Application Report

SNOA084B – May 2004 – Revised May 2004

AN-1153 COP8™ Flash ISP HANDBOOK – Virtual E^2 Guide

ABSTRACT

This application note describes the COP8™ Virtual E^2 Methodology. Emulated E^2 allows the programmer to treat flash memory as if it were E^2.

Contents

List of Figures

List of Tables

1 INTRODUCTION

The following appnote describes the setup and use of an emulated E^2. This is also known as Virtual E^2. This method of storage utilizes non-volatile memory. Detailed macros utilizing Virtual E^2 routines are included in this application note.

2 INTRODUCTION TO THE VIRTUAL E^2 CONCEPT

The Virtual E^2 concept arose from the need to have E^2 PROM peripheral on a microcontroller. It is a natural extension of the COP8 Flash Family Boot ROM firmware. There are user accessible Virtual E^2 entry points contained in the COP8™ Flash Family Boot ROM. An implementation example is shown in Figure 1.

3 MEMORY PARTITONING BETWEEN VIRTUAL E^2, RAM, AND FLASH

In order to use Virtual E^2, the COP8 Flash Family device should be split up into two partitions. Partition one is responsible for updating to the flash. This is where the software will use the Virtual E^2 entry points. Partition two is where the data to be modified is located. Section two may be data memory or it may be program memory or it may be a mixture of both. The same partitioning scheme must be implemented on the RAM side.

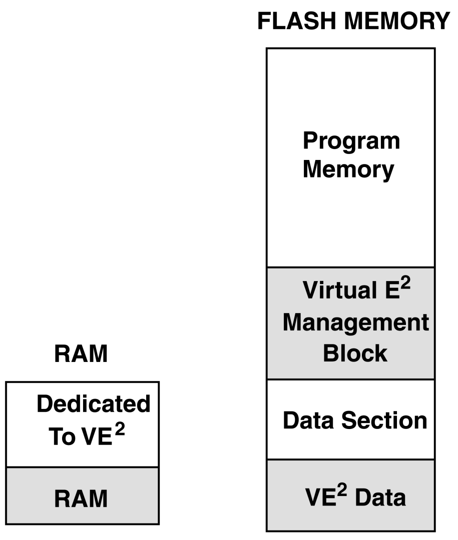

Figure 1. Sample Virtual E^2 Partition Arrangement

4 VIRTUAL E^2 SUPPORT BLOCKS

This section deals with the Virtual E^2 Support Block. Entry point locations are shown in Table 1. Registers are shown in Table 3. In addition, each description contains details about security dependencies. There are no checks made for the validity of the ISP Address and the BYTECOUNTHI register. Data transfers will take place from whatever RAM locations are specified by the segment and BYTECOUNTLO registers.

4.1 *JSRB LABELS For Virtual E^2 Routines*

Entry points for the Virtual E^2 routines are shown in Table 1.

Table 1. Virtual E^2 Entry Points and Their Associated Labels

Command/Labels	ROM Address
ve2pgerase	0x17
ve2readbf	0x11

Table 1. Virtual E^2 Entry Points and Their Associated Labels (continued)

Command/Labels	ROM Address
ve2blockr	0x26
ve2writebf	0x14
ve2blockw	0x23

To execute commands listed in Table 1, the JSRB instruction must be used. In order for correct behavior to occur, a “KEY” must be set prior to executing the JSRB instruction. The PGMTIM register must also be set prior to any write or erase commands. It is up to the user to enforce security when using these commands.

5 DESCRIPTION OF VIRTUAL E^2 ROUTINES/ENTRY POINTS

Table 2 shows the Virtual E^2 function in detail.

Table 2. Virtual E^2 Entry Points

Command/ Label	Function	Entry Point Location	Parameters	Return Data
ve2pgerase	Page Erase	0x17	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address.	N/A (A page of memory beginning at ISPADHI, ISPADLO will be erased).
v2readbf	Read Byte	0x11	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address.	Data Byte in Register ISPRD.
ve2blockr	Block Read	0x26	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. X pointer contains the beginning RAM address where the result(s) will be returned. Register BYTECOUNTLO contains the number of n bytes to read ($0 \le n \le 255$). Limited to 128 bytes due to RAM Segmentation. Register BYTECOUNTHI is **ignored**. It is up to the user to setup the segment register.	n Data will be returned beginning at a location pointed to by the RAM address in X.
ve2writebf	Write Byte	0x14	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. Register ISPWR contains the Data Byte to be written.	N/A
ve2blockw	Block Write	0x23	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. Register BYTECOUNTLO contains the number of n bytes to write ($0 \le n \le 16$) X pointer contains the beginning RAM address of the data to be written, It is the user's responsibility to initialize the segment register. Data must be placed with-in the 1/2 page segment (64 byte for 32k devices and 32 byte for 1k and 4k devices). This limitation is due to the multi-byte write limitation.	N/A

6 REGISTERS

Table 3 shows the register that are used in the Virtual E^2 commands.

Table 3. Registers Associated with the Virtual E^2 Routines

Register Name	Purpose	RAM Location
ISPADHI	High Address of Flash Memory	0xA9
ISPADLO	Low Address of Flash Memory	0xA8
ISPWR	The value to be written must stored in this register before jumping into the write byte routine.	0xAB
ISPRD	Data will be returned to this register after the read byte routine execution.	0xAA
ISPKEY	This register holds the KEY value. The KEY value is utilized to verify that a JSRB execution is requested in the next 6 instruction cycles.	0xE2
BYTECOUNTLO	Holds the low byte of the counter.	0xF1
SIOR	Microwire Buffer	0xE9
PGMTIM	Write Timing Register	0xE1
KEY	Must transferred to the ISPKEY register before a JSRB executed.	0x98

7 THE PGMTIM REGISTER

Table 4 show the valid values for the PGMTIM Register. The user's program **MUST** load the listed value into the PGMTIM register (located 0xE1) prior to using any Virtual E^2 routine which writes to the Flash Array (e.g., ve2writebf, ve2pgerase, etc).

Table 4. Values for the PGMTIM Register

Bit Values for the PGMTIM Register								Hex Value	CKI Frequency Range
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	1	0x01	25 kHz–33.3 kHz
0	0	0	0	0	0	1	0	0x02	37.5 kHz–50 kHz
0	0	0	0	0	0	1	1	0x03	50 kHz–66.67 kHz
0	0	0	0	0	1	0	0	0x04	62.5 kHz–83.3 kHz
0	0	0	0	0	1	0	1	0x05	75 kHz–100 kHz
0	0	0	0	0	1	1	1	0x07	100 kHz–133 kHz
0	0	0	0	1	0	0	0	0x08	112.5 kHz– 150 kHz
0	0	0	0	1	0	1	1	0x0B	150 kHz–200 kHz
0	0	0	0	1	1	1	1	0x0F	200 kHz–266.67 kHz
0	0	0	1	0	0	0	1	0x11	225 kHz–300 kHz
0	0	0	1	0	1	1	1	0x17	300 kHz–400 kHz
0	0	0	1	1	1	0	1	0x1D	375 kHz–500 kHz
0	0	1	0	0	1	1	1	0x39	500 kHz–666.67 kHz
0	0	1	0	1	1	1	1	0x2F	600 kHz–800 kHz
0	0	1	1	1	1	1	1	0x3F	800 kHz–1.067 MHz
0	1	0	0	0	1	1	1	0x47	1 MHz–1.33 MHz
0	1	0	0	1	0	0	0	0x48	1.125 MHz–1.5 MHz
0	1	0	0	1	0	1	1	0x4B	1.5 MHz–2 MHz
0	1	0	0	1	1	1	1	0x4F	2 MHz–2.67 MHz
0	1	0	1	0	1	0	0	0x54	2.625 MHz–3.5 MHz
0	1	0	1	1	0	1	1	0x5B	3.5 MHz–4.67 MHz
0	1	1	0	0	0	1	1	0x63	4.5 MHz–6 MHz
0	1	1	0	1	1	1	1	0x6F	6 MHz–8 MHz
0	1	1	1	1	0	1	1	0x7B	7.5 MHz–10 MHz
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W		

8 MANUEVERING BACK AND FORTH BETWEEN FLASH MEMORY AND BOOT ROM

When using Virtual E^2 routines, at some point, it will be necessary to maneuver between the flash program memory and the Boot ROM, even when using customized ISP routines. This is because it's not possible to execute from the flash program memory while it's being programmed.

The JSRB instruction is used to jump from flash memory to Boot ROM, and the RETF is used to return from the Boot ROM back to the flash program memory. See the instruction chapter for specific details on the operation of these instructions. The JSRB instruction must be used in conjunction with the Key register. This is to prevent jumping to the Boot ROM in the event of runaway software. For the JSRB instruction to actually jump to the Boot ROM, the Key bit must be set. This is done by writing the value shown in Table 5 to the Key register. The Key is a 6 bit key and, if the key matches, the KEY bit will be set for 8 instruction cycles. The JSRB instruction must be executed while the KEY bit is set. If the KEY does not match, then the KEY bit will not be set and the JSRB will jump to the specified location in the flash memory. In emulation mode, if a breakpoint is encountered while the KEY is set, the counter that counts the instruction cycles will be frozen until the breakpoint condition is cleared. The Key register is a memory mapped register. Its format when writing is shown in Table 5.

Table 5. KEY Register Write Format

KEY when Writing							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
1	0	0	1	1	0	X	X

Bits 7–2:—Key value that must be written to set the KEY bit.

Bits 1–0:—Don't care.

8.1 *Interrupt Lock Out Time*

Interrupts are inhibited during execution from Boot ROM. Table 6 shows the amount of time that the user is **LOCKED OUT** of their interrupt service routine(s). The servicing of interrupts (pending) will be resumed once the ISP Boot ROM returns the user to the Flash. The user should take into account the amount of time they are locked out of their interrupts. Some of the **LOCK OUT** times are dependent upon the PGMTIM. PGMTIM is a value entered into the PGMTIM register (refer to Section 7). The user code **MUST** set the PGMTIM register before any write routines occur (e.g., a LD PGMTIM,#06F is needed to specify a CKI frequency of 6 MHz).

Table 6. Required Interrupt Lockout Time (in Instruction Cycles)

Flash Routines (User Accessable)	Minimum Interrupt Delay Time Required (In Instruction Cycles)
ve2pgerase	120 + 100*PGMTIM[(1)]
ve2readbf	100
ve2blockr	100
ve2blockr	140/BYTE
ve2blockw	100 + 3.5*PGMTIM/BYTE[(1)] + 68/BYTE
ve2writebf	168 + 3.5*PGMTIM[(1)]

(1) Refer to Table 4 for additional information on the PGMTIM variable.

8.2 *ve2pgerase—Virtual E^2 Entry Point: Erase a Page of Flash Memory*

This routine requires that ISPADHI and ISPADLO are loaded before the jump. A KEY is a number which must be loaded into the KEY Register (at location 0xE2) before issuing a JSRB instruction. Table 5 shows the format of the KEY number. Loading the KEY, and a “JSRB ve2pgerase” are all that is needed to complete the call to the routine. No acknowledgement will be sent back regarding the operation. For details regarding the registers ISPADHI and ISPADLO please refer to Table 3. See Table 8 for details on the number of endurance cycles and the number of page erase commands that should be issued prior to writing data into the erase page. Since this is a Virtual E^2 command, this routine will work regardless of security (security independent). Example 1 is an example of how to use the ve2pgerase function—assembly version. Example 2 shows the C version of the ve2pgerase function. Table 7 shows the necessary resources needed to run the routine.

Example 1. SAMPLE ve2pgerase (PAGE ERASE) EXECUTION—Assembly Version

```
; ERASE A PAGE OF FLASH, at 0x80
; ASSUME A 6 MHz CKI FREQUENCY
.INCLD COP8CBR.INC              ; INCLUDE FILE FOR THE FLASH PART
VE2PGERASE = 017                ; ENTRY POINT FOR ISP CALL
.SECT CODE,ROM,ABS=0            ; BEGINING CODE SPACE
MAIN:                           ; THE MAIN ROUTINE
LD PGMTIM,#06F                  ; SET PGMTIM FOR 6 MHz CLOCK
LD ISPADHI,#000                 ; LOAD HIGH ADDRESS WITH 0x00
LD ISPADLO,#080                 ; LOAD LOW ADDRESS WITH 0x80
LD ISPKEY,#098                  ; LOAD THE KEY
JSRB VE2PGERASE                 ; CALL THE ROUTINE
HERE:
JP HERE
.END MAIN                       ; END OF PROGRAM
```

Example 2. SAMPLE ve2pgerase(unsigned long) EXECUTION—C Version

```
#include<dev/flashcop.h>   // include file for the FLASH part
#include<flash.h>                // include file for the ISP routines
void main()
{
PGMTIM=0x6F;                     // for a 6 MHz Clock
page_erase(0x0280);             // call erase function
}
```

Table 7. Resource Utilization for the Command: ve2pgerase (Page Erase)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
ISPADHI ISPADLO	YES	YES	NO	YES	YES	NONE	120 +100* PGMTIM[a]	4

Table 8. Typical Endurance Cycles vs. Erase Time and Temperature

Erase Time in ms	Low End of Operating Temp Range			
	-40°C	-20°C	0°C	25°C
1	60k	60k	60k	100k
2	60k	60k	60k	100k
3	60k	60k	60k	100k
4	60k	60k	100k	100k
5	70k	70k	100k	100k
6	80k	80k	100k	100k
7	90k	90k	100k	100k
8	100k	100k	100k	100k

8.3 ve2readbf—Virtual E^2 Entry Point: Read a Byte of Flash Memory

This routine requires that ISPADHI and ISPADLO are loaded before the jump. Loading the KEY, and a "JSRB ve2readbf" are all that is needed to complete the call to the routine. Data will be returned to the ISPRD Register. No acknowledgement will be sent back regarding the operation. For details regarding the ISPADHI, ISPADLO, and ISPRD registers please refer to Table 3. Since this is a Virtual E^2 command, this routine will work regardless of security (security independent). Example 3 is an example of how to use the ve2readbf function—assembly version. Example 4 shows the C version of the ve2readbf function. Table 9 shows the necessary resources needed to run the routine.

Example 3. SAMPLE ve2readbf (Read a Byte of Flash Memory) EXECUTION

```
;READ A BYTE OF FLASH AT 0x80 AND STORE IN RAM AT 0x05;ASSUME A 6 MHz CKI FREQUENCY       .INCLD
COP8CBR.INC      ; INCLUDE FILE FOR THE FLASH PART              VE2READBF = 011  ; ENTRY POINT FOR
ISP       .SECT     CODE,ROM,ABS=0    ; BEGINNING CODE SPACEMAIN:                          ; THE MAIN
ROUTINE       LD    PGMTIM,#06F             ; SET PGMTIM FOR 6 MHz CLOCK       LD ISPADHI,#000
 ; LOAD HIGH ADDRESS WITH 0x00       LD ISPADLO,#080         ; LOAD LOW ADDRESS WITH 0x80       LD
ISPKEY,#098          ; LOAD THE KEY       JSRB VE2READBF          ; CALL THE FUNCTION       LD
A,ISPRD              ; LOAD THE READ BYTE       X  A,005                ; STORE THE READ BYTEHERE:
   JP HERE       .END MAIN               ; END OF PROGRAM
```

Example 4. SAMPLE ve2readbf(unsigned long) EXECUTION—C Version

```
#include <dev/flashcop.h>                 // include file for the Flash Microcontroller
#include <flash.h>                                    // include file that contain the flash routines
unsigned int storage[10] @ 0x10;

                                                                // storage array at RAM 0x10
void main(){
PGMTIM = 0x6F;

                                                            // set PGMTIM for 6 MHz crystal
readbf(0x01);

                                                        // read a byte of FLASH at address 0x01
storage[5] = AC;

                                                    // store it into address 0x15}
```

Table 9. Resource Utilization for the Command: ve2readbf (Read a Byte of Flash Memory)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
ISPADHI ISPADLO	YES	YES	NO	YES	YES	Data/ISPRD Register	100	4

8.4 ve2blockr—Virtual E^2 Entry Point: Read a Block of Flash Memory

The ve2blockr routine will read multiple bytes from the flash memory. ISPADHI and ISPADLO are assumed to be loaded before the jump. Register BYTECOUNTLO is also assumed to be loaded. The X pointer contains the address where the data will be placed. The BYTECOUNTLO register is used by the microcontroller to send back N number of bytes (i.e., N=BYTECOUNTLO). If N=0 then the firmware will abort. Data is saved into the RAM address pointed to by the X pointer. It is up to the user to setup the segmentation register. This routine is capable of reading up to 256 bytes of flash memory to RAM (limited due to the memory available). In addition this routine is limited to reading blocks of 128 bytes due to the RAM segmentation. If an attempt to read greater than 128 bytes is issued, the firmware will begin to write to RAM locations beginning at 0x80 and above (possibly corrupting the I/O and CONTROL REGISTERS).

After the X pointer and the BYTECOUNTLO are set, the KEY must be loaded, and a “JSRB ve2blockr” must be issued. No acknowledgement will be sent back regarding the operation. For details regarding the ISPADHI, ISPADLO, and BYTECOUNTLO registers please refer to Table 3. Since this is a Virtual E^2 command, this routine will work regardless of security (security independent). Example 5 is an example of how to use the ve2blockr function—assembly version. Example 6 shows the C version of the ve2block_read function. Table 10 shows the necessary resources needed to run the routine.

Example 5. SAMPLE ve2blockr (Read a Block of Flash Memory) EXECUTION —Assembly Version

```
;READ 3 BYTE OF FLASH AT 0x80 AND WRITE TO RAM AT 0x0D;ASSUME A 6 MHz CKI FREQUENCY
.INCLD COP8CBR.INC      ; INCLUDE FILE FOR THE FLASH PART
 VE2BLOCKR = 026        ; ENTRY POINT FOR ISP
.SECT  CODE,ROM,ABS=0   ; BEGINNING CODE SPACE
MAIN:                   ; THE MAIN ROUTINE
LD     PGMTIM,#06F         ; SET PGMTIM FOR 6 MHz CLOCK
LD ISPADHI,#000         ; LOAD HIGH ADDRESS WITH 0x00
LD ISPADLO,#080         ; LOAD HIGH ADDRESS WITH 0x80
LD     S,#000              ; LOAD S POINTER WITH 0x00
LD     X,#00D              ; LOAD X POINTER WITH 0x0D
LD BYTECOUNTLO,#003     ; SET BYTECOUNT TO 0x03
LD ISPKEY,#098          ; LOAD THE KEY
JSRB    VE2BLOCKR          ; CALL THE FUNCTION
HERE:                   ; INFINITE LOOP
JP HERE
.END MAIN               ; END OF PROGRAM
```

Example 6. SAMPLE ve2block_read(unsigned long, unsigned int, unsigned long) C—Version

```
#include <dev/flashcop.h>             // include file for the FLASH part
#include <flash.h>                                                // include file for the ISP routines
 void main(){
PGMTIM = 0x6F;                                                                           // for
6 MHz crystal
block_readf(0x01,3,0x10);                                           // read a 3 byte of flash


                                                   // at 0x01 to RAM 0x10}
```

Table 10. Resource Utilization for the Command: ve2blockr (Block Read of the Flash Memory)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
BYTCONTLO ISPADHI ISPADLO X - Pointer	YES	YES	YES	YES	YES	DATA/RAM[X]	140/BYTE	6

8.5 ve2blockw—Virtual E^2 Entry Point: Write to a Block Flash Memory

ISPADHI and ISPADLO must be set by the user prior to the jump into command. The BYTECOUNTLO variable is used by the microcontroller to transfer N number of bytes (i.e, N=BYTECOUNTLO). This variable also must be set prior to the jump into command. If N=0 then the firmware will abort. The maximum number of bytes that can be written are 16. If the number of bytes exceeds 16, then the user can not be guaranteed that all of the bytes were written. The data cannot cross 1/2 page boundaries (i.e. all data must be within the 64 bytes segment for 32k devices and within 32 bytes for 4k, and 1k devices). Data is read from the RAM address pointed to by the X pointer. It is up to the user to setup the segmentation register. Data transfers will take place from whatever RAM locations are specified by the X pointer and segment register. However, if the X pointer exceeds the top of the segment, the firmware will begin to transfer from 0x80 (I/O and CONTROL REGISTERS) and above. After the X pointer and the BYTECOUNTLO are set, the KEY must be loaded, and a “JSRB ve2blockw” must be issued. For details regarding the ISPADHI, ISPADLO, and BYTECOUNTLO registers please refer to Table 3. Since this is a Virtual E^2 command, this routine will work regardless of security (security independent). Example 7 is an example of how to use the ve2blockw function—assembly version. Example 8 shows the C version of the ve2block_write function. Table 11 shows the necessary resources needed to run the routine.

Example 7. SAMPLE ve2blockw (Write to a Block of Flash Memory) EXECUTION

```
;WRITE 10 BYTE TO FLASH AT 0X80 FROM RAM AT 0X08;ASSUME A 6 MHz CKI FREQUENCY
       .INCLD COP8CBR.INC     ; INCLUDE FILE FOR THE FLASH PART
             VE2BLOCKW = 023  ; ENTRY POINT FOR ISP
.SECT      CODE,ROM,ABS=0                               ; BEGINNING CODE SPAC
MAIN:                         ; THE MAIN PROGRAM
LD PGMTIM,#06F                                       ; SET PGMTIM FOR 6 MHz CLOCK
 LD ISPADHI,#000                                     ; LOAD HIGH ADDRESS BYTE  WITH 0x00
LD ISPADLO,#080                                      ; LOAD LOW ADDRESS BYTE WITH 0x80
 LD S,#000                                           ; LOAD S POINTER WITH 0x00
LD X,#008                                            ; LOAD X POINTER WITH 0x80
LD BYTECOUNTLO,#010                                  ; NUMBER OF BYTES TO WRITE
LD ISPKEY,#098                                       ; LOAD THE KEY
JSRB    VE2BLOCKW                                       ; CALL THE FUNCTION


HERE:                                                ; INFINITE LOOP
JP HERE
.END MAIN                                            ; END OF PROGRAM
```

Example 8. SAMPLE ve2block_write (unsigned long, unsigned int, unsigned long) C—Version

```
#include <dev/flashcop.h>               // include file for the FLASH part
#include <flash.h>                                          // incldue file for the ISP routines

void main(){

PGMTIM = 0x6F;                                                     // for 6 MHz CKI Clock crystal
block_writef(0x10,16,0x100);                                      // write 16 bytes starting at RAM
0x10

                                                    // to flash at 0x100}
```

Table 11. Resource Utilization for the Command: ve2blockw (Write to a Block of Flash Memory)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Return Data	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
BYTCOUNTLO Data is Assumed to be in the ***RAM[X]*** Location(s)	YES	YES	YES	YES	YES	NONE	100 + 3.5*PGM-TIME/BYTE[(1)] + 68 BYTE	6

(1) Refer to Table 4 for additional information on the PGMTIM variable.

8.6 ve2writebf—Virtual E2 Entry Point: Write a Byte to the Flash Memory

This routine requires that ISPADHI, ISPADLO, and ISPWR be loaded prior to the jump. Loading the KEY and a “JSRB ve2writebf” are all that is needed to complete this call. No acknowledgement will be sent back regarding the operation. For details regarding the ISPADHI, ISPADLO, and ISPRD registers please refer to Table 3. Since this is a Virtual E^2 command, this routine will work regardless of security (security independent). Example 9 is an example of how to use the ve2writebf function—assembly version. Example 10 shows the C version of the ve2writebf function. Table 12 shows the necessary resources needed to run the routine.

Example 9. SAMPLE ve2writebf (Write a Byte to Flash Memory) EXECUTION—Assembly Version

```
; WRITE A BYTE TO THE FLASH [0x100] = 5; ASSUME A 6 MHz CKI FREQUENCY
       .INCLD  COP8CBR.INC               ; INCLUDE FILE FOR THE FLASH PART
                VE2WRITEBF = 014   ; ENTRY POINT FOR ISP
.SECT   CODE,ROM,ABS=0                                     ; BEGINNING CODE SPACE

MAIN:                                       ; THE MAIN ROUTINE
        LD    PGMTIM,#06F                   ; SET PGMTIM FOR 6 MHz CLOCK
                                    LD    ISPWR,#005              ; LOAD WRITE REGISTER WITH 0x05
        LD    ISPADHI,#001            ; LOAD HIGH ADDRESS WITH 0x01
        LD    ISPADLO,#000            ; LOAD LOW ADDRESS WITH 0x00
        LD    ISPKEY,#098                         ; LOAD THE KEY        J
                                SRB    VE2WRITEBF               ; CALL THE FUNCTION

HERE:                              ; INFINITE LOOP
JP    HERE
    .END MAIN                                                         ; END OF PROGRAM
```

Example 10. SAMPLE ve2writebf(unsigned int, unsigned long, unsigned int) (Write a byte to flash memory) EXECUTION—C Version

```
#include <dev/flashcop.h>      // incldue file for the FLASH part
#include <flash.h>                                 // include file for the ISP routines

void main(){

PGMTIM = 0x6F;                                            // for 6 MHz crystal
cwritebf(0x120, 2);                                       // write a byte with a value of 2 to


                     // flash at 0x120}
```

Table 12. Resource Utilization for the Command: ve2writebf (Write a Byte to the Flash).

Input Data	Accumulator or A Used?	B Pointer Used?	X Pointer Used?	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
ISPWR Contains the Data	YES	YES	NO	YES	NONE	168 + 3.5* PGMTIME[(1)]	4

(1) Refer to Table 4 for additional information on the PGMTIM variable.

9 WATCHDOG™ SERVICES

The Watchdog register will be serviced periodically in order to ensure that a watchdog event has not occurred. All routines in the ISP Boot ROM incorporate automatic watchdog services. Periodically the Boot ROM firmware will service the watchdog if a routine will take greater than the 8k upper window requirement.

10 VIRTUAL E^2 MACRO ROUTINES

There are five MACRO routines listed in Table 13. Information reading functional descriptions, parameters and the returned data are listed.

Table 13. Macro Routines

Macro Routine	Function	Parameters	Return Data
VE2RASE	Page Erase A 128 Byte Block	Erases the VE2 Memory Page using an embedded shadow ROM. Base on the ve2pgerase function. **PAGE** range 0–255 (assumes a 32k Flash)	N/A (A page of memory beginning at PAGE will be erased)
VE2RD	Read Byte	Address is in RAM memory. Returned data is loaded into A after being read. IF **TO==“A”** then the datum is swapped with the addressed RAM location. The S register must be value for the **TO** address.	Data Byte is saved into the accumulator.
VE2RDBL	Block Read	**FROM LIMITATION:** MOD 128 + **SIZE** >128 is an error (i.e., must fit a VE^2 page) **TO** MOD 128 + **SIZE** > 128 is an error (i.e., it must fit in a RAM page) NOTE: Register **X** and **0F1** are used by the function and are not restored.	n Data Bytes if Security not set, Data will be returned beginning at a location pointed to by the RAM address by **TO**.
VE2WR	Write Byte	A register, literal or memory segment offset address. Assumes **S** is valid for memory reference. Base Address in VE^2 to which data is written.	N/A
VE2WRBL	Block Write	**FROM**: Base Address in RAM from which data is written. **TO**: Base Address in VE^2 memory. **SIZE**: Size of the array. Register **X** and **0F1** are used by the function and not restored. Resultant data must be in a 64 byte segment (1/2 page size limitation in the block write routine for a 32k device).	N/A

Example 11 contains the Virtual E^2 byte erase routine. Data regarding its use and calling parameters are listed in the figure.

Example 11. VE2RASE Macro Routine: Erase a Page from the VE^2 Archive

```
;; -----------------------------------------------------------------
;;VE2RASE Erases the VE2 Memory Page using embedded shadow ROM
;;Based Virtual EE Page Erase Function
;;Parameters:
;;PAGE range 0-255 (assumes 32k Flash)
;; -----------------------------------------------------------------

.MACRO  VE2RASE,PAGE
.IFNDEF_VE2REGS
.INCLD VE2REGS.INC
.ENDIF
.IF PAGE>255
.ERROR               ;Parameter PAGE is too large
.ELSE
 LD ISPADHI,#(0x7F AND PAGE / 2)
 LD ISPADLO,#((PAGE MOD 2) SHL 7)
LD ISPKEY,#098        ;Load the key to enable JSRB
JSRB    CPGERASE
.ENDIF
.ENDM
```

Example 12 contains information regarding the block write routine to the VE2WRBL routine. Data regarding its use and calling parameters are shown in the figure.

Example 12. VE2WRBL Macro Routine: Write to a Block of VE^2 Archive

```
;; ------------------------------------------------------------------
;;Write a block of data to VE2 archive
;;Parameters:
;;TO:    Base Address in VE2 memory
;;FROM:    Base Address in RAM from which data is written
;;SIZE:Size of the array.
;;Restrictions:
;;TO MOD 64 + SIZE > 64 is an error
;;FROM MOD 64 + SIZE > 64 is an error
;;Registers X and 0F1 are used by the function and not restored
;; -----------------------------------------------------------------

.MACRO VE2WRBL,TO,FROM,SIZE

.IFTO MOD 64 + SIZE > 64
 .ERROR               ;DATA BLOCK EXCEEDS VE2 PAGE LIMIT
.ELSE
.IF FROM MOD 64 + SIZE > 64
    .ERROR               ;SOURCE EXCEEDS RAM SEGMENT LIMIT
.ELSE
.IFNDEF_VE2REGS
 .INCLD  VE2REGS.INC
.ENDIF
LD ISPADHI,#HIGH(TO)
LD ISPADLO,#LOW(TO)
LD S,#HIGH(FROM)
LD X,#LOW(FROM)
LD 0F1,#SIZE
LD ISPKEY,#098        ;Load the key to enable JSRB
JSRB CBLOCKW
    .ENDIF
.ENDIF
ENDM
```

Example 13 contains information regarding the block read routine for the VER2RDBL routine. Data regarding its use and calling parameters are also shown in this figure.

Example 13. VE2RDBL Macro Routine: Read a Block of Data from VE^2 Archive

```
;; ------------------------------------------------------------------
                        ;;Read a block of data from VE2 archive to RAM
                        ;;Parameters:
                        ;;TO:Base Address in RAM memory
                        ;;FROM:Base Address in VE2 from which data is written
                        ;;SIZE:Size of the array.
                        ;;Restrictions:
                        ;;TO MOD 128 + SIZE > 128 is an error - must fit a RAM page
                        ;;FROM MOD 128 + SIZE > 128 is an error - must fit a VE2 page
                        ;;Registers X and 0F1 are used by the function and not restored
                        ;; ------------------------------------------------------------------
     .MACRO VE2RDBL,TO,FROM,SIZE
                        .IF    TO MOD 128 + SIZE > 128
                     .ERROR               ;DATA BLOCK EXCEEDS RAM SEGMENT LIMIT
                        .ELSE
                    .IF FROM MOD 128 + SIZE > 128
                        .ERROR               ;SOURCE EXCEEDS VE2 PAGE LIMIT
                     .ELSE
                    .IFNDEF_VE2REGS
                    .INCLD  VE2REGS.INC
                    .ENDIF
                        LD  ISPADHI,#HIGH(FROM)
```

Example 13. VE2RDBL Macro Routine: Read a Block of Data from VE^2 Archive (continued)

```
                      LD   ISPADLO,#LOW(FROM)
                         LD   S,#HIGH(TO)
                         LD   X,#LOW(TO)
                         LD   0F1,#SIZE
                      LD   ISPKEY,#098     ;Load the key to enable JSRB
                         JSRB CBLOCKR
                     .ENDIF
                     .ENDIF
                     .ENDM
```

Example 14 contains information regarding the VE2RD routine. Information regarding its use and parameters are also shown in the figure.

Example 14. VE2RD Macro Routine: Read a Byte of VE^2 Archive

```
;; -------------------------------------------------------------------
                         ;;Read a byte of data from VE2 archive
                         ;;Parameters:
                         ;;TO:Address in RAM memory; Symbol A handled as well
                         ;;FROM:Address in VE2 from which data is read
                         ;;Returned data is loaded into A after being read. If TO == "A"
                         ;;then the datum is swapped with the addressed RAM location.  The
                         ;;S register must be valid for the TO address
                         ;; -------------------------------------------------------------------
                                      .MACRO VE2RD,TO,FROM
                                          LD   ISPADHI,#HIGH(FROM)
                                          LD   ISPADLO,#LOW(FROM)
                                             LD   ISPKEY,#098       ;Load the key to enable JSRB
                                          JSRB CREADBF
                                             LD   A,ISPRD
                                             .ifstr "TO" NE "A"
                                              X A,LOW(TO)
                                                 .endif
                                                 .ENDM
```

Example 15 contains information regarding the VE2WR routine. Information regarding its use and calling parameters are shown in the figure.

Example 15. VE2WR Macro Routine: Write to a Byte of VE^2 Archive

```
;; -------------------------------------------------------------------
                             ;;Write a byte of data to VE2 archive
                             ;;Parameters:
                             ;;TO:Base Address in VE2 to which data is written
                             ;;FROM:A register, literal or memory segment offset address
                             ;;Assumes S is valid for memory reference
                             ;; -------------------------------------------------------------------

                                     .MACRO VE2WR,TO,FROM
                                  LD   ISPADHI,#HIGH(TO)
                                     LD   ISPADLO,#LOW(TO)
                                 .ifstr "FROM" NE "A"
                                      LD A,FROM
                                     .endif
                                     X   A,ISPWR
                                  LD ISPKEY,#098        ;Load the key to enable JSRB
                                     JSRB CWRITEBF
                                     .ENDM
```

Example 16 contains information regarding the use and calling of the MACRO files.

Example 16. Sample Program Demonstrating the Use of the Virtual E^2

```
; -------------------------------------------------------------------
                        ; assembly test program
                        ; -------------------------------------------------------------------
       .incld  cop8cbr.inc

                            .incld MACROS.MAC
                            .sects1,seg,abs=0x100
                                siza1=15
        ary1:  .dsbsiza1
        par1:  .dsb1
                                    .endsect
                                    .sects2,seg,abs=0x200
                                        siza2=8
        ary2:  .dsbsiza2
        par2:     .dsw1
                                    .endsect
    ; ---- define a virtual page of 128 bytes for archive data
                                ve2_page=0x53
    ; ---- define a virtual section structure
    ;archive data block definitions...symbol definitions using DSx
    ;directives result in a warning because symbols are presumed to
    ;be in ROM.  The definitions are restricted to fit a 128 byte
    ;VE2 memory page.
                 ve2_arch=ve2_page shl 7       ; base address at page
                 ve2_a1=ve2_arch + 0           ; array 1, length 15
                 ve2_p1=ve2_a1 + siza1         ; byte
                 ve2_a2=ve2_p1 + 1             ; array 2, length 8
                 ve2_p2=ve2_a2 + siza2

                                    ; word, length 2

 .sect   code,rom,abs=0

START:

VE2RASE ve2_page                                ; archive literal 15
VE2WR   ve2_p1,#15                           ; archive ary2 into ve2_a2
VE2WRBL ve2_a2,ary2,siza2              ; archive ary1 into ve2_a1
VE2WRBL ve2_a1,ary1,siza1                    ;    archive ram    loc<&apos>n 0F0 into ver2_p2

VE2WR ve2_p2,0F0              ; read A from P2
VE2RD A,ve2_p2                                                          ; restore par1 from
archive
    VE2RD par1,ve2_p1                                            ; restore ary1 from archive
    VE2RDBL ary1,ve2_a1,siza1                ; restore par2 from P2
            VE2RD par2,ve2_p2

.end START
```

Example 17 contains information regarding the entire set of MACRO files. Information regarding its use and calling parameters are shown in the figures.

Example 17. MACRO File. Contains the Subroutines Responsible for Virtual E^2 Operations

```
;; -------------------------------------------------------------------
;;      This file defines a set of macros to facilitate access of
;;      data in Virtual EE memory for the COP8CBR/CCR/CDR part family.
;; -------------------------------------------------------------------
```

Example 17. MACRO File. Contains the Subroutines Responsible for Virtual E^2 Operations (continued)

```
;; ---------------------------------------------------------------------
;;VE2RASE Erases the VE2 Memory Page using embedded shadow ROM
;;Based Virtual EE Page Erase Function
;;Parameters:
;;PAGE range 0-255 (assumes 32K Flash)
;; ---------------------------------------------------------------------
    .MACRO  VE2RASE,PAGE
.IFNDEF_VE2REGS
.INCLD VE2REGS.INC
.ENDIF
.IF PAGE>255
.ERROR                  ;Parameter PAGE is too large
.ELSE
LD  ISPADHI,#(0x7F AND PAGE / 2)
LD  ISPADLO,#((PAGE MOD 2) SHL 7)
LD  ISPKEY,#098       ;Load the key to enable JSRB
JSRB CPGERASE
.ENDIF
.ENDM

;; ---------------------------------------------------------------------
;;Write a block of data to VE2 archive
;;Parameters:
;;TO:Base Address in VE2 memory
;;FROM:Base Address in RAM from which data is written
;;SIZE:Size of the array.
;;Restrictions:
;;TO MOD 128 + SIZE > 128 is an error
;;FROM MOD 128 + SIZE > 128 is an error
;;Registers X and 0F1 are used by the function and not restored
;; ---------------------------------------------------------------------
    .MACRO VE2WRBL,TO,FROM,SIZE
    .IF    TO MOD 64 + SIZE > 64
    .ERROR                ;DATA BLOCK EXCEEDS VE2 PAGE LIMIT
    .ELSE
.IF FROM MOD 64 + SIZE > 64
.ERROR                ;SOURCE EXCEEDS RAM SEGMENT LIMIT
.ELSE

.IFNDEF__VE2REGS
.INCLD  VE2REGS.INC
.ENDIF
 LD  ISPADHI,#HIGH(TO)
LD  ISPADLO,#LOW(TO)
LD  S,#HIGH(FROM)
LD  X,#LOW(FROM)
LD  0F1,#SIZE
LD  ISPKEY,#098      ;Load the key to enable JSRB

JSRB CBLOCKW
.ENDIF
.ENDIF
.ENDM
;; ---------------------------------------------------------------------
;;Read a block of data from VE2 archive to RAM
;;Parameters:
;;TO:Base Address in RAM memory
;;FROM:Base Address in VE2 from which data is written
;;SIZE:Size of the array.
;;Restrictions:
;;TO MOD 128 + SIZE > 128 is an error - must fit a RAM page
;;FROM MOD 128 + SIZE > 128 is an error - must fit a VE2 page
;;Registers X and 0F1 are used by the function and not restored
;; ---------------------------------------------------------------------
```

Example 17. MACRO File. Contains the Subroutines Responsible for Virtual E^2 Operations (continued)

```
.MACRO VE2RDBL,TO,FROM,SIZE
.IFTO MOD 128 + SIZE > 128
.ERROR ;DATA BLOCK EXCEEDS RAM SEGMENT LIMIT
.ELSE
.IF FROM MOD 128 + SIZE > 128
.ERROR ;SOURCE EXCEEDS VE2 PAGE LIMIT
.ELSE
.IFNDEF__VE2REGS
.INCLD VE2REGS.INC
.ENDIF
LD ISPADHI,#HIGH(FROM)
LD ISPADLO,#LOW(FROM)
LD S,#HIGH(TO)
LD X,#LOW(TO)
LD 0F1,#SIZE
LD ISPKEY,#098 ;Load the key to enable JSRB
JSRB CBLOCKR
.ENDIF
.ENDIF
.ENDM
;; ------------------------------------------------------------------
;;Read a byte of data from VE2 archive
;;Parameters:
;;TO:Address in RAM memory; Symbol A handled as well
;;FROM:Address in VE2 from which data is read
;;Returned data is loaded into A after being read. If TO == <&quot>A<&quot>
;;then the datum is swapped with the addressed RAM location. The
;;S register must be valid for the TO address
;; ------------------------------------------------------------------
.MACRO VE2RD,TO,FROM
LD ISPADHI,#HIGH(FROM)
LD ISPADLO,#LOW(FROM)
LD ISPKEY,#098 ;Load the key to enable JSRB
JSRB CREADBF
LD A,ISPRD
.ifstr <&quot>TO<&quot> NE <&quot>A<&quot>
X A,LOW(TO)
.endif
.ENDM
;; ------------------------------------------------------------------
;;Write a byte of data to VE2 archive
;;Parameters:
;;TO:Base Address in VE2 to which data is written
;;FROM:A register, literal or memory segment offset address
;;Assumes S is valid for memory reference
;; ------------------------------------------------------------------
.MACRO VE2WR,TO,FROM
LD ISPADHI,#HIGH(TO)
LD ISPADLO,#LOW(TO)
.ifstr <&quot>FROM<&quot> NE <&quot>A<&quot>
LD A,FROM
.endif
X A,ISPWR
LD ISPKEY,#098 ;Load the key to enable JSRB
JSRB CWRITEBF
.ENDM
```

11 VIRTUAL E^2 INTERFACE MECHANISMS

The following will be used to allow the user to interface directly to the routines in the boot ROM.

11.1 JSRB—Jump Subroutine in Boot ROM

Syntax:— JSRB ADDR

Description:— The JSRB instruction causes execution to begin at the address specified within the first 256 bytes of the Boot ROM. The switch to Boot ROM is only successful if the JSRB instruction was immediately preceded by writing a valid key to the ISP KEY register. The instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address in Boot ROM. If the key has not been written, or if the key was invalid, the instruction jumps to the same address in program memory.

The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address is now saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, the values found in the second byte of the instruction are transferred to PCL. PCU is loaded with 0. The program then jumps to the program memory location accessed by PC in the Boot ROM, if the key write was successful, or in program memory if it was not.

Operation:— [SP] <- PCL
[SP - 1] <- PCU
[SP - 2]: SET UP FOR NEXT STACK REFERENCE
PC14-8 <- 00
PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Address Mode	Instruction Cycles	Bytes	Hex Op Code
JSRB ADDR	Absolute	5	2	61/LOADDR

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		