41CL Image Database

Starting with version -3A of the 41CL Extra Functions, the **PLUGxx** functions use a stand-alone Image Database to hold information about the module images stored in Flash memory. This Image Database can be modified by users so that the **PLUGxx** functions can access new module images stored in either Flash memory or RAM.

Image Database Format

An entry in the Image Database consists of four words, and the format of the database is set up to make adding new entries very easy. An unprogrammed Image Database entry contains 0xFFFF in all four words. This is the default contents of Flash memory, where the database is stored, which means that new entries can simply be added to the Image Database without first needing to erase the Flash sector where the database is stored.

The **PLUGxx** functions will return a ***NO ENTRY*** error message if the user attempts to use a module identifier corresponding to an unprogrammed database entry. An Image Database entry can be "erased" by writing 0x0000 to the first two words. In this case the **PLUGxx** functions will return a ***NULL ENTRY*** error message for the corresponding module identifier.

To preserve backwards-compatibility with previous versions of the 41CL Extra Functions, the Image Database entries are addressed as a function of the first and fourth characters of the module identifier.

In order to limit the size of the database to 4K words only the characters A-Z, 1-5 and 9 (32 possibilities) are allowed for the first and fourth character of a module identifier. In addition, when the first character of the module identifier is 9 and only the characters A-Z are allowed for the fourth character of the module identifier, to provide some space for housekeeping in the database. Any character is allowed for the second and third characters of the module identifier.

The contents of each database entry are shown in the table below.

Image Data base entry word	Address LSBs	digit 3 meaning	digit 2 meaning	digit 1 meaning	digit 0 meaning
1	00	always 0	always 0	image type	address<5>
2	01	always 0	always 0	address<4>	address<3>
3	10	always 0	always 0	character 3 of module identifier	
4	11	always 0	always 2	character 2 of module identifier	

Digit 3 is always 0 in all four words of a database entry. These values are not checked by the **PLUGxx** functions, so these digits may be used for new functionality in the future.

Digit 2 is always 0 in the first three words of a database entry and 2 in the last word of a database entry. These values are not checked by the **PLUGxx** functions, but they may be used for new functionality in the future.

Digit 1 in the first word of a database entry specifies the type of image, according to the table below. Only these values are currently valid as far as the **PLUGxx** functions are concerned, and any other value will cause a ***TYPE ERR*** error message to be returned for the module identifier.

word 1, digit 1	image type for this module identifier
0	4K image (one page)
1	8K image (two pages)
2	16K image (all four banks in one page)
3	16K (four pages)
4	32K (all four banks in two pages)

A type digit of 4 is a slightly special case. The **PLUGxx** functions treat this image type as 32K words, consisting of four banks to be loaded into two adjacent pages. However, both of the images that use this type identifier really only use the first two banks in the second page. This leaves two 4K word sections of memory available to store other images, and the 41CL takes advantage of this space. So the database search functions treat a type digit of 4 as 24K words in length, and return search results accordingly.

Digit 0 of the first word and digits 1 and 0 of the second word of a database entry hold the starting memory address for the image referenced by the module identifier. This is a physical address that can be in either Flash memory or RAM. Note that if the starting address is 000, along with a type digit of 0, the entry is considered a null entry.

Digits 1 and 0 of the third and fourth words of a database entry hold the middle two characters of the module identifier. This allows an address-based search of the database to return the full module identifier. It would also allow the **PLUGxx** functions to check for

the full module identifier, but this feature was not implemented to preserve backwards-compatibility. These two words can use either the “display” encoding, where A-Z are encoded as 0x41-0x5A, or the “assembly language” encoding, where A-Z are encoded as 0x01-0x1A. All of the original entries in the Image Database use the “assembly language” encoding, while any user-added entires will always use the “display” encoding.

As mentioned previously, entries in the Image Database are addressed using the first and fourth characters of a module identifier. Each of these characters must be translated to a 5-bit field to create an address to index the database. The table below shows the translation algorithm.

character	41 code	address field
A	41	00000
B	42	00001
C	43	00010
D	44	00011
E	45	00100
F	46	00101
G	47	00110
H	48	00111
I	49	01000
J	4A	01001
K	4B	01010
L	4C	01011
M	4D	01100
N	4E	01101
O	4F	01110
P	50	01111
Q	51	10000
R	52	10001
S	53	10010
T	54	10011
U	55	10100
V	56	10101
W	57	10110
X	58	10111
Y	59	11000
Z	5A	11001
1	31	11010
2	32	11011
3	33	11100
4	34	11101
5	35	11110
9	39	11111

The address for an Image Database entry is formed as shown below:

Image Database address nibble 2				Image Database address nibble 1				Image Database address nibble 0			
3	2	1	0	3	2	1	0	3	2	1	0
First character module identifier address field				Fourth character module identifier address field				word identifier			

Image Database Functions

The Image Database Functions (in the 41CL Extra Functions Plus) allow the user to add, modify, store, and search the Image Database. Although the Image Database normally resides in Flash memory, it is also possible to operate on a copy of the Image Database that has been stored in the Y-Functions Buffer Area of RAM, located at physical address 0x805000.

IMDBF (no arguments)

Executing **IMDBF** sets an internal flag so that the **IMDB?** and **IMDBINS** functions will use address 0x0DF000 in Flash memory as the location of the Image Database. There is no default selection of Flash or RAM, so this command (or the **IMDBR** command) should be issued before using either the **IMDB?** and **IMDBINS** functions. The selection information is stored in RAM at location 0x804014, in the least-significant digit.

IMDBR (no arguments)

Executing **IMDBR** sets the internal flag so that the **IMDB?** and **IMDBINS** functions will use the Y-Functions Buffer area as the location of the Image Database. Note that all other functions, including the **PLUGxx** functions, will continue to use the Image Database residing in the Flash memory.

IMDBF? (no arguments, returns with ***NO*** (Image Database in RAM) or ***YES*** (Image Database in Flash) in the display)

Executing **IMDBF?** tests the Image Database location flag, returning with ***NO*** in the display if the RAM version of the Image Database is selected and ***YES*** in the display if the Flash version of the Image Database is selected. When used in a program, if the Flash ver-

sion of the Image Database is selected the next program line will be executed; and if the RAM version of the Image Database is selected the next line in the program is skipped.

IMDBCPY (no arguments)

Executing **IMDBCPY** copies the 4K Image Database at address 0x0DF000 to the Y-Functions Buffer area at address 0x805000. This function tests that the Image Database is present at address 0x0DF000, and returns with a ***NO IMDB*** error message if this is not true. This function automatically executes in 50x Turbo mode and is equivalent to **YMCPY** with ***0DF>805*** in the ALPHA register.

IMDBUPD (no arguments)

Executing **IMDBUPD** writes the contents of the Y-Functions Buffer area at address 0x805000 to the Flash memory starting at address 0x0DF000. This function tests that the Image Database is present at address 0x805000, and returns with a ***NO IMDB*** error message if this is not true. This function checks that it is executing from RAM, and returns with a ***CODE=ROM*** error message if this is not the case. This function is equivalent to **YFWR** with ***805>0DF*** in the ALPHA register.

IMDBINS (module identifier, type, and starting page address in ALPHA register)

Executing **IMDBINS** inserts a database entry at the appropriate location in the Image Database. This function tests that the Image Database is present, and returns with a ***NO IMDB*** error message if this is not true.

The table below shows the formatting required for the module identifier, type, and address in the ALPHA register for the **IMDBINS** function. Module identifier characters must be valid (A-F, 1-5 or 9), and the type and address characters must be valid hex digits (0-9 or A-F). Any other characters, or the absence of the "-" characters, will result in a ***DATA ERROR*** message.

character	10	9	8	7	6	5	4	3	2	1
physical address	**M4**	**M3**	**M2**	**M1**	**-**	**T**	**-**	**P5**	**P4**	**P3**

When using the copy of the Image Database in the Y-Functions Buffer Area the new entry is unconditionally written to the appropriate location.

When using the copy of the Image Database in Flash memory the function first checks that it is executing from RAM, and returns with a ***CODE=ROM*** error message if this is not

the case. When writing to the Image Database in Flash memory the existing database entry should be unprogrammed, or the entry will likely be corrupted by the write. Writing 0 to the type field, along with an address of 000 will always create a null entry.

IMDB? (module identifier or page address in ALPHA register, returns with the database entry in the ALPHA register as well as the display)

Executing **IMDB?** searches the selected Image Database using the either module identifier or page address in the ALPHA register and returns the corresponding database information. This function tests that the Image Database is present, and returns with a ***NO IMDB*** error message if this is not true.

The module identifier must be properly formatted in the ALPHA register or a ***BAD ID*** error message will result. The page address must be properly formatted in the ALPHA register or a ***DATA ERROR*** error message will result.

character	4	3	2	1
module identifier	**M4**	**M3**	**M2**	**M1**
page address	**-**	**P5**	**P4**	**P3**

The database information is returned in the ALPHA register in the format below. Since a module image may be up to 32K in length (8 pages), more than one physical address can return with a match, but only the information in the actual database entry is returned. If no address match is found the function will result in a ***NO MATCH*** error message. Only the first match (the search proceeds from lowest database address upwards) will ever be returned.

character	10	9	8	7	6	5	4	3	2	1
physical address	**M4**	**M3**	**M2**	**M1**	**-**	**T**	**-**	**P5**	**P4**	**P3**

This function automatically executes the search in the 50X Turbo mode, but even so the search may take several seconds when searching for an address match.

The 41CL Extra Function -4A includes the **IMDB?** function, but in this case the function always searches the Flash version of the Image Database. This is necessary because the **IMDBF** and **IMDBR** functions are not available in the regular 41CL Extra Functions.

Memory Verification Function

The 41CL Extra Functions Plus also contains a function that can help verify the contents of a memory page. For example, this function can be used to verify that a page transferred over the serial port was communicated correctly, or that a page in Flash memory has not been corrupted.

The function calculates the 32-bit Cyclic Redundancy Check (CRC) used for Ethernet, over the 4K words of a memory page. The resulting CRC will be unique for each page, and is capable of detecting all burst errors up to 32 bits in length.

YCRC (page address in ALPHA register, returns with the 8-digit CRC result in the ALPHA register as well as the display)

Executing **YCRC** calculates the CRC over an entire page. The function automatically executes in 50X Turbo mode, but still requires several seconds to complete. The execution time is data-dependent, with a minimum execution time of about 4.5 seconds and a maximum execution time of about 12 seconds.

The page address must be properly formatted in the ALPHA register or a ***DATA ERROR*** error message will result.

character	3	2	1
page address	**P5**	**P4**	**P3**