

FI Reference	608/618/638FI_1.x
--------------	-------------------

1 Introduction

This application note describes a Voice Storage and Retrieve (VSR) function that can be uploaded to the CMX608, CMX618 and CMX638 RALCWI¹ Vocoder ICs.

- The VSR function is supplied as a Function ImageTM that can be stored in the host microcontroller's flash memory for uploading into the RALCWI Vocoder following device initialisation.
- PE0002 scripts are provided for easy evaluation using the RALCWI Vocoder evaluation kits, EV6180 or EV6380, and the PE0002 Interface Card.

The VSR option equips the vocoder device with the ability to store and playback previously vocoded speech. Speech is stored in encoded form in on-chip RAM at the same time as it is decoded. Once stored, it is available for playback until either deleted or the device is reset.

The amount of memory available for speech storage depends on the features supplied in the Function ImageTM. There will be a minimum of 40 seconds of storage available.

Up to 8 storage buffers may be used, to store 8 discrete periods of speech. The memory for these buffers is allocated dynamically in 1-second blocks as the speech is being stored. The device will indicate when all available memory has been used up.

The VSR option is controlled by one sixteen-bit write register, one sixteen-bit read register, and an extra interrupt / status-bit in the status register.

2 C-BUS registers

2.1 VSRW register address \$19

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	Buffer			0	0	0	0	Command			

This write-only register controls all aspects of the VSR feature. The 4 least significant bits specify a command that the sub-system should perform. Bits 8 through 10 inclusive specify the buffer to be used. Bits 4 through 7 inclusive and 11 through 15 inclusive are reserved and should be set to 0.

After this register has been written, bit 15 of the STATUS register (\$40) will be set and, if enabled, IRQN will go low. The SVCACK register (\$2E) will contain a flag in bit 0 indicating whether the write to this register was successful. If this bit is set, the action was successful, if clear, then the action failed.

Commands that return information to the controlling host, will place that information in the VSRR register (\$39). This register may be read after bit 15 of the STATUS register has been set, and SVCACK register indicates that the action was successful.

¹ RALCWI is a CML digital voice technology

Command	Mnemonic	Description
\$0	VSR_RESET	Resets the VSR subsystem. This command should be the first one issued after downloading the feature. This restores all internal data structures to a known default state thereby preparing the feature for use. The buffer field should be set to \$00 for correct operation. This command will fail if the device is currently recording or playing back.
\$1	VSR_ERASE_ALL	Erase all buffers, returning any used memory to the main pool for re-use. The buffer field should be set to \$00 for correct operation This command will fail if the device is currently recording or playing back.
\$2	VSR_ERASE	Erase a single buffer, returning used memory to the main pool for re-use. The buffer field should specify which buffer is to be erased. This command will fail if the buffer specified is currently being used to record or play back.
\$3	VSR_PSTART	Start playing the contents of the buffer specified in the buffer field. Playback will start at the beginning of the buffer. If, whilst playing, the end of the buffer is reached, then playback will stop and bit 3 of the STATUS register (\$40) will be set and, If enabled, IRQN will go low. This command will fail if the device is already recording or playing back.
\$4	VSR_PSTOP	Stop playing the currently playing buffer. The current playing position is saved so playback may be resumed. This command will fail if the device is not currently playing back.
\$5	VSR_PRESUME	Start playing the contents of the buffer specified in the buffer field. Playback will start from where it was last stopped. If, whilst playing, the end of the buffer is reached, then playback will stop and bit 3 (VSR) of the STATUS register (\$40) will be set and, If enabled, IRQN will go low. This command will fail if the device is already recording or playing back.
\$6	VSR_RSTART	Start saving vocoder packets in the buffer specified in the buffer field. If the buffer was not empty when this command is sent, then the vocoder packets will be appended to the buffer. If, whilst recording, no more memory is available, recording will stop and bit 3 (VSR) of the STATUS register (\$40) will be set and, If enabled, IRQN will go low. This command will fail if the device is already recording or playing back.
\$7	VSR_RSTOP	Stop the current recording operation. This command will fail if the device is not currently recording.
\$8	VSR_FREE	Return the amount of free space available for storage in the VSRR register (\$39). The value returned is the number of 20 millisecond frames that remain in the free memory pool. This value should be divided by 50 to obtain the number of seconds available.

Command	Mnemonic	Description
\$9	VSR_LENGTH	Return the amount of speech stored in the buffer specified in the buffer field. The value returned in the VSRR register (\$39) is the number of 20 millisecond frames that have been recorded.
\$A	VSR_STATUS	Returns a status value in VSRR register (\$39) indicating whether the VSR subsystem is recording, playing back, or idle. The active buffer is also included where the status indicates that it is not idle. This command is always successful.

2.2 VSRR register address \$39

[illegible]

This read-only register contains the results returned by some of the VSR commands. For commands that return a frame count, this can be read as a 16 bit unsigned value.

The VSR STATUS command returns two flags and a buffer number.

Bit 15 : R : This flag is set to 1 if the device is recording. The buffer field indicates which is the active buffer.

Bit 14 : P : This flag is set to 1 if the device is playing. The buffer field indicates which is the active buffer.

When both bits 15 and 14 are clear, the buffer value has no meaning.

Bits 13 down to 3 should be ignored.

2.3 STATUS register address \$40 & IRQENAB register address \$1F

The VSR option adds one new status bit and corresponding IRQ enable bit. This is bit 3 VSR.

When recording, this bit will be set when all available memory has been used up and recording has been stopped.

When playing back, this bit will be set when the end of the buffer has been reached and playing has been stopped. Please note that audio may take up to 20ms after this bit has been set to stop playing. This is because the output buffer needs time to empty.

3 Description

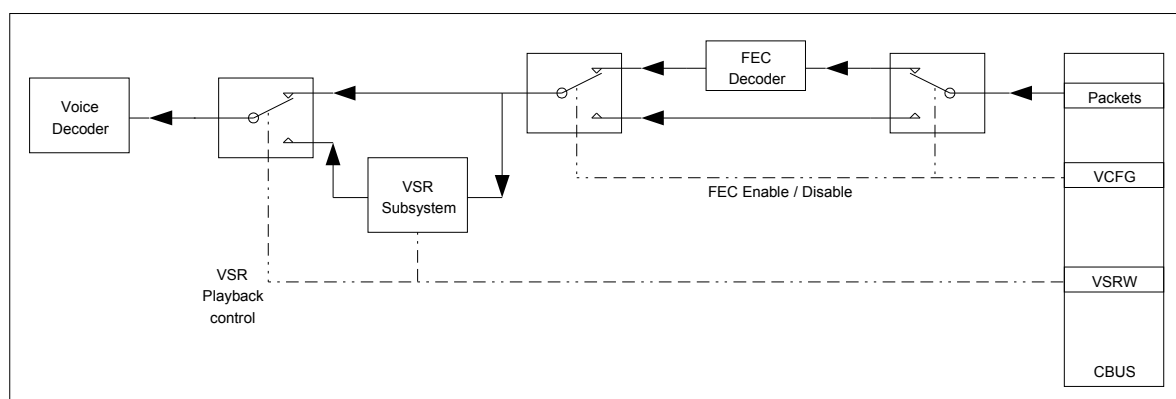


Figure 1. VSR enhancements to existing decoder half of the RALCWI Vocoder

The VSR functions are an enhancement to the decoding part of the device. The encoder half of the device is not affected.

VSR works by intercepting vocoder frames after the FEC decoder, but before the vocoder's decoder. The VSR functions work irrespective of the vocoder packet size, the FEC mode, or the bit rate.

When recording, before giving the frames to the decoder, the device takes a copy of each frame's data and puts it in memory associated with the desired buffer. The memory is allocated in 1 second chunks as and when it is required. As well as the encoded speech, the results from the FEC decoder that enable PLC are also stored, providing PLC is enabled prior to recording. The bit rate, however, is not saved. The device must be set to the same bit rate for the recorded speech to be intelligible when played back.

When playing back, the vocoder should be set up to decode as normal, but instead of supplying packets of data to decode, the host should tell the vocoder to play the packets stored in one of the buffers. Once playback has finished, packets may again be supplied by the host for decoding.

When playing back, If enabled, STD and DTMF detection will behave in the same way as if the packets were supplied by the host.

Saved speech remains in memory until either it is explicitly deleted, the device or the VSR subsystem is reset, or the device is power cycled. Any recorded speech will still be available after the device had been put into deep power save and then reactivated.

4 Example

The following sequence of instructions detail how to load the Function Image™ into the device, and then, how to use it to store and play back messages. The whole sequence is contained within a single script for the PE0002 which can be used in conjunction with an EV6180 or EV6380 evaluation kit.

4.1 Running the example script:

Connect the CBUS connector (J3) of an EV6180 or EV6380 to the CBUS 1 connector a PE0002. Connect some powered speakers or a headset to the CODEC output connector (J10). The encode part of the device is unused in this example, so no sound source need be connected to the CODEC input connector. Connect up a suitable power supply to both boards and turn the supply on.

Assuming that the PE0002 software has been successfully installed and that the USB connection has been made, start up the PE0002 executable (ES0002xx.exe) and, using the *script handler* tab, click on the *select script* button, and choose the demonstration script *vsrdemo1.pes*

Click on the *run* button to start the script.

The script will reset the vocoder device, download the VSR Function Image™, reset the VSR subsystem, and show how many frames of storage are available. In addition to this, a listing of all the buffers is done, showing them all to be empty. The script will pause at this stage, click OK to continue.

When the script resumes, the device will be setup to decode packets of single raw vocoder frames with a bit rate of 2400bps. A set of 9 pre-encoded speech samples will be played into the vocoder for decoding. Each speech sample will be stored into a different vocoder buffer as well as being decoded and reproduced in the attached speaker/earphone. The 9th sample will be appended to the first buffer (buffer 0). The script will then pause again, click OK to continue.

After resuming, the script will display how many frames of storage are now available, together with a listing of all the buffers showing that each buffer contains a varying number of frames. The script will pause again, click OK to continue

Once running again, the script will playback the samples stored in each buffer. It will do this in the reverse order; buffer 7 down to 0. Part way through playing buffer 0, the script will pause. Click OK to continue the script

When continued, the script will play the remaining part of buffer 0, by using the VSR_PRESUME command. Once finished, the script will stop.

The speech samples will remain stored in the buffers. Further scripts may be run, that just play back the contents of these buffers, providing the device is not reset. A second script has been supplied, *vsrdemo2.pes* that does this. This script plays the buffers in a different order.

4.2 Detailed look at the example script *vsrdemo1.pes*

A full listing of the main demonstration script appears at the end of this document. Please refer to this whilst reading the following explanation rather than the script included in the kit. There may be small

variations in the script that is supplied in the kit due to improvements etc..., and the version included in this document has been annotated with line numbers to aid the following explanations.

The script is split into three main sections. The first section, from line 011 to line 169, defines symbolic names for many of the CBUS registers in the vocoder, and the values that these registers can take. Additionally, a few variables are declared and the widths of the non 16-bit CBUS registers are declared (The PE0002 assumes registers are 16 bits wide unless declared otherwise).

The second section, from line 170 to line 438, implement the previously described demonstration. Most of this section comprises calls to subroutines defined in the last section of the script, with the setting of any associated variables and messages that are displayed as the demonstration proceeds. Each subsection of the sequence has been preceded by comments which indicate what the section does.

Lines	Explanation
183 – 190	Check that the vocoder device is functioning with the PE0002 by calling the subroutine <i>CheckForDevice</i> . This subroutine will stop the script if it fails to communicate with the vocoder. This is followed by establishing an interrupt handler to deal with the vocoder's IRQN signal. The vocoder will signal that a task has completed, by pulling IRQN low, which in turn, causes the interrupt handler to run.
196-207	Download the Function Image™ to the device and then initialise the VSR function by using the VSR_RESET command. All command writes to the VSRW register (\$19) are followed by a call to the <i>WaitRDY</i> subroutine. When a VSR command has finished executing, it signifies this by setting bit 15 of the STATUS register (\$40) and (if enabled) pulling IRQN low. The <i>WaitRDY</i> subroutine waits for this condition to occur. Using the VSR_FREE command, find out how much storage space is actually available. After a VSR_RESET, the minimum value reported by this command will be 2000, which is the number of frames in 40 seconds (50 frames per second). The actual value returned depends on the size of the Function Image™, which in turn, depends on the number of functions provided. A call to the <i>VsrBufferStats</i> subroutine is then made, to show that all buffers are currently empty.
213-223	Open a file of vocoder frames. Configure the vocoder for the correct bit rate, packet size etc... Turn on the decoder and wait for the device to ready.
228-320	Play back 8 sections of speech and at the same time, save these sections in the vocoder's 8 buffers. For each section, the buffer number is placed in the variable <i>VssBuffer</i> and the number of 20ms frames is placed in the <i>VdfAmount</i> variable. These variables are used by the following 2 subroutine calls which set the device up for saving and then send the frames to the device for decoding. Once the frames have been sent, saving is then stopped.
325-331	Play back the ninth section of speech, and at the same time, save the section in buffer 0. This will append the section of speech to buffer 0. So, buffer 0 will then hold the first and ninth sections of speech.
333-348	Stop the device decoding, display the remaining free space available, and list out how much speech is stored in each of the buffers.
357-362	Change the IRQ enables so that RDY and VSR cause interrupts. VSR is required, as it indicates when a buffer has finished playing.
367-418	Playback the saved speech in each of the buffers. Each section is essentially the same; place the buffer number in the variable <i>VspBuffer</i> , call the subroutine to start the playback, and then call the subroutine that waits until the VSR bit in the STATUS register is set, indicating that playback has finished.

Lines	Explanation
423-431	Playback the speech saved in buffer 0, with a pause and resume. After starting the playback, delay for 3½ seconds and then stop the playback. After resuming the script, the script resumes playback of buffer 0, and waits for the rest of the speech to be played.
436-437	Turn off interrupts and stop the script. The device will be left in the position where it can replay speech from the buffers by using another script that does not reset the device etc... A second script <i>vsrdemo2.pes</i> has been included that does this.

The third section of the script consists of the subroutines that control the vocoder device. Most of the CBUS reads and writes needed to exercise the VSR functions are encapsulated in these subroutines. All variables used in subroutines are declared within the subroutine and have names that make them unique within the script. This is because all variables in the scripting language are global. Because expressions are not permitted for the source argument of a *copy*, to make it clear which bits are being set in a register, a temporary variable is used in order to set bits by ORing.

Lines	Explanation
460-493	VsrMode2400SingleVdw: Configure the device to accept single frame packets and to notify when more data is required. Set IRQENAB (\$1F) to enable RDY and VDW interrupts. Set CLOCK (\$1D) to 5 and enable the setting by writing to DTMFATTEN (\$0A) Set POWERSAVE (\$09) to enable BIAS and the on-board CODEC Set VCFG (\$07) to configure the device for 2400bps and single frame packets with no FEC. At the same time, store the frame size (6 bytes) into the variable <i>VdfFrameSize</i>. This is used by the subroutine that sends encoded voice to the device. Wait for the RDY status bit to be set, indicating that the configuration has finished Read SVCACK (\$2E) and check the configuration was successful. Delay for 100ms – this is to ensure that BIAS has settled Set the high watermark for VDW notification to 150 samples and the low watermark to 130 samples. These values are suitable for single frame packets. Set the analogue output gain to 0 dB.
513-533	VsrDecodeFrames: Send in a number of single frame packets for decoding. This subroutine expects a file of vocoder data to be opened prior to use. The file should be a text file containing a 2 digit hexadecimal number (representing a byte) on each line. For each frame, 8 bytes are used. For 2400bps, only 6 bytes are required, so the other 2 bytes are ignored. A loop is used to read bytes from the data file, then send them out to the vocoder. 8 bytes are read from the file, <i>VdfFrameSize</i> bytes are written to the vocoder's DECFRAME (\$10) register using the C-BUS streaming <i>write</i> command. After sending the data into the vocoder, the script waits for the VDW bit to be set in the STATUS register. Finally, the loop counter is incremented and tested against <i>VdfAmount</i>, which was set before the subroutine was called. Whilst the counter is still less than <i>VdfAmount</i>, jump back to the top of the loop.

Lines	Explanation
548-651	VsrResumePlaying, VsrStartPlaying, VsrStopPlaying, VsrStartSaving & VsrStopSaving: All these subroutines are wrappers for sending VSR commands to the device. The stop-playing and stop-saving do not require a buffer number, so only copy the VSR command to VSRW (\$19) and then wait for the RDY bit to be set in the STATUS register. The other 3 subroutines require a buffer number – each subroutine has a variable for holding this. The buffer number is copied to a temporary variable. This is then shifted to the left by 8 bits (VSR_PAR_SHIFT). The VSR command is then ORed into this temporary variable, which in turn, is copied to VSRW (\$19). The script then waits on the RDY bit to be set, before reading SVCACK (\$2E) to verify that the command was successful.
665-689	VsrBufferStats: Display a list of all 8 buffers and the amount stored in each. This subroutine is just a loop that, for each buffer from 0 to 7, copies a VSR_LENGTH request with the buffer number to VSRW (\$19), waits for the RDY bit to be set in the STATUS register, and if the command was successful, reads the number of frames from VSRR (\$39). The values, with explanatory text, are then written to the results area.
705-745	VsrSendImage: Send a Function Image™ to the vocoder device. This subroutine implements the procedure outlined in section 6.7 of the data sheet. A version of the VSR Function Image™ has been produced for this example in a form that the script can read. Like the vocoder frames, the image data is presented as a text file containing a 2-digit hexadecimal number (representing a byte) on each line. The file containing the Function Image™ data must be opened prior to calling this subroutine. The loop tests how much data is left and, if there is at least 128 bytes (PRSIZE) left, then another 128 byte packet is read out of the file into a buffer. This in turn is written, using the C-BUS streaming <i>write</i> command, to DECFRAME (\$10). The script then waits for the SVC bit to be set in STATUS before checking whether the packet has been accepted. If, at any time in the Function Image™ download, the device indicates that there is a problem, the script will stop.
759-794	CheckForDevice: Check that a vocoder device is connected to CBUS #1 This subroutine checks that a vocoder device is connected to the PE0002 by writing test values to two of the vocoder's registers that have readable mirrors. First, 8-bit CBUS writes are executed using the write register VCFG (\$07) which are mirrored and should be read back from MVCFG (\$2C). Second, 16-bit CBUS writes are executed using the write register VCTRL (\$11) which are mirrored and should be read back from MVCTRL (\$3C). After these writes have been carried out, and checked, the device will be reset via a CBUS RESET command.

Lines	Explanation
808-849	WaitVDW, WaitVSR, WaitRDY & WaitSVC: Wait for status bits to be set. All these subroutines operate in the same way. They enter a loop reading the gStatus value, until the bit they are waiting for becomes set. These routines rely on the interrupts being enabled in the vocoder (IRQENAB) and the interrupt handler being established on the PE0002. Once set, the bit that was being waited for is cleared.
864-868	HandleIrqn1: Interrupt handler When IRQN goes low signifying an interrupt, this routine reads the vocoder's STATUS register (\$40) and ORs the value read into the gStatus variable. This value can then be used by the rest of the script to indicate whether a status bit has been set. Typically, this will be the WaitAAA routines above.

4.3 Appendix - Listing of vsrdemo1.pes script for PE0002

```

001 ; ++++++
002 ; +   vsrdemo1.pes : CMX 618/638 VSR demonstration script for PE0002
003 ; +
004 ; +   Application support                CML Microcircuits (UK) Ltd
005 ; +
006 ; +   9th of June 2008                  ix.vi.mmviii
007 ; +
008 ; +   Revision A                        Issue 1
009 ; ++++++
010
011 ; *****
012 ; *****
013 ; *
014 ; *   Script preamble - define constants, variables and register widths
015 ; *
016 ; *****
017 ; *****
018
019
020 ; ++++++
021 ; +   0 bit CBUS write register names
022 ; ++++++
023 VR_RESET      const   $01                ; CBUS General reset
024 VR_SYNC       const   $02                ; SYNC command
025
026 ; ++++++
027 ; +   8 bit CBUS write register names and bit names
028 ; ++++++
029 VR_SYNCCTRL   const   $04                ; SYNC control
030 VR_AIG        const   $05                ; Analogue input gain
031 VR_AOG        const   $06                ; Analogue output gain
032
033 VR_VCFG       const   $07                ; Vocoder configuration
034
035 VR_VCFG_1FRAME const   $01
036 VR_VCFG_2FRAME const   $02
037 VR_VCFG_3FRAME const   $03
038 VR_VCFG_4FRAME const   $00
039 VR_VCFG_FRAME  const   $03
040 VR_VCFG_RESERVE const   $00
041 VR_VCFG_2400BPS const   $04
042 VR_VCFG_2750BPS const   $08
043 VR_VCFG_BPS    const   $0c
044 VR_VCFG_FEC    const   $10
045 VR_VCFG_HDD    const   $20
046 VR_VCFG_DTX    const   $40
047 VR_VCFG_DTMFF  const   $80
048
049 VR_POWERSAVE   const   $09                ; Powersave
050 VR_PS_BIAS     const   $01
051 VR_PS_CODEC    const   $02
052 VR_PS_DACON    const   $04
053 VR_PS_ADCON    const   $08
054 VR_PS_THROTTLE const   $10

```

```

055
056 VR_DTMFATTEN      const   $0a           ; DTMF attenuation
057 VR_SVCREQ         const   $0e           ; Service request
058 SVCREQ_DLOAD      const   $01
059
060 VR_DECFRAME        const   $10           ; Decode frames
061
062 ; ++++++
063 ; + 16 bit CBUS write register names and bit names
064 ; ++++++
065 VR_VCTRL           const   $11           ; Vocoder control
066 VR_VCTRL_DEC       const   $0001
067 VR_VCTRL_ENC       const   $0002
068 VR_VCTRL_FECLOOP   const   $0004
069 VR_VCTRL_PLC       const   $0008
070 VR_VCTRL_DVDW      const   $0010
071 VR_VCTRL_DDTMFD    const   $0020
072 VR_VCTRL_DDTMFG    const   $0040
073 VR_VCTRL_DSTDD     const   $0080
074 VR_VCTRL_DSTDG     const   $0100
075 VR_VCTRL_DSTDP     const   $0200
076 VR_VCTRL_BIT10     const   $0400
077 VR_VCTRL_EDTMFD    const   $0800
078 VR_VCTRL_ESTDD     const   $1000
079 VR_VCTRL_ESTDP     const   $2000
080 VR_VCTRL_BIT14     const   $4000
081 VR_VCTRL_BIT15     const   $8000
082
083 VR_VSRW            const   $19           ; VSR write
084 VSR_CMD_SHIFT      const   0
085 VSR_PAR_SHIFT      const   8
086 VSR_CMD_MASK       const   $000f
087 VSR_PAR_MASK       const   $ff00
088 VSR_BUF_MASK       const   $0007
089
090 VSR_RESET          const   $00
091 VSR_ERASE_ALL      const   $01
092 VSR_ERASE          const   $02
093 VSR_PSTART         const   $03
094 VSR_PSTOP          const   $04
095 VSR_PRESUME        const   $05
096 VSR_RSTART         const   $06
097 VSR_RSTOP          const   $07
098 VSR_FREE           const   $08
099 VSR_LENGTH         const   $09
100 VSR_STATUS         const   $0a
101
102 VR_CLOCK            const   $1d           ; Clock
103 VR_CLOCK_SLOW      const   $05
104 VR_CLOCK_FAST      const   $06
105
106 VR_VDWHLWM         const   $1e           ; High and low watermarks
107 VR_VDW_HIGH        const   $8000
108
109 VR_IRQENAB         const   $1f           ; IRQ enable
110

```

```

111 ; ++++++
112 ; + 8 bit CBUS read registers
113 ; ++++++
114 VR_MVCFG          const   $2c          ; Mirror of configuration
115 VR_SVCACK         const   $2e          ; Service acknowledgement
116 SVCACK_OK         const   $01
117
118 VR_ENCFRAME        const   $30          ; Encode frames
119
120 ; ++++++
121 ; + 16 bit CBUS read registers
122 ; ++++++
123 VR_VSRR           const   $39          ; VSR read
124
125 VR_MVCTRL          const   $3c          ; Mirror of control
126
127 VR_STATUS          const   $40          ; Status
128 VS_VDA             const   $0001
129 VS_EFDA            const   $0002
130 VS_DFDA            const   $0004
131 VS_VSR             const   $0008
132
133 VS_VDW             const   $0100
134
135 VS_SVC              const   $4000
136 VS_RDY             const   $8000
137
138 ; ++++++
139 ; + Other handy values
140 ; ++++++
141 PRSIZE             const   128          ; Image record size
142 DRSIZE             const    8          ; Data record size
143 ; ++++++
144 ; + Script variables
145 ; ++++++
146 gStatus            word     0          ; Global status register shadow
147
148 temp               word     0          ; Temporary variable
149 FileSize           word     0
150

```

```
151 ; ++++++
152 ; +   Register non-16 bit CBUS registers to correct data width
153 ; ++++++
154 register    #1,VR_RESET,#0
155 register    #1,VR_SYNC,#0
156
157 register    #1,VR_SYNCCTRL,#1
158 register    #1,VR_AIG,#1
159 register    #1,VR_AOG,#1
160 register    #1,VR_VCFG,#1
161 register    #1,VR_POWERSAVE,#1
162 register    #1,VR_DTMFATTEN,#1
163 register    #1,VR_SVCREQ,#1
164 register    #1,VR_DECFRAME,#1
165
166 register    #1,VR_MVCFG,#1
167 register    #1,VR_SVCACK,#1
168 register    #1,VR_ENCFRAME,#1
169
170 ; *****
171 ; *****
172 ; *
173 ; *   Main script sequence.
174 ; *
175 ; *****
176 ; *****
177
178
179 ; ++++++
180 ; +   Check vocoder device is attached to the CBUS, establish an
181 ; +       interrupt handler, and then turn on interrupts.
182 ; ++++++
183 disp        "Script starting, checking device is present"
184 jsr         CheckForDevice
185 disp        "Device seems to be present"
186
187 setvect     1,HandleIrqn1           ; Install interrupt handler
188 copy        *$40,temp               ; Clear any pending interrupt
189 copy        #0,gStatus              ; Clear global status shadow
190 intson                      ; Enable interrupts
191
```

```
192 ; ++++++
193 ; +   Send in VSR function image, reset the VSR subsystem and then
display
194 ; +       some statistics.
195 ; ++++++
196   fopenr      "fi6x8_1000.ped","%02x",FileSize
197   jsr         VsrSendImage
198
199   copy        VSR_RESET,*VR_VSRW      ; Reset VSR subsystem
200   jsr         WaitRDY
201
202   copy        VSR_FREE,*VR_VSRW      ; Request free space
203   jsr         WaitRDY
204   copy        *VR_VSRR,temp
205   disp        "Vocoder reports %d frames of storage available",temp
206
207   jsr         VsrBufferStats          ; Do a buffer report
208
209 ; ++++++
210 ; +   Establish the source of frames for decoding, configure the
vocoder,
211 ; +       and then enable the decoder with Vocoder Data Wanted
notification.
212 ; ++++++
213   dialog      "Start playback and speech storage"
214
215   fopenr      "data2400.vrf"."%02x",FileSize
216
217   jsr         VsrMode2400SingleVdw    ; Configure the vocoder
218
219   copy        VR_VCTRL_DEC,temp
220   or          VR_VCTRL_DVDW,temp
221   copy        temp,*VR_VCTRL          ; Start decoder
222   jsr         WaitRDY
223
224 ; ++++++
225 ; +   First section of speech.
226 ; +       Save into buffer 0
227 ; ++++++
228   copy        #0,VssBuffer
229   copy        #135,VdfAmount
230   disp        "Storing %d frames \c",VdfAmount
231   disp        "to buffer %d",VssBuffer
232   jsr         VsrStartSaving
233   jsr         VsrDecodeFrames
234   jsr         VsrStopSaving
235
236
```

```
237 ; ++++++
238 ; +   Second section of speech.
239 ; +       Save into buffer 1
240 ; ++++++
241     copy        #1,VssBuffer
242     copy        #125,VdfAmount
243     disp        "Storing %d frames \c",VdfAmount
244     disp        "to buffer %d",VssBuffer
245     jsr         VsrStartSaving
246     jsr         VsrDecodeFrames
247     jsr         VsrStopSaving
248
249 ; ++++++
250 ; +   Third section of speech.
251 ; +       Save into buffer 2
252 ; ++++++
253     copy        #2,VssBuffer
254     copy        #140,VdfAmount
255     disp        "Storing %d frames \c",VdfAmount
256     disp        "to buffer %d",VssBuffer
257     jsr         VsrStartSaving
258     jsr         VsrDecodeFrames
259     jsr         VsrStopSaving
260
261 ; ++++++
262 ; +   Fourth section of speech.
263 ; +       Save into buffer 3
264 ; ++++++
265     copy        #3,VssBuffer
266     copy        #145,VdfAmount
267     disp        "Storing %d frames \c",VdfAmount
268     disp        "to buffer %d",VssBuffer
269     jsr         VsrStartSaving
270     jsr         VsrDecodeFrames
271     jsr         VsrStopSaving
272
273 ; ++++++
274 ; +   Fifth section of speech.
275 ; +       Save into buffer 4
276 ; ++++++
277     copy        #4,VssBuffer
278     copy        #135,VdfAmount
279     disp        "Storing %d frames \c",VdfAmount
280     disp        "to buffer %d",VssBuffer
281     jsr         VsrStartSaving
282     jsr         VsrDecodeFrames
283     jsr         VsrStopSaving
284
```

```
285 ; ++++++
286 ; +   Sixth section of speech.
287 ; +       Save into buffer 5
288 ; ++++++
289     copy        #5,VssBuffer
290     copy        #155,VdfAmount
291     disp        "Storing %d frames \c",VdfAmount
292     disp        "to buffer %d",VssBuffer
293     jsr         VsrStartSaving
294     jsr         VsrDecodeFrames
295     jsr         VsrStopSaving
296
297 ; ++++++
298 ; +   Seventh section of speech.
299 ; +       Save into buffer 6
300 ; ++++++
301     copy        #6,VssBuffer
302     copy        #150,VdfAmount
303     disp        "Storing %d frames \c",VdfAmount
304     disp        "to buffer %d",VssBuffer
305     jsr         VsrStartSaving
306     jsr         VsrDecodeFrames
307     jsr         VsrStopSaving
308
309 ; ++++++
310 ; +   Eighth section of speech.
311 ; +       Save into buffer 7
312 ; ++++++
313     copy        #7,VssBuffer
314     copy        #150,VdfAmount
315     disp        "Storing %d frames \c",VdfAmount
316     disp        "to buffer %d",VssBuffer
317     jsr         VsrStartSaving
318     jsr         VsrDecodeFrames
319     jsr         VsrStopSaving
320
321 ; ++++++
322 ; +   Ninth section of speech.
323 ; +       Appended to buffer 0
324 ; ++++++
325     copy        #0,VssBuffer
326     copy        #140,VdfAmount
327     disp        "Storing %d frames \c",VdfAmount
328     disp        "to buffer %d",VssBuffer
329     jsr         VsrStartSaving
330     jsr         VsrDecodeFrames
331     jsr         VsrStopSaving
332
333     copy        #0,*VR_VCTRL                ; Turn off decoder
334     jsr         WaitRDY
335
```

```
336 ; ++++++
337 ; +   Show stats
338 ; ++++++
339 dialog      "Show VSR statistics"
340 disp        ""
341 disp        ""
342 copy        VSR_FREE,*VR_VSRW
343 jsr         WaitRDY
344 copy        *VR_VSRR,temp
345 disp        "Vocoder reports %d frames of storage available",temp
346 jsr         VsrBufferStats
347 disp        ""
348 disp        ""
349
350 ; ++++++
351 ; +   Playback the saved speech.
352 ; +
353 ; +   The vocoder device is still configured for the desired bit rate
354 ; +       All that needs doing is to unmask the VSR status bit and
355 ; +       set the vocoder back to decoding, without the VDW enabled
356 ; ++++++
357 dialog      "Playback previously saved speech"
358 copy        VS_RDY,temp
359 or          VS_VSR,temp
360 copy        temp,*VR_IRQENAB      ; Interrupt on RDY or VSR
361 copy        VR_VCTRL_DEC,*VR_VCTRL ; Start decoder
362 jsr         WaitRDY
363
364 ; ++++++
365 ; +   Play buffer 7
366 ; ++++++
367 copy        #7,VspBuffer
368 disp        "Playback from buffer %d",VspBuffer
369 jsr         VsrStartPlaying
370 jsr         WaitVSR
371
372 ; ++++++
373 ; +   Play buffer 6
374 ; ++++++
375 copy        #6,VspBuffer
376 disp        "Playback from buffer %d",VspBuffer
377 jsr         VsrStartPlaying
378 jsr         WaitVSR
379
380 ; ++++++
381 ; +   Play buffer 5
382 ; ++++++
383 copy        #5,VspBuffer
384 disp        "Playback from buffer %d",VspBuffer
385 jsr         VsrStartPlaying
386 jsr         WaitVSR
387
```

```
388 ; ++++++
389 ; +   Play buffer 4
390 ; ++++++
391 copy      #4,VspBuffer
392 disp      "Playback from buffer %d",VspBuffer
393 jsr       VsrStartPlaying
394 jsr       WaitVSR
395
396 ; ++++++
397 ; +   Play buffer 3
398 ; ++++++
399 copy      #3,VspBuffer
400 disp      "Playback from buffer %d",VspBuffer
401 jsr       VsrStartPlaying
402 jsr       WaitVSR
403
404 ; ++++++
405 ; +   Play buffer 2
406 ; ++++++
407 copy      #2,VspBuffer
408 disp      "Playback from buffer %d",VspBuffer
409 jsr       VsrStartPlaying
410 jsr       WaitVSR
411
412 ; ++++++
413 ; +   Play buffer 1
414 ; ++++++
415 copy      #1,VspBuffer
416 disp      "Playback from buffer %d",VspBuffer
417 jsr       VsrStartPlaying
418 jsr       WaitVSR
419
420 ; ++++++
421 ; +   Play buffer 0 with a pause and resume
422 ; ++++++
423 copy      #0,VspBuffer
424 disp      "Playback from buffer %d",VspBuffer
425 jsr       VsrStartPlaying
426 delay     #3500
427 jsr       VsrStopPlaying
428 dialog    "Playback paused - click OK to continue"
429 copy      #0,VrpBuffer
430 jsr       VsrResumePlaying
431 jsr       WaitVSR
432
433 ; ++++++
434 ; +   Clean up and stop script
435 ; ++++++
436 intsoff
437 stop
438
```

```

439 ; *****
440 ; *****
441 ; *
442 ; *      Support subroutines
443 ; *
444 ; *****
445 ; *****
446
447 ; ++++++
448 ; +      Set up the device to encode/decode single raw frames at 2400bps.
449 ; +      Sets up VDW notification and the high and low watermarks.
450 ; +
451 ; +      Requires:
452 ; +      VsrDecodeFrames subroutine with its variables must be present
453 ; +
454 ; +      Example:
455 : +      jsr      VsrMode2400SingleVdw
456 ; +
457 ; +      Results:
458 ; +      On error, displays message and stops the script
459 ; ++++++
460 VmsvTemp      word      0
461 VsrMode2400SingleVdw
462      copy      VS_RDY,VmsvTemp
463      or        VS_VDW,VmsvTemp
464      copy      VmsvTemp,*VR_IRQENAB          ; Interrupt on RDY or VDW
465      copy      VR_CLOCK_SLOW,*VR_CLOCK
466      copy      #0,*VR_DTMFATTEN
467      delay     #1
468      copy      VR_PS_BIAS,VmsvTemp
469      or        VR_PS_CODEEC,VmsvTemp
470      copy      VmsvTemp,*VR_POWERSAVE        ; Turn on the BIAS and CODEC
471      copy      VR_VCFG_1FRAME,VmsvTemp
472      or        VR_VCFG_2400BPS,VmsvTemp
473      copy      VmsvTemp,*VR_VCFG              ; Write the configuration
474      copy      #6,VdfFrameSize                ; 2400bps yields 6 byte frames
475      jsr      WaitRDY
476      copy      *VR_SVCACK,VmsvTemp            ; Check configuration was
accepted
477      and      SVCACK_OK,VmsvTemp
478      jmpc     VmsvTemp == SVCACK_OK, VmsvWait
479      disp     "Vocoder did not accept configuration"
480      stop
481 VmsvWait
482      delay     #100                            ; Allow BIAS to settle
483
484      copy      #150,VmsvTemp                    ; Set high water mark
485      or        VR_VDW_HIGH,VmsvTemp
486      copy      VmsvTemp,*VR_VDWHLWM
487      jsr      WaitRDY
488      copy      #130,*VR_VDWHLWM                ; Set low water mark
489      jsr      WaitRDY
490
491      copy      #7,*VR_AOG                        ; Set output gain to 0dB
492
493      return
494

```

```

495 ; ++++++
496 ; +   Send a number of raw vocoder frames to the device for decoding.
497 ; +
498 ; +   Requires:
499 ; +       File of vocoder frames to be opened with fopenr using "%02x"
format
500 ; +       File item count to be placed in the variable 'FileSize'
501 ; +       Variable 'VdfAmount' to hold the number of frames to send
502 ; +       Variable 'VdfFrameSize' to hold number of bytes in the frame
503 ; +
504 ; +   Example:
505 ; +       fopenr  "data2400.rvf", "%02x", FileSize
506 ; +       copy    #6, VdfFrameSize
507 ; +       copy    #120, VdfAmount
508 ; +       jsr     VsrDecodeFrames
509 ; +
510 ; +   Results:
511 ; +       Displays message if it runs out of frames
512 ; ++++++
513 VdfAmount      word    0
514 VdfCount       word    0
515 VdfIndex       word    0
516 VdfFrameSize   word    6
517 VdfFrame       buffer  8
518 VsrDecodeFrames
519     copy        #0, VdfCount
520 VdfLoop
521     jmpc        FileSize >= 8, VdfHaveFrame
522     disp        "Ran out of frames"
523     return
524 VdfHaveFrame
525     copy        #0, VdfIndex
526     while       VdfIndex < 8
527         filer    VdfFrame[VdfIndex++]
528     endwhile
529     write        VdfFrame[0], *VR_DECFRAME, VdfFrameSize
530     jsr          WaitVDW
531     add          #1, VdfCount
532     jmpc        VdfCount < VdfAmount, VdfLoop
533     return
534

```

```

535 ; ++++++
536 ; +   Resume playing speech in a buffer
537 ; +
538 ; +   Requires:
539 ; +       Variable 'VrpBuffer' to hold buffer number to use
540 ; +
541 ; +   Example:
542 ; +       copy    #4,VrpBuffer
543 ; +       jsr     VsrResumePlaying    ; Resume playback from buffer 4
544 ; +
545 ; +   Results:
546 ; +       On error, displays message and then stops script
547 ; ++++++
548 VrpBuffer    word    0
549 VrpTemp      word    0
550 VsrResumePlaying
551     copy      VrpBuffer,VrpTemp
552     lsl       VSR_PAR_SHIFT,VrpTemp
553     or        VSR_PRESUME,VrpTemp
554     copy      VrpTemp,*VR_VSRW
555     jsr       WaitRDY
556     copy      *VR_SVCACK,VrpTemp
557     jmpc      VrpTemp == SVCACK_OK,VrpStarted
558     disp      "Vocoder reported error on PRESUME"
559     stop
560 VrpStarted
561     return
562
563 ; ++++++
564 ; +   Start playing speech in a buffer
565 ; +
566 ; +   Requires:
567 ; +       Variable 'VspBuffer' to hold buffer number to use
568 ; +
569 ; +   Example:
570 ; +       copy    #3,VspBuffer
571 ; +       jsr     VsrStartPlaying    ; Start playing from buffer 3
572 ; +
573 ; +   Results:
574 ; +       On error, displays message and then stops script
575 ; ++++++
576 VspBuffer    word    0
577 VspTemp      word    0
578 VsrStartPlaying
579     copy      VspBuffer,VspTemp
580     lsl       VSR_PAR_SHIFT,VspTemp
581     or        VSR_PSTART,VspTemp
582     copy      VspTemp,*VR_VSRW
583     jsr       WaitRDY
584     copy      *VR_SVCACK,VspTemp
585     jmpc      VspTemp == SVCACK_OK,VspStarted
586     disp      "Vocoder reported error on PSTART"
587     stop
588 VspStarted
589     return
590

```

```

591 ; ++++++
592 ; +   Stop playing speech
593 ; +
594 ; +   Requires:
595 ; +       Nothing
596 ; +
597 ; +   Example:
598 ; +       jsr     VsrStopPlaying
599 ; +
600 ; +   Results:
601 ; +       None
602 ; ++++++
603 VsrStopPlaying
604     copy        VSR_PSTOP,*VR_VSRW
605     jsr         WaitRDY
606     return
607
608 ; ++++++
609 ; +   Start saving speech in a buffer
610 ; +
611 ; +   Requires:
612 ; +       Variable 'VssBuffer' to hold buffer number to use
613 ; +
614 ; +   Example:
615 ; +       copy    #2,VssBuffer
616 ; +       jsr     VsrStartSaving      ; Start saving speech to buffer 2
617 ; +
618 ; +   Results:
619 ; +       On error, displays message and then stops script
620 ; ++++++
621 VssBuffer    word    0
622 VssTemp      word    0
623 VsrStartSaving
624     copy      VssBuffer,VssTemp
625     lsl       VSR_PAR_SHIFT,VssTemp
626     or        VSR_RSTART,VssTemp
627     copy      VssTemp,*VR_VSRW
628     jsr       WaitRDY
629     copy      *VR_SVCACK,VssTemp
630     jmpc      VssTemp == SVCACK_OK,VssStarted
631     disp      "Vocoder reported error on RSTART"
632     stop
633 VssStarted
634     return
635

```

```

636 ; ++++++
637 ; +   Stop saving speech
638 ; +
639 ; +   Requires:
640 ; +       Nothing
641 ; +
642 ; +   Example:
643 ; +       jsr     VsrStopSaving
644 ; +
645 ; +   Results:
646 ; +       None
647 ; ++++++
648 VsrStopSaving
649     copy         VSR_RSTOP,*VR_VSRW
650     jsr          WaitRDY
651     return
652
653 ; ++++++
654 ; +   Report buffer usage
655 ; +
656 ; +   Requires:
657 ; +       Nothing
658 ; +
659 ; +   Example:
660 ; +       jsr     VsrBufferStats
661 ; +
662 ; +   Results:
663 ; +       displays all buffer sizes in log window
664 ; ++++++
665 VbsTemp      word    0
666 VbsCount     word    0
667 VsrBufferStats
668     disp       "Buffer report"
669     disp       "-----"
670     copy       #0,VbsCount
671 VbsLoop1
672     copy       VbsCount,VbsTemp
673     lsl        VSR_PAR_SHIFT,VbsTemp
674     or         VSR_LENGTH,VbsTemp
675     copy       VbsTemp,*VR_VSRW
676     jsr        WaitRDY
677     disp       "Buffer %d \c",VbsCount
678     copy       *VR_SVCACK,VbsTemp
679     jmpc       VbsTemp == SVCACK_OK,VbsGotVal
680     disp       "length is not available"
681     jmp        VbsDoneVal
682 VbsGotVal
683     copy       *VR_VSRR,VbsTemp
684     disp       "has %d frames used",VbsTemp
685 VbsDoneVal
686     add        #1,VbsCount
687     jmpc       VbsCount < 8,VbsLoop1
688     disp       "-----"
689     return
690

```

```

691 ; ++++++
692 ; +   Send a function image to the device
693 ; +
694 ; +   Requires:
695 ; +       Image file to be opened with fopenr with "%02x" format
696 ; +       File item count to be placed in the variable 'FileSize'
697 ; +
698 ; +   Example:
699 ; +       fopenr  "func_image.ped","%02x",FileSize
700 ; +       jsr     VsrSendImage
701 ; +
702 ; +   Results:
703 ; +       Subroutine stops script on error
704 ; ++++++
705 VsiTemp      word    0
706 VsiCount     word    0
707 VsiBuffer    buffer  128
708 VsrSendImage
709     copy      #0,gStatus
710     copy      #0,*VR_RESET
711     jsr       WaitRDY
712     disp      "Device reset"
713     copy      VS_RDY,VsiTemp
714     or        VS_SVC,VsiTemp
715     copy      VsiTemp,*VR_IRQENAB
716     copy      SVCREQ_DLOAD,*VR_SVCREQ
717     jsr       WaitSVC
718     copy      *VR_SVCACK,VsiTemp
719     and       SVCACK_OK,VsiTemp
720     jmpc      VsiTemp == SVCACK_OK,VsiSendData
721     disp      "Device has rejected download request"
722     stop
723 VsiSendData
724     jmpc      FileSize < PRSIZE,VsiEndData
725     copy      0,VsiCount
726     while    VsiCount < PRSIZE
727         filer  VsiBuffer[VsiCount++]
728     endwhile
729     write     VsiBuffer[0],*VR_DECFRAME,PRSIZE
730     jsr       WaitSVC
731     copy      *VR_SVCACK,VsiTemp
732     and       SVCACK_OK,VsiTemp
733     jmpc      VsiTemp == SVCACK_OK,VsiSendData
734     disp      "Device rejected data packet"
735     stop
736 VsiEndData
737     copy      SVCREQ_DLOAD,*VR_SVCREQ
738     jsr       WaitSVC
739     copy      *VR_SVCACK,VsiTemp
740     and       SVCACK_OK,VsiTemp
741     jmpc      VsiTemp == SVCACK_OK,VsiEndImage
742     disp      "Device rejected function image"
743     stop
744 VsiEndImage
745     return
746

```

```

747 ; ++++++
748 ; +   Check that the device is present and CBUS works OK
749 ; +
750 ; +   Requires:
751 ; +       Nothing
752 ; +
753 ; +   Example:
754 ; +       jsr       CheckForDevice
755 ; +
756 ; +   Results:
757 ; +       Subroutine displays message and stops script if device not
found
758 ; ++++++
759 CfdTemp      word      0
760 CheckForDevice
761     copy      #$55,*VR_VCFG
762     delay     #10
763     copy      *VR_MVCFG,CfdTemp
764     jmpc      CfdTemp == #$55,CfdPart2
765     disp      "Device does not seem to be there (8bit write/read $55)"
766     stop
767 CfdPart2
768     copy      #$aa,*VR_VCFG
769     delay     #10
770     copy      *VR_MVCFG,CfdTemp
771     jmpc      CfdTemp == #$aa,CfdPart3
772     disp      "Device connection problem (8bit write/read $AA)"
773     stop
774 CfdPart3
775     copy      #0,*VR_RESET
776     delay     #10
777     copy      #$5555,*VR_VCTRL
778     delay     #10
779     copy      *VR_MVCTRL,CfdTemp
780     jmpc      CfdTemp == #$5555,CfdPart4
781     disp      "Device connection problem (16 bit write/read $5555)"
782     stop
783 CfdPart4
784     return
785     copy      #$aaaa,*VR_VCTRL
786     delay     #10
787     copy      *VR_MVCTRL,CfdTemp
788     jmpc      CfdTemp == #$aaaa,CfdPart5
789     disp      "Device connection problem (16 bit write/read $AAAA)"
790     stop
791 CfdPart5
792     copy      #0,*VR_RESET
793     delay     #10
794     return
795

```

```

796 ; ++++++
797 ; + Subroutines to wait for specific status bits.
798 ; +
799 ; + Requires:
800 ; + Variable 'gStatus' to be declared and updated on interrupt
801 ; +
802 ; + Example:
803 ; + jsr WaitRDY ; Wait for ready bit to be set.
804 ; +
805 ; + Results:
806 ; + Clears the bit waited for in gStatus and then returns
807 ; ++++++
808 wTemp word 0
809
810 ; ++++++
811 ; + Wait for VDW - Vocoder Data Wanted
812 ; ++++++
813 WaitVDW
814 copy gStatus,wTemp
815 and VS_VDW,wTemp
816 jmpc wTemp != VS_VDW,WaitVDW
817 xor #$ffff,wTemp
818 and wTemp,gStatus
819 return
820 ; ++++++
821 ; + Wait for VSR - Voice Storage Retrieve
822 ; ++++++
823 WaitVSR
824 copy gStatus,wTemp
825 and VS_VSR,wTemp
826 jmpc wTemp != VS_VSR,WaitVSR
827 xor #$ffff,wTemp
828 and wTemp,gStatus
829 return
830 ; ++++++
831 ; + Wait for RDY - Ready
832 ; ++++++
833 WaitRDY
834 copy gStatus,wTemp
835 and VS_RDY,wTemp
836 jmpc wTemp != VS_RDY,WaitRDY
837 xor #$ffff,wTemp
838 and wTemp,gStatus
839 return
840 ; ++++++
841 ; + Wait for SVC - Service
842 ; ++++++
843 WaitSVC
844 copy gStatus,wTemp
845 and VS_SVC,wTemp
846 jmpc wTemp != VS_SVC,WaitSVC
847 xor #$ffff,wTemp
848 and wTemp,gStatus
849 return
850

```

```
851 ; ++++++
852 ; +   Interrupt handler
853 ; +
854 ; +   Requires:
855 ; +       Variable 'gStatus' to be declared
856 ; +
857 ; +   Example:
858 ; +       setvect 1,HandleIrqn1
859 ; +       intson
860 ; +
861 ; +   Results:
862 ; +       Updates the variable 'gStatus'
863 ; ++++++
864 HiTemp      word      0
865 HandleIrqn1
866     copy     *$40,HiTemp                ; Read the status value from
device
867     or       HiTemp,gStatus              ; Or it into the global value
868     rfi
869
```

Trademarks

RALCWI is a trademark of CML Microsystems Plc.

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 62 888129 Fax: +65 62 888230 Sales: sg.sales@cmlmicro.com Tech Support: sg.techsupport@cmlmicro.com
- www.cmlmicro.com -		