So it's possible to have temporary "orphan" buffers, which will show a question mark next to the id# in the display. This is a rather strange occurrence, so most likely won't be shown – but it's there just in case.

BFCAT has a few hot keys to perform the following actions in manual mode:

1. **[R/S]** stops the automated listing and toggles with the manual mode upon repeat pressings.
2. **[D]** – for instant buffer deletion – there's *no way back, so handle with care!*
3. **[H]** - to decode the buffer header register. Its structure contains the buffer ID#, as well as some other relevant information in the specific fields - all buffer dependent.
4. **[**SHIFT**]** to flag the listing to go backwards – both in manual and auto modes.
5. **[SST]** and [**BST**] to move the listing in manual mode, until the end (or beginning) is reached
6. **[<-]** Back Arrow to cancel out the process and return to the OS.

Like it's the case with the standard Catalogues, the buffer listing in Auto mode will terminate automatically when the last buffer (or first if running backwards) has been shown. In manual mode the last/first entry will remain shown until you press BackArrow or R/S.

Should no buffers are present, the message *"NO BUFFERS"* will be shown and the catalog will terminate. *Note also that the catalogue will be printed if in NORM/TRACE mode*, producing a record of all buffers present in the syste

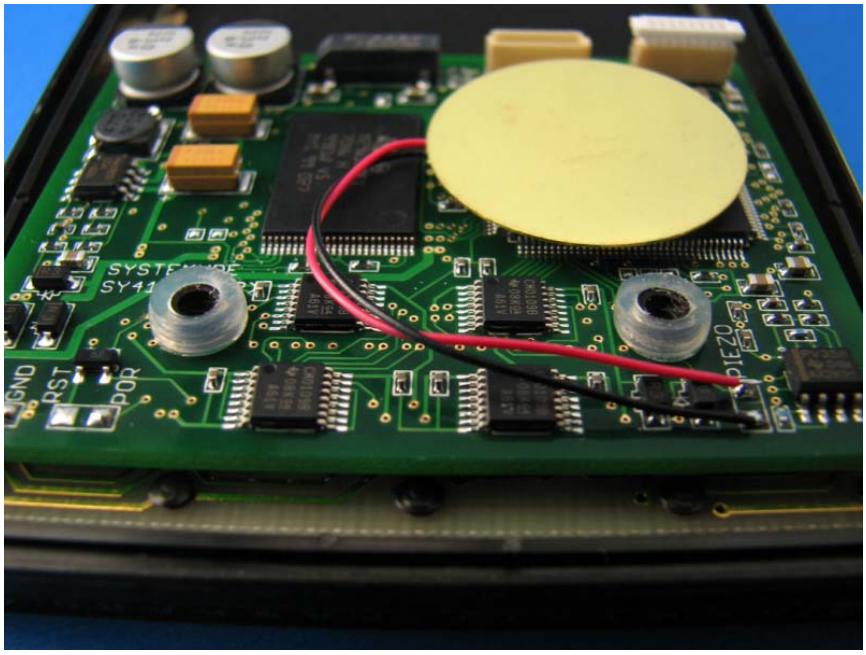


Photo courtesy of Geoff Quickfall.

2.2.3. Interrogating the MMU.

MMUCAT	**MMU CATalogue**	No inputs	
ADRID	**Gives ROM id# from ADR**	Expects string in Alpha	
FYNZ?	**FYNS Location Finder**	No inputs	
PLUGG?	**ROM id# in page by X**	Prompts for page#	Valid inputs are 4, 6-F

MMUCAT is really a FOCAL program that drives the function **ADRID,** the real engine behind it – not to be confused with the capital city of a country I know quite well. **ADRID** is obviously programmable. The idea is simple: produce a list of the MMU mappings into the different pages, showing either the ROM id# or the address (Flash or SRAM) currently mapped to the port.

A loop is executed starting on page #3, and up until page #F. Each iteration retrieves (pokes more appropriately) the address written into the corresponding MMU register, then searches it against the internal ROM id# table written in bank-2 of the POWER_CL module. More about this later.

Note that full-port modules will return the ROM id# attached to the lower half, and the Flash address mapped to the upper half. sRAM MMU entries will return the corresponding sRAM address.

While similar to the CAT2 concept, this really has an MMU-oriented perspective of things, and thus is purely a 41 CL feature – it'll render all entries zero if used on a "regular" 41. The program listing is rather simple – as **ADRID** does all the weight lifting under the hood:

	01	LBL "MMUCAT"		20	LBL 00	
	02	"0-0000"		21	"8040"	*prefix*
	03	ASTO X		22	XTOA	
	04	52	*"4"*	23	ARCL Y	*page#*
	05	XEQ 00		24	**YPEEK**	*read MMU rg,*
	06	CLX		25	YSWAP-	*swap around "-"*
	07*	54,057	*"6" to "9"*	26	YCL-	*delete from "-"*
	08	LBL 02		27	YBSP	*back space*
	09	XEQ 00		28	ATOX	
	10	ISG X	*next*	29	RDN	
	11	GTO 02		30	ADRID	*decode address*
	12	CLX		31	XTOA	
	13*	65,07	*"A" to "F"*	32	"|-: "	
	14	LBL 01		33	-3	
	15	XEQ 00		34	AROT	
	16	ISG X	*next*	35	RDN	
	17	GTO 01		36	AVIEW	*show ID#*
	18	CLD		37	PSE	*pause*
	19*	RTN		38	END	

A related function is **YFNZ?**, which returns the page number the YFNS is currently plugged in. This can come very handy in your programs to avoid overwriting it with other modules – as we'll see in the HEPAX configuration routines.

Another related function is **PLUGG?** - It interrogates the MMU to find out which module is plugged into a given page – the input to the function placed in X. This is all **page**-driven, and not based on the *port* number. There is no restriction in the input to the page number, however the returned values for pages 0,1,2,3, and 5 don't quite have the same meaning.

PLUGG? Also uses **ADRID** to decode the string returned by **YPEEK –** which provides the MMU address mapping the corresponding page. In the **YFNZ?** case there's no need to look up in the ROM id# table since we know what we're looking for – just need to check all pages looking for that specific string.

2.2.4. A wealth of a Library – with two access modes.

ROMLIB _	**ROM Library by type**	Prompts for type: E:F:G:M;S:U;X	Has Hotkeys
CLLIB _	**CL Library by name**	Prompts for A-Z	Has Hotkeys
[P]	Invokes **PLUG _**		
[A]	Copies id# shown to Alpha		

One of the most notable features of the CL is its extensive ROM image library, allowing you to plug almost any conceivable module ever made (of which I have contributed a few) into your 41CL just by using one of the **PLUG**xx functions. The input syntax requires that the correct ROM ID string be placed in Alpha, and certainly there are a few of those to remember – and rather similar to each other since the string is only 4 characters long.

These two functions come to the rescue – by providing either an alphabetical or by type listing of all the module ID's so you can review them and –eventually – plug the ROM directly from the catalogue, for convenience sake.

ROMLIB prompts for a ROM type, whereas **CLLIB** prompts for an alphabetical section, A to Z. Pressing [**SHIFT**] at this point toggles between both modes, alphabetical or by type. Both catalogues can run in auto mode of can be stopped using R/S, and then the listing can proceed in manual mode using SST and BST as you can expect.

It is in manual mode where you can use the other shortcuts or "hot keys", as follows:

- **[ENTER^]** skips to the next section (or previous if running backwards)
- **[A]** will copy the id# shown to Alpha
- **[P]** will exit the catalog and invoke the **PLUG_** function launcher
- **[**SHIFT**]** changes the direction of the listing, backwards <-> forwards
- **[<-]** Back Arrow will cancel out the catalog.

The enumeration terminates in auto mode when the last ROM id# (or first one if running backwards) has been reached. Also keeping any key depressed in RUN mode will halt the sequence displaying until it's released again, so it's easier to keep tabs with the enumeration.

The same considerations made about plugging modules can be made here – be careful not to inadvertently overwrite anything you're using with a new ROM image (specially important for YFNS), as there's no check whether the target location is already used or not – with the only exception being the functions **PLUGG** and **PLUGGX**

As you can guess there is a lot of code sharing between **ADRID** and these two ROM library catalogue functions. Fundamentally they all use a ROM id# table within bank-2 of the POWER_CL ROM to look up for the string, and fetch the address in Flash of the corresponding image. This table is quite long, occupying more than 1k in the ROM – yet worth every byte.

The prompt in **ROMLIB** suggests the different ROM types as follows:

- **[E]** represents Engineering-related modules in all the main branches: EE, ME, etc.
- **[F]** represents Finance-related modules – you gotta love those too…
- **[G]** represents Games and entertainment related modules;
- **[M]** represents Math-related modules, the heart of the machine for some;
- **[S]** represents Applied Science-related modules, excluding Engineering;
- **[U]** represents the Utilities group, all those nice packs adding extra tools to the system;
- **[X]** represents the system-Extensions group, all what makes the 41 much more than just a calculator.

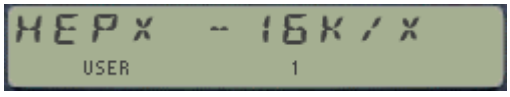
As you can see these groupings are somewhat loosely defined, yet it's simple and intuitive enough to have a good handle on the categories – which is the main objective for the searches. Also there's no distinction between HP-made or 3rd. Party modules in this scheme.

The "**A-Z**" prompt entry in **CLLIB** is a refinement of the same idea: it provides a handy shortcut to start your search in the appropriate section, so there's no need to review all the preceding ones – which can be very lengthy considering the sheer number of them, even if you used **ENTER^** to skip sections. The implementation is quite nice, even if it's the author who says it – have a look at the POWERCL_Blueprint if you're curious about the MCODE implementation details.

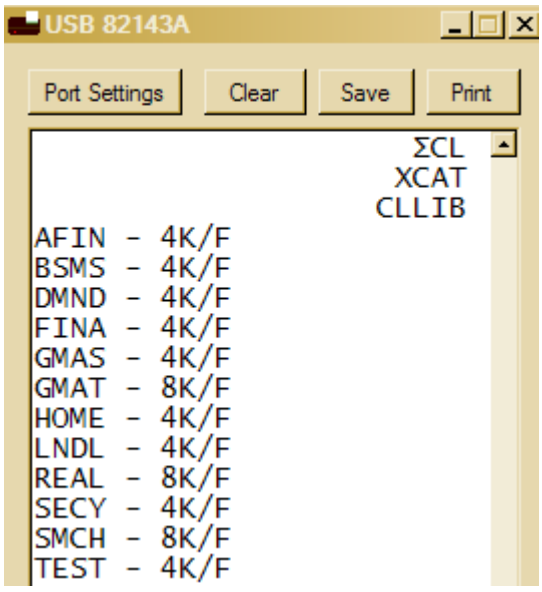
If the section doesn't have any ROM id# starting with such letter (which currently only occurs with the [W] letter) the message *"NO MATCH"* will be shown. Non-alphabetical keys are not valid entries for **CLLIB**, and will cause the display to just blink and maintain the prompt. Lastly, selecting [X] will list the general-purpose placeholders; refer to the CL manual for details on those.

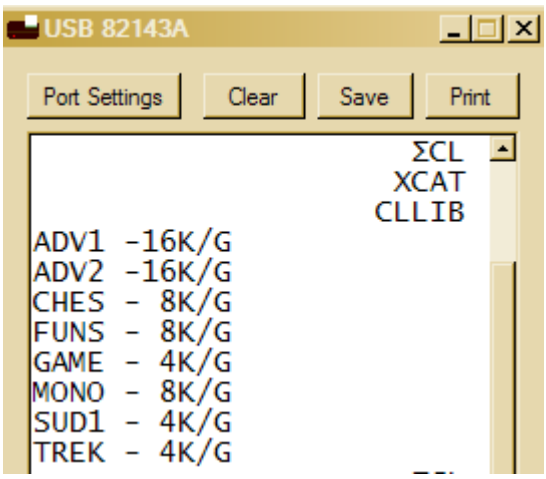
The enumeration output includes three pieces of information, as follows:

(1) The ROM ID name (entry used by the CL in the PLUGxx functions),
(2) Its size (either single or multi-page modules), and
(3) The type it belongs to according to the classification explained above.



Like it was the case for **BFCAT**, *these two functions fully support the printer TRACE/NORM modes*, generating a complete printout record of the ROM enumeration for your convenience. See appendixes for a complete listing of the module database, and the example below listing just the Finance and Games modules, taken as a direct cut & paste from the output using the USB-41 module:





At any given time the automatic enumeration can be stopped using [**R/S**], and continued manually in single-step mode using **SST**, either forwards or backwards controlled by the [SHIFT] key – yet the entry will only be printed going forwards, to eliminate redundancy.

Finally, the hotkeys **[A]** and **[P]** will copy just the ROM CL_ID text into Alpha, without the size/type details so it's ready to be used by any **PLUGxx** function.

2.3. HEPAX & SECURITY	

2.3.1. Configuring the HEPAX system.

HEPX	**HEPAX Fns. Launcher**	Prompts "4:8:6:I:D"	Accessible from **ΣCLF**
HEPINI	**Initializes File System**	Prompts for values	
HEPYX	**Initializes File System**	Takes inputs from Stack	Author: Howard Owen

Use these functions to initialize the HEPAX File System on the CL. This is needed on the CL because this feature is disabled in the HEPAX ROM image included in the CL Library, and therefore the addition here.

The function takes two parameters: **the number of HEPAX RAM pages to configure** and **the address of the first one**. These are expected to be in the Y and X registers respectively for **HEPYX**, whereas they'll be manually entered as prompts in **HEPINI**.

Note that even if the first prompt is a DECIMAL entry, the double quotes will remind you that the second one is in HEX, with valid inputs being 8,9, and A-F.

The procedure consists of writing a few bytes into strategic locations within each HRAM page so that the HEPAX will recognize them as being part of its File System. Those locations and byte values are shown in the table below:

Address	***Byte value***
x000	ROM id# => equal to the page#
xFE7	Previous HRAM page id# (zero if first)
xFE8	Next HRAM page id# (zero if last)
xFE9	Fixed value = 091
xFED	Fixed value = 090
xFEF	Fixed value = 091
xFF1	Fixed value = 0E5
xFF2	Fixed value = 00F
XFF3	Fixed value = 200 (or 100)

For this to work the target pages must be mapped to sRAM – or otherwise the byte values could obviously not be changed. So it is expected that the appropriate number of HRAM pages are configured, which is the subject of the functions describe in the next section.

The maximum number of HRAM pages accepted by the function is 9, but typical HEPAX configurations have TWO pages (Standard HEPAX, 8k) or FOUR (Advanced HEPAX, 16k). The ROM id#'s are assigned by *giving it the same value as the page number* – be aware that this may conflict with other ROMS currently plugged in your CL, notably **CLUTILS** uses ROM id# "C", and **YFNS** uses "F" so *those two pages will have to be their id# manually re-issued to avoid any issues (!).* Use HEXEDIT for that.

HEPINI and **HEPYX** won't disturb the actual contents of the HRAM FileSystem, so they can be used at any time provided that the entries used are compatible with the HRAM arrangement. It is also possible to use them to configure only a subset of the available HRAM, as long as such subset uses the lower pages. An later example will clarify this.

HPX4	**4k RAM HEPAX Setup**	RAM page F, ROM page E	
HPX8	**8k RAM HEPAX Setup**	RAM pages E-F, ROM page D	
HPX16	**16k RAM HEPAX Setup**	RAM pages C-F, ROM page B	

These three functions will prepare the CL ports to hold a properly configured HEPAX file system, starting from the scratch. The process can be divided into four distinct parts:

1. First copying the HEPAX RAM template from Flash into the appropriate number of SRAM blocks, as many times as needed.
2. Followed by mapping those SRAM blocks to the 41 ports, and
3. Then configuring them using **HEPYX or HEPINI -** so that they are enabled for the HEPAX ROM to use.
4. Besides that, the functions will also map the HEPX ROM image to the page preceding the first HRAM block, as shown in the table above.

So even if they don't require any input parameter you must be fully aware that the previous MMU mapping to those ports will be overwritten. ***The exception being the YFNS ROM itself*** – as the programs will check whether it is currently mapped to the page being copied – and abort if that's the case. A nice built-in protection that avoids getting in trouble.

Obviously these functions are not to be used frequently, ***since each execution will wipe off the contents of the HRAM pages***, overwritten with blank FLASH templates (!). Therefore the "medium" warning sign, proceed with caution.

See the appendix 2 for a listing of the FOCAL programs that implement this functionality.

Example:- Say you have used **HPX16** described above to configure 16k of HRAM, that is pages C to F contain copies of the HEPAX RAM template. You may want to use some pages for the FileSys, and others to hold other ROM images, and this in a dynamic way.

Then the following options are available to configure the HEPAX FileSystem:

N	PG#	Result	Comment
1	C	Page C	*Can extend upwards to {C,D}, {C,D,E}, or {C,D,E,F}*
1	D	Page D	*Can extend upwards to {D,E}; or {D,E,F}*
1	E	Page E	*Can extend upwards to {E,F}*
1	F	Page F	
2	C	Pages C,D	*Can extend upwards to {C,D,E}; or {C,D,E,F}*
2	D	Pages D,E	*Can extend upwards to {D,E,F}*
2	E	Pages E,F	
3	C	Pages C,D,E	*Can extend upwards to {C,D,E,F}*
3	D	Pages D,E,F	
4	C	Pages C,D,E,F	

Notice that the configurations can always be extended to include other pages located at upper addresses, but not the other way around. This is because the HEPAX code searches for the blocks sequentially to determine whether they belong to its FileSystem, starting at page 8. So once they are configured, changing the location of the first page to a lowered-number block will create a conflict.

Note: Using the prompt [**D]** in the **HEPX** function launcher will invoke the **HEPDIR** function from the HEPAX ROM – which obviously is expected to be mapped to the MMU (or physically plugged to the CL) for this to work.

2.3.2 Page-Plug functions.

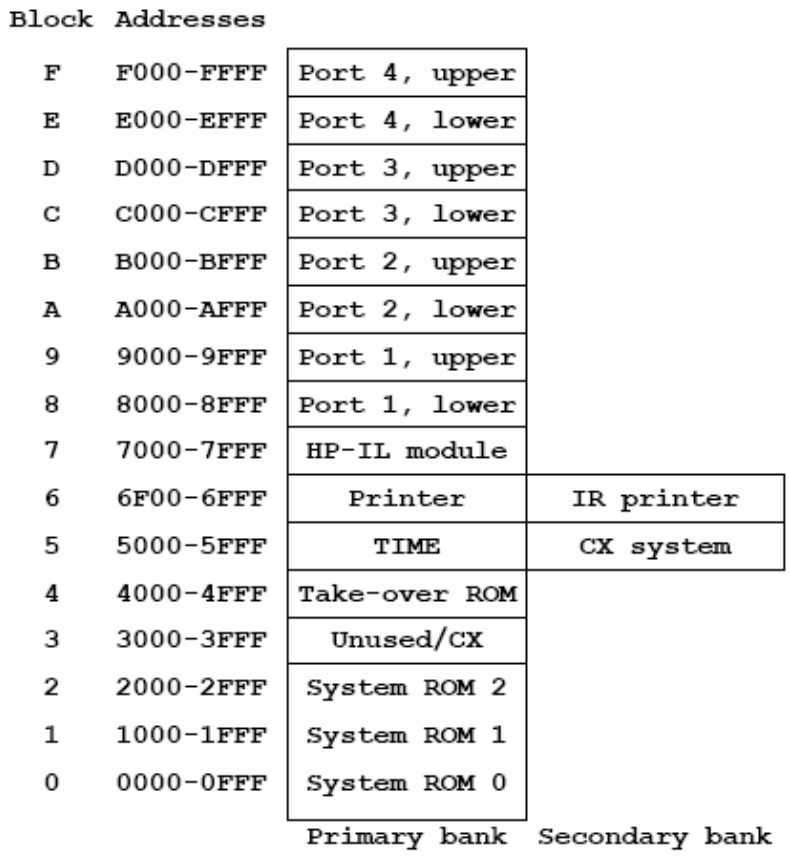
PLUGGX	**PLUG Page by X**	Page# in X	4k ROMS only
PLUGG _	**PLUG page by prompt**	Prompts for page: **"6-F"**	4k ROMS only
PLGG#4	**PLUG page #4**	Prompts for ROM: **"F:L:S"**	Take-Over ROMS
PLUGG? _	**Get plugged ID#**	Prompts for page: **"6-F"**	4k ROMS only
UPLUGA	**Unplugs ROM by Alpha**	ROM Signature in ALPHA	

As mentioned before, the HEPAX configuration functions also take care of plugging the HEPAX ROM into the appropriate page. This is accomplished by a single function, using a parameter to define the page address. This function is **PLUGGX**, or *"Plug Page by X*" (and its prompting *doppelgänger* **PLUGG**).

As before, pressing [SHIFT] will toggle between **PLUGG** and **UPLGG** – all interconnected for quick changes and convenience sake.

Contrary to the port-related convention of the "native" CL functions we're now referring to a page-related one, whereby the arguments of the function are the ROM id# in Alpha (same as usual) and the page# in X – removing the hard-coded dependency of the location used by the **PLUGLxx** and **PLUGUxx** functions.

The picture below (taken from the HEPAX manual) provides the relationship between ports and pages, also showing the physical addresses in the bus and those reserved for special uses (like OS, Timer, Printer, HP-IL, etc). Note that some pages (also called 4k-blocks or simply "blocks") are bank-switched. As always, a picture is worth 1,024 words:



Note that **PLUGG** and **PLUGG?** are mutually complementary functions, as they both operate on page id# and will take or return the corresponding ROM id# from/to Alpha. You could use **PLUGG?** to interrogate the MMU about page#4, and *you can use **PLUGG** to plug take-over ROMS to page#4* – by directly invoking the dedicated function **PLGG#4**, which will be covered in section 2.4 of the manual later on.

The following error conditions can occur:

- Because of dealing with *pages* and not full ports, **PLUGG_** and **PLUGGX** will only accept 4k ROMS, or otherwise *"DATA ERROR"* will occur.

- Main valid page# inputs are within to the 6-F range. Letters other than A-F will be inactive during the prompt, but all numeric keys will be allowed - yet values less than 6 will also be rejected, resulting in a *"DATA ERROR"*.

- Also a valid special input is "4" – but it requires the string *"OK"* placed in ALPHA to accept it. Any other value will trigger a *"DATA ERROR"* message.

- Attempting to plug a ROM to the page currently used by YFNS will also trigger an error code, to prevent accidental overwrite. The display will show *"PAGE=YF"* and no action will occur. If you want to relocate it you need to use one of the "standard" **PLUG** functions instead.

- If the string in Alpha is not a valid ROM id# you'll get *"BAD ID"* – as expected.

- If the YFNS ROM is not present (not mapped to the MMU or running on a standard 41 without the CL board) you'll get *"NONEXISTENT"* error.

Using the Page Signatures for Unplugging.

Function **UPLUGA** introduces a new aspect to the ROM plug/unplug chapter in that it uses the text ALPHA as input, and not page or port information – which may not be know to the user at the moment of the unplugging action.

The string in Alpha is expected to be the **Page Signature**, a four-letter string written at the end of each 4k-page comprising the module, just before the checksum word at the very bottom of the page. Notice that despite being a similar concept, the page signature is ***not*** the same as the ROM ID used by the CL's **PLUGxx** functions – and listed by **ROMLIB** or **CLLIB**.

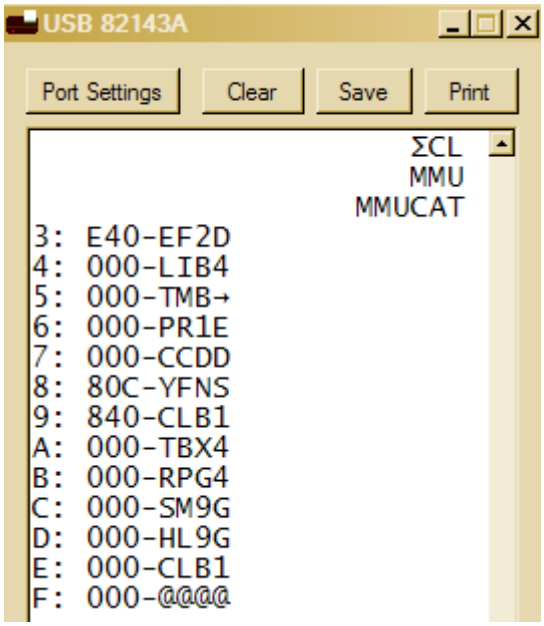
Typically the page signature is not known to the normal user, but power-users are of course a different kind. There is a couple of ways to find out the signature for any page:

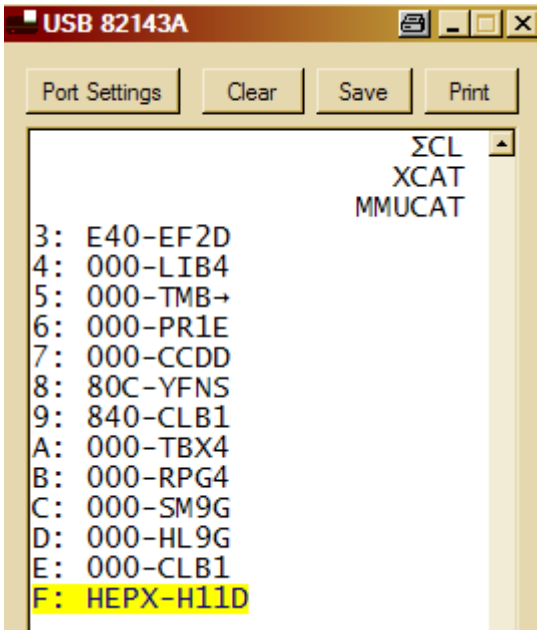
1. Using the function **PGSIG;** which prompts for the page number in decimal format (from 00 to 15), returning the signature string to ALPHA and the display. Note that it's also programmable, and than *in PRGM mode the page# is taken from the X register instead.*

2. Using **MMUCAT;** now also adding the signature to the information shown in the display. This enhances the output and provides more details as to the type of mapping for each page, which may be for a module in Flash (showing its ROM CL_ID and the page Signature), or a module in sRAM (showing the address and the Signature), or not part of the MMU (showing zeros and signature), or even a blank page with nothing plugged in – virtually or physically - (showing zeros and the "@@@@" string). All in all, a better characterization of the system configuration, *which will also be printed if the NORM/TRACE mode is set.*

At this point an example should clarify. The screenshot below shows the output of **MMUCAT** for a system with the USB-41 module, and a couple of MMU-plugged modules. Note that the Page#4 Library is included in the USB-41 part, and thus the MMU address is zero - as it corresponds to any "real" module (i.e. not virtual).

Note also that in pages #8 and #9 the sRAM addresses are listed (as opposed to CL_ID' s) since I'm not using the Flash versions of the CLLIB or YFNS modules. Lastly, note as well the "all empty" nature of page #F in the left-side picture.





Plugging the HEPAX module from Flash to page #F will change the last line as seen on the right-side picture, showing both the CL_ID and the page signature side-by-side.

© Photo by Geoff Quickfall, 2011

2.3.3 Security functions.

The following group of functions are a small detour, in that they aren't directly related to the CL but they come to full fruition when used on this platform.

SECURE	**Activate Security**	Author: Nick Harmer	Source: Data File
UNLOCK	**Deactivate Security**	Author: Angel Martin	
XPASS	**Change Password**	Author: Nick Harmer	Source: Data Fie

Here we have a nice practical application of advanced system control. Use these functions to manage a password-protection scheme for your CL – so nobody without authorized access can use it.

They were published in Data File back in 198x by Nick Harmer, and implemented in Q-RAM devices (a.k.a MLDL). Obvious caveat there was that removing the MLDL from the machine dismantled the whole scheme – but the CL has made it possible as integral part of the core system now.

The protection works as follows:-

1. Function **SECURE** activates the security by setting the protection flag. The execution also switches off the machine. This sets up a process executed on each CALC_ON event, causing to prompt the user for the password during the start-up process.
2. Function **UNLOCK** deactivates the security by clearing the protection flag.
3. Function **XPASS** allows the user to change the password from the default one to his/her favorite one. The length of the password is limited to six (6) characters.

Enter code (up to 6 chrs. long) and end with **R/S**

Inputting the password is very simple but very unforgiving as well: at the prompt *"PASSWORD=?"* just type the letters one by one until completing the word, and you're done. If you make a mistake the machine will switch itself off and it'll be "groundhog day" all over gain – until you get it right.

Each keystroke will be acknowledged by a short tone, but no change to the display – so nothing like "*****" as you type the word. If the wrong letter is entered a lower-pitch sound will be heard and the calculator will go to sleep.

Be especially careful when entering a new password code – as there is no repeat input to confirm the entry, so whatever key combination you type will be taken when ending the sequence with R/S. The initial password ("factory default", so to speak) is "CACA".

Enter code (up to 6 chrs. long) and end with **R/S**

Here again it comes without saying that this will only work when the CL-UTILS module is mapped to a SRAM block in the MMU – or otherwise none of the ROM writing will work.

Note: this is how you'd get yourself out of trouble if somehow you forgot the right code: do a memory lost to disable the MMU, then reload the CLUTILS from flash – which has the protection flag cleared. Map it to the right page and enable the MMU again – you're back in charge.

2.4.1. Using Page#4

As mentioned previously page#4 is a special case that requires its own dedicated (un)plugging functions, not covered by **PLUGGX** or the native (**U**)**PLUG** ones either.

PPG#4	**Plugs ROM in page#4**	Prompts F:L:S	**WARNING**
UPGG4	**Unplugs ROM from p4**		

The 41 OS reserves Page #4 as a special location. There are frequent checks done during strategic moments to specific locations that can be used to take control on the system, even over the OS itself if that was required – as it happens with the diagnostics executed from the different SERVICE ROMS.

Because of that, only "take-over" ROMS can be plugged in page#4. They have been written specifically for it and will either take complete control of the system (like the FORTH Module), or drive it from their own directive (like the LAITRAM Module).

Function **PPG#4** prompts for the ROM to plug into the page, options being just those three mentioned above: FORTH, LAITRAM, or SERVICE modules – by their initials: "**F:L:S**". Once the selection is made the function transfer execution to a hidden FOCAL program that writes the appropriate entries into the MMU registers, so that the mapping is correct. Refer to the CL manual for details on this.

WARNING: Be aware that once the order is complete you'll be at the mercy of the plugged module. Going back to the "normal" OS may not be as simple as you think, specially with the Service ROM plugged – which requires removing the batteries, then clearing the MMU entry with the MMU disabled after you switch it back on.

For the other instances it is possible to "exit" back to the OS, and thus you could execute **UPPGG4** to unplug the module from the page. Obviously no inputs are needed in this case.

Note that because of their titles being not directly key-able using **XEQ** (an intentional measure) you'll have to use another approach to invoke them. It's a trivial task with the CCD-style CAT'2, either during the catalog run or through a previous assignment to any USER key. Of course as a CL owner you're only one **YPOKE** away from a permanent solution if CLUTILS resides in sRAM ☺.

And what about the Library#4?

This is an important point, so make sure you have the Library#4 configured always in order to use the POWER_CL module. This configuration must be done manually, *as the routines used by many functions in the module are expected to be already present* for proper execution. Failure to do so will create unexpected and unpleasant events!

WARNING: Be aware that just plugging/unplugging the Library#4 using any of the **(U)PLUGxx** functions (once you have it burned in Flash or copied in sRAM) will not check for its presence, and therefore it's strongly recommended to power-cycle the machine in order to perform the Library#4 existence check. As always, refer to the Library#4 documentation for additional details.

2.4.2. RAM and ROM Editors

Placing the ROM image library on bank 2 freed up a large amount of space for additional functions to be included in the POWER-CL module, as can be seen by looking at its full-FAT list. The choice of functions added over previous CL_UTILS incarnations was clearly meant to have a comprehensive and self-contained function set that included some the best examples ever written for the HP-41 system. RAM and ROM editors are no doubt amongst these, and as such are available in the POWER-CL.

RAMED	**RAM Editor**	Uses GETKEY [KEYFNC]	**WARNING**
ROMED	**ROM Editor**	Uses Partial Data Entry	**WARNING**

Editing RAM memory with RAMED.

Written by Håkan Thörngren, this powerful RAM editor rivals with (and exceeds it in several aspects) the ZENROM implementation. It was first published in PPCJ V13 N4 p26.-, you're encouraged to check his original contribution for a complete description of the functionality and usage.

The starting address is taken from the X register in RUN mode (as decimal value between 0 and 999), or from the program pointer in PRGM mode. The display shows two distinct fields, with the nibble & byte section shown on the left side and the actual register content shown on the right – as a 7-digit scrollable field controlled by the USER and PRGM keys – very much like the CX's ASCII file editor **ED**.

Nibble D (the 13th within the register) is selected upon start-up, with the cursor centered in the middle of the field and its value blinking on the display. At this point you can use the control characters to move between both areas and within the fields, or the digit keys plus A-F to input the nibble HEX values being edited. Scrolling includes a tone to signal the wrap-around condition within the register, as the nibble being edited is updated in the address field on the left. A real tour-de-force and a masterful implementation without any doubt.

The screens below show a couple of examples, editing the leftmost nibble of the Y register (address: D002) and the rightmost digit of the X register (address 0003). The screenshots don't capture its magic, you really need to use it to appreciate its simple and powerful functionality.

The control keys for RAMED are as follows:

[**USER**]: moves down to the previous nibble or position within the field
[**PRGM**]: moves up to the next nibble or position within the field
[**+**]: moves up to the next register
[**-**]: moved down to the previous register
[**.**]: the Radix key moves between both fields, used to change the register address
[**1**]-[**9**],[**A**]-[**F**] the nibble value being edited
[**<-**] back-arrow cancels out and exits the editing
[**ON**]: tiurns the calculator OFF

RAMED is completely located in bank-2, with only the function name and a small code snippet in the first bank to transfer the execution. I have only minimally altered the source code to take advantage of the CX- and Library#4 routines.

RAMED uses a more power-demanding technique that will have some drains on the battery life if used extensively. Do not leave it run idle for a prolonged time.

Editing ROM areas with ROMED.

Written by W. Doug Wilder, another MCODE grand-master - this ROM editor has all the basic functionality required for the most common needs; perhaps just a couple of notches below the tremendous HEPAX's **HEXEDIT** – but in a much more concise foot-print implementation and not exempt of wonders on its own.

The initial prompt requests the address to edit, ranging from 000 to FFFF as you would expect. Once entered, the display is identical to that one in **HEXEDIT**, with three distinct fields showing the address being edited, the current value, and three underscore characters where the new value will be written as the input progresses.

Usual rules of the game apply: the first character can only be 0,1,2,3; and obviously there must be a Q-RAM block for the input to be actually written in. A nice touch (lacking in **HEXEDIT**) is a *"ROM"* message shown when the destination is read-only.

The control keys for ROMEDIT are as follows:

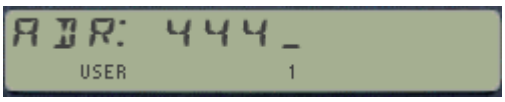
[**SST**] :	moves up one word
[**BST**], [**TAN**]	moves down one word
[1]-[**9],**[**A**]-[**F**]	the nibble value being edited
[**ENTER^**]:	inputs three zeroes as word value
[**<-**]:	first back-arrow prompts for a new address, second exits the editing
[**ON**]	turns the calculator OFF

ROMED is completely located in bank-1. Besides the changes *"of rigueur"* to use the Library#4 routines, I have made a couple of enhancements to the original implementation, adding the underscores for the edited field and the [**ENTER^**] control key for a closer resemblance with the HEPAX implementation in **HEXEDIT**.

ROMED can only edit the first bank, so the only missing functionality is perhaps this; no access to the other banks in bank-switched modules (like the HEPAX, Advantage, Timer, or POWER_CL itself). Certainly not a big deal in the 90% of the cases; and sure enough well worth the admission price.

[**Note:** The other missing functionality is the ability to do live-editing of polling points and other OS-controlled hot addresses, which the HEPAX manages by preventing the calculator from going into light sleep – definitely hardly used in the 99,9% of the cases.]

The screenshots below show editing of the Library#4 contents on-the-fly – a real godsend for MCODERS to quickly test small code changes without having to re-compile / rebuild the ROM images.



ROMED uses the partial-key entry technique, more gentle on the battery drain requirements – and incidentally also the reason why it cannot be located on bank-2, as a technical detail.

Final remark.- The original **ROMED** is available in the DISASM module, under the name **WROM,** and **RAMED** is also included in the RAMPAGE module, named **RAMEDIT** there**.** Thanks to its gigantic module image library there's certainly no shortage of modules to use on the CL, but having all these functions included in a lean 4k-ROM is very advantageous from the usability and compatibility standpoint, eliminating the need to alter the system configuration to access another module containing the desired function.

2.5. SYSTEM EXTENSIONS

2.5.1. Alpha and Display Utilities.

The following functions relate to Alpha string manipulation, as the main vehicle for many YFNS functions and are included in the CLUTILS for added convenience.

YINPT _	**Input Y-String**	Prompts for string	
YBSP	**Alpha Back Space**		Author: W&W GmbH
YCL-	**Alpha Delete from "-"**		Author: W&W GmbH
YCL>	**Alpha Delete from ">"**		
YSWAP-	**Swap around "-"**		
YSWAP>	**Swap around ">"**		
DTST	**Display Test**	Author: Chris L. Dennis	Source: PPCJ V18 N8 p14

The reason why characters **"-"** and **">"** are so relevant is the formatting required by many of the YFNZ functions, like **YPEEK, YPOKE**, **PLUG*xx*,** etc. To that effect the most useful function of this group is no doubt **YINPT**, which redefines the keyboard as a hex entry {0-9, A-F}, plus a few special control characters, as follows:

- [**J**] adds the control character ">" to the display
- [**Q**] adds the control character "-" to the display
- [**K**] adds the string "16K" to the display
- [**L**] adds the string "DBL" to the display
- [**M**] adds the string "RAM" to the display
- [**SST**] adds the string "MAX" to the display
- [**ENTER^**] adds three zeroes to the display
- [**<-**] BA removes the last character (or groups above), or cancel out if Empty
- [**R/S**] terminates the entry process, and copies the display content to ALPHA.

Using this function expedites the construction of the Alpha strings required by all other Y-Functions, make sure you have it assigned to a handy key as it's likely to be used quite frequently.

Notice that because it uses the display, the valid length accepted by **YINPT** *is limited to 11 characters.* This will suffice for all parameter input purposes within YFNS functions.

DTST Simultaneously lights up all LCD segments and indicators of the calculator display, preceded by all the comma characters (which BTW will be totally unnoticed if your CL is running at 50x Turbo!). Use it to check and diagnose whether your display is fully functional. No input parameters are required.

The remaining utilities from this group are pretty much self-explanatory, performing partial Alpha deletions and text swapping around the control characters **"-"** and **">"**. They are used by some of the FOCAL programs included in the module.

2.5.2. Other Utilities.

The following functions perform housekeeping tasks and are included in the POWERCL for added convenience. Some are a remake of the native YFNS with slightly improved behavior, while others just add up for a "rounder pack".

CHECKF	**Check system config**	No input	
ROMLST	**Shows XROM id#'s list**	No input	
HEXKB _	**HEX Entry**	1-9, and A-F	Author: Clifford Stern
DCD	**NNN to HEX string**	NNN in X	Author: W&W GmbH
PGSIG _ _	**Retrieves ROM signature**	Page# in prompt	
SIGPG?	**Finds page# if plugged**	Signature in ALPHA	
YFNZ?	**Location for YFNZ**	No Input	Author: Monte Dalrymple
XROM _ _	**Rom function Launcher**	Prompts for values	Author: Clifford Stern
ROMCAT _ _	**ROM Catalog**	Starts CAT'2 at ROM id#	Author: J.D. Dodin
SPEED	**CPU cycles per second**	No input	Author: Doug Wilderz
ASG _	**Multi-byte ASN**	Supports synthetics	Author: Frits Ferwerda
READPG _ _	**Reads page from HP-IL**	Page# and FileName	Author: R. del Tondo (?)
WRTPG _ _	**Writes page to HP-IL**	Page# and FileName	Author: R. del Tondo (?)
T>BS _ _	**Base Ten to Base**	Prompts for Base	Author: Ken Emery
XCF _ _	**Xtended CF _ _**	Allows ANY flag#	Author: Michael Katz
XSF _ _	**Xtended SF _ _**	Allows ANY flag#	Author: Michael Katz

Some brief comments pertaining to each function follow:

- **CHECKF** is a very useful routine to check for incompatibilities n the system configuration, as may occur when two ROMs with the same XROM id# are plugged. The function will scan all the ROM blocks looking for repeat values, showing a confirmation or a warning message depending on the case. It will also report the offending id# in case of conflicts, as many as there exist. Use it as frequently as you can, the best way to ensure that things are fine after plugging any of the many modules available on the CL library – a match made in heaven.

- **ROMLST** has somewhat of a similar purpose: it will produce a list in Alpha with the XROM id#'s of the plugged modules on the system, so you can check for dups. Because of the 24-char limit in the Alpha string, only the last 8 modules will be shown – sufficient in the majority of cases, specially considering that pages 3,4, and 5 are most likely unique because of being dedicated to the X-Functions, the Library#4, and the Time Module.

- **HEXKB** is the well-known **HEXENTRY** function published in Ken Emery's book "*MCODE for beginners*", and originally written by Clifford Stern. For all purposes it supercedes **CODE** (or **CDE**), which is available in the AMC_OS/X module anyway.

- **DCD** is the classic NNN to Hex utility, also used as subroutine throughout the module and thus made available to the user as individual functions as well.

- **PGSIG** was covered already. It'll retrieve the signature of the ROM plugged in the page entered at the prompt – or in the X register if used in a program. If no ROM is plugged it'll return four "@" characters.

- **SIGPG?** Is the inverse function: it'll find the page# where the ROM with signature input in ALPHA is plugged – returning *"NO MATCH"* if it's not found.

- **YFNZ?** is completely equivalent to **YFNS?**, in fact is just a code stub that invokes the latter. It must be in the CLUTILS module for subroutine purposes. Incidentally, this is how **PLUGGX** checks for YFNS being currently mapped to the target page, and discards the request if so.

[Note: there's an ulterior motive for having this function included in the POWERCL module, and such is as a back-door recovery method from a "lost YFNS" contingency. This may occur when the YFNS is overwritten with another module image by the MMU, typically as result of inadvertently using **PLUGxx** on the same location. That being the case, **YFNZ?** will disable the MMU so you can take charge again and rebuild the MMU to correct the issue - with no need to resort to ML or removing the batteries anymore]

- **XROM** is a well-known function to directly call any function within a plug-in ROM, knowing its ROM id# and function#. Written by Clifford Stern in the heydays of the 41 systems, with a real inside knowledge of the internal OS routines. Both prompt inputs are to be entered as DECIMAL values.

- **ROMCAT** was written by Jean-Daniel Dodin, well-known MCODE pioneer and old HP-41 hand. It prompts for the XROM id# and starts CAT'2 from the position used by such ROM (if present). The usual conventions apply, whereby only the ROM headers are listed unless the catalog is stopped and ENTER^ is pressed in manual mode.

- **SPEED** is a curious gem, although I'm not completely sure I managed to transcribe it well. It's supposed to return the number of CPU cycles per second, so I thought it'd be ideal for the CL given the different TURBO modes. Alas, it always returns the same value (1,126.316), irrespective of the TURBO setting. This is about 6 times bigger than the normal HP-41 result, (167,333) for what is worth. We have Doug Wilder to thank (again) for writing it, using the Time module to keep pace with things.

- **ASG** is another example of first-class MCODE programming: imagine being able to directly input multi-byte functions (with synthetics support) into the ASN prompt, so to assign "LBL IND e", or "RCL M" to a key - not using key codes or byte tables? Well no need to imagine it, just use ASG instead. **ASG** is taken from the MLROM, and it resides completely in the Page#4 Library, with only the FAT entry in the POWERCL FAT calling it. You're encouraged to refer to the MLROM documentation for further details.

- **READPG** and **WRTPG** are the mandatory read/write entire blocks (a.k.a. pages) to the HP-IL disk drive. Very much equivalent to the HEPAX' **READROM** and **WRTROM**, although here the destination page is provided in the prompt. Their code is entirely contained in the Library#4, so this is another example of the "free-riders" only needing the FAT entry and the calling stub footprint. They are taken from the CCD OS/X, thus I attributed authorship to R. del Tondo – which to this date is unconfirmed.

- **T>BS** is the always-useful universal base converter. Originally written by Ken Emery and published in "*MCODE for beginners*", I added a few usability enhancements and turned it into a prompting function. The decimal value is expected in X, and the base is entered in the prompt (or in Y in program mode). Also added code to place the result in ALPHA as well as in the display so it can be used by other functions.

- **XCF** and **XSF** are natural extensions to the mainframe standard Clear/Set Flag functions. Like those, they will prompt for the flag number - to be input as a decimal, to-digit entry. Unlike them, they'll accept anyone of the 56 flags from 0 to 55. When used in a program just ignore the prompt (won't be taken as program line) and place the flag value in X instead – in the preceding program line.

2.5.3. Unit Management System.

Bank-switched modules are a fantastic invention, but there are a few system design limitations that cannot be overcome, not even using this advanced technique. Such is the 64-entries limit in the FAT, one of the absolute barriers (or boundary conditions if you prefer) that exerts its controlling power regardless of the number of shadowed-banks we care to set-up.

It is because of this that bank switching lends itself very well to large bodies of code with few FAT entries, as apposed to many functions of reduced size. The best example being the HEPAX module, where dedicated banks are arranged for gigantic-code functions such as **HEXEDIT** and **DISASM**.

In the POWERCL case the triggering point was the ROM table holding the image library information, but even being large enough (over 1k) it doesn't fill up the second bank completely. The idea of adding the Unit Management System came as a natural to "fill the gap", because it too has the ideal attributes to be placed on the second bank: huge pieces of code, used sequentially and independently from the rest of the module, with tables and hard-coded values to be read.

The idea was very appealing; as it removes the need to have the Unit Conversion module loaded and saves one block for other uses – two big scores in one. It also provided the opportunity to write a Constants Library *a la* 33S or 35S, sorely missing even in the UMS module.

So chances are you'll never use it, but even then it's not adding any burden to your system configuration, or compromising the available resources. And if you use it, well I think you'll be pleased with the end result. Can anyone ask for more?

-SI	**Direct Unit Conversion**	FROM-TO in Alpha	Author: HP Co. / A. Martin
APPEND	**APPEND function**	Programmable	Author: Doug Wilder
KLIB	**Constant Library**	4 line-ups w/ 5 each	
SI-	**Inverse Unit Conversion**	TO-FROM in Alpha	Author: HP Co. / A. Martin
UCAT	**UNIT Catalog**	Prompts for Type	Co-author: P. Platzer
Y/N?	**Prompts for Y/N**	Only Y/N/RS/ON	Author: PANAME ROM

They have been slightly modified to comply with the bank-2 requirements, and **UCAT** is also enhanced with a new unit type prompt (**"G:F:T:W:E"** for Geometry, Force, Temp/Mass, Work, and Electrical units) and better internal structure.

Much more detailed documentation for these functions is available as independent documents – refer to the specific one as needed.

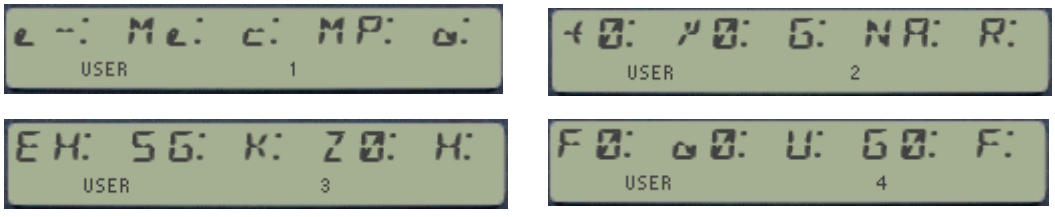


Figure 2.5.3 showing the four line-ups available in **KLIB,** to select the constant from the library. Note the flag annunciator indicating the one currently selected.

Farewell.

And with this you've reached the end of the POWERCL manual. – I hope these few pages have proven useful to you in your quest to become familiar with its capabilities and whet your appetite for even more to come.

The 41CL is an innovative realization nothing short of incredible, with amazing possibilities that open the door to yet new developments on the HP-41 platform; all this still happening 33+ years after the original 41 was launched. *Now that's what I call an achievement!*

Before and after: 33 years separate these boards:

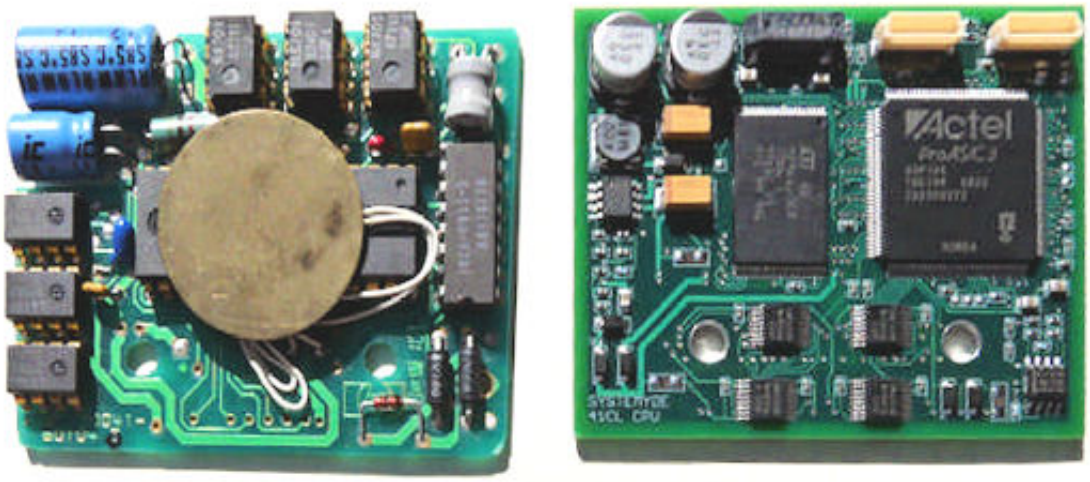


Photo: Steve Leibson, (c) May 2011

SYSTEMYDE INTERNATIONAL 41CL

Appendix 0. Summary of ΣCL Function launching functionality.

There are several function launchers in the CLUTILS. We've also seen that they show very flexible and interconnected combinations, sometimes "warping" around the complete module in a lot of ways. It's initially a bit confusing, yet you should not get intimidated by the multiple layers - as you'll get very quickly acquainted to the design and comfortable using it.

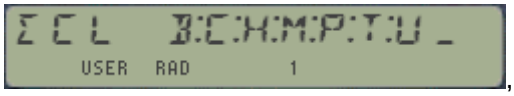
Notice that in some instances the same function can be accessed from two different places (i.e. HEPDIR, MMUCAT). Note also that they may be required to sequentially input the prompts from two (or more) launchers to finally get to the desired end result. –

For instance, say you want to plug the **AMC"OS/X** ROM into port 7 (the HP-IL one). Say you don't remember its CL ROMid#, nor the function name to accomplish that (which happens to be **PLUGH**). You decide to start at the very beginning, executing **ΣCL**, then chose ""**C**" for CATALOGS, then "**L**" for CLLIB, then "**A**" to start the enumeration from the letter A, then **R/S** at the "AOSX", followed by "**P**" to invoke the PLUG action. Then "**G**" to go to the page settings, then "**7**". *A breeze !*! ☺

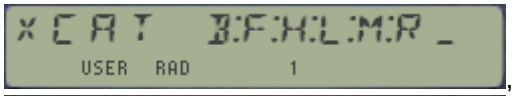
Putting it all together now.-

Granted this is one of the more complex examples (but not the worst one!) Here it is again with more details so it's easier to follow:- Let the display be our guide in this dialog with our trusty companion…

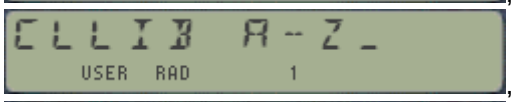
Start off by executing **ΣFL**, which will present:



, enter **"C"** for the Catalog Launcher [XCAT]



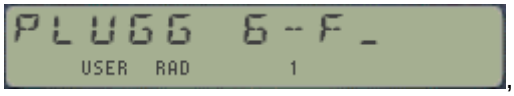
, enter "**L**" for the CL Library [CLLIB]



, input "**A**", **R/S**, **SST** to navigate until you see:

, enter **"P"** to trigger the PLUG functionality [PLUG]

, enter **"G"** to invoke the page-plugging [PLUGG]



, enter **"7"** - the page number to complete the job.

Easy does it – and you didn't have to remember any mnemonic or function name; nor does it require any key assignment other than **ΣCL -** to call up to 58 different functions. Does it get any better than this?

As a handy summary the table in next page summarizes all possible combinations to access a given function, using the different function launchers. Cells show the keys to press to access to the function, starting from the launcher on the column header.

BLACK fns. YFNS Functions
BLUE fns. POWERCL Functions

#	F. Name	ΣCL	BAUD	XCAT	HEPX	MMU	TURBO	PLUG	UPLUG
1	BAUD12	B,1	1						
2	BAUD24	B,2	2						
3	BAUD48	B,4	4						
4	BAUD96	B,9	9						
5	BFCAT	C,F		F					
6	BLCAT	C,B		B					
7	CLLIB _	C,L		L					
8	HEPDIR	H,D		H	D				
9	HEPINI _"_	H,I			I				
10	HPX16 _	H,6			6				
11	HPX4 _	H,4			4				
12	HPX8 _	H,8			8				
13	KLIB _	C,K		K					
14	MMU?	M,?				?			
15	MMUCAT	M,T		M		T			
16	MMUCLR	M,C				C			
17	MMUDIS	M,D				D			
18	MMUEN	M,E				E			
19	PLUG1	P,1		L,*,P,1				1	[ ],1
20	PLUG1L	P,L,1		L,*,P,L,1				L,1	[ ],L,1
21	PLUG1U	P,U,1		L,*,P,U,1				U,1	[ ],U,1
22	PLUG2	P,2		L,*,P,2				2	[ ],2
23	PLUG2L	P,L,2		L,*,P,L.2				L,2	[ ],L,2
24	PLUG2U	P,U,2		L,*,P,U,2				U,2	[ ],U,2
25	PLUG3	P,3		L,*,P,3				3	[ ],3
26	PLUG3L	P,L,3		L,*,P,L,3				L,3	[ ],L,3
27	PLUG3U	P,U,3		L,*,P,U,3				U,3	[ ],U,3
28	PLUG4	P,4		L,*,P,4				4	[ ],4
29	PLUG4L	P,L,4		L,*,P,L,4				L,4	[ ],L,4
30	PLUG4U	P,U,4		L,*,P,U,4				U,4	[ ],U,4
31	PLUGG _	P,G		L,*,P,G				G	[ ],G
32	PLUGG? _	P,A		L,*,P,A		G		A	[ ],A
33	PLUGH	P,H		L,*,P,H				H	[ ], H
34	PLUGP	P,P		L,*,P,P				P	[ ],P
35	PPG#4 _	P,G,4		L,*,P,G,4				G,4	[ ],G,4
36	ROMLIB _	C,L,[ ]		L,[ ]					
37	SERINI	B,I	I						
38	TURBO?	T,?					?		
39	TURBO10	T,1					1		
40	TURBO2	T,2					2		
41	TURBO20	T,0					0		
42	TURBO5	T,5					5		
43	TURBO50	T,";"					";"		
44	TURBOX	T,X					X		
45	UCAT _	C,U							
46	UPLGG _	U,G		L,*,P,[ ],G				[ ],1	G
47	UPLUG1	U,1						[ ],L,1	1
48	UPLUG1L	U,L,1						[ ],U,1	L,1
49	UPLUG1U	U,U,1						[ ],2	U,1
50	UPLUG2	U,2						[ ],L,2	2
51	UPLUG2L	U,L,2						[ ],U,2	L,2
52	UPLUG2U	U,U,2						[ ],3	U,2
53	UPLUG3	U,3						[ ],L,3	3
54	UPLUG3L	U,L,3						[ ],U,3	L,3
55	UPLUG3U	U,U,3						[ ],4	U,3
56	UPLUG4	U,4						[ ],L,4	4
57	UPLUG4L	U,L,4						[ ],U,4	L,4
58	UPLUG4U	U,U,4						[ ],G	U,4
59	UPLUGA	U,A						[ ],A	A
60	UPLUGH	U,H						[ ], H	H
61	UPLUGP	U,P						[ ],P	P
62	UPPG4	U,G,4						[ ],G,4	G,4

Appendix 1 – Detailed ROM id# table – in alphabetical order.

#	ID	Size	Name	Author / Compiler
1	A41P	12k	Advantage Pac	HP Co.
2	AADV	4k	Advantage Applications	J-F Garnier
3	ADV1	16k	Adventure_1	Angel Martin
4	ADV2	12k	Adventure_2	Angel Martin
5	AEC3	8K	AECROM 13-digit	Angel Martin
6	AECR	8k	AECROM	Red Shift
7	AFDE	8k	AFDC1	GunZen
8	AFDF	8k	AFDC2	GunZen
9	AFIN	4k	Auto Finance	GMAC
10	ALGG	8k	Algebra ROM	Angel Martin
11	ALGY	4k	Astro*ROM	Elgin Knowles & Senne
12	ALPH	4k	ALPHA ROM	A. Martin & D. Wilder
13	AOSX	4k	AMC OS/X	Angel Martin
14	ASM4	4k	Assembler4	??
15	ASMB	4k	Assembler3	??
16	ASTT	16k	ASTRO-2010 Module	Jean-Marc Baillard
17	AUTO	4k	Auto-Start / Dupl ROM	HP Co.
18	AV1Q	4k	AV1 ROM	Beechcraft
19	AVIA	4k	Aviation Pac	HP Co.
20	B52B	8k	B-52 ROM	Boeing
21	BCMW	4k	BCMW ROM	??
22	BESL	8k	Bessel ROM	A. Martin & JM Baillard
23	BLDR	8k	BLD ROM	W. Doug Wilder
24	BLND	4k	Bufferland ROM	Angel Martin
25	CCDP	8k	CCD Plus	Angel Martin
26	CCDR	8k	CCD Module	W&W GmbH
27	CCDX	4k	CCD OS/X	Raymond del Tondo
28	CHEM	4k	Chemistry User ROM	??
29	CHES	8k	Chess/Rubik's ROM	Claude Roetlgen
30	CIRC	4K	Circuit Analysis Pac	HP Co.
31	CLIN	4K	Clinical Lab Pac	HP Co.
32	CLUT	4k	CL Utilities	Angel Martin
33	CURV	8k	Curve-Fitting Module	Angel Martin
34	CVPK	8k	Cv-Pack ROM	??
35	DA4C	4k	DisAssembler 4C	W. Doug Wilder
36	DACQ	8k	Data Acquisition Pac	HP Co.
37	DASM	4k	DisAssembler 4D	W. Doug Wilder
38	DAVA	4K	David Assembler 2C	David van Leeuwen
39	DEVI	8k	HP-IL Development	HP Co.
40	DIIL	4k	HP-IL Diagnostics	HP Co.
41	DMND	4k	Diamond ROM	??
42	DYRK	4k	Dyerka ROM	David Yerka
43	E41S	8k	ES41 Module	Eramco
44	ESML	4k	ES MLDL 7B	Eramco
45	EXIO	4k	Extended I/O Module	HP Co.
46	EXTI	4k	Extended-IL ROM	Ken Emery
47	FACC	4k	300889_FACC	??
48	FINA	4k	Financial Pac	HP Co.
49	FRTH (*)	8k	FORTH Module	Serge Vaudenay
50	FUNS	8k	Fun Stuff Module	Angel Martin
51	GAME	4k	Games Pac	HP Co.
52	GMAS	4k	Auto Fiance-2 Module	GMAC
53	GMAT	8k	Auto Fiance-3 Module	GMAC
54	HCMP	4k	HydraComp ROM	Paul Monroe
55	HEPR	4k	HEPAX RAM Template	VM Electronics

56	HEPX	16k	HEPAX Module	VM Electronics
57	HOME	4k	Home Management. Pac	HP Co.
58	ICDO	4k	Icode ROM	??
59	IDC1	8k	ML-ICD	BCMC 1987
60	IDC2	4k	BG/UG IDC	BCMC 1985
61	JMAT	8k	JMB Math	Jean-Marc Baillard
62	JMTX	8k	JMB Matrix	Jean-Marc Baillard
63	ILBF	4k	IL-Buffer	Angel Martn
64	KC135	12k	Weight & Balance Comp.	??
65	L119	8k	AFDC-1E-003	Zengun
66	LAIT (*)	4k	LaitRAM XQ2	LaitRam Corp.
67	LAND	4k	Land Navigation ROM	Warren Furlow
68	LBLS	4k	Labels ROM	W. Doug Wilder
69	MADV	12k	Modified Advantage ROM	Angel Martn
70	MATH	4k	Math Pac	HP Co.
71	MCHN	4k	Machine Construction Pac	HP Co.
72	MDP1	8k	AFDC-1F ROM	Zengun
73	MDP2	8k	AFDC-1F ROM	Zengun
74	MELB	4k	Melbourne ROM	PPC Members
75	MILE	8k	Military Engineering ROM	??
76	MLBL	4k	Mainframe Labels	David van Leeuwen
77	MLRM	4K	ML ROM	Frits Ferwerda
78	MLTI	8k?	Multi-Prec. Library	Peter Platzer
79	MTRX	4k	MATRIX ROM	Angel Martin
80	MTST	4k	MC Test ROM	??
81	MUEC	8k	Muecke ROM	Mücke Software GmbH
82	NAVI	8k	Navigation Pac	HP Co.
83	NCHP	4k	NoVoCHAP	G. Isene & A. Martin
84	NFCR	4k	NFC ROM	Nelson F. Crowe
85	NPAC	8k	NavPac ROM	??
86	NVCM	8k	NaVCOM 2	??
87	OILW	8k	OilWell Module	Jim Daly
88	P3BC	16k	Aviation for P3B/C	??
89	PANA	8k	PANAME ROM	S. Bariziene & JJ Dhenin
90	PARI	4k	PARIO ROM	Nelson F. Crowe
91	PCOD	4k	Proto-Coder 1A	Nelson F. Crowe
92	PETR	8k	Petroleum Pac	HP Co.
93	PLOT	8k	Plotter Module	HP Co.
94	PMLB	4k	PPC Melb ROM	PPC Members
95	POLY	8k	Polynomial Analysis	A. Martin & JM Baillard
96	PPCM	8k	PPC ROM	PPC Members
97	PRFS	4k	ProfiSet	Winfried Maschke
98	PRIQ	8k	PRIDE ROM	??
99	QUAT	8k	Quaternion ROM	Jean-Marc Baillard
100	RAMP	4k	RAMPage Module	Angel Martin
101	REAL	8k	Real State Pac	HP Co.
102	ROAM	4k	ROAM Module	Wilson B. Holes
103	ROMS	4k	SV's ROM	Serge Vaudenay
104	SANA	12k	SandMath-12k	Angel Martin
105	SBOX	8k	SandBox	Angel Martin
106	SEAK	4k	SeaKing MK5	Navy Air
107	SECY	4k	Securities Pac	HP Co.
108	SGSG	4k	Gas Module	SGS Redwood
109	SIMM	16k	SIM Module	??
110	SKWD	4k	Skwid's BarCode	Ken Emery
111	SMCH	8k	Speed Machine	Alameda Mngmt. Corp.
112	SMPL	4k	Simplex Module	Phillipe J. Roussel
113	SMTS	8k	SandMath-8k	Angel Martin

114	SND2	8k	SandMath-II	Angel Martin
115	SPEC	4k	Spectral Analysis	Jean-Marc Baillard
116	SRVC (*)	4k	Service ROM	HP Co.
117	STAN	4k	Standard Pac	HP Co.
118	STAT	4k	Statistics Pac	HP Co.
119	STRE	4k	Stress Analysis Pac	HP Co.
120	STRU	8k	Structural An, Pac	HP Co.
121	SUPR	8k	SUP-R-ROM	James W. Vick
122	SURV	4k	Surveying Pac	HP Co.
123	THER	4k	Thermal Pac	HP Co.
124	TOMS	4k	Tom's ROM	Thomas A. Bruns
125	TOOL	4k	ToolBox-II	Angel Martin
126	TREK	4k	Start Trek	Angel Martin
127	TRIH	4k	83Trinh	Phil Trinh
128	UNIT	4k	Unit Conversion	Angel Martin
129	USPS	8k	Mail Delivery	USPS
130	XXXA	4k	*Empty*	*Not listed*
131	XXXB	4k	*Empty*	*Not listed*
132	XXXC	4k	*Empty*	*Not listed*
133	XXXD	8k	*Empty*	*Not listed*
134	XXXE	8k	*Empty*	*Not listed*
135	XXXF	16k	*Empty*	*Not listed*
136	YFNS	4k	Alternate YFNS	Monte Dalrymple
137	YFNZ	4k	Main YFNS	Monte Dalrymple
138	Z41Z	8k	41Z Module	Angel Martin
139	ZENR	4k	Zenrom	Zengrange Ltd.
140	ZEPR	4k	Programmer	Zengrange Ltd.

(*) Take-over ROMS

Other modules not included in the Library:-

For sure many more of these abound, yet these are the ones I have close knowledge of – feel free to complete the list with your own entries, and don't forget to share them with the whole community.

1.	CCD Advanced Apps.	4k	Ángel Martin
2.	DigitPac ROM	4k	Ángel Martin
3.	Geometry 2011	4k	Jean-Marc Baillard
4.	Market Forecast	4k	Forecaster?
5.	MONOPOLY ROM	8k	Thomas Rodke
6.	Mortar Fire Data Calculator	8k	MDN Canada
7.	Mountain Computer EPROM	4k	Paul Lind
8.	Number Theory ROM	4k	Jean-Marc Baillard
9.	Dr. Z RaceTrack Module	4k	William T. Ziemba
10.	SNEAP1/2/3	*3x* 8k	SNEAP Society (F)
11.	SUDOKU & Sound	4k	JM Baillard & Á. Martin
12.	VECTOR Analysis	4k	Ángel Martin
13.	Yach Computer	4k	Bobby Schenk

Modules included in Flash without Module ID#

1.	ISENE ROM	0x0C9	Geir Isene
2.	Bus Sales/Mkt/Stat.	0x0CA	HP Co.
3.	Control Systems	0x0CB	HP Co.
4.	Electrical Eng.	0x0CC	HP Co. + ÁM
5.	Lend, Lease & Sav.	0x0CD	HP Co.
6.	Test Statistics	0x0CE	HP Co.
7.	Mechanical Eng.	0X0CF	HP Co.+ ÁM
8.	Antennas	0x0D1	HP Co.
9.	Optometry I & II	0x0D2	HP Co.
10.	Physics	0x0D3	HP Co.
11.	Geometry	0x0D4	HP Co. + ÁM
12.	High-Level Math	0x0D5	HP Co.
13.	Interchang. Sol.	0x0D6	UPLE
14.	Module Database	0x0D7	MD

Appendix 2. FOCAL program Listings.

Provided for your reference and in case you feel like experimenting with your own settings.

As always, mind the potential conflicts with other modules when plugging stuff, and pay special attention not to overwrite YFNS. (you're safe if using **PLUGGX** – it won't let you to :-)

In the HEPAX configuration code the role of **HEPINI** is to write the appropriate words into the HRAM pages, as per the description provided before. This could also be done using **YPOKE**, but the memory requirements are much larger due to all the alpha strings that would be required to do so.

For example, see below for the 16k case, using pages C,D,E, and F.

This would mean having to write on each page the four page id#s, plus the pointers to the previous and next pages, for a total of 10x – or equivalent to 110 bytes:

"809FE7-000C"
"808000-000C"

"808FE8-000D"
"80AFE7-000D"
"809000-000D"

"809FE8-000E"
"80BFE7-000E"
"80A000-000E"

"80AFE8-000F"
"80B000-000F"

```
01  LBL "HPX4"     Entry for 4k
02  E
03  GTO 02
04  LBL "HPX8"     Entry for 8k
05  2
06  GTO 02
07  LBL "HPX16"    Entry for 16k
08  0
09  LBL 02
10  X<>F           flag setting
11  TURBO50        run as fas as possible
12  "0B9>808"      prepare for 0x808
13  AVIEW          provide feedback
14  YMCPY          copy to 0x808 in RAM
15  FS? 00         4k case?
16  GTO 00         yes, skip the rest
17  YBSP           remove last chr
18  "`9"           replace it
19  AVIEW          provide feedback
20  YMCPY          copy to 0x809 in RAM
21  FS? 01         8k case?
22  GTO 00         yes, skip the rest
23  YBSP           remove last char
24  "`A"           replace it
25  AVIEW          provide feedback
26  YMCPY          copy to 0x80A in RAM
27  YBSP           remove last char
28  "`B"           replace it
29  AVIEW          provide feedback
30  YMCPY          copy to 0x80B in RAM
31  LBL 00
32  "SETUP..."     announce last phase
33  AVIEW
34  "-RAM"         buffer common string
35  ASTO L         in temporary storage
36  "808"          build first mapping text
37  ARCL L
38  PLUG4U         plug 0x808 to page#F
39  FS? 00         4k case?
40  GTO 01         yes, jump over
41  "809"          build 2nd. Mapping text
42  ARCL L
43  PLUG4L         plug 0x809 to page#E
44  FS? 01         8k case?
45  GTO 03         yes, jupm over
46  "80A"          build 3er. Mapping text
47  ARCL L
48  PLUG3U         plug 0x80A to page#D
49  "80B"          build 4th. Mapping text
50  ARCL L
51  PLUG3L         plug 0x80B to port#C
52  4              configuration parameters:
53  ENTER^         four pages
54  12             starting at page#C
55  GTO 00
56  LBL 03         8k case
57  2              config parm:
58  ENTER^         two pages
59  14             starting at page#E
60  GTO 00
61  LBL 01         4k case
62  E              config parm:
63  ENTER^         just one page,
64  15             starting at page#F
65  LBL 00
66  HEPINI         configure HEPAX FileSys
67  E              get page# below
68  -
69  "HEPX"
70  PLUGGX         plug HEPAX there
71  HEPDIR         show we've done it
72  END            done.
```

Apendix 3.- MCODE Listing showing the Alphabetical sections prompting code.

The function CLLIB begins by building the prompt text in the display. Using the OS routine [PROMF2] is helpful to save bytes, so there's no need to write the function name again, "CLLIB". Alpha is cleared using [CLA], just to prepare for a possible copy of the ROM id# to Alpha using the [A] hot-key in run mode. Then we get into a partial data entry "condition", waiting for a key to be pressed.

Back Arrow sends the execution to [EXIT3], to do the housekeeping required to reset everything back to the standard OS-required status (disable Display, resetting Keyboard bits, CPU flags, etc.).

Since the valid keys are quite a lot [A-Z] we need to use multiple conditions in the logic. The first two rows are the easiest; as they set up CPU flag#4 and that can be tested easily. In this case we copy the mantissa sign in A to C[S&X], then store it in B[S&X] and we move on.

1	CLLIB	BCKARW	A66E	341	PORT DEP:	
2	CLLIB		A66F	08C	GO	
3	CLLIB		A670	36E	->A36E	[EXIT3]
4	CLLIB	Header	A671	082	"B"	
5	CLLIB	Header	A672	009	"I"	
6	CLLIB	Header	A673	00C	"L"	
7	CLLIB	Header	A674	00C	"L"	**CL Library**
8	CLLIB	Header	A675	003	"C"	Prompting - Alphabetical
9	CLLIB	CLLIB	A676	000	NOP	
10	CLLIB		A677	158	M=C ALL	
11	CLLIB		A678	345	?NC XQ	Clears Alpha
12	CLLIB		A679	040	->10D1	[CLA]
13	CLLIB		A67A	3C1	?NC XQ	Enable & Clear Display
14	CLLIB		A67B	0B0	->2CF0	[CLLCDE]
15	CLLIB		A67C	198	C=M ALL	fct id#
16	CLLIB		A67D	34D	?NC XQ	
17	CLLIB		A67E	014	->05D3	[PROMF2]
18	CLLIB		A67F	3BD	?NC XQ	
19	CLLIB		A680	01C	->07EF	[MESSL]
20	CLLIB		A681	001	"A"	
21	CLLIB		A682	02D	"-"	
22	CLLIB		A683	21A	"Z"	
23	CLLIB		A684	115	?NC XQ	**Partial Data Entry!**
24	CLLIB		A685	038	->0E45	**[NEXT1]**
25	CLLIB		A686	343	JNC -24d	
26	CLLIB		A687	04C	?FSET 4	**rows 1 & 2?**
27	CLLIB		A688	063	JNC +12d	
28	CLLIB		A689	130	LDI S&X	
29	CLLIB		A68A	00A	CON:	
30	CLLIB		A68B	35E	?A#0 MS	was "J" pressed?
31	CLLIB		A68C	023	JNC +04	yes, skip next
32	CLLIB		A68D	046	C=0 S&X	
33	CLLIB		A68E	0BE	A<>C MS	
34	CLLIB		A68F	2FC	RCR 13	
35	CLLIB		A690	0E6	C<>B S&X	save chr# in B[S&X]
36	CLLIB		A691	369	PORT DEP:	Transfer to next section
37	CLLIB		A692	03C	GO	
38	CLLIB		A693	2E9	->A6E9	[MOVEON]
39	CLLIB		A694	0B0	C=N ALL	pressed keycode
40	CLLIB		A695	106	A=C S&X	save in A[S&X] for comparisons
41	CLLIB		A696	21C	PT= 2	adjust pointer
42	CLLIB		A697	05A	C=0 M	reset counter
43	CLLIB		A698	130	LDI S&X	upper value
44	CLLIB		A699	01A	"Z"	from there down!
45	CLLIB		A69A	1BC	RCR 11	put it in C[M]
46	CLLIB		A69B	013	JNC +02	skip over line
47	CLLIB		A69C	343	JNC -24d	pipe-lining up

For the rest [K-Z] we'll need to read the keycode of the pressed key and act accordingly. Also *we need to discard any non-letter key,* rejecting it if its keycode value is outside of the [A,Z] range.

Now the show is about to start: see how the key pressed value (in N) is compared with every possible value in the [K-Z] range, building the "pointer" in C[S&X] by repeat one-additions until coming up to its final result.

48	CLLIB		A69D	130	LDI S&X	
49	CLLIB		A69E	120	CON:	XEQ keycode [120]
50	CLLIB		A69F	366	?A#C S&X	
51	CLLIB		A6A0	1C3	JNC +56d	[K]
52	CLLIB		A6A1	222	C=C+1 @PT	keycode [220]
53	CLLIB		A6A2	366	?A#C S&X	
54	CLLIB		A6A3	1B3	JNC +54d	[L]
55	CLLIB		A6A4	222	C=C+1 @PT	keycode [320]
56	CLLIB		A6A5	366	?A#C S&X	
57	CLLIB		A6A6	1A3	JNC +52d	[M]
58	CLLIB		A6A7	130	LDI S&X	
59	CLLIB		A6A8	030	CON:	ENTER^ keycode [030]
60	CLLIB		A6A9	366	?A#C S&X	
61	CLLIB		A6AA	18B	JNC +49d	[N]
62	CLLIB		A6AB	222	C=C+1 @PT	keycode [130]
63	CLLIB		A6AC	222	C=C+1 @PT	keycode [230]
64	CLLIB		A6AD	366	?A#C S&X	
65	CLLIB		A6AE	173	JNC +46d	[O]
66	CLLIB		A6AF	222	C=C+1 @PT	keycode [330]
67	CLLIB		A6B0	366	?A#C S&X	
68	CLLIB		A6B1	163	JNC +44d	[P]
69	CLLIB		A6B2	130	LDI S&X	
70	CLLIB		A6B3	040	CON:	"-" keycode [040]
71	CLLIB		A6B4	366	?A#C S&X	
72	CLLIB		A6B5	14B	JNC +41d	[Q]
73	CLLIB		A6B6	222	C=C+1 @PT	keycode [140]
74	CLLIB		A6B7	366	?A#C S&X	
75	CLLIB		A6B8	13B	JNC +39d	[R]
76	CLLIB		A6B9	222	C=C+1 @PT	keycode [240]
77	CLLIB		A6BA	366	?A#C S&X	
78	CLLIB		A6BB	12B	JNC +37d	[S]
79	CLLIB		A6BC	222	C=C+1 @PT	keycode [340]
80	CLLIB		A6BD	366	?A#C S&X	
81	CLLIB		A6BE	11B	JNC +35d	[T]
82	CLLIB		A6BF	130	LDI S&X	
83	CLLIB		A6C0	050	CON:	"+" keycode [050]
84	CLLIB		A6C1	366	?A#C S&X	
85	CLLIB		A6C2	103	JNC +32d	[U]
86	CLLIB		A6C3	222	C=C+1 @PT	keycode [150]
87	CLLIB		A6C4	366	?A#C S&X	
88	CLLIB		A6C5	0F3	JNC +30d	[V]
89	CLLIB		A6C6	222	C=C+1 @PT	keycode [250]
90	CLLIB		A6C7	366	?A#C S&X	
91	CLLIB		A6C8	0E3	JNC +28d	[W]
92	CLLIB		A6C9	222	C=C+1 @PT	keycode [350]
93	CLLIB		A6CA	366	?A#C S&X	
94	CLLIB		A6CB	0D3	JNC +26d	[X]
95	CLLIB		A6CC	130	LDI S&X	
96	CLLIB		A6CD	060	CON:	"^" keycode [060]
97	CLLIB		A6CE	366	?A#C S&X	
98	CLLIB		A6CF	0BB	JNC +23d	[Y]
99	CLLIB		A6D0	222	C=C+1 @PT	keycode [160]
100	CLLIB		A6D1	366	?A#C S&X	
101	CLLIB		A6D2	0AB	JNC +21	[Z]
102	CLLIB		A6D3	265	?NC XQ	Blink Display - pass #1
103	CLLIB		A6D4	020	->0899	[BLINK1]
104	CLLIB		A6D5	265	?NC XQ	Blink Display - pass #2
105	CLLIB		A6D6	020	->0899	[BLINK1]
106	CLLIB		A6D7	22B	JNC -59d	ONE PROMPT
107	CLLIB	[K]	A6D8	27A	C=C-1 M	
108	CLLIB	[L]	A6D9	27A	C=C-1 M	
109	CLLIB	[M]	A6DA	27A	C=C-1 M	
110	CLLIB	[N]	A6DB	27A	C=C-1 M	
111	CLLIB	[O]	A6DC	27A	C=C-1 M	
112	CLLIB	[P]	A6DD	27A	C=C-1 M	
113	CLLIB	[Q]	A6DE	27A	C=C-1 M	
114	CLLIB	[R]	A6DF	27A	C=C-1 M	
115	CLLIB	[S]	A6E0	27A	C=C-1 M	
116	CLLIB	[T]	A6E1	27A	C=C-1 M	
117	CLLIB	[U]	A6E2	27A	C=C-1 M	
118	CLLIB	[V]	A6E3	27A	C=C-1 M	
119	CLLIB	[W]	A6E4	27A	C=C-1 M	
120	CLLIB	[X]	A6E5	27A	C=C-1 M	
121	CLLIB	[Y]	A6E6	27A	C=C-1 M	
122	CLLIB	[Z]	A6E7	03C	RCR 3	place it in C[S&X]
123	CLLIB		A6E8	0E6	C<>B S&X	save chr# in B[S&X]

The last part is about presenting the chosen key – allowing NULLing if it's held down long enough – Resetting everything back to normal conditions [CLNUP], and see whether there actually exists such a section – before we launch into a blindfold enumeration. This is done by the subroutine [SRCHR], which will fetch the address in the ROM id #table where the section starts. With that we'll transfer the execution to the **ROMLIB** function code where the actual enumeration will take place - only with a padded value to start from, as opposed to doing it from the top of the table.

124	CLLIB	MOVEON	A6E9	379	PORT DEP:	CleanUp and Show
125	CLLIB		A6EA	03C	XQ	Allows NULLing
126	CLLIB		A6EB	261	->A661	[CLNUP]
127	CLLIB		A6EC	149	?NC XQ	Disable PER, enable RAM
128	CLLIB		A6ED	024	->0952	[ENCP00]
129	CLLIB		A6EE	215	?NC XQ	Reset BIT sequence
130	CLLIB		A6EF	00C	->0385	[RSTSQ]
131	CLLIB		A6F0	130	LDI S&X	Location of first ID
132	CLLIB		A6F1	32B	<start of search>	[romtbl]
133	CLLIB		A6F2	106	A=C S&X	save offset in A[S&X]
134	CLLIB		A6F3	349	PORT DEP:	Search Char in B[S&X]
135	CLLIB		A6F4	08C	XQ	start offset in A[S&X]
136	CLLIB		A6F5	353	->A353	[SRCHR]
145	CLLIB		A6F6	023	JNC +04	not found, bail out with style
146	CLLIB		A6F7	341	PORT DEP:	Transfer code to Main LIB
147	CLLIB		A6F8	08C	GO	
148	CLLIB		A6F9	386	->A386	[MERGE]
149	CLLIB	INFMSG	A6FA	261	?NC XQ	debounce keyboard
150	CLLIB		A6FB	000	->0098	[RSTKB]
151	CLLIB		A6FC	3B5	PORT DEP	Display Message
152	CLLIB		A6FD	08C	XQ	Unless Error Flag is set.
153	CLLIB		A6FE	3EE	->AFEE	[DSPERR]
154	CLLIB		A6FF	00E	"N"	
155	CLLIB		A700	00F	"O"	
156	CLLIB		A701	020	" "	
157	CLLIB		A702	013	"S"	
158	CLLIB		A703	015	"U"	
159	CLLIB		A704	003	"C"	
160	CLLIB		A705	208	"H"	
161	CLLIB		A706	389	PORT DEP	Output Message
162	CLLIB		A707	08C	GO	
163	CLLIB		A708	016	->A816	[APEREX]

Note how [SRCHR] is really part of the **ADRID** function code, which also does table look-ups for its own purpose. This code is written around the table structure; refer to the Blueprints for more details.

32	ADRID	SRCHR	A353	04E	C=0 ALL	Expects chr# in B[S&X]
33	ADRID		A354	35D	?NC XQ	
34	ADRID		A355	000	->00D7	[PCTOC]
35	ADRID		A356	03C	RCR 3	get page number
36	ADRID		A357	130	LDI S&X	
37	ADRID		A358	300	CON:	
38	ADRID		A359	1E6	C=C+C S&X	double it up
39	ADRID		A35A	206	C=C+A S&X	add offset to it
40	ADRID		A35B	1BC	RCR 11	put it in C(6:3)
41	ADRID		A35C	0BA	A<>C M	put it in A(6:3)
42	ADRID		A35D	066	A<>B S&X	put reference in A[S&X]
43	ADRID	NXTADR	A35E	0BA	A<>C M	bring adr to C[M]
44	ADRID		A35F	11A	A=C M	keep it in A[M]
45	ADRID		A360	330	FETCH S&X	read value at address
46	ADRID		A361	2E6	?C#0 S&X	value non-zero?
47	ADRID		A362	3A0	?NC RTN	NO, TERMINATE HERE!
48	ADRID		A363	366	?A#C S&X	are they different?
49	ADRID		A364	033	JNC +06	no, -> [FOUND]
50	ADRID		A365	05A	C=0 M	add offset : 5 BYTES
51	ADRID		A366	01C	PT=3	
52	ADRID		A367	150	LD@PT- 5	
53	ADRID		A368	15A	A=A+C M	next addr field
54	ADRID		A369	3AB	JNC -11d	loop back
55	ADRID	FOUND	A36A	1B0	POPADR	
56	ADRID		A36B	23A	C=C+1 M	
57	ADRID		A36C	170	PUSHADR	increase RTN adr
58	ADRID		A36D	3E0	RTN	

And that's all folks - easy when you know the tricks ☺

Appendix 4.- Quick & Dirty sRAM Editors.

RAMED and **ROMED** are capable of editing the complete content of the RAM and ROM plugged to the CL via the MMU settings, but obviously cannot be used to edit the content of the sRAM on the CL board.

The small FOCAL programs below close that gap, in a minimalistic approach that provides a simple but elegant way to drive YPEEK and YPOKE instructions automatically, so that the sRAM words can be edited.

There are two versions: **YEDIT** and **YEDIT+**. The first one only reviews the contents (useful to check whether the block contains the expected information), and the second also allows actual changes to it. Both versions use functions **WSIZE** and **ARCLH**, present in the AMC_OS/X module – therefore it must be plugged as well.

The execution starts by asking for the sRAM address, then it falls into a loop reviewing (and possibly editing) every word in a sequential manner, until it is ended. **YEDIT** allows moving to a different address without re-starting the program, just by entering the offset from the current one in X (in decimal) just before pressing **R/S**. For **YEDIT+** you'll need to re-start the program and input the new address.

01*	LBL "YEDIT+"
02	YINPT
03	ASTO X
04	12
05	WSIZE
06	CLX
07	LBL 05
08	CLA
09	ARCL Y
10	ARCLH
11	"|- -8888"
12	YPEEK
13	AVIEW
14	Y/N?
15	X<0?
16	GTO 02
17	E
18	+
19	GTO 05
20	LBL 02
21	YCL-
22	APPEND
23	YPOKE
24	GTO 05
25	END

01*	LBL "YEDIT"
02	YINPT
03	ASTO X
04	12
05	WSIZE
06	CLX
07	LBL 05
08	CLA
09	ARCL Y
10	ARCLH
11	"|- -8888"
12	YPEEK
13	PROMPT
14	E
15	+
16	GTO 05
17	END

It comes without saying that YEDIT+ wont be able to edit the data in Flash sectors, although you can use it to review its contents.

Appendix 5.- Serial Transfer CLWRITE source code. – by Raymond Wiker.

```
1) Copy YFNS-1A to RAM at 80C000, and patch for items 2, 5, 8. These
 affect the operation of YIMP. I did this with a variation of the
 PATCHIT program posted earlier.

 2) Execute TURBO50.

 3) Execute SERINI

 4) Execute BAUD12. From the documentation, this should not be
 necessary, but I had to explicitly set 1200 baud to get the transfer
 to work.

 5) The file yfns-1e.rom has the opposite byte order of what YIMP
 expects, so the transfer program needs to perform byte swapping.
 Alternatively, you might do the byte-swapping before you do the
 transfer.

 6) Transfer the ROM; I chose to transfer it to 80D000 (i.e, put
 80D000-0FFF in the alpha register, start YIMP). For the transfer, I
 used the CLWriter program that I posted a few days back, with the
 command

 CLWriter.exe yfns-1e-fixed.rom com1 1200 5

 --- the file yfns-1e-fixed.rom is the byte-swapped version of
 yfns-1e.rom. I probably should have chosen a slightly different
 name, but that does not really matter. The "5" means that I put a 5
 millisecond delay after each byte. It may not actually be necessary;
 I added it because I got timeouts, but these were probably because I
 left out step 4 (BAUD12).

 7) Execute PLUG1L with "80D-RAM" in the Alpha register.

 8) Verify (using CATALOG 2) that I'm now running YFNS-1E.
```

```
using System;
using System.IO;
using System.IO.Ports;
using System.Threading;

public class CLWriter
{
   public static void Main(string [] args)
   {
         int baudrate = 1200;
      int delay = 0;
         if (args.Length < 2) {
            Console.Error.WriteLine("Usage:");
            Console.Error.WriteLine("    {0} file port [baudrate [delay]]", "CLWriter");
            Console.Error.WriteLine();
            Console.Error.WriteLine("Where baud defaults to {0}", baudrate);
            Console.Error.WriteLine("and delay defaults to {0}", delay);
            Console.Error.WriteLine("Available Ports:");
            Console.Error.WriteLine();
            foreach (string s in SerialPort.GetPortNames())
            {
                  Console.Error.WriteLine("   {0}", s);
            }
            return;
         }
         string filename = args[0];
         string portname = args[1];
         if (args.Length > 2) {
            baudrate = int.Parse(args[2]);
            if (baudrate != 1200 && baudrate != 2400 &&
                baudrate != 4800 && baudrate != 9600) {
                Console.Error.WriteLine("Invalid baudrate {0}; should be one of", baudrate);
                Console.Error.WriteLine("1200, 2400, 4800, 9600");
                return;
            }
         }
      if (args.Length > 3) {
            delay = int.Parse(args[3]);
            if (delay > 10) {
              Console.Error.WriteLine("delay {0} probably too large.", delay);
              return;
            }
         }

         if (!File.Exists(filename)) {
            Console.Error.WriteLine("File {0} does not exist.", filename);
            return;
         }

         FileStream fstream = File.Open(filename, FileMode.Open);

         if (fstream.Length > 8192) {
           Console.Error.WriteLine("WARNING: {0} is over 8192 bytes long ({1});", filename,
fstream.Length);
           Console.Error.WriteLine("Will only transfer the first 8192 bytes.");
         }

         BinaryReader binReader = new BinaryReader(fstream);
```

```
        SerialPort serialport = new SerialPort();
        serialport.PortName = portname;
        serialport.BaudRate = baudrate;
        serialport.Parity = Parity.None;
        serialport.DataBits = 8;
        serialport.StopBits = StopBits.One;
        serialport.Handshake = Handshake.None;

        serialport.Open();

        try {
           byte[] buffer = new byte[8192];
           int count = binReader.Read(buffer, 0, 8192);

           // swap high & low bytes:
           for (int i = 0; i < count; i+= 2) {
              byte tmp = buffer[i];
                 buffer[i] = buffer[i+1];
                 buffer[i+1] = tmp;
           }

           for (int i = 0; i < count; i++) {
                 Console.Write("{0:x2} ", buffer[i]);
                 if (i % 16 == 15) {
                   Console.WriteLine();
                 }
                 serialport.Write(buffer, i, 1);
                 if (delay > 0) {
                    Thread.Sleep(delay);
                 }
           }
           Console.WriteLine();
      }
        catch (EndOfStreamException) {
           // nada
        }
      serialport.Close();
   }
}
```

HP-41 MCODE: Checking ROMs Configuration

*Posted by **Ángel Martin** on **18 Mar 2012, 3:23 a.m.**, Report post*

Problem Description.

You just plugged a module (Clonix, Standard, or "virtual" on the CL / MLDL) and your trusty 41 starts acting up when you try to execute any function. You of course suspect the amateur programmer who put it together and didn't do enough testing - but on second thought you wonder if there could be any XROM conflict in the current configuration.

Even HP-made modules suffered from this in the old days (they were marked with an "x" on the module case label) - but today is more frequent given the open access to every module on Clonix/MLDL, with the CL topping the list: a simple PLUG command for a ROM in its vast library can take you there.

Function Features.

The Function listed below determines whether such a conflict really exists. It shows a "CONFIG OK/BAD" statement to assess the status, and if there are conflicts it will sequentially show ALL conflicting XROM numbers if multiple offenders, ending with the BAD status message.

It will also discard "false positives" caused by multi-page ROMS (i.e. 2 or more 4k-blocks) with just one single FAT.

Checking Method.

Here's a crude sketch in basic to help define the context to get started. The idea is to check the first byte in pages 3 to F, looking for duplicates.

```
FOR J= 3 TO 14
W = FETCH(J)
FOR K=J+1 TO 15
V = FETCH (K)
IF V=W THEN -> BAD STUFF
NEXT K
NEXT J
```

A check for the second byte is used to determine whether the 4k page has an XROM id# but it has NO functions.

Routine Code.

Listed below is my implementation of CHKCGF. It assumes a system with the CX OS, as there are a couple of calls to the message routines.

The FOR-NEXT loop J and K indexes are stored in CPU registers N and M respectively.

The enumeration starts at page#3, thus catching the CX-XFUNCT. It is compatible with Page#4-equipped systems, since its "fake" XROM id# is 35 - more about this on another article.

```
1       REPORT  215     ?NC XQ      Build Msg - all cases
2               0FC     ->3F85      [APRMSG2]
3               003     "C"
4               00F     "O"
5               00E     "N"
6               006     "F"
7               009     "I"
8               007     "G"
9               220     " "
10              24C     ?FSET 9     were there issues?
11              037     JC  +06     [BADONE]
12              3BD     ?NC XQ
13              01C     ->07EF      [MESSL]
14              00F     "O"
15              20B     "K"
16              033     JNC  +06    [SHOWME]
17      BADONE  3BD     ?NC XQ
18              01C     ->07EF      [MESSL]
19              002     "B"
20              001     "A"
21              204     "D"
22      SHOWME  1F1     ?NC GO      Show and Halt
23              0FE     ->3F7C      [APEREX]
24              087     "G"
25              006     "F"
26              003     "C"
27              00B     "K"
28              008     "H"
29              003     "C"
30      CHKCFG  1A0     A=B=C=0     Start w/ clean slate
31              244     CLRF 9
32              15C     PT=  6
33              090     LD@PT- 2    puts "2" in C(6)
34              070     N=C ALL     now in N(6)
35      NEXTJ   0B0     C=N ALL
36              15C     PT=  6
37              222     C=C+1 @R    increases page#
38                      JC  -37d    [REPORT]
39              070     N=C ALL     update 1st. Counter
40              330     FETCH S&X   get 1st. Word (j)
41              106     A=C  S&X    store in A[S&X]
42              23A     C=C+1 M     increase address
43              330     FETCH S&X   get 2nd. Word (j)
44              2E6     ?C#0 S&X    empty FAT?
45              3B3     JNC -10d    yes, [NEXTKJ]
46              0B0     C=N ALL     bring counter back
47              222     C=C+1 @R    add one: out of bounds?
48              28F     JC  -47d    yes, we're done
49              158     M=C ALL     reset 2nd. Counter
50              033     JNC +06     1st. Time is different
51      NEXTK   198     C=M ALL
52              15C     PT=  6
53              222     C=C+1 @R    increases page#
```

```
54              36F     JC -19d     checked all pages?
55              158     M=C ALL     update 2nd. Counter
56      1STONE  330     FETCH S&X   get 1st. Word (k)
57              0E6     C<>B S&X    store in B[S&X]
58              23A     C=C+1 M     increase address
59              330     FETCH S&X   get 2nd. Word (k)
60              2E6     ?C#0 S&X    empty FAT?
61              3B3     JNC -10d    yes, -> [NEXTK]
62              0C6     C=B  S&X    get 1st. Word (k)
63              366     ?A#C S&X    same xrom id#'s ?
64              39F     JC -13d     no, -> [NEXTK]
65              248     SETF  9     yes, flag it as "dirty" cfg
66              215     ?NC XQ      Build Msg - all cases
67              0FC     ->3F85      [APRMSG2]
68              004     "D"
69              015     "U"
70              010     "P"
71              020     " "
72              018     "X"
73              012     "R"
74              00F     "O"
75              00D     "M"
76              220     " "
77              066     A<>B S&X    value to convert in A[S&X]
78              01E     A=0 MS
79              3A1     ?NC XQ      Generate number ->display!
80              014     ->05E8      [GENNUM]
81              031     ?NC XQ      Display not halting
82              100     ->400C      [APERX4]
83              37D     ?NC XQ      Cancels TURBO mode
84              13C     ->4FDF      [TURBO0]
85              046     C=0 S&X
86              2A6     C=-C-1 S&X  Delay loop
87              266     C=C-1 S&X
88              3FB     JNC  -01
89              38D     ?NC XQ      Sets TURBO mode
90              13C     ->4FE3      [TURB50]
91              2C3     JNC -40d    [NEXTK]
```

Notes.

Lines 83-84 and 89-90 are used to remove the TURBO mode during the displaying of the messages. They're only relevant to CL systems and have no effect on other setups.

Lines 81-82 are a call to the Page#4 Library. It's for a non-halting [APEREX].

As you can see it's relatively simple and requires no deep knowledge of the 41 OS internals. I use it all the time on my CL - for obvious reasons - where I´m changing the configuration very frequently, and now that I have it I can´t live without it. It´s a very handy tool, I´m surprised nobody has done it before.

"AM

Edited: 19 Mar 2012, 4:31 p.m.

Edit Message Delete Message

[**Return to the Message Index**]