41CL Update Functions

Every effort has been made to ensure the accuracy of the information contained herein. If you find errors or inconsistencies please bring them to our attention.

Notice:

Acknowledgements:

Sylvain Côté provided the impetus for this set of CL functions, and wrote the *clupdate* host-side communication software. Sylvain Côté, Robert Prosperi and Ángel Martin served as beta testers.

Table of Contents

Introduction

Keeping the contents of Flash memory on your 41CL current can be a daunting task, as new images have been added, and existing images updated, fairly regularly over the life of the 41CL. While it is possible to use existing functions in the *41CL Extra Functions* or *41CL Extreme Functions* to update your 41CL, the *41CL Update Functions* provide a set of dedicated functions that will significantly simplify the update process, and this is the recommended software to use to update the 41CL. Note that you must have the serial connector installed in your 41CL to use the *41CL Update Functions*, because these functions use the 41CL serial port.

When updating Flash memory, it is useful to be familiar with the physical memory organization of the 41CL. Refer to the *41CL Calculator* manual and the *41CL Memory Reference* documents for this information.

The *41CL Update Functions* are completely independent of any other software, and can be plugged into a port on the 41CL on an as-needed basis. Any port (6 through F) can be used for the *41CL Update Functions*.

Most of the functions required for the update process itself assume that the 41CL is connected to a host machine that contains the latest Flash Database and all of the latest software images. The *41CL Update Functions* communicate directly with the host machine over the serial port to provide status information and request downloads of data for writing to the Flash memory. This obviously means that the host machine must be running a dedicated 41CL update program, and have access to the latest ROM images to write to the Flash memory.

The host machine can be either another 41CL (running the *41CL Clone Functions* software) or a computer (PC, MAC or Linux box) running the *clupdate* software written by Sylvain Côté. Refer to the *41CL Clone Functions* manual and the *clupdate* manual for the details of operation of the host machine software.

The *41CL Update Functions* are grouped into seven categories:

1. The Flash Status functions provide a means to check on the exact configuration of the Flash memory device on the 41CL board, as well as a way to check on the revision date of both the Flash Database (FLDB) and the Image Database (IMDB) on the board.

2. The Page functions allow pages to be copied, initialized (in RAM), erased from Flash memory, or written to the Flash memory.

3. The Flash Sector functions are included to make the *41CL Update Functions* easier to use for manual updates. These functions allow sectors (eight contiguous pages) to be copied, initialized (in RAM), erased from Flash memory, updated (erased and written in one contiguous operation), or written to the Flash memory.

4. The Correlated Flash Database functions manage the database that is central to the update process. Functions allow you to make database entries manually or automatically, to query the database and to delete the database.

5. The Communication functions can be used individually to communicate with the software running on the host machine, by sending the protocol-specific commands as well as pages of information in both directions. Functions are provided to open and close the communication channel, to download and upload the Correlated Flash Database, and to download or upload pages. These functions can be used to build your own communications protocol, or to manually interact with the update software running on the host machine. The functions which upload and download the Correlated Flash Database also communicate the board version running the *41CL Update Functions*, to guarantee that the update process is correct. If the host computer needs to know the board version (*clupdate* does) this means that you must download or upload the CFLDB to the host machine to communicate this version information.

6. The Auto-update functions automate the update process, communicating directly with the update software running on the host machine, while updating all, or part of, the Flash memory. Most users will use these functions, which make the update process simple.

7. The Mode Control functions allow you to select the mode of operation for some of the update functions.

Many of the *41CL Update Functions* use dynamic paging, where code is transiently loaded to Page 4 while a function is executed. Any image loaded to Page 4 (like Library-4) will be temporarily displaced by these functions and then restored before the function finishes. No physical module that uses Page 4 should be present in the calculator when using the *41CL Update Functions*. The HP-IL Module with the Printer Function Switch in the "DISABLE" position uses Page 4.

Warning

The auto-update functions do not require detailed knowledge of the 41CL physical memory organization, but if you plan to manually update the 41CL memory you should be familiar with both the normal *41CL Extra Functions* and the organization of the 41CL physical memory. The *41CL Update Functions* provide some protection against user errors, but it is always possible to accidently delete the wrong thing if you are not paying attention.

Getting the *41CL Update Functions* on your 41CL

The *41CL Update Functions* are new in 2017, so the vast majority of 41CL machines do not have this software pre-loaded in Flash memory. This means that you will need to download this software to your 41CL before you can actually use it to update the Flash memory. This section will detail how to do this. Once you have updated the Flash memory on your 41CL you won't need to download the *41CL Update Functions* again, because at that point the image will be resident in Flash memory for you to plug in and use at any time.

What follows requires the use of the serial port on the 41CL. If you did not install the serial connector you will need to find another way to update the Flash memory, because the *41CL Update Functions* only use the 41CL serial port.

Starting from the MEMORY LOST state, do the follwing:

XEQ "MMUCLR"	Make sure that the MMU registers are cleared. The MEMORY CLEAR condition does not affect the MMU registers in memory.
"YFNZ"	Select the default *41CL Extra Functions* for the PLUG command.
XEQ "PLUG1L"	Plug the *41CL Extra Functions* into the lower half of Port 1. This doesn't take effect yet because the MMU is still disabled.
XEQ "TURBO50"	The 50x Turbo mode is required for the 41CL to be able to keep up with a continuous stream of data from the PC.
XEQ "SERINI"	Initialize the serial port. This command sets the serial port to Async mode and clears any error conditions.
XEQ "BAUD48"	Set the baud rate to 4800. Although the 41CL serial port is capable of 9600 baud, at this rate an inter-character gap is required, even with 50x Turbo mode. So 4800 baud is safer. Don't forget that the selected baud rate is lost when the 41CL is turned off.
"830000-0FFF"	Address/length specifying physical address 0x83000 (page 0x830) and a block length of 4k words (8k bytes).
XEQ "YIMP"	Import the data. See below for the commands on the host side.

"830-RAM"	Direct address specification for Plug command
XEQ "PLUG1U"	Plug the RAM copy of the *41CL Update Functions* into the upper half of Port 1.
XEQ "MMUEN"	Enable the MMU.
CAT 2	Verify that both the *41CL Extra Functions* and the *41CL Update Functions* are plugged into the 41CL.

On the PC (using COM3 in these examples) you can either use the *CLWriter* software or the *clupdate* software. Note that you'll need to specify the current version of the *clupdate* software as well as the *41CL Update Functions*. The examples below are current as of the date of this document.

The *CLWriter* software is a little less typing: "CLWriter UPDAT-2B.ROM COM3 4800"

Since you're going to be using the *clupdate* software for the rest of the update, perhaps it doesn't matter: "java -jar clupdate-0.6.0.jar --upload UPDAT-2B.ROM COM3 4800"

If you have access to a 41CL that has an updated Flash memory, a 41CL-to-41CL serial cable, and are familiar with serial transfers on the 41CL, it's also possible to use another 41CL as the source for the *41CL Update Functions*. In this case you only need to do a complimentary **YEXP** to send the image. Don't forget that at least one of the 41CLs involved needs to contain a V4 or higher board revision, so that the serial port can be turned on via software.

Batteries and Execution Time

Several of the functions in the *41CL Update Functions* require significant time to complete, and as a consequence the condition of the 41CL batteries is important when doing an update. It is safest to install new batteries in the 41CL before starting an update, particularly if you are checking and updating the entire Flash memory. Even if you are only updating a portion of the Flash memory, the serial port itself nearly doubles the current drawn from the batteries while connected to the host computer, so fresh batteries are always a good idea.

The three functions in the *41CL Update Functions* where the battery condition is most important are the **FDBCHK?** (*Check Flash Database*) function, the **FLCHK?** (*Check Flash Memory Against Flash Database*) function, and the **FLUPD** (*Update Flash Memory*) function. All three of these functions are capable of operating on the entire Flash memory, and can take significant time complete even though they all automatically run in the 50x Turbo mode.

All three of the above functions can be aborted from the keyboard, and each time the keyboard is checked (either per page or per sector) the battery voltage is also checked. If a low battery condition is detected the function is automatically aborted and the ***BAT*** annunciator is activated in the display. If the ***BAT*** annunciator is already on when one of these functions is executed, the function will also abort, before execution of the function.

Many of the functions in the *41CL Update Functions* require more than the typical time to execute. The table below shows the average execution time for these functions.

Function	Execution time
BFRCHK?	1m, 21s
CDBDEL or **PGINI**	4s
CDBEXP or **PGEXP**	19s
CDBIMP or **PGIMP**	18s
FDB2CDB or **PGBFR** or **PGCPY**	5s
P8BFR or **P8CPY**	37s
P8ERASE or **PGERASE**	6s
P8INI	17s
P8UPD	2m, 7s
P8WR	2m, 0s
PGCRC	10s
PGWR	15s

The time required to check the entire Flash memory can be significant. The table below shows the worst case of the **FDBCHK?** function with a "*****" in the ALPHA register, Note that these times do not include the time required to download or upload the CFLDB.

Flash memory size	Execution time (approximate)
256 pages (V2)	42m
512 pages (V3 and V4)	1h, 15m
1024 pages (V5)	2h, 30m

The time required to update a sector (eight pages) in Flash memory depends on the number of pages that need to be downloaded before the update can proceed. The table below shows the different cases, using **P8CPY** plus n times **PGIMP** plus **P8UPD**.

Pages to update	Execution time (approximate)
1	3m, 1s
2	3m, 19s
3	3m, 37s
4	3m, 55s
5	4m, 13s
6	4m, 31s
7	4m, 49s
8	5m, 7s

The time required to update the entire Flash memory obviously depends on the number of pages that need to be updated. The absolute worst case will require updating every page in every sector except for the OS sector. The table below shows the different cases:

Flash memory size	Execution time (approximate)
256 pages (V2)	2h, 39m
512 pages (V3 and V4)	5h, 23m
1024 pages (V5)	10h, 50m

Note that all of the times in the preceeding three tables are calculated averages, rather than measured results.

Function Arguments

Many functions in the *41CL Update Functions* require memory page or sector addresses as arguments. These arguments are entered as hexadecimal (hex) values, using the ALPHA register. Valid hex digits are the numerals ***0*** - ***9*** and the letters ***A*** - ***F***. Any other character entered as a hex digit will result in a ***HEX ERR*** message when the function is executed. Leading zeros must be present for hex numbers.

A page address can be any three-digit hex number, but a sector address must have either a "0" or an "8" as the least-significant digit. If a function requires a sector address and an invalid address is used the function will abort with a ***SCT ERR*** error message.

Pairs of hex arguments are separated by the "***>***" character, and this delimiter must be present in the proper location or a ***FMT ERR*** message will result.

The figure below shows the formatting used when a single memory page address or sector address is required by a function.

	ALPHA register		
	3	2	1
	page/sector		
page address	***P5***	***P4***	***P3***
sector address	***S5***	***S4***	***S3***

The figure below shows the formatting used when a function requires both a source address and destination address pair or a begin address and end address pair. In the case of a begin address and end address pair, the begin address must be less than or equal to the end address, and an ***SSS>DDD*** error message will result if this is not the case.

	ALPHA register						
	7	6	5	4	3	2	1
	source				destination		
source and destination addresses	***S5***	***S4***	***S3***	***>***	***D5***	***D4***	***D3***
begin and end addresses	***B5***	***B4***	***B3***	***>***	***E5***	***E4***	***E3***

Several functions (**FDBCHK?**, **FLCHK?** and **FLUPD**) allow a combination of these formats, as shown in the figure below. Note that the "all of Flash memory" option for the **FLUPD** funtion automatically excludes the first sector of Flash memory (page addresses 0x000-0x007), where the operating system resides, unless the optional OS Update mode has been enabled.

	ALPHA register						
	7	6	5	4	3	2	1
	source				destination		
all of Flash memory							*
individual page					P5	P4	P3
begin and end addresses	B5	B4	B3	>	E5	E4	E3

Different functions have different restrictions on the page or sector addresses used as arguments, and these restrictions are shown in the header that describes the function. In the example below the source address can be in either Flash memory or RAM (the [F/R]) and the destination address is restricted to RAM (the [R]) only. Some functions allow only Flash memory (shown as [F]) addresses as arguments. A source address in the wrong type of memory will result in a ***SRC ERR*** error message, while a destination address in the wrong type of memory will result in a ***DST ERR*** message.

FUNCTION (source [F/R] and destination[R] page addresses in ALPHA register)

All of the functions in the *41CL Update Functions* are aware of the size of both the Flash memory and RAM on the 41CL where they are executed, and automatically check for valid address arguments. Any address outside the valid range will result in a ***RNG ERR*** error message.

All of the functions in the *41CL Update Functions* automatically protect the areas of both Flash memory and RAM that are used by the Operating System from being overwritten. For Flash memory this is pages 0x000 through 0x007, while for RAM it is pages 0x800 through 0x804. Any address in these areas, where the function can change the contents of memory, will result in a ***OS AREA*** error message. The only exception to this rule is for the two Flash memory update functions (**FLUPD** and **P8UPD**), and even then, only when the OS Update option has been explicitly enabled.

Do not attempt to erase (using **PGERASE** or **P8ERASE**) a page that is plugged in via the MMU, or a sector containing a page plugged in via the MMU. Doing so will result in an ***MMU ERR*** error message. This restriction does not apply to the two Flash memory update functions (**FLUPD** and **P8UPD**) even though they erase Flash memory.

Do not attempt to initialize (using **PGINI** or **P8INI**) a page in RAM that is plugged in via the MMU, or a sector in RAM that contains a page plugged in via the MMU. Doing so will result in an ***MMU ERR*** error message.

Operation Details

Central to the operation of the *41CL Update Functions* is a modified version of the Flash Database (FLDB, stored in Flash memory at page address 0x0DE), called the Correlated Flash Database (CFLDB.)

The original FLDB contains the two words of the expected CRC value for each page in Flash memory, and each page of Flash memory is allocated four words in the Flash Database. By default the first two words are used to hold the CRC value, and the remaining two words both contain 0xFFFF, as shown in the table below.

FLDB entry word	default contents
1	CRC LSWord
2	CRC MSWord
3	0xFFFF
4	0xFFFF

The FLDB and CFLDB require one page to hold all of the information for any version of the 41CL. The entries are organized as shown in the table below. Entries for pages not present in Flash memory are ignored by the *41CL Update Functions*, the *41CL Clone Functions* and the *clupdate* program, which makes the format compatible across all 41CL versions.

	word							
word address	0 or 8	1 or 9	2 or A	3 or B	4 or C	5 or D	6 or E	7 or F
0x000-0x007	page 0x000 CRC		0xFFFF	0xFFFF	page 0x200 CRC		0xFFFF	0xFFFF
0x008-0x00F	page 0x001 CRC		0xFFFF	0xFFFF	page 0x201 CRC		0xFFFF	0xFFFF
0x010-0x017	page 0x002 CRC		0xFFFF	0xFFFF	page 0x202 CRC		0xFFFF	0xFFFF
:	:		:	:	:		:	:
0xFE8-0xFEF	page 0x1FD CRC		0xFFFF	0xFFFF	page 0x3FD CRC		0xFFFF	0xFFFF
0xFF0-0xFF7	page 0x1FE CRC		0xFFFF	0xFFFF	page 0x3FE CRC		0xFFFF	0xFFFF
0xFF8-0xFFF	page 0x1FF CRC		0xFFFF	0xFFFF	page 0x3FF CRC		0xFFFF	0xFFFF

The CFLDB is stored in RAM at page address 0x806, and replaces the 0xFFFF words with values that mark pages as up-to-date, or not up-to-date. The default 0xFFFF words identify a page as unverified. The various cases are shown in the table below.

CFLDB words 3 & 4	meaning
0xFFFFFFFF	default, unverified page
0x00000000	page verified up-to-date
any other value	page needs updating

The CFLDB provides a way for both the software running on the host machine and the 41CL to know which pages in Flash memory need updating, and is normally created on the 41CL being updated. The CFLDB can be created automatically by the 41CL using the **FLCHK?** function, or manually using the **CDBINV** function to mark each invalid page that you want to update. It is also possible to create the CFLDB on the host machine and then merely download it the the 41CL for use during the update.

The *41CL Update Functions* utilize specific pages in RAM to hold the CFLDB and the Flash memory sector being updated, as shown in the table below. Make sure that you are not using these RAM pages before executing any *41CL Update Functions*, because any pre-existing data in these pages will be lost.

page	Usage	
0x806	Correlated Flash Database	
0x810	page 0 of a Flash memory sector	Sector Buffer
0x811	page 1 of a Flash memory sector	
0x812	page 2 of a Flash memory sector	
0x813	page 3 of a Flash memory sector	
0x814	page 4 of a Flash memory sector	
0x815	page 5 of a Flash memory sector	
0x816	page 6 of a Flash memory sector	
0x817	page 7 of a Flash memory sector	

The CFLDB only resides in RAM, and should not be manually written to Flash memory in place of the regular FLDB. The *41CL Update Functions* distinguish between the FLDB and the CFLDB, and will always properly update the FLDB in Flash memory if the appropriate sector is included in the range of pages to update.

A number of functions in the *41CL Update Functions* beep to signal progress or completion. These beeps can be disabled by clearing Flag 26. The table below lists the various cases:

function	tone	meaning
BFRCHK?	3	invalid CRC (each page)
FDBCHK?	7	valid CRC (each page)
FLCHK?	9	function complete
PGERASE **PGWR** **P8ERASE**	3	confirmation timeout
FLUPD	9	function complete
P8UPD	3	confirmation timeout
P8WR	9	function complete

Flash Status Functions

The Flash Status functions provide a way to check on the configuration of the Flash memory device on the 41CL board. Knowledge of the Flash configuration is required for proper manual programming of any special areas of the Flash memory.

Other Flash Status functions access the revision dates of the Flash Database (FLDB) and the Image Database (IMDB), which make it easier to check whether or not the Flash memory contents are up to date.

FDBVER?

Executing **FDBVER?** (*Flash Database Version?*) reads the revision date embedded in the Flash Database (FLDB) located in Flash memory (at page address 0x0DE.) This date is written to the ALPHA register in either DD/MM/YYYY or MM/DD/YYYY format, depending on the state of the DMY/MDY flag (flag 31). In Run mode the date is also returned in the display.

Early versions of the Flash Database did not include a revision date, in which case this function will return a ***DATE ERR*** message.

FLASH?

Executing **FLASH?** (*Read Flash Device Code*) issues the special control sequence that causes the Flash device to return the "device code," which uniquely identifies the organization of the Flash. The four-digit hexadecimal device code is written to the ALPHA register, and in Run mode the device code is also returned in the display.

The table below shows the device code values for the various Flash devices that have been used in the 41CL.

Version	device code	Flash device	Flash Size
V2 "top"	0x22C4	M29W160ET	1M x 16
V2 "bottom"	0x2249	M29W160EB	1M x 16
V3/V4 "top"	0x2256	M29W320ET	2M x 16
V3/V4 "bottom"	0x2257	M29W320EB	2M x 16
V5 "top"	0x227E	M29W640GT	4M x 16

Flash memory is erased by sectors, which are usually 64K bytes in size (32K words, or eight pages) for the 41CL. But either the top or the bottom sector of the Flash memory is special, in that it is erased in pieces smaller than the entire sector. You will need to take this into account when working with the top or bottom sector.

For V2 boards, the special sector is broken into four pieces: 8K words (two pages), 4K words (one page), 4K words (one page) and 16K words (four pages). The table below shows the page addresses for these two cases.

device code	8K	4K	4K	16K
0x22C4	0x0FE-0x0FF	0x0FD	0x0FC	0x0F8-0x0FB
0x2249	0x000-0x001	0x002	0x003	0x004-0x007

For V3 and V4 boards either the bottom or the top sector is broken into eight identical pieces of 4K words (one page). For V5 boards only devices with the top sector segmented have been used. The table below shows the page addresses for these cases.

device code	4K	4K	4K	4K	4K	4K	4K	4K
0x2256	0x1FF	0x1FE	0x1FD	0x1FC	0x1FB	0x1FA	0x1F9	0x1F8
0x2257	0x000	0x001	0x002	0x003	0x004	0x005	0x006	0x007
0x227E	0x3FF	0x3FE	0x3FD	0x3FC	0x3FB	0x3FA	0x3F9	0x3F8

IDBVER?

Executing **IDBVER?** (*Image Database Version?*) reads the revision date for the Image Database (IMDB) located at page address 0x0DF. This date is written to the ALPHA register in either DD/MM/YYYY or MM/DD/YYYY format, depending on the state of the DMY/MDY flag (flag 31). In Run mode this date is also returned in the display.

Early versions of the Image Database did not include a revision date, in which case this function will return a ***DATE ERR*** message.

Page Functions

The Page functions (a page is 4k words) are special versions of similar functions originally available in the *41CL Extra Functions*, and are included to make the *41CL Update Functions* completely stand-alone. All of these functions automatically execute at 50x Turbo mode, but the current Turbo mode is restored when the function completes.

The Page functions will be useful if you are doing manual or semi-automatic updates of the Flash memory. If you only plan to do automatic updates you can skip the remainder of this section.

PGBFR (source [F] page address in ALPHA register)

Executing **PGBFR** (*Copy Flash Memory Page to Sector Buffer*) copies one page from Flash memory to the appropriate page in the sector buffer starting at address 0x810.

In Run mode the function sends ***CPY***, plus the source page address, to the display during the actual transfer.

The table below shows the destination address within the sector buffer for the various source address possibilities.

source address	destination page
0xXX0 or 0xXX8	0x810
0xXX1 or 0xXX9	0x811
0xXX2 or 0xXXA	0x812
0xXX3 or 0xXXB	0x813
0xXX4 or 0xXXC	0x814
0xXX5 or 0xXXD	0x815
0xXX6 or 0xXXE	0x816
0xXX7 or 0xXXF	0x817

PGCPY (source [F/R] and destination[R] page addresses in ALPHA register)

Executing **PGCPY** (*Copy Page to RAM*) copies one page from Flash memory or RAM to another page in RAM.

In Run mode the function sends ***CPY***, plus the source page address, to the display during

the actual transfer.

This function can be used to copy pages to arbitrary RAM addresses. Pages cannot be copied to the OS area (pages 0x800 through 0x804) of RAM.

PGCRC (page [F/R] address in ALPHA register)

Executing **PGCRC** (*Calculate Page CRC*) calculates the CRC for a page in memory.

In Run mode the function sends ***CRC***, plus the page address, to the display during the calculation. The calculated CRC is written to the ALPHA register, and in Run mode the CRC is also returned in the display.

PGERASE (page [F] address in ALPHA register)

Executing **PGERASE** (*Erase Flash Memory Page*) erases the sector of Flash memory that contains the selected page. Because it will erase the entire sector containing the specified page, this function should only be used to erase the special sectors of Flash memory that are not the regular size.

In Run mode, the function requires verification, by sending ***ERS***, plus the page address, plus ***OK?*** to the display. Pressing the **R/S** key confirms that the erase should be performed. Pressing any other key will cancel the function. If no key is pressed within 10 seconds the function will be cancelled and a ***NULLED*** error message will be written to the display.

Erases of pages in the OS sector (0x000-0x007) of Flash memory are not allowed. Erases of pages currently plugged in via the MMU are not allowed, but this check is only done for the normal Flash sectors, not the topmost sector.

Never erase a special sector that contains a page that is plugged in via the MMU, as this will cause the machine to lock up.

PGINI (page [R] address in ALPHA register)

Executing **PGINI** (*Initialize Page in RAM*) initializes an entire page to 0xFFFF (the

unprogrammed Flash value) in every location. This function is useful to initialize an unused page in RAM before writing a sector to Flash memory.

In Run mode the function ***INI***, plus the page address, is written to the display during the initialization.

Initialization of pages in the OS area (0x800-0x804) of RAM is not allowed. Initialization of pages currently plugged in via the MMU is not allowed.

PGWR (source [R] and destination [F] page addresses in ALPHA register)

Executing **PGWR** (*Write Page to Flash Memory*) copies one page from RAM to another location in Flash memory.

In Run mode, the function requires verification, by sending ***WR***, plus the page address, plus ***OK?*** to the display. Pressing the **R/S** key confirms that the write should be performed. Pressing any other key will cancel the function. If no key is pressed within 10 seconds the function will be cancelled and a ***NULLED*** message will be written to the display.

Because of the unique way that Flash memory writes work (only a "1" can be changed to a "0") this function should only be used to write to pages in Flash memory that have been erased.

Writes to the OS sector (page addresses 0x000-0x007) of Flash memory are not allowed.

Flash Sector Functions

The Flash memory in the 41CL can only be erased by sectors, and each sector is normally eight pages. The functions here operate on entire sectors, and significantly reduce the number of commands that would be required if operating on pages. All of these functions require that any address be sector-aligned. In other words the least-significant digit of a sector address must be either 0 or 8.

The Flash Sector functions will be useful if you are doing manual or semi-automatic updates of the Flash memory. If you only plan to do automatic updates you can skip the remainder of this section.

BFRCHK? (sector [F] address in ALPHA register)

The **BFRCHK?** (*Check Sector Buffer*) function calculates the CRC values for the eight consecutive pages (an image of Flash memory) stored in the Sector Buffer, and compares these values to the corresponding entries in the CFLDB. This function is useful for verifying that all of the data in the Sector Buffer is valid before writing it to Flash memory. This function is not programmable.

In Run mode the function sends ***CRC***, plus the current page address in the Sector Buffer, to the display during the CRC calculation. The 41CL beeps at the end of each CRC calculation. Tone 7 is signalled for a valid CRC and Tone 3 is signalled for an invalid CRC. When all of the CRC calculations have finished a Tone 9 is signalled.

In Run mode this function continues to completion unless halted with the **R/S** key. The function can be stopped and restarted using the **R/S** key. The function is cancelled (while stopped) using the backspace key. The function only samples the **R/S** key in the case of an invalid CRC, after appending ***BAD*** to the ***CRC*** and page address in the display. In the case of an invalid CRC the function waits for 3 seconds for a keypress before continuing.

P8BFR (source [F] sector address in ALPHA register)

Executing **P8BFR** (*Copy Flash Memory Sector to Sector Buffer*) copies eight contiguous pages of Flash memory to the Sector Buffer starting at address 0x810.

In Run mode the function sends ***CPY***, plus the current source page address, to the display

during the actual transfers.

P8CPY (source [F/R] and destination [R] sector addresses in ALPHA register)

Executing **P8CPY** (*Copy Sector to RAM*) copies the contents of eight contiguous pages of Flash memory or RAM to another eight contiguous pages of RAM.

In Run mode the function sends ***CPY***, plus the current source page address, to the display during the actual transfers.

Pages cannot be copied to the OS area (pages 0x000 through 0x005) of RAM.

P8ERASE (sector [F] address in ALPHA register)

Executing **P8ERASE** (*Erase Flash Memory Sector*) erases an entire sector of Flash memory.

In Run mode, the function requires verification, by sending ***ERS***, plus the sector address, plus ***OK?*** to the display. Pressing the **R/S** key confirms that the erase should be performed. Pressing any other key will cancel the function. If no key is pressed within 10 seconds the function will be cancelled and a ***NULLED*** message will be written to the display.

The **P8ERASE** function always assumes that the top sector is segmented, and does eight erase operations on the eight individual pages for that sector. This means that the erase of the top sector takes eight times as long as normal to execute.

An erase of the OS sector (at address 0x000) of Flash memory is not allowed. An erase of a sector containing a page that is currently plugged in via the MMU is not allowed.

P8INI (sector [R] address in ALPHA register)

Executing **P8INI** (*Initialize Sector in RAM*) initializes eight pages (the equivalent of a Flash memory sector) to 0xFFFF (the unprogrammed Flash value) in every location.

In Run mode the function sends ***INI***, plus the current page address, to the display during

the initializations.

This function will be useful to initialize the sector buffer in the case where a sector in Flash memory to be updated is mostly empty. The sector buffer never needs to be initialized before executing **P8BFR**, because that function writes all eight pages of the sector buffer.

Pages in the OS area (pages 0x800 through 0x804) of RAM cannot be initialized. Pages currently plugged in via the MMU cannot be initialized.

P8UPD (sector [F] address in ALPHA register)

Executing **P8UPD** (*Update Sector in Flash Memory*) updates the selected sector in Flash memory with the contents of the Sector Buffer in pages 0x810-0x817. The selected sector is erased prior to the writes. This function does not affect the CFLDB in any way.

In Run mode, the function requires confirmation, by sending ***ERS***, plus the sector address, plus ***OK?*** to the display. Pressing the **R/S** key confirms that the erase and writes should be performed. Pressing any other key will cancel the function. If no key is pressed within 10 seconds the function will be cancelled and a ***NULLED*** message will be written to the display.

The **P8UPD** function always assumes that the top sector is segmented, and does eight erase operations on the eight individual pages for that sector, which means that the erase will takes eight times as long as normal to execute. In Run mode the function sends an ***ERS*** message, plus the sector address, to the display during the entire erase and then a ***WR*** message, plus the current page address, is written to the display during the writes. The function beeps with tone 9 to signal overall completion.

The **P8UPD** function is unique in that it can be used on a sector containing an image that is currently plugged in via the MMU. This is possible because the entire function is copied to RAM before execution, and the function never returns to the OS until the update is complete.

By default, updates of the OS sector (at address 0x000) of Flash memory are not allowed. However, if the OS Sector Update mode has been enabled (via the **OSUPDT** function) this function can be used to update the OS sector of Flash memory. Be absolutely sure that the Sector Buffer contains the correct data to write before executing the **P8UPD** function on the OS sector. This function assumes that the OS sector is segmented, so that erase is done once per page, and will require about eight times as long as normal to complete.

P8WR (source [R] and destination [F] sector addresses in ALPHA register)

Executing **P8WR** (*Write Sector to Flash memory*) copies eight contiguous pages in RAM to another eight contiguous pages in Flash memory.

In Run mode, the function requires confirmation, by sending ***WR***, plus the page address, plus ***OK?*** to the display. Pressing the **R/S** key confirms that the write should be performed. Pressing any other key will cancel the function. If no key is pressed within 10 seconds the function will be cancelled and a ***TIMEOUT*** message is written to the display.

The function beeps with tone 9 to signal completion.

Because of the unique way that Flash memory writes work (only a "1" can be changed to a "0") this function should only be used to write to pages in Flash memory that have been erased.

Writes to the OS sector (at address 0x000) of Flash memory are not allowed, even if the OS Sector Update mode has been selected. This is because for the OS sector the entire erase-write operation must be atomic (uninterrupted by a return to the OS). Use the **P8UPD** function if you really need to manually update the OS sector of Flash memory.

Correlated Flash Database Functions

The Flash Database (FLDB) was originally introduced to the 41CL to help with keeping track of the contents of Flash memory. The FLDB contains the CRC value for each page in Flash memory, which allows users to verify that the contents of Flash memory on their 41CL are current.

The Correlated Flash Database (CFLDB), which is a modified version of the FLDB, is central to the operation of the *41CL Update Functions*. In addition to the expected CRC value for each page in Flash memory, the CFLDB contains information about whether or not each page actually has the correct CRC. This status information consists of one of three choices: *unverified* (not checked yet), *valid* (calculated CRC matches expected CRC) or *invalid* (calculated CRC does not match expected CRC).

The base CFLDB has all pages marked as *unverified*. The base CFLDB is normally downloaded from a host computer, and then functions are executed on the 41CL to populate the CFLDB with the status of pages. This status can be generated either automatically or manually, and once complete, the CFLDB can be transferred back to the host computer for analysis.

The primary use of the CFLDB is by the 41CL itself, to control the update process. Functions in the *41CL Update Functions* use the status in the CFLDB to request the latest version of pages from the host computer, assembling the pages in a dedicated buffer in RAM for writing to the Flash memory. Because Flash memory is only erased by sectors (eight pages) this process is done one sector at a time until the entire selected area of Flash memory has been updated. This update process can be done on as little as one page or as much as the entire Flash memory.

The CFLDB is always located in RAM at page address 0x806 and is marked as valid, using an internal status bit, when it is initially loaded. This CFLDB status is checked by functions that use the CFLDB. Once the update operation is complete the CFLDB can be marked as invalid. This provides some protection against erroneous update operations. The MEMORY LOST condition marks the CFLDB as invalid, but the CFLDB valid status is preserved when the 41CL is turned off.

The Correlated Flash Database functions will be useful if you are doing manual or semi-automatic updates of the Flash memory. If you only plan to do automatic updates you can skip the remainder of this section.

CDBDEL

Executing **CDBDEL** (*Delete Correlated Flash Database*) deletes the Correlated Flash

Database by writing 0xFFFF to every location in page 0x806 and then marking the CFLDB as invalid.

In Run mode the function sends ***INI 806*** to the display during the writes that clear page 0x806.

Overwriting the CFLDB guarantees that you will never be using a stale version.

CDBVER?

Executing **CDBVER?** (*Correlated Flash Database Version?*) reads the revision date embedded in the Correlated Flash Database, This date is written to the ALPHA register in either DD/MM/YYYY or MM/DD/YYYY format, depending on the state of the DMY/MDY flag (flag 31). In Run mode the date is also returned in the display.

FDBCHK? (begin [F] and end [F] page addresses in ALPHA register)

The **FDBCHK?** (*Check Flash Database*) function calculates the CRC value for a page and compares this value to the corresponding entry in the CFLDB. If the CRC values agree the page of Flash memory is up-to-date and 0x00000000 is written to the spare two words in the corresponding entry in the CFLDB. If the two CRC values are different the calculated CRC value is written to the spare two words. This function is not programmable.

This status word provides a way for both the host software and the 41CL to know which pages of Flash need updating, because 0x00000000 is never a valid CRC value. In a similar fashion, the default value of 0xFFFFFFFF that is normally in the spare two words of a CFLDB entry can be used to identify those pages that were not checked by the **FDBCHK?** function. The value 0xFFFFFFFF is a valid CRC, but no known Flash contents have this CRC value.

This process continues until all of the selected pages have been checked. This function may take a significant of time to complete, depending on the range of pages being checked. See the "Batteries and Execution Time" section for details.

In Run mode the function sends ***CRC***, plus the current page address, to the display during the CRC calculation. The 41CL beeps at the end of each CRC calculation. Tone 7 is signalled for a valid CRC and Tone 3 is signalled for an invalid CRC. When all of the CRC calculations have finished a Tone 9 is signalled. In addition to each tone, the 41CL

sends an "Y" (0x59) over the serial port to keep the host machine from going to sleep.

This function continues to completion unless halted with the **R/S** key. The function can be stopped and restarted using the **R/S** key. The function is cancelled using the backspace key (while stopped), but this means that the CFLDB will not be complete for the entire selected address range. If the function is cancelled or aborted (because of a low battery condition) the ALPHA register is updated to reflect the pages remaining to be checked.

The page in Flash memory at address 0x0DE is used to hold the FLDB for all board versions. Since the FLDB cannot hold an entry containing the CRC value for itself, the slot that would normally hold this CRC is instead used to hold the revision date of the FLDB. If the range for the **FDBCHK?** function includes the address of the FLDB (0x0DE) this function merely compares the date stored in the FLDB with the corresponding date in the CFLDB, and marks the page as either *valid* or *invalid* in the CLFDB with the result of this comparison.

FDB2CDB

Executing **FDB2CDB** (*Make Correlated Flash Database from Flash FLDB*) copies the FLDB from Flash memory to page 0x806 in RAM, creating a basis for the CFLDB.

In Run mode the function ***CPY 0DE*** to the display during the actual transfer and then sets the internal CFLDB status to valid.

This function can be used to initialize the CFLDB if you know that the FLDB in Flash memory is mostly current or when you only need to manually update a few pages in Flash memory. But it is almost always better to download the current FLDB from the host machine to use as the CFLDB.

The FLDB must be in its default state before it can be used as a basis for the CFLDB. If you have modified the FLDB in Flash memory using functions in the *41CL Memory Functions*, you must execute **FDBGC** (also in the *41CL Memory Functions*) to clean up the FLDB before attempting to use it as a basis for the CFLDB.

PGCDB? (page [F] address in ALPHA register)

Executing **PGCDB?** (*Fetch Expected CRC from Correlated Flash Database*) retrieves the expected CRC from the Correlated Flash Database for a page address in Flash memory. The CRC is written to the ALPHA register, and in Run mode the CRC is also

returned in the display.

PGINV (page [F] address in ALPHA register)

Executing **PGINV** (*Mark Page as Invalid in Correlated Flash Database*) marks the selected page as invalid in the Correlated Flash Database, and then increments the page address in the ALPHA register in preparation for another operation to mark the next page.

This function can be used in place of the **FLCHK?** function if there are only a few pages to update and you know their page addresses. The function marks the pages as *invalid* by writing 0x0101 to both of the words that indicate status in the CFLDB.

PGSTA? (page [F] address in ALPHA register)

Executing **PGSTA?** (*Fetch Page Status from Correlated Flash Database*) retrieves the page status from the Correlated Flash Database for a page address in Flash memory. The page status is written to the ALPHA register, and in Run mode the page status is also returned in the display.

The **PGSTA?** function allows you to see exactly what page status is contained in the CFLDB. The various cases are shown in the table below.

CFLDB page status	meaning
0xFFFFFFFF	default, unverified
0x00000000	verified up-to-date
any other value	needs updating

PGUNV (page [F] address in ALPHA register)

Executing **PGUNV** (*Mark Page as Unverified in Correlated Flash Database*) marks the selected page as unverified in the Correlated Flash Database, and then increments the page address in the ALPHA register in preparation for another operation to mark the next page.

This function marks the pages as *unverified* by writing 0xFFFF to both of the words that indicate status in the CFLDB.

PGVAL (page [F] address in ALPHA register)

Executing **PGVAL** (*Mark Page as Valid in Correlated Flash Database*) marks the selected page as valid in the Correlated Flash Database, and then increments the page address in the ALPHA register in preparation for another operation to mark the next page.

This function can be used in place of the **FLCHK?** function if you know that certain pages are valid and you know their page addresses. The function marks the pages as *valid* by writing 0x0000 to both of the words that indicate status in the CFLDB. Since the update functions treat *unverified* status and *valid* status the same, there is not really a need to mark pages with *valid* status.

Communication Functions

The communication functions allow you to communicate with software running on a host machine and perform the Flash updates manually or semi-automatically. These functions are used by the automatic update functions described in the next section.

All communications functions use 4800 baud for the serial data rate, and all functions that expect a response from the host machine include a 30-second timeout. If no response is received from the host machine before this timeout expires, the function terminates with a ***TIMEOUT*** error message.

The communications channel is automatically marked as closed when the 41CL is turned off and by the ***MEMORY LOST*** condition.

Only the **CMOPEN** (*Open Communication Channel*) and **CMCLOSE** (*Close Communication Channel*) functions are required if you are doing automatic updates of the Flash memory. If you only plan to do automatic updates you can skip the remainder of this section.

CDBEXP

Executing **CDBEXP** (*Export Correlated Flash Database*) sends either a "K" (0x4B, for a 1Mx16 device), an "M" (0x4D, for a 2Mx16 device), or an "O" (0x4F, for a 4Mx16 device), over the serial port and waits for an "L" (0x4C), an "N" (0x4E) or a "P" (0x50) in response. Once the response is received the function sends the entire CFLDB over the serial port. In Run mode an ***EXP 0DE*** message is written to the display during the actual transfer.

If the communications channel is not open this function merely sends the entire CFLDB over the serial port.

CDBIMP

Executing **CDBIMP** (*Import Correlated Flash Database*) sends a "C" (0x43, for a 1Mx16 device), an "E" (0x45, for a 2Mx16 device), or a "G" (0x47, for a 4Mx16 device) over the serial port and expects the host software to return either the latest Flash Database or the Correlated Flash Database in response. The function stores the returned page in the *41CL Memory Buffer* (page 0x806 in RAM) and sets the internal CFLDB status to valid. In Run mode an ***IMP 0DE*** message is written to the display during the actual transfer.

If the communications channel is not open this function merely awaits reception of the entire CFLDB over the serial port.

Once the latest Flash Database is returned, the **FDBCHK?** function can be used to update status in the CFLDB on the 41CL. The software on the host can then use the CFLDB to verify that all of the Flash pages that require updating are available for the host software to download.

CMCLOSE

Executing **CMCLOSE** (*Close Communication Channel*) sends an "W" (0x57) over the serial port and waits for a "X" (0x58) in response. The software running on the host machine should terminate after sending this response to the 41CL.

If the communications channel is not open this function does nothing.

CMOPEN

Executing **CMOPEN** (*Open Communication Channel*) initializes the 41CL serial port, sends an "A" (0x41) over the serial port and waits for a "B" (0x42) in response. Once the response is sent the software running on the host machine should be prepared to accept messages from the 41CL.

This function operates independently of whether or not the communications channel is already open.

PGEXP (source [F/R] and destination [F] page addresses in ALPHA register) OR (source [F/R] page address in ALPHA register)

Executing **PGEXP** (*Export Page*) sends a "U" (0x55) followed by the three-digit destination page address (in ***D3***, ***D4***, ***D5*** order) over the serial port and waits for a "V" (0x56) in response. Once the response is received the function sends the entire page at the source address over the serial port.

If the communications channel is open the function expects both a source address and destination address in the ALPHA register. If the communications channel is not open the function expects only a source address in the ALPHA register, and sends the page without any handshake with the host machine.

In Run mode the function sends ***EXP***, plus the current source page address, to the display during the transfer.

PGIMP	**(source [F] and destination [R] page addresses in ALPHA register)** **OR** **(destination [R] page address in ALPHA register)**

Executing **PGIMP** (*Import Page*) sends a "S" (0x53) followed by the three-digit source page address (in ***S3***, ***S4***, ***S5*** order) over the serial port and expects the host software to return the latest version for that page of Flash memory. This function will signal an error if the entire 4k words of the page are not received.

If the communications channel is open the function expects both a source address and destination address in the ALPHA register. If the communications channel is not open the function expects only a destination address in the ALPHA register, and expects to receive the page without any handshake with the host machine.

In Run mode the function sends ***IMP***, plus the current destination page address, to the display during the transfer.

Auto-update Functions

The Auto-update functions automate the 41CL update process as much as possible, but the process still involves several steps. First, the communication channel must be opened, and the pages in Flash memory that need updating must be identified, for both the 41CL and the software running on the host computer. Once these pages have been identified the actual update process can begin. After the update process is complete the communication channel should be closed.

Both of these functions automatically execute at 50x Turbo mode, but the current Turbo mode is restored when the function completes.

Example 1, shown later in this document, illustrates the automatic update process.

FLCHK?	**(begin [F] and end [F] page addresses in ALPHA register)** **OR** **(page [F] address in ALPHA register)** **OR** **(* in ALPHA register)**

Executing **FLCHK?** (*Check Flash Memory Against Flash Database*) checks the selected pages against the information in the CFLDB. First the CFLDB is downloaded, then the checks are done, and finally the CFLDB is uploaded to the host computer. This function is not programmable.

This function requires that the communication channel be open. The function first sends a CFLDB import request over the serial port and expects the host software to return either the latest Flash Database or the Correlated Flash Database in response. The returned page is stored in the *41CL Memory Buffer* (page 0x806 in RAM) and the internal CFLDB status is set to valid. An ***IMP ODE*** message is written to the display during the actual transfer.

The function calculates the CRC value for a page, compares this value to the corresponding entry in the CFLDB, and the page is marked as either *valid* or *invalid* in the CFLDB. The function sends ***CRC***, plus the current page address, to the display during this CRC calculation. The 41CL beeps at the end of each CRC calculation. Tone 7 is signalled for *valid* and Tone 3 is signalled for *invalid*. This process continues until all of the selected pages have been checked. When all of the CRC calculations have finished a Tone 9 is signalled.

At this point the function sends a CFLDB export request over the serial port, followed by

the entire CFLDB. An ***EXP 0DE*** message is written to the display during this transfer.

This function continues to completion unless halted with the **R/S** key. The function can be stopped and restarted using the **R/S** key during the CRC calculation phase. The function is cancelled using the backspace key (while stopped), but this means that the creation of the CFLDB may not be complete. In this case the CFLDB will still be uploaded to the host machine. If the function is cancelled or aborted (because of a low battery condition) the ALPHA register is updated to reflect the pages remaining to be checked.

FLUPD	**(begin [F] and end [F] page addresses in ALPHA register)** **OR** **(page[F] address in ALPHA register)** **OR** **(* in ALPHA register)**

Executing **FLUPD** (*Update Flash Memory*) updates the selected pages, by stepping through the corresponding entries in the CFLDB to determine which pages need updating. If a CFLDB entry indicates *invalid*, the sector containing the page is first copied to RAM and then the invalid pages are requested from the host machine. Once all of the pages in the sector are current the sector is erased and the new sector information is written to the Flash memory. Then all of the pages in that sector are marked as *unverified* in the CFLDB. This function is not programmable.

This function requires that the communication channel be open. When a page marked as *invalid* is found in the CFLDB the corresponding sector in Flash memory is first copied to the sector buffer. During these copies a ***CPY*** message, plus the current source page address, is sent to the display.

Next, for each page in the sector marked as *invalid*, a page request is sent over the serial port and the host software returns the latest version for that page of Flash memory. An ***IMP*** message, plus the current destination page address (in the sector buffer), is sent to the display during the transfer. After the transfer is complete, if the Auto-verify mode has been selected, the CRC for the downloaded page (in the sector buffer) is checked before continuing.

Once all of the pages in the sector marked as *invalid* have been downloaded the sector in Flash memory is erased. Then the eight pages from the sector buffer are written to the Flash memory. During these writes a ***WR*** message, plus the current page address, is written to the display. After these eight pages are written to Flah memory, all eight pages are marked as *unverified* in the CFLDB.

This process continues until the entire selected range of Flash memory has been updated, and the function beeps with tone 9 to signal overall completion, and an ***UPD DONE*** message is written to the display.

Only those sectors containing the selected pages are update candidates, and among the candidates, only those marked has having *invalid* status in the CFLDB are actually downloaded and replaced. When a page within a sector is included in the range of updates, the entire sector will be considered for update by this function, because the CFLDB for every page within a sector is checked to see if it requires updating.

Downloaded pages are written directly to the appropriate page in the sector buffer in RAM before the Flash sector is erased and then written. The Flash sector will not be erased unless the necessary pages have been successfully downloaded.

The pages downloaded are not normally verified before being written to Flash memory. However, for the OS sector (0x000-0x007), the CRC for each downloaded page always is verified against the CFLDB before the sector is erased and written. This is because the OS sector is so critical to the operation of the 41CL. If the optional Auto-verification mode is selected (via the **AUTOVFY** function) every downloaded page (except for the FLDB at page 0x0DE) is verified before being written to Flash memory.

This function continues until halted with the **R/S** key. The update process can be stopped and restarted using the **R/S** key. The function is cancelled using the backspace key, but this means that not all of the selected sectors will have been updated. If the function is cancelled or aborted (because of a low battery condition) the ALPHA register is updated to reflect the pages remaining to be checked.

If the function completes normally, but the 41CL automatically turns of after being idle for 10 minutes, the ***UPD DONE*** message will be lost. You can still determine that the function completed though, because the ALPHA register is unchanged in the case of normal completion.

Update Protocol

The diagram below shows the update process, with the 41CL functions, the messages on the serial line (for the 2Mx16 case), and the state of the software on the host machine.

This update protocol is used when the host computer is a PC running the *clupdate* program with the *--update* option, or when the host computer is another 41CL executing the **CLONE** function. In this protocol the 41CL being updated is always the master, and all communication is initiated by the master. The host computer, whether a PC or another 41CL, is a slave to the 41CL being updated.

<table>
<tr><th colspan="2"></th><th>auto</th><th>manual</th><th>serial comm</th><th>host SW</th></tr>
<tr><td colspan="2" rowspan="2"></td><td rowspan="2">CMOPEN</td><td rowspan="2">CMOPEN</td><td rowspan="2">----- (0x41) ----->
<----- (0x42) -----</td><td>start</td></tr>
<tr><td>send_ack</td></tr>
<tr><td colspan="2" rowspan="5"></td><td rowspan="5">FLCHK?</td><td rowspan="2">CDBIMP</td><td rowspan="2">----- (0x45) ----->
<----- (FLDB) -----</td><td>idle</td></tr>
<tr><td>send_fldb</td></tr>
<tr><td>FDBCHK?</td><td>----- (0x59) ----->
</td><td rowspan="2">idle</td></tr>
<tr><td rowspan="2">CDBEXP</td><td rowspan="2">----- (0x4D) ----->
<----- (0x4E) -----
----- (CFLDB) -----></td></tr>
<tr><td>send_ack</td></tr>
<tr><td colspan="2"></td><td></td><td></td><td></td><td>recv_cfldb</td></tr>
<tr><td rowspan="4">loop</td><td rowspan="4">loop</td><td rowspan="4">FLUPD</td><td>P8CPY</td><td></td><td rowspan="2">idle</td></tr>
<tr><td>PGIMP</td><td rowspan="2">-- (0x53 + page no.) -->
<----- (page) -----</td></tr>
<tr><td>send_page</td></tr>
<tr><td>P8UPD</td><td></td><td rowspan="2">idle</td></tr>
<tr><td colspan="2" rowspan="3"></td><td rowspan="3">CMCLOSE</td><td rowspan="3">CMCLOSE</td><td>----- (0x57) ----->
</td></tr>
<tr><td><----- (0x58) -----</td><td>send_ack</td></tr>
<tr><td></td><td>done</td></tr>
</table>

The table below shows all possible messages and responses. Where a response byte is required, it is the transmitted byte incremented by one. An incorrect response generates a ***COM ERR*** error message.

41CL Byte(s) sent	Response	Action by host machine
"A" (0x41)	**"B" (0x42)**	**Open Communication channel**
"C" (0x43)	**CFLDB page**	**Send 1Mx16 CFLDB**
"E" (0x45)	**CFLDB page**	**Send 2Mx16 CFLDB**
"G" (0x47)	**CFLDB page**	**Send 4Mx16 CFLDB**
"K" (0x4B)	**"L" (0x4C)**	**Receive 1Mx16 CFLDB**
"M" (0x4D)	**"N" (0x4E)**	**Receive 2Mx16 CFLDB**
"O" (0x4F)	**"P" (0x50)**	**Receive 4Mx16 CFLDB**
"S" (0x53) + page no.	**page**	**Send specified page**
"U" (0x55) + page no.	**"V" (0x56)**	**Receive specified page**
"W" (0x57)	**"X" (0x58)**	**Close Communication Channel**
"Y" (0x59)	**none**	**stay awake**

Page and page number transfers use the normal little-endian byte order used by all other 41CL serial block transfer functions. Page number transfers are deliberately slowed down, by inserting a 20mS delay between bytes, to allow the *41CL Clone Functions* time to process the bytes. This delay should be imperceptible to the user.

Mode Control Functions

The Mode Control functions allow the user to override the default operation of some of the *41CL Update Functions*. The default operation will almost always be the most appropriate for the majority of 41CL users.

OSPROT

Executing **OSPROT** (*Protect OS Sector*) disables the OS Update mode. In this case, which is the default, the OS Sector of Flash memory (page addresses 0x000-0x007) cannot be erased or written by any of the functions in the *41CL Update Functions*.

The OS Update mode is automatically disabled, protecting the OS Sector of Flash memory, by the ***MEMORY LOST*** condition, as well as every time the 41CL is turned off.

OSUPDT

Executing **OSUPDT** (*Enable OS Sector Update*) enables the OS Update mode. In this mode the **P8UPD** function and **FLUPD** function are allowed to operate on the OS Sector of Flash memory. Only these two functions are affected by the OS Update mode because only these functions complete the operation on a sector without returning to the Operating System.

AUTOVFY

Executing **AUTOVFY** (*Enable Auto-verification*) enables the Auto-verification mode. In this mode every page downloaded (except for the FLDB itself) by the **FLUPD** function is verified against the CFLDB before another page is fetched or a sector is written to Flash memory. The communications channel is normally very reliable, which is why most pages are not automatically verified.

The FLDB page cannot be verified because there is no way to include the CRC for the FLDB page within the FLDB itself.

This mode adds the normal CRC calculation time (about 10 seconds) to each page downloaded. In the case of a CRC error in the downloaded page the **FLUPD** function aborts with an error, rather than retrying the download.

NOVFY

Executing **NOVFY** (*Disable Auto-verification*) disables the Auto-verification mode. In this case, which is the default, only pages in the OS Sector (page addresses 0x000-0x007) are verified against the CFLDB before being written to Flash memory.

The Auto-verification mode is automatically disabled by the ***MEMORY LOST*** condition, as well as every time the 41CL is turned off.

Error Conditions

All functions in the *41CL Update Functions* that require an argument are capable of generating multiple errors, but only the first error found will be reported. Functions that need one or two page addresses check for errors in the following order: format (two-address functions only), valid hexadecimal, address(es) within range, ending address greater than starting address, valid sector (sector functions only), destination is in RAM (if appropriate) or Flash (if appropriate), and source is in RAM (if appropriate) or Flash (if appropriate). Functions requiring only one page address treat that address as a destination as far as error messages are concerned.

Arguments are always checked before any communication takes place, so the communications-related errors can only be generated if the arguments are deemed correct. The timeout limit of 30 seconds does not roll over from one byte to the next, but is restarted for each byte.

All error conditions cause a function to abort. Those functions that operate repetitively (**FDBCHK?**, **FLCHK?**, and **FLUPD**) will update the beginning address in the ALPHA register in the case of an error, to indicate how much progress was made towards completion.

The table below lists all possible error messages returned by the *41CL Update Functions*, along with the meaning of the message.

Error Message	Function	Meaning
CDB ERR	**BFRCHK?** **CDBEXP** **CDBVER?** **FDBCHK?** **FLUPD** **PGCDB?** **PGFDB?** **PGINV** **PGUNV** **PGVAL**	Invalid CFLDB
COM ERR	All communication functions requiring a response	Incorrect response from host (if comm channel is open)
	FLCHK? **FLUPD**	Comm channel is closed

DATE ERR	**CDBVER?** **FDBVER?** **IDBVER?**	Invalid revision date
DST ERR	All functions requiring an address	Invalid destination/end address
FMT ERR	All functions accepting an address pair	Format error in address pair (missing ">")
HEX ERR	All functions using hex	Invalid hexadecimal digit
OS AREA	**PGCPY** **PGIMP** **PGINI** **P8CPY** **P8INI**	Attempted operation on Operating System area of RAM
	PGERASE **PGWR** **P8ERASE** **P8UPD** **P8WR**	Attempted operation on Operating System area of Flash
MMU ERR	**PGERASE** **P8ERASE**	Attempted erase operation on a page plugged in via the MMU
NULLED	Erase and write functions requiring confirmation	No keypress in 10 seconds (function aborted)
OVR ERR	Serial receive functions	Receiver overrun
RNG ERR	All functions requiring an address	Address is outside of valid Flash or RAM address range
SCT ERR	**BFRCHK?** **P8CPY** **P8ERASE** **P8INI** **P8UPD** **P8WR**	Address not sector-aligned
SRC ERR	**FDBCHK?** **FLUPD** **PGIMP** **PGWR** **P8WR**	Invalid source/start address
SSS>DDD	**FLCHK?** **FLUPD**	Start address greater than Finish address
TIMEOUT	All communication functions requiring a response	No response from host in 30 seconds

TX ERR	Serial transmit functions	Transmit error

Examples

There is considerable flexibility built into the *41CL Update Functions*. This section shows the functions necessary to update the 41CL in a number of different ways. Most of these examples assume that the 41CL is connected to a host computer running the *clupdate* software.

EXAMPLE 1

The most time-consuming, but complete, option is to use the automatic update functions over the entire Flash memory address range. This will update all pages that are out-of-date.

XEQ "CMOPEN"	Open the communication channel with the host machine.
"*"	Specify all of memory. This selection will remain in effect for both of the auto-update functions.
XEQ "FLCHK?"	Download the CFLDB, verify all of memory, and then upload the updated CFDLB.
XEQ "FLUPD"	Update all Flash sectors with stale contents.
XEQ "CMCLOSE"	Close the communication channel with the host machine.
XEQ "CDBDEL"	Clear page 0x806 in RAM and mark the CFLDB as invalid.

EXAMPLE 2

In this example, there has been a single update since the Flash memory was last updated, to the *Library-4* image at page address 0x120. Rather than doing it manually, do a restricted scan and automatic update. A number of known-good sectors will be scanned, but the operation can proceed unattended.

XEQ "CMOPEN"	Open the communication channel with the host machine.
"0D8>127"	Specify a range of sectors covering the pages that are known to be out-of-date. This means the FLDB at 0x0DE and Library-4 at 0x120. This selection will remain in effect for both of the auto-update functions.
XEQ "FLCHK?"	Download the CFLDB, verify the specified area of memory, and then upload the updated CFDLB.
XEQ "FLUPD"	Update the Flash sectors within the specified area that contain stale contents.
XEQ "CMCLOSE"	Close the communication channel with the host machine.
XEQ "CDBDEL"	Clear page 0x806 in RAM and mark the CFLDB as invalid

Note that you can save some time by not scanning the known-good pages in the two affected sectors. Instead of specifying "0D8>127" for the address range, you can just as easily specify "0DE>120" for the check and update. The **FLCHK?** function will save some time by not scanning the pages at 0x0D8-0x0DD and 0x121-0x127. The **FLUPD** function automatically converts the range of page addresses to sector addresses covering the affected sectors.

EXAMPLE 3

In this example, there has been a single update since the Flash memory was last updated, to the *Library-4* image. In this case is may be quicker to manually mark the two pages to be updated (don't forget that the FLDB will need updating) rather than doing it automatically.

XEQ "FDB2CBD"	Copy the FLDB to the CFLDB and mark the CFLDB as valid.
"0DE"	FLDB address.
XEQ "PGINV"	Indicated that FLDB needs updating.
"120"	Library-4 address.
XEQ "PGINV"	Indicate that 4LIB needs updating.
XEQ "CMOPEN"	Open the communication channel with the host machine.
"120"	Library-4 sector address.
XEQ "FLUPD"	Update sector starting at 0x120.
"0DE"	Flash Database page address.
XEQ "FLUPD"	Update sector starting at 0x0D8.
XEQ "CMCLOSE"	Close the communication channel with the host machine.
XEQ "CDBDEL"	Clear page 0x806 in RAM and mark the CFLDB as invalid.

Note that the two updates can be done in either order. It is safest to specify the exact page being updated for the **FLUPD** function, but the function will always convert the page address to the appropriate sector address internally

EXAMPLE 4

In this example, there has been a single update since the Flash memory was last updated, to the *41CL Library Functions* image. Rather than doing it manually, do a pair of restricted scans and then an automatic update. The scan operations will be quicker, because the two images are quite a ways apart in memory, but it takes about the same amount of work as manually marking the two pages as out-of-date.

XEQ "CMOPEN"	Open the communication channel with the host machine.
"0DE"	Specify the page containing FLDB.
XEQ "FLCHK?"	Download the CFLDB, verify the specified area of memory, and then upload the updated CFDLB.
"00B"	Specify the page containing YLIB.
XEQ "FDBCHK?"	Check the page containing YLIB.
"*"	Specify all of Flash.
XEQ "FLUPD"	Update the Flash sector containing out-of-date images.
XEQ "CMCLOSE"	Close the communication channel with the host machine.
XEQ "CDBDEL"	Clear page 0x806 in RAM and mark the CFLDB as invalid

Note that the CFLDB returned to the host computer by the **FLCHK?** function will not have the page containing YLIB marked as out-of-date, because that page hasn't been checked yet. This is okay, because only the 41CL being updated needs to know that YLIB needs updating, and the **FDBCHK?** function marked page 00B as out-of-date. The 41CL being updated will request the page be downloaded and updated during the **FLUPD** function execution.

EXAMPLE 5

In this example, there has been a single update since the Flash memory was last updated, to the *41CL Library Functions* image. Do a manual update. In this case the various commands to send a page from the host machine to the 41CL will need to be executed on the host machine. This is very similar to the old method of updating the Flash memory, but utilizing the streamlined commands from the *41CL Update Functions*.

"0D8"	Specify the sector containing FLDB.
XEQ "P8BFR"	Copy this sector to the Sector Buffer.
"816"	Specify the page within the Sector Buffer to be replaced.
XEQ "PGIMP"	Download FLDB. Manually specify the image to download on the host machine.
"0D8"	Specify the sector to be updated.
XEQ "P8UPD"	Do the update.
XEQ "FDB2CDB"	Make the CFLDB from the just-downloaded FLDB.
"008"	Specify the sector containing YLIB.
XEQ "P8BFR"	Copy this sector to the Sector Buffer.
"813"	Specify the page within the Sector Buffer to be replaced.
XEQ "PGIMP"	Download YLIB. Manually specify the image to download on the host machine.
"008"	Specify the sector to be updated.
XEQ "P8UPD"	Do the update.
"008>00F"	Specify the pages to be verified.
XEQ "FDBCHK?"	Verify that the sector containing YLIB is correct.
"0D8>0DF"	Specify the pages to be verified.
XEQ "FDBCHK?"	Verify that the sector containing FLDB is correct.

XEQ "CDBDEL"	Clear page 0x806 in RAM and mark the CFLDB as invalid

As you can see, doing even this simple update manually takes much more work than using the automatic functions, mainly because the FLDB also needs to be updated at the same time. But this method does not require that the 41CL update program be running on the host computer.

It is worth noting that the above procedure works even if the YFNX image, located in the same sector as the YLIB image, is plugged in at the time. This is because the entire erase/write procedure in the **P8UPD** function is executed out of RAM without returning to the operating system.

EXAMPLE 6

This example will demonstrate an update using another 41CL as the host machine. It doesn't make sense to keep the two machines connected during the entire update process, because this just drains the batteries on both machines. So we will download the CFLDB from the host machine and then disconnect the two machines and update the CFLDB off-line. Once the CFLDB is ready to use, we'll reconnect the two machines, restart the *41CL Clone Functions* and finish the update. The CFLDB does not have to be transferred back to the host 41CL unless you want to know exactly how many pages need to be updated.

	First do **XEQ "CLONE"** on the host 41CL
XEQ "CMOPEN"	Open the communications channel
XEQ "CDBIMP"	Download the CFLDB.
XEQ "CMCLOSE"	Close the communications channel.
	Turn off both machines and disconnect the serial port between the two machines. Turn on the 41CL.
"*"	Specify all of Flash memory
XEQ "FDBCHK?"	update the CFLDB.
	Turn off the 41CL and reconnect the serial port between the two machines. Turn them both back on.
	Do **XEQ "CLONE"** on the host 41CL.
XEQ "CMOPEN"	Open the communications channel.
XEQ "FLUPD"	Update all of Flash memory. (The "*" from the check is still in the ALPHA register.)
XEQ "CMCLOSE"	Close the communications channel. The host 41CL automatically terminates the CLONE function.
	Turn off both machines and disconnect the serial port between the two machines.

Internal Details

The table below shows the XROM numbers for functions in the *41CL Update Functions*.

Function	XROM Number
FDBVER?	XROM 31,01
FLASH?	XROM 31,02
IDBVER?	XROM 31,03
PGBFR	XROM 31,04
PGCPY	XROM 31,05
PGCRC	XROM 31,06
PGERASE	XROM 31,07
PGINI	XROM 31,08
PGWR	XROM 31,09
P8BFR	XROM 31,10
P8CPY	XROM 31,11
P8ERASE	XROM 31,12
P8INI	XROM 31,13
P8UPD	XROM 31,14
P8WR	XROM 31,15
CDBDEL	XROM 31,17
CDBVER?	XROM 31,18
FDB2CDB	XROM 31,20
PGCDB?	XROM 31,21
PGFDB?	XROM 31,22
PGINV	XROM 31,23
PGUNV	XROM 31,24
PGVAL	XROM 31,25
CMCLOSE	XROM 31,26
CMOPEN	XROM 31,27
CDBEXP	XROM 31,28
CDBIMP	XROM 31,29
PGEXP	XROM 31,30
PGIMP	XROM 31,31
OSPROT	XROM 31,34
OSUPDT	XROM 31,35
AUTOVFY	XROM 31,36
NOVFY	XROM 31,37

In addition to the pages in RAM used for the CFLDB and the Sector Buffer, the *41CL Update Functions* use dedicated locations in RAM to store information required for proper operation. The table below lists these locations and their use

address	Usage	meaning
0x804030	Page 4 MMU during dynamic paging	
0x804031	Communications channel status	0xFFFF = open
0x804032	CFLDB valid	0xFFFF = valid
0x804033	Flash memory device code	
0x804034	SRC address	
0x804035	DST address	
0x804036	Temporary storage	
0x804037	OS Update mode	0xFFFF = enable
0x804038	Auto-verify mode	0xFFFF = enable
0x804039	reserved	
0x804800 - 0x8048FF	code during dynamic paging	

Some of these memory locations are initialized by 41C polling points, according to the table below:

Polling Point	address	Usage	init value
Memory Lost	0x804032	CFLDB valid	0x0000
Memory Lost or Calculator On	0x804031	Communications channel status	0x0000
	0x804037	OS Update mode	0x0000
	0x804038	Auto-verify mode	0x0000
	0x804039	reserved	0x0000

Note that the area of RAM used for dynamic paging is always cleared (written with all zeros) after use, because it is dangerous to leave code that can erase Flash memory hanging around.

Revision History

02/08/2017	Cleaned up and reorganized
02/14/2017	Added CDBVER?, note about tones, auto-inc CFLDB functions
02/18/2017	Added Mode Control functions, Internal Details
02/24/2017	Added BFRCHK? function
02/27/2017	Typos and grammer.
03/01/2017	Error messages. Added another example.
03/08/2017	Typo on p. 24
03/11/2017	Typo on p. 35
03/12/2017	Typos on p. 35, 42
04/01/2017	Verbiage to support *41CL Clone Functions* software
04/20/2017	Clarified which functions are not programmable.
04/21/2017	Corrected and expanded Mode Control function descriptions.
05/13/2017	Added to Error Message table. Clarified the abort operation for several functions.
05/25/2017	Added note about batteries, small editing changes, and another example.
06/01/2017	Added an explanation about FLDB date handling. Other minor edits.
06/08/2017	Added section about initial download of *41CL Update Functions*
06/25/2017	Added table showing organization of FLDB and CFLDB.
08/04/2017	Added tables showing execution time.
08/06/2017	Corrected Device ID for V5 boards.
08/10/2017	Changed FDB2CDB operation. Added invalidation feature to FLUPD.
08/12/2017	Corrected table of tones signalled. rearranged examples.
08/15/2017	Complete reorganization, to emphasize automatic updates.