AN1756

Demonstration of a Bootloader Method of Reprogramming FLASH on the MMC2107 Through Implementation of a Field Update System

By C. William Eno
Home Networking Group
Austin, Texas

Introduction

The MMC2107 is a general-purpose 32-bit microcontroller unit (MCU) that uses an M•CORE CPU (central processor unit) and contains 128 Kbytes of onboard FLASH. This application note shows how to program the FLASH without using a special FLASH programming setup. It also demonstrates how to use the FLASH programming routines by implementing a field update system requiring only a serial port and a PC.

Two separate programs are used to implement this field update system:

- A FLASH downloader that allows programming of the FLASH through an SCI (serial communications interface) port from a terminal program running on a PC
- A bootloader that is included in an application that allows a user to choose between running the FLASH downloader or running the “main” application

This application note also provides a detailed description of the FLASH programming model and a set of tools to program the FLASH. The SCI

and PIT (programmable interval timer) modules are also discussed briefly, and a set of tools for common usage of the SCI ports are presented.

Development System

The system used to write and debug these programs was a PC with Windows® running CodeWarrior® for M•CORE™ and a CMB2107 development board.

Background

The FLASH has a special programming algorithm that must be implemented in software. It involves varying programming voltages and programming pulse widths. The FLASH is arranged in eight 16-Kbyte blocks to form one 128-Kbyte array. Also, due to the nature of the FLASH and the fact there is only one array, it cannot be programmed from within itself; therefore, the FLASH downloader must be run from RAM. In an application, the bootloader accomplishes this by copying a block of data that contains the FLASH downloader from the FLASH into RAM and then executing it.

Figure 1 represents a memory map of the first part of the FLASH as used in the example application. It shows the location of the exception vector table, user supplied SCI information, the FLASH downloader code, and the main program.

The first word of FLASH is the reset, or boot, vector. The CPU fetches the address stored at this location and then jumps to it, in this case the beginning of the main program. The SCI data is the port, clock frequency, and baud rate that the SCI is using. The FLASH downloader is actually compiled separately and then included with the rest of the application. The information about the SCI is put in a known location so the downloader can find and use it. If the SCI information changes in the rest of the application, the FLASH downloader does not be need to be recompiled. This is explained in greater detail in **FLASH Downloader**.

Windows is a registered trademark of Microsoft in the U.S. and other countries. CodeWarrior is a registered trademark of Metrowerks, Inc., a wholly owned subsidiary of Motorola, Inc. M•CORE is a trademark of Motorola, Inc.

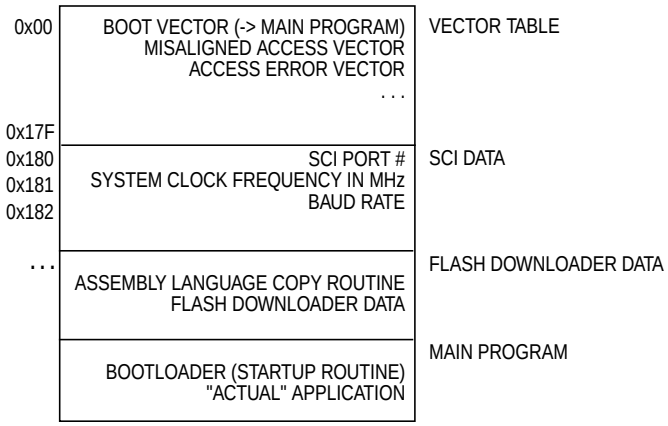


Figure 1. Memory Map of FLASH as Used in Example Application

The bootloader is run as the first routine of the main program. If the user chooses not to run the FLASH downloader, it returns to the main program and runs the rest of the application. If the user selects the FLASH downloader, an assembly language copy routine is called which copies the downloader into RAM and then executes it.

Module Descriptions

Both the bootloader and the FLASH downloader share the same SCI communication routines. In addition, the bootloader uses the PIT module to keep track of the 10-second countdown timer.

SCI Routines

All the SCI routines are included in `\example\sci_util\sci_util.c`. To use the routines first call `setup_sci()` with the port, baud rate and clock frequency as parameters to set up the SCI. All the routines require the port to be sent as an argument. The SCI is set to 8/N/1 (8 bits, no parity, 1 stop bit).

The functions implemented are:

- `check_for_byte()` — See if a byte is available for read, return it if so; otherwise, return 0.
- `get_a_byte()` — Wait for a byte and then return it.

- `get_bytes()` — Wait for multiple bytes and return them in a character array.
- `send_byte()` — Send a byte.
- `send_string()` — Send a null terminated string.
- `setup_sci()` — Set up an SCI port.

More detailed descriptions of the functions can be found in the listing of `sci_util.c`. The comments include the specific methods for enabling the SCI and calculating the divisors for various baud rates and clock frequencies.

PIT Routines

One PIT is used in the bootloader to count off the seconds. The routines are in `\example\bootloader\bootloader.c`. One routine, `start_timer()`, accepts a clock frequency and the number of seconds to delay. It assumes a prescalar of 32,768 because accuracy is not critical. It then sets the timer to the right number of seconds of delay. The other routine, `timeup()`, checks to see if the timer flag is set. If it is not set, it returns false. If it is set, it disables the timer and returns true. Commented code is available in the listings for `bootloader.c`.

Description of the FLASH Programming and Erasing Routines

The FLASH erase routine described in this application note allows erasing of one block (16 Kbytes) of FLASH at a time or erasing the whole array. The FLASH programming routine allows one 64-byte page at a time to be programmed. All the FLASH control registers and program/erase algorithms are detailed in **Section 9. Non-Volatile Memory FLASH (CMFR)** of *MMC2107 Advance Information*, Motorola document order number MMC2107/D.

NOTE: *The descriptions presented here outline one specific implementation of the algorithms. Care should be taken if the routines are to be modified to conform to all program/erase specifications. The routines presented here have been tested and versions of them are used in production FLASH test cycling.*

The FLASH cannot really be over-erased, but implementing the wrong algorithm in programming can cause problems. And, while over-programming the FLASH does not really damage the FLASH, it will cause margin read failures and the integrity of the data programmed cannot be guaranteed.

The 128-K FLASH array is arranged in eight 16-Kbyte blocks, numbered zero to seven. The blocks are accessed (read) as 512 32-byte pages and programmed as 256 64-byte pages. The minimum resolution for erasing is a block and the minimum resolution of programming is a 64-byte page.

NOTE: *Due to the limitations of FLASH, program and erase operations cannot be performed with a system clock below 8 MHz.*

FLASH Register Descriptions

Three registers are used to program and erase the FLASH. The control bits related to the actual programming and erasing are described here. More detailed descriptions of the FLASH registers are available in **Section 9. Non-Volatile Memory FLASH (CMFR)** of *MMC2107 Advance Information*.

CMFRMCR — CMFR Module Configuration Register

The CMFRMCR controls the operation of the FLASH module. The shadow information enable (SIE) bit enables the shadow row for reading and programming. The protect bits (PROTECT[7:0]) allow FLASH blocks to be protected from programming and erasing. Also in this register are bits that can restrict various blocks to supervisor mode access only and bits that can restrict the blocks to data use only.

CMFRMTR — CMFR Module Test Register

The CMFRMTR controls the voltage levels applied during programming and erasing. The negative voltage range (NVR) and pulse amplitude width select (PAWS) bits are used in different combination to produce different voltages. The gate or drain/source select bit (GDB) bit control what part of the FLASH cell the programming and erase voltages are applied to.

CMFRCTL — CMFR High-Voltage Control Register

The CMFRCTL register controls the programming and erase operations of the FLASH. The high-voltage status (HVS) bit indicates if there are high-voltage operations taking place. The system clock range field (SCLKR[2:0]) bits control the scaling of the clock. The clock period exponent field (CLKPE[2:0]) bits and the clock period multiplier field (CLKPM[6:0]) bits are combined to produce the different lengths of programming and erase pulses. The BLOCK bits control which block the programming and erase voltage are applied to. The program or erase select (ERASE) bit selects programming or erasing. The start/end sequence (SES) bit starts a programming pulse.

Erasing

The process for erasing is described in **Section 9. Non-Volatile Memory FLASH (CMFR)** of *MMC2107 Advance Information*. Here an implementation of the required algorithm is described. This algorithm requires the manipulation of gate voltages and the changing of pulse widths to provide for optimal erasing. (See **Table 1**.)

NOTE: *The CMFR FLASH erased state is 0xFF (all 1s).*
The shadow row always erases with block 0.

Table 1. Required Erase Algorithm

Voltage Step	PAWS[2:0]	Negative Voltage Range	Pulse Width	No. of Pulses
–2 V	100	1	100 ms	1
–3 V	101	1	100 ms	1
–4 V	110	1	100 ms	1
–5 V	111	1	100 ms	1
–6 V	100	0	100 ms	1
–7 V	101	0	100 ms	1
–8 V	110	0	100 ms	1
–9 V	111	0	100 ms	20

Note: A margin read is required after each –9-volt pulse.

In **Table 1**, voltage step is the gate voltage which is selected by a combination of PAWS[2:0] (pulse amplitude/width select field) and the NVR (negative voltage range select) bit. The pulse width is set up in the CMFRCTL register.

The `bulk_erase()` routine takes the system clock frequency and a block to erase as parameters. If a block number greater than seven is passed in to the routine the whole array will be erased. The clock frequency is needed to calculate the registers values for the pulse width.

This flow follows the flowchart in the erase section of the FLASH specification.

1. Just to be sure, write the reset values into the CMFRCTL (CMFR high-voltage control) and CMFRMCR (CMFR module configuration) registers.
2. In the CMFRMTR (CMFR module test register), set the PAWS[2] bit to 1 to override the firmware modulation.

NOTE: *Although the FLASH module does contain a firmware erase and program algorithms, they should not be used, as it has not been updated to reflect the required algorithm. The MMC2107 is not tested using this algorithm, and reliability cannot be guaranteed if it is used, especially low temperature data retention. The advance information book for the part mentions the firmware, but the technical data book will not include references to it.*

3. Clear the PROTECT bits to protect none of the blocks.

NOTE: *There are two ways to control the erasing and programming of a block of FLASH. The PROTECT bits are set to protect a block of FLASH from being modified. The PROTECT bits themselves can be locked by the LOCKCTL bit in the CMFRMCR register, a write-once bit that can be disabled only by a master reset. Also, the BLOCKS bits are set to control which blocks the programming voltages get applied to. It seems a little redundant, but makes sense if you really want to control how the FLASH gets modified.*

4. Set up the clock pulse width.

   a. The first step to setting up the pulse width is to determine the correct clock scaling (R) for the frequency range in use and to set the SCLKR[2:0] to the appropriate value. See **Table 2**.

Table 2. System Clock Range

SCKLR[2:0]	System Clock Frequency (MHz)		Clock Scaling (R)
	Minimum	Maximum	
000	Reserved		
001	8	12	1
010	12	18	3/2
011	18	24	2
100	24	36	3
101	36	40	4
110 and 111	Reserved		

NOTE: *Maximum system clock frequency is 33 MHz.*

The clock scaling (R) is selected based on the frequency parameter and saved as a numerator, `clksc_num`, and a denominator, `clksc_den`. This avoids floating point math and makes sure that in subsequent integer math calculations that values are not truncated too soon.

   b. Then CLKPE[1:0] (clock period exponent field) is determined.

CLKPE[1:0] and the exponent (N) are determined from information in **Table 3**. Because this is an erase and ERASE will be set to 1, the range is chosen from the lower half of the table. Inspection of the table shows that all of the ranges would include 100 ms, but for increased accuracy the smallest range is selected (CLKPE[1:0] = 00), which chooses an exponent of 15 (N = 15).

Table 3. Clock Period Exponent and Pulse Width Range

ERASE	CLKPE [1:0]	Exponent (N)	Pulse Width Range for All System Clock Frequencies from 8.0 MHz to 33.0 MHz	
			Minimum 2^N x 1.25E – 7	Maximum(1) 2^N x 128 x 8.33E – 8
0	00	5	4.00 µs	0.34 ms
	01	6	8.00 µs	0.68 ms
	10	7	16.00 µs	1.36 ms
	11	8	32.00 µs	2.73 ms
1	00	15	4.096 ms	349.5 ms
	01	16	8.192 ms	699.0 ms
	10	17	16.39 ms	1.398 s
	11	18	32.77 ms	2.796 s

1. The maximum system clock frequency is 33 MHz.

c. Then CLKPM[6:0] (clock period multiplier field) is determined.

The formula as presented in the specification is

$$\text{Pulse width} = \text{system clock period} * R * 2^N * M$$

Where M is CLKPME[6:0], system clock period = 1/system clock frequency, and R and N were determined in the previous two steps.

In the `bulk_erase()` procedure the formula is reduced to

CLKPM = (clockfreq*1000000*clksc_den) / (10*clksc_num*32,768)

Note that the `clockfreq` is passed into the function in megahertz. 2^N or 2^{15} has been expanded to 32,768. The pulse width (100 ms) has been changed to the fraction 1/10. This has been arranged so that the integers do not get truncated too soon and an accurate result is received.

The desired pulse width (100 ms) is now correctly set up for the system clock frequency.

5. Set the ERASE bit in the CMFRCTL register.
6. Set the BLOCK bits according to the block parameter passed into the routine. In the supplied code, if the block number is greater than seven, all the bits are set to erase the whole array. If the block number is seven or less, only the corresponding block bit is set. The BLOCK bits are where the code controls where the erase voltages get applied.
7. To begin the erase sequence, the SES (start/end sequence) bit in the CMFRCTL register is set. This is followed by an interlock write, which is a write to any location in FLASH to be erased.
8. The actual erase pulses are then applied:
    a. Set the PAWS[2:0] and NVR bits to the appropriate values for the current pulse (see algorithm).
    b. Set the EHV (enable high voltage) bit in the CMFRCTL register.
    c. Wait for the HVS (high-voltage status) bit in the CMFRCTL register to clear.
    d. Clear the EHV bit.
    e. Perform margin read, if required.
    f. Repeat pulses until the margin read is successful or more than 20 pulses are applied.
9. At the end of the routine, the CMFRMCR and CMFRCTL bits are set back to their reset states.

A margin read consists of reading the whole area being erased and verifying that a certain electrical margin is achieved in a FLASH cell to guarantee reliability.

During the margin read, successfully erased bits read as a 1. According to the recommended algorithm, a margin read is required only after the first (–9 V) pulse because the first eight pulses virtually guarantee that the array is erased.

Programming

The programming algorithm follows a similar process as the erase algorithm. The main difference is the necessity to change the pulse widths at certain points. This section describes in detail the required algorithm shown in **Table 4**. Again, a complete description is available in **Section 9. Non-Volatile Memory FLASH (CMFR)** of *MMC2107 Advance Information*.

Table 4. Required Programming Algorithm

Voltage Step	PAWS[2:0]	Negative Voltage Range	Pulse Width	No. of Pulses
–2 V	100	1	250 µs	4
–3 V	101	1	250 µs	4
–4 V	110	1	250 µs	4
–5 V	111	1	250 µs	4
–6 V	100	0	50 µs	20
–7 V	101	0	50 µs	20
–8 V	110	0	50 µs	20
–9 V	111	0	50 µs	20
–9 V	111	0	100 µs	Any additional needed

Note: Margin reads are required after every pulse.

The `program_page()` routine takes as parameters the system clock frequency, the address to program in FLASH, and the address in the RAM of a 64-byte (page size) array.

This flow follows the flowchart in the program section of the FLASH specification:

1. Just to be sure, write the reset values into the CMFRCTL (CMFR high-voltage control) and CMFRMCR (CMFR module configuration) registers.
2. Set the PAWS[2] and GDB bits in the CMFRMTR registers. Clear the PROTECT bits to protect none of the blocks

3. Set up the clock pulse width.
   a. Clock scaling is determined identically to the erase flow.
   b. Then CLKPE[1:0] (clock period exponent field) is determined exactly as described in the erase flow.

      CLKPE[1:0] and the exponent (N) are determined from the information in **Table 3**. Because this is program, and the ERASE bit will be set to 0, the range is chosen from the upper half of the table. Inspection of the table shows that the first range (PE[1:0] = 00) includes the three values needed (250 µs, 100 µs, and 50 µs), which chooses an exponent of 5 (N = 5).

   c. Then CLKPM[6:0] (clock period multiplier field) is determined. This is also identical to the erase flow. Here, because three pulse widths are needed, the values are calculated and saved in the three variables — `clkpm_250us`, `clkpm_100us`, and `clkpm_50us`.

      CLKPM is then set to the initial value of 250 µs.

4. The BLOCK bits are now set so that the programming voltage gets applied only to the FLASH blocks to be programmed.

   This is determined by masking off the lower 13 bits of the address, which leaves the block number in bits [16:14]. This value is then shifted right by 14 to obtain a block number. Then a 1 is shifted left by the block number to get the mask needed.

```
reg_CMFRCTL.bit.BLOCK = ( 1 << ( (address & 0x0001c000) >> 14) )
```

5. The values to be programmed are then written to the page.

NOTE: *This is done a byte at a time in the code because word and half-word reads must be aligned appropriately and the boundaries cannot be guaranteed for the data in the RAM.*

6. SES bit is set to 1.
7. Pulse loop is then entered:
   a. Set the PAWS[2:0] and NVR bits to the appropriate values for the current pulse (see algorithm).

b. Set the EHV (enable high voltage) bit in the CMFRCTL register.

c. Wait for the HVS (high-voltage status) bit in the CMFRCTL register to clear.

d. Clear the EHV bit.

e. Perform margin read.

f. Repeat pulses until the margin read is successful or more than the maximum pulses are applied.

The programming margin read performs the same function as the erase margin read. During the margin read, a successfully programmed bit reads as 0. The critical thing to note is that at least one read in each half of the page must be read to perform the actual margin read for that half of the page. The algorithm stops as soon as it sees a non-programmed bit, but if a read has not yet occurred in the second half of the page, it does a dummy read to the second half of the page. Once again, the margin read verifies the electrical margin of the FLASH cells to guarantee reliability.

8. At the end of the routine, the CMFRMCR and CMFRCTL bits are set back to their reset states.

Bootloader

The bootloader is inserted at the beginning of a project and compiled with the user program to allow a choice between running either the main program or running the FLASH downloader. The bootloader scheme requires the MMC2107-based system to be connected to the serial communications port of a PC.

Functional Description

The bootloader communicates with the user through a serial connection. A terminal program such as HyperTerminal®, which is included with

HyperTerminal is a registered trademark of Hilgraeve Inc.

Windows, is required. The bootloader in this example allows the user to either wait 10 seconds to run the main program or to hit any key to run the FLASH downloader. If the user hits a key, the bootloader will run an assembly language routine that copies a block of specially formatted data into RAM and executes the instructions contained in the data. The confirmation portion of the code could be rewritten to look for a certain port pin driven high or low or some other external stimulus.

Perl Script s2asm.pl

The copied data is an assembly text file that has been converted from a S-record via a Perl script `s2asm.pl`. The data is the machine code of the FLASH downloader compiled to run in RAM. The conversion is necessary so the data can be compiled into the bootloader program, which is used from FLASH. The data is formatted similar to an S-record and consists of a location in RAM to copy to and the number of bytes to copy. The actual data follows that. This is repeated until an end-of-data signal is given to the assembly language routine in the form of an address of 0x99999999.

For a complete description, see the listing of `s2asm.pl`. To use this script, Perl must be installed on the PC. In **Notes**, see the Perl entry that tells how to acquire Perl, a free programming/scripting language.

Assembly Language Copy Routine

The copy routine is all in assembly and works solely from the registers. This is necessary because it is possible that the copy routine will copy the data over the stack.

The assembly language routine is stored in the file `copyblock.h`. It is named with `a.h` so that the file can be included in the CodeWarrior project and the compiler will not try to compile it by itself. It is pulled into the file `downloader.s` through an `#include` statement. Note that it keeps track of the page alignment in one register so it can always shift to the next full word (4-byte) boundary to read the next address. It executes starting at the address stored in the first word of the data. See the listing of `copyblock.h` for a more detailed description.

FLASH Downloader

The FLASH downloader allows a specially processed S-record to be downloaded to the microcontroller through the SCI port.

Use of the FLASH Downloader

The FLASH downloader can be used in two separate instances:

- Standalone FLASH downloader
- With the bootloader setup

Stand–Alone Mode

The only change that needs to be made to run the project standalone is the uncommenting of the `#define STANDALONE` if the program is to be used standalone. This `#define` is located at the beginning of `downloader.c` in the downloader project, as in this code snippet.

```
#include "sci_util.h"
#include "sikaFLASH.h"
#include "mmc2107.h"

/* uncomment STANDALONE to run the FLASH programmer by itself */
/* #define STANDALONE */
#define SA_PORT                                 1
#define SA_BAUD                                 9600
#define SA_FREQ                                 16
/******************/

/* comment out USING_EVB to disable the write to enable the Vpp on the EVB */
#define USING_EVB;
/******************/
```

Compiling for Inclusion with the Bootloader

Compiling for inclusion of the bootloader requires the `#define STANDALONE` be commented out. The actual use of the downloader remains the same.

With the CMB/EVB

Using the downloader on the CMB/EVB requires that the `#define USING_EVB` not be commented out. Also, the CMB/EVB must be in master mode (M0 = ON and M1 = ON) so that V_{PP} can be enabled; it cannot be enabled when the board is in single-chip mode. The downloader program actually sets up the chip selects so that the CMB/EVB can be used. If the board is not in master mode or the `#define USING_EVB` is commented out and the program is being run

on the CMB/EVB, the program will hang due to a transfer error. It is possible to provide your own V_{PP} to the CMB/EVB by removing a 0-Ω resistor and soldering in a header. (See **Notes** for details on how to do this.)

Using with Target Application

To use in a finished product, V_{PP} (+5 volts) must be supplied, and `#define USING_EVB` must be commented out. It shouldn't matter whether the part is in master mode or single-chip mode. Also, it is necessary to provide a user-defined PLL (clock) setup routine. A call to `user_set_PLL()` is put in the code when `#define USING_EVB` is commented out. The compile will generate an error if this routine does not exist or is not removed. This serves as a reminder to take care of the clock properly when using the downloader not on the EVB/CMB. The interface to the SCI ports also needs to be considered. An RS-232 transceiver is required between the PC and the SCI ports of the MMC2107. (See **Notes** for a possible transceiver setup.)

Processing the S-Record

Before using the downloader, the S-records for the new FLASH data must be processed for the downloader to recognize it. The data can be programmed only on page (64-byte) boundaries. This processing page aligns all the data and makes sure that only complete pages of data are sent. This processing is also done so that the terminal program can do a blind download with pauses between lines. Additionally, the processing reduces the size of the downloader programmer by eliminating the need to interpret and page align an S-record directly.

The processing program is a short Perl script included in the `.zip` file named `sikadown.pl`. The program is a command line program used with this format:

```
sikadown.pl inputfile.elf.s outputfile.txt
```

Where `inputfile.elf.s` is the S-record to convert and `outputfile.txt` is the name of the output file.

Setting Up the Downloader

Open a terminal program (such as HyperTerminal) and set up the connection to whatever COM port you want to use and the baud rate that the FLASH downloader program is set to (original value is 19,200). The COM port in the terminal needs to be set up in 8/N/1 mode with at least

a 75-ms pause between lines. See **Notes** about using HyperTerminal. The target board is then turned on.

Programming the FLASH

After the program is started, the terminal program should display:

```
MMC2107 (Sika) FLASH Programmer
:
```

Hitting the ? (question mark symbol) key will display a list of available options:

```
MMC2107 (Sika) FLASH Programmer
: ?
To download to FLASH:
  1. bulk erase if necessary
  2. Make sure the delay time between lines
     is set to at least 75 ms
  3. Type f
  4. Send S-record (processed with perl
      script "sikadown.pl") as a text file
f - Program FLASH
v - Verify FLASH
b - bulk erase all of the FLASH
:
```

The procedure to program the FLASH is described. Erase the FLASH if necessary. Type f (lowercase f), then download the processed S-record as a text file. If there is any error during the programming the downloader will report an error message and what page the error occurred on.

To additionally verify the FLASH programming was successful, type v (lowercase v) and resend the S-record. The downloader will check the file against the contents of the FLASH. If a page does not verify, it will show an error message.

Compiling and Using the Example Project

This section details how to compile the example program and the downloader. **Figure 2** shows the complete compile process all the way to a `.txt` file ready to be downloaded with the downloader.

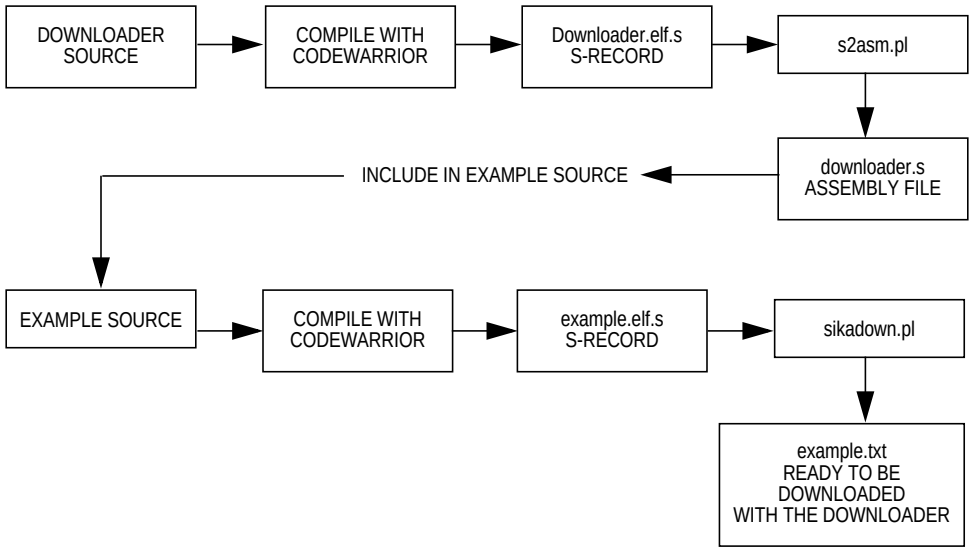


Figure 2. Compile Process

Compiling the Downloader for Use with the Bootloader

Follow these steps to compile the downloader.

NOTE: *Skip this section if the `downloader.s` file in the `example\bootloader` directory hasn't been changed.*

1. Make sure that the `#define STANDALONE` is commented out in the `downloader.c` file. This will force the program to get its baud rate and clock frequency information from the saved location in FLASH.
2. Compile the program. In this case, the output name should be `downloader.elf`. To set the name of the output file, go to the Base Project settings->Target->Mcore Target. In addition, make

sure that an S-record will be generated by checking Generate S-record File in Base Project settings->Linker->Mcore Linker.

NOTE: *The maximum size of the file should be 0, which indicates no limit on size*

3. The S-record generated will be named `downloader.elf.s`. This needs to be processed by a Perl script that converts it into a hex file for compiling into the main program. The Perl script is called `s2asm.pl` and is included in the `.zip` file. The script format is:

```
s2asm.pl downloader.elf.s downloader.s
```

   Where `downloader.elf.s` is the name of the input file and `downloader.s` is the name of the output file.

4. This will generate the `downloader.s` assembly file, which needs to be copied into the `example\bootloader` directory.

NOTE: *The output name is important because the start label in the assembly is generated from it, and the code that copies the downloader into RAM depends on that label.*

A portion of the generated code is:

```
/* File: downloader.s */
/* Generated by s2asm v0.1 9/27/2000 AM */


#include "copyblock.h"
                          .data
                          .export                 _downloader_s_start
_downloader_s_start:
                          .long                   0x00800000
                          .byte                   0xFA
                          .byte                   0x00
                          .byte                   0x80
                          .byte                   0x00
                          .byte                   0x04
```

In this section of code, the start label was named `_downloader_s_start`. The assembly language copy routine in `copyblock.h` depends on that label being named correctly.

A snippet of `copyblock.h` is:

```
/* load the value of the start of the downloader program */
                                  lrw             r7,_downloader_s_start
/* load the first address to copy to*/
                                  ld.w            r1,(r7,0)
```

Compiling the Example Program

Follow these steps to compile the example program.

1. After making sure the `downloader.s` file is in the `example\bootloader\` directory, the project needs to be compiled. This project also needs to generate an S-record for download into FLASH. (See **Notes** on compiling the bootloader.)
2. Finally, get the S-record into FLASH. If you have a FLASH downloader you can use that; otherwise, run the generated S-record through the `sikadown.pl` script and use the downloader to download the processed S-record as detailed in the description of the downloader.

Using the Example Program

After the example program is loaded into FLASH, reset the part and the following should appear in the terminal program:

```
MMC2107 FLASH Programmer Bootloader

Hit any key to start the FLASH Programmer
Otherwise wait 10 seconds to run the
main program.
...
```

This will give the user 10 seconds to hit any key and start the FLASH downloader or just wait and run the main program. The example main program is just an infinite loop that echoes back all the characters that are sent to it via the terminal program.

NOTE: *To get the program to boot out of FLASH, it was necessary to make sure that CodeWarrior was not currently in a debug session and to flip the power switch on the CMB/EVB the first time. After that, the reset switch worked.*

Including Bootloader in Another Project

To include the bootloader and FLASH programmer in another project, copy the `\bootloader` and `\sci_util` directories into the project directory, then include `bootloader.h` and call `bootloader` with the SCI port, baud rate, and clock frequency being used before the main program. The example program is meant as one possible setup, with the `main()` routine calling the bootloader and then calling the “real” main routine. Removing the bootloader from the program only requires commenting out the call to the bootloader. The bootloader adds about 4 K to the overall program.

Once a project with the bootloader and FLASH programmer is compiled into a project and written into FLASH, it can be used to download the project again and again. The project will have the bootloader and downloader embedded in it and will be redownloaded with each new version of the program.

Notes

Free software and specifications helpful for using this application note are available on the World Wide Web.

Perl

A free version of Perl for Windows is available through ActiveState Tool Corp. at the company’s Web site. ActiveState sells many Perl-related development tools, but the distribution of Perl is GNU (free to use). The company calls its distribution ActivePerl™, found in the Products sections of its Web site.

RS-232 Transceiver

Several chips are available for construction a transceiver. One key aspect is the ability to operate at a 3.3-V level for the MMC2107.

ActivePerl is a trademark of ActiveState Tool Corp.

Motorola has a surface mount part, MC145583V, which is used on the CMB/EVB. Specifications are available on the Motorola Web site, www.motorola.com.

Maxim Integrated Products, Inc. has a 20-PIN DIP part, MAX218, that might be more convenient for hand wiring a test board. Specification documentation is available at the Maxim Web site.

The transceiver circuit could be included on a custom board or simply as an interface when programming the FLASH as needed (for instance, on a separate board).

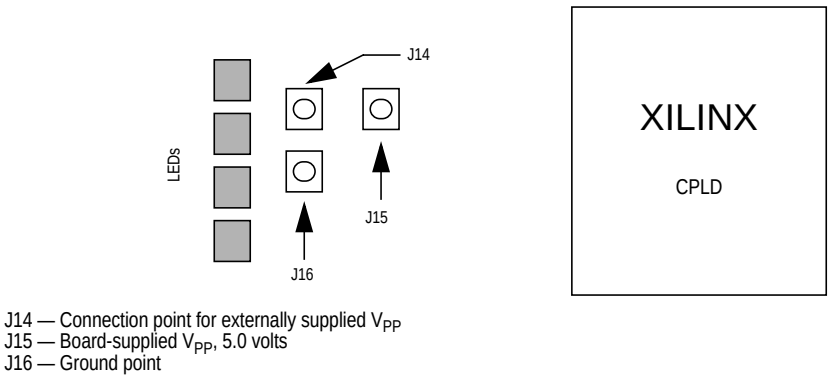
HyperTerminal

The pause between line setting is found in File->Properties->Settings->Ascii setup in HyperTerminal. A working setup for HyperTerminal is included in the `.zip` file as `sikacom1.ht` and `sikacom2.ht` for com 1 and com 2, respectively. Both are set at 19,200 baud.

Make sure the serial cable is connected to the correct SCI on the device and the PC. When using the CMB/EVB2107, port A refers to SCI1 and port B refers to SCI2.

Setting Up the EVB/CMB for an External V_{PP} Source

The EVB/CMB can be modified to accept an optional external V_{PP}.



J14 — Connection point for externally supplied V_{PP}
J15 — Board-supplied V_{PP}, 5.0 volts
J16 — Ground point

Figure 3. Detail of the EVB/CMB

1. This procedure is recommended only if you have a fair amount of confidence in your soldering ability (or the person doing the soldering).

NOTE: *Do not overheat the board, as it could possibly damage the multi-layer board.*

2. Remove the 0-Ω resistor R68 from the back of the board (behind the area shown in the **Figure 3**).
3. Solder headers into J14, J15, and optionally J16 (a connection to ground).
4. Attach the external V_{PP} to J14. Be careful not to connect to J15, especially when the red Prog LED is on. There is a 10-Ω resistor between J15 and the output of the voltage supply, so it probably wouldn't damage anything, but better to be safe than sorry. To use the board-supplied V_{PP}, simply put a jumper between J14 and J15.

Note that there is a convenient source of +5 volts on the board in the breadboard area near the serial ports. A header soldered into the 5-volts section and a jumper wire from it to J14 will provide the V_{PP} for single-chip mode.

Code Listings

Table 5 contains a summary of the code listing entries that appear in this application note.

Table 5. Code Listing Summary

File	Description
\example\sikadown.pl	.S record converter for downloader
\example\s2asm.pl	Converts .S records to assembly
\example\downloader\sikaflash\sikaflash.c	FLASH programming/erasing routines
\example\downloader\startup.s	Startup file for the downloader
\example\downloader\downloader.c	Main program for the downloader program
\example\bootloader\bootloader.c	Bootloader routines
\example\bootloader\bootloader.h	Header for bootloader.c
\example\bootloader\copyblock.h"	Assembly language copy routine
\example\sci_util\sci_util.c	SCI routines
\example\sci_util\sci_util.h	Header for sci_util.c
\example\main.c	Main file for the example application
\example\startup.s	Startup file for main application

Perl Scripts

\example\sikadown.pl

```
#!/usr/local/bin/perl
use strict;

my $versioninfo = "sikadown v0.1 9/12/2000 AM";

my $infile="";
my $outfile="";
my $aline="";
my $count=64;
my $lastaddress;
my $address;
my @bytes;
my ($length,$i,$j);
my $firsttime=1;
```

```
my $lineid;

my $addstr="";

$ARGV[0] || go_die();
$ARGV[1] || go_die();

$infile = $ARGV[0];
$outfile = $ARGV[1];
open (INFILE, $infile) || go_die("Can't open $infile fo read",1);
open (OUTFILE, ">$outfile") || go_die("Can't open $outfile for write",1);


print "$versioninfo\n";
print "infile:  $infile\n";
print "outfile: $outfile\n";
print "------------------------------------------------------------\n";


chomp ($aline=<INFILE>);
unless ($aline=~ /^S0/) { go_die("$infile is not an S-record",1); }

while(1)
{
        unless(($aline=<INFILE>) && (!($aline=~/^S7/))) {#as long as we have a line and it is
not the S7 line
                        while ($count <64){     #finish off page
                               print OUTFILE "FF";
                               $count++;
                        }
                        print OUTFILE "\n";
                        print OUTFILE "99"x68;  #address of 99999999 is end of file
                        print OUTFILE "\n";
                        go_die ("S-Record end\n",1)
        }
        chomp($aline);
        #get address off line
        #print "$aline\n";
        ($lineid,$length,$address,@bytes) = unpack"A2A2A8"."A2"x300,$aline;

        $length=hex($length);
        $length-=5;                             #subtract off 4 to account for the length of
the address, 1 for the checksum
        #print "$length\n";
        $address=hex($address);
        #print "$address\n";

        unless ($firsttime){
                if ($lastaddress==$address){
                } elsif ( ($address-$lastaddress)<(64-$count) ){
                        print (64-$count);
                        print "\n";
                        printf "%X - %X\n",$lastaddress, $address;
                        for ($i=0;$i<($address-$lastaddress);$i++){
                               print OUTFILE "FF";
                               $count++;
                        }
                } else {
                        while ($count <64){     #finish off page
                               print OUTFILE "FF";
                               $count++;
```

```
                        }
                        print OUTFILE "\n"
                }
        } else {
                $firsttime=0;
        }
        $i=0;
        while ($i<$length) {
                if ($count==64){
                        if ($address % 64) {    #make sure we're page aligned
                               $addstr = sprintf("%X", ($address - ($address %64) ));
                               $addstr="0"x(8-length($addstr)).$addstr;
                               print OUTFILE $addstr;
                               for ($j=1; $j<=$address %64; $j++) {
                                        print OUTFILE "FF";
                               }
                               $count = $address % 64;
                        } else {
                               $addstr=sprintf("%X",$address);
                               $addstr="0"x(8-length($addstr)).$addstr;
                               print OUTFILE $addstr;
#                              printf OUTFILE "%8.lx",$address;#print address
                               $count=0;
                        }
                }
                print OUTFILE "$bytes[$i]";
                $i++;
                $count++;
                $address++;
                if($count == 64) { print OUTFILE "\n"; }
        }
        $lastaddress=$address; #save off address
        #go_die("test");

}
#*************************************************************************************
sub go_die {
        my($message,$supressformat)=@_;
        if ($message) {print "$message";}
        close (INFILE);
        close (OUTFILE);
        unless ($supressformat) {
                print "format is: \n";
                print "                              sikadown <s-file> <outfile>\n";
        } else {
                print "-------------------------------------------------------\n";
        }
        die ("\n");
}
#############################################################################################
```

\example\s2asm.pl

```
#!/usr/local/bin/perl
#############################################################################################
# File: s2asm
# Purpose: Perl script that takes an .elf.s and translates it into
# an assembly language file that contains the data in the s record
#
```

```
# This is so you can compile a program to run in ram, translate it
# with this program and include it in a program that goes in flash,
# then you can copy the block of data from flash into ram and run
# the program.
#
# The format of the created assembly file is:
#       .long [starting address to write to]
#       .byte [number of bytes until the next address]
#       .byte [first byte]
#       ...
#       .byte [last byte of the line]
#       .long [second starting address]
#       .byte [number of bytes]
#   ...
#   ...
#       .long 0x99999999                        (indicates the last chunk of memory)
#
#   the label it creates to reference the data is the output filename
#       with the '.' replaced with '_' and a '_start' appended
#       ie. base_s.s => _base_s_s_start
#
#   it puts an #include "copyblock.h" which contains the code to copy
#       the block of code into ram from flash.  I named it an .h file so I
#       can put it in a CodeWarrior project and it wouldn't try to compile
#       it by itself.
##############################################################################################
use strict;

my $versioninfo = "s2asm v0.1 9/27/2000 AM";

my $infile="";
my $outfile="";
my $aline="";
my $count;
my $address;
my @bytes;
my ($length,$i,$j);
my $firsttime=1;
my $lineid;

my $label;

my $addstr="";

$ARGV[0] || go_die();
$ARGV[1] || go_die();

$infile = $ARGV[0];
$outfile = $ARGV[1];
open (INFILE, $infile) || go_die("Can't open $infile fo read",1);
open (OUTFILE, ">$outfile") || go_die("Can't open $outfile for write",1);


print "$versioninfo\n";
print "infile:  $infile\n";
print "outfile: $outfile\n";
print "-----------------------------------------------------------\n";


chomp ($aline=<INFILE>);
unless ($aline=~ /^S0/) { go_die("$infile is not an S-record",1); }
```

```
print OUTFILE "\/* File: $outfile *\/\n";
print OUTFILE "\/* Generated by $versioninfo *\/\n";

print OUTFILE "\n\n";

$label=$outfile;
$label=~s/\./_/g;                                   #replace '.' with _
$label="_".$label."_start";

print OUTFILE "#include \"copyblock.h\"\n";
print OUTFILE "\t.data\n";
print OUTFILE "\t.export\t$label\n";
print OUTFILE "$label:\n";
#print OUTFILE "\t.long\t$label\n";
#format will be
#       .long address to copy to
#       .byte num of bytes to copy
#       followed by the appropriate number of bytes
#   end will be marked by:
#       .long 0x99999999

while(1){
        unless($aline=<INFILE>){
                print OUTFILE "\t.long\t0x99999999\n";
                go_die("end of .S record reached without a S7 line encountered",1); }
        chomp($aline);
        if ($aline=~/S7/) {
                print OUTFILE "\t.long\t0x99999999\n";
                go_die("end of .S record reached",1);
        }

        ($lineid,$length,$address,@bytes) = unpack"A2A2A8"."A2"x300,$aline;
        $length=hex($length);
        $length-=5;                                 #subtract off 4 to account for the length of
the address, 1 for the checksum
        print OUTFILE "\t.long\t0x$address\n";
        printf OUTFILE "\t.byte\t0x%X\n",$length;

        for ($count=0; $count<$length; $count++) {
                print OUTFILE "\t.byte\t0x$bytes[$count]\n";
        }
}

#*********************************************************************
sub go_die {
        my($message,$supressformat)=@_;
        if ($message) {print "$message\n";}
        close (INFILE);
        close (OUTFILE);
        unless ($supressformat) {
                print "format is: \n";
                print "                            s2asm <s-file> <outfile>\n";
        } else {
                print "-------------------------------------------------------\n";
        }
        die ("\n");
}
#############################################################################################
```

Downloader Code

\example\downloader\sikaflash\sikaflash.c – FLASH Programming/Erasing Routines

```
//Desc: Flash programming for Sika
//Author: Billy Eno
//Date:   10-17-2000*/

#include "sikaflash.h"


/*****************************************************************************************
*********
 bulk_erase
 bulk erase a block or the whole array, shadow row erase with block 0
 send the clock frequency and the block to erase (0-7)
 or a number bigger than 7 to erase the whole array.
/*****************************************************************************************
********/
int bulk_erase(INT8U clockfreq, INT8U block)
{

    INT32Upulse;    /*pulse number*/
   INT32Uver_add;/*verify address*/
   INT32U  erase_start;/*start of erase*/
   INT32U  erase_end;/*end of erase */
   INT8Uclksc_num;  /*numerator and denominator for clock scalar (R)*/
   INT8Uclksc_den = 1;
   /*-------------------------------------------*/

   reg_CMFRCTL.reg = 0x00000000;/* reset CTL to reset state (just to be sure)*/
   reg_CMFRMCR.reg = 0x000000FF;/* reset CMFR_MCR to reset state (just to be sure) */

   /* get the start and end addresses */
   if (block>7) /* if we want to erase the whole array*/
   {
      erase_start = 0x00000000;
      erase_end = 0x0001FFFF;
   }
   else
   {
      erase_start = block * 0x4000;/* 16K blocks */
      erase_end = erase_start + 0x3FFF;
   }

   reg_CMFRMTR.bit.PAWS = 7; /*Set PAWS[2:0]=111 (disable firmware PWM) */

   reg_CMFRMCR.bit.PROTECT = PROTECT_NONE;/* set PROTECT to protect none */

   /*determine SCLKR[2:0], CLKPE = 00, N=15 */
   if (clockfreq < 12)
   {
      clksc_num = 1;
      reg_CMFRCTL.bit.SCLKR = 1; /*SCLKR[2:0]=b001;*/
   }
   else if (clockfreq<=18)
   {
      clksc_num = 3;
      clksc_den = 2;
```

```
    reg_CMFRCTL.bit.SCLKR = 2; /*SCLKR[2:0]=b010;*/
}
else if (clockfreq<=24)
{
    clksc_num = 2;
    reg_CMFRCTL.bit.SCLKR = 3; /*SCLKR[2:0]=b011;*/
}
else if (clockfreq<=33)
{
    clksc_num = 3;
    reg_CMFRCTL.bit.SCLKR = 4; /*SCLKR[2:0]=b100;*/
}


/*  Find clkpm (M) for a 100 mS pulse
    pulse width = System Clock Period * (clksc_num/clksc_den) * 2^N * M */
reg_CMFRCTL.bit.CLKPM = (clockfreq*1000000*clksc_den)/(10*clksc_num*32768);

reg_CMFRCTL.bit.ERASE = 1;/* set ERASE bit */

if (block>7) /* if we are erasing the whole array*/
    reg_CMFRCTL.bit.BLOCK = 0xFF;/* set up all the blocks to erase */
else
    reg_CMFRCTL.bit.BLOCK = (1<<block);/* set the bit of the block to erase */

reg_CMFRCTL.bit.SES = 1;   /* set the start erase sequence bit */
(*(volatile INT32U *)(erase_start)) = 0xffffffff; /* erase interlock write*/


pulse=1;
while (pulse <= MAX_ERASE_PULSE)
{
    if (pulse<=PULSE1) {reg_CMFRMTR.reg = 0x00000c00;} /*set NVR=1, PAWS=100*/
    else if (pulse<=PULSE2) {reg_CMFRMTR.reg = 0x00000d00;}/*set NVR=1, PAWS=101*/
    else if (pulse<=PULSE3){reg_CMFRMTR.reg = 0x00000e00;}/*set NVR=1, PAWS=110*/
    else if (pulse<=PULSE4){reg_CMFRMTR.reg = 0x00000f00;}/*set NVR=1, PAWS=111*/
    else if (pulse<=PULSE5){reg_CMFRMTR.reg = 0x00000400;}/*set NVR=0, PAWS=100*/
    else if (pulse<=PULSE6){reg_CMFRMTR.reg = 0x00000500;}/*set NVR=0, PAWS=101*/
    else if (pulse<=PULSE7){reg_CMFRMTR.reg = 0x00000600;}/*set NVR=0, PAWS=110*/
    else          {reg_CMFRMTR.reg = 0x00000700;}/*set NVR=0, PAWS=111*/

    reg_CMFRCTL.bit.EHV = 1 ; /*set EHV*/

    while (reg_CMFRCTL.bit.HVS);/*wait for HVS bit to clear*/

    reg_CMFRCTL.bit.EHV = 0; /*clear EHV*/

     /*margin verify*/
    if (pulse > PULSE_BEFORE_CHECK) /*if we are at the point we want to start margin reads*/
    {
     ver_add = erase_start;
     while (ver_add < erase_end)/* margin verify loop */
        {
       /* if the address doesn't read as all ones, break and then handle problems */
       if ( ((*(volatile INT32U *) (ver_add)) != 0xffffffff) )
           break;
       ver_add+=4;
     }
     /* see if margin read passed */
     if (ver_add >= erase_end) /* if we have gone through the whole area to erase */
       break;           /* break out of the while loop */
```

```
      }

      pulse++;/* go to next pulse */
   } /*end while (pulse <= MAX_ERASE_PULSE)  */

   reg_CMFRCTL.reg = 0x00000000;/* reset CTL to reset state */
   reg_CMFRMCR.reg = 0x000000FF;/* reset CMFR_MCR to reset state */

   if (ver_add < erase_end)/* if it didn't make it to the end of the block to erase*/
      return FALSE /* return false*/
   else
      return TRUE; /* otherwise return true */
}

/*****************************************************************************
program_page()
program a page,
clockfreq - clock frequency
address - starting address of page in flash
dataloc - starting address of page in RAM
*****************************************************************************/
int program_page(INT8U clockfreq, INT32U address, INT32U dataloc)
{
   INT8U i=0;            /* counter */
   INT32U curr_add = address;/* holds the current address to be programmed */
   INT32Udata_add = dataloc; /* holds the current address to be read from */

   INT32Upulse = 1;      /* number of pulses */

   INT8Uall_programmed = 0;/* see if programmed */
   INT8Uword_programmed = 0;/* individual word programmed */

   INT8Uclksc_num;       /*numerator and denominator for clock scalar (R)*/
   INT8Uclksc_den = 1;

   volatile INT32U junk; /*make this volatile so it doesn't get optimized
                         away somehow*/

   INT8Uclkpm_250us;
   INT8Uclkpm_100us;
   INT8Uclkpm_50us;
   /*-----------------------------------------*/

   reg_CMFRCTL.reg = 0x00000000;/* reset CTL to reset state (just to be sure)*/
   reg_CMFRMCR.reg = 0x000000FF;/* reset CMFR_MCR to reset state (just to be sure) */

   reg_CMFRMTR.bit.PAWS = 7; /*Set PAWS[2:0]=111 (disable firmware PWM) */
   reg_CMFRMTR.bit.GDB= 1;/*set GDB=1*/

   reg_CMFRMCR.bit.PROTECT = PROTECT_NONE;/* set PROTECT to protect none */

   /*determine SCLKR[2:0], CLKPE = 00, N=5 */
   if (clockfreq < 12)
   {
      clksc_num = 1;
      reg_CMFRCTL.bit.SCLKR = 1; /*SCLKR[2:0]=b001;*/
   }
   else if (clockfreq<=18)
   {
      clksc_num = 3;
      clksc_den = 2;
```

```
        reg_CMFRCTL.bit.SCLKR = 2; /*SCLKR[2:0]=b010;*/
    }
    else if (clockfreq<=24)
    {
        clksc_num = 2;
        reg_CMFRCTL.bit.SCLKR = 3; /*SCLKR[2:0]=b011;*/
    }
    else if (clockfreq<=33)
    {
        clksc_num = 3;
        reg_CMFRCTL.bit.SCLKR = 4; /*SCLKR[2:0]=b100;*/
    }


    /*  Find clkpm (M) for the 250uS, 100uS and 50uS
        pulse width = System Clock Period * (clksc_num/clksc_den) * 2^N * M
        clkpm=(clock*clksc_den)/((1/PW)*clksc_num*2^5)  <-- use (1/PW) to keep all the calcula-
tions above 0 */

    /*for 250us*/
    clkpm_250us = (clockfreq*1000000*clksc_den)/(4000*clksc_num*32);
    /*for 100us*/
    clkpm_100us = (clockfreq*1000000*clksc_den)/(10000*clksc_num*32);
    /*for 50us*/
    clkpm_50us = (clockfreq*1000000*clksc_den)/(20000*clksc_num*32);


    /* ERASE bit remains clear*/

    reg_CMFRCTL.bit.CLKPM = clkpm_250us;/*set initial CLKPM*/

    /* to determine the block we're in, mask off the lower 13 bits,
       which leaves the block number in bits [16:14], shift right the
       block number by 14. Shift left a one by the resulting block number
       to form the mask.
       (this only works with the 16K blocks like on MMC2107) */

    reg_CMFRCTL.bit.BLOCK = ( 1 << ((address & 0x0001c000) >> 14) );

    /*set SES bit */
    reg_CMFRCTL.bit.SES=1;

    /*now start programming the page*/

    /*write to buffer*/
    for (i=0;i<64;i++)/*page has 16 words, 64 bytes*/
    {
        (*(volatile INT8U *) (curr_add)) = (*(volatile INT8U *) (data_add));
        curr_add++;
        data_add++;
    }

    all_programmed = FALSE; /*initialize flag*/

    /* program */
    while (pulse <=MAX_PROG_PULSE)
    {
        /* change the voltages according to which step we are on*/

        if (pulse <= PROG_STEP8) /*don't waste time if past when stuff changes*/
        {
```

```
      if (pulse == PROG_STEP1)
        reg_CMFRMTR.reg = 0x00000c20;/* set NVR=1, PAWS=100, GDB=1 */
      else if (pulse == PROG_STEP2)
        reg_CMFRMTR.reg = 0x00000d20;/* set NVR=1, PAWS=101, GDB=1 */
      else if (pulse == PROG_STEP3)
        reg_CMFRMTR.reg = 0x00000e20;/* set NVR=1, PAWS=110, GDB=1 */
      else if (pulse == PROG_STEP4)
        reg_CMFRMTR.reg = 0x00000f20;/* set NVR=1, PAWS=111, GDB=1 */
      else if (pulse == PROG_STEP5)
        reg_CMFRMTR.reg = 0x00000420;/* set NVR=0, PAWS=100, GDB=1 */
      else if (pulse == PROG_STEP6)
        reg_CMFRMTR.reg = 0x00000520;/* set NVR=0, PAWS=101, GDB=1 */
      else if (pulse == PROG_STEP7)
        reg_CMFRMTR.reg = 0x00000620;/* set NVR=0, PAWS=110, GDB=1 */
      else if (pulse == PROG_STEP8)
        reg_CMFRMTR.reg = 0x00000720;/* set NVR=0, PAWS=111, GDB=1 */
    }

    /*do the pulse*/
    reg_CMFRCTL.bit.EHV = 1;/*set EHV*/
    while (reg_CMFRCTL.bit.HVS);   /*wait for HVS bit to clear*/

    /* change the pulse width if needed*/
    if (pulse == PROG_50US) /* if we are going to 50uS pulses*/
     reg_CMFRCTL.bit.CLKPM = clkpm_50us;
    else if (pulse == PROG_100US)/* if we are going to 100uS pulses*/
     reg_CMFRCTL.bit.CLKPM = clkpm_100us;

    reg_CMFRCTL.bit.EHV = 0;/*clear EHV*/

    /* margin verify loop */
    curr_add = address;
    all_programmed = FALSE;

    i=0;
    while(i<16)/* check the 16 words of the page */
    {
     /* check to see if programmed, if not then break out of loop*/
     if ( ((*(volatile INT32U *) (curr_add))) != 0x00000000 )
       break;
     curr_add+=4;
     i++;
    }

    if (i<8)/* if we didn't make it to the second half of the page*/
    {
     /* reading the second half of the page is critical to the margin read,
        make sure this doesn't get optimized away some how */
     junk = ((*(volatile INT32U *) (address+32))); /*do a dummy read*/
    }
    else if (i==16)/*else if we got through all 16 pages*/
    {
     all_programmed = TRUE;/* set flag to true */
     break;
    }

    pulse++;
  }/* end while (pulse <=MAX_PROG_PULSE) */


  reg_CMFRCTL.reg = 0x00000000;/* reset CMFRCTL */
```

```
    reg_CMFRMCR.reg = 0x000000FF;/* reset CMFRMCR */

    return all_programmed;
}
```

\example\downloader\sikaflash\sikaflash.h — Header File for sikaflash.c

```
/* sikaflash.h */
/* header for sikaflash.c*/
/* 10-17-2000*/

#ifndef SIKAFLASH_H
#define SIKAFLASH_H

#include "mmc2107.h"


/* FLASH Defines */
#define FLASHSTART                              0x00000000
#define FLASHEND                                0x0001FFFF/* 128K flash */
#define FLASHSIZE                               0x00020000
#define BLOCKSIZE                               0x4000/* size of a block 16K */

#define PROTECT_NONE                            0x00/*mask to protect none */

/* programming defines */
#define PULSE_BEFORE_CHECK                      7/*number of erase pulses before the first
margin read */
#define MAX_ERASE_PULSE                         28/*max erase pulses */

#define PROG_STEP1                              1   /*first pulse count when NVR=1, PAWS=100
*/
#define PROG_STEP2                              5   /*first pulse count when NVR=1, PAWS=101
*/
#define PROG_STEP3                              9   /*first pulse count when NVR=1, PAWS=110
*/
#define PROG_STEP4                              13   /*first pulse count when NVR=1, PAWS=111
*/
#define PROG_STEP5                              17   /*first pulse count when NVR=0, PAWS=100
*/
#define PROG_STEP6                              37   /*first pulse count when NVR=0, PAWS=101
*/
#define PROG_STEP7                              57   /*first pulse count when NVR=0, PAWS=110
*/
#define PROG_STEP8                              77   /*first pulse count when NVR=0, PAWS=111
for 50us pulses*/
#define PROG_STEP9                              97   /*first pulse count when NVR=0, PAWS=111
for 100us pulses*/

#define PROG_50US                               16/*pulse count before switching to 50US*/
#define PROG_100US                              96/*pulse count before switching to 100US*/

#define MAX_PROG_PULSE                          1000   /*max program pulses */


#define MAX_NVR1_PAWS100_1                     4   /*max pulse count when NVR=1, PAWS=100 */
#define MAX_NVR1_PAWS101_2                     8   /*max pulse count when NVR=1, PAWS=101 */
#define MAX_NVR1_PAWS110_3                     12   /*max pulse count when NVR=1, PAWS=110 */
#define MAX_NVR1_PAWS111_4                     16   /*max pulse count when NVR=1, PAWS=111 */
```

```
#define MAX_NVR0_PAWS100_5                      36   /*max pulse count when NVR=0, PAWS=100 */
#define MAX_NVR0_PAWS101_6                      56   /*max pulse count when NVR=0, PAWS=101 */
#define MAX_NVR0_PAWS110_7                      76   /*max pulse count when NVR=0, PAWS=110 */
#define MAX_NVR0_PAWS110_8_50US                 96    /*max pulse count when NVR=0, PAWS=111
for 50us pulses*/
#define MAX_PROG_PULSE                           1000   /*max program pulses */
                                                 /* (after the 96th pulse use NVR=0, PAWS=111
for 100us pulses*/

/*erase defines*/
#define PULSE1                                   0x00000001/*used for pulse counting*/
#define PULSE2                                   0x00000002
#define PULSE3                                   0x00000003
#define PULSE4                                   0x00000004
#define PULSE5                                   0x00000005
#define PULSE6                                   0x00000006
#define PULSE7                                   0x00000007

/* used with the bulk_erase() routine */
#define BLOCK0                                   0
#define BLOCK1                                   1
#define BLOCK2                                   2
#define BLOCK3                                   3
#define BLOCK4                                   4
#define BLOCK5                                   5
#define BLOCK6                                   6
#define BLOCK7                                   7
#define ALLBLOCKS                                8

/*prototypes*/
int bulk_erase(INT8U clockfreq, INT8U block);
int program_page(INT8U clockfreq, INT32U address, INT32U dataloc);
#endif /*#ifndef SIKAFLASH_H*/
```

example\downloader\startup.s — Startup File for the Downloader

```
//*****************************************************************************
/
// startup.s
// -- MCore Startup code.
//  > Initialize Stack Pointer register (ie. R0);
//  > Init BSS section to 0;
//  > Call main()
//*****************************************************************************//

        // Following are defined by MW Linker.
        .extern __sbss
        .extern __ebss
        .extern __stack_begin
        .extern __stack_end
        //.extern __call_static_initializers

        .extern main

 .text
 .export   _start
 .export   _program_end
```

```
//*****************************************************************************//
// exception and interupt vectors
//
//*****************************************************************************//
        .align 2

        //set up the vbr
        //
                        .long                   _start//0 - reset vector
                        //exceptions
                /*      .long                   0x0000//1 - Misaligned Access
                        .long                   0x0000//2 - Access error
                        .long                   0x0000//3 - Divide by zero
                        .long                   0x0000//4 - Illegal instruction
                        .long                   0x0000//5 - Privilage violation
                        .long                   0x0000//6 - Trace exception
                        .long                   0x0000//7 - Breakpoint exception
                        .long                   0x0000//8 - Unrecoverable Error
                        .long                   0x0000//9 - Soft Reset
                        .long                   0x0000//10 - /INT autovector
                        .long                   0x0000//11 - /FINT autovector
                        .long                   0x0000//12 - Hardware accelerator
                        .space                  0x0C//(13-15) - reserved vectors
                        .long                   0x0000// 16 - TRAP #0 instruction vector
                        .long                   0x0000// 17 - TRAP #1 instruction vector
                        .long                   0x0000// 18 - TRAP #2 instruction vector
                        .long                   0x0000// 19 - TRAP #3 instruction vector
                        .space                  0x2C//(20-31) - reserved vectors
                        //normal interupts
                        .long                   0x0000// 32 (priority 0) lowest
                        .long                   0x0000// 33 (priority 1)
                        .long                   0x0000// 34 (priority 2)
                        .long                   0x0000// 35 (priority 3)
                        .long                   0x0000// 36 (priority 4)
                        .long                   0x0000// 37 (priority 5)
                        .long                   0x0000// 38 (priority 6)
                        .long                   0x0000// 39 (priority 7)
                        .long                   0x0000// 40 (priority 8)
                        .long                   0x0000// 41 (priority 9)
                        .long                   0x0000// 42 (priority 10)
                        .long                   0x0000// 43 (priority 11)
                        .long                   0x0000// 44 (priority 12)
                        .long                   0x0000// 45 (priority 13)
                        .long                   0x0000// 46 (priority 14)
                        .long                   0x0000// 47 (priority 15)
                        .long                   0x0000// 48 (priority 16)
                        .long                   0x0000// 49 (priority 17)
                        .long                   0x0000// 50 (priority 18)
                        .long                   0x0000// 51 (priority 19)
                        .long                   0x0000// 52 (priority 20)
                        .long                   0x0000// 53 (priority 21)
                        .long                   0x0000// 54 (priority 22)
                        .long                   0x0000// 55 (priority 23)
                        .long                   0x0000// 56 (priority 24)
                        .long                   0x0000// 57 (priority 25)
                        .long                   0x0000// 58 (priority 26)
                        .long                   0x0000// 59 (priority 27)
                        .long                   0x0000// 60 (priority 28)
                        .long                   0x0000// 61 (priority 29)
                        .long                   0x0000// 62 (priority 30)
                        .long                   0x0000// 63 (priority 31) highest
```

```
                        //fast interupts
                        .long                   0x0000// 64 (priority 0) lowest
                        .long                   0x0000// 65 (priority 1)
                        .long                   0x0000// 66 (priority 2)
                        .long                   0x0000// 67 (priority 3)
                        .long                   0x0000// 68 (priority 4)
                        .long                   0x0000// 69 (priority 5)
                        .long                   0x0000// 70 (priority 6)
                        .long                   0x0000// 71 (priority 7)
                        .long                   0x0000// 72 (priority 8)
                        .long                   0x0000// 73 (priority 9)
                        .long                   0x0000// 74 (priority 10)
                        .long                   0x0000// 75 (priority 11)
                        .long                   0x0000// 76 (priority 12)
                        .long                   0x0000// 77 (priority 13)
                        .long                   0x0000// 78 (priority 14)
                        .long                   0x0000// 79 (priority 15)
                        .long                   0x0000// 80 (priority 16)
                        .long                   0x0000// 81 (priority 17)
                        .long                   0x0000// 82 (priority 18)
                        .long                   0x0000// 83 (priority 19)
                        .long                   0x0000// 84 (priority 20)
                        .long                   0x0000// 85 (priority 21)
                        .long                   0x0000// 86 (priority 22)
                        .long                   0x0000// 87 (priority 23)
                        .long                   0x0000// 88 (priority 24)
                        .long                   0x0000// 89 (priority 25)
                        .long                   0x0000// 90 (priority 26)
                        .long                   0x0000// 91 (priority 27)
                        .long                   0x0000// 92 (priority 28)
                        .long                   0x0000// 93 (priority 29)
                        .long                   0x0000// 94 (priority 30)
                        .long                   0x0000// 95 (priority 31) highest*/

//*****************************************************************************//
// _start -- program entry point.
//*****************************************************************************//

        //      .function "_start", _start, _end_start - _start
_start:
    // Just to make "__stack_end" appear in final ELF.
    lrw                         r7, __stack_end
        // init stack pointer.
    lrw                         r8, __stack_begin
    mov                         r0,r8

        // Initialize BSS section to 0.
        lrw     r5, __sbss
        lrw     r6, __ebss
        cmphs   r5,r6
        bt      _call_main
        subi    r6,1
        xor     r4,r4
_zero_bss:
        st.b    r4,(r6,0)
        cmplt   r5,r6
        loopt   r6, _zero_bss
//_call_static_init:
//      jbsr                    __call_static_initializers
_call_main:
    jbsr    main            // pull in main()
```

```
        bkpt

_program_end:
    bkpt
    bkpt

//_exit:
//    bkpt
//    bkpt

_end_start:

        .literals
```

\example\downloader\downloader.c — Main Program for the Downloader Program

```
/*****************************************************************************/
/* File: downloader.c
/* Author: Billy Eno, Product Engineer, Motorola Microcontroller Division
/* A flash downloader.
/* This allows the flash on MMC2107 (Sika) to be programmed through the serial
/* port.  A S-record that has been massaged with the Perl script "sikadown.pl"
/* can be sent via a terminal program (like Hyperterminal) as a text file and
/* the program will interpret the file and program the flash
/*
/* Is designed to be either run stand alone, with the STANDALONE #define uncommented
/* or as part of the bootloader program.  If the program is run as part of the
/* bootloader program, it retrieves the SCI port to use, the clock frequency and
/* the baudrate to use from flash at 0x180,0x181 and 0x182 respectively.
/* These locations are the first four bytes after a full vector table that starts
/* at 0x0000.
/*****************************************************************************/
#include "sci_util.h"
#include "sikaflash.h"
#include "mmc2107.h"

/* uncomment STANDALONE to run the flash programmer by itself */
#define STANDALONE
#define SA_PORT                                 1
#define                 SA_BAUD                 19200
#define                 SA_FREQ                 32
/******************/

/* comment out USING_EVB to disable the write to enable the Vpp on the EVB */
#define USING_EVB;
/******************/

void command_loop(INT8U port, INT8U clockfreq);
void program_flash(INT8U port, INT8U clockfreq);
void get_a_page(INT8U port, INT32U * addr, char save_addr[], INT8U data[]);
void verify_flash(INT8U port);

/*****************************************************************************/
/* command_loop
/* Command loop for the flash programmer.
/*****************************************************************************/
void command_loop(INT8U port, INT8U clockfreq)
{
        char inny;
```

```
	while(1)
	{
		send_string(port, "\n\r: ");
		inny = get_a_byte(port);
		send_byte(port, inny);

		/* instrunctions */
		if (inny=='?')
		{
			send_string(port, "\n\rTo download to flash:");
			send_string(port, "\n\r  1. bulk erase if necessary");
			send_string(port, "\n\r  2. Make sure the delay time between lines");
			send_string(port, "\n\r     is set to at least 75 ms");
			send_string(port, "\n\r  3. Type f");
			send_string(port, "\n\r  4. Send S-record (processed with perl");
			send_string(port, "\n\r     script \"sikadown.pl\") as a text file");
			send_string(port, "\n\r");
			send_string(port, "\n\rf - Program flash");
			send_string(port, "\n\rv - Verify flash");
			send_string(port, "\n\rb - bulk erase all of the flash");
		}
		/* send a file */
		else if (inny=='f')
		{
			send_string(port, "\n\rProgram flash.");
			send_string(port, "\n\rSend file now.\n\r");
			program_flash(port,clockfreq);
		}
		/* bulk erase flash */
		else if (inny=='b')
		{
			send_string(port, "\n\rErasing flash.");
			if (bulk_erase(clockfreq,ALLBLOCKS))
				send_string(port, "\n\rDone.");
			else
				send_string(port, "\n\rError Erasing flash.");
		}
		/* verify program */
		else if (inny=='v')
		{
			send_string(port, "\n\rVerify flash.");
			send_string(port, "\n\rSend file now.\n\r");
			verify_flash(port);
		}
		/* otherwise tell the user how to get instructione */
		else
		{
			send_string(port, "\n\r type ? for help");
		}

	}
}
/*****************************************************************************/
/* program_flash
/* program the flash with a massaged S-record
/* the text file must be sent with sufficient pauses between the lines to
/* allow the flash to program
/*****************************************************************************/
void program_flash(INT8U port, INT8U clockfreq)
{
	INT32U		addr=0;			/*address to program*/
```

```
        INT8U           data[64];               /*data to program*/

        INT32U          dataloc;                /* location of the data */
        char            save_addr[9];           /*save the address string*/
        while (addr!=0x99999999)
        {
        get_a_page(port,&addr,save_addr,data);
        dataloc = (INT32U)(&data);              /* get the location in RAM of the data */
        if (addr!=0x99999999)

                if (!program_page(clockfreq,addr,dataloc))
                {
                        send_string(port, "\n\rError page address:");
                        send_string(port, save_addr);
                }

        send_string(port, ".");
        }
        send_string(port, "\n\rDone.");
}
/*****************************************************************************/
/* verify_page
/* verify a page of flash
/*****************************************************************************/
INT8U verify_page(INT32U addr, INT8U data[])
{
        INT8U i;

        /* page is 64 bytes long */
        for (i=0; i<64; i++)
        {
                if ( (*(volatile INT8U*) (addr)) != data[i] )
                        return FALSE;
                addr++;
        }
        return TRUE;
}
/*****************************************************************************/
/* verify_flash
/* verify that a downloaded program is correct in flash
/*****************************************************************************/
void verify_flash(INT8U port)
{
        INT32U          addr=0;                 /*address to program*/
        INT8U           data[64];               /*data to program*/

        char            save_addr[9];           /*save the address string*/
        while (addr!=0x99999999)
        {
        get_a_page(port,&addr,save_addr,data);

        if (addr!=0x99999999)
                if (!verify_page(addr,data))
                {
                        send_string(port, "\n\rError page address:");
                        send_string(port, save_addr);
                }

        send_string(port, ".");
        }
        send_string(port, "\n\rDone.");
```

```
}

/*****************************************************************************/
/* trans_hex
/* translates an (ascii) hex digit into a numerical value, must be capital
/*****************************************************************************/
char trans_hex(char adigit)
{
        adigit -= '0';                          /* subtract off '0' to take care of values for
0->9 */
        if (adigit > 9)                         /* if the value is over 9 subtract off 7 for
A->F                                            */
                adigit -= 7;
        return adigit;
}
/*****************************************************************************/
/*****************************************************************************/
void get_a_page(INT8U port, INT32U * addr, char save_addr[], INT8U data[])
{
        INT32U          temp=0;

        char            aline[136];
        char            addrstr[8];
        char            indata[128];
        INT8U           i;

        get_bytes(port, 8,addrstr);
        get_bytes(port, 128,indata);

        get_a_byte(port);                       /*get the end of line character*/

        *addr=0;
        /*translate the address to numeric and save off the character representation*/
        for (i=0;i<8;i++)
        {
                save_addr[i]=addrstr[i];        /* save off the address string*/
                *addr += trans_hex(addrstr[i]); /* translate hex digit and add it in*/
                if (i<7)
                        *addr<<=4;              /*shift it up if i<7*/
        }
        save_addr[i]=0; /*add a end of string*/

        /*translate all the digits*/
        for (i=0; i<128; i++)
                indata[i] = trans_hex(indata[i]);

        /*calculate all the bytes*/
        for (i=0; i<128; i=i+2)
                data[i/2] = (indata[i]<<4)+(indata[i+1]);

        return;
}
/*****************************************************************************/
/*****************************************************************************/
int main(void)

{
        INT8U           port;
        INT8U           clockfreq;
        INT16U          baud;
```

```
        #ifdef STANDALONE
        /* if stand alone use the stand alone settings */
        port = SA_PORT;
        baud = SA_BAUD;
        clockfreq = SA_FREQ;
        #else

        /* otherwise get the sci setup from the flash*/
        port = (*(volatile INT8U*) (0x180));
        clockfreq = (*(volatile INT8U*) (0x181));
        baud = (*(volatile INT16U*) (0x182));
        #endif

        /* if USING EVB set up Chip selects so Vpp can be enabled*/
        #ifdef USING_EVB

        /*Chip Select Control - CSR0 */
        reg_CSCR0.reg = 0x3403;

        /*Chip Select Control - CSR1*/
        reg_CSCR1.reg = 0x3103;

        /*Chip Select Control - CSR2*/
        reg_CSCR2.reg = 0x3403;

        /*Chip Select Control - CSR3*/
        reg_CSCR3.reg =  0x3403;

        /*turn on Vpp*/
        (*(volatile INT8U*) (0x807ffffd)) = 0x10;

        /*setup PLL, this assumes an 8MHz input clock and produces a 32MHz clock*/
        reg_SYNCR.reg = 0x2000;

        #else
        /*this is put in here so that an error will be generated
                if the EVB is not used and a user-defined set PLL (clock)
                routine is not defined*/
        user_set_PLL();
        #endif

        /* disable watchdog timer */
        reg_WCR.bit.EN = 0;

        setup_sci(port, baud, clockfreq);
        send_string(port, "\n\rMMC2107 (Sika) Flash Programmer");
        command_loop(port, clockfreq);

        return;
}
```

Bootloader Code

\example\bootloader\bootloader.c — Bootloader Routines

```
/*****************************************************************************/
/* File: bootloader.c
/* Author: Billy Eno, Product Engineer, Motorola Microcontroller Division
/* A bootloader for MMC2107.
/* Waits for input from the specified SCI port and if recieved loads into ram and
/* executes another program, in this case a flash programmer
/*****************************************************************************/
#include "mmc2107.h"
#include "bootloader.h"

/*****************************************************************************/
/* private prototypes
/*****************************************************************************/
void start_timer(INT8U clockfreq, INT8U seconds);
BOOLEAN timeup();

/*****************************************************************************/
/* bootloader
/* Waits 10 seconds for an input from the SCI
/* if a key is pressed it calls _copyblock() to run the flash programmer
/* otherwise returns to run the "main" program
/*****************************************************************************/
void bootloader(INT8U port, INT16U baud, INT8U clockfreq)
{
        INT8U i;                /*counter*/
        char junk;

        setup_sci(port, baud, clockfreq);
        send_string(port, "\n\rMMC2107 Flash Programmer Bootloader\n\r");
        send_string(port, "\n\rHit any key to start the Flash Programmer");
        send_string(port, "\n\rOtherwise wait 10 seconds to run the");
        send_string(port, "\n\rmain program.\n\r");

        for (i=0;i<10;i++)
        {
                start_timer(clockfreq, 1);
                while (!timeup());
                {
                        junk = check_for_byte(port);
                }
                if (junk != 0)                  /*if any key was hit, break, run programmer*/
                        break;
                send_byte(port, '.');
        }
        if (junk==0)
        {
                return;                         /* just return and go to the main program */
        }
        else
        {
                _copyblock();                   /* copy flash programmer into ram, execute */
        }
        return;
}
/*****************************************************************************/
```

```
/*      start_timer
/*  start PIT1 for the number of seconds
/*      maximum of 60 seconds @ 32MHz
/*      maximum of 120 seconds @ 16MHz
/*****************************************************************************/

void start_timer(INT8U clockfreq, INT8U seconds)
{
        INT16U counter = seconds * ((clockfreq*1000000)/32768);/*get the counter value*/

        /*set up with prescalar 32,768, Counter Overwrite,  */
        /*Clear PIT flag, Counter Reload, and enable*/
        reg_PCSR1.reg = 0x0F17;
        reg_PMR1 = counter; /*put counter into Modulus Register */

        return;
}
/*****************************************************************************/
/* timeup
/* see if the time is up, if so disable timer, return true
/*****************************************************************************/
BOOLEAN timeup()
{
        if (reg_PCSR1.bit.PIF)                  /* if PIT flag set, */
        {
                reg_PCSR1.reg = 0x0004;         /*Disable PIT */
                return TRUE;                    /*return true */
        }
        else
        {
                return FALSE;
        }
}
```

\example\bootloader\bootloader.h — Header for bootloader.c

```
#ifndef BOOTLOADER_H
#define BOOTLOADER_H

#include "sci_util.h"

/*****************************************************************************/
/* prototypes
/*****************************************************************************/

void bootloader(INT8U port, INT16U baud, INT8U clockfreq);

#endif
```

\example\bootloader\copyblock.h — Assembly Language Copy Routine

```
/*****************************************************************************/
/* File: copyblock.h
/* Author: Billy Eno, Product Engineer, Motorola Microcontroller Division
/* An assembly language routine that copies a block of data from flash into ram
/* and jumps to the location held in the first word of the data.
/*
```

```
/* The data to load needs to be created from an S record by the perl script
/* 's2asm.pl' and has a statement that includes this file.
/* to use with other programs, the label '_down_s_s_start' has to be changed to
/* match the start label in the created assembly file
/*****************************************************************************/

        .text
        .export _copyblock
                /* r7 - pointer to data in flash (source) */
                /* r1 - pointer to destination in RAM (destination) */
                /* r2 - holds remaining number of bytes to copy */
                /* r3 - intermidiate storage for data */
                /* r6 - holds alignment data, the number of bytes to the next word boundary */
                /* r8 - saves the starting location of the ram code */
                /* r9 - holds the compare value for the last address (99999999)*/

                /* create the compare value by adding in 9 and shifting */
_copyblock:
                movi            r9,9
                lsli            r9,4
                addi            r9,9
                lsli            r9,4
                addi            r9,9
                lsli            r9,4
                addi            r9,9
                lsli            r9,4
                addi            r9,9
                lsli            r9,4
                addi            r9,9
                lsli            r9,4
                addi            r9,9

                /* load the value of the start of the downloader program */
                lrw             r7,_downloader_s_start
                /* load the first address to copy to*/
                ld.w            r1,(r7,0)
                /*  this will also be the location of the jmp vector, so save a copy of it */
                ld.w            r8,(r7,0)

                /* initialize r6 so it indicates there are four bytes to next word boundary*/
                movi            r6,4

next_address:
                /* increment r7 by 4 to get to the number of bytes starting at this loc*/
                addi            r7,4
                /* load r2 with number of bytes, will be used as counter */
                ld.b            r2,(r7,0)
                /* increment r7 to get to first byte of data, decrement r6*/
                addi            r7,1
                subi            r6,1

next_byte:
                /* load data into r3 from flash, store r3 into ram */
                ld.b            r3,(r7,0)
                st.b            r3,(r1,0)

                /* add 1 to r1 and r7 to get to the next set of locations */
                addi            r7,1
                addi            r1,1
```

```
                /*decrement r6, compare for zero, reinit to 4 if true */
                subi            r6,1
                cmpnei          r6,0
                bt              skip_reinit
                movi            r6,4
skip_reinit:
                /*subtract off 1 from r2, compare for not equal zero, branch if true */
                subi            r2,1
                cmpnei          r2,0
                bt              next_byte

                /*see if we are on a word boundary (r6 = 4) if so go ahead and load the word*/
                /*otherwise, add r6 to r7 to get to next word boundary, and then reinit r6 to 4*/
                cmpnei          r6,4
                bf              load_next_add
                addu            r7,r6
                movi            r6,4
load_next_add:
                ld.w            r1,(r7,0)
                cmpne           r1,r9
                bt              next_address

                /* now just jump to the location stored at first word of the transfered data */

                ld.w            r1,(r8,0)
                jmp             r1
        .literals
```

\example\sci_util\sci_util.c — SCI Routines

```
/*****************************************************************************/
/* File: sci_util.c
/* Author: Billy Eno, Product Engineer, Motorola Microcontroller Division
/*
/* A set of sci_utilities that allow for text communication between the SCI
/* and a terminal program (or any another SCI)
/* Uses 8/N/1 format.
/* Can be used with either port,any clock frequency and any baud rate
/*****************************************************************************/
#include "sci_util.h"

/*****************************************************************************/
/* check_for_byte
/* get a single byte if one is available
/*
/* arguments
/* --------------------
/* port:                        SCI1 or SCI2
/*****************************************************************************/
char check_for_byte(INT8U port)
{
        if (port == SCI1)
        {
                if (reg_SCI1SR1.bit.RDRF)         /*if the RDRF is set*/
                        return reg_SCI1DRL;       /*return the data*/
                else return 0;                    /*otherwise return 0*/
        }
        else
        {
                if (reg_SCI2SR1.bit.RDRF)         /*if the RDRF is set*/
                        return reg_SCI2DRL;       /*return the data*/
                else return 0;                    /*otherwise return 0*/
```

```
        }
}
/*****************************************************************************/
/* get_a_byte
/* wait for and get a single byte
/*
/* arguments
/* ---------------------
/* port:                        SCI1 or SCI2
/*****************************************************************************/
char get_a_byte(INT8U port)
{
        if (port == SCI1)
        {
                while (!reg_SCI1SR1.bit.RDRF);    /*wait for RDRF*/
                return reg_SCI1DRL;               /*return the data*/
        }
        else
        {
                while (!reg_SCI2SR1.bit.RDRF);    /*wait for RDRF*/
                return reg_SCI2DRL;               /*return the data*/
        }
}

/*****************************************************************************/
/* get_byte
/* wait for and get a specific number of bytes
/*
/* arguments
/* ---------------------
/* port:                        SCI1 or SCI2
/* strlength:                   number of bytes to get
/* astring:                                       an array of length strlength
/*****************************************************************************/
void get_bytes(INT8U port, INT16U strlength, char astring[])
{
        int i;
        for(i=0;i<strlength;i++)
                astring[i]=get_a_byte(port);
        return;
}

/*****************************************************************************/
/* send_byte
/* send a single byte and wait for TDRE
/*
/* arguments
/* ---------------------
/* port:                        SCI1 or SCI2
/*****************************************************************************/
void send_byte(INT8U port, char achar)
{
        if (port == SCI1)
        {
                reg_SCI1DRL = achar;
                while(!reg_SCI1SR1.bit.TDRE);     /*wait for TDRE*/
        }
        else
        {
                reg_SCI2DRL = achar;
                while(!reg_SCI2SR1.bit.TDRE);     /*wait for TDRE*/
        }

        return;
}

/*****************************************************************************/
/* send_string
```

```
/* send a null terminated single TC
/*
/* arguments
/* ---------------------
/* port:                        SCI1 or SCI2
/* astring:                                    a null terminated string
/*****************************************************************************/
void send_string(INT8U port, char astring[])
{
        INT16U i=0;
        while (astring[i] != 0x00)
        {
                send_byte(port,astring[i]);
                i++;
        }
        while(!reg_SCI1SR1.bit.TC); /*wait for TC to set (transmission complete)*/
        return;
}

/*****************************************************************************/
/* setup_sci
/* initialize sci to 8 bit mode and the baud rate, then enable
/*
/* arguments
/* -------------
/* port:                        SCI1 or SCI2
/* baud:                        baud rate in bps
/* clockfreq:                                  clock frequency in MHz
/*****************************************************************************/
void setup_sci(INT8U port, INT16U baud, INT8U clockfreq)
{
        INT8U junk;
        INT16U divisor;

        /* calculate divsor based on clock freq (in MHz) and baud rate*/
        divisor = (clockfreq * 1000000)/(16*baud);

        if (port == SCI1)
        {
                reg_SCI1BD = divisor;
                reg_SCI1CR1.reg = 0x00;          /*reset values (8/N/1)*/
                reg_SCI1CR2.bit.TE = 1;          /*enable transmitter*/
                reg_SCI1CR2.bit.RE = 1;          /*enable reciever*/
                junk = reg_SCI1SR1.reg;          /*transmitter is actually enabled here*/
                /*wait for TDRE to go high (wait for preamble)*/
                while(!reg_SCI1SR1.bit.TDRE);
        }
        else
        {
                reg_SCI2BD = divisor;
                reg_SCI2CR1.reg = 0x00;          /*reset values (8/N/1)*/
                reg_SCI2CR2.bit.TE = 1;          /*enable transmitter*/
                reg_SCI2CR2.bit.RE = 1;          /*enable reciever*/
                junk = reg_SCI2SR1.reg;          /*transmitter is actually enabled here*/
                /*wait for TDRE to go high (wait for preamble)*/
                while(!reg_SCI2SR1.bit.TDRE);
        }

        return;
}
/*****************************************************************************/
```

\example\sci_util\sci_util.h — Header for sci_util.c

```
#ifndef SCI_UTIL_H
#define SCI_UTIL_H

#include "mmc2107.h"

#define SCI1                                    1
#define SCI2                                    2

#define                 BAUD2400                2400
#define BAUD4800                                4800
#define BAUD9600                                9600
#define                 MHZ16                   16
#define                 MHZ32                   32

/* prototypes */

char check_for_byte(INT8U port);
char get_a_byte(INT8U port);
void get_bytes(INT8U port, INT16U strlength, char astring[]);
void send_byte(INT8U port, char achar);
void send_string(INT8U port, char astring[]);
void setup_sci(INT8U port, INT16U baud, INT8U clockfreq);

#endif /* end of #ifndef SCI_UTIL_H */
```

Example Application

\example\main.c— Main File for the Example Application

```
/*****************************************************************************/
/* Project: example.mcp
/* File: main.c
/* Author: Billy Eno, Product Engineer, Motorola Microcontroller Division
/*
/* Demonstration of a flash programmer / bootloader.
/*
/* This project calls a bootloader that allows the user to
/* choose between the "main program" (just returns) or
/* a flash programmer that is copied into RAM and executed.
/*
/* The flash programmer allows the user to download an S record
/* that has been massaged with a Perl script.
/*
/* The flash programmer is a seperate project that has been compiled
/* to run in RAM and then converted to an assembly data file via
/* a Perl script.
/*
/* Both the bootloader and the flash programmer also demonstrate
/* the use of the SCI port for text communication with a terminal
/* program.
/*****************************************************************************/

#include "mmc2107.h"
#include "bootloader.h"
```

```
/* sci_setup.h contains the port, baudrate and clockfreq used */
/* this is seperate so startup.s can put the values into the beginning */
/* of the program so the flash programmer can read the values and */
/* continue communication with the terminal program */
#include "sci_setup.h"

void mainprogram(void);

/*****************************************************************************/
/* main
/* starting point
/* calls the bootloader and if the bootloader returns, calls mainprogram()
/*****************************************************************************/
int main(void)
{
        /* disable watchdog timer */
        reg_WCR.bit.EN = 0;

        bootloader(SCIPORT, BAUDRATE, CLOCKFREQ);

        mainprogram();
}
/*****************************************************************************/
/* mainprogram
/* this represents a "real" application, all it does is echo back characters
/* to the SCI
/*****************************************************************************/
void mainprogram(void)
{
        char junk;

        setup_sci(SCIPORT, BAUDRATE, CLOCKFREQ);
        send_string(SCIPORT, "\n\rNow running the main program...");
        send_string(SCIPORT, "\n\r(all this does is echo back characters)\n\r");

        while(1)
        {
                junk=get_a_byte(SCIPORT);
                send_byte(SCIPORT,junk);
        }

        return;
}
```

\example\startup.s — Startup File for Main Application

```
//*****************************************************************************/
// startup.s
// -- MCore Startup code.
//  > Initialize Stack Pointer register (ie. R0);
//  > Init BSS section to 0;
//  > Call main()
//*****************************************************************************//
        #include "sci_setup.h"

        // Following are defined by MW Linker.
        .extern __sbss
        .extern __ebss
        .extern __stack_begin
```

```
        .extern __stack_end
        //.extern __call_static_initializers

        .extern main

 .text
 .export   _start
 .export   _program_end


//*****************************************************************************//
// exception and interupt vectors
//
//*****************************************************************************//
        .align 2

        //set up the exception vector table
        //
                        .long                   _start//0 - reset vector
                        //exceptions
/*                      .long                   0x0000//1 - Misaligned Access
                        .long                   0x0000//2 - Access error
                        .long                   0x0000//3 - Divide by zero
                        .long                   0x0000//4 - Illegal instruction
                        .long                   0x0000//5 - Privilage violation
                        .long                   0x0000//6 - Trace exception
                        .long                   0x0000//7 - Breakpoint exception
                        .long                   0x0000//8 - Unrecoverable Error
                        .long                   0x0000//9 - Soft Reset
                        .long                   0x0000//10 - /INT autovector
                        .long                   0x0000//11 - /FINT autovector
                        .long                   0x0000//12 - Hardware accelerator
                        .space                  0x0C//(13-15) - reserved vectors
                        .long                   0x0000// 16 - TRAP #0 instruction vector
                        .long                   0x0000// 17 - TRAP #1 instruction vector
                        .long                   0x0000// 18 - TRAP #2 instruction vector
                        .long                   0x0000// 19 - TRAP #3 instruction vector
                        .space                  0x2C//(20-31) - reserved vectors
                        //normal interupts
                        .long                   0x0000// 32 (priority 0) lowest
                        .long                   0x0000// 33 (priority 1)
                        .long                   0x0000// 34 (priority 2)
                        .long                   0x0000// 35 (priority 3)
                        .long                   0x0000// 36 (priority 4)
                        .long                   0x0000// 37 (priority 5)
                        .long                   0x0000// 38 (priority 6)
                        .long                   0x0000// 39 (priority 7)
                        .long                   0x0000// 40 (priority 8)
                        .long                   0x0000// 41 (priority 9)
                        .long                   0x0000// 42 (priority 10)
                        .long                   0x0000// 43 (priority 11)
                        .long                   0x0000// 44 (priority 12)
                        .long                   0x0000// 45 (priority 13)
                        .long                   0x0000// 46 (priority 14)
                        .long                   0x0000// 47 (priority 15)
                        .long                   0x0000// 48 (priority 16)
                        .long                   0x0000// 49 (priority 17)
                        .long                   0x0000// 50 (priority 18)
                        .long                   0x0000// 51 (priority 19)
                        .long                   0x0000// 52 (priority 20)
                        .long                   0x0000// 53 (priority 21)
```

```
                        .long                   0x0000// 54 (priority 22)
                        .long                   0x0000// 55 (priority 23)
                        .long                   0x0000// 56 (priority 24)
                        .long                   0x0000// 57 (priority 25)
                        .long                   0x0000// 58 (priority 26)
                        .long                   0x0000// 59 (priority 27)
                        .long                   0x0000// 60 (priority 28)
                        .long                   0x0000// 61 (priority 29)
                        .long                   0x0000// 62 (priority 30)
                        .long                   0x0000// 63 (priority 31) highest
                        //fast interupts
                        .long                   0x0000// 64 (priority 0) lowest
                        .long                   0x0000// 65 (priority 1)
                        .long                   0x0000// 66 (priority 2)
                        .long                   0x0000// 67 (priority 3)
                        .long                   0x0000// 68 (priority 4)
                        .long                   0x0000// 69 (priority 5)
                        .long                   0x0000// 70 (priority 6)
                        .long                   0x0000// 71 (priority 7)
                        .long                   0x0000// 72 (priority 8)
                        .long                   0x0000// 73 (priority 9)
                        .long                   0x0000// 74 (priority 10)
                        .long                   0x0000// 75 (priority 11)
                        .long                   0x0000// 76 (priority 12)
                        .long                   0x0000// 77 (priority 13)
                        .long                   0x0000// 78 (priority 14)
                        .long                   0x0000// 79 (priority 15)
                        .long                   0x0000// 80 (priority 16)
                        .long                   0x0000// 81 (priority 17)
                        .long                   0x0000// 82 (priority 18)
                        .long                   0x0000// 83 (priority 19)
                        .long                   0x0000// 84 (priority 20)
                        .long                   0x0000// 85 (priority 21)
                        .long                   0x0000// 86 (priority 22)
                        .long                   0x0000// 87 (priority 23)
                        .long                   0x0000// 88 (priority 24)
                        .long                   0x0000// 89 (priority 25)
                        .long                   0x0000// 90 (priority 26)
                        .long                   0x0000// 91 (priority 27)
                        .long                   0x0000// 92 (priority 28)
                        .long                   0x0000// 93 (priority 29)
                        .long                   0x0000// 94 (priority 30)
                        .long                   0x0000// 95 (priority 31) highest

/* org to 180 so the downloader knows wher to find the sci info */
/* this also leaves room for the vbr*/
        .org 0x180
_sci_port:
        .byte           SCIPORT

_clock_freq:
        .byte           CLOCKFREQ

_baud_rate:
        .short          BAUDRATE
```

```
/*****************************************************************************************
 _start -- program entry point.
*****************************************************************************************/

        //      .function "_start", _start, _end_start - _start
_start:
        // Just to make "_sci_port", "_clock_freq", and "_baud_rate" appear in final ELF.
 /*   lrw                         r7, _sci_port
    lrw                         r7, _clock_freq
    lrw                         r7, _baud_rate*/

    // Just to make "__stack_end" appear in final ELF.
    lrw                         r7, __stack_end
        // init stack pointer.
    lrw                         r8, __stack_begin
    mov                         r0,r8

        // Initialize BSS section to 0.
        lrw     r5, __sbss
        lrw     r6, __ebss
        cmphs   r5,r6
        bt      _call_main
        subi    r6,1
        xor     r4,r4
_zero_bss:
        st.b    r4,(r6,0)
        cmplt   r5,r6
        loopt   r6, _zero_bss
//_call_static_init:
//      jbsr                    __call_static_initializers
_call_main:
    jbsr    main            // pull in main()
        bkpt

_program_end:
    bkpt
    bkpt

//_exit:
//    bkpt
//    bkpt

_end_start:

        .literals
```

Application Note

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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

freescale™
semiconductor