AN2541
Rev. 1, 8/2005

Bootstrapping the MSC812x Devices Over the DSI Port

By Joe Rebello

The Freescale MSC812x devices are StarCore™-based DSPs containing four SC140 cores for use in high density applications such as transcoding or packet telephony systems. The peripherals for streaming data into and out of the MSC812x devices include four time-division multiplexing (TDM) modules, a selectable 64/32-bit system bus, and a 64/32-bit direct slave interface (DSI) port. A host processor can assess the DSI port as an SRAM memory-mapped peripheral. Typically, the DSI functions as a high bandwidth path for data transfers, but it can also be used to bootstrap the device. When the MSC812x devices are bootstrapped through the DSI, the host must write a reset configuration word to the DSI port to bring it out of reset. Then the DSI is ready to receive program data written to its internal memory.

This application note and the associated software driver address the bootstrapping of a slave MSC812x device through the slave DSI port from an external MSC8103 host. The software driver provides a utility to convert an S-record into a byte-formatted C array that the MSC8103 host can write to the internal MSC812x memory. This utility is a reference example for developers and not a supported product. This document details the physical interconnections between a host MSC8103 system bus operating in Single-Master mode and the DSI port of the slave MSC812x, as well as the required host memory controller settings. A software section describes the boot process through the DSI port. The next section presents a method of creating a code image for the target slave DSP. The final section discusses the example hardware set-up of the MSC8122ADS board and gives step-by-step guidelines for running the example code, along with a listing of the example code.

CONTENTS

Note: The MSC8122 allows extended addressing of the DSI for a larger window size and access to the 60x bus. This application note does not discuss the extended mode.

1 Hardware Implementation

The DSI port operation is demonstrated on an MSC8122 Application Development System (MSC8122ADS), with the system bus of the host MSC8103 connected to the DSI port of the slave MSC812x. The host MSC8103 treats the DSI port of the slave MSC812x device as a memory-mapped region that is accessed through the system bus using the host MSC8103 memory controller UPM-controlled chip select.

The MSC812x DSI is a relatively simple interface that can be accessed as an asynchronous SRAM or a synchronous SRAM. This document illustrates only the asynchronous DSI mode. The host MSC8103 accesses the DSI port by asserting a chip select together with address, read/write, and byte strobes. The DSI also has four CHIP_ID[0–3] pins to identify up to 16 individual devices without the need for external logic. Several DSI data lines are also multiplexed for power-on reset options such as 32 or 64-bit mode, synchronous or asynchronous operation, and clock selections.

Figure 1 illustrates the MSC8122ADS connection between the MSC8103 and MSC812x devices. The DSI port on the MSC812x device is Big-Endian, so the data bus connection between the host system bus and slave DSI port is D[0–31/63] → HD[0–31/63], with the address bus connected so that A[11–29] → HA[11–29]. Address lines A[7–10] on the host processor connect to the HCID[0–3] signals that the MSC812x compares with the CHIPID pins. The MSC8103 GPL2 signal connects to the DSI HRW, and the DQM[0–3] byte strobes connect to $\overline{\text{HDBS}}$[0–3] for individual byte lane control.

1.1 Slave Hardware Pin Configuration

Several hardware pins sampled at power-on reset determine the boot source and operating mode. **Table 1** shows the pins and required values that enable the MSC812x for DSI bootstrap in 32-bit mode.

Table 1. Slave Hardware Pin Configuration

Pin	Value	Description
CNFGS	1	Reset configuration write occurs through the DSI port.
$\overline{\text{RSTCONF}}$	0	
EE0	0	SC140 core starts in normal processing mode after reset.
BM[0–2]	001	BM = 001 MSC812x boots from external host (DSI).
DSI64	0	DSI is 32 bits wide.
SWTE	0	Software watchdog is disabled.
DSISYNC	0	Asynchronous mode.
CHIPID[0–3]	1111	
MODCK[1–2]	11	Clock mode 11.

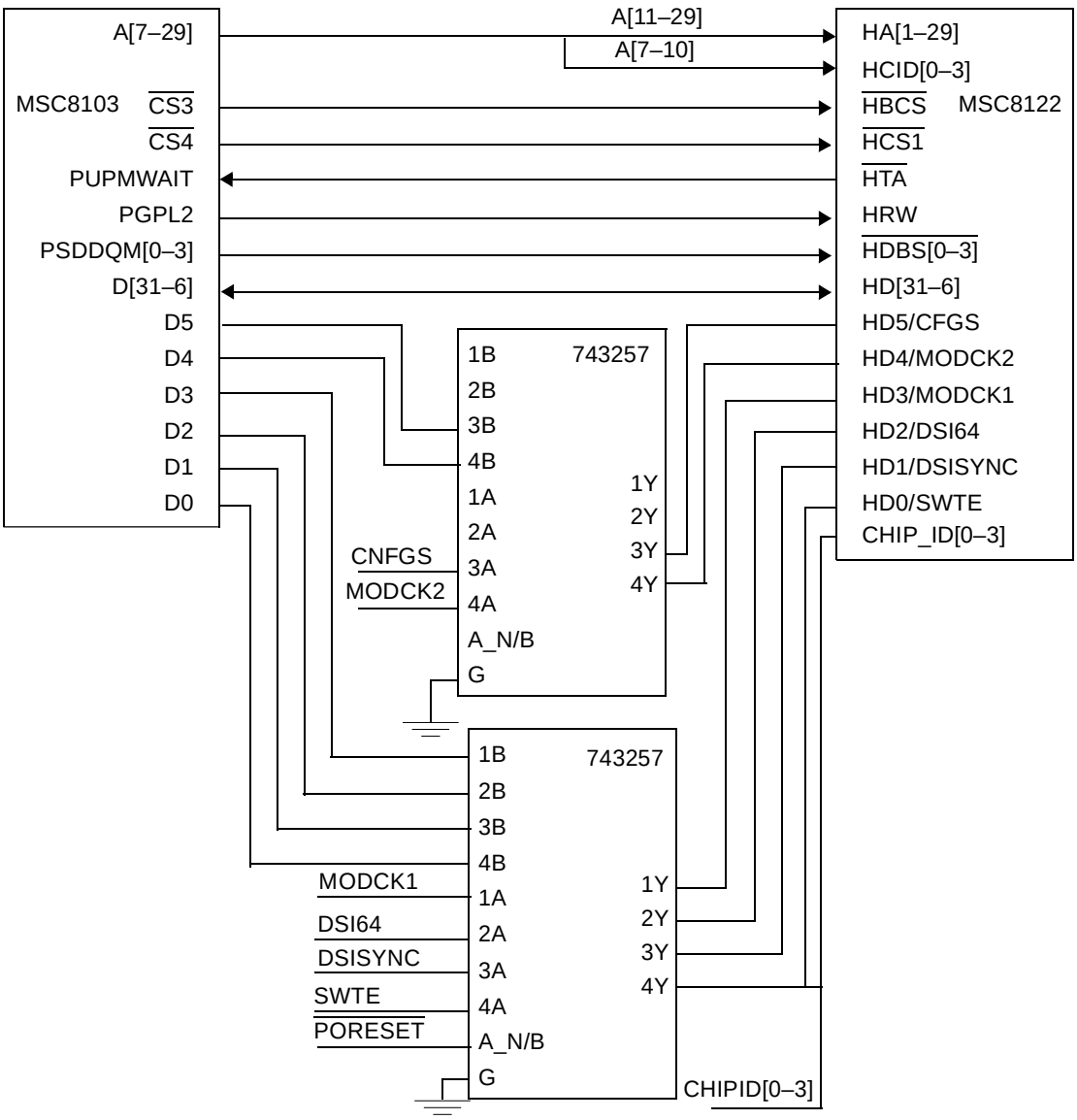

Figure 1. MSC8103 Host-to-MSC812x DSI Hardware Interface

1.2 Host MSC8103 Memory Controller Settings

The configurable MSC8103 memory controller provides a user-programmable machine (UPM), a general-purpose chip select machine (GPCM), and an SDRAM machine. The UPM-controlled system bus and the MSC812x DSI port are both programmable. You can program the MSC8103 memory controller and UPM RAM to meet all MSC812x DSI port timings. The UPM offers options to control the memory control signals to one quarter of one clock resolution. However, depending on the CPM:Bus clock ratio, the relative phases of this clock granularity may vary. Timing needs may change with different clock ratios. To ensure that the timing recommendations discussed here hold true at any clock speed or ratio, the analysis is performed using the maximum bus clock of 100 MHz for asynchronous mode and using only the invariable one half of one clock boundaries (T1 and T3) to change signals. Therefore, the recommendations hold true for anything less than a 100 MHz bus clock. The DSI port acknowledge signal, $\overline{\text{HTA}}$, indicates whether there is valid data to be read or the DSI is ready to accept a write. The $\overline{\text{HTA}}$ signal is fed into the $\overline{\text{PUPMWAIT}}$ pin of the MSC8103 memory controller. The MxMR[GPL4DIS] bit must be set to enable this mode, and the WAEN bit must be set in the corresponding UPM word to indicate a freeze of the UPM memory controller signals while $\overline{\text{PUPMWAIT}}$ is asserted. **Figure 2** shows the UPM-controlled DSI read

access, and **Figure 3** shows the write access. Six clocks are required for both read and write accesses on the system bus in Single-Master MSC8103 mode without data buffering.

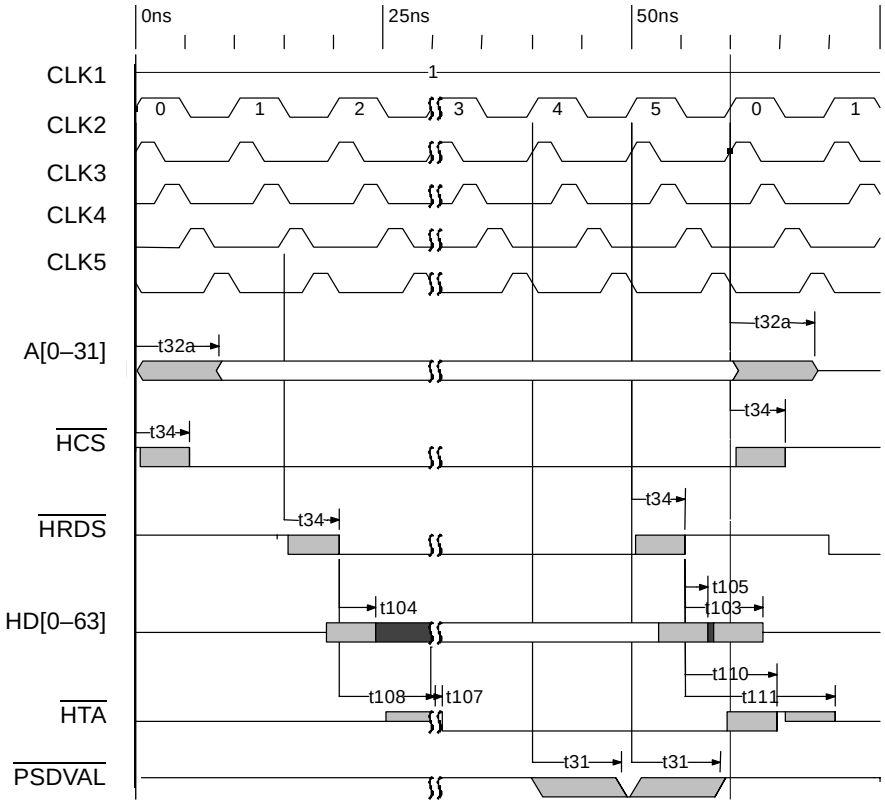

Figure 2. DSI UPM Read Cycle

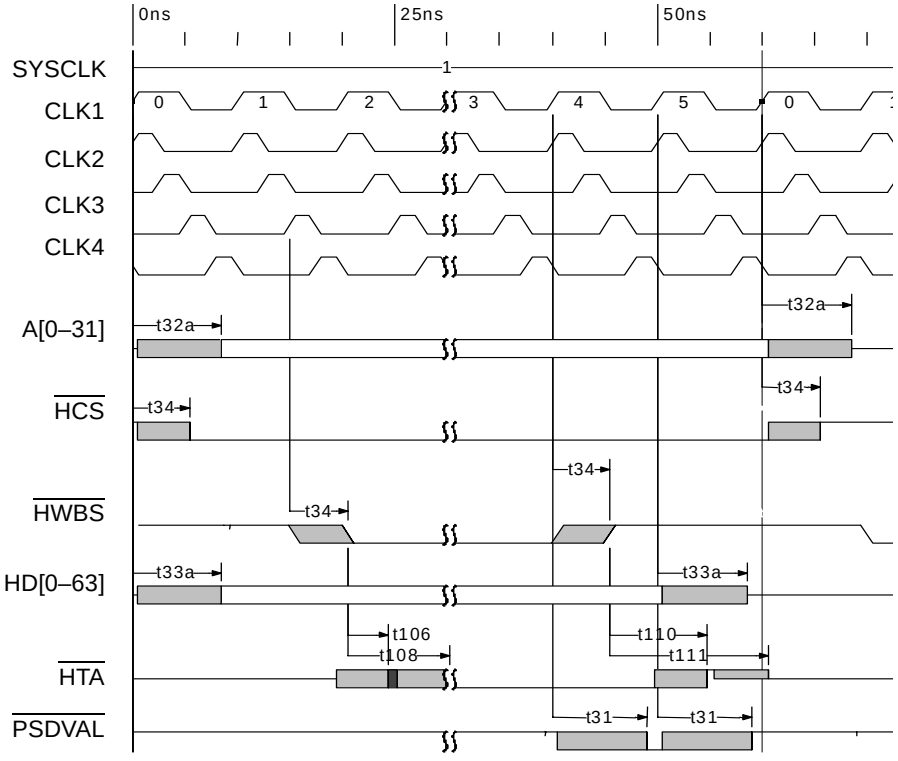

Figure 3. DSI UPM Write Cycle

To program the UPM with the memory profile of the MSC8103 system bus-to-DSI interface, the single-beat read and write entries are programmed as listed in **Table 2**. All other values in the array representing bursts and periodic timers are not required and can be disabled by setting them to 0xFFFFFFFF (or 0xFFFFFCFF).

Table 2. DSI UPM Settings

Cycle Type		Single Read	Burst Read	Single Write	Burst Write	Refresh	Exception
Offset in UPM		0x0	0x8	0x18	0x20	0x30	0x3C
Contents @Offset+	0x0	0x0F0FCC00	0xFFFFFFFF	0xFF0FCC00	0xFFFFFFFF	0xFFFFFFFF	0xFFFFC000
	0x1	0x0F0ECC00	0xFFFFFFFF	0x0C0FCC00	0xFFFFFFFF	0xFFFFFFFF	0xFFFFC005
	0x2	0x0F0CDD40	0xFFFFFFFF	0x000FDD40	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF
	0x3	0x0F0CCC04	0xFFFFFFFF	0x0F0FCC04	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF
	0x4	0x0F0FCC01	0xFFFFFFFF	0x0F0FCC01	0xFFFFFFFF	0xFFFFFFFF	
	0x5	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	
	0x6	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	
	0x7	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	
	0x8		0xFFFFFFFF		0xFFFFFFFF	0xFFFFFFFF	
	0x9		0xFFFFFFFF		0xFFFFFFFF	0xFFFFFFFF	
	0xA		0xFFFFFFFF		0xFFFFFFFF	0xFFFFFFFF	
	0xB		0xFFFFFFFF		0xFFFFFFFF	0xFFFFFFFF	
	0xC		0xFFFFFFFF		0xFFFFFFFF		
	0xD		0xFFFFFFFF		0xFFFFFFFF		
	0xE		0xFFFFFFFF		0xFFFFFFFF		
	0xF		0xFFFFFFFF		0xFFFFFFFF		

For our example, the following register settings are required.

Table 3. Host Memory Controller Register Settings

Register	Value	Description
BR[3]	0x26001801	Broadcast chip select at 0x26000000, GPCM controlled
OR[3]	0xFFE00884	
BR[4]	0x24001881	Base address of DSI port at 0x24000000, 32-bit port size, UPMA
OR[4]	0xFE000100	32 MB memory space, nonburst
MAMR	0x00C48880	60x system bus assigned, no refresh, normal operation
BCR	0x00000000	Single-Master MSC8103 bus mode

2 Bootstrapping the Slave MSC812x

When the DSI is selected as the boot module, an external host must bootstrap the device. This section describes the procedure for bootstrapping the slave MSC812x over the DSI port and the actions the host must take to download the reset configuration word and application code. The bootstrapping process has two phases:

- Writing the Hard Reset Configuration Word (HRCW) to the slave DSI port
- Writing the target application image to the slave DSI port

2.1 Writing the Reset Configuration Word to the DSI Port

When the MSC8103 is bootstrapped through the DSI port, it remains in reset until the HRCW is written via the DSI. A 32-bit value is written into the DSI mapped address space. After the HRCW is written, the MSC812x device locks the PLL and DLL, exits reset, and begins to execute its boot ROM program. The main actions of the slave MSC812x boot ROM program are as follows [4]:

1. Core 0 sets up the SIU and memory controller. Cores 1–3 wait for LICB interrupt 21.
2. Core 0 checks the boot mode, and if the mode is not TDM or UART, it sets the BR10[V] bit.
3. Core 0 waits for Virtual Interrupt 1 to be set.
4. When Core 0 receives a Virtual Interrupt 1, it wakes up Cores 1, 2, and 3.

To summarize, after Core 0 completes device initialization, it waits for the host to write to Virtual Interrupt 1. Core 0 then initializes the other three cores. All cores then jump to location 0 and execute code. **Figure 4** illustrates the host and slave data flows; the dotted arrows indicate dependencies.

The external host waits a number of bus cycles after writing the HRCW to ensure that the slave MSC812x device has exited reset before the host checks the BR10[V] bit. This Valid bit indicates that Core 0 has reached a certain point in the boot ROM program. The number of cycles to wait depends on the MSC812x CLKIN predivision factor (PDF) and the system bus (REFCLK) frequencies. **Table 4** shows how to determine the delay required.

Table 4. SPLL to $\overline{\text{SRESET}}$ Deassertion Timings

SPLL Lock Time = 6400 (CLKIN/RDF). SPLL lock to $\overline{\text{HRESET}}$ deassertion 512/REFCLK. SPLL lock to $\overline{\text{SRESET}}$ deassertion 515/REFCLK.	
Delay from reset configuration write to reset exit ($\overline{\text{SRESET}}$ negation):	
Assuming DLL is enabled:	= 1 + 3 = 6400 (CLKIN/RDF) + (515/REFCLK)

2.2 Writing the Target Application to the DSI Port

After the delay between writing the HRCW and checking the BR10[V] bit expires, the host downloads the code image to all the slave MSC812x DSP cores. Then the host writes to the Virtual Interrupt Register to initialize all the SC140 cores and start execution. Because all the SC140 cores begin executing code from location 0, the downloaded code image should indicate the entry point, and valid code must exist in all the SC140 cores. Note that the MSC812x M1 and M2 memory areas have different addresses as viewed through the DSI port. The host must interpret the addresses and select the area of memory in which to place the target application code. **Example 1** shows a piece of code to illustrate this process.

Example 1. Code Distribution to M1 and M2 Memories

```
void WriteProgramBlock(UByte *pusiData, UWord32 uliAddress)
{
   UByte   uliSize;
   UByte   *puliTempData;
   UWord32 ulii;
   UByte   uclast;
   UByte   *puscdsipointer;
   UWord32 uliDestAddress;
   uclast = 0;

   puliTempData = (UByte*)pusiData;

   while (uclast == 0)
   {
      /* Read the size of the S-record from the array*/
      uliSize = *puliTempData++;

      /* If size is 0 then end load */
      if(uliSize != 0)

      {
         /* Read the next 4 bytes to create the 32-bit address */
         uliDestAddress = (*puliTempData++)<<24;
         uliDestAddress |= (*puliTempData++)<<16;
         uliDestAddress |= (*puliTempData++)<<8;
         uliDestAddress |= *puliTempData++;

         if(uliDestAddress < 0x38000) /* Size of M1 memory */
         {
            /* Write Data  to each core into M1 memory*/

            for(ulii=0; ulii<uliSize; ulii++)
            {
               puscdsipointer = (UByte*)(uliDestAddress+CORE0M1_DSI_ADDRESS+uliAddress);
               *puscdsipointer = *puliTempData;
               puscdsipointer = (UByte*)(uliDestAddress+CORE1M1_DSI_ADDRESS+uliAddress);
               *puscdsipointer = *puliTempData;
               puscdsipointer = (UByte*)(uliDestAddress+CORE2M1_DSI_ADDRESS+uliAddress);
               *puscdsipointer = *puliTempData;
               puscdsipointer = (UByte*)((uliDestAddress++)+CORE3M1_DSI_ADDRESS+uliAddress);
               *puscdsipointer = *puliTempData++;
            }
         }

         /* if too destination in L2 then Write Data  L2 memory*          */
         else
         {
            puscdsipointer = (UByte*)(uliDestAddress+uliAddress-0x01000000);
           for(ulii=0; ulii<uliSize; ulii++)
           {
              *puscdsipointer++ = *puliTempData++;
           }
         }

      }
      else
      {
         uclast = 1;
      }
   }
}
```

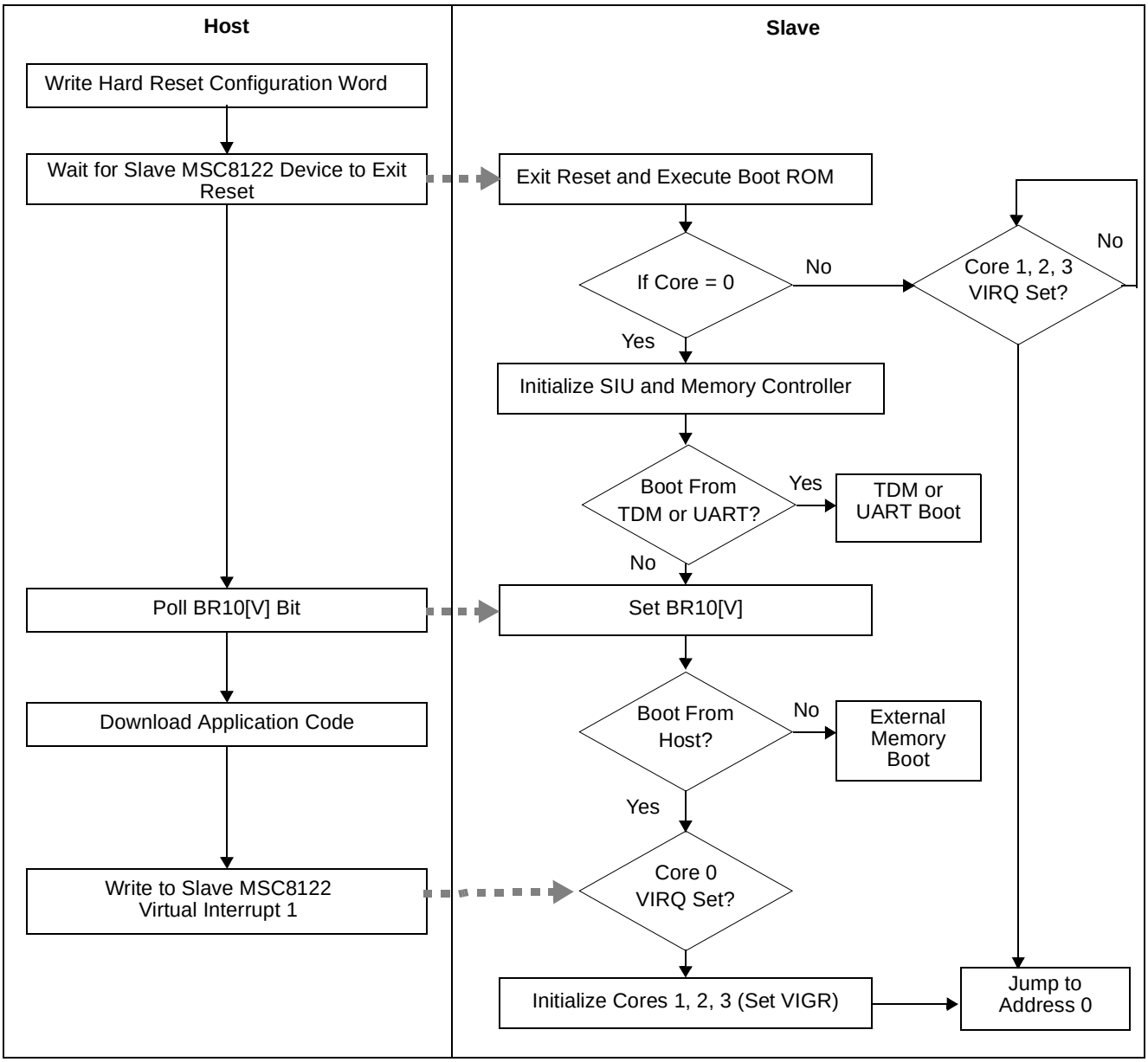

Figure 4. Host and Slave Bootstrap Data Flows

3 Creating the Target Application Image

The fact that all the MSC812x SC140 cores begin to execute code at about the same time has a number of implications on how the target code is written. For example, you must decide whether to have individual code images for all four SC140 cores. Individual code images may be appropriate if the SC140 cores are running completely different applications, but it increases the amount of data space required by the host. If the SC140 cores are executing the same piece of code or similar code, one project can be implemented that makes decisions based on the Core ID number contained within the Core ID Register (CIDR). The use of the Core ID number slightly increases the size of the target application code, but it uses less host memory space and makes the target application more manageable. The example discussed in this document uses the Core ID number method for simplicity, though allocating individual code images to the SC140 cores is an equally valid method. Only one SC140 core should set

up common registers, such as the SIU or IPBus-mapped registers. Typically Core 0 is used to set up the chip configuration, and a hardware or software semaphore is used to indicate to the other SC140 cores that the MSC812x device is initialized into a known state.

The host MSC8103 device must write the program code into the internal memory space of the slave MSC812x. However, there is no stipulated method for writing this code. One method is to use the S-Record format that can be generated using the CodeWarrior™ for StarCore tools from the compiled embedded link format object (elf), as described in **Figure 5**.

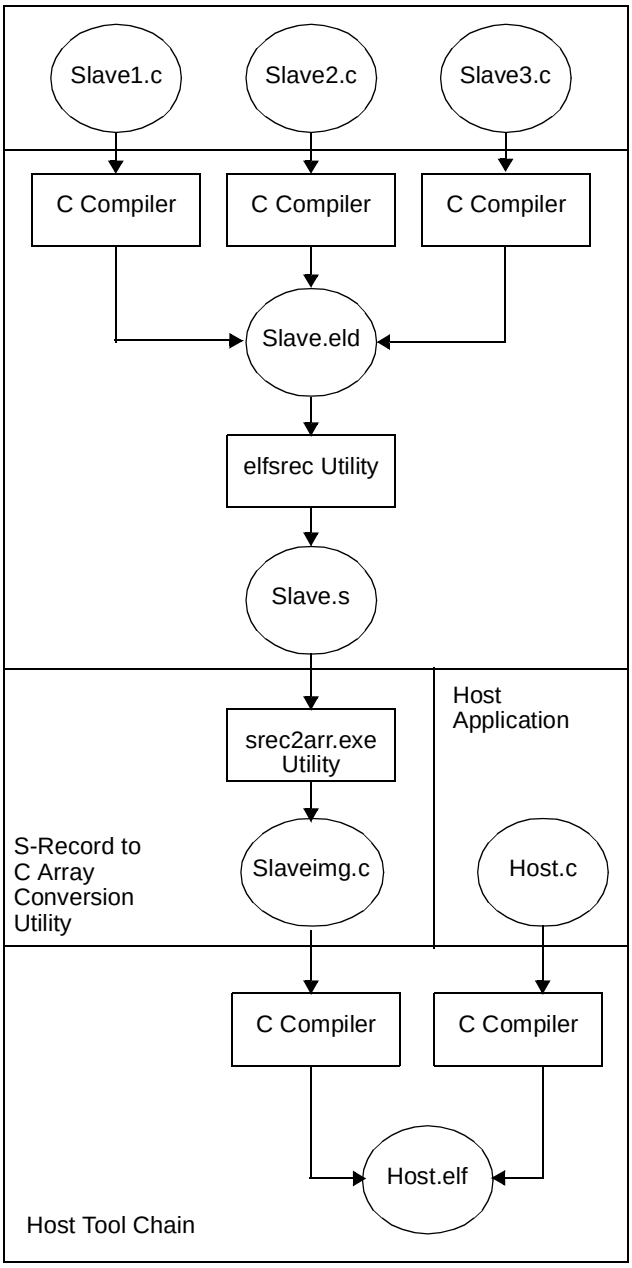

Figure 5. Conversion Process

Using the CodeWarrior tools, the slave application files are compiled, assembled, and linked to produce an elf object (`.eld` file), which is first converted to an S-record using the CodeWarrior `elfsrec.exe` application and then to a C array that can be linked into the host bootstrap application The application is invoked with the following DOS command:

```
elfsrec.exe -l <filename.eld>
```

The – l specifies that an S3 format (32-bit) S-record is to be generated with the resulting output file name being `<filename.eld.s>`, which should be renamed to `<filename.s>`. The next step is to run the `srec2arr.exe` utility on this file with the following DOS command:

```
srec2arr.exe
```

A new window appears that prompts you for an input and output filename to generate an output file. `srec2arr.exe` is a small utility that takes the address, length, and data from each line of the S3 record and converts it to a C array in a byte format, as follows:

- 1 byte length
- 4 byte address
- Number of bytes described in length as data

Table 5 illustrates the structure of one line of an example S-record (numbers in hexadecimal) and the converted equivalent C array format.

Table 5. Format of One Line of an S-Record and Converted C Array

S-Record Type	Length	Address	Data/Data Words	CRC
S3	1D	0x000011C0	0x3104303A810090C090C090C0 90C090C090C090C0E0010000	E0
S3	0x18	0x00,0x00,0x11,0xC0 ,	0x31,0x04,0x30,0x3A,0x81,0x00,0x90,0xC0, 0x90,0xC0,0x90,0xC0, 0x90,0xC0,0x90,0xC0, 0x90,0xC0,0x90,0xC0,0xE0,0x01,0x00,0x00,	

The conversion process first strips the S-record type. The S-record length is a byte value with five bytes subtracted from it (four byte address and a one byte CRC). The address field of an S3 record is four bytes and is written to the array. The `srec2arr.exe` utility does not support the S1 and S2 record types, although the code can easily be modified to handle them. The actual data words are converted to bytes with the CRC removed. The last line of the array includes an end block (length 0 address 0), which can be used to indicate to the host that all the code has been downloaded. The `srec2arr.c` listing is provided in **Section 6**, *Code Listing: SREC_TO_ARRAY.C*, on page 13 for reference and can be freely modified to suit individual requirements.

4 MSC8122ADS Test Configuration Example

A simple example of bootstrapping an MSC812x device through the DSI is provided to test the hardware set-up and the software driver functionality. The test software initializes the host MSC8103 memory controller so that the DSI of the slave MSC812x device is mapped as a 64-bit port at location 0x25E00000. At start-up, the host writes the HRCW to the slave and then for reset to complete before loading the target application. The slave begins to execute the downloaded code, which, in this example, simply writes to locations in the MSC812x M2 memory to indicate that the SC140 cores are initialized. **Table 6** details the hardware set-up for the MSC8122ADS board.

Table 6. Slave MSC8122ADS Hardware Switch Settings

Switch	Value	Description
SW4 [1] [2] [3] [4]	 OFF ON ON OFF	 MODCK [2–1] = 11 Clock mode 11 with MODCK[3–5] 010. Boot and configuration through the DSI port. Cores do not enter Debug mode after reset.
SW5 [1] [2] [3] [4]	 OFF ON ON ON	 Single MSC8103 JTAG mode. System bus is 64 bits wide. N./A
SW6 [1] [2] [3] [4] [5] [6] [7] [8]	 OFF ON ON ON ON ON OFF ON	 MSC8103 clock mode. MODCK[3–1] = 100. Clock mode 1. MODCK[6–4] = 000. Configuration from BCSR. Core enters Debug mode after reset.
SW7 [1] [2] [3] [4]	 OFF ON ON OFF	 Ethernet is OFF.
JP 1	CLOSED	
JP 2	CLOSED	OSC
JP 3	OPEN	
JP 4	ALL OPEN	
JP 5	2–3	CLKOUT
JP 7	OPEN	
JP 8	OPEN	
JP 9	8122_2	
JP 10	OPEN	
JP 2000	CLOSED	
U31	50 MHz	MSC8103 clock oscillator.
U5	33.3 MHz	MSC8122 clock oscillator.

4.1 Create the Slave Image C File

1. Load the slave application project into the CodeWarrior tools by double clicking on the `MSC8122Boot.mcp` file in `. . . \8122_DSI_Boot_Example\MSC8122Boot\build\`.
2. Build the project to create the `.eld` file, which is named `MSC8122.eld`, and close the project.
3. Execute the `elfsrec` utility in a DOS window using the command line: `elfsrec – l MSC8122.eld`.
4. After the `MSC8122.eld.s` file is created, rename it to `MSC8122.s`.

5. Execute the `srec2arr.exe` utility on the `MSC8122.s` file by double clicking on the file, or in a DOS window, use the `srec2arr` command.

   Note that the `srec2arr.exe` file is located in `. . .\MSC8122_DSI_Boot_Example\MSC8122Boot\build`.

6. When a window appears and prompts you for the input filename, enter `MSC8122.s` and press return.

   When you are prompted to enter an output file name, enter `MSC8122.c`. Note that you can choose any name. The name used here is for this particular example.

7. Close the window so that the output file can be closed.

8. After the `MSC8122.c` file is created, copy it to the location `. . . \MSC8122_DSI_Boot_Example\host\source`.

9. Note that steps 3–8 can be executed using the `Generate_Array.bat` batch file contained in the folder:

   `. . .\MSC8122_DSI_Boot_Example\MSC8122Boot\build`

10. Open the project `host.mcp` file in the following folder and compile the project:

    `. . .\MSC8122_DSI_Boot_Example\host\build`

4.2 Download the File Onto MSC8122ADS Via EOnCE

1. Initialize the host and slave switch settings to those indicated in **Table 6**.

2. Connect the CodeWarrior JTAG OnCE™ Wiggler to P14.

3. Switch on the board and press PRESET (SW9).

4. Load the software (CodeWarrior project `host.mcp`) onto the MSC8122ADS using the debug port.

   The LD12 and LD13 LEDs should be ON, indicating that the MSC812x is in reset.

5. Start the program

   The LD12 and LD13 LEDs should turn OFF.

6. Stop the debugger and open a memory window in CodeWarrior. Change to the 32-bit view and look at location 0x25e00000.

   The first four rows of the first two word columns are as shown in **Figure 6**, indicating that the SC140 cores have booted and have executed application code that writes the Core ID number to M2 memory.

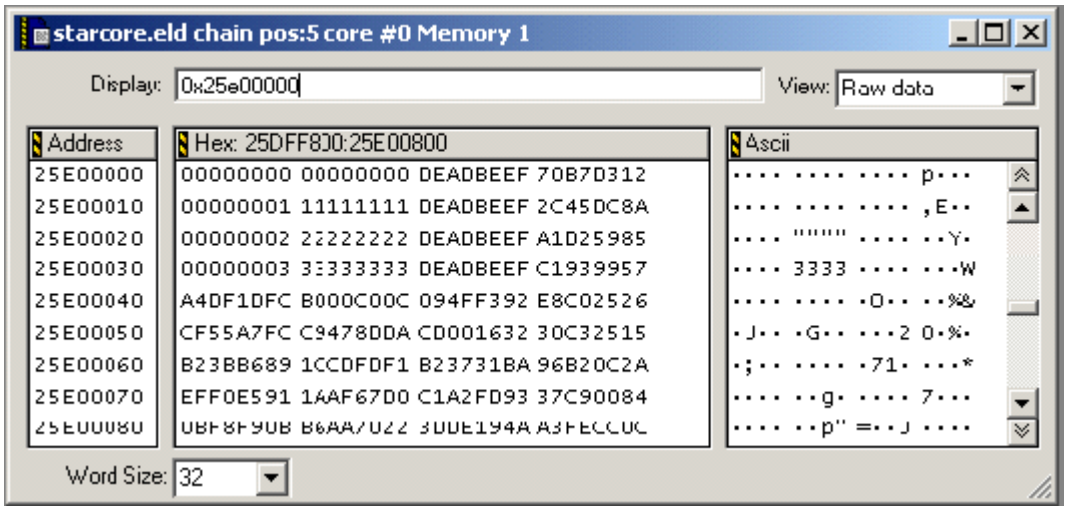

Figure 6. CodeWarrior Memory Window of M2 Memory Seen from the MSC8103 Device

5 Related Documents

[1] *SC140 DSP Core Reference Manual* (MNSC140CORE)

[2] *MSC8103 Reference Manual* (MSC8103RM)

[3] *MSC8103 Application Development System User's Manual* (MSC8103ADSUM)

[4] *MSC8122 Reference Manual* (MSC8122RM).

[5] *MSC8126 Reference Manual* (MSC8126RM).

6 Code Listing: SREC_TO_ARRAY.C

```
#include <stdio.h>          //'sprintf'
#include <string.h>         //'strlen', 'strncmp'
#include <stdlib.h>         //'atoi'

/************************
*| DEFINES
************************/
/* ASCII characters */
#define CHAR_NUMBER0     0x30
#define CHAR_NUMBER1     0x31
#define CHAR_NUMBER2     0x32
#define CHAR_NUMBER3     0x33
#define CHAR_NUMBER4     0x34
#define CHAR_NUMBER5     0x35
#define CHAR_NUMBER6     0x36
#define CHAR_NUMBER7     0x37
#define CHAR_NUMBER8     0x38
#define CHAR_NUMBER9     0x39

#define CHAR_LETTER_A    0x41
#define CHAR_LETTER_B    0x42
#define CHAR_LETTER_C    0x43
#define CHAR_LETTER_D    0x44
#define CHAR_LETTER_E    0x45
#define CHAR_LETTER_F    0x46
```

```
void main()
{
FILE *pfiIn;
FILE *pfiOut;
char ucCh0;
char ucCh1;
int  uliCounter;
char ucLength;
char aucTemp[2];
int  ulii;
int  ulij;
int  uliRemainder;
int  uliSize;
int  uliEndFlag;
char szCurrentString[120];
char szTempString[120];
char input[80],output[80];

uliEndFlag = 0;
        printf("Enter the input file name:");
        scanf("%s", input);

        if ((pfiIn=fopen(input, "rb")) == NULL)
        {
             printf("Error opening file\n");
             exit(1);
        }

        printf("Enter the output file name:");
        scanf("%s", output);

        if ((pfiOut=fopen(output, "wb")) == NULL)
        {
             printf("Error creating file\n");
             exit(1);
        }
/* Get line header S? */
ucCh0 = getc(pfiIn);
ucCh1 = getc(pfiIn);

if ((ucCh0=='S')&&(ucCh1=='0'))
{
    uliCounter = 0x0000;
    for (ulii=0; ulii < 2; ulii++)
    {
         uliCounter <<= 4;
         ucLength = getc(pfiIn);

         if (ucLength == CHAR_NUMBER1) uliCounter |= 0x0001;
         if (ucLength == CHAR_NUMBER2) uliCounter |= 0x0002;
         if (ucLength == CHAR_NUMBER3) uliCounter |= 0x0003;
         if (ucLength == CHAR_NUMBER4) uliCounter |= 0x0004;
         if (ucLength == CHAR_NUMBER5) uliCounter |= 0x0005;
         if (ucLength == CHAR_NUMBER6) uliCounter |= 0x0006;
         if (ucLength == CHAR_NUMBER7) uliCounter |= 0x0007;
         if (ucLength == CHAR_NUMBER8) uliCounter |= 0x0008;
         if (ucLength == CHAR_NUMBER9) uliCounter |= 0x0009;

         if (ucLength == CHAR_LETTER_A) uliCounter |= 0x000A;
         if (ucLength == CHAR_LETTER_B) uliCounter |= 0x000B;
         if (ucLength == CHAR_LETTER_C) uliCounter |= 0x000C;
         if (ucLength == CHAR_LETTER_D) uliCounter |= 0x000D;
         if (ucLength == CHAR_LETTER_E) uliCounter |= 0x000E;
         if (ucLength == CHAR_LETTER_F) uliCounter |= 0x000F;
    }
```

```
}
for(ulii=0; ulii<(uliCounter*2); ulii++)
{
    aucTemp[0] = getc(pfiIn);
}
/* get carriage return */
aucTemp[0] = getc(pfiIn);
aucTemp[0] = getc(pfiIn);
fprintf(pfiOut, "#include \"prototype.h\"// global defines \n");
fprintf(pfiOut, "extern UByte ausiImage1[]={");
/* Get line header S? */
ucCh0 = getc(pfiIn);
ucCh1 = getc(pfiIn);
while(uliEndFlag==0)
{
    putc('0',pfiOut);
    putc('x',pfiOut);
    uliCounter = 0x0000;
    for (ulii=0; ulii < 2; ulii++)
    {
        uliCounter <<= 4;
        ucLength = getc(pfiIn);
        if (ucLength == CHAR_NUMBER1) uliCounter |= 0x0001;
        if (ucLength == CHAR_NUMBER2) uliCounter |= 0x0002;
        if (ucLength == CHAR_NUMBER3) uliCounter |= 0x0003;
        if (ucLength == CHAR_NUMBER4) uliCounter |= 0x0004;
        if (ucLength == CHAR_NUMBER5) uliCounter |= 0x0005;
        if (ucLength == CHAR_NUMBER6) uliCounter |= 0x0006;
        if (ucLength == CHAR_NUMBER7) uliCounter |= 0x0007;
        if (ucLength == CHAR_NUMBER8) uliCounter |= 0x0008;
        if (ucLength == CHAR_NUMBER9) uliCounter |= 0x0009;
        if (ucLength == CHAR_LETTER_A) uliCounter |= 0x000A;
        if (ucLength == CHAR_LETTER_B) uliCounter |= 0x000B;
        if (ucLength == CHAR_LETTER_C) uliCounter |= 0x000C;
        if (ucLength == CHAR_LETTER_D) uliCounter |= 0x000D;
        if (ucLength == CHAR_LETTER_E) uliCounter |= 0x000E;
        if (ucLength == CHAR_LETTER_F) uliCounter |= 0x000F;
    }
    uliSize = uliCounter-5;      /* take away ucLength and crc */
    if (uliSize <= 0xF)
    {
        fprintf(pfiOut, "0%X", uliSize);
    }
    else
    {
        fprintf(pfiOut, "%X", uliSize);
    }
    /* Write the 4 byte address */
    for (ulij=0; ulij<4; ulij++)
    {
                putc(',',pfiOut);
                putc('0',pfiOut);
                putc('x',pfiOut);
```

```
                    aucTemp[0] = getc(pfiIn);
                    putc(aucTemp[0],pfiOut);
                    aucTemp[0] = getc(pfiIn);
                    putc(aucTemp[0],pfiOut);
               }
               /* Get data words */
               for (ulii=0; ulii<(uliCounter-5); ulii++)
               {
                    putc(',',pfiOut);
                    putc('0',pfiOut);
                    putc('x',pfiOut);
                    aucTemp[0] = getc(pfiIn);
                    putc(aucTemp[0],pfiOut);
                    aucTemp[0] = getc(pfiIn);
                    putc(aucTemp[0],pfiOut);
               }
               /* read checksum and carriage return */
                    aucTemp[0] = getc(pfiIn);
                    aucTemp[0] = getc(pfiIn);
                    aucTemp[0] = getc(pfiIn);
                    aucTemp[0] = getc(pfiIn);
                    /* Get line header S? */
                    ucCh0 = getc(pfiIn);
                    ucCh1 = getc(pfiIn);
                    if((ucCh0=='S')&&(ucCh1=='7'))
                    {
                         uliEndFlag = 1;
                    /* write end block for host to see end of array
                    length = 0 */
               for(ulii=0; ulii<8; ulii++)
               {
                    putc(',',pfiOut);
                    putc('0',pfiOut);
                    putc('x',pfiOut);
                    putc('0',pfiOut);
                    putc('0',pfiOut);
                    putc('0',pfiOut);
                    putc('0',pfiOut);
               }
               fprintf(pfiOut, "};");
               putc('\n',pfiOut);
               break;
               }
               else
               {
                    putc(',',pfiOut);
                    putc('\n',pfiOut);
                    putc(' ',pfiOut);
               }
          }
          printf("\n ");
          printf("Conversion Complete");
}
```

NOTES:

NOTES:

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 München, 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

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

Document Order No.: AN2541
Rev. 1
8/2005