Freescale Semiconductor
Application Note

Document Number: AN3404
Rev. 1, 03/2007

How to Do EEPROM Emulation Using Double Flash Array on MC9S08LC60

by: Ronald Gonzalez and Tatiana Orofino
RTAC Americas

1 Introduction

This application note explores how to do EEPROM emulation using two arrays. This is demonstrated through an example that retrieves data from SCI, stores the data into emulated EEPROM, and shows EEPROM status and data on the display.

Flash memory is intended primarily for program storage. In-circuit programming allows the operating program to be loaded into the flash memory after the application product's final assembly.

Data storage is becoming common in flash arrays because of new technologies that support longer data retention and higher rates of write cycles.

The MC9S08LC60/36 is the first Freescale 8-bit microcontroller to contain two flash arrays. Program and erase operations can be conducted on one array while executing code from the other. This feature allows easy EEPROM emulation while the microcontroller runs. This can improve the layout and eliminate external components, which reduces costs.

Contents

1.1 MC9S08LC60/36 Flash-Memory Characteristics

- Flash size:
  - — MC9S08LC60 — 63,232 bytes (30,464 bytes in flash B, 32,768 bytes in flash A)
  - — MC9S08LC36 — 36,864 bytes (12,288 bytes in flash B, 24,576 bytes in flash A)
- Single power supply program and erase
- Command interface for fast program and erase operation
- Up to 100,000 program/erase cycles at typical voltage and temperature
- Flexible block protection
- Security feature for flash and RAM
- Auto power-down for low-frequency read accesses minimizes run I_{DD}
- Flash read/program/erase over full operating voltage or temperature

1.2 MC9S08LC60/36 Memory Map

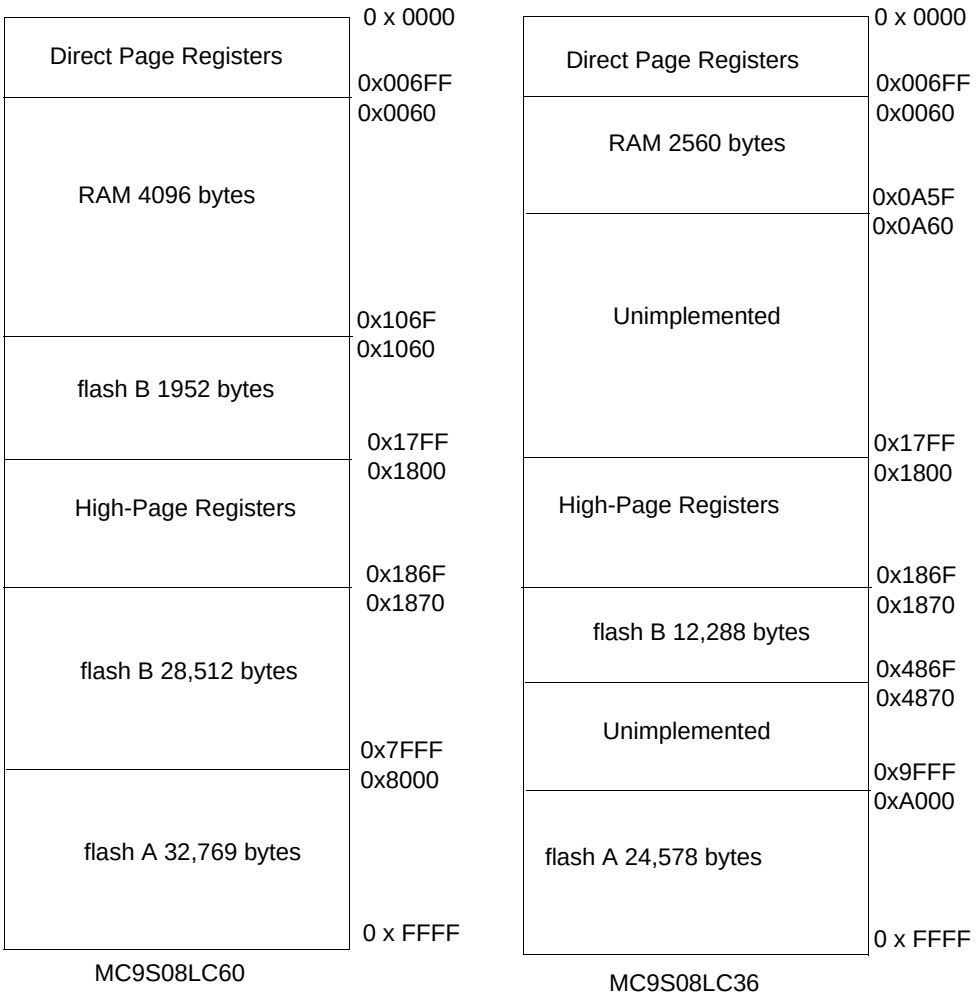

Figure 1. MC9S08LC60/36 Memory Map

2 Flash Programming

This flash-memory module includes integrated program/erase-voltage generators and separate command-processor state machines that can perform automated byte programming, page (512 bytes flash) or mass erase, and blank-check commands. Commands are written to the command interface. Status flags report errors and indicate when commands are complete.

The block-protection feature prevents the protected region of flash from accidental program or erase changes.

A security mechanism can be engaged to prevent unauthorized access to the flash and RAM memory contents. An optional user-controlled, back-door key mechanism can be used to allow controlled access to secure memory contents for development purposes.

2.1 Program and Erase Times

One advantage of emulated EEPROM is that program and erase times are very fast compared with other technologies.

Table 1 shows program and erase times. The shown times include overhead for the command-state machine and enabling and disabling of program and erase voltages.

Table 1. Program and Erase Times

Parameter	Cycles of FCLK	Time if FCLK = 200KHz
Byte program	9	45 µs
Byte program (burst)	4	20 µs[1]
Page erase	4000	20 ms
Mass erase	20.000	100 ms

[1] Excluding start/end overhead

2.1.1 Flash Clock

Before any program or erase command can be accepted, the flash clock divider register (FCDIV) must be written to set the internal clock for the flash module to a frequency (f_{FCLK}) between 150 kHz and 200 kHz.

NOTE

Observe that the flash clock is referenced as the bus clock.

2.1.1.1 Flash Clock Divider Register (FCDIV)

This register can be written only once, so normally this write is done during reset initialization. FCDIV cannot be written if the access error flag, FACCERR in FSTAT, is set. You must ensure that FACCERR is not set before writing to the FCDIV register. One period of the resulting clock (1/fFCLK) is used by the command processor to time program and erases pulses. To complete a program or erase command, the

command processor uses an integer number of these timing pulses. Bit 7 of this register is a read-only status flag. Bits 6 through 0 may be read at any time but can be written one time only. Before any erase or programming operations are possible, write to this register to set the clock frequency for the nonvolatile memory system within acceptable limits.

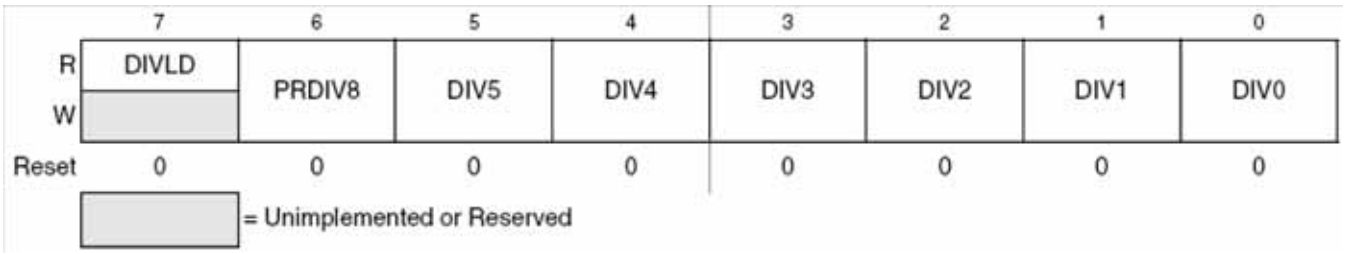

Table 2. FCDIV Field Description

Field	Description
7 DIVLD	Divisor Loaded Status Flag — When set, this read-only status flag indicates that the FCDIV register has been written since reset. Reset clears this bit, and the first write to this register causes this bit to become set regardless of the data written. 0 FCDIV not written since reset; erase and program operations disabled for flash. 1 FCDIV written since reset; erase and program operations enabled for flash.
6 PRDIV8	Prescale (Divide) flash Clock by 8 0 Clock input to the flash clock divider is the bus rate clock. 1 Clock input to the flash clock divider is the bus rate clock divided by 8.
5 DIV[5:0]	Divisor for flash Clock Divider — The flash-clock divider divides the bus-rate clock (or the bus-rate clock divided by 8 if PRDIV8 = 1) by the value in the 6-bit DIV5:DIV0 field plus one. The resulting frequency of the internal flash clock must fall within 200 kHz to 150 kHz for proper flash operations. Program/erase timing pulses are one cycle of this internal flash clock, which corresponds to a range of 5 µs to 6.7 µs. The automated programming logic uses an integer number of these pulses to complete an erase or program operation.

Table 3. Flash Clock Divider Settings

f_{Bus}	PRDIV8 (Binary)	DIV5:DIV0 (Decimal)	$^{f}FCLK$	Program/Erase Timing Pulse (5µs Minimum, 6.7 µs Maximum)
20 MHz	1	12	192.3 kHz	5.2 µs
10 MHz	0	49	200kHz	5 µs
8 MHz	0	39	200 kHz	5 µs
4 MHz	0	19	200 kHz	5 µs
2 MHz	0	9	200kHz	5 µs
1 MHz	0	4	200 kHz	5 µs
200 kHz	0	0	200 kHz	5 µs
150 kHz	0	0	150 kHz	6.7 µs

2.2 Program and Erase Flash Algorithms

All the program and erase algorithms, such as turn on/off charge pumps, control times, etc. are done by a hardware state machine that takes care about all processes without interfering in the microcontroller functionality. You must determine where the data is stored and launch the program sequence. The details of using these algorithms are explained in the following section.

2.2.1 Program and Erase Command Execution

Initialize the FCDIV register and clear error flags beofore command execution. Command-execution steps (see Figure 2 for flowchart):

1. Write a data value to an address in the flash array.
   — The address and data information from this write is latched into the flash interface. This write must be the first step in any command sequence. For erase- and blank-check commands, the the data value is not important. For page-erase commands, the address may be any address in the 512-byte page of flash to be erased. For mass-erase and blank-check commands, the address can be any address in the flash memory. Whole pages of 512 bytes are the smallest block of flash that may be erased. In some boundary conditions with RAM or high page registers, the accessible block size is less than 512 bytes.
2. Write the command code for the desired command to FCMD.
   — The five valid commands are blank check ($05), byte program ($20), burst program ($25), page erase ($40), and mass erase ($41).
   — The command code is latched into the command buffer.
3. Write a 1 to the FCBEF bit in FSTAT to clear FCBEF and launch the command (including its address and data information).
   — A strictly monitored procedure must be obeyed for the command to be accepted. This minimizes the possibility of any unintended changes to the flash memory contents. The command-complete flag (FCCF) indicates when a command is complete.

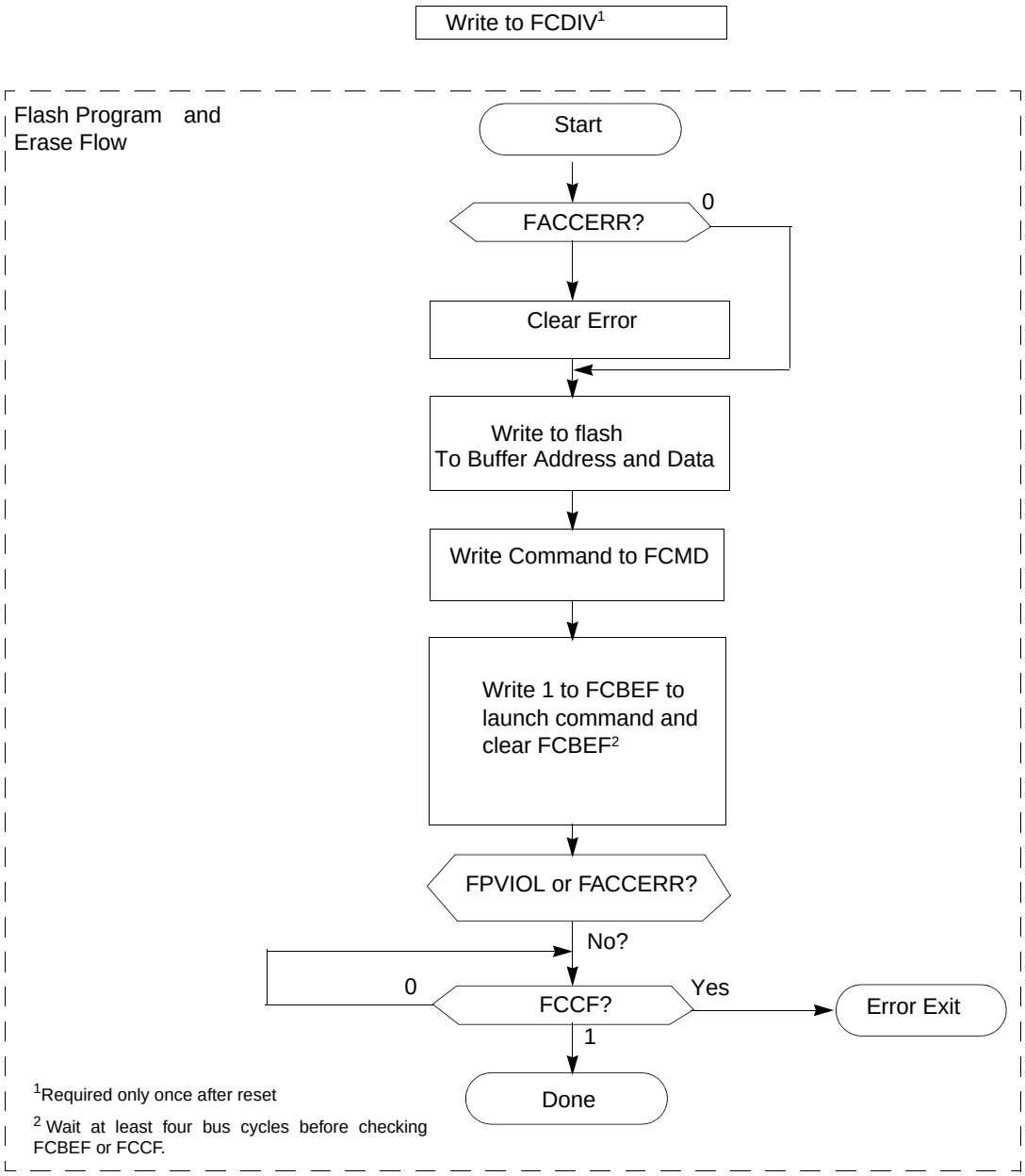

Figure 2. Program and Erase Command Execution Flowchart

2.2.2 Burst Program Execution

The burst program command is used to program sequential bytes of data in less time than would be required using the standard program command. For the MC9S08LC60/36, you can burst across flash-array boundaries as long as the addresses are consecutive. This is possible because the high voltage to the flash array does not need to be disabled between program operations. Ordinarily, when a program or erase command is issued, an internal charge pump associated with the flash memory must be enabled to supply high voltage to the array. Upon completion of the command, the charge pump is turned off. When a burst

program command is issued, the charge pump is enabled and remains enabled after completion of the burst program operation if two conditions are met:

- The next burst program command has been queued before the current program operation completes.
- The next sequential address selects a byte on the same physical row as the current byte being programmed. A row of flash memory consists of 64 bytes. A byte within a row is selected by addresses A5 through A0. A new row begins when addresses A5 through A0 are all zero.

Programming the first byte of a sequential bytes series in burst mode takes as long as having a byte programmed in standard mode. Subsequent bytes are programmed in the burst program time, provided the above conditions are met. When the next sequential address is begins a new row, the program time for that byte becomes the standard time instead of the burst time because the high voltage applied to the array must be disabled and then enabled again. If a new burst command is not queued before the current command completes, the charge pump is disabled and high voltage is removed from the array.

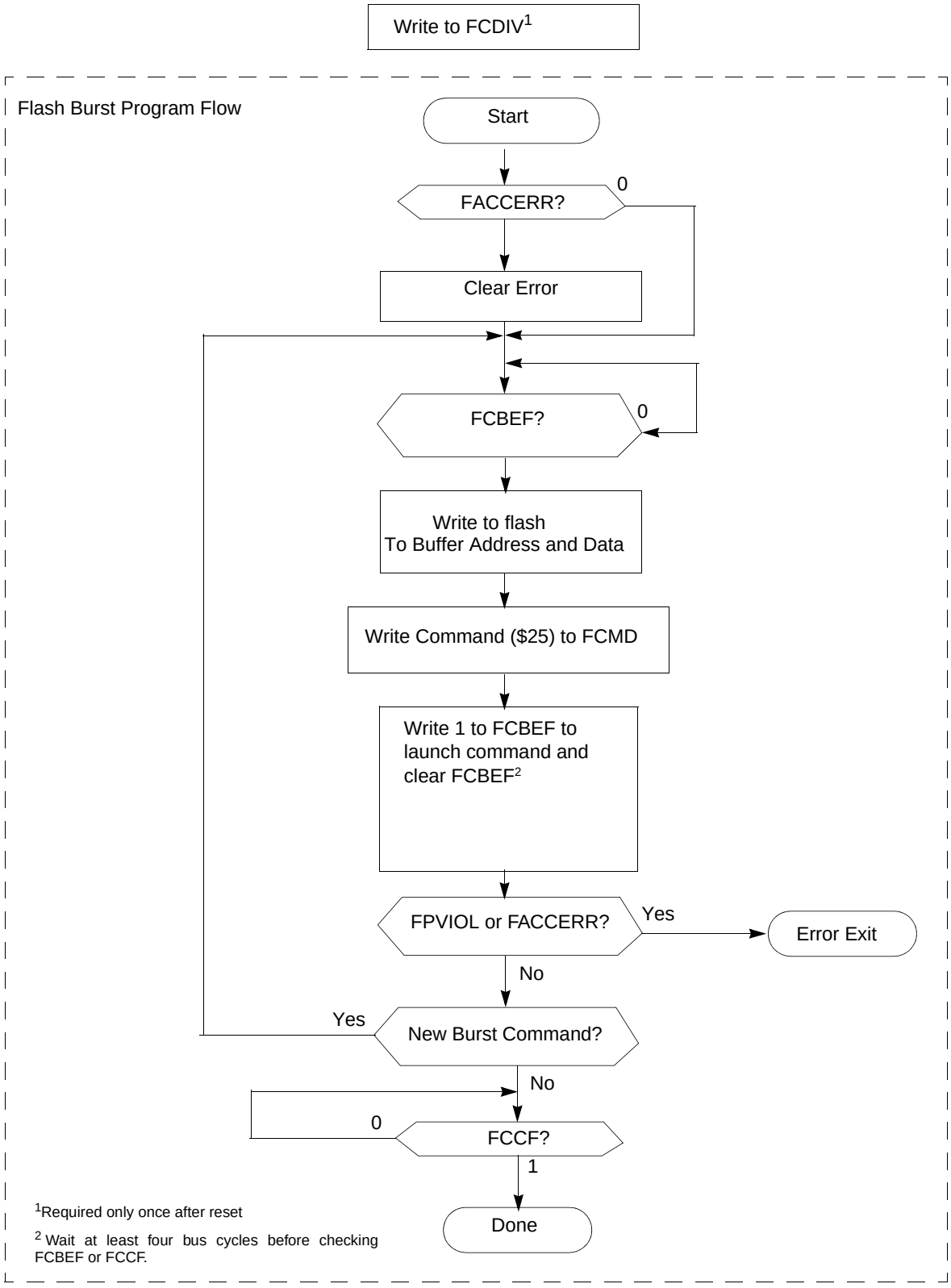

Figure 3. Burst Program Execution Flowchart

2.3 Flash Block Protection

This feature is very important when EEPROM emulation is used because it prevents an accidental flash program or erase outside of EEPROM reserved area. In our example, we protect all flash A (0x8000 – 0xFFFF) array reserved to program.

Block protection is controlled through the flash-protection register (FPROT). When enabled, block protection begins at any 512 byte boundary below the last address of flash, 0xFFFF.

After exit from reset, FPROT is loaded with the contents of the NVPROT location in the nonvolatile register block of the flash memory. FPROT cannot be changed directly from application software so a runaway program cannot alter the block-protection settings. Because NVPROT is within the last 512 bytes of flash, if any amount of memory is protected, the application software cannot alter (intentionally or unintentionally) NVPROT. FPROT can be written through background debug commands, which allow a way to erase and reprogram a protected flash memory.

The block-protection mechanism is illustrated in Figure 4. The FPS bits are used as the upper bits of the last address of unprotected memory. This address is formed by concatenating FPS7:FPS1 with logic 1 bits as shown. For example, to protect the last 8192 bytes of memory (addresses 0xE000 through 0xFFFF), the FPS bits must be set to 1101 111. This results in the value 0xDFFF as the last address of unprotected memory. In addition to programming the FPS bits to the appropriate value, FPDIS (bit 0 of NVPROT) must be programmed to logic 0 to enable block protection. Therefore, the value $DE must be programmed into NVPROT to protect addresses 0xE000 through 0xFFFF.

Figure 4. Block-Protection Mechanism

3 EEPROM Emulated Demo

After understanding how the program- and erase-flash commands work, we apply these concepts to EEPROM emulation. The DEMO9S08LC60 board and SCI and LCD display demo software from Softec were used in this example. This EEPROM emulation software presents a two-option menu for password storing and verification. You can set a new password and verify the current password saved in the emulated EEPROM by pressing buttons PTC7 and PTC5 from the DEMO9S08LC60 board.

The display is used in this demo as a user interface for all software processing. First, the program slides the options through the display and waits for any input:

```
void menu(){
  // Prints a welcome string
  SlideString("PTC7 - New psswd",250);
Delay(3000);

  SlideString("PTC5 - Verify psswd",250);
  Delay(3000);

SlideString("Make your choice", 250);
}
```

When you press PTC7 or PTC5, an interrupt routine launches and checks which button was pressed. Either functionality expects to receive as input an eight alphanumeric password. Hit ENTER. This format is mandatory; otherwise, software does not recognize the typed data.

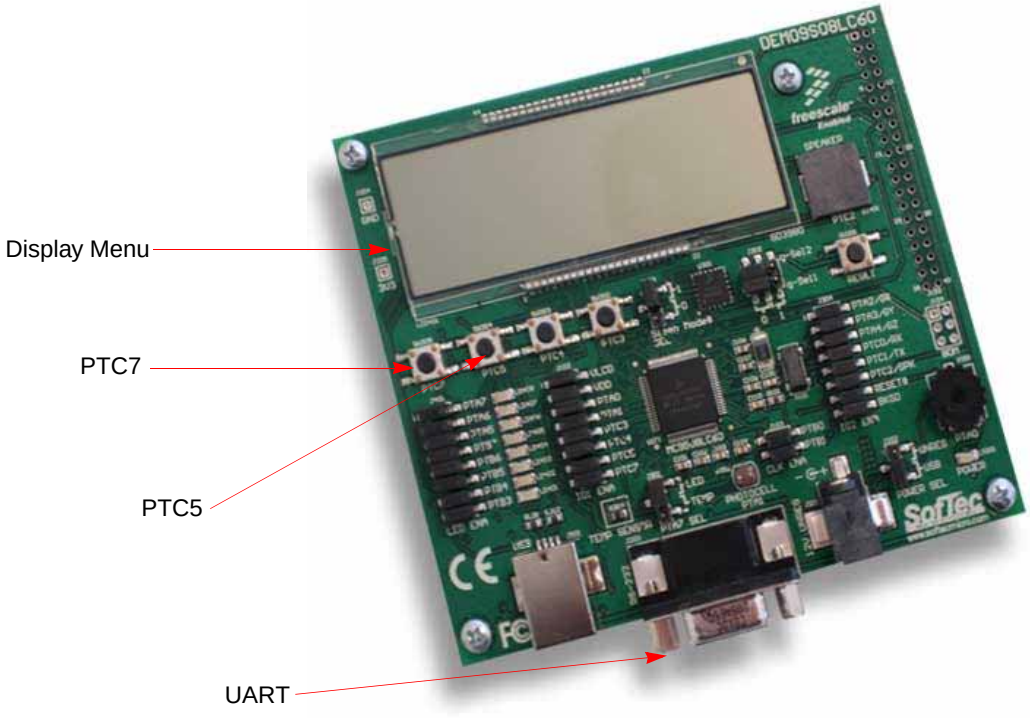

Figure 5. DEMO9S08LC60 Board

3.1 Display

The display driver and all the source code used are the provided with the DEMO9S08LC60 because it supports the display used on the hardware.

This application note basically uses 9 alphanumeric characters available in the display. These alphanumeric characters show the data read from SCI and the system message, “MEMORY FULL”.

3.2 SCI Interface

This microcontroller has an SCI interface implemented in hardware, that is easily configured through eight 8-bit registers that control baud rate, select SCI options, report SCI status, and for transmit/receive data.

The software configures the SCI interface as follows:

- BAUD rate — 9600
- Data bits — eight
- Parity — None
- Stop bits — one

Softec has implemented software for interfacing SCI. All data is received through this interface using the `SCIGetBuffer()` functions, which waits for an SCI input and stores it in a buffer.

NOTE

When connecting the MCU to an external peripheral, it is important to set the right voltages according with the interface used.

3.3 Data Storage

The 0x1870 - 0x286F address range is used for data storage, divided into eight blocks of 512 bytes (this is the minimum erasable block). Each block is separated into small blocks of eight bytes because it is the quantity of characters received from SCI.

The mechanism used is the following:

Supposing the EEPROM reserved area is clear at the first time the microcontroller is turned on, the first eight bytes received are stored in the first eight addresses from EEPROM (0x1870 – 0x1877).

For storing the next eight bytes received from SCI, the software finds the next open address in the EEPROM (containing 0xFF) and assumes this position as the first address of the next where these 8 bytes are stored.

When the EEPROM is full, the address 0x286F is reached. The routine then erases this block and starts writing at 0x1870 again.

To clear the memory at any time, press switch SW305. The eight pages are cleared.

For data storage, a simple code `void NVM_Write_Byte (unsigned char *dest, char data)` from EEPROM.c is executed, following the suggested program execution flowchart from Figure 3.

Following is the code for verifying and validation the address to be written into:

```
NVMStartAddress = &EEPAdrs; // pointer to start address


        do
        {
          *NVMStartAddress++;
          data = *NVMStartAddress;
        }
        while(data != 0xFF);   //looks for empty space in EEPROM

        //Checks if reached end of EEProm
        if (NVMStartAddress == &EEPAdrsEnd+1) {

          DisableInterrupts;

          // Erases EEPROM sector : Memory Full
          NVM_Erase_Page(&(*NVMStartAddress));      //EEPROM.c erasing routine

          EnableInterrupts;

          // reset pointer to EEPROM sector
          NVMStartAddress = &EEPAdrs;

        }

        if (FSTAT_FCBEF == 1)  //Check if the command buffer is empty
        {
          DisableInterrupts;

          //EEPROM.c writing routine
          NVM_Write_Byte (&(*NVMStartAddress), *p);

          *NVMStartAddress++;
         EnableInterrupts;

        }
```

3.4 Data Verification

When the PTC5 button is pressed, the main routine waits for the password input as eight alphanumeric characters input followed by ENTER. After that, the reading routine starts and restores the last saved password. If the password matches the input, the LCD display shows a success message.

```
        NVMStartAddress = &EEPAdrs;

        do
        {
          *NVMStartAddress++;
          data = *NVMStartAddress;
        }
        while(data != 0xFF);  //looks for empty space in EEPROM
```

```
NVMReadAddress = NVMStartAddress - 8; //sets address for last valid passwd typed

if (NVMReadAddress <= &EEPAdrs) {
  NVMReadAddress = &EEPAdrs + 1;
}

t = &messageTX;

for(i = 0; i != NVMRowSize; i++)
{
  NVM_Read_Byte(&(*NVMReadAddress), t); // stores in messageTX the read data
  t++;
  NVMReadAddress++;
  if ( messageTX[i]== 0xFF ) {
    messageTX[i] = 0;
  }
}
```

4 Conclusion

MC9S08LC60/36 has many features (LCD controller, IIC, SCI, A/D, analog comparator, etc.). One feature explored was the flash memory with two arrays. When implementing EEPROM emulations, you can program and erase the flash memory without stopping the application thanks to a state machine and two arrays of flash already implemented in the MCU hardware.

The MC9S08LC60/36 features make this MCU a perfect device for display applications and storage necessities.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3404
Rev. 1
03/2007

freescale™
semiconductor