Freescale Semiconductor

AN1288

Programming the MC68HC(8)05K3′s Personality EEPROM on the MMDS and MMEVS

By Derrick B. Forte
CSIC Development Tools
Austin, Texas

Introduction

The Freescale MMDS and MMEVS development systems give embedded systems developers the ability to emulate members of the M68HC05 Family of microcontroller units. The MMDS/MMEVS host software command set provides users with read and write access to an emulated MCU's on-chip registers and peripherals. The design of some HC05 peripherals, however, precludes their being directly accessed by the MMDS/MMEVS. An example of such a peripheral is the personality EEPROM (PEEPROM) found on MC68HC(8)05K3 microcontrollers.

The PEEPROM consists of 128 bits of EEPROM memory arranged in an array of 16 rows x 8 columns. The PEEPROM's bits can be accessed only serially, one bit at a time, by software stored in on-chip user nonvolatile memory. An on-chip charge pump provides the voltage necessary to program or erase the PEEPROM as long as the MCU's supply voltage is 3 volts or higher. The PEEPROM can be used to store application data and variables. Possible uses for the PEEPROM include storage of codes for digital lock and key applications or seed values used in the implementation of random number algorithms.

freescale™
semiconductor

During the development process, a developer may want to initialize the MMDS/MMEVS's resident MC68HC(8)05K3's PEEPROM with values that may be permanent or temporary application data. Currently, the only means of programming the PEEPROM — apart from user application software — is with a standalone programmer available from Freescale. This application note will discuss the design and implementation of a DOS software utility, hereafter referred to as K3EEPROG, that enables a developer to program or read the PEEPROM of a MC68HC(8)05K3 that is the resident MCU on an MMDS or MMEVS development system. The basic design of K3EEPROG can be modified and applied in the creation of other utilities that interface the operation of the MMDS/MMEVS with the DOS environment.

Scope

It is assumed that the readers of this application note are acquainted with the basic operation of Motorola's MMDS or MMEVS development systems and the command set of P&E Microcomputer Systems, Inc.'s host computer software for the MMDS or MMEVS. For further information, refer to the *MMDS Operations Manual* (Freescale document number MMDS0508OM/D) or the *MMEVS Operations Manual* (Freescale document number MMEVS0508OM/D). Since the M68EM05K3 emulator module will be used with the development systems, additional information on the M68EM05K3 can be obtained from its user manual (Freescale document number M68EM05K3U/M). Familiarity with the C programming language also is recommended.

AN1288

2

Alternate Solutions

As mentioned earlier, the design of the MC68HC(8)05K3's PEEPROM prevents the MMDS/MMEVS development systems from providing users with direct access to this peripheral. Unlike with emulator pseudo-RAM or on-chip EEPROM, users cannot modify or read the contents of the PEEPROM with instructions from the MMDS/MMEVS command line. Two features of the PEEPROM's design in particular prevent the MMDS and MMEVS systems from gaining access to the PEEPROM. The first feature is that the PEEPROM data can be accessed only one bit at a time through the manipulation of on-chip registers by software stored in user memory space. Secondly, the PEEPROM is not a part of the MC68HC(8)05K3's memory map. This requires that alternative methods be found to provide users with access to this peripheral during the development process. Before considering the design and implementation of K3EEPROG, two alternate methods of accessing the PEEPROM will be discussed in this application note.

The simplest method of accessing the PEEPROM on an MMDS's or MMEVS's resident MC68HC(8)05K3 is to remove the MCU from the M68EM05K3 emulator board, program or read the PEEPROM with the programmer provided for the MC68HC(8)05K3, and reinstall it on the M68EM05K3.

When a developer wants to read or program the PEEPROM more frequently, small assembly code routines can be written and kept for use whenever the PEEPROM is to be accessed. When the PEEPROM is to be programmed, the user can fill a designated section of the MMDS/MMEVS's memory with the data to be programmed into the PEEPROM. The assembly code module designed to program the PEEPROM then can be loaded and executed from the MMDS/MMEVS command line. The software then can read the user-entered data from the emulator's memory and program the PEEPROM with it. The same process can be used to read the contents of the PEEPROM, except instead of reading data from the designated memory, the assembly code module writes the contents of the PEEPROM into a designated area of the emulator's memory where it can be read by the user. This method can be automated by utilizing the MMDS/MMEVS's scripting feature.

3

Software Design Overview

The main purpose of the K3EEPROG utility is to enable a user to program or read the contents of the PEEPROM of an MC68HC(8)05K3 that is the resident MCU on an MMDS or MMEVS without removing the MCU from the development system. K3EEPROG uses the development system to do the actual programming and reading of the PEEPROM. Since all MMDS and MMEVS host computer software currently is DOS based, K3EEPROG is designed to execute from the DOS command line.

A simplified program flow of K3EEPROG is as follows:

1. If a user-created file called **Startup.05** exists, rename it as STARTUP.SAV.

2. Wait for the user to select a command from the main menu. The user can select one of three commands: **Program PEEPROM**, **Read PEEPROM**, and **Exit to DOS**.

3. If the user selects the **Exit to DOS** command, rename STARTUP.SAV as STARTUP.05 and exit to DOS.

4. Construct a temporary S-record file that can either program or read the PEEPROM depending on the user's command.

5. Construct a temporary MMDS/MMEVS **Startup.05** file to control the execution of the development system.

6. Start execution of the host computer software.

7. Delete the temporary **Startup.05** and S-record files.

8. Evaluate the results of the development system's execution. If errors are detected, display an error dialog box and return to the main menu.

9. If the PEEPROM was read, display its contents to the user and return to the main menu. If the PEEPROM was programmed, inform the user with a dialog box.

10. Return to the main menu.

AN1288

Figure 3 illustrates the main program flow of K3EEPROG.

To implement this program flow, functions must be implemented to perform the following tasks:

1. Provide a means to start execution of the MMDS/MMEVS host software.

2. Provide a means to control and stop the execution of the MMDS/MMEVS host software once its execution has started.

3. Communicate user commands and data to the MMDS/MMEVS.

4. Communicate data and system responses, which can evaluate and present them to the user, from the MMDS/MMEVS to K3EEPROG.

Since the number of functions that K3EEPROG is required to perform is small, a simple DOS text menu is sufficient as a user interface. The menu provides the user with three commands: **Program PEEPROM**, **Read PEEPROM**, and **Exit to DOS**. Simple text-based prompts and dialog boxes are used to display error messages, prompt for user data, and display data.

A simple user interface serves to complement the first design requirement, that of starting the MMDS or MMEVS operation from the DOS environment. DOS provides a means of doing this with its spawn feature. In DOS, a program can become dormant and spawn temporarily or start the execution of another program, while remaining resident in the computer's memory. After the callee executes and exits to DOS, execution of the caller resumes. The only constraint placed on the calling and called programs is that they both must be able to fit in the host computer's memory at the same time. This size constraint limits the complexity of the functionality of the utility and its user interface. This was the method selected to launch the host software of the development systems.

After spawning or starting the execution of the development system's host software, K3EEPROG becomes dormant but still must be able to control the execution of the host software, directing the system to either read or program the PEEPROM on the resident MCU. Also, when programming the PEEPROM, data entered by the user must be

transferred from K3EEPROG to the MMDS/MMEVS. When reading the PEEPROM, data must be transferred from the MMDS or MMEVS to K3EEPROG which can display it to the user. K3EEPROG also must be able to determine if the development system software executed properly or not so as to take appropriate action.

User commands and data are communicated from K3EEPROG to the MMDS/MMEVS by means of the host software's startup file feature. This feature frequently is used to automate the configuring of a development system at startup and take it to a predetermined state.

An MMDS/MMEVS startup file is an ASCII text file consisting of a list of MMDS/MMEVS commands. To be a valid startup file, a file must have the name **Startup.05** and must reside in the same directory as the host software executables. When operation of either MMDS or MMEVS is started, the host software eschews its resident directory for a startup file. If one is found, the development system executes each command in the file sequentially. Command execution continues until the end of the file is reached or an invalid command is encountered. Only then does the host software return to the MMDS/MMEVS host software command line. Nearly all MMDS/MMEVS commands can be executed from a startup file, making it possible to control the operation of both the MMDS and the MMEVS. Since startup files can contain the MMDS/MMEVS **EXIT** command, MMDS/MMEVS host software execution can be stopped and control returned to the DOS prompt without user intervention at the MMDS/MMEVS command line. Making effective use of this MMDS/MMEVS feature satisfies K3EEPROG's requirement to control and stop MMDS/MMEVS operation.

After programming or reading the PEEPROM, the MMDS/MMEVS host computer software exits to DOS and the execution of K3EEPROG resumes. It is at this time that K3EEPROG must evaluate the results of the MMDS/MMEVS's execution and in the case of a reading command, extract PEEPROM data for display to the user. To accomplish this task, K3EEPROG makes use of the logging feature of the MMDS/MMEVS host computer software. The logging feature allows the MMDS/MMEVS to record the execution of MMDS/MMEVS commands and the system's response to them. By using the MMDS/MMEVS memory display, **MD**, or the register display, **REG**, the contents of any area of emulator memory

AN1288

or the contents of MCU registers can be logged also. Part of the logging feature is the ability of the MMDS/MMEVS host software to write the results of a logging session to an ASCII text file. This text file provides the means for the MMDS/MMEVS software to communicate with K3EEPROG even though the host software has exited to the DOS prompt.

Software Implementation

K3EEPROG's implementation starts with the design of the code used to program and read the PEEPROM. As mentioned earlier, the PEEPROM is composed of 128 bits arranged in a 16 row x 8 column matrix. This arrangement is analogous to 16 bytes with each column corresponding to bit positions within the byte. Data can be accessed only one bit at a time through the manipulation of two registers: the PEEPROM bit select register (PEBSR) and the PEEPROM status/control register (PESCR). The PEEPROM bit select register, illustrated in **Figure 1**, is located at $0E and selects the next PEEPROM bit on which a PEEPROM read or programming operation is to be performed. An individual bit can be selected by setting bits 3–6 of the register to the row or byte in which the bit resides and setting bits 0–2 to the bit's location within the row or by writing the bit's position, 0–127, in the array.

Address: $0E

Bit 7	6	5	4	3	2	1	Bit 0
PEB7	PEB6	PEB5	PEB4	PEB3	PEB2	PEB1	PEB0
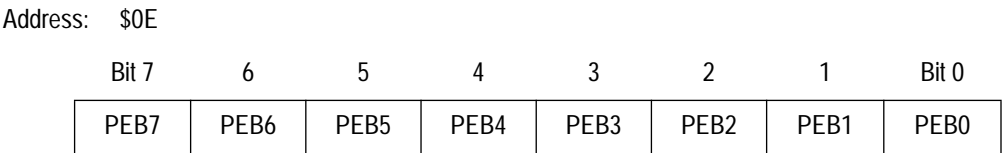
Figure 1. PEBSR Select Register

The PEEPROM status/control register (PESCR), illustrated in **Figure 2**, contains the control and status bits used in reading and programming the PEEPROM.

Address: $0F

Bit 7	6	5	4	3	2	1	Bit 0
PEDATA	PEBULK	PEPGM	PEBYTE	CPEN	CPCLK	0	PEPCZF

Figure 2. PESCR Status/Control Register

When programming the PEEPROM, K3EEPROG programs the entire array with 16 bytes of data entered by the user. K3EEPROG stores this data in locations $C0–$CF in the emulator's memory.The algorithm for programming one PEEPROM bit is outlined in the *General Release Specification for the MC68HC(8)05K3* (HC805K3GRS/D). Since K3EEPROG deals with data in bytes, the programming algorithm must be modified to program eight bits at a time. The PEEPROM facilitates this by providing the PEPCZF bit, bit 0, in the PESCR status/control register. This bit is set whenever a bit in the 0 column is being accessed. If the column 0 bit is taken to be the least significant bit of a row or byte, it is possible to differentiate between the end of one row and the start of the following row.

The algorithm developed for programming the PEEPROM is:

1. Load the row (byte) counter variable with a value of 16.

2. Bulk erase the PEEPROM.

   a. Set the CPEN bit in the PECSR control/status register.

   b. Set the PEBULK bit in the PECSR control/status register.

   c. Delay 30 milliseconds.

   d. Clear the CPEN bit in the PECSR control/status register.

   e. Clear the PEBULK bit in the PECSR control/status register.

3. Initialize the DATA_PTR variable.

4. Load the accumulator with data contained in the location pointed to by address $C0 plus the offset in the DATA_PTR variable.

AN1288

5. Jump to the byte programming routine.

   a. Shift out the least significant bit of the user data.

   b. If the bit to be programmed is a logical 1, set the PEPGM bit. Set the CPEN bit to enable the PEEPROM's on-chip charge pump.

   c. Delay for 10 milliseconds.

   d. Clear the PEPGM and CPEN bits.

   e. Increment the value in the PEBSR select register to point to the next bit to be programmed.

   f. If the PEPCZF bit is set in the PSCR status/control register, indicating that the least significant bit of the next row is being accessed, exit the programming routine.

6. Increment the value in the DATA_PTR variable to point to the next user data to be programmed.

7. Decrement the row counter. If the counter reaches 0, fall through to the PEEPROM verification routine. Otherwise, program the next row or byte.

After programming the complete array with the 16 bytes contained in locations $C0–$CF, the contents of each row are read back and compared with its corresponding value in the user data buffer. If a bit fails to verify, program execution is stopped at address $13F. If programming and verification are successful, program execution is stopped at address $13D. The contents of the PC register thus serves as a flag as to whether programming was successful. For a full listing of the code used to implement the programming algorithm, see **Assembly Code Modules**.

The algorithm for reading a PEEPROM bit is outlined in the *General Release Specification for the MC68HC(8)05K3* (HC805K3GRS/D).

The algorithm implemented in K3EEPROG is:

1. Clear the PEBSR bit select register to point to the first bit in the PEEPROM.

2. Load the row counter with a value of 16.

3. Clear the index register.

4. Read data from the current PEEPROM row.

   a. Rotate location $0F, the PECSR status/control register, to the left. Bit 7 of the register contains the data in the bit that is being accessed currently.

   b. Rotate the data bit into the accumulator.

   c. Increment the PEBSR bit select register to point to the next bit.

   d. Check to see if the PEPCZF bit, bit 0, in the PECSR status/control register is set. If the bit is set, all the bits of the current row have been read and the 0 column bit of the next row is being accessed. If the bit is clear, continue reading the bits from the row.

5. The accumulator now contains the data read from a complete row of the PEEPROM. Store this data in the location pointed to by address $C0 plus the current offset in the index register.

6. Increment the index register to point to the next address in the user data buffer.

7. Decrement the row counter variable. If the result is 0, all the rows in the PEEPROM have been read and the reading algorithm ends by unconditionally branching to address $11B. Otherwise, read the next row or byte.

For a full listing of the code used to implement the programming algorithm, see **Assembly Code Modules**.

K3EEPROG was written and compiled in C with Borland's C++ version 3.1 package. The program was linked with Borland's TLINK version 5.22b. K3EEPROG provides a simple interface for interaction with the user. The interface consists of a three-line text menu with three commands: **Program PEEPROM**, **Read PEEPROM**, and **Exit to DOS**.

AN1288

To use this utility, the user must have an MMDS or MMEVS development system configured to emulate an MC68HC(8)05K3. From a hardware perspective, this system consists of an MMDS control station or an MMEVS platform board, and an M68EM05K3 emulator board. Also required are the MMDS or MMEVS PC-based host computer software, and the MMDS/MMEVS device personality file designed to support the MC68HC(8)05K3. To function properly, the K3EEPROG executable must be resident in the same directory as the development system's directory.

The following is a detailed description of K3EEPROM's program flow:

1. K3EEPROG is invoked from the DOS command line with the following parameters: k3eeprog <development system (MMDS or MMEVS)> <COM port(COM1, COM2, COM3, COM4)>

2. K3EEPROG searches the current directory for the presence of the following files:

   a. 00014Vxx.MEM – The MMDS/MMEVS device personality file for the MC68HC(8)05K3.

   b. MMDS05.EXE or MMEVS05.EXE – Depending on the second parameter of the command line.

3. If the user fails to invoke K3EEPROG with the proper parameters or if any of the required files are not found in the current directory, an error dialog box is displayed and K3EEPROG exits to DOS.

4. K3EEPROG waits for a user to enter a command from a DOS text-based command menu. The user can select from one of the following three commands:

   a. Program PEEPROM.

   b. Read PEEPROM.

   c. Exit to DOS.

5. If the user decides to program the PEEPROM, the user interface prompts him/her for the data to be programmed.

AN1288

11

6. A temporary S-record file is created with the algorithm to either read or program the PEEPROM, depending on the user entered command. (See **Assembly Code Modules** for listings of the S-record used to program and read the PEEPROM.)

7. K3EEPROG searches the current directory for a user **Startup.05** file and if one is found, renames it as **Startup.SAV.** It then creates a temporary **Startup.05** file with MMDS/MMEVS commands that will be executed by the MMDS or MMEVS when its host software is spawned. The file created depends on the command to be carried out.

8. K3EEPROG spawns, or starts, execution of the MMDS or MMEVS host software. This temporarily suspends execution of K3EEPROG. However, it still remains resident in the host PC's memory. The development system finds the temporary **Startup.05** and executes the commands listed in it. The commands cause the development system to perform these tasks:

    a. If the user decided to program the PEEPROM, the data to be programmed into the PEEPROM is stored in locations $C0–$CF of emulator memory with the MMDS/MMEVS **MEMORY MODIFY (MM)** command.

    b. The temporary S-record file to read or program the PEEPROM is loaded into emulator memory using the MMDS/MMEVS **LOAD** command.

    c. Execute the MMDS/MMEVS **RESET** command.

    d. The development system starts execution of the algorithm code with the MMDS/MMEVS **GO** command.

    e. After waiting 1 second, execution of the algorithm code is stopped with the MMDS/MMEVS **STOP** command.

    f. An MMDS/MMEVS log file is opened with the MMDS/MMEVS **LOG FILE (LF)** command. An MMDS/MMEVS log file is the method used to transfer data between the development system and the DOS environment where it can be accessed by K3EEPROG. Every MMDS/MMEVS system command and

AN1288

the development system's response to the command from this point on will be directed to an MMDS/MMEVS text file called **K3EEPROG.LOG**.

g. The MMDS/MMEVS's **DISPLAY REGISTERS (REG)** command is called. The PEEPROM's programming and reading algorithms are designed so that the address stored in the program counter will indicate whether the task was carried out correctly.

h. If an attempt was made to read the PEEPROM, the MMDS/MMEVS **DISPLAY MEMORY (MD)** command is then called to display the contents of locations $C0–$CF. The PEEPROM's reading algorithm's code places the data that it read from the PEEPROM into this memory space.

i. The log file is closed by calling the **LOG FILE (LF)** again.

j. The MMDS/MMEVS **EXIT** command is called to exit the MMDS/MMEVS host software and return the user to DOS.

9. After the development system has completed a read or programming operation, the host software exits to DOS and execution of K3EEPROG resumes.

10. K3EEPROG evaluates the results of the development system's operation by opening and parsing the **K3EEPROG.LOG** file. The absence of a log file or the address recorded in the **K3EEPROG.LOG** signals to K3EEPROG that the MMDS/MMEVS failed to carry out the user's command. An error dialog box is displayed to inform the user of this fact after which program control is returned to the user menu.

11. If the user attempted to read the PEEPROM, K3EEPROG displays its contents to the user. Otherwise, the user is notified that the programming operation was successful.

12. The temporary **Startup.05** file, S-record file, and the **K3EEPROG.LOG** file are deleted, leaving no residual traces of K3EEPROG's execution. If a user startup file was present, it is renamed to **Startup.05**

13. K3EEPROG waits for another user command.

AN1288

13

Summary

K3EEPROG provides the user with a convenient way to program and read the PEEPROM of an MC68HC(8)05 during the development process. The K3EEPROG design can be modified to implement similar utilities that may be needed in the future to accommodate nonstandard MCU peripherals.

AN1288

Main Program Flowchart
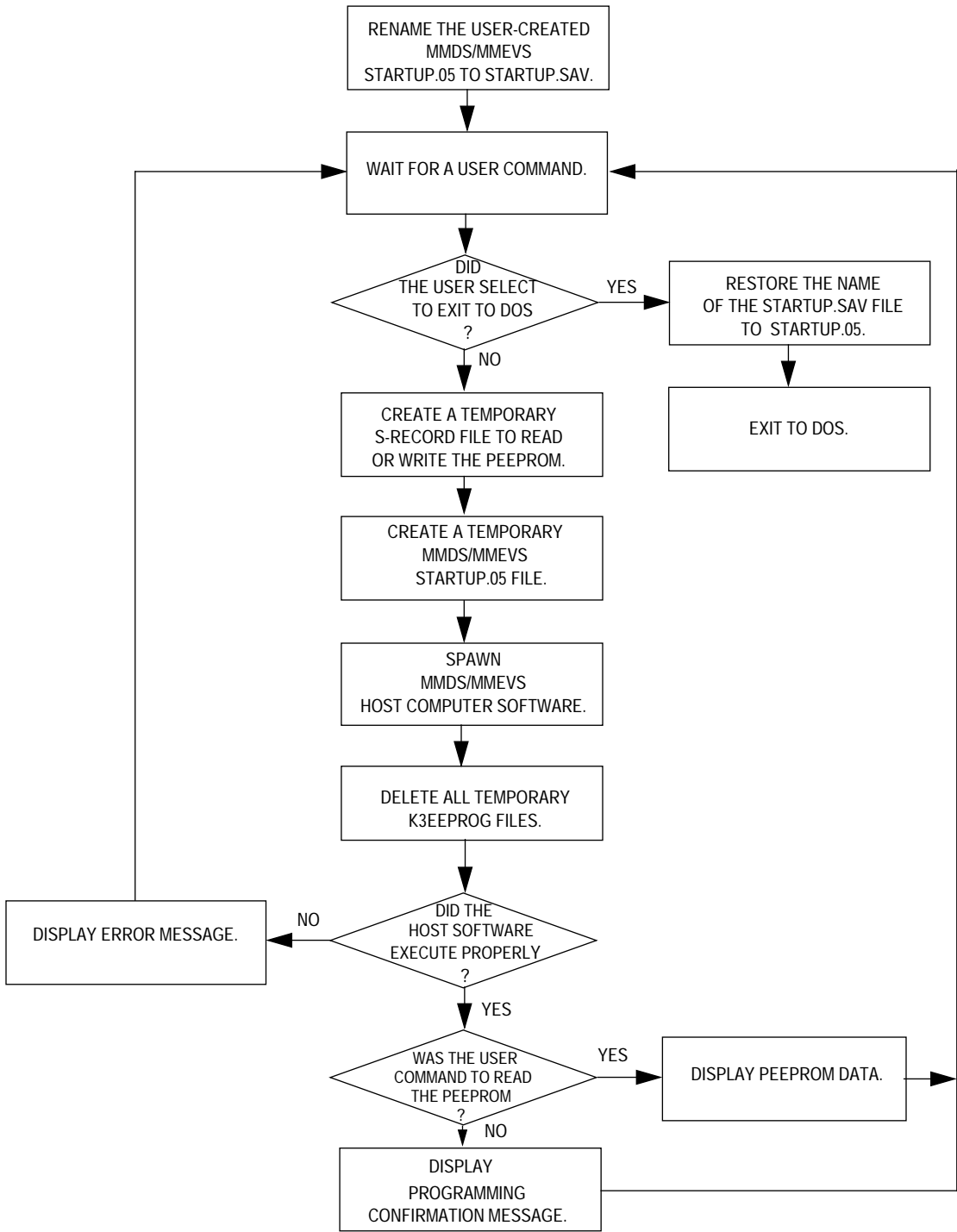
Figure 3. K3EEPROG Program Flowchart

AN1288

15

K3EEPROG Makefiles

Compile.Bat – Compilation and Link Batch File

```
bcc @compile.rsp
tlink @link.rsp
```

Compile.Rsp – Compilation Response File

```
-c -d -ml -Hu -v -Ic:\lang\borlandc\include
-j5  k3eeprog
```

Link.Rsp – Link Response File

```
/m  /v C0L k3eeprog
k3eeprog

c:\lang\borlandc\lib\cl c:\lang\borlandc\lib\emu
c:\lang\borlandc\lib\mathl
```

AN1288

Source Code

K3EEPROG.H
K3EEPROG
Header File

```
\*****************************************************************************
*                                                                           *
* Freescale reserves the right to make changes without further notice to any *
* product herein to improve reliability, function, or design. Freescale does *
* not assume any liability arising out of the application or use of any      *
* product, circuit, or software described herein; neither does it convey any *
* license under its patent rights nor the rights of others. Freescale        *
* products are not designed, intended, or authorized for use as components   *
* in systems intended for surgical implant into the body, or other          *
* applications intended to support life, or for any other application in     *
* which the failure of the Freescale product could create a situation where  *
* personal injury or death may occur. Should Buyer purchase or use Freescale *
* products for any such intended or unauthorized application, Buyer shall    *
* indemnify and hold Freescale and its officers, employees, subsidiaries,    *
* affiliates, and distributors harmless against all claims, costs, damages,  *
* and expenses, and reasonable attorney fees arising out of, directly or     *
* indirectly, any claim of personal injury or death associated with such     *
* unintended or unauthorized use, even if such claim alleges that Freescale  *
* was negligent regarding the design or manufacture of the part. Freescale   *
* and the Freescale Logo are registered trademarks of Freescale Semiconductor Inc.
*                                                                           *
/*****************************************************************************
*
         GLOBAL VARIABLE                                                     *
*                                                                           *
                                                                            *
*
*****************************************************************************/
#define read 0
#define program 1

// Emulator type variable: 0 - MMDS05, 1 - EVS05.
int emulator_type;

// Data verification flag.
int verified_data = 1;

// User data array.
char data[16] [3];
```

```
/*****************************************************************************
 *
 *              FUNCTION PROTOTYPES
 *
 *****************************************************************************/

// Constructs a startup file to program the resident MCU's PEEPROM.
int program_startup_file(void);

// Function calls the discrete functions that allow this program to read
// the resident MCU's PEEPROM.
void read_eeprom(char port[]);

// Constructs a startup file to read the resident MCU's PEEPROM.
int read_startup_file(void);

// Evaluates the results of a programming operation.
int evaluate_program_results(void);

// Evaluates the results of a read operation.
int evaluate_read_results(void);

// Allows the user to select the type of emulation system and COM port being
// used.
int system_setup(int command line_number, char command line_emulator[],
char command line_port[]);

// Allows the user to input the data to be programmed into the K3's PEEPROM.
void enter_data(void);

// Displays a box with user instructions.
void user_instruction_box(void);

// Displays an error box if MMDS05.EXE is not found in the current
// directory.
void no_mmds05(void);

// Displays an error box if the user enters an incorrect number or
// type of command line parameters.
void command line_error_box();

// Displays an error box if EVS05.EXE is not found in the current
// directory.
void no_evs05(void);

// Sounds a beep when an error is committed.
void beep(void);

// Displays an error box if the personality file for the K3 EM board is
// not found in the current directory.
void no_personality_file(void);
```

AN1288

```
// Checks for MMDS05 or EVS05, and 00014VXX.MEM. These files are necessary for
// the program to operate.
int files_present(void);

// Displays K3EEPROG's opening screen.
void main_screen(void);

// Displays the program's main menu.
char user_selection(void);

// Calls the discrete functions that program the K3's Personality EEPROM.
void program_eeprom(char port[]);

// Displays a message box that asks for verification of the user entered data.
int data_verification_box(int mode);

// Displays an error box informing the user that an emulator error has
//    occurred.
void emulator_error_box(void);

// Displays an error message box when the user enters an invalid menu choice.
void invalid_choice(void);

// Displays an error box informing the user that an error has occurred while
// programming the K3's Personality EEPROM.
void eeprom_error_box(void);

// Displays an error screen informing the user that a memory allocation error
// has occurred.
void memory_error_box(void);

// Displays a message box informing the user that the K3's Personality EEPROM
// has been successfully programmed with user data.
void programmed_box(void);

// Determines if a character entered by the user is a valid hexadecimal number.
int not_hex_number(char number);

// Prints a centered string on the screen.
void print_center(int y, char string[]);
```

AN1288

19

```
// Embedded S-record PEEPROM programming algorithm.
char S19_data[]  [45] = {"S1130100A603B7083F0EA610B7D6160F1C0FA61EDF",
                         "S1130110AD481D0F170F3FD7BED7E6C0AD233CD760",
                         "S11301203AD626F44F3F0E5FA610B7D6390F463C99",
                         "S11301300E010FF8E1C026075C3AD626EF20FE2018",
                         "S1130140FE442410160F1A0FB7D8A60AAD0CB6D861",
                         "S11301501B0F170F3C0E010FE881AEC69D9D9D5AE3",
                         "S10D016026FB4A26F54FC703F08181",
                         "S10503FE0100F8",
                         "S9030000FC"};

// Embedded S-record PEEPROM reading algorithm.
char read_data[] [45] = {"S1130100A603B7083F0EA610B7D64F5F390F463C7B",
                         "S11001100E010FF8E7C05C3AD626F120FE80",
                         "S10503FE0100F8",
                         "S9030000FC"};
```

AN1288

K3EEPROG Main
Program File
K3EEPROG.CPP

```
\*****************************************************************************
*                                                                          *
* Freescale reserves the right to make changes without further notice to any  *
* product herein to improve reliability, function, or design. Freescale does  *
* not assume any liability arising out of the application or use of any       *
* product, circuit, or software described herein; neither does it convey any *
* license under its patent rights nor the rights of others. Freescale         *
* products are not designed, intended, or authorized for use as components    *
* in systems intended for surgical implant into the body, or other            *
* applications intended to support life, or for any other application in      *
* which the failure of the Freesca;e product could create a situation where   *
* personal injury or death may occur. Should Buyer purchase or use Freescale  *
* products for any such intended or unauthorized application, Buyer shall      *
* indemnify and hold Freescale and its officers, employees, subsidiaries,      *
* affiliates, and distributors harmless against all claims, costs, damages,   *
* and expenses, and reasonable attorney fees arising out of, directly or      *
* indirectly, any claim of personal injury or death associated with such      *
* unintended or unauthorized use, even if such claim alleges that Freescale   *
* was negligent regarding the design or manufacture of the part. Freescale    *
* and the Freescale Logo are registered trademarks of Freescale Semiconductor Inc.
*                                                                          *
/*****************************************************************************


#include <dos.h>
#include <dir.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <process.h>
#include "k3eeprog.h"
 void main(int argc, char *argv[])
     {
      char choice; // user selection from the program's main window

      // Turn off the cursor.
      _setcursortype(_NOCURSOR);

      // Evaluate the user command line entries.
      if(system_setup(argc, argv[1], argv[2]))
          exit(0);
```

AN1288

```
// Check to see if EVS05 or MMDS05, and 00014VXX.MEM are present in the
// current directory. If either one of the files is missing, exit to DOS.
// Remove any residual K3EEPROG files if they are present.
if(files_present())
   exit(0);

// Display program opening screen.
main_screen();

// Display user instruction box.
user_instruction_box();

// Rename the user-created STARTUP.05 to a temporary new name.
rename("STARTUP.05","STARTUP.SAV");

// Display the user menu. Continue to process the user's commands until
// the EXIT command is selected.
do
 {
  switch(choice = user_selection())
  {
   case '1':
   clrscr();
   program_eeprom(argv[2]);
   break;
   case '2':
   clrscr();
   read_eeprom(argv[2]);
   break;
   case '3':
   break;
   default:
   invalid_choice();
   break;
  }

   // Clear the screen.
   clrscr();
   fflush(stdin);
 }while(choice != '3');
    // Restore the user-created STARTUP.05 to its original name.
    rename("STARTUP.SAV","STARTUP.05");

    // Clear screen.
    clrscr();

    exit(0);
}
```

AN1288

```
/*                        user_selection function
*
* Function input variables: None.
*
* Function outputs: an integer.
*                   1: The user has selected to program the Personality EEPROM.
*                   2: The user has selected to read the Personality EEPROM.
*                   3: The user has selected to exit the program.
*/

char user_selection(void)
    {
     char selection;

     printf("1) Program Personality EEPROM\n");
     printf("2) Read Personality EEPROM Bytes\n");
     printf("3) Quit\n");
     printf("\n\n");
     printf("Enter a selection: ");

     selection = getchar();
     return (selection);

    }

/*                        program_eeprom function
*
* Function input variables: char port[]
*
* Function outputs: None.
*
* This function programs the Personality EEPROM of the MMDS05's resident
* device.
*
*/
void program_eeprom(char port[])
    {
     int startup_file_present; // flag indicating that a MMDS05 startup file
     int flag;// is present
     int return_flag; //generic flag

     // Get the data to be programmed into the K3 from the user.
     // After the data has been entered, have the user verify it.
     // If the data is not correct, have the user re-enter the data.
    do
      {
       enter_data();
       return_flag = data_verification_box(program);
       if(return_flag == 1)
       return;
       }while(return_flag == -1);
```

```
// Construct a temporary STARTUP.05 file to program the device.
// If the file cannot be opened, display an error box and
// proceed with a program shut down.
if(program_startup_file())
  {
   // Display memory error screen.
   memory_error_box();

   // Restore the user-created STARTUP.05 to its original name.
   rename("STARTUP.SAV", "STARTUP.05");

   // Exit to DOS.
   exit(0);
   }

// Call either MMDS05 or EVS05 and wait for the result.
if(emulator_type)
    spawnl(P_WAIT, "evs05.exe","evs05", port, NULL);
else
    spawnl(P_WAIT, "mmds05.exe", "mmds05",port, NULL);

// Turn the off the cursor.
_setcursortype(_NOCURSOR);

// Delete the S-record programming file.
remove("K3PROG.S19");

// Delete the startup file.
remove("STARTUP.05");

// Evaluate the results of the emulator's execution.
// If the results are good, display a message box and exit to DOS.
// Otherwise display an appropriate error box and exit to DOS.
switch(evaluate_program_results())
      {
       case -1:
                    emulator_error_box();
       break;
       case 0:
       programmed_box();
       break;
       case 1:
       eeprom_error_box();
       break;
       default:
       break;
       }
```

AN1288

```
      // Remove the MMDS05 log file.
      remove("K3EEPROG.LOG");

      return;
      }

/*                             read_eeprom function
*
* Function input variables: char port[]
*
* Function outputs: None.
*
* This function read the Personality EEPROM of the MMDS05's resident
* device.
*
*/
void read_eeprom(char port[])
      {
      int startup_file_present; // flag indicating that a MMDS05/EVS05 startup
      int flag;// file is present
      int return_flag; //generic flag


      // Construct a temporary STARTUP.05 file to read the device.
      // If the file cannot be opened, display an error box and
      // proceed with a program shut down.
      if(read_startup_file())
        {
        // Display memory error screen.
        memory_error_box();

        // If a user startup file has been renamed, restore its proper name.
        rename("STARTUP.SAV", "STARTUP.05");


        // Exit to DOS.
        exit(0);
        }

      // Call either MMDS05 or EVS05 and wait for the result.
      if(emulator_type)
          spawnl(P_WAIT, "evs05.exe","evs05", port, NULL);
      else
          spawnl(P_WAIT, "mmds05.exe", "mmds05",port, NULL);

      // Turn the off the cursor.
      _setcursortype(_NOCURSOR);

      // Delete the S-record reading algorithm.
      remove("K3READ.S19");
```

```
        // Delete the startup file.
        remove("STARTUP.05");

        // Evaluate the results of the emulator's execution.
        // If the results are good, display a message box and exit to DOS.
        // Otherwise display an appropriate error box and exit to DOS.
        switch(evaluate_read_results())
                {
                 case -1:
                  emulator_error_box();
                  break;
                 case 0:
                  data_verification_box(read);
                  break;
                 case 1:
                  emulator_error_box();
                  break;
                  default:
                  break;
                }

    // Remove the MMDS05 log file.
    remove("K3EEPROG.LOG");

    return;
    }



    /*                       system_setup function
    *
    * Function input variables: int command line_number;
    *                           char command line_emulator[];
    *                           char command line_port[];
    *
    * Function outputs: an integer;
    *                   0: If the system setup was successful.
    *                   1: If the system setup failed.
    *
    * This function determines emulation system and the COM port will be used by
    * K3EEPROG.EXE.
    *
    */
```

AN1288

```
int system_setup(int command line_number, char command line_emulator[],
            char command line_port[])
    {
     int flag=0; // function return flag
     int port;
     // Check the number of command line parameters to make sure that two or
     // three parameters are present.
     if((command line_number < 2) || (command line_number > 3))
         {
          command line_error_box();
          return(1);
         }

     // Check the emulator type parameter and set the emulator_type flag
     // appropriately.
     if(!strcmpi("MMDS05", command line_emulator))
         emulator_type = 0;

     else if(!strcmpi("EVS05", command line_emulator))
                     emulator_type = 1;
     else
         {
          command line_error_box();
          return(1);
         }

     // Check the COM port command line parameter.
     if(command line_number == 2)
         port = 1;
     else
         port = atoi(command line_port);
     if(port<1 || port > 4)
         {
          command line_error_box();
          flag=1;
         }

     return(flag);
     }
/*                         enter_data function
```

```
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function accepts data from the user to be programmed into the
* K3's personality EEPROM.
*
*/


void enter_data(void)
     {
      int i,j;
      char c;

      // Let the user enter the data to be programmed into the K3.
      for(i=0;i<16;i++)
          {
          data[i][3] = '\0';
          printf("Enter the data for bits %3.3d-%3.3d: ", (i*8), ((i+1)*8)-1);
          j=0;
          while(j<2)
                {
                 c = getch();
                 // Check to see if the user entered a valid hexadecimal number.
                 // If not, beep and do not accept the character.
                 if(not_hex_number(c))
                   beep();
                 else
                   {
                    data[i] [j] = c;
                    putchar(c);
                    j++;
                   }
                }
           printf("\n");
          }

      // Convert all user entered data to upper case.
      for(i=0;i<16;i++)
          strupr(data[i]);
      return;
      }
```

AN1288

```
/*                      program_startup_file function
 *
 * Function input variables: None.
 *
 * Function outputs: an integer;
 *                   0: If the startup file was successfully opened.
 *                   1: If the startup file failed to open.
 *
 *
 * This function constructs a temporary startup file to control emulator
 * operation so that it can program the PEEPROM.
 *
 */

int program_startup_file(void)
    {
     int i; // generic counter variable
     FILE *startup_file;  // temporary STARTUP.05 file pointer
     FILE *programming_file; // temporary S-record file to program PEEPROM.

     // Create a temporary S-record file to program PEEPROM.
     // If the file fails to open return a one.
     if((programming_file = fopen("K3PROG.S19","w")) == NULL)
         return(1);

     // Incorporate the S-record data into the file.
     for(i=0;i<9;i++)
         fprintf(programming_file,"%s\n",S19_data[i]);

     // Close temporary S-record programming file.
     fclose(programming_file);

     // Open temporary startup file. If it fails to open, return a one.
     if((startup_file = fopen("startup.05","w")) == NULL)
         return(1);

     // Incorporate the user data in the startup file.
     for(i=0;i<16;i++)
         fprintf(startup_file,"mm %02X %s\n", (0xC0+i), data[i]);

     // Commands to execute the S19 file.
     fprintf(startup_file,"load k3prog.s19\n");
     fprintf(startup_file,"reset\n");
     fprintf(startup_file,"br 13D\n");
     fprintf(startup_file,"br 13F\n");
     fprintf(startup_file,"g\n");
     fprintf(startup_file,"wait 2\n");
```

AN1288

29

```
        // Commands to open a MMDS05 log file.
        fprintf(startup_file,"stop\n");
        fprintf(startup_file,"lf k3eeprog.log\n");
        fprintf(startup_file,"reg\n");
        fprintf(startup_file,"lf\n");
        fprintf(startup_file,"exit");

        // Close startup file.
        fclose(startup_file);

        return(0);
        }

/*                        read_startup_file function
*
* Function input variables: None.
*
* Function outputs: an integer;
*                   0: If the startup file was successfully opened.
*                   1: If the startup file failed to open.
*
*
* This function constructs a temporary startup file to control emulator
* operation.
*
*/

int read_startup_file(void)
    {
    int i; // generic counter variable
    FILE *startup_file;  // temporary STARTUP.05 file pointer
    FILE *read_file; // temporary S-record file to read PEEPROM.

    // Create a temporary S-record file to read PEEPROM.
    // If the file fails to open, return a one.
    if((read_file = fopen("K3READ.S19","w")) == NULL)
        return(1);

    // Incorporate the S-record data into the file.
    for(i=0;i<4;i++)
        fprintf(read_file,"%s\n",read_data[i]);

    // Close temporary S-record programming file.
    fclose(read_file);

    // Open startup file. If it fails to open, return a one.
    if((startup_file = fopen("startup.05","w")) == NULL)
        return(1);
```

AN1288

```
            // Commands to execute the S19 file.
            fprintf(startup_file,"load K3READ.S19\n");
            fprintf(startup_file,"reset\n");
            fprintf(startup_file,"wait 1\n");
            fprintf(startup_file,"br $11B\n");
            fprintf(startup_file,"g\n");
            fprintf(startup_file,"wait 2\n");

            // Commands to open a MMDS05 log file.
            fprintf(startup_file,"stop\n");
            fprintf(startup_file,"wait 1\n");
            fprintf(startup_file,"lf k3eeprog.log\n");
            fprintf(startup_file,"reg\n");
            fprintf(startup_file,"md c0\n");
            fprintf(startup_file,"lf\n");
            fprintf(startup_file,"exit");

            // Close startup file.
            fclose(startup_file);

            return(0);
            }

/*                          files_present function
*
* Function input variables: None.
*
* Function outputs: an integer;
*               0: all the necessary files are present in the current directory.
*               1: one or more of the necessary files are missing.
*
* This function checks for either EVS05 or MMDS05, and 00014VXX.MEM. These files
* are necessary for this program to operate. Remove any residual files that may
* be left over if a previous running of K3EEPROG was unsuccessful.
*
*/
int files_present(void)
    {
    struct find_t fb;  // structure for the _dos_findfirst() function
    int flag = 0; // function return flag
```

```c
// Check to see if either MMDS05 or EVS05 are present in the current
// directory. If neither are, display an error box and exit to DOS.
if(emulator_type)
   {
    if(_dos_findfirst("EVS05.EXE", _A_NORMAL, &fb))
      {
       no_evs05();
       flag = 1;
      }
   }
else
   {
      if(_dos_findfirst("MMDS05.EXE", _A_NORMAL, &fb))
         {
          no_mmds05();
          flag = 1;
         }
   }
 // Check to see if the K3 personality file (00014XXX.MEM)
// is present in the current directory.
// If it isn't display an error box and exit to DOS.
if(_dos_findfirst("00014*.MEM", _A_NORMAL, &fb))
   {
    no_personality_file();
    flag = 1;
   }

// If the program's log file is present, remove it.
if(!_dos_findfirst("K3EEPROG.LOG", _A_NORMAL, &fb))
 remove("K3EEPROG.LOG");


// If the PEEPROM's programming algorithm is present, remove it.
if(!_dos_findfirst("K3PROG.S19", _A_NORMAL, &fb))
    remove("K3PROG.S19");

// If the PEEPROM's read algorithm is present, remove it.
if(!_dos_findfirst("K3READ.S19", _A_NORMAL, &fb))
    remove("K3READ.S19");

return(flag);
}
```

AN1288

```
/*                      evaluate_results_program function
*
* Function input variables: None.
*
* Function outputs: an integer;
*                   -1: If the emulator log file failed to open.
*                    0: If the K3's personality EEPROM was successfully
*                       programmed.
*                    1: If the K3's personality EEPROM failed to program.
*
* This function reads the log file made by the emulator and from the address
* of the PC register, determines if the EEPROM was successfully programmed or
* not. The addresses are as follows: $013D-EEPROM was correctly programmed,
* $013F-EEPROM failed to be programmed.
*
*/


int evaluate_program_results(void)
    {
     FILE *k3eeprog_log_file;  // file pointer to emulator log file.
     char linestring[80];  // generic character string.


     // Open emulator log file. If it fails to open, return a negative one.
     if((k3eeprog_log_file = fopen("k3eeprog.log","r")) == NULL)
     return(-1);

     // Find the line in the log file that contain the register contents.
     // When it is found, determine the location of the PC register.
     while(fgets(linestring,80,k3eeprog_log_file) != NULL)
           {
            if(strstr(linestring,"PC") != NULL)
           {
           // If the PC address is $13D return a zero.
           if(strstr(linestring,"13D") != NULL)
         return(0);

     // If the PC address is $13F return a one.
     else if(strstr(linestring,"13F") != NULL)
     return(1);

     // If neither address is found, return a negative one.
     else
     return(-1);
           }
           }

     return(-1);
       }
```

```
/*                      evaluate_read_results function
*
* Function input variables: None.
*
* Function outputs: an integer;
*                  -1: If the emulator log file failed to open.
*                   0: If the K3's personality EEPROM was successfully
*                      read.
*                   1: If the K3's personality EEPROM failed to be read.
*
* This function reads the log file made by the emulator and from the address
* of the PC register, determines if the EEPROM was successfully read or
* not. The addresses are as follows: $011B-EEPROM was correctly read,
* any other address- EEPROM failed to be read.
*
*/


int evaluate_read_results(void)
    {
     FILE *k3eeprog_log_file;  // file pointer to emulator log file.
     char linestring[80];  // generic character string.
     int i; // counter variable


     // Open emulator log file. If it fails to open, return a negative one.
     if((k3eeprog_log_file = fopen("k3eeprog.log","r")) == NULL)
         return(-1);

     // Find the line in the log file that contain the register contents.
     // When it is found, determine the location of the PC register.
     while(fgets(linestring,80,k3eeprog_log_file) != NULL)
          {
             if(strstr(linestring,"PC") != NULL)
               {
               // If the PC address is $11B return a zero.
               if(strstr(linestring,"11B") != NULL)
             {
             // Find the memory display command.
             fgets(linestring,80,k3eeprog_log_file);
             fgets(linestring,80,k3eeprog_log_file);
             fgets(linestring,80,k3eeprog_log_file);
             strcpy(&linestring[0], &linestring[6]);
```

AN1288

34

Freescale Semiconductor, Inc.

```
                    // Get the data from the log file.
                    for(i=0;i<16;i++)
                        {
                        data[i][2] = '\0';
                        strncpy(data[i],linestring,2);
                        strcpy(&linestring[0], &linestring[3]);
                        }

                        return(0);
                        }

                // If the PC address is any other address return a one.
                else
                    return(1);
                }
            }
      return(1);
      }
/*                       not_hex_number function
*
* Function input variables: char number;
*
* Function outputs: an integer;
*                   0: If the character is a hex number.
*                   1: If the character is not a hex number.
*
* This function determines if a character entered by the user is a valid hex
* number.
*
*/


int not_hex_number(char number)
    {

    // Return a one if the character is a valid hexadecimal number,
    // otherwise return a zero.
    if((('a'<= number) && (number <= 'f')) ||
         (('0'<= number) && (number <='9')) ||
         (('A'<= number) && (number <='F')))
          return(0);
    else
       return(1);
    }
```

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************
*                                                                           *
*                           MESSAGE BOX ROUTINES                            *
*                                                                           *
*****************************************************************************/

/*                        main_screen function
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function displays the program's opening screen.
*
*/

void main_screen(void)
     {
      int i; // counter variable

     // message box character array
char text[7][80]={"****************************************************\n",
*                                                            *\n",
*      Freescale MC68HC05K3 Personality EEPROM Programming Utility      *\n",
*                   Alpha Version 1.0                        *\n",
*                     Copyright (c) 1995                     *\n",

*                                                            *\n",

"****************************************************************************\n"};

     // Clear screen.
     clrscr();

     // Display message box on the screen.
     for(i=0;i<7;i++)
          print_center(i+9,text[i]);
     print_center(16,"PRESS ANY KEY TO CONTINUE");

     // Wait for any key to be pressed.
     while(!getch())
             ;

     // Close the opening screen.
     clrscr();

     return;
     }
```

AN1288

Freescale Semiconductor, Inc.

```c
/*                          user_instruction_box function
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function displays a box giving the user program operating instructions.
*
*/

void user_instruction_box(void)
    {
     int i; // counter variable

     // message box character array
char text[6][80]={"********************USERINSTRUCTIONS***********************",

    *                                                           *",
    * Connect your MMDS05 or EVS05 to a serial port of the host computer.   *",
    *                Turn on your MMDS05 or EVS05.                          *",

    *                                                           *",

    ***************************************************************************"};


     // Clear screen.
     clrscr();

     // Display error box.
     for(i=0;i<6;i++)
         print_center(i+9,text[i]);

     print_center(16,"PRESS ANY KEY TO CONTINUE");


     // Wait for the user to hit any key.
     while(!getch())
             ;

     // Close error box.
     clrscr();

     return;
     }
```

```
/*                          data_verification_box function
*
* Function input variables: int mode.
*
* Function outputs: an integer;
*                      0: If the user data is correct.
*                      1: If the user data is incorrect and must be re-entered.
*
* This function displays a message box prompting the user to verify
* the data that is to be programmed into the device.
*
*/


int data_verification_box(int mode)
    {
      int i;  // generic counter variable
      char c;
      int return_flag; // function return value

      // message box character array
char text[11][80] = {"******** YOU HAVE ENTERED THE FOLLOWING DATA: **********",
          ** THE RESIDENT DEVICE CONTAINS THE FOLLOWING DATA:        **",

          *                                                          *",

          *                                                          *",

          *                                                          *",
          *        Press <ENTER> if the data is correct.             *",
          *          Press <ESC> to abort this command.              *",
          *        Press any other key to re-enter data.             *",
          *      Press any key to return to the main menu.           *",

          *                                                          *",

          **********************************************************"};

      char data_string[80];

      // Clear the screen.
      clrscr();

      // Display verification box.
      if(mode)
          print_center(1,text[0]);
      else
          print_center(1,text[1]);
```

AN1288

```
for(i=2;i<4;i++)
    print_center(i,text[i]);
sprintf(data_string,"*       Bits(0-7):      %s      Bits(64-71):      %s     *",
        data[0], data[8]);
print_center(4,data_string);
sprintf(data_string,"*                                                       *");
print_center(5,data_string);
sprintf(data_string,"*       Bits(8-15):     %s      Bits(72-79):      %s     *",
        data[1], data[9]);
print_center(6,data_string);
sprintf(data_string,"*                                                       *");
print_center(7,data_string);
sprintf(data_string,"*       Bits(16-23):    %s      Bits(80-87):      %s     *",
        data[2], data[10]);
print_center(8,data_string);
sprintf(data_string,"*                                                       *");
print_center(9,data_string);
sprintf(data_string,"*       Bits(24-31):    %s      Bits(88-95):      %s     *",
        data[3], data[11]);
print_center(10,data_string);
sprintf(data_string,"*                                                       *");
print_center(11,data_string);
sprintf(data_string,"*       Bits(32-39):    %s      Bits(96-103):     %s     *",
        data[4], data[12]);
print_center(12,data_string);
sprintf(data_string,"*                                                       *");
print_center(13,data_string);
for(i=0;i<3;i++)
    {
    sprintf(data_string,"*       Bits(%d-%d):    %s      Bits(%d-%d):  %s     *",

            (i*8)+40,(i*8)+47,data[i+5],(i*8)+104,(i*8)+111,data[i+13]);
     print_center((i*2)+14,data_string);
     sprintf(data_string,"*                                                       *");
     print_center((i*2)+15,data_string);
     }

if(mode)
  {
  for(i=5;i<8;i++)
      print_center(i+15,text[i]);
  for(i=9;i<11;i++)
      print_center(i+14,text[i]);
  }
else
  {
  for(i=8;i<11;i++)
      print_center(i+12,text[i]);
  }
```

AN1288

```
        // Wait for a key to be pressed. If the <ENTER>key is pressed,
        // set the verified_data flag to zero.
        // Otherwise, return to the user data entry screen.
        while(!kbhit())
              ;
        c = getch();

        if(mode)
          {
           switch(c)
            {
           case 0x0D:
              return_flag = 0;
           break;
           case 0x1B:
              return_flag = 1;
           break
           default:
              return_flag = -1;
           break;
           }
           }

        // Close error box.
        clrscr();

        return(return_flag);
        }
/*                      programmed_box function
*
* Function input variables: None.
*
* Function outputs: None.
*
*
* This function displays a box informing the user that K3's personality
* EEPROM has been successfully programmed.
*
*/

void programmed_box(void)
      {
       int i; //counter variable
```

```c
// message box character array
char text[5][80]={"****************************************************",
        "*                                                          *",
        "*      The resident MC69HC05K3's Personality EEPROM has been     *",
        "*       programmed.                                        *",
        "*                                                          *",

        "****************************************************";

    // Clear screen and sound beep.
    clrscr();
    beep();

    // Display error message box.
    for(i=0;i<5;i++)
        print_center(i+9,text[i]);
    print_center(15,"PRESS ANY KEY TO EXIT TO DOS");

    // Wait for any key to be pressed.
    while(!getch())
            ;

    // Close error message box.
    clrscr();

    return;
    }

/****************************************************************************
*                                                                          *
*                    ERROR INFORMATION BOX ROUTINES                        *
*                                                                          *
****************************************************************************/

/*                      invalid_choice function
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function displays a error message box if the user enters an invalid
* entry in the command menu.
*/
```

```
void invalid_choice(void)
    {
     int i; // counter variable

     // message box character array
     char text[7][80] =              {"************************************",
                                "*                                  *",
                                "*          Invalid choice.         *",
                                "*    Please enter 1, 2, or 3.      *",
                                "*                                  *",
                                  "************************************",
                                   "Press <RETURN> to continue."};

     // Clear the screen and beep.
     clrscr();
     beep();

     // Display error message box.
     for(i=0;i<6;i++)
        print_center(i+9,text[i]);

     print_center(16,text[6]);

     // Wait for the user to press the RETURN key.
     while((getch() != '\r'))
            ;

     // Clear screen.
     clrscr();
     return;
     }
/*                       no_evs05 function
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function displays an error box if EVS05.EXE is not found in the
* current directory.
*
*/

void no_evs05(void)
    {
     int i; // counter variable
```

AN1288

```
                // message box character array
                char text[6][80] ={"******************** ERROR **************************",

                                   "*                                                   *",
                                   "* EVS05.EXE was not found in the current directory.  *",
                                   "*  This program requires EVS05.EXE to operate.       *",

                                   "*                                                   *",

                                   "****************************************************"};


        // Clear the screen and sound a beep.
        clrscr();
        beep();

        // Display error message box.
        for(i=0;i<6;i++)
              print_center(i+9,text[i]);
        print_center(16,"PRESS ANY KEY TO EXIT TO DOS");

        // Wait for any key to be pressed.
        while(!getch())
                ;

        // Close error box.
        clrscr();

        return;
        }

/*                           no_mmds05 function
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function displays an error box if MMDS05.EXE is not found in the
* current directory.
*
*/

void no_mmds05(void)
     {
      int i; // counter variable
```

```
        // message box character array
    char text[6][80] = {"********************* ERROR *************************",

                  "*                                                    *",
                  "* MMDS05.EXE was not found in the current directory.   *",
                  "*    This program requires MMDS05.EXE to operate.      *",

                  "*                                                    *",

                  "*****************************************************"};


        // Clear the screen and sound a beep.
        clrscr();
        beep();

        // Display error message box.
        for(i=0;i<6;i++)
             print_center(i+9,text[i]);
        print_center(16,"PRESS <RETURN> TO EXIT TO DOS");

        // Wait for any key to be pressed.
        while(!getch())
             ;

        // Close error message box.
        clrscr();

        return;
        }
/*                      no_personality_file function
*
* Function input variables: None.
*
* Function outputs: None.
*
* This function displays an error box if a K3 personality file
* found in the current directory.
*
*/
```

```
void no_personality_file(void)
    {
      int i; // counter variable

      // message box character array
char text[6][80]={"****************************ERROR****************************",

            "*                                                          *",
            "*        The device personality for the K3 was not found   *",
            "         in the current directory.                         *",
            "*        This program requires personality file 00014VXX.MEM *",
            "         to operate.                                       *",

            "*                                                          *",

            "************************************************************"};

      // Clear the screen and sound a beep.
      clrscr();
      beep();

      // Display error message box.
      for(i=0;i<6;i++)
           print_center(i+9,text[i]);
      print_center(16,"PRESS ANY KEY TO EXIT TO DOS");

      // Wait for the any key to be pressed.
      while(!getch())
             ;

      // Close error message box.
      clrscr();

      return;
      }
/*                      emulator_error_box function
*
* Function input variables: None.
*
* Function outputs: None.
*
*
* This function displays an error box informing the user that an error has
* occurred in the operation of the emulator.
*
*/
```

```c
void emulator_error_box(void)
    {
     int i; // counter variable

     // message box character array
     char text[5][80]= {"******************** ERROR **************************",

                        "*                                                  *",
                        "* An error has occurred in the emulator's operation. *",

                        "*                                                  *",

                        "****************************************************"};

     // Clear the screen and sound a beep.
     clrscr();
     beep();

     // Display error message box.
     for(i=0;i<5;i++)
          print_center(i+9,text[i]);
     print_center(15,"PRESS <RETURN> TO EXIT TO DOS");

     // Wait for any key to be pressed.
     while(!getch())
            ;

     // Close error box.
     clrscr();

     return;
     }
/*                       eeprom_error_box function
 *
 * Function input variables: None.
 *
 * Function outputs: None.
 *
 *
 * This function displays an error box informing the user that the K3's
 * personality EEPROM has failed to program.
 *
 */
```

```
void eeprom_error_box(void)
     {
      int i; // counter variable

//messagebox character array
char text[5][80]={"*********************ERROR********************************",

             "*                                                         *",
             "*          The resident MC68HC05K3's Personality EEPROM    ",
             "                has failed to program.                    *",
             "*                                                         *",

             "**********************************************************"};

     // Clear the screen and sound a beep.
     clrscr();
     beep();

     // Display error message box.
     for(i=0;i<5;i++)
          print_center(i+9,text[i]);
     print_center(15,"PRESS ANY KEY TO EXIT TO DOS");

     // Wait for any key to be pressed.
     while(!getch())
            ;

     // Close error message box.
     clrscr();

     return;
     }
/*                   command line_error_box function
*
* Function input variables: None
*
* Function outputs: None
*
* This function displays a screen informing the user that he/she has
* made an error in entering command line parameters.
*
*/
```

```c
void command line_error_box(void)
    {
      int i; // counter variable

      // message box character array
      char text[11][80]={"************************ERROR**********************",

      "*                                                         *",

      "*                                                         *",
      "* Incorrect command line parameters were entered.         *",
      "*            Use the following syntax to envoke this program:    *",
      "*               k3eeprog [Emulator Type] [COM Port].      *",
      "* Type the following for the Emulator Type option: MMDS05 or EVS05   *",
      "* Type the following for the Com Port option: 2 COM2 3 COM3 4 COM4   *",
      "*                (COM1 is the default)                    *",

      "*                                                         *",

      "****************************************************************"};

      // Clear the screen and sound a beep.
      clrscr();
      beep();

      // Display error box.
      for(i=0;i<12;i++)
          print_center(i+7,text[i]);
      print_center(18,"PRESS ANY KEY TO EXIT TO DOS");

      // Wait for the user to hit any key.
      while(!getch())
            ;

      // Close the error box.
      clrscr();

      return;
      }

/*                      memory_error_box function
 *
 * Function input variables: None
 *
 * Function outputs: None
 *
 * This function displays a screen informing the user that he/she has
 * made an error in entering command line parameters.
 *
 */
```

AN1288

```c
void memory_error_box(void)
     {
     int i; // counter variable

     // message box character array
     char text[5][80] = {"******************************************",
                         "*                                        *",
                         "* A memory allocation error has occurred *",
                         "*                                        *",

                         "******************************************"};

     // Clear the screen and sound a beep.
     clrscr();
     beep();

     // Display error message box.
     for(i=0;i<5;i++)
          print_center(i+9,text[i]);
     print_center(15,"Press any key to exit to DOS.");

     // Wait for any key to be pressed.
     while(!getch())
             ;

     return;
     }
/*                          beep function
 *
 * Function input variables: None.
 *
 * Function outputs: None.
 *
 * This function sounds a beep when an error is committed.
 *
 */

void beep(void)
     {
     sound(500);
     delay(100);
     nosound();
     return;
     }
```

```
/*                              print_center function
*
* Function input variables: int y;
*                           vertical position at the string will be printed.
*                           char string[];
*                           string to be centered and printed on the screen.
*
* Function outputs: None.
*
* This function prints the character string passed to it in the center of the
* screen.
*
*/
void print_center(int y, char string[])
     {
      // Position the string in the center of the string.
      gotoxy (40 - (strlen(string)/2), y);

      // Print the string to the string.
      printf("%s",string);
     }
```

AN1288

Assembly Code Modules

K3PROG.ASM

```
PEBSR     EQU      $000E            ; PEBSR Bit Select Register
PECSR     EQU      $000F            ; PECSR Status\Control Register
PEPCZF    EQU      0                ; Column 0 flag bit
CPEN      EQU      3                ; Charge pump enable bit
PEPGM     EQU      5                ; PEEPROM Program bit
PEBULK    EQU      6                ; PEEPROM Bulk Erase bit
PEDATA    EQU      7                ; PEEPROM Data bit



          org      $00C0
DATA          rmb      16           ; User data buffer
COUNTER       rmb      1            ; PEEPROM row counter
DATA_PTR      rmb      1            ; User data offset pointer
TEMP          rmb      1            ; Temporary storage space



          org     $100

START     lda     #$03             ; Set cop to longest timeout
          sta     $08
          clr     PEBSR
          lda     #$10             ; Load counter with 16
          sta     COUNTER
          bset    CPEN,PECSR       ; Bulk erase the PEEPROM
          bset    PEBULK,PECSR
          lda     #$1E             ; Delay 30 msec
          bsr     DELAY
          bclr    PEBULK,PECSR
          bclr    CPEN,PECSR
          clr     DATA_PTR         ; Set offset to point to first data in
PROGLOOP  ldx     DATA_PTR         ; user data buffer
          lda     DATA,x           ; Load accumulator with data from $C0+offset
          bsr     PROGBYTE         ; Program byte
          inc     DATA_PTR         ; Increment the user data offset variable
          dec     COUNTER
          bne     PROGLOOP
```

AN1288

```
VERIFY          clra                ; Verify programming of the PEEPROM
                clr    PEBSR
                clrx
                lda    #$10
                sta    COUNTER
VERILOOP        rol    PECSR        ; Read the data in the current row
                rora
                inc    PEBSR
                brclr  PEPCZF,PECSR,VERILOOP
                cmp    DATA,x       ; Compare the read data with the user data
                bne    ERROR
                incx                ; Increment the offset to the user data buffer
                dec    COUNTER
                bne    VERILOOP
GOOD            bra    GOOD
ERROR           bra    ERROR


PROGBYTE        lsra                ; Shift out the least significant bit to the
                bcc    CLEARBIT     ; Carry bit
                bset   CPEN,PECSR   ; Set the charge pump enable bit
                bset   PEPGM,PECSR  ; Set PEEPROM programming bit
                sta    TEMP         ; Delay for 10msec
                lda    #$0A
                bsr    DELAY
                lda    TEMP
                bclr   PEPGM,PECSR
                bclr   CPEN,PECSR
CLEARBIT        inc    PEBSR        ; Continue programming bits until the PEPCZR
                brclr  PEPCZF,PECSR,PROGBYTE   ; bit is set
                rts


DELAY           ldx    #$C6
                 nop
DELOOP          nop
                nop
                decx
                bne    DELOOP
                deca
                bne    DELAY
                clra
                sta    $3F0
                rts


                org    $3FE
                fdb    start
```

K3READ.ASM

```
PEBSR    EQU    $000E              ; PEBSR Bit Select Register
PECSR    EQU    $000F              ; PECSR Control\Status Register


         org    $00C0
DATA     rmb    16        ; User data buffer
COUNTER  rmb    1         ; Byte counter

         org    $100

START    lda    #$03      ; Set cop to the longest timeout
         sta    $08
         clr    PEBSR     ; Point to the first bit in PEEPROM array
         lda    #$10
         sta    COUNTER   ; Initialize the byte counter to 16.
         clra
         clrx
LOOP     rol    PECSR     ; Read the data from the current row.
         rora
         inc    PEBSR
         brclr  0,PECSR,LOOP
         sta    DATA,X    ; Store the data read from the current row
         incx             ; in the user data buffer. An point to the
         dec    COUNTER   ; next location in the user buffer.
         bne    LOOP
END      bra    END

         org    $3FE
         fdb    start
```

K3EEPROG User's Manual

System Requirements

These items are required for K3EEPROG's operation:

- Freescale MMDS or MMEVS/08 development system

- FreescaleM68EM05K3 emulator module

- P&E's MMDS or MMEVS development system host computer software, including the device personality file for the MC68HC(8)05K3 MCU – 00014Vxx.MEM

- IBM AT or PS/2 compatible host computer capable of running P&E's MMDS or MMEVS host software

Consult the *MMDS Modular Development System Operations Manual* (MMDS0508OM/D) for installation and configuration instructions for the MMDS and its host software. For the MMEVS system, consult the *MMEVS Modular Evaluation System Operations Manual* (MMEVS0508OM/D). Installation and configuration instructions for the M68EM05K3 emulator module can be found in the *M68EM05K3 User's Manual* (M68EM05K3UM/D).

Software Installation

Copy K3EEPROG.EXE to the directory that contains the MMDS or MMEVS host computer executable files.

AN1288

54

Software Operation

K3EEPROG is invoked from the DOS command line with this string:

K3EEPROG <development system type> <COM port number>

Where:

The development system type parameter may be MMDS or EVS05 depending on the system being used.

The COM port number parameter may be 1, 2, 3, or 4 to designate the communications port used.

Example: K3EEPROG MMDS 2.

After the program starts, follow the directions given in the program's dialog boxes and prompts.

For More Information On This Product,
Go to: www.freescale.com