

Script Language Reference

Publication: D/PE0002/UM/4 January 2012

User Manual for PE0002 Scripting Language

1 Introduction

This document provides a reference to the commands and syntax for the script language used in the PE0002 Evaluation Kit Interface Card.

1.1 History

Version	Changes	Date
1	New Issue – pre-release	11/03/08
1	Full Release: Copyright date changed and definition of white space character added.	13/03/08
2	Copyright date updated. Minor text changes to expand or correct previous text. Glossary expanded. Section on data handling added. Initialisation of arrays at creation added. Fileopenr extended to include optional 'size of file' parameter New commands added: Waitfile Setvect, intson, intsoff, rfi Else, elseif Ones Microdelay Logon, logoff Port1 description now includes hardware configuration bits for PE0002 RevB Mod1 and higher. Runtime error messages section updated. More example script included.	02/05/08
3	Enhancement to delay and microdelay description Buffer definition added. Added length limit on strings. Enhanced description of conditionals. Description of the commands used for handling interrupts extended. Restrictions on use extended. Error report information extended to make locating errors easier.	17/09/10
4	Introductory information now includes static buffers and associated commands. Section on timers added Description of file types added Summary and detailed descriptions added for the following new commands: Waituplink, Bufcfg, Bufopenr, .Bufopenw, Bufread, Bufwrite, Filetobuf, Buftofile, Dialogyesno, Dialogentry, Gettime, Settime, Getport, Getdevice, Settimer, Starttimer, Stoptimer, Added Timer to Setvect command Section on de-bug messages added	24/01/12

1.2 Glossary

CML	CML Microsystems plc group of companies
<CR>	A carriage return plus a line-feed character or the 'Enter' key on host PC keyboard
PC	Script program counter
host PC	A personal computer running the GUI application and connected to the PE0002.
White space character	A space, tab or carriage return character
ISR	Interrupt Service Routine

Contents

1	Introduction	2
1.1	History	2
1.2	Glossary	2
2	Background & System description	4
3	PE0002 data handling	4
4	File types	5
4.1	Text files	5
4.2	Wav files	5
4.3	Binary files	6
5	Static Buffers	6
6	Script - Command Format	7
7	Syntax	7
7.1	Operands	7
7.1.1	Constants	7
7.1.2	C-BUS handling	7
7.1.3	C-BUS Addresses	7
7.1.4	Strings	7
7.1.5	Variables	8
7.1.6	Conditionals	9
8	Commands – Summary	9
8.1	IO involving the host PC	10
8.2	Arithmetic and logic operations	11
8.3	Streaming C-BUS	11
8.4	Program flow	11
8.5	Static Buffers	12
8.6	Timers	13
8.7	Miscellaneous	13
9	Commands – Detailed Description	13
9.1	IO involving the host PC	14
9.2	Arithmetic and logic operations	19
9.3	Streaming C-BUS	21
9.4	Program flow	21
9.5	Static Buffers	24
9.6	Timers	27
9.7	Miscellaneous	28
10	Error Messages	32
10.1	De-bug Messages	32
10.2	Build Errors	32
10.3	Runtime Errors	32
10.3.1	PC out of range	32
10.3.2	Invalid opcode	32
10.3.3	Data index out of range	32
10.3.4	Stack overflow	32
10.3.5	Stack underflow	32
10.3.6	Micro text out buffer overflow	33
10.3.7	Micro File read buffer underflow	33
10.3.8	Micro File read buffer overflow	33
10.3.9	Attempt to read past end of file	33
10.3.10	Invalid Jump Destination	33

10.3.11	Attempt to reference non-existent string	33
10.3.12	Unable to open file	33
10.3.13	Error - RX Queue is full.	33
10.3.14	Static buffers allocation exceeds allowed size	33
10.3.15	Attempt to write past end of static buffer	33
10.3.16	Attempt to read past end of static buffer	33
10.3.17	Static buffer not open for reading/writing	34
10.3.18	Uplink buffer is smaller than requested space	34
10.3.19	UART overflow / USB comms error / USB queue full /Serial framing error	34
10.3.20	Other possible errors	34
10.4	Example 1	35
10.5	Example 2	35
10.6	Example 3	35
10.7	Example 4	36
10.8	Example 5	36
10.9	Example 6	39

2 Background & System description

This script language was developed for evaluating CML's new generation ICs including *FirmASIC*[®]-based products, and greatly simplifies the approach to the evaluation and design-in process.

A host PC-based GUI loads, compiles and controls scripts, which are plain text files, and executes them on the PE0002. The scripts use a simple syntax to allow the user to read and write up to 2 C-BUS ports on the PE0002 and includes flexible program flow control, data manipulation and message display. The result is much better real-time performance than when executing the script on the host PC. The script language also provides debugging, tracing and I/O control facilities.

3 PE0002 data handling

The memory on the PE0002 is divided into a number of discrete segments. At run time, the entire script is compiled and downloaded to the PE0002 "Command Buffer", a memory segment of 8192 script Commands. A fixed "Data Buffer" is set aside for the contents of variables and arrays used in the script. The size of the Data Buffer is currently 61440 words. If the Command Buffer or Data Buffer is exceeded then the script will abort and a "Data out of range" message will be displayed in the log window at run-time.

When a **filew** command is encountered, data is transferred from the PE0002 to the host PC via a fixed buffer (FIFO). This is transparent to the user and the buffer is unlikely to overflow and cannot underflow in use.

When a **filer** command is encountered, data is transferred from the host PC to the PE0002 via another fixed buffer (FIFO), the "File Buffer". Its size is 294512 words and this buffer cannot overflow but may underflow in use.

It is important to understand how data is handled and transferred between the PE0002 and the host PC using these commands.

The PE0002 packs all data in a 16-bit word, regardless of format. This includes data transferred to and from the host PC.

A **fopenw** command opens or creates a file on the host PC that will be used to receive data from the PE0002 script. Each time a **filew** command is encountered, 16-bits of data are transferred from the script to a buffer. The GUI application on the host PC will read this buffer in bursts and write the data to the file in the required format.

A **fopenr** command opens a file on the host PC that will be used to send data to the PE0002 script. The script is suspended while the file is downloaded to the PE0002 File Buffer. The data from the file is read using the format specified in the **fopenr** command and, if smaller, padded to 16-bits. If possible, the entire file will be transferred to the File Buffer and the script will continue. If the file is too large, the script will suspend until the File Buffer is full and then resume. Each time a **filer** command is encountered, 16-bits of data are transferred from the File Buffer to the destination specified. As data is consumed by **filer** commands, the host PC will send bursts of data to the File Buffer until the entire file has been transferred.

If the data is consumed faster than it is sent then the script will abort and a run-time error message, "File buffer underflow" will be displayed in the log window. To avoid an underflow, the script tool provides the **waitfile** command. This causes the script to suspend until the File Buffer contains at least the number of words specified by the command.

A **fopenr** command has an optional variable associated with it. This variable acts as a size-of- file at the time the file is opened although it will report files greater than \$FFFF as \$FFFF – the size limitation of variables. After the first **filer** command is encountered, the variable will act as a number-of-file-reads-remaining since its value is re-calculated following every **filer**.

Note there is no direct connection between the size of the file reported by the **fopenr** variable and the real size of the file. This is because **fopenr** is reporting the number of possible **filer** cycles that can be performed on the file not its size.

The current sizes of the Command, Data and File Buffers, are given in the About box of the PC GUI.

The data transfer rate between the host PC and the PE0002 depends on the host PC processors speed, the OS and other applications that are running. Data is transferred across the USB link in bursts so file reads and writes are best managed in bursts with a script delay imposed between. For optimal data handling between the PE0002 and the target card, try and keep as much data on the PE0002 as possible: Ensure that files read from the host PC will require less than 256k filer cycles. The script will be suspended while these files are downloaded.

Try to avoid writing data to files on the host PC where the write cycles will be continuous or with little script processing in between. Also try to limit the rate at which messages are sent to the GUI. The PE0002 buffers this data until it is consumed by the PC/GUI. If the buffer fills faster than the PC/GUI can empty it, an overflow will occur and the script will terminate. See – Micro File read buffer underflow.

Where possible, write to an array on the PE0002 and then transfer the data to the host PC at the end of the script or while the script is busy on other tasks or where delays can be inserted between the **filew** commands. If a delay in script execution can be tolerated use the **waituplink** command to stall script execution until the data is transferred.

4 File types

The script language recognises three filetypes; wav, binary and text. The filetype is inferred from the extension. '.wav' is treated as wav, '.bin' as binary and anything else as text.

4.1 Text files

Reading: Files should have one value per line. Lines should be formatted as per the string supplied in the **fopenr** command. Blank lines and leading/trailing whitespace are ignored. Any text after a semicolon is ignored. **Fopenr** reports the number of values that can be read from the file.

Writing: Line will be written as per the formatted string in **filew**. Only one value per **filew** command, but /c can be used to put more values on a line.

4.2 Wav files

Only 16-bit PCM, 1 or 2 channels are supported.

Reading: Header is ignored, except for checking valid sample size and number of channels, and reporting the length to the script. This value is the total number of samples in the file (as read from the header), regardless of number of channels. If there are 2 channels, consecutive **filer** commands will read samples from alternate channels. The formatted string in **fopenr** is ignored, but must be present if the 'filesize' parameter is used.

Writing: Header will be written with the sample rate and number of channels supplied in **fopenw** command. If there are 2 channels, consecutive **filew** commands will write samples to alternate channels. The string in **filew** is ignored, but must be present.

4.3 Binary files

Reads and writes are 16-bit. **Fopenr** reports the number of 16-bit words that can be read from the file. The formatted string in **fopenr** is ignored, but must be present if the 'filesize' parameter is used. The string in **filew** is ignored, but must be present.

5 Static Buffers

The script can configure microprocessor memory as one or two static buffers. The static buffers differ from the Command and Data Buffers in that data written to these buffers is preserved between script runs. The PE0002 has \$280000 words of space available for static buffers. However, because the script language is limited to a 16-bit environment, it is only possible to read and write a maximum of \$FFFF words of data per **bufread** or **bufwrite** command. Commands **buftofile** and **filetobuf** can transfer all data in one pass.

The script language for static buffers uses the concept of 'data count' and 'space count' to indicate the amount of data that can be read from the buffer, and written to the buffer, respectively. These concepts are described in more detail below and throughout the detailed descriptions for individual commands later in this document.

There are a number of commands associated with the static buffers.

Before accessing the buffers and before any further commands can be issued, the size of the buffers must be configured using the **bufcfg** command. This allocates one or two buffers for use and sets the size of the buffers. Before any read or write access, a buffer must be opened with the **bufopenr** or **bufopenw** commands respectively.

The **bufopenr** command opens the buffer for reading and includes an optional variable which will be set to the number of items which can be read from the buffer. This is referred to as the Data Count'.

Bufopenw has a similar feature, but will indicate the number of items which can be written to the buffer. This is referred to as the Space Count. These values are updated as data is read from and written to the buffers. Writing into these variables from the script is not recommended.

'Data count' and 'Space Count' are returned to optional script variables. The size of all variables is 16-bit, which allows a maximum value of \$FFFF. The buffers can be up to \$280000 words which means that 'Space Count' can only accurately reflect the state of the buffers when \$FFFF or less space remains. Otherwise 'Space Count' returns \$FFFF. Similarly, 'Data Count' can only indicate that \$FFFF or less items are stored in the buffer. If more than \$FFFF items are stored in the buffer, 'Data Count' returns \$FFFF. Where data transfers to/from the buffer are not limited, the script must control the accesses to ensure that the buffers are not overflowed or underflowed.

Bufopenr resets the read pointer to the start of the buffer, and updates the data count accordingly.

Bufopenw resets both read and write pointers (effectively clearing all current content from the buffer) and updates both data and space counts accordingly. **Bufcfg** also resets both pointers and counts, but for both buffers.

Data is transferred between a static buffer and a C-BUS register or variable using the **bufread** and **bufwrite** commands. These can transfer up to \$FFFF words per pass, due to the maximum size of the controlling variable. This assumes an the variable is an array or a streaming C-BUS register (though the array/streaming C-BUS FIFO is unlikely to be this big). The script will abort with an error if there is an attempt to write past the end of the buffer, or read past the end of the data.

Data can be transferred between a static buffer and a file using the **filetobuf** and **buftofile** commands.

Filetobuf will transfer the entire file to the buffer, unless the file is bigger than the buffer in which case the script will abort with an error. **Buftofile** will transfer the entire buffer to the file. Before these commands can be used, the file must be opened using the **fileopenr** or **fileopenw** commands and the buffer must be opened using **bufopenr** or **bufopenw** as appropriate.

6 Script - Command Format

Commands take the general form of:

label command operand1,operand2,..... ;comment

- Labels must start in the first column and must start with a letter or underscore character. Labels are case sensitive.
- Commands must be separated from a Label by at least one white space character except <CR>. Commands are not case sensitive.
- The command must NOT start in the first column.
- Only one command is allowed per line.
- Operands must be separated from the command by at least one white space character except <CR>.
- Operands must be separated by a comma or at least one white space character except <CR>. The number of operands required by each command varies and some commands have a flexible syntax.
- Blank lines, lines with just a label, just a comment or just a command (plus operands) are all allowed.

7 Syntax

7.1 Operands

7.1.1 Constants

Constants are declared as follows:

```
STATUS            const    $C6  
BIT1_MASK        const    2
```

The following are all treated as 16-bit:

```
#1234            Constant, decimal value 1234.  
678             Constant, decimal value 678.  
#$56AB          Constant, hex value 56AB.  
$FE             Constant, hex value FE.
```

24-bit constants

Whilst the script generally only deals with 16-bit values, some commands can accept values of up to 24 bits (0xFF FFFF). This is noted in the relevant entry in the command tables and in the descriptions.

7.1.2 C-BUS handling

C-BUS register accesses comprise an address byte followed by zero or more data bytes. The script compiler assumes a default length of two data bytes. To change this, use the **register** command. For streaming C-BUS register access, use the **read** and **write** commands.

7.1.3 C-BUS Addresses

A * character is used to specify the address of a C-BUS register (C-BUS address).

```
*45             C-BUS address specified in decimal, value 45  
*$A7            C-BUS address specified in hex, value A7  
*STATUS        C-BUS address specified by the constant STATUS
```

7.1.4 Strings

Strings have a maximum length of 64 characters. If this length is exceeded the scripting tool will abort without warning.

" hello world." Strings presented in double quotes.

7.1.4.1 Formatted strings

Some of the commands make use of 'formatted strings'. These work much like the 'printf' function in C, with the following differences:

- A carriage-return and line-feed (\n in the C language) is assumed at the end of a line (use \c to continue if you don't want a new line).
- The **disp**, **dialog** & **filew** commands support only one parameter after the string, so only one format specifier can be used in the string.
- Formatted strings have a maximum length of 64 characters. If this length is exceeded the scripting tool will abort without warning.
- The standard format specifiers supported are:
 - %d (signed decimal).
 - %u (unsigned decimal).
 - %x and %X (hexadecimal).
 - %f (floating point. When used in filer it is converted to a 16-bit signed value and a warning generated if out of range).

The usual flags, field width and precision specifiers can be used with these.

- The following additional format specifiers are also available
 - %b (binary) no leading zero suppression.
 - %q (Q15 fixed point)

7.1.5 Variables

Variables are stored and treated as 16-bit unsigned values throughout the script except when operated on by the arithmetic shifts, asl and asr, when the sign bit, b15, will be preserved.

Variables are declared as follows:

```
bar    word n           ;Specifies a 16-bit variable 'bar', initialised to
                        ; value 'n'.
foo    buffer x          ;Specifies an array of x 16-bit words, all
                        ;initialised to zero.
foo    word 1,2,3,4,5    ;Specifies an array of 5 16-bit words,
                        ;initialised to the values 1 through 5.
```

Note that 'bar' & 'foo' are labels that identify the variables, so must start in the first column.

Variables are global and can be declared and used at any point in a script. They can also be used before they are declared.

Variables are case sensitive.

Only one instance of any variable can exist.

Arrays of variables are indexed from 0.

When initialising an array, the values must be on the same line and separated by at least one white space character except <CR>.

Variables can be accessed in the following ways:

bar	Return the variable bar
foo	Return the first word in the array foo.
foo[0]	Return the first word in the array foo.
foo[3]	Return the fourth word in the array foo.
foo[bar]	Return the word at n+1 in the array foo. (bar = n)
foo[bar++]	As above, and increment bar.
foo[bar--]	As above, but decrement bar.

If the post-increment and post-decrement operators are used in arrays, no bounds checking is done by the script. The user must ensure that the indexing remains in bounds or the script may behave unexpectedly.

7.1.6 Conditionals

The following conditional evaluators and operators can be used with the **jmpc**, **jsrc**, **if**, **elseif** and **while** commands:

<	Less than
>	Greater than
= or ==	Equal to
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
!&	Logical inverse of bitwise AND
!	Logical inverse of bitwise OR
!^	Logical inverse of bitwise XOR

Bitwise !& (NOT AND) is equivalent to operand1 AND operand2 with the result being negated. The following example compares the !& to &:

operand1 & operand2	TRUE if (operand1 AND operand2) is equal to zero FALSE if (operand1 AND operand2) is not equal to zero
operand1 !& operand2	TRUE if (operand1 AND operand2) is not equal to zero FALSE if (operand1 AND operand2) is equal to zero

The same is true for the other 'logical inverse' conditions, !| and !^.

The script environment assumes only unsigned integer values are used. Be cautious when using evaluators as there is no concept of values less than zero. For example:

```
copy <var> to count          ;get number of items to be processed
while count > 0
    ..                        ; do something
    ..
    sub 2 count
endwhile
```

This code fragment will work as expected provided count is assigned an even number before the loop. If count is assigned an odd number, it will be decremented; 0005.. 0003.. 0001.. FFFF – The loop will *never* exit.

The evaluation of a condition does not change any variables. For example:

```
copy *Status to sstatus      ; read the device status into shadow

If sstatus & $1000            ; Test status bits without changing the shadow
    ..                        ; do something if b12 = 1
    ..
elseif sstatus !& $10
    ..                        ; do something if b4 = 0
    ..
Endif
```

8 Commands – Summary

A detailed description of the commands is available in section 9. The following nomenclature is used throughout this section:

"str" A formatted string or formatted string as described in section 7.1.4
 <cond> A condition code as listed in section 7.1.6
 <label> A label referencing a script line, used in **jmp**, **jsr**, **if** & **while** commands.
 #<const> A constant as described in section 7.1.1
 #<const24> A 24-bit constant as described in 7.1.1
 *<C-BUS> A C-BUS address as described in section 7.1.3
 <var> A variable as described in section 7.1.5
 <any> Any one of #<const>, or *<C-BUS> or <var> (as specified in the description)
 <any24> Any one of #<const>, #<const24>, *<C-BUS> or <var>
 Where more than one parameter of the same type is used, an identifier will be added, for example:
 <any24a> <any24b>
 <var1> <var2>

8.1 IO involving the host PC

Command	Parameters	Description
Fopenr	"filename", "str"	Instructs host PC to open file "filename" for reading using formatted string <str>
	"filename", "str", <var>	As above but writes the number of filer cycles remaining into the variable <var>
Fopenw	"filename"	Instructs host PC to open file "filename" for writing.
	"filename", <any24>, <any>	Instructs host PC to open a wav file "filename" for writing. The sample rate is defined in <any24> and the number of channels in <any>.
Waitfile	#<any>	Pause the script until the file open for reading has downloaded at least enough data to complete <const> filer cycles
Filer	*<C-BUS>	Read a value from the PE0002 FIFO created by fopenr and write it to the C-BUS address *<C-BUS>.
	<var>	As before, but write to variable <var>
Filew	"str", <any>	Read the C-BUS address, variable or constant <any> and write the data to the host PC file using formatted string "str"
	"str"	Write the string "str" to the host PC file
Waituplink	<any>	Pause the script until the buffer for sending to the GUI has enough space for <any> messages.
Disp	"str", <any>	Read the C-BUS address, variable or constant <any> and display the data in the log window using formatted string "str"
	"str"	Write the string "str" to the log window
Dialog	"str", <any>	Read the C-BUS address, variable or constant <any> and display the data in a dialog box using formatted string "str".
	"str"	Display a dialog box containing the string "str"
Dialogyesno	"str", <var>	As dialog , except this dialog box has three buttons; 'Yes', 'No' and 'Abort'. If the user presses 'Yes' <var> will be set to 1. If the user presses 'No' <var> will be set to 0. 'Abort' will abort the script.
Dialogentry	"str", <var>	As dialog , except this dialog box has an edit box; If the user presses 'OK' <var> will be set to the value typed in the edit box (decimal, or hex preceded by \$ or 0x are accepted). 'Abort' will abort the script.
Cls	None	Clear the contents of the log window

8.2 Arithmetic and logic operations

These commands take the form; command <source>, <destination>.
The result is always in the <destination> operand.

Command	Parameters	Description
Copy	<any>, <var>	<var> = <any>
	<any>, *C-BUS>	*C-BUS = <any>
Add	<any>, <var>	<var> = <var> + <any>
Sub	<any>, <var>	<var> = <var> - <any>
And	<any>, <var>	<var> = <var> & <any>
Or	<any>, <var>	<var> = <var> <any>
Xor	<any>, <var>	<var> = <var> ^ <any>
Lsl	<any>, <var>	<var> = <var> << <any>
Lsr	<any>, <var>	<var> = <var> >> <any>
Asl	<any>, <var>	<var> = <var> << <any> (Sign bit preserved)
Asr	<any>, <var>	<var> = <var> >> <any> (Sign bit preserved)

8.3 Streaming C-BUS

These two commands only apply to C-BUS when used in streaming mode. Not all C-BUS devices support streaming mode.

Command	Parameters	Description
Read	*C-BUS>, <var>, <any>	Read C-BUS in streaming mode from C-BUS address *C-BUS> and store the results into an array starting at <var>. <any> determines the number of words read.
Write	<var>, *C-BUS>, <any>	Read from an array starting at <var> and write to C-BUS address *C-BUS>. <any> determines number of words copied.

8.4 Program flow

Throughout this section, zero equates to FALSE, any non-zero value equates to TRUE.

Command	Parameters	Description
Jmp	<label>	PC=<label>
Jmpc	<any> <cond> <any> <label>	PC=<label> if <cond> is TRUE
Jsr	<label>	Push PC+1, PC=<label>
Jsrc	<any> <cond> <any> <label>	As jsr if <cond> is TRUE
Return	None	Pop PC
Setvect	#<const>, <label>	On interrupt from device <const> Push PC+1, PC=<label>
Intson		Turn on script interrupt handling
Intsoff		Turn off script interrupt handling
Rfi		Pop PC and re-enable interrupts
If	<any><cond><any>	If <cond> evaluates as FALSE: Jump to matching 'endif' or first matching 'else' or 'elseif'. If <cond> evaluates as TRUE: Execute until matching 'endif', or execute until first matching 'else' or 'elseif' then jump to matching 'endif'
	<any>	As before but <any> is evaluated instead of <cond>
Else	None	This is just a marker for if

Elseif	<any> <cond> <any>	If <cond> evaluates as FALSE: Jump to matching 'endif' or next matching 'else' or 'elseif'. If <cond> evaluates as TRUE: Execute until matching 'endif', or execute until next matching 'else' or 'elseif' then jump to matching 'endif'
	<any>	As before but <any> is evaluated instead of <cond>
Endif	None	This is just a marker for if
While	<any> <cond> <any>	Jump past matching 'endwhile' if <cond> evaluates as FALSE.
	<any>	As before but <any> is evaluated instead of <cond>
Endwhile	None	Jump to matching 'while'
Stop	None	Stop executing script

8.5 Static Buffers

Command	Parameters	Description
Bufcfg	<any24>	Set the size of static buffer 1 to <any24a>
	<any24a>, <any24b>	Set the size of static buffer 1 to <any24a> and static buffer 2 to <any24b>
Bufopenr	<any>, <var>	Open buffer const or var <any> for reading. <var> number of values can be read.
	<any>	Open buffer const or var <any> for reading.
Bufopenw	<any>, <var>	Open buffer const or var <any> for writing. <var> number of values can be written.
	<any>	Open buffer const or var <any> for writing.
Bufread	<any1>, <var>, <any2>	Copy data from static buffer const or var <any1> to var or array <var>. <any2> sets the number of words to copy. Defaults to one word if omitted.
	<any1>, *<C-BUS>, <any2>	Copy data from static buffer const or var <any1> to C-BUS address *<C-BUS>. <any2> sets the number of words to copy. Defaults to one word if omitted.
Bufwrite	<any1>, <var>, <any2>	Copy data from var or array <var> to static buffer const or var <any1>. <any2> sets the number of words to copy. Defaults to one word if omitted.
	<any1>, *<C-BUS>, <any2>	Copy data from C-BUS address *<C-BUS> to static buffer const or var <any1>. <any2> sets the number of words to copy. Defaults to one word if omitted.
Filetobuf	<any>	Transfer all data from the file opened with fileopenr to static buffer const or var <any>
Buftofile	<any>, "str"	Transfer all data from static buffer const or var <any> to the file opened with fileopenw . "str" is required in all cases, but ignored for binary and wav files.

8.6 Timers

Command	Parameters	Description
Settime	<any>	Set current value of count up timer to <any>
Gettime	<var> or *<C-BUS>	Read current value of count up timer to <var> or <C-BUS>
Settimer	<any>	Set start value of count down timer to <any>
Starttimer	None	Start count down timer running
Stoptimer	None	Stop count down timer running.

8.7 Miscellaneous

Command	Parameters	Description
Port	<any>	Select Port <any> to be 'current'
Getport	<var>	Read currently selected port into <var>
Portc	<any>	Configure current port with value of <any>. A 1 in a bit position sets the corresponding bit in the port to be an INPUT. A 0 sets it to be an OUTPUT.
Portw	<any>	Set hardware output pins of current port to value of <any>.
Portr	<var>	Set <var> or *<C-BUS> to value read from hardware input pins of current port.
	*<C-BUS>	As before but send the value read to C-BUS address *<C-BUS>
Ones	<any>,<var>	Count the number of ones in <any>. Put the result in <var>.
Device	<any>	Select Device <any> (C-BUS device 1 or 2)
Getdevice	<var>	Read currently selected device into <var>
Delay	<any>	Script waits for <any> milliseconds
Microdelay	<any>	Script waits for <any> x 10 microseconds
Register	#<const1>, #<const2>, #<const3>	<const1> is the device <const2> is the C-BUS register address <const3> is the number of bytes required after the address byte
Logon	None	Start logging C-BUS transactions
Logoff	None	Stop logging C-BUS transactions

9 Commands – Detailed Description

Commands are not case sensitive; Fopenr is the same as fopenr.

Commands must not start in the first column but they can start in any other column.

A label, but no other text, can precede a command on the same line.

A command and its parameters must all be on same line. If the line is broken then compile errors or unexpected script operation will occur.

A command's parameters can separated by a single comma, any number of white space characters (except carriage return) or any mix of these. Eg.

```
fopenr "myFile.txt" "04x" FileSize
fopenr "myFile.txt", "04x", FileSize
fopenr "myFile.txt", "04x", FileSize
```

are all valid and will give the same result.

The following structure is used throughout this section:

Command

Parameters

Description

Example usage

Restrictions

The following nomenclature is used throughout this section:

"str"	A formatted string as described in section 7.1.4.1
<cond>	A condition code as listed in section 7.1.6
<label>	A label referencing a script line, used in jmp , jsr , if & while commands.
#<const>	A constant as described in section 7.1.1
#<const24>	A 24-bit constant as described in 7.1.1
*<C-BUS>	A C-BUS address as described in section 7.1.3
<var>	A variable as described in section 7.1.5
<any>	Any one of #<const>, or *<C-BUS> or <var> (as specified in the description)
<any24>	Any one of #<const>, #<const24>, *<C-BUS> or <var>
Where more than one parameter of the same type is used, an identifier will be added, for example:	
	<any24a> <any24b>
	<var1> <var2>

9.1 IO involving the host PC

Fopenr

Parameters - "filename", "str", <var> or "filename", "str"

Description – The script instructs the host PC to open file "filename" for reading. If the file/folder does not exist then an error message will be displayed. The file can be opened from a sub-folder using path specifiers in front of the file name. The string "str" describes the format of the data read.

The optional parameter <var> can be used to determine the effective size of the file or as an end of file marker. At run-time, the host PC will determine the number of possible **filer** cycles that are possible with the file. This value will be written to the variable <var> and updated every time a **filer** is executed. If the number of possible **filer** cycles exceeds \$FFFF, the variable limit, then the variable will read \$FFFF until the number of remaining **filer** cycles can be stored precisely.

At runtime, data that matches the format specified in the **fopenr** command is copied from the host PC file. This data is converted to unsigned 16-bit word and stored in the File Buffer. The entire file is processed in this way or until File Buffer is full. If the File Buffer is smaller than the file, the file's content will be piped in bursts until the entire file has been transferred.

When opening wav or binary files, the formatting string is ignored but must be present if the variable indicating the size of the file is required.

Each **filer** cycle accesses one word sequentially from the File Buffer. Should the data in the File Buffer be consumed faster than it is transferred from the host PC, an empty buffer may occur. This event will terminate script execution with an error message. Such buffer underflows can be avoided using the **waitfile** command.

The same file can be opened for reading and writing but note that the **fopenw** command erases the file contents when executed by the script. Any file previously opened for reading will be closed before the specified file is opened.

Example Usage – The following script fragment uses the parameter 'FileSize' in fopenr to check that the size of the file is not too large for the array, "Buffer". It then uses the same parameter to control download of the entire file content to the array.

```
fopenr "inputfile.txt", "%04X" FileSize ;Tell host PC to open file
                                         ;inputfile.txt and stream
                                         ;data down to the PE0002
                                         ;File Buffer

if FileSize > #$FF
    Dialogue "Error in Source File - Aborting"
    stop
endif

while FileSize != #0 ;While the file contains valid data
```

```

        filer Buffer[BufSize++]      ;read it into the array and
                                   ;increment the index variable
    endwhile                       ;end of while loop

```

Also see Script Examples, Example 4

Restrictions – The file name, including any path specifiers, cannot be longer than 64 characters and the filetype should be specified. Unspecified filetypes will default to NULL. The folder in which the source script resides is the default.

Only one file can be open at any time for reading.

The rate at which the PE0002 File Buffer is filled depends on many factors including the USB link and the host PC hardware and OS. Any activity that reduces the rate at which data is sent over the USB link may cause the File Buffer to underflow.

The format specifier determines how binary data is read from the file. The data is stored as unsigned 16-bit words in the File Buffer.

Waitfile

Parameters - <any>

Description – This command allows the script to pause while data is downloaded from the host PC, potentially avoiding File Buffer underflows. At run time, script execution pauses at this command until the PE0002 file buffer contains at least the number of **filer** cycles that can be executed on the file specified in <any>. If there is insufficient data in the file, the script will pause until the last of the data has been downloaded and then the script will continue even though the PE0002 File Buffer has not been filled to <any> bytes. Data that does not match the required format specified in the **fopenr** command will be ignored. Waitfile is only required when the file being read contains more than 256k **filer** cycles. For files smaller than this, the script will pause at the **fopenr** command until all the data is downloaded.

Example Usage – This script fragment reads data bursts from a large external file (>256k). To ensure that the PE0002 File Buffer always contains enough data for the loop, a **waitfile** has been inserted. The script will only pause at the **waitfile** if there is insufficient data in the PE0002 File Buffer. If the PE0002 File Buffer contains at least 27 **filer** cycles, the script will not pause at the **waitfile** command.

```

const FrameSize 27

    copy #0 count          ;reset counter
    WaitFile FrameSize
    while count < FrameSize ;Get another Frame ready
        filer Frame[count++]
    endwhile

```

Restrictions – The maximum **waitfile** delay is 64k filer reads. This limitation is due to the maximum value that can be assigned to variables.

Fopenw

Parameters - "filename", <any24>, <any> or "filename"

Description - The script instructs the host PC to open file "filename" for writing. If the file/folder-list does not exist then it will be created. The file can be opened from a sub-folder using path specifiers in front of the file name. The format of the data written is controlled by the **filew** command. The file contents, if any, will be erased when the **fopenw** command is executed by the script. The write is piped to the host PC via a single buffer. The script will terminate with an error message if the buffer overflows.

The same file can be opened for reading and writing but note that the **fopenw** command erases the file contents when executed by the script.

If the file to be opened is a wav file then the sample rate <any24> and the number of channels <any> must be given.

Example Usage - None

Restrictions – The file name cannot be longer than 64 characters and the filetype should be specified. Unspecified filetypes will default to NULL. The folder in which the source script resides is the default. The rate at which the PE0002 buffer is emptied depends on many factors including the USB link rate, the host PC hardware and OS activity. Any activity, including PE0002 commands and data, which reduces the rate at which data is consumed by the host PC may cause the buffer to overflow.

Filer

Parameters - *<C-BUS> or <var>

Description – Read the next value from the File Buffer created by **fopenr** and write it to the C-BUS register <C-BUS> or variable <var>.

Data that matches the format specified in the previous **fopenr** command is copied from the host PC file. This data is converted to unsigned 16-bit word and stored in the File Buffer. The entire file is processed in this way. Each **filer** cycle accesses one word sequentially from the File Buffer.

Values read by **filer** are converted to the format required by the destination. Leading zeroes are added where necessary.

*<C-BUS>, write the value to the C-BUS address as MSB/LSB if the register is 16-bit or LSB only if the register is 8-bit.

<var> - write the value to the variable as a 16-bit unsigned value.

Example Usage – The following table shows the data that will be passed to a variable by the **filer** command. Note how the data is padded with leading zeroes to maintain the 16-bit unsigned integer format of variables. If the target is a C-BUS register address, then leading zeroes will be added to accommodate the number of data bytes required by the C-BUS register specified.

File Content	format string in fopenr command		
	01x	02x	01d
ABCD	000A	00AB	Ignored
AB CD	000A	00AB	Ignored
0	0000	0000	0000
01	0000	0001	0000
A B	000A	000A	Ignored
h	Ignored	Ignored	Ignored
1	0001	0001	0001

See Script Examples, Example 4

Restrictions – See **fopenr** command

Filew

Parameters - "str", <any> or "str"

Description – Read the C-BUS register address, variable or constant <any> and write it to the file created by **fopenw**. The format of the data written is specified by the parameter "str". The <any> parameter can be omitted so that only the string "str" will be written to the file. If the file being written is a binary or wav file then the formatting string is ignored.

Example Usage -

```
filew "Received Data" ;Start the file with a header
```

Restrictions – The length of the string cannot exceed 64 characters. Only one placeholder can appear in the string and if present, must be followed by the second parameter <any>.

Waituplink

Parameters - <any>

Description – File writes are written into a buffer as they are executed, and sent from the micro to the GUI as soon as possible. If the script writes faster than the data can be sent the buffer will overflow and the script will abort with an error message. To avoid an overflow, and if delay in script execution can be tolerated, the script provides the **waituplink** command. This causes the script to suspend until the buffer has enough space for the number of messages specified by <any>.

Example Usage - None

Restrictions – It is possible that **waituplink** may fail in some instances. This is due to the rate at which the PE0002 and the GUI are processing. **waituplink** is more likely to fail on a faster PC.

Disp

Parameters - "str", <any> or "str"

Description – Read the C-BUS register address, variable or constant <any> and write it to the log window. The format of the data written is specified by the parameter "str". The <any> parameter can be omitted so that only the string "str" will be written to the log window. Use **disp** to send messages to the log window that indicate script progress or to help de-bug scripts. To provide user prompts, use Dialog

Example Usage –

```
disp  "Data Exchanged"                ;Debug - Table Header in log window
disp  "Write Value =%x", OutValue      ;Value written in hex
disp  "Read Value =%b", RetValue       ;Result returned in binary
```

Also see Script Examples, Example 4

Restrictions – The length of the string cannot exceed 64 characters. Only one placeholder can appear in the string and if present, must be followed by the second parameter <any>.

Dialog

Parameters - "str", <any> or "str"

Description – Read the C-BUS register address, variable or constant <any> and display it in a dialog box. The format of the data written is specified by the parameter "str". The <any> parameter can be omitted so that only the string "str" will be written to the log window.

Script execution will be suspended until one of the dialogue buttons is clicked. The dialog box is modal so must be addressed. Clicking 'OK' will resume the script. Clicking 'Abort' will terminate the script.

Dialog is best used to provide user prompts or warnings. It can be used with the Trace facility to debug script because it provides an exact point at which to stop and abort script execution. The trace facility can then be accessed from the GUI window. Use **disp** to send messages to the log window that indicate script progress or to help de-bug scripts.

Example Usage - Same as **disp** but the string is displayed in a modal dialog box. Also see Example 4

Restrictions – The length of the string cannot exceed 64 characters. Only one placeholder can appear in the string and if present, must be followed by the second parameter <any>. The dialog box is modal so will prevent the GUI application being focused until closed.

Dialogyesno

Parameters – “str”,<var>

Description – Produces a dialog box with a message “str” and three buttons: ‘Yes’, ‘No’ and ‘Abort’. Script execution will be suspended until one of the dialogue buttons is clicked. The dialog box is modal so must be addressed. If the user clicks ‘Yes’, a 1 will be returned to the variable and the script resumes execution. If the user clicks ‘No’, a 0 will be returned to the variable and the script resumes execution. If ‘Abort’ is selected the script will be aborted.

Example Usage –

```
response word                                ;declare variable response

start                                        ;start of script
..                                          ;do something
..
dialog "Repeat Transmission" response      ;display the dialogue
If response == 1                          ;if yes..
    jmp start                             ;loop back to the start
elseif
response == 0                             ;if no..
    jmp shutdown                          ;close down gracefully
                                           ;Abort will stop the script but will
                                           ;not stop the hardware

endif

shutdown                                    ;shutdown routine
..                                          ;script to shutdown the hardware and
                                           ;perhaps
Stop                                       ;read some debug values
```

Restrictions – The length of the string cannot exceed 64 characters. The dialog box is modal so will prevent the GUI application being focused until closed.

Dialogentry

Parameters – “str”,<var>

Description – This command gives the user the ability to modify a variable independent of the script. This is useful for testing and debugging scripts. When the script reaches the line at which **dialogentry** is located, the dialog will be displayed with the message “str” and an edit box. The script will be suspended at this point and the user can enter a value in the box. Values can be hexadecimal, prefixed with \$ or 0x, or integer. Integers are treated as signed and so negative numbers will be limited to 15-bit and the sign bit set to 1. When the user clicks ‘OK’ the value entered will overwrite the current value of the variable <var>. If the user clicks ‘Abort’ the script will be aborted.

Example Usage –

```
; This loop fragment did not exit so dialogentry has been inserted to force the
;loop to exit on the required loop_count. De-bug code has been added at the end of
the loop to determine the problem.
while loop_count < max
    ..
    ..
    add 1 loop_count
    dialogentry "Modify loop_count" loop_count
endwhile
;run some debug code
```

Restrictions – The length of the string cannot exceed 64 characters. The dialog box is modal so will prevent the GUI application being focused until closed. Integer values entered via the edit box will be treated by the environment as unsigned. It is the user's responsibility to handle signing appropriately.

Cls

Parameters - None

Description – Clear screen. Clears the log window and returns the cursor to the upper, left-hand corner.

Example Usage - None

Restrictions – None

9.2 Arithmetic and logic operations

These commands take the form:

command <source>,<destination>

The result is always in the <destination> operand.

Copy

Parameters - <any>,<var> or <any>,*<C-BUS>

Description – Read the C-BUS register address, variable or constant <any> and copy it to the variable <var> or alternatively to the C-BUS register address *<C-BUS>

Example Usage –

```
copy    #1234, bar           ;bar is assigned 1234d
copy    *$E0, bar           ;Read C-BUS register E0h and put the result
                                ;in bar
copy    bar, foo[1]         ;Copy the value in bar to buffer at index1
copy    bar, *$E0           ;Copy data in bar to C-BUS register E0h
copy    *$E4, *$E5         ;loopback data in C-BUS
```

Restrictions – Variables are not typed so use caution when using signed values or other formats. When *<C-BUS> is used as either or both operands, **device** determines which C-BUS connector is active. The last example cannot be used to copy data from one C-BUS device to another because only one device is active at any time.

Copy is the only arithmetic and logic operations command that accepts *<C-BUS> as a destination in its parameter list. The rest of the commands in this section are identical to **add** below. Also, the destination for **add** and the following commands can only be a <var>.

Add, Sub

Parameters - <any>,<var>

Description – Read the C-BUS register address, variable or constant <any> and add to (or subtract from) the value stored in variable <var>. The result is stored in <var>

Example Usage –

```
add    #1, bar                ;bar is incremented by 1
add    *$E0, bar              ;Read C-BUS register E0h and add to
                               ;the value in bar
add    bar, foo[1]            ;Add the value in bar to buffer at
                               ;index 1
```

Restrictions – Variables are not typed so use caution when using signed values or other formats. Variables can only store 16-bits and arithmetic overflows are ignored.

And, Or, Xor

Parameters - <any>,<var>

Description – Read the C-BUS register address, variable or constant <any> and logically AND (OR or XOR) with the value stored in variable <var>. The result is stored in <var>. The **and**, **or** and **xor** functions are particularly useful for masking variables to determine bit settings, set specific bit patterns or to test for bit changes.

Example Usage –

```
;*****
;Subroutine, ReadStat, to read Ready Flag in Status register
;*****
ReadStat
Status word 0                ;Variable status is instantiated
ReadyFlag    word 0          ;Status of Data Ready flag

    copy    *$E0, Status      ;Read C-BUS register E0h and copy to Status
    and #0040, Status         ;zero all bits except b6
    jmpc Status = 0, NotSet    ;ReadyFlag shadows C-Bus
    copy #1, ReadyFlag        ;register E0h
    return
NotSet
    copy #0, ReadyFlag
    return
```

Restrictions – Variables are not typed so use caution when using signed values or other formats.

Lsl, Lsr, Asl, Asr

Parameters - <any>,<var>

Description -

lsl – Logical Shift Left
lsr – Logical Shift Right
asl – Arithmetic Shift Left
asr – Arithmetic Shift Right

All four commands shift the bits stored in variable <var> in the direction stated (either left or right) by the number of places stored in the variable or constant <any>. Bits shifted right beyond b0 are discarded. For logical shifts, bits shifted left beyond b15 are discarded. For Arithmetic shifts, bits shifted beyond b14 are discarded but b15 (the sign bit) is always preserved. The result is stored in <var>.

Example Usage - None

Restrictions – Variables are not typed so use shift with caution when using signed values and other formats.

9.3 Streaming C-BUS

Read, Write

Parameters:

Read - *<C-BUS>, <var>, <any>

Write - <var>, *<C-BUS>, <any>

Description – These two commands allow streaming accesses between C-BUS registers and variables. Three parameters are required; an address of the C-BUS register <C-BUS>, an array, indexed with the start location of the data <var> and a length parameter <any>. The number of data 'items' transferred is equal to the length parameter. Data 'items' are stored or read consecutively beginning at the array index. The array index is automatically incremented. When an 8-bit C-BUS register is read, the least-significant byte of the variable is used to store the data. The most-significant byte is padded with zeroes. When an 8-bit C-BUS register is written, only the least-significant byte of the variable is read.

Example Usage - See Script Examples, Example 3

Restrictions – Reads or writes to an array are not checked to be within range. If the specified index is outside the array boundary, then it may produce unexpected results. A fixed-size memory pool is assigned for variables, including arrays. When the script is running, a check is made to ensure that the available pool is not exceeded. If it is, the script will terminate with an error message.

9.4 Program flow

Throughout this section, zero equates to FALSE, any non-zero value equates to TRUE.

Jmp

Parameters - <label>

Description – Jump Always. Start executing script from label <label>. (The current PC value is replaced by the script address marked by label <label>).

Example Usage - None

Restrictions – None

Jmpc

Parameters - <any> <cond> <any>, <label>

Description – Jump on Condition. Evaluate the condition <cond> and if the result is TRUE, continue script execution from label <label>. The parameters <any> are not affected by the evaluation. If the result is FALSE, continue script execution from the next script line. (The current PC value is replaced by the script address marked by label <label> only if the condition <cond> evaluates to a non-zero value.).

Example Usage – See Script Examples, Example 4

Restrictions – The script environment assumes only unsigned integer values are used. Be cautious when using evaluators as there is no concept of values less than zero.

Jsr

Parameters - <label>

Description – Jump to Subroutine. Continue script execution from **label** <label>. (The current PC value is pushed onto the stack and the PC loaded by the script address marked by label <label>).

Example Usage – See Script Examples, Example 4

Restrictions – There can be as many **return** commands as the programmer wishes but if the script encounters a return before a **jsr** or **jsrc** have been taken, the stack will underflow. This will terminate the script execution and an error message will be displayed.

Setvect

Parameters - #<const>, <label>

Description – Set the vector for a device interrupt or countdown timer timeout (when it reaches 0). When an interrupt occurs on the device connected to the device interface specified by the value <const>, script execution will continue from the label <label>. The <const> value 3 is reserved for the countdown timer. When the countdown timer reaches zero, script execution will continue from the label <label>. The script interrupt handler must have been previously turned on by **intson**. The script interrupt handler can be disabled by the command **intsoff**. There is no limit to the number or the placing of **intson** and **intsoff** in the script.

Example Usage - See Script Examples, Example 5

Restrictions – <const> can only be 1, 2 or 3. Any other value will cause unpredictable effects. There should only be one instance of **setvect** for each device number. If additional instances occur only the last one declared in the script will be used. Ie.

```
Setvect 1 ISR1          ;Set up an interrupt vector
..
..                    ;some other script lines
Setvect 1 ISR2          ;Set up a second interrupt vector
Setvect 3 ISR3          ;Set up a countdown timer interrupt vector
ISR1                   ;This ISR will never be taken
..                    ;because the second incidence of
..                    ;setvect 1 supercedes it.
rfi

ISR2                   ;This ISR will be taken when
..                    ;device 1 interrupts
..
rfi

ISR3                   ;This ISR will be taken when the countdown timer
..                    ;reaches 0
..
..
rfi
```

Script interrupt handling is automatically disabled when a jump to the interrupt vector occurs so it is not possible for a second interrupt to interrupt the current ISR. This is true for both interrupts so IRQN1 cannot pre-empt IRQN2, for example.

Intson should not be used if an interrupt vector has not been defined by **Setvect**.

Try to limit the number of script lines in the ISR. This makes the ISR exit more quickly and can avoid getting stuck in the ISR.

Intson, intsoff

Parameters - None

Description – Turn on and off script interrupt handling. See **setvect** above.

Example Usage - See Script Examples, Example 5

Restrictions – **Intson** should not be used if an interrupt vector has not been defined at some point in the script.

Rfi

Parameters – None

Description – Return from Interrupt. Start executing script from the line after the line that the jump to the ISR was taken from and re-enable interrupts. (The current PC value is replaced by the value popped off the stack). See **setvect** above.

Example Usage - See Script Examples, Example 5

Restrictions – There can be as many instances of **rfi** as required but if the script encounters an **rfi** while it is not executing an ISR, the stack will underflow. This will terminate the script execution and an error message will be displayed.

If

Parameters - <any> <cond> <any> or <any>

Description – Evaluate the condition <cond> and if the result is TRUE, continue script execution until the next **else** or **elseif**, then jump to the matching **endif**. If the result of the evaluation is FALSE, jump directly to the next matching **else** or **elseif**. If there is no matching **else** or **elseif**, then jump directly to the next matching **endif**. The parameters <any> are not affected by the evaluation.

If the conditional statement is not present, then read the C-BUS register address, variable or constant <any>. The value determines whether or not the jump is taken exactly as if the condition had been evaluated. **If** can be nested, together with **elseif** where required, to any depth. Brackets can be used around the conditional evaluation for clarity.

Example Usage –

```
if foo != 0                                ;is the same as
if (foo != 0)
```

Restrictions – The script environment assumes only unsigned integer values are used. Be cautious when using evaluators as there is no concept of values less than zero.

Elseif

Parameters - <any> <cond> <any> or <any>

Description – Evaluate the condition <cond> and if the result is TRUE, continue script execution until the next matching **else** or **elseif**, then jump immediately to the matching **endif**. If the result of the evaluation is FALSE, jump directly to the next matching **else** or **elseif**. If there is no matching **else** or **elseif**, then jump directly to the next matching **endif**. The parameters <any> are not affected by the evaluation. **Elseif** can be nested, together with **if** where required, to any depth.

Example Usage – None

Restrictions – The script environment assumes only unsigned integer values are used. Be cautious when using evaluators as there is no concept of values less than zero.

Else, Endif

Parameters - None

Description – Markers to enclose the script bound to **if** and **elseif**

Example Usage - None

Restrictions – None

While, Endwhile

Parameters - <any> <cond> <any> or <any>

Description – Do While Condition is True. Evaluate the condition <cond> and if the result is TRUE, execute the script between the **while** and the matching **endwhile**. The condition is evaluated after each pass and the enclosed script executed until the condition evaluates to FALSE. Script execution will then continue at the line after the matching **endwhile**. The parameters <any> are not affected by the evaluation.

If the conditional statement is not present, then read the C-BUS register address, variable or constant <any>. The value determines whether or not the enclosed script (between **while** and **endwhile**) is executed exactly as if the condition had been evaluated. **While** can be nested to any depth. Brackets can be used around the conditional evaluation for clarity.

Example Usage –

```
while foo != 0                ;is the same as
while (foo != 0)

while(1)                      ;do forever
```

Restrictions – The script environment assumes only unsigned integer values are used. Be cautious when using evaluators as there is no concept of values less than zero.

Stop

Parameters - None

Description – Terminates script execution. A stop is required at some point in all scripts that are not intended to loop forever.

Example Usage – See Script Examples

Restrictions – None

9.5 Static Buffers

Bufcfg

Parameters - <any24a> <any24b> or <any24>

Description – This command allocates one or two buffers for use. If one buffer is required <any24> sets the size of the buffer. If two buffers are required <any24a> sets the size of buffer 1 and <any24b> sets the size of buffer 2. The maximum total size shared between the two buffers is 0x280000. If a user tries to allocate more space than is available, the system will return an error message. If **bufcfg** is repeated at a later point in the script both buffer pointers will be reset, the Space Count(s) will be set to the new buffer size(s) and the Data Count(s) will be set to zero. This will effectively clear the old buffer(s) contents. Before the buffers can be accessed the **bufopenr** or **bufopenw** commands must be used to reopen the buffers.

The buffers and their respective properties will be preserved between script runs. For example, a script can configure and write a number of items to a buffer. A second script can be started which will

be able to access that buffer's content or add further content. The Data Count and Space Count variables are also preserved and can be used between multiple scripts.

Example Usage –

```
buf1size    const $1FFF

bufcfg $1FFF                                ;allocates a single buffer,
                                              ;buffer 1, with $1FFF
                                              ;words of space

bufcfg $1FFF, 10000                        ;allocates 2 buffers
                                              ;buffer 1 is $1FFF words in size
                                              ;buffer 2 is 10000d words in size
```

Restrictions – The maximum size shared between the two buffers cannot exceed 0x280000.

Bufopenr

Parameters - <any> or <any>, <var>

Description – This command opens buffer number <any> for reading. This value can only be 1 or 2. The read point is set back to the start of the buffer and the optional Data Count variable <var> is set to the number of items in the buffer. If more than \$FFFF items are in the buffer, \$FFFF will be displayed as this is the maximum value that can be assigned to a variable. . As items are read from the buffer, the data count variable will be decremented. When less than \$FFFF items remain to be read, the Data Count will reflect the actual count.

Example Usage –

```
bufopenr 1 data_left                        ;open buffer 1 for reading, put number of
                                              ;items that can be read in variable
                                              ;'data_left'
```

Restrictions – The buffer number can only be 1 or 2. Before the specified buffer can be opened for reading, it must be configured and its size set using the **bufcfg** command. This could be done by a previous script. The amount of data that can be read from a buffer at any time is limited by the size of the destination. The Data Count variable must not be written to by the script.

Bufopenw

Parameters - <any> or <any>, <var>

Description – This command opens buffer number <any> for writing. The read point and write point are both reset which effectively clears all current content. The optional Space Count variable <var> will be set to the maximum number of items that can be written to the buffer. This value will decrease by the number of items written to the buffer. The Space Count cannot indicate more than \$FFFF due to the size limitations of variables. When the amount of space is less than \$FFFF words, Space Count will accurately reflect the remaining space. The Space Count variable must not be written to by the script.

Example Usage –

```
bufopenw 2 space_left                      ;open buffer 2 for writing, put number of
                                              ;items that can be written in variable
                                              ;'space_left'
```

Restrictions – The buffer number can only be 1 or 2. Before the buffer can be opened for write, it must first be configured and its size must be set using the **bufcfg** command. This could be done by a previous script. The amount of data written to the buffer at any time is limited by the size of the source parameter.

Bufread

Parameters - <any1>, <var> or <any1, <var>, <any2> or
<any1>, *<C-BUS> or <any1>, *<C-BUS>, <any2>

Description – The command transfers a maximum of \$FFFF words from the buffer <any1> to variable <var>, (for example, an array), or C-BUS register *<C-BUS>. The optional count const or variable <any2> will set the number of words to copy and if this is omitted the operation will default to reading 1 word.

Example Usage –

```
bufread 1 *TX_FIFO 12      ;copy 12 words from static buffer 1 to the
                           ;Tx FIFO, using streaming CBUS
```

Restrictions – Before the buffer can be read from, it must first be configured and its size set using the **bufcfg** command. It must also be already open for read use using the **bufopenr** command. If an attempt is made to read more data from the buffer than is available the script will abort and an error message will be generated.

Bufwrite

Parameters - <any1>, <var> or <any1, <var>, <any2> or
<any1>, *<C-BUS> or <any1, <C-BUS>, <any2>

Description – The command transfers data from the variable <var> (for example an array) or C-BUS register *<C-BUS> to the buffer <any1>. As each item is written to the buffer, the Data Count variable is incremented and the Space Count variable is decremented. . <any1> is buffer 1 or 2 and <var> is the C-BUS address or an array starting at the value set in <var>. <any2> will set the number of words to write and if this is omitted the operation will default to 1 word.

Example Usage –

```
bufwrite 2 rx_data 20      ;copy 20 words from the rx_data array to
                           ;static buffer 2
```

Restrictions – Before the buffer can be written to, it must first be configured and its size set using the **bufcfg** command. It must also be already open for write use using the **bufopenw** command. If an attempt is made to write past the end of the buffer the script will abort and an error message will be generated. Only \$FFFF items can be transferred at a time due to the size limit of the controlling constant or variable <any2>

Filetobuf

Parameters - <any>

Description – transfers an entire file to the buffer. If the size of the file is bigger than the buffer the script will abort and an error message will be produced. While the file transfer is in progress the buffer is locked and the script handling is suspended until the transfer is complete.

Example Usage – TBD

Restrictions – Before the command can execute the file must first be opened for reading using the **fileopenr** command.

Bufile

Parameters - <any>, “str”

Description – transfers the entire buffer <any> to a file. The format of the data transferred is specified by “str”. While the file transfer is in progress the buffer is locked and the script handling is suspended until the transfer is complete.

Example Usage – TBD

Restrictions – Before the command can execute the file must first be opened for writing using the **fileopenw** command. The format specifier “str” above is ignored when writing binary or wav files.

9.6 Timers

There are two timers associated with the scripting language: Time (which is a system timer) and Timer (which is a countdown timer). Time is an incrementing timer, running constantly from zero when the script starts up, and Timer is a decrementing timer (used, for example, for timeouts) which only runs when enabled. Both Time and Timer count in milliseconds and are reasonably accurate. However, precise measurements cannot be guaranteed.

Commands Settime and gettime work with ‘Time’.

Commands settimer, starttimer and stoptimer work with ‘Timer’

Settime

Parameters - <any>

Description – This command sets the Time to <any>. Setting Time to a particular value occurs as the script line is executed. Time will continue incrementing from the value <any> and will continue to \$FFFF, roll over to 0 and continue incrementing. This command is particularly useful for timing events or scheduling a series of events.

Example Usage –

```
settime 0                                ;initialise Time to 0
..                                       ;do something..
..
gettime current_time                    ;read the current value of Time
disp "Time lapse is %u ms" current_time;display how long'something'
                                       ;took to complete
```

Restrictions – There is no signal to indicate when Time rolls over so the script must keep track of this.

Gettime

Parameters - #<var>

Example Usage – See settime

Description – Reads the current value of Time and returns it to the variable <var>.

Settimer

Parameters - <any>

Description – Sets the Timer to an initial value <any>. Timer will not begin decrementing until started with **starttimer**. When Timer decrements to zero (if an interrupt vector has been defined), execution will continue from the interrupt vector defined by **setvect3**.

Example Usage –

This script fragment sets up Timer as timeout while waiting for the PF flag to set in the IRQ Status register. Without the timeout, the script will wait forever if the PF flag does not get set.

```
settimer 5                                ;set Timer to 5ms
starttimer                               ;and start it

wait_on_PFlag                            ;Timer is running
jmpc *IRQ_STATUS !& $4000 wait_on_PFlag ;Poll the Status register
                                         ;until the PF flag is set or
                                         ;Timer times out (=0)

stoptimer

Timer_Int                                ;Timer has timed out
    disp "Timeout waiting for PF flag"   ;so display a message
    jmp shutdown                         ;and shutdown cleanly
    rfi
```

Restrictions – The maximum value for Timer is \$FFFF or just over 6.5 seconds.

Starttimer

Parameters – None

Description – Enables the countdown timer. Timer then decrements every millisecond until it reaches 0 and then stops.

Example Usage – See *settimer*

Restrictions – none

Stoptimer

Parameters – None

Description – Freezes Timer at the current value. **Starttimer** will then restart the countdown from its current value.

Example Usage – See *settimer*

Restrictions – none

9.7 Miscellaneous

There are two ports in the IO space of the PE0002, Port 1 and Port 2.

When the GUI is first started all the configurable I/O lines are set as inputs. If a script command changes the state or direction of an I/O pin, the setting will be maintained even when the script completes or is aborted. This is helpful where the test/evaluation environment needs to be maintained while scripts are cascaded. The default direction and state will only be restored when the GUI is closed and restarted. Inputs directly signal the state of the connector pins to which they are connected – there is no inversion.

Port 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IO15	IO14	IO13	IO12	IO11	IO10	IO9	IO8	GPIO3	GPIO2	GPIO1	GPIO0	DO4	DO3	DO2	DO1

- b15-8 I/O lines connected to Device 2 – J3. Can be configured as input or output (default is input). Target kits may use these pins and require their states to be defined. Check the relevant kit's user manual.
These pins can only be used on PE0002 RevB boards. For earlier PE0002 boards, read as 0.
- b7-4 General purpose I/O lines connected to J6. These can be configured as input or output (default is input). The external connections are open circuit and will drift if configured as inputs without external hardware pre-setting the state.
- b3-0 Outputs connected to LED bank. Always configured as output. These pins are connected via open-drain, current sinks. Writing a 1 to the bits will turn the respective LED on. Default state is 0 (LED off).

Port 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	IO7	IO6	IO5	RS232\C-BUS	BOO TEN2	BOO TEN1	IRQN 2	IRQN 1	IO4	IO3	IO2	IO1	IO0

- b15-13 Not available. Writing the port configuration or writing the state of these bits will have no effect. These bits will always read as 0.
- b12-b10 I/O lines connected to Device 1 - J5. Can be configured as input or output (default is input). Target kits may use these pins and require their states to be defined. Check the relevant kit's user manual.
- b9 RS232/C-BUS mode selection output line connected to Device 1 - J5 and Device 2 - J3. Always configured as output. Default state is 0.
- b8-b7 Boot mode selection output lines connected to Device 1 - J5 and Device 2 - J3. Always configured as outputs. Default state is 0.
- b6-5 Interrupt input lines connected to Device 1 - J5 and Device 2 - J3. Interrupts normally signal active low – therefore 0 = Interrupt. Always configured as inputs.
- b4-0 I/O lines connected to Device 1 - J5 and Device 2 - J3. Can be configured as input or output (default is input). Target kits may use these pins and require their states to be defined. Check the relevant kit's user manual.

Port

Parameters - <any>

Description – Selects the currently active port <any>.. Only values of 1 or 2 are valid.

Example Usage – None

Restrictions – None

Getport

Parameters - <var>

Description – Getport <var> reads the currently selected port into the variable <var>

Portc

Parameters - <any>

Description – Read a configuration word from C-BUS register address, variable or constant <any> and copy it to the current Port's data direction register. A 1 in a bit position sets the corresponding bit in the port to be an INPUT. A 0 sets it to be an OUTPUT. See the tables above for mapping of bits to pins.

Example Usage – None

Restrictions – Changes to unused bits will be ignored. Changes to bits with a fixed direction will be ignored.

Portw

Parameters - <any>

Description – Read a configuration word from C-BUS register address, variable or constant <any> and copy it to the current Port's hardware output pins.

Example Usage – None

Restrictions – Target kits may use these pins to determine start-up or running conditions. Certain bit states may have to be pre-defined or appropriately defined in use. Check the relevant kit's user manual for any restrictions.

Portr

Parameters - <var> or *<C-BUS>

Description – Read the current port and copy the value to variable <var> or C-BUS register address *<C-BUS>.

Example Usage – None

Restrictions – Target kits may use these pins to determine start-up or running conditions. Certain bit states may have to be pre-defined or appropriately defined in use. Check the relevant kit's user manual for any restrictions. Unconnected pins will drift and return arbitrary values.

Ones

Parameters - <any>,<var>

Description – Count the number of ones in the value read from C-BUS register address, variable or constant <any> and put the result in variable <var>.

Example Usage –

ones	#\$0000, count	; count = 0
ones	#\$1111, count	; count = 4
ones	#\$1248, count	; count = 4
ones	#\$aa55, count	; count = 8
ones	#\$ffff, count	; count = 16

Restrictions – None

Device

Parameters - <any>

Description – Selects the currently active Device (C-BUS pins on Connector J3 or J5) <any> Only values of 1 or 2 are valid.

Example Usage – Example 2

Restrictions – None

Getdevice

Parameters - <var>

Description – Getdevice <var> reads the currently selected device into the variable <var>

Delay

Parameters - <any>

Description – Pauses script for a number of milliseconds equal to value <any>.

Example Usage –

```
delay 10 ; pause script processing for 10ms
```

Restrictions – None

Microdelay

Parameters - <any>

Description – Pauses script for a number of microseconds equal to the value <any> multiplied by 10.

Example Usage –

```
microdelay 10 ;suspend script processing for approx 100us
```

Restrictions – Microdelay is provided to obtain a shorter or less granular delay than **delay** provides. The resolution of **microdelay** is not guaranteed and can be greatly affected by time consuming tasks such as co-incident USB activity. Do not use for critical timing delays.

Register

Parameters - #<const1>, #<const2>, #<const3>

Description – C-BUS registers comprise an address byte followed by zero or more data bytes. The **register** command allows the user to select a device (port <const1>) and assign C-BUS registers (<const2>) connected to that port with the number of data bytes (<const3>) that follows the address byte.

If undeclared, C-BUS registers will be assumed to require two data bytes after the address byte. Thus only C-BUS registers that require 0 data bytes (normally only General Reset) or 1 data byte, need be declared.

Example Usage –

```
register 1, $01, 0 ;Set C-BUS port $01 on Device 1
;to receive zero data bytes
;following the address byte
```

Note that the second parameter, \$01, in the register command is a constant that represents a C-BUS register. The * that infers a C-BUS address must not be used or a build error will occur.

Restrictions – Device can only be 1 or 2. Setting a C-BUS register to receive an incorrect number of data bytes may cause unexpected operation.

Logon, Logoff

Parameters - None

Description – Start or stop logging C-BUS transactions. Logon turns on a C-BUS tracing feature that records all C-BUS transactions, with a timestamp, until a logoff is encountered. There can be more than one incidence of logon and logoff in the script. When the script completes or is aborted, the “See Trace” button on the GUI is enabled. When clicked, this button uploads the Log from the PE0002 Trace Memory. See the PE0002 User Manual for a description of this facility.

Example Usage – None

Restrictions – Logging places an extra load on the PE0002 and will cause the script to run more slowly. This may cause buffer overflows or underflows that will not occur if logging is not enabled.

10 Error Messages

10.1 De-bug Messages

When a script is aborted the current line number is reported back to the GUI. The console will display script aborted at line number *n*.

10.2 Build Errors

Build errors are reported in the script log window and show the script line number where the problem was encountered. An error may occur in a line preceding the line number at which the error is reported so check back from the line number given. In the case of missing **endif/endwhile** commands the line number for the unmatched **if/while** is given. The error messages are listed below:

- Unrecognised command
- **If/while** without matching **endif/endwhile**
- Unresolved label(s)
- Command requires *n* parameters
- Parameter type not allowed
- Duplicate label

10.3 Runtime Errors

Runtime errors that occur on the PE0002 are always reported in the script log window and give the value of the PC where the problem was encountered.

10.3.1 PC out of range

This means that the PC (i.e. the address of the next instruction) is not within the script. Check for unexpected **jmp**, **jsr** or **return** commands.

10.3.2 Invalid opcode

This means that the PE0002 is trying to execute a command it doesn't recognise.

10.3.3 Data index out of range

A fixed-size memory pool is assigned for variables, including arrays. A check is made to ensure that the available pool is not exceeded. This message signals that the variable addressed has exceeded the available pool size. Probably caused by excessive use of the `foo[bar++/--]` syntax. This message also occurs if the variable name used in a loop control argument has been misspelled or has not been declared. Usually the error line number returned will not be associated with the error.

10.3.4 Stack overflow

The stack is used to hold return addresses when **jsr** or **jsrc** commands are used. Overflow usually indicates repeatedly using **jsr** without a matching return. An overflow also occurs if jumps are used in loops or subroutines. Jumping leaves call data on the stack and eventually the stack will overflow.

10.3.5 Stack underflow

See Stack overflow above. Underflow usually results from using return without having first used **jsr**.

10.3.6 Micro text out buffer overflow

This is an error in the PE0002 buffer used to transfer data from the PE0002 to the GUI. An overflow indicates that data is being sent too fast. Commands that send data via this buffer include **disp** and **filew** so try inserting delays around these. Note that the actual strings are not sent over this link so reducing their length will not help. If script delays can be tolerated, use **waituplink**.

10.3.7 Micro File read buffer underflow

This is an error in the PE0002 File Buffer used to transfer data from the PE0002 to the GUI and will not normally underflow. Data may underflow if the script uses the data faster than the GUI can supply it. Use the data more slowly by inserting delays or use the **waitfile** command. Also ensure that the number of host PC applications running are minimised and disconnect other USB devices that are not necessary.

10.3.8 Micro File read buffer overflow

This is an error in the PE0002 File Buffer used to transfer data from the GUI to the PE0002. The amount of data in the File Buffer is monitored and managed by the PE0002 in such a way that it should never overflow. See advice on Micro File read buffer underflow above.

10.3.9 Attempt to read past end of file

An attempt has been made to read more data than is available from a file on the host PC. This may be an indexing error or not checking for data of the correct format. Use the variable in the **fopenr** to check that the file is not empty and then to control the number of read cycles executed.

10.3.10 Invalid Jump Destination

This means that the value which a **jmp**, **jsr** return, if or while command is trying to write to the script program counter (i.e. the address of the next instruction) is not within the script.

10.3.11 Attempt to reference non-existent string

This means that the PE0002 has passed up a string reference that the GUI doesn't recognise. The GUI couldn't open the file specified. Check access permissions.

10.3.12 Unable to open file

The GUI couldn't open the file specified. Check the file exists and that filename and path are correct. Also check the permissions allow access for reading or writing as appropriate.

10.3.13 Error - RX Queue is full.

This is an error in the host PC buffer used to transfer data from the PE0002 to the GUI. An overflow indicates that data is being sent too fast. Commands that send data via this buffer include **disp** and **filew**. Limit the use of **disp** and try inserting delays around **filew** to find the source of errors. Note that the contents of the string are not sent over this link, so reducing the length of the strings will not help.

10.3.14 Static buffers allocation exceeds allowed size

The size of both buffers specified by **bufcfg** has exceeded the amount of memory available. Make them smaller. The PE0002 has 0x280000 words available.

10.3.15 Attempt to write past end of static buffer

Bufwrite has gone past the size specified in **bufcfg**. Write less, or specify a bigger buffer. Or remember to reset the write pointer with **bufopenw**.

10.3.16 Attempt to read past end of static buffer

Bufread has gone past the valid data in the buffer. Read less, or write more. Or remember to reset the read pointer with **bufopenr**.

10.3.17 Static buffer not open for reading/writing

This means that **bufread/bufwrite** or **buftofile/filetobuf** have been used without first opening the buffer with **bufopenr** or **bufopenw**.

10.3.18 Uplink buffer is smaller than requested space

This means that **waituplink** has requested more space than the size of the buffer. The buffer on the PE0002 can hold 200 messages.

10.3.19 UART overflow / USB comms error / USB queue full /Serial framing error

These all relate to problems with the USB link between the PE0002 and the GUI. Sending less data may help, or using a faster PC.

10.3.20 Other possible errors

Ensure that all incidences of a variable or constants are spelt identically including case. These are NOT tested in loops or conditional expressions during compilation. A misspelt variable or constant is effectively a new item but it will not have been declared. The scripting tool will exit abnormally at runtime when it tries to process the misspelt item.

Ensure that evaluations are complete. Use of **elseif** without a condition will NOT be identified during compilation. The scripting tool will exit abnormally at runtime when it tries to process the evaluation.

Script Examples

10.4 Example 1

```
;*****  
;Send a General Reset to device 1  
;*****  
GEN_RESET      const $01  
  
    register    1, GEN_RESET, 0      ;Set C-BUS port $01 on Device 1  
                                         ;to receive zero data bytes  
                                         ;following the address byte  
    copy #0 *GEN_RESET              ;Send General Reset  
    stop                                           ;End of script
```

10.5 Example 2

```
;*****  
;Copy a block of data from one device to another  
;Consecutive reads of a data block are not possible on all CML devices  
;*****  
bar          buffer 10  
index        word   0  
  
    device 1                                ;Select device 1  
    copy #0, index  
    while index < #10  
        copy  *$B5, bar[index++] ;Read from C-BUS, store in bar[n]  
                                         ;and inc index  
    endwhile  
  
    device 2                                ;Select device 2  
    copy #0, index  
    while index < #10  
        copy  bar[index++], *$A7 ;Read from bar[n] store to C-BUS  
                                         ;and inc index  
    endwhile  
    stop                                           ;End of script
```

10.6 Example 3

```
;*****  
;Copy from one device to another using streaming C-BUS  
;*****  
bar          buffer 10  
RxDat        const $B5  
TxDat        const $A7  
  
    device 1                                ;Select device 1  
    read  *RxDat, bar[0], #10              ;Read from C-BUS, store in bar[n]  
  
    device 2                                ;Select device 2  
    write bar[0], *TxDat, #10              ;Read from bar[n] store to C-BUS  
    stop                                           ;End of script
```

10.7 Example 4

```
;*****
;Read from a file into a C-BUS address, controlled by flags
;*****
count      word  1
status     word  1
temp       word  1

      fopenr "inputfile.txt","%04X"      ;Tell host PC to open file & stream
                                           ;data down
      cls                                ;Clear log window
      copy  #0, count                    ;Set up loop counter
      while count < #32                  ;Start while loop
          filer temp                      ;Read from file buffer
          write temp,$A7                  ;Write to C-BUS
          jsr  read_and_wait              ;Wait for status flag
          add  #1, count                  ;Increment counter
          disp "Written %d values.\n", count
                                           ;Write to log window
      endwhile                          ;End of while loop
      dialog "Finished"                  ;Send end to message box
      stop

read_and_wait
      copy  *$C1, status                  ;Read C-BUS address $C1 into
                                           ;variable
      and   #$0010,status                ;Mask off the wanted bits
      jmpc  status != #$0010, read_and_wait ;If bit 4 not set, try again.
      return
```

Note that if data is consumed by the script (**filer** in **while** loop) at a rate faster than the PC can supply it, a buffer underflow may occur – see section 10.3.8. for further information.

10.8 Example 5

```
;*****
;This script is to test an encode from the vocoder in the EV8610 kit.
;Vocoder mode is set from an external file, CodecSetup.txt file, which
;can be found at the end of this example.
;Interrupt driven.
;The sound source is from a PC sound card.
;*****

;Variables
CodeMethod  word  #$37                    ;read external file to set these to
                                           ;something other than default
FrameSize   word  #$1B                    ;External file format:
                                           ;CodeMethod <CR> FrameSize <CR>

count       word  0                      ;G/P counter
Port2cpy    word  0                      ;Copy of Port2
Status      word  0                      ;Shadow Status
templ       word  0                      ;G/P register
Frame       buffer 144                   ;buffer to read streaming C-Bus. Could be
                                           ;up to 144 bytes
FrameCount  word  0                      ;Counter to count the number of frames
                                           ;saved to disc

;Register defines
```

```

register #1 #$09 #1      ;Powersave device, reg, parameter length
register #1 #$07 #1      ;VCFG-Vocoder Configuration
register #1 #$2E #1      ;SVCACK - Service Acknowledge
register #1 #$05 #1      ;AIG - Analogue Input Gain
register #1 #$06 #1      ;AOG - Analogue Output Gain
register #1 #$30 #1      ;ENCFRAME # - Encoded frame

fopenr "CodecSetup.txt" "%02x"      ;Get vocoder mode from external file
filer CodeMethod
filer FrameSize

setvect 1 WaitIRQn1      ;Set interrupt 1 vector

;Open the file for the encoded frames - the PC takes a short time to open ;files and
pass back handles
fopenw "Vocoder.smp"

;Configure the CMX618
jsr GenReset      ;Reset the CMX618
intson      ;turn on the interrupts
jsr WaitRDY      ;wait until RDY flag is set
copy #$03 *$09      ;Codec/Bias=Enable
copy #$0F *$05      ;MicAmp=0dB, IPGain=22.5dB
copy #$8001 *$1F      ;unmask VDA IRQ (RDY already unmasked by
                    ;default)
copy CodeMethod *$07      ;default - HardCoding, FEC=Enable,
                    ;2400bps, 3x20msFrames
jsr WaitRDY      ;Wait for configuration to complete
jsr SvcAck      ;Service acknowledged? SvcAck checks can be
                    ;removed once code is stable.
delay 100      ;Wait for VBIAS to settle

copy #$0002 *$11      ;Turn on encoder
jsr WaitRDY      ;This is another service
jsr SvcAck

disp "Recording Started"
copy #0 FrameCount
Recording
jsr WaitVDA      ;wait for an encoded sample
read *$30 Frame[0] FrameSize      ;load Frame buffer with 27 bytes
                    ;(default)
copy #0 count      ;reset counter
while count < FrameSize      ;write the buffer to the PC file
    filew "%02x" Frame[count++]
endwhile
add #1 FrameCount      ;update the frame counter
jmpc FrameCount < 200 Recording      ;Set the maximum recording length
                    ;here (200x60ms by default)
EndRecording
copy #$0000 *$11      ;Turn off vocoder
jsr WaitRDY      ;Another service
jsr SvcAck
intsoff      ;Turn off the interrupts
disp "Recording Stopped"
disp "%d Frames stored" FrameCount

stop

;*****Wait on IRQn1*****
;ISR. When the IRQn pin goes low on the CMX618 this ISR will run
WaitIRQn1

```

```

        copy *$40 Status                ;Read Status
        rfi

;*****

;*****wait for flags*****
;Wait here for a interrupt to set the appropriate flag
WaitRDY
        copy Status templ
        and #$8000 templ
        jmpc templ != #$8000 WaitRDY
        and #$7FFF Status                ;Clear the Ready flag in shadow
        return

WaitVDA
        ;
        copy Status templ
        and #$0001 templ
        jmpc templ != $0001 WaitVDA
        and #$FFFE Status                ;Clear the VDA flag in shadow
        return

;*****

;*****General Reset*****
GenReset
RESET  const $01
        register #1 RESET #0
        copy #0 *RESET
        return

;*****

;*****Read Ack from SVCACK*****
;Poll here until the SVC flag is set.
SvcAck
        copy *$2E Status                ;Copy SVCACK reg to status
        and #$0001 Status                ;mask for ACK flag
        jmpc Status != #$0001 SvcAck     ;loop until SVC flag set
        return

;*****

;*****
CodecSetup.txt file
;*****
;This external text file is used to set the required codec format.
;Format:
;  CodeMethod (hex)
;  FrameSize (hex)
;Remove the ';' at the beginning of the line for the two values required.
;Only the first two incidences matching the fopenr format will be used.
;The other codec formats can be easily added except those using soft bit coding.

;2050bps, 1x20ms
;01
;06

;2400bps, 3x20ms
37
1B

```

```
;2750bps, 3x20ms, FEC, Hard bits
;3B
;1B
```

```
;2050, 4x20ms, FEC, Hard bits
;30
;24
```

```
;2750, 4x20ms, FEC, Hard bits
;38
;24
```

10.9 Example 6

```
;*****
;This script fragment uses the ones command in a bit error rate (BER) test.
;A copy of the being data sent from the far-end modem is in the file ;data.txt. This
file is read and compared to the data received by the modem. ;Any bit differences are
flagged as ones (bit=1) which can be counted.
;*****
count          word      0
rx_data        word      0
file_data      word      0
error_bits     word      0
tot_error_bits word      0

      fopen      "data.txt", "%04X"          ;Open file with expected data.

      while (count < 100)                    ;Do 100 words.
        while (*$AB != 1)                   ;Wait for modem to indicate
                                          ;word received.
          endwhile

          copy *$12, rx_data                ;Read word from modem.
          filer file_data                   ;Read word from file.
          xor file_data, rx_data             ;Compare.
          ones rx_data, error_bits          ;Count bits which are
                                          ;different.

          add error_bits, tot_error_bits    ;Keep running total.
          add #1, count                     ;Increment count
        endwhile

      disp      "Received 100 words, %d bits in error", tot_error_bits

      stop
```

CML does not assume any responsibility for the use of any algorithms, methods or circuitry described. No IPR or circuit patent licenses are implied. CML reserves the right at any time without notice to change the said algorithms, methods and circuitry and this product specification. CML has a policy of testing every product shipped using calibrated test equipment to ensure compliance with this product specification. Specific testing of all circuit parameters is not necessarily performed.

 CML Microcircuits (UK) Ltd COMMUNICATION SEMICONDUCTORS	 CML Microcircuits (USA) Inc. COMMUNICATION SEMICONDUCTORS	 CML Microcircuits (Singapore) Pte Ltd COMMUNICATION SEMICONDUCTORS
Tel: +44 (0)1621 875500 Fax: +44 (0)1621 875600 Sales: sales@cmlmicro.com Tech Support: techsupport@cmlmicro.com	Tel: +1 336 744 5050 800 638 5577 Fax: +1 336 744 5054 Sales: us.sales@cmlmicro.com Tech Support: us.techsupport@cmlmicro.com	Tel: +65 67450426 Fax: +65 67452917 Sales: sg.sales@cmlmicro.com Tech Support: sg.techsupport@cmlmicro.com
- www.cmlmicro.com -		