AN2311
Rev. 1, 11/2005

Bootstrapping the MSC8101 Device Through the HDI16 Port

By Joe Rebello

The StarCore™-based MSC8101 device incorporates a communications processor module (CPM) with high-speed serial communications interfaces and a system integration unit (SIU). The SIU can connect to external memory, such as SRAM, Flash memory, SDRAM, and peripheral devices. The external system bus can be configured in either a 64-bit or 32-bit wide mode, the latter enabling the use of the 16-bit wide slave host data interface (HDI16). An external host can read and write to the slave HDI16 port to transfer control information and data and to bootstrap the device. When bootstrapping through the HDI16 port, the external host is required to write the Hard Reset Configuration Word (HRCW) as well as the desired target application to the slave MSC8101 device. A small boot program in the internal ROM of the slave MSC8101 receives the application code in a certain format, which it parses and executes.

This document describes a software driver to bootstrap a slave MSC8101 device through the slave HDI16 port from an external MSC8101 host. The driver software includes a utility to convert an S-record into the format required by the MSC8101 internal boot ROM. The device driver is provided (in the accompanying `.zip` file `AN2311SW.zip`) as an example for developers and not as a supported product. The physical connections between a host MSC8101 system bus in Single-Master mode and the HDI16 port of the slave MSC8101 are also described. After a discussion of the boot process through the HDI16 port and a method of creating a code image for the target slave DSP, there is an example hardware set-up of two MSC8101 application development system (MSC8101ADS) boards and instructions on how to run the example code.

CONTENTS

1 Hardware Implementation 2
2 Bootstrapping Process 5
3 Target Application Image 8
4 MSC8101ADS Test Configuration Example 10
5 Related Documents 13

1 Hardware Implementation

The HDI16 port operation requires two MSC8101 devices (a host and a slave), with the system bus of the host connected to the HDI16 port of the slave (see **Figure 1**). The host accesses the HDI16 port of the slave device as a memory-mapped region. The host gains access through the system bus using its own memory controller UPM-controlled chip select. The HDI16 port has two chip select signals that are logically ORed internally. The first ($\overline{\text{HCS1}}$) selects individual devices, and the other ($\overline{\text{HCS2}}$) typically broadcasts data to a number of devices—for example, in DSP farm applications. For the driver discussed here, $\overline{\text{CS6}}$ on the host DSP connects to $\overline{\text{HCS1}}$ of the slave. The HDI16 port is big-endian, so the data bus connection between the host system bus and slave HDI16 port is D[0–15] → HD[0–15], with the address bus connected so that A[27–30] → HA[0–3]. The interface uses the dual strobe mode with separate read ($\overline{\text{HRD}}$) and write ($\overline{\text{HWR}}$) data strobes connected to PGPL2 and $\overline{\text{PDQM0}}$, respectively, on the host.

The slave HDI16 port generates interrupts to the host in either single or dual request mode. Dual request mode is usually preferred because separate request lines indicate a read ($\overline{\text{HRRQ}}$) or write ($\overline{\text{HTRQ}}$) request, whereas the single request mode indicates only that HDI16 is ready to read or write data. Therefore, the host must poll the HDI16 registers to determine whether the request is for a read or a write, thus adding overhead. Since sufficient $\overline{\text{IRQ}}$ inputs are available on the host, the driver uses dual request mode. The two request lines from the slave MSC8101 connect to $\overline{\text{IRQ}}$[4–5] on the host for interrupt servicing options.

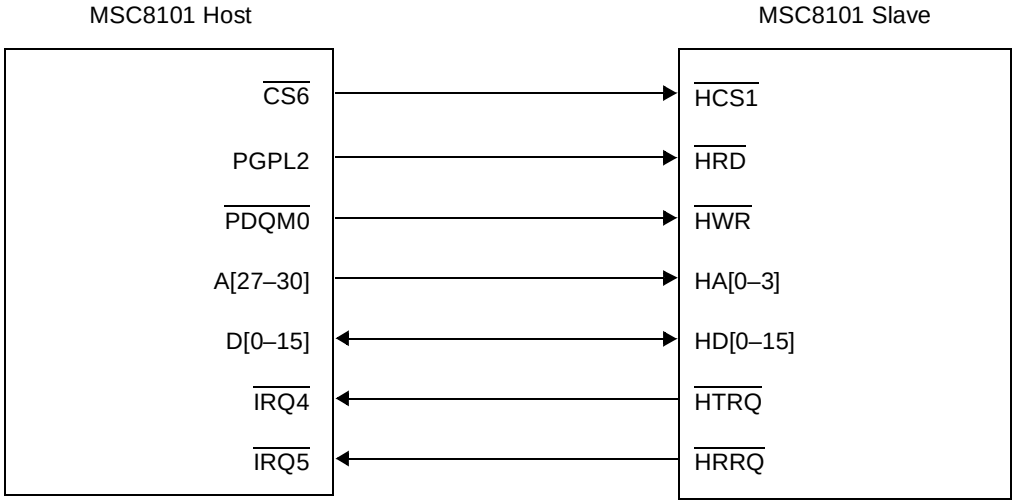

Figure 1. MSC8101 Host to HDI16 Hardware Interface

At power-on reset, several pins that determine the boot source and chip mode of operation are sampled. **Table 1** shows the pins that enable the MSC8101 for HDI16 operation in 16-bit dual data strobe mode.

Table 1. Slave Hardware Pin Configuration

Pin	Value	Description
$\overline{\text{RSTCONF}}$	1	Configuration slave selected
EE0	0	SC140 core starts in normal processing mode after reset
HPE/EE1	1	HPE=1, Host port Enabled
BTM[0–1]/EE[4–5]	01	BTM=01 MSC8101 boots from HDI16
HDDS	1	Dual data strobe mode enabled
H8BIT	0	16-Bit mode selected

The MSC8101 device has a highly configurable memory controller with a user-programmable machine (UPM) interface, a general-purpose chip select machine (GPCM), and an SDRAM machine.

The UPM-controlled 60x-compatible system bus and HDI16 port are both programmable. You can program the memory controller and UPM RAM to meet all the MSC8101 host port timing requirements. The UPM offers very flexible memory control options with one quarter clock resolution. However, depending upon the CPM:bus clock ratio, the relative phases of this one quarter of one clock granularity may vary. Timing needs may change with different clock ratios. To ensure that the timing recommendations developed here hold true at any clock speed or ratio, the analysis is performed using the maximum bus clock of 100 MHz 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.

During a read access, the MSC8101 device latches data on the falling edge rather than on the usual rising clock edge. The result is a sufficient timing margin to incorporate data buffer data delay with the same timing settings still in effect. The DLT3 bit must be set in the corresponding UPM word to indicate the data latch point on the falling clock, and MxMR[GPL4DIS] must be set to enable this mode. In a real system scenario, as shown in **Figure 2**, buffering can be required, so the timings must be adjusted accordingly. Furthermore, the read and write strobe deassertion times are readily met with the illustrated UPM configuration, but this is difficult to achieve with a competitive memory access profile in the alternative GPCM-controlled case.

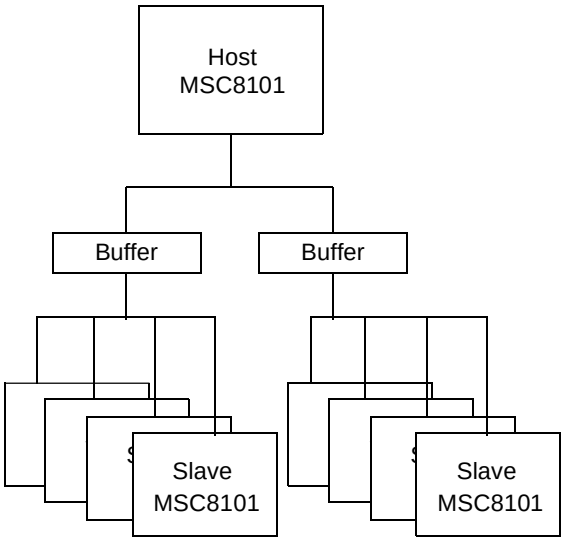

Figure 2. DSP Farm System

The UPM-controlled HDI16 read and write accesses are illustrated in **Figure 3** and **Figure 4**, respectively. Both the read and write accesses on the system bus operating in Single-Master MSC8101 mode without data buffering can be accessed within five clocks. In both figures, the box numbers refer to the MSC8101 timing specification. For details, see the *MSC8101 Technical Data* sheet.

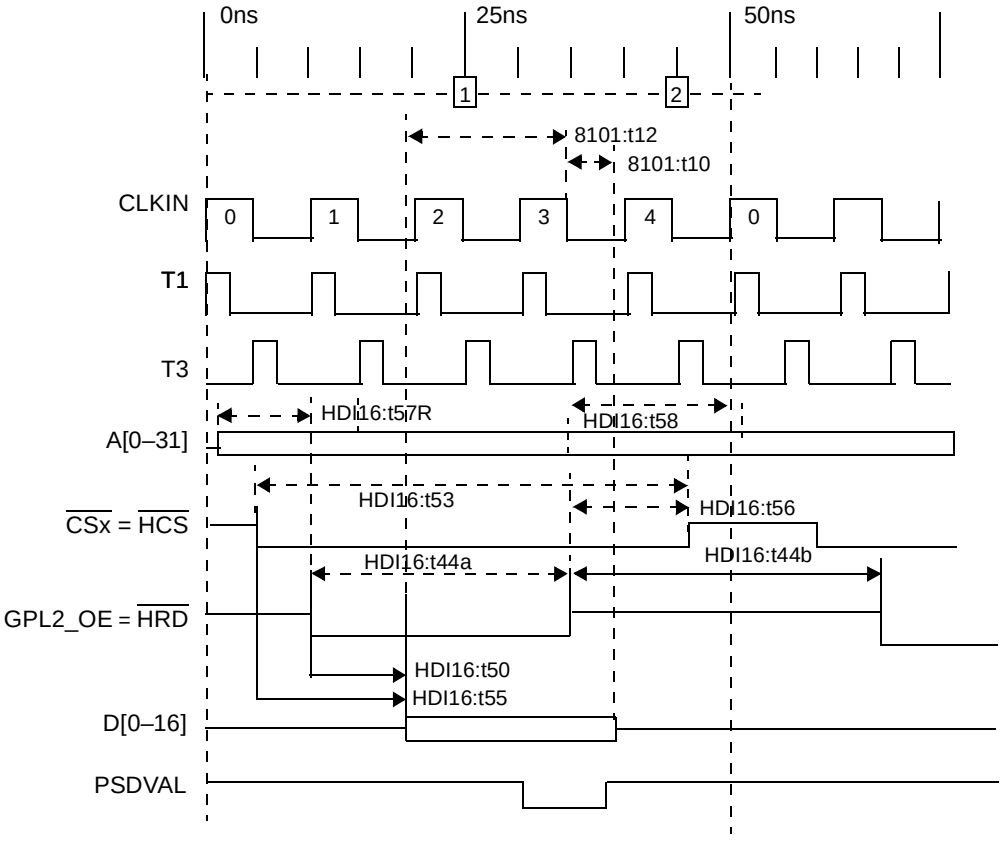

Figure 3. HDI16 UPM Read Cycle

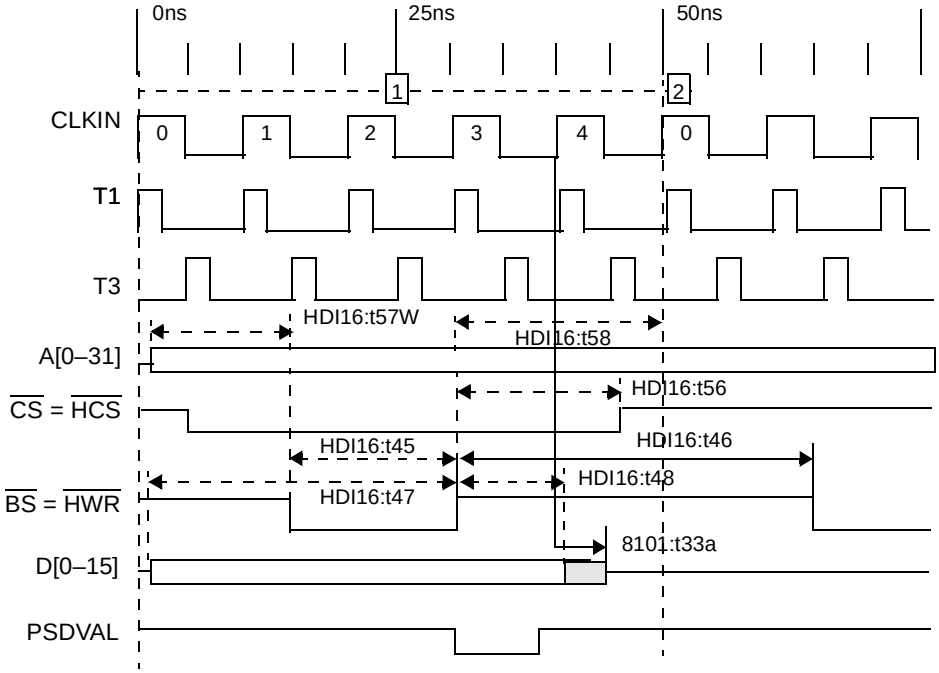

Figure 4. HDI16 UPM Write Cycle

To program the UPM with the memory profile for the MSC8101 system bus-to-HDI16 interface, the single-beat read and write entries are as listed in **Table 2**. All other values in the array representing bursts and periodic timers are not required, and you can set them to 0xFFFFFFFF (or 0xFFFFFCFF) to disable them.

Table 2. HDI16 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	0xCFFFEC00	0xFFFFFFFF	0xCFFFCC00	0xFFFFFFFF	0xFFFFFFFF	0xFFFFC000
	0x1	0x0FFCEC00	0xFFFFFFFF	0x0CFFCC00	0xFFFFFFFF	0xFFFFFFFF	0xFFFFC005
	0x2	0x0FFCEC00	0xFFFFFFFF	0x00FFCC00	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF
	0x3	0x0FFDEC04	0xFFFFFFFF	0x0FFFCC04	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF
	0x4	0x3FFFEC01	0xFFFFFFFF	0x3FFFCC01	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 the example driver, the register settings shown in **Table 3** are required.

Table 3. Host Memory Controller Register Settings

Register	Value	Description
BR6	0x04001081	Base address of the HDI16 port at 0x04000000, 16-bit port size, UPMA
OR6	0xFFFF8100	32 KB memory space
MAMR	0x00000000	System bus assigned, no refresh, normal operation
BCR	0x00000000	Single Master MSC8101 bus mode

2 Bootstrapping Process

When the HDI16 mode of operation is selected, an external host must bootstrap the device. This section describes the procedure for bootstrapping the slave MSC8101 over the HDI16 port and the host actions required to download the reset configuration word and application code.

The bootstrapping process has to two parts:

- Writing the Hard Reset Configuration Word (HRCW) to the slave HDI16.
- Writing the target application image to the slave HDI16.

When the MSC8101 device is bootstrapped through the HDI16 port, it remains in reset until the HRCW is downloaded via the HDI16 by writing four 8-bit values to the Reset Configuration Registers (RSCFG[0–3]) at host address offsets 0x8, 0x9, 0xA, and 0xB, respectively. The order of the bytes is important because the LSB of the HRCW must be written to location 0x8. Once all reset configuration bytes are written to RSCFG[0–3], the MSC8101 device locks the PLL and DLL, exits reset, and begins executing its boot ROM. The main actions of the slave MSC8101 boot ROM are as follows:

1. Get the IMMR of the device using the ISBSEL value in the HRCW.
2. Initialize the internal SRAM ($\overline{\text{CS10}}$ and UPMC), depending on the SCCR settings.
3. Disable the software watchdog.
4. Enable and set up the host port.
5. Load all program blocks, with size not equal to 0.
6. Load the end block with size equal to 0 and execute the application code.

The host MSC8101 device performs step 5 by reading the application image array and downloading it word by word to the HDI16 port. **Figure 5** shows the data transfer flow performed by the host, as well as the respective actions from the slave MSC8101 boot ROM. Steps 4, 5, and 6 are indicated on the figure.

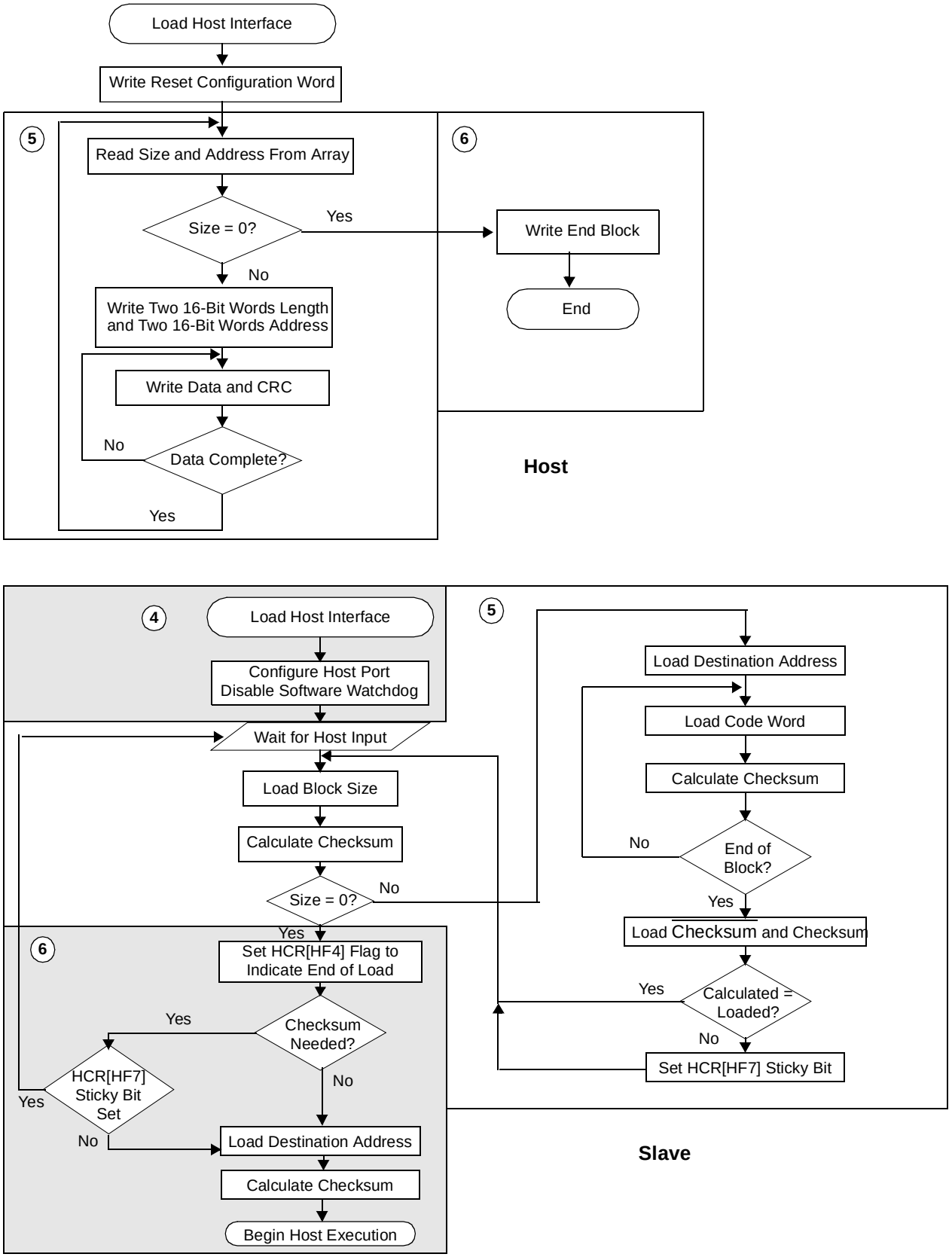

Figure 5. Host and Slave Bootstrap Data Flows

3 Target Application Image

The bootstrap process over the HDI16 port requires data to be transferred in blocks. **Table 4** shows the block format.

Table 4. Boot Image Format

Word	Description
1	Block size in 16 bits of the program block to be loaded, most significant part
2	Block size in 16 bits of the program block to be loaded, least significant part
3	Word Address where the program block is to be loaded most significant part
4	Address where the program block is to be loaded least significant part
5	Program data
6	Program data
7	Checksum, XOR
8	$\overline{\text{Checksum}}$, $\overline{\text{XOR}}$

The MSC8101 ROM resident bootstrap code is sufficiently flexible to complete the whole download task using the process described here (see **Figure 6**). The application code is converted from an embedded link format (ELF) object into an S-record using an ELF to S-record utility. Then, it is converted to a C array that can be linked into the host bootstrap application. Each line of an S-record is essentially a program block downloaded to the host port.

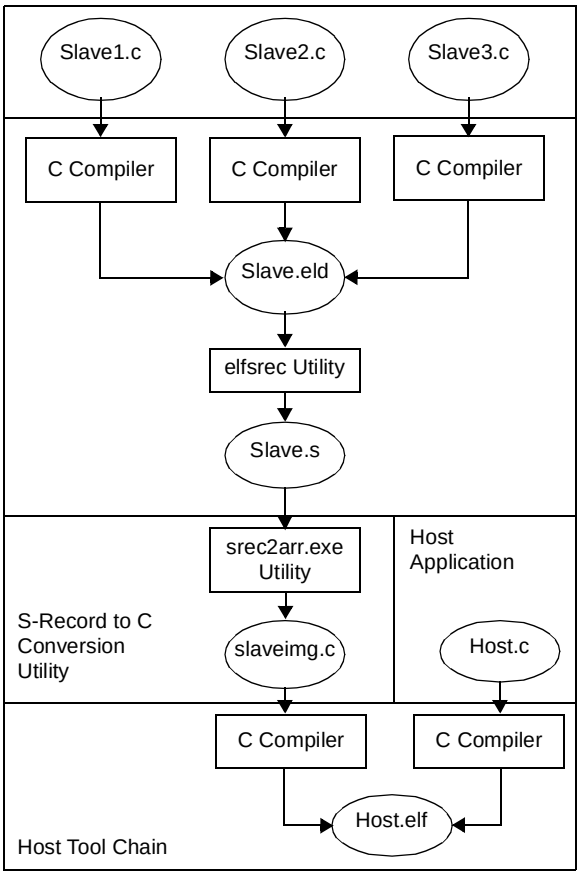

Figure 6. Conversion Process

The slave application files are compiled, assembled, and linked to produce an ELF object (`.eld` file) using the Codewarrior™ tools. This file is then converted to an S-record using the `elfsrec.exe` application provided with the Codewarrior tools. The application is invoked with the following DOS command:

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

The –w specifies that an S2 format S-record is generated, and the resulting output file name is `<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  <filename.s>
```

This command generates an output file named `<slaveimg.c>`. The `srec2arr.exe` program is a small utility that takes the address, length, and data from each line of the S2 record and converts it to a C array in the following format:

- 2 words length (4 bytes)
- 2 words address (4 bytes)
- Number of words described in length as data
- 2 words CRC (4 bytes)
- 2 words $\overline{\text{CRC}}$ (4 Bytes)

Table 5 illustrates 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	CRC	
S2	14	0013C0	3104203A800190C0 90C090C0D8030000	3D	
	Length	**Address**	**Data Words**	**CRC**	$\overline{\text{CRC}}$
	0x0000, 0x000A,	0x0000, 0x13C0,	0x3104,0x203A,0x8001,0x90C0, 0x90C0,0x90C0,0xD803,0x0000,	0x0000,0x0000,	0x0000,0x0000,

The conversion process first strips the S-record type. The number of bytes in length within the S-record is changed to a 32-bit value described in two 16-bit words, and the value is halved to describe 16-bit words rather than bytes. The length described in the C array must leave a remainder of two when it is divided by four to match the boot ROM program requirements. The length for the driver example is 0xA, which meets this requirement. If necessary, additional dummy 16 bit words (0x0000) are added at the end of the real data words, and the length value is incremented appropriately. The address field for an S2 record is three bytes, which are converted to two 16-bit words. The `srec2arr.exe` utility does not support the S1 and S3 record types, although the code can easily be modified to handle them.

The actual data words are then converted to 16-bit quantities. Finally, the CRC and $\overline{\text{CRC}}$ are added. These are not calculated but are simply padded with 0x0000 because the CRC check in the boot ROM can optionally be ignored. The source code (`srec2arr.c`) is provided in Appendix A to add CRC generation, if required.

The last line of the array includes the end block (length 0 address 0), which ends the download process and starts executing the downloaded program. If the start address is not zero, it must be modified to the appropriate value, either in the `srec2arr.c` source code or in the generated C array. The `srec2arr.c` listing is provided in Appendix A for reference and can be modified as needed.

4 MSC8101ADS Test Configuration Example

A simple example of bootstrapping the MSC8101 through the HDI16 is provided to test the hardware interconnection and software driver functionality. The test software initializes the host memory controller so that the slave board HDI16 is mapped as a 16-bit port at location 0x04000000. At start-up, the slave board is in Reset mode waiting for the reset configuration word to be written to it. The host writes the HRCW to the slave and then waits for reset to complete before loading the target application. When the download completes, the slave begins executing the downloaded code, which simply flashes the Red LED (LD9) on the slave MSC8101ADS board. **Table 6** and **Table 7** detail the hardware set-up for the host and slave MSC8101ADS boards.

Note: Connecting two MSC8101ADS boards results in triple buffering that theoretically does not meet the timing specifications or provide aggressive timing profiles required in a typical system where no buffering or a single level of buffering is expected. The timings (and UPM settings) provided therefore do not include buffering delays. However these timings have been verified to work between two MSC8101ADS boards and can be used for test purposes.

Table 6. Host MSC8101ADS Hardware Switch Settings

Switch	Value	Description
SW9		MODCK [1-6] = 000101 Clock mode 40
1	ON	
2	ON	
3	ON	
4	OFF	
5	ON	
6	OFF	
7	OFF	Reset Configuration Word read from FPGA
8	OFF	HPE = 0, Host Port not enabled
SW10		
1	ON	EE1 = 0, Device is in debug mode after reset
2	ON	EE[4-5] = 00 = boot from Bus
3	ON	
4	ON	N/A
SW1		
1	ON	HDSP = 0, Active Low data strobe polarity
2	ON	HDDS = 0, Single data strobe mode
3	ON	H8BIT = 0, 16 bit Mode
4	ON	N/A
U18	20 MHz	**Note:** The oscillator value should be the same for both host and slave boards. 16.384 MHz and 20 MHz have been tested.

Table 7. Slave MSC8101ADS Hardware Switch Settings

Switch	Value	Description
SW9 1 2 3 4 5 6 7 8	 ON ON ON OFF ON OFF OFF ON	 MODCK [1-6] = 000101 Clock mode 40 N/A in Host port mode HPE = 1, Host Port enabled
SW10 1 2 3 4	 OFF ON OFF ON	 SC140 core runs after exiting reset EE[4-5] = 01 = boot from HDI16 Port N/A
SW1 1 2 3 4	 ON OFF ON ON	 HDSP = 0, Active Low data strobe polarity HDDS = 1, Dual data strobe mode H8BIT = 0, 16 bit Mode N/A
U18	20 MHz	**Notes:** The oscillator value should be the same for both host and slave boards. 16.384 MHz and 20 MHz have been tested.

Connect the two MSC8101ADS boards using the wiring description in **Table 8**.

Table 8. Cable Interconnect Wiring Description

Host Board Pin	Signal		Signal	
(P1)C4	TOOLCSb1		$\overline{\text{HCS1}}$	(P4)25
(P1)C5	TOOLCSb2		$\overline{\text{HCS2}}$	(P4)26
(P1)A15	EXPA30		HA3	(P4)24
(P1)A14	EXPA29		HA2	(P4)23
(P1)A13	EXPA28		HA1	(P4)22
(P1)A12	EXPA27		HA0	(P4)21
(P1)D9	EXPGPL2b		HRD/HRW	(P4)29
(P1)D4	EXPWE0b		HWR/$\overline{\text{HDS}}$	(P4)30
(P1)C14	EXPD0		HD0	(P4)3
(P1)C15	EXPD1		HD1	(P4)4
(P1)C16	EXPD2		HD2	(P4)5
(P1)C17	EXPD3		HD3	(P4)6
(P1)C18	EXPD4		HD4	(P4)7
(P1)C19	EXPD5		HD5	(P4)8

Table 8. Cable Interconnect Wiring Description (Continued)

Host Board Pin	Signal		Signal	
(P1)C20	EXPD6		HD6	(P4)9
(P1)C21	EXPD7		HD7	(P4)10
(P1)C22	EXPD8		HD8	(P4)11
(P1)C23	EXPD9		HD9	(P4)12
(P1)C24	EXPD10		HD10	(P4)13
(P1)C25	EXPD11		HD11	(P4)14
(P1)C26	EXPD12		HD12	(P4)15
(P1)C27	EXPD13		HD13	(P4)16
(P1)C28	EXPD14		HD14	(P4)17
(P1)C29	EXPD15		HD15	(P4)18
(P1)C31	$\overline{\text{IRQ5}}$		$\overline{\text{HRRQ}}$/$\overline{\text{HACK}}$	(P4)27
(P1)C30	$\overline{\text{IRQ4}}$		$\overline{\text{HREQ}}$/$\overline{\text{HTRQ}}$	(P4)28

The slave image C file is created as follows:

1. Load the slave application project into the CodeWarrior tools by double clicking on the `led_flash.mcp` file in `...\hdi16_appnote\Led_flash\build\`.
2. Build the project to create the `.eld` file, which is named `starcore.eld`. Close the project.
3. Execute the elfsrec utility in a DOS window using the command line: `elfsrec -w starcore.eld`.
4. When the `starcore.eld.s` file is created, rename it to `starcore.s`.
5. Execute the utility srec2arr.exe on the starcore.s file using the command line: srec2arr starcore.s Note that the srec2arr.exe file is located in `...\hdi16_appnote\srec_to_array_hdi16\`.
6. When the `slaveimg.c` file is created, copy it to the location `...\hdi16_appnote\host\source`.
7. Open the project host.mcp in `...\hdi16_appnote\host\build`, add `slaveimg.c`, and compile the project.

Download code onto the MSC8101ADS board via the EOnCE debug interface, as follows:

1. Initialize host and slave switch settings to those indicated in **Table 6** and **Table 7**.
2. Connect the two boards as shown in **Table 8**.
3. Switch on both boards and press PRESET (SW8) on both boards.
4. Load the software (CodeWarrior project `host.mcp`) onto the host MSC8101ADS using the debug port.

   The slave board LEDs LD8 and LD9 should be ON.
5. Start the program

Slave board LEDs LD8 and LD9 should go out and then LD9 should flash intermittently.

5 Related Documents

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

[2] *MSC8101 Reference Manual* (MSC8101RM/D).

[3] *MSC8101ADS User's Manual* (MSC8101ADSUM).

[4] *MSC8101 Technical Data* sheet (MSC8101).

APPENDIXES

A SREC_TO_ARRAY.C Listing

```
                    #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(int argc, char *argv[])
                    {
                    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];
```

```
uliEndFlag = 0;
/* Ignore the application name            */
/* argv[0] contains application name      */
/* decrease arg count, increase arg value */
argv++;

strcpy(szCurrentString,*argv);            /* Parse parameters */
strcpy(szTempString,&szCurrentString[0]); /* get loader file name */

if((pfiIn = fopen(szTempString,"rb")) ==NULL)
{
       printf("error \n");
       exit(1);
}
if((pfiOut = fopen(".\\slaveimg.c","wb")) ==NULL)
{
       printf("error2 \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 UWord16 ausiImage1[]={");
```

```
/* Get line header S? */
ucCh0 = getc(pfiIn);
ucCh1 = getc(pfiIn);

while(uliEndFlag==0)
{
        putc('0',pfiOut);
        putc('x',pfiOut);
        putc('0',pfiOut);
        putc('0',pfiOut);
        putc('0',pfiOut);
        putc('0',pfiOut);
        putc(',',pfiOut);
        putc('0',pfiOut);
        putc('x',pfiOut);
        putc('0',pfiOut);
        putc('0',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-4;       /* take away ucLength and crc       */
        uliRemainder=(uliSize+4)%8;  /* add 4 bytes for bootload crc and */
                              /* see how short of div by 8 it is  */

        if (uliRemainder != 0)
        {
                uliRemainder = (8-uliRemainder);
        }

        uliSize = (uliSize+uliRemainder)/2;

        if (uliSize <= 0xF)
        {
                fprintf(pfiOut, "0%X", uliSize);
        }
        else
        {
                fprintf(pfiOut, "%X", uliSize);
        }

        putc(',',pfiOut);
        putc('0',pfiOut);
```

```
putc('x',pfiOut);
putc('0',pfiOut);
putc('0',pfiOut);
aucTemp[0] = getc(pfiIn);
putc(aucTemp[0],pfiOut);
aucTemp[0] = getc(pfiIn);
putc(aucTemp[0],pfiOut);
putc(',',pfiOut);
putc('0',pfiOut);
putc('x',pfiOut);

for (ulij=0; ulij<4; ulij++)
{
        aucTemp[0] = getc(pfiIn);
        putc(aucTemp[0],pfiOut);
}
/* Get data words */
for (ulii=0; ulii<((uliCounter-4)/2); ulii++)
{
        putc(',',pfiOut);
        putc('0',pfiOut);
        putc('x',pfiOut);

        for (ulij=0; ulij<4; ulij++)
        {
                aucTemp[0] = getc(pfiIn);
                putc(aucTemp[0],pfiOut);
        }
}
/* if ucLength of block is not divisible by 8 bytes add padding words. */

if (uliRemainder != 0)
{
        printf("%d ",uliRemainder);
        for (ulij=0; ulij<((uliRemainder)/2) ; ulij++)
        {
                putc(',',pfiOut);
                putc('0',pfiOut);
                putc('x',pfiOut);
                putc('0',pfiOut);
                putc('0',pfiOut);
                putc('0',pfiOut);
                putc('0',pfiOut);
        }
}

/* read checksum and carriage return */
aucTemp[0] = getc(pfiIn);
aucTemp[0] = getc(pfiIn);
aucTemp[0] = getc(pfiIn);
aucTemp[0] = getc(pfiIn);

/* add two 16 bit word for checksum */
putc(',',pfiOut);
putc('0',pfiOut);
putc('x',pfiOut);
putc('0',pfiOut);
putc('0',pfiOut);
putc('0',pfiOut);
putc('0',pfiOut);
putc(',',pfiOut);
putc('0',pfiOut);
putc('x',pfiOut);
```

```
        putc('0',pfiOut);
        putc('0',pfiOut);
        putc('0',pfiOut);
        putc('0',pfiOut);

        /* Get line header S? */
        ucCh0 = getc(pfiIn);
        ucCh1 = getc(pfiIn);

        if((ucCh0=='S')&&(ucCh1=='8'))
        {
                uliEndFlag = 1;

                /* Add End Block */
                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.: AN2311
Rev. 1
11/2005