Application Note

AN2442/D
Rev. 0, 1/2003

Booting the MSC8102 Device Through TDM

by Barbara Johnson

CONTENTS

The Motorola StarCore® MSC8102 device supports booting from the time-division multiplexing (TDM) interface in which a boot master downloads program and data to one or more MSC8102 devices. This application note discusses the steps in the bootloader program:

- detect the selected boot mode
- synchronize to the boot master TDM signals
- initialize the TDM
- begin the block transfer protocol.

This document also provides example boot master code that uses the MSC8101 to boot the MSC8102 from the TDM module.

1 Detecting Boot Mode

The MSC8102 operating mode is configured by the external boot mode BM[0–2] pins, which are sampled on the rising edge of $\overline{\text{PORESET}}$. As part of the reset configuration sequence, the logic states of these pins are copied to the BM field in the SIU Module Configuration Register (SIUMCR). The bootloader program checks this field to determine the boot mode. If SIUMCR[7–9]:BM = 010, the MSC8102 device is configured to boot through the TDM interface. The bootloader program jumps to the TDM boot section in the ROM.

2 Synchronizing the Boot Signals

An MSC8102 TDM boot system consists of a boot master and one or more slave MSC8102 devices. The boot master generates the clock and frame sync signals for both the receive and transmit. The MSC8102 device must determine the frame parameters so that it can initialize the number of channels and the size of each channel in the TDM. The MSC8102 TDM adaptation machine synchronizes with the boot master frame sync signal to determine these frame parameters. **Figure 1** shows an example TDM boot system.

The bootloader program enables the TDM adaptation machine and waits for a receive frame sync to arrive. It checks the number of bits between the last back-to-back receive frame sync events. This check is repeated until the number of bits is stable 16 times. The same frame sync detection and procedure occurs for the transmit frame sync. The adaptation machine is disabled after the frames are determined to be stable. **Figure 2** shows the flow diagram of the synchronization procedure.

A TDM frame has an even number of channels, and the channel size can be an even number of bits up to 256 bits. For example:

- When there are 8 bits in a frame, each of the two channels is 4 bits wide.
- When there are 16 bits in a frame, each of the two channels is 8 bits wide.

- When there are 32 bits in a frame, each of the four channels is 8 bits wide .
- When there are 193 bits in a frame, each of the 24 channels is 8 bits wide. This frame is a T1 frame that consists of 192 data bits in addition to a framing bit.

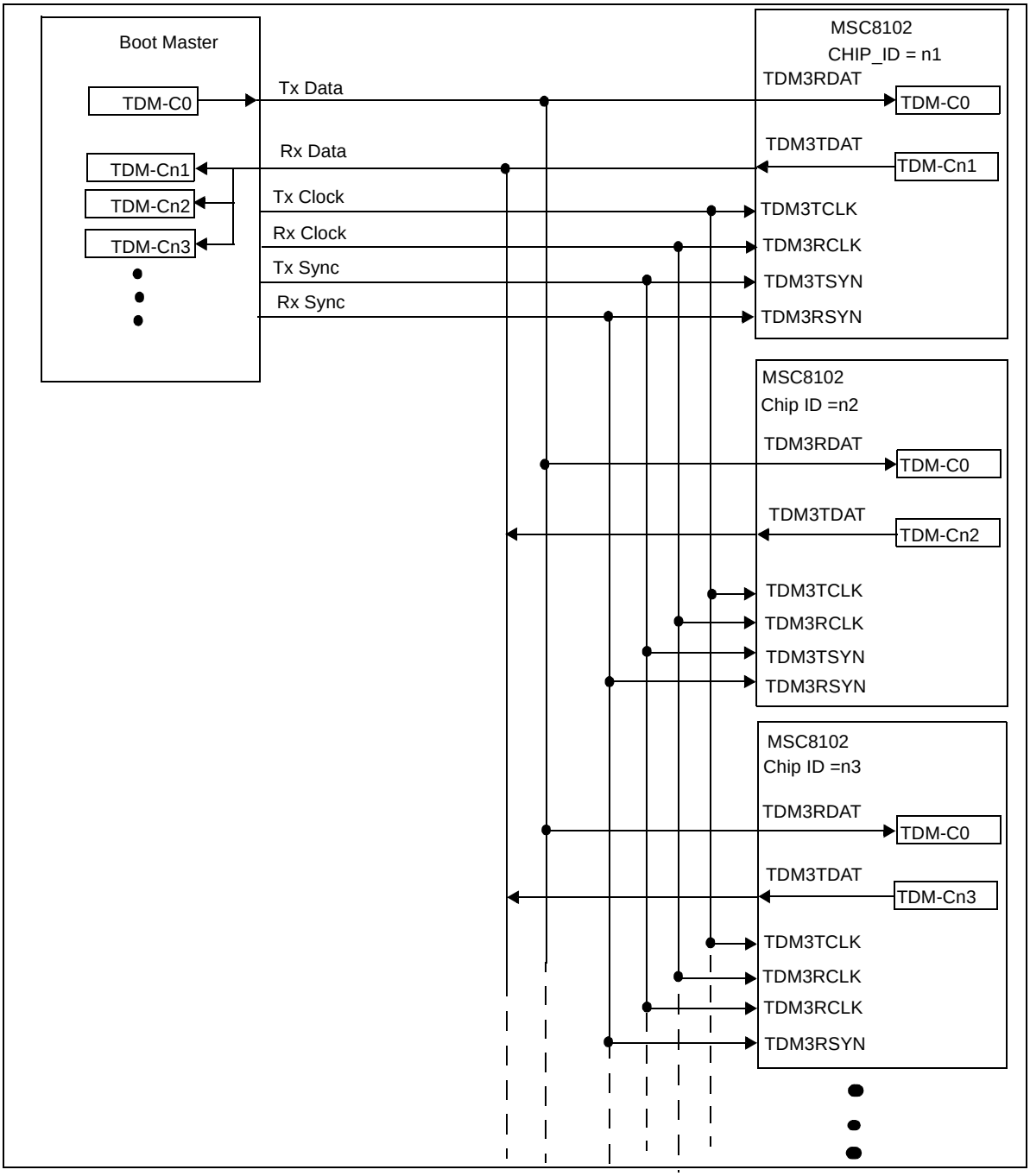

Figure 1. TDM Boot System

The TDM3 Receive Frame Parameters Register (TDM3RFP) and the TDM3 Transmit Frame Parameters Register (TDM3TFP) are configured according to the detected frame size. The TDM3RFP[8–15]:RNCF and TDM3TFP[8–15]:TNCF are updated with the number of channels in the TDM frame. The TDM3RFP[26–29]:RCS and TDM3TFP[26–29]:TCS fields are updated with the channel size.

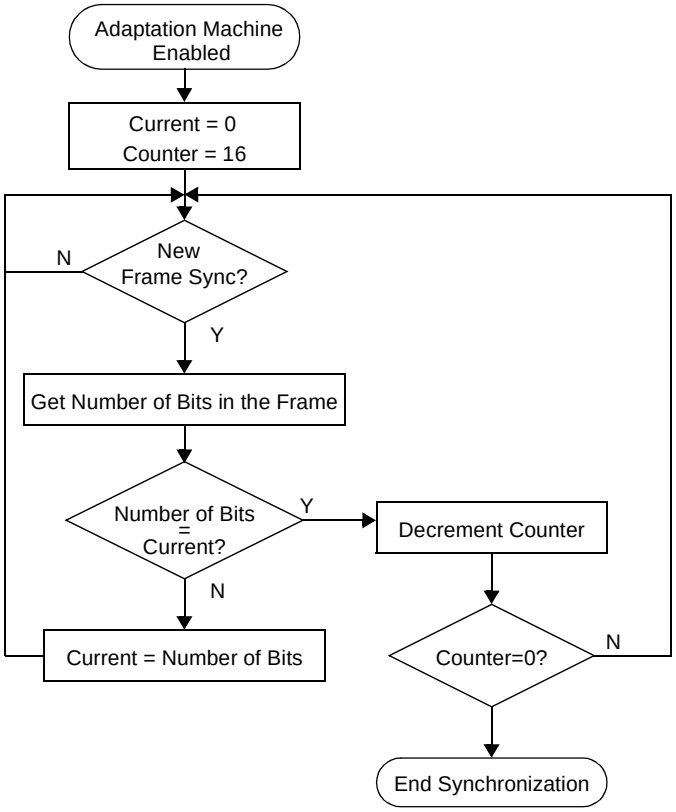

Figure 2. Adaptation Machine Flow Diagram

3 Initializing the TDM

After the TDM ynchronizes with the boot master TDM clock and frame sync signals, the MSC8102 device can initialize the TDM interface. The bootloader program configures only TDM3. TDM[0–2] are not used. Since TDM3 is the only TDM employed in the boot process, signal sharing with other TDM modules is disabled. The receive and transmit sections are independent, with separate clock and frame sync. There is one receive data link and one transmit data link, as shown in **Table 1**.

Table 1. Receive and Transmit Sections

Transmit	Receive
Transmit frame sync, TDM3TSYN	Receive frame sync, TDM3RSYN
Transmit clock, TDM3TCLK	Receive clock, TDM3RCLK
Transmit data, TDM3TDAT	Receive data, TDM3RDAT

The receive frame sync is active on logic '1.' Both the receive frame sync and the receive data are sampled on the rising edge of the receive clock. The receive frame sync occurs one cycle before the first bit of the receive data is sampled. **Figure 3** shows the relative timing of the receive section of the TDM. The transmit frame sync is active on logic '1.' The transmit frame sync is sampled on the rising edge of the transmit clock. The transmit data is driven out on the falling edge of the clock. The transmit frame sync occurs one-half cycle before the first bit of the transmit data is driven out. **Figure 4** shows the relative timing of the transmit section of the TDM.

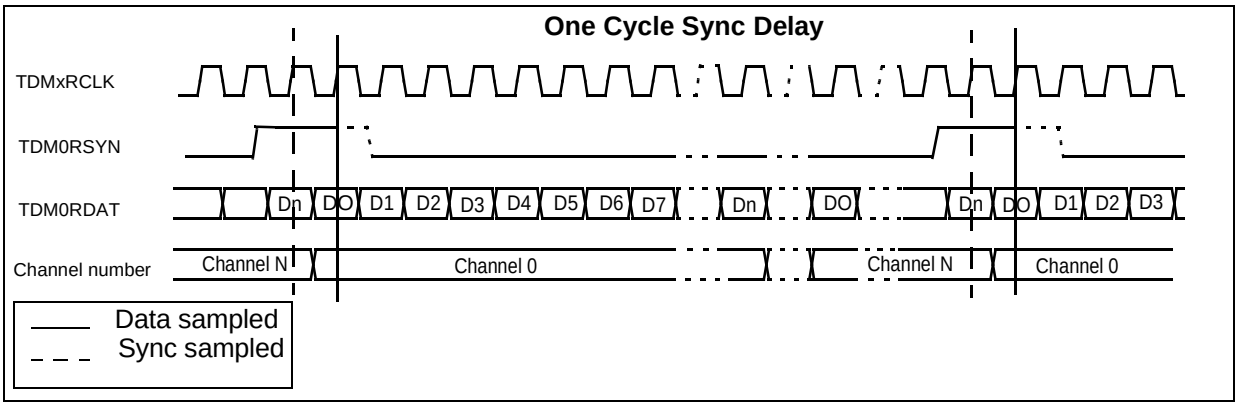

Figure 3. Receive Frame

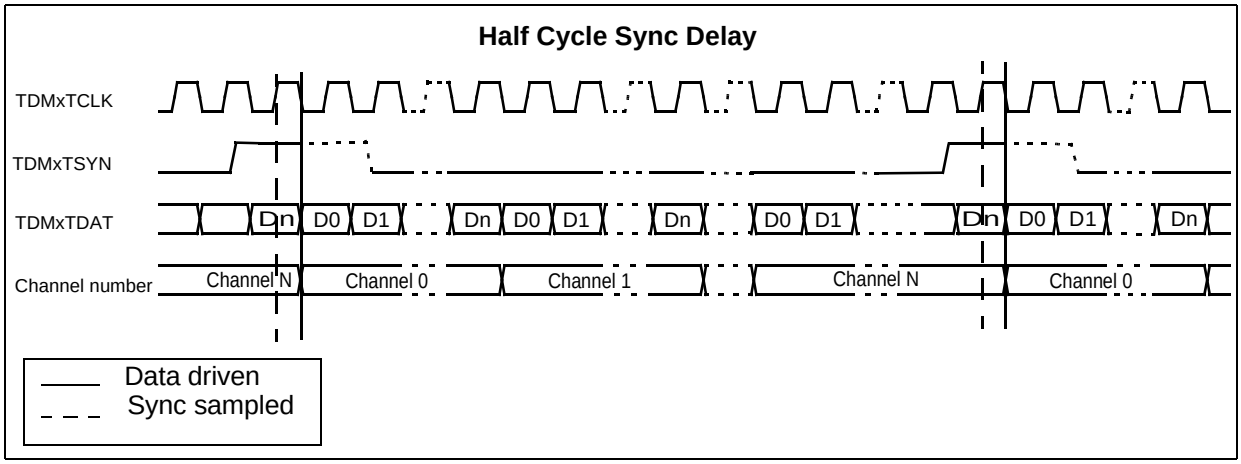

Figure 4. Transmit Frame

The boot master transmits messages to one or more slave MSC8102 devices on TDM channel 0. Each slave MSC8102 device transmits back on a different TDM channel with a value that equals its CHIP_ID value. For example, the boot master transmits a block of data on channel 0. The MSC8102 device with CHIP_ID = 1 transmits back an acknowledge message on channel 1. The CHIP_ID[0–3] pins are sampled on the rising edge of $\overline{\text{PORESET}}$. Since the boot master transmits messages on channel 0, the MSC8102 receive channel 0 is enabled. The other channels remain disabled. Also, the MSC8102 transmit channel CHIP_ID is enabled since this channel is used to transmit messages back to the boot master device. In this example, the MSC8102 is configured to have a CHIP_ID = 3, so transmit channel 3 is active. These channels are activated in the TDM3 Receive Channel Parameter Register (TDM3RCPR_n) and the TDM3 Transmit Channel Parameter Register (TDM3TCPR_n).

4 Perform the Block Transfer of Code and Data

The boot master device writes blocks of code and data into the memory of one or more MSC8102 devices. The MSC8102 implements a block transfer protocol to ensure that the boot master sends the message to the target MSC8102 device and writes the code to the specified memory location. The block transfer protocol also ensures correct message transmission by performing cyclic redundancy check (CRC). The block transfer protocol implements the exchange of data and control code using two types of messages: a block transfer message (BTM) and a block transfer acknowledge (BTAM) message.

4.1 Block Transfer Message

A BTM contains blocks of code or data that are written to the MSC8102 memory (see **Table 2**).

Table 2. Block Transfer Message Structure

Block Transfer Message			Description
Size (Bytes)	Name	Value	
4	PRM	0x44332211	**Preamble**. Indicates the start of the message. The first byte sent is 0x11.
1	DCID	CHIP_ID or 0xFF	**Destination CHIP_ID**. Identifies the target MSC8102 slave device to accept this message. DCID = 0xFF broadcasts the BTM to all MSC8102 devices connected to the boot master.
1	SN	0x00 to 0xFF	**Sequence Number** modulo 256. Indicates the sequence number of the BTM. The boot master generates a unique SN for every BTM sent in a TDM boot session.
1	EB	0x00 or 0xFF	**End Block**. EB = 0xFF indicates the last BTM. After the last message, all SC140 cores jump to address 0x0 of their M1 memory.
3	PLDS	0 to 2^{24}	**Payload Size**. Indicates the size in bytes. PLDS must be divisible by 2.
4	DA		**Destination Address**. Indicates the destination address for the payload in the slave MSC8102 memory. For valid addresses, refer to the SC140 core internal memory map chapter in the *MSC8102 Reference Manual*. Addresses 0x01076E00–0x01076FFF are reserved and cannot be used.
2	HCRC		**CRC-16 of PRM, DCID, SN, EB, PLDS and DA**. The HCRC field is a 16-bit CRC represented by $x^{16} + x^{15} + x^2 + 1$.
Up to 2^{24}	PLD		**Payload**. Specifies the data to be written to the destination address (DA).
2	CRC		**CRC-16 of PLD**. The CRC field is a 16-bit CRC represented by $x^{16} + x^{15} + x^2 + 1$.

The code shown on the left side of **Figure 5** is broadcast to all MSC8102 devices connected to the boot master via TDM. The code writes 1024 bytes of 0xAA and 512 bytes of 0x99 to memory locations 0x1010000 and 0x2010200, respectively. This code must be formatted as a BTM so that the boot master can transmit it to the slave MSC8102 device via TDM. The BTMs of each set of code are shown on the right side of **Figure 5**. The first four bytes consist of the preamble to indicate the start of the message. Notice that 0x11 is the first byte sent. The next byte contains a value of 0xFF to indicate that the message is being broadcast to all MSC8102 devices. The next byte contains a value of 0x00 to indicate that this is the first message in the sequence. The next byte contains a value of 0x00 to indicate that this message is not the last message in the sequence. The next three bytes contain a value of 0x000400 to indicate the size of the payload (1024 bytes) that convert to a size of 0x400 bytes. The next four bytes indicate the

destination address of 0x01010000. Again, the least significant byte of the address is sent first. The next two bytes are the result of the CRC-16 calculation of the header fields, which is 0xA199. The next 1024 bytes are the payload, which consists of 0xAA. Finally, the last two bytes contain the result of the CRC-16 calculation of the payload, which is 0x8781. Note that when the CRC-16 of the header is received with no error but the CRC-16 of the payload is received with error, corrupt data is written to the MSC8102 slave memory. The second BTM with sequence number 0x01 is generated the same way. This code writes 512 bytes of 0x99 to the destination address 0x02010200. The other five BTMs are for the MSC8102 with CHIP_ID = 3. **Figure 6** shows the code that is written to the MSC8102 memory. The BTM sequence number, 0x02, is also shown on the right

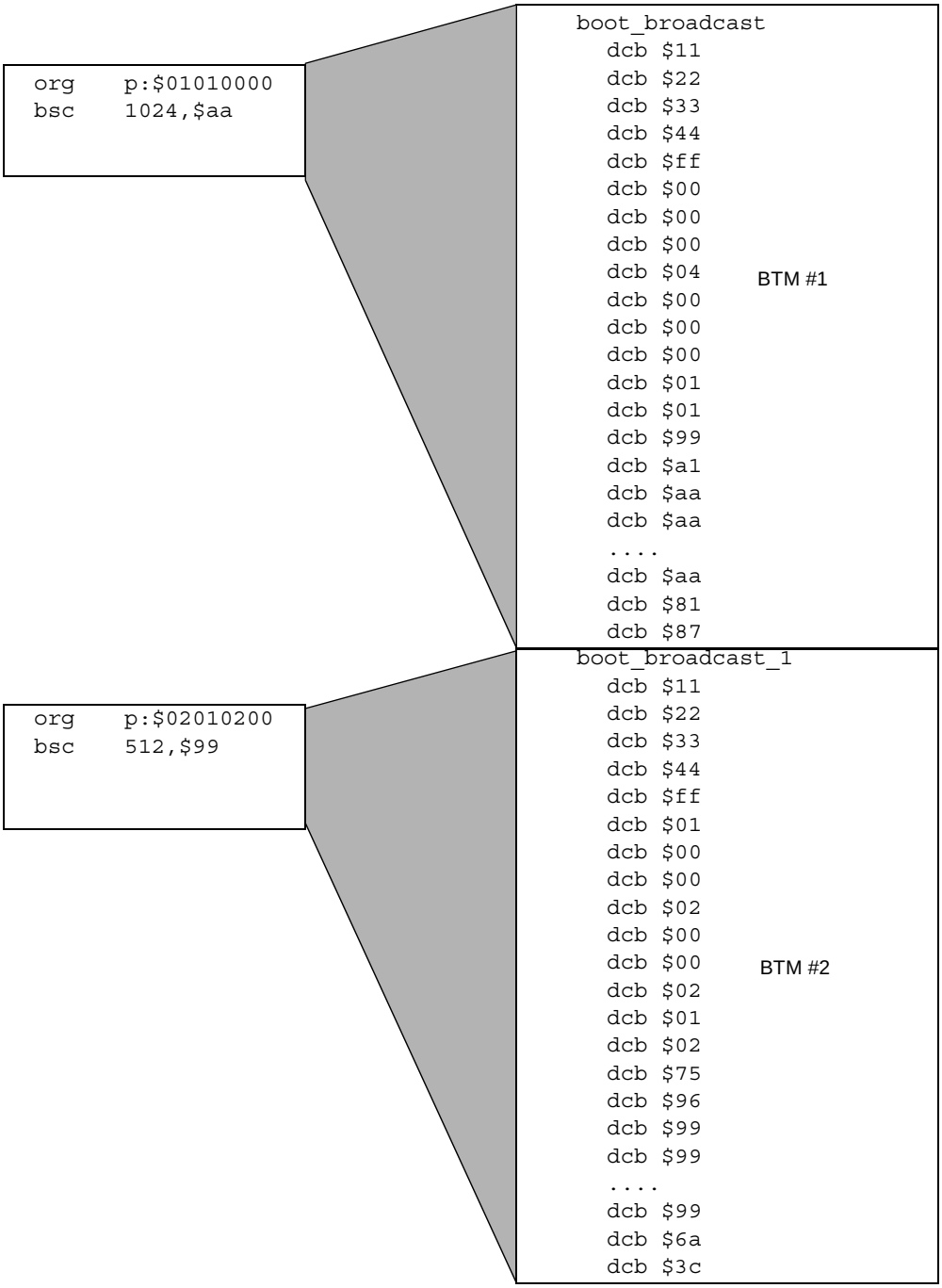

Figure 5. Broadcast Messages

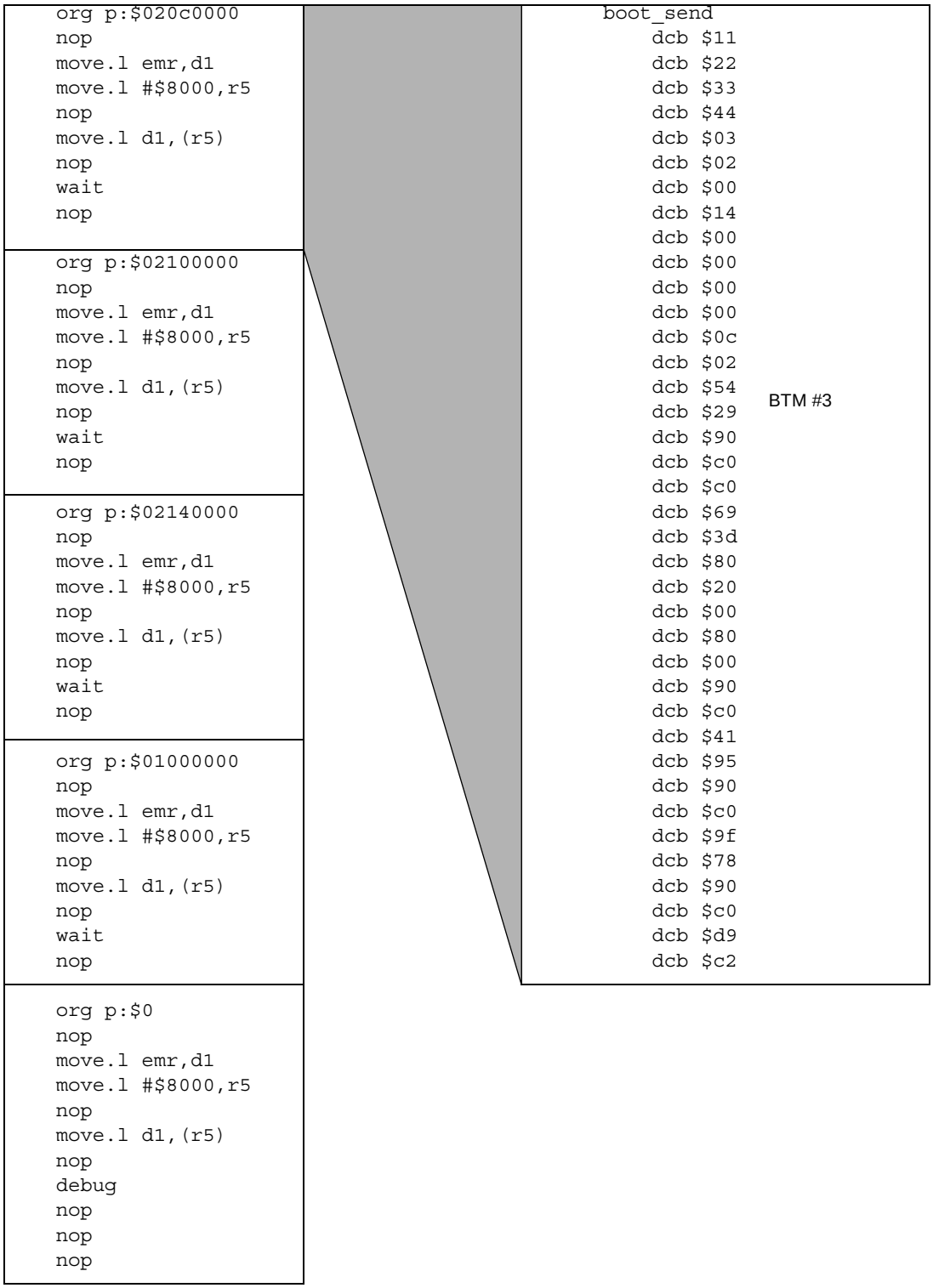

Figure 6. Messages for CHIP_ID = 3

4.2 Block Transfer Acknowledge Message

The BTAM is the message that the slave MSC8102 sends back to the boot master. It contains information about the previously received BTM (see **Table 3**).

Table 3. Block Transfer Acknowledge Message Structure

Block Transfer Message			Description
Size (Bytes)	Name	Value	
2	APRM	0x6655	**Preamble**. Indicates the start of the acknowledge message. First byte sent is 0x55.
1	SCID	CHIP_ID	**Source CHIP_ID**. Identifies the target MSC8102 slave device that accepted the BTM.
1	RN	0x00 to 0xFF	**Receive Sequence Number** modulo 256. Indicates the expected sequence number to receive next.
2	ACRC		**CRC-16 of APRM, SCID and RN**. EB = 0xFF indicates the last BTM. After the last message, all SC140 cores jump to address 0x0 of their M1 memory.

The MSC8102 slave device implements a logic layer protocol to receive data and ensure correct transmission from the boot master device. **Figure 7** shows the flow diagram of the logic layer protocol, which works as follows:

1. The MSC8102 synchronizes with the preamble (PRM) field of the BTM.
2. If there is an error in the CRC-16 of the header (HCRC), then return to step 1. Otherwise, continue to the next step.
3. If the destination CHIP_ID (DCID) is equal to the MSC8102 CHIP_ID or if it is equal to 0xFF for a broadcast message, then the payload data (PLD) is written to the destination address (DA). Otherwise, return to step 1.
4. If the CRC-16 of the payload (CRC) contains no error, the MSC8102 device sends a BTAM with its CHIP_ID in the SCID field and the receive sequence number (RN). Otherwise, continue to the next step.
5. If the expected sequence number of the next message (RN) is equal to the sequence number of the current message (SN), then the MSC8102 device sends a BTAM with an RN and SCID with its CHIP_ID. Otherwise, the RN is incremented and the MSC8102 device sends a BTAM with the updated RN and SCID with its CHIP_ID.
6. If the end block (EB) flag has a value of 0xFF, the TDM boot session ends. Otherwise, return to step 1 to continue with the next BTM.

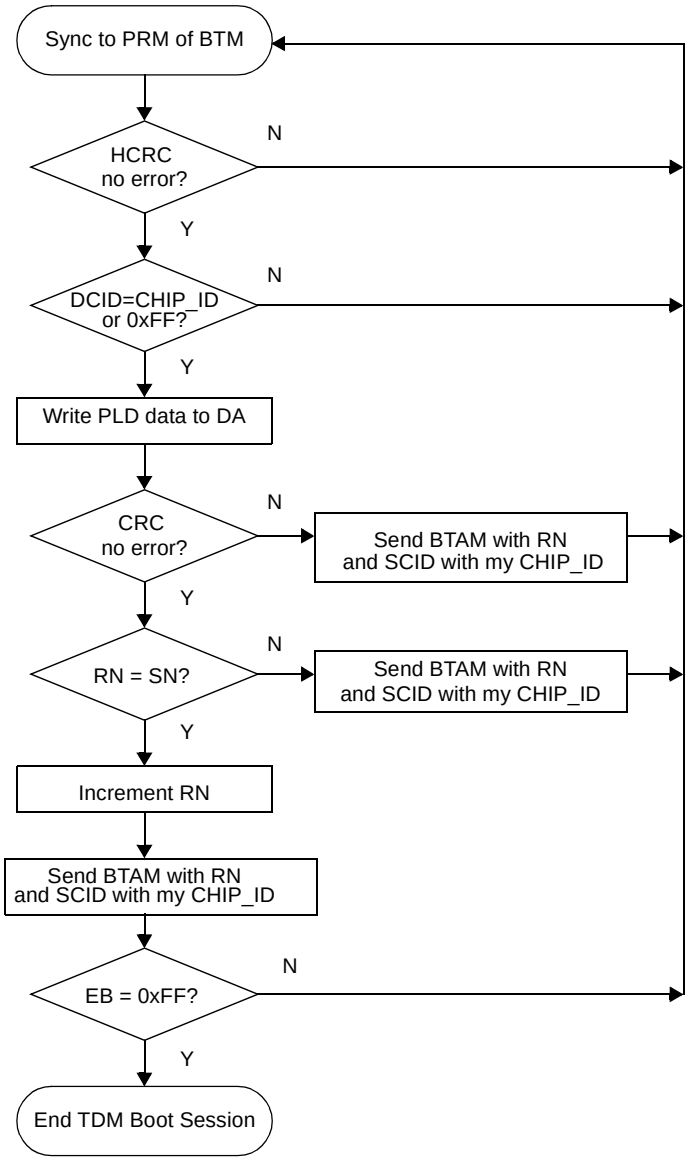

Figure 7. MSC8102 Logic Layer Protocol

Figure 8 shows the BTAMs sent from the MSC8102 to the boot master device. The MSC8102 device with a CHIP_ID value of 3 transmits these BTAMs on channel 3. The boot master receives the BTAMs in the receive buffer for channel 3. The first two bytes of each BTAM are the preamble, which has a fixed value of 0x5566. The next byte is the source CHIP_ID (SCID), which is 0x03 to indicate the device CHIP_ID. The next byte is the receive sequence number (RN). After the MSC8102 device receives the first BTM from the boot master and determines that the CRC is error-free, it sends the first BTAM with RN = 0x01 and SCID = 0x03. The RN field indicates the sequence number of the next BTM the MSC8102 device is to receive. The RN value is initialized to zero at the start of the TDM boot session. In this case, the next BTM has SN = 0x01. The last two bytes of the BTAM represents the CRC-16 of the preamble, source CHIP_ID, and receive sequence number. Since the boot master has sent seven BTMs (SN = 0x00 through 0x06), the last BTAM contains RN = 0x07.

```
5566030130E3
5566030270E2
55660303B122
55660304F0E0
556603053120
556603067121
55660307B0E1
```

Figure 8. Acknowledge Messages

The boot master device works in either the handshake mode or the non-handshake mode. The handshake mode implements a stop and wait technique in which the boot master waits for the BTAM message after sending the BTM. If the boot master does not receive a BTAM within a 32-frame time period, it resends the BTM and waits for a BTAM again. This method should be used to ensure proper operation. In the non-handshake mode, the boot master device does not wait for the BTAM messages. Instead, the BTM messages are sent in sequence without any wait time. The MSC8102 device returns the BTAM messages, but their correctness is not guaranteed.

5 Example Boot Master Code

This section illustrates how to boot the MSC8102 device from the TDM module using the MSC8102ADS. The on-board MSC8101 is configured as the boot master, and the slave MSC8102 device CHIP_ID is set to 3. The MSC8101 Multichannel Controller 1 (MCC1) performs the TDM communication. The MSC8101 and MSC8102 are clocked with 66 MHz and 41.6 MHz oscillators, respectively. Note that other devices with a TDM interface can be used as the boot master.

5.1 Initialize the Boot Mode

The boot mode selection and initialization proceeds as follows:

1. Assign the BTMD[0–2] field a value of 0b010 to boot from the TDM by writing a value of 0xFA to Board Control/Status Register 3 (BCSR3).

   The values written to BCSR3 BTMD[0–2] drive the boot mode BM[0–2] pins during the power-on-reset sequence.

2. Select the reset configuration mode and configuration source by writing a value of 0x07 to Board Control/Status Register 2 (BCSR2).

   This step clears the RSTCNF and CNFGS fields so that the Hard Reset Configuration Word (HRCW) is written through the system bus. It also sets the MSC8102 as the configuration master. In this example, the HRCW is programmed from the on-board Flash EPROM. The values written to BCSR2[RSTCNF] and BCSR2[CNFGS] drive the $\overline{\text{RSTCONF}}$ and CNFGS pins during the power-on-reset sequence.

3. Initiate the power-on-reset sequence by writing a value of 0x7F to the Board Control/Status Register 1 (BCSR1).

   This step clears the RECONF field to assert the $\overline{\text{PORESET}}$ signal. Writing 0xFF to BCSR1 negates the $\overline{\text{PORESET}}$ signal.

After the power-on-reset sequence, the MSC8102 executes the TDM bootloader program in ROM. The boot master is now ready to send messages to the slave MSC8102 device. **Example 1** shows how to initialize the boot mode. The example code in this application note uses the macros write_l, write_w, and write_b to move long word, word, and byte-sized data to a register.

Example 1. Initialize the Boot Mode

```
; ----------------------------------------
; Initialize MSC8102 boot mode
;
; MSC8101 - 66 MHz oscillator
; MSC8102 - 41.6 MHz oscillator
;
; MSC8102ADS switch settings
; SW4[1:4] = OFF OFF ON OFF
; SW6[1:8] = ON OFF OFF ON ON ON OFF OFF
; JP5 open
; ----------------------------------------
; Set MSC8102 boot from TDM
write_l #$FA000000,BCSR3

; Get HCW from system bus
write_l #$07000000,BCSR2

; Assert /PORESET
write_l #$7F000000,BCSR1

; Negate /PORESET
write_l #$FF000000,BCSR1

; Wait ~15ms
move.l #$C0000,d0
jsr    delay
```

5.2 Initialize the TDM Pins

The MSC8101 boot master generates the TDM clock and frame sync signals that are input to the MSC8102. **Figure 9** shows the pin connections. The output of the baud-rate generator BRG5O connects to the MSC8102 receive and transmit clock pins, TDM3RCLK and TDM3TCLK. It also connects to the MSC8101 receive and transmit clock pins L1RCLK and L1TCLK. The CMX Clock Route Register (CMXSI1CR) selects CLK1 and CLK2 as the receive and transmit clocks. BRG5O is also the source of the frame sync signals. It connects to the timer input TIN2 pin. The timer divides the TIN2 input frequency to generate the frame sync pulse on TOUT2 for every 32 clock cycles on TIN2. TOUT2 connects to the MSC8102 receive and transmit frame sync pins, TDM3RSYN and TDM3TSYN. TOUT2 also connects to the MSC8101 frame sync pin, L1RSYN.

Finally, the MSC8101 receive and transmit data pins L1TCLK and L1RCLK connect to the MSC8102 receive and transmit data pins TDM3RDAT and TDM3TDAT. **Example 2** shows the pin configuration code.

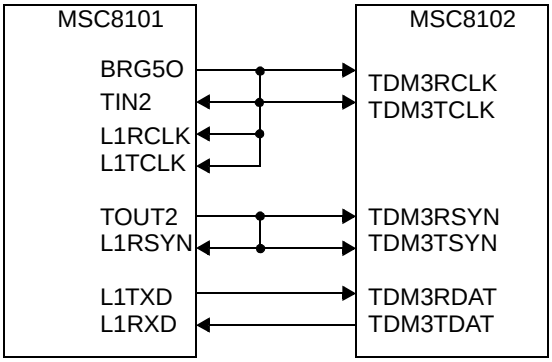

Figure 9. TDM Pin Connection

Example 2. Configure TDM Pins

```
; ------------------------------------------
; Pin Configuration
; PC31 - CLK1
; PC30 - CLK2
; PC29 - TIN2
; PC28 - TOUT2_B
; PC27 - BRG5O
; ------------------------------------------
; Configure TDM pins
write_l  #$03c00000,PPARA
write_l  #$03c00000,PSORA
write_l  #$00000000,PDIRA

write_l  #$00000fff,PPARB
write_l  #$00000fff,PSORB
write_l  #$00000000,PDIRB

; Configure Timer and BRG pins
write_l  #$0000001f,PPARC
write_l  #$00000008,PSORC
write_l  #$00000018,PDIRC

; ------------------------------------------
; Configure the clock and frame sync
; ------------------------------------------
; TDM clock ~ 446kHz
write_l #$00010048,BRGC5

; Generate TDM frame sync (32 clocks per frame)
write_l #$10000000,TGCR1
write_w #$000E,TMR2
write_w #$001f,TRR2

; ------------------------------------------
; TDMA1 Rx clock - CLK1
; TDMA1 Tx clock - CLK2
; ------------------------------------------
write_b #$00,CMXSI1CR
```

5.3 Initialize the Transmit and Receive Buffers

The example discussed in this section uses one buffer for each channel so that there are four transmit buffers and four receive buffers. Each buffer holds 0x1000 bytes of data. These buffers are located at the addresses shown in **Table 4**.

Table 4. Buffer Addresses

Address	Channel	Address	Channel
0x10000	Transmit buffer channel 0	0x20000	Receive buffer channel 0
0x11000	Transmit buffer channel 1	0x21000	Receive buffer channel 1
0x12000	Transmit buffer channel 2	0x22000	Receive buffer channel 2
0x13000	Transmit buffer channel 3	0x23000	Receive buffer channel 3

Since the boot master transmits messages to the slave MSC8102 devices on channel 0, the transmit buffer for channel 0 is initialized with the message to send. Transmit buffers for channels 1–3 and receive buffers for channels 0–3 and the interrupt queues are cleared as shown in **Example 3**.

Example 3. Set up the Tx and Rx Buffers

```
; ----------------------------------------
; Clear Tx buffer channels 1-3 and
; Rx buffers channels 0-3
; Tx buffer channel 0 is initialized with
; message to send
; Clear Tx and Rx interrupt queue tables
; ----------------------------------------
    move.l #$00011000,r0
    move.l #$00050000,r1
    move.l #$0,d0
    move.l #$0,d1
    jsr set_block
set_block
     move.2l d0:d1,(r0)+
     cmpeqa r0,r1
     bf set_block
     rts
```

Data to be sent to the MSC8102 device is written into transmit buffer channel 0 as shown in **Example 4**. The first 0x1000 bytes of the buffer are filled with a value of 0x2211. Since this data does not contain the required 0x44332211 preamble, the MSC8102 does not process this data as valid data. The included file `boot_broadcast.iasm` contains the preamble and message that is broadcast to all MSC8102 devices in the boot system. The contents of this file are shown in **Figure 5** on page 6. The other included file `boot_send_chipid_3.iasm` contains the preamble and message that is sent only to the MSC8102 device with CHIP_ID = 3. The contents of this file are shown in **Figure 6** on page 7.

Example 4. Initialize the Tx Buffer Channel 0

```
;----------------------------------------
; Initialize the Tx buffer
;----------------------------------------
org p:$10000
    bsc 1000,$2211
    include "boot_broadcast.iasm"
    include "boot_send_chipid_3.iasm"
    bsc       4096,$00
```

5.4 Configure the Buffer Descriptors

The buffer descriptors contain the control and status information for the buffers, buffer location, and buffer data length. The code to configure the buffer descriptors is shown in **Example 5**. This code sets the receive buffers as empty and sets the transmit buffers as ready for transmission. It also writes the buffer length of 0x1000 bytes and memory locations of each buffer into the buffer descriptors.

Example 5. Configure the Buffer Descriptors

```
; ----------------------------------------
; Configure Rx Buffer Descriptors
; Rx Buffer Channel 0 = 0x02020000
; Rx Buffer Channel 1 = 0x02021000
; Rx Buffer Channel 2 = 0x02022000
; Rx Buffer Channel 3 = 0x02023000
; ----------------------------------------
    move.l#$0,r0
    move.l#RNUM_CH,d0
    move.l #RBUFF_DL,d1
    move.l #RBUFF_BASE_ADD,d2
    move.l #$b000,d3

_loop_rbd
; Set Empty, Wrap bits in RxBD
    move.w d3,(r0)+

    ; Set data length
    move.w d1,(r0)+

    ; Set Rx buffer base address
    move.l d2,(r0)+
    ; Repeat for other channels
    add d1,d2,d2
    deceq d0
    bf _loop_rbd
; ----------------------------------------
; Configure Tx Buffer Descriptors
; Tx Buffer 0 = 0x02010000
; Tx Buffer 1 = 0x02011000
; Tx Buffer 2 = 0x02012000
; Tx Buffer 3 = 0x02013000
; ----------------------------------------
    move.l #TNUM_CH,d0
    move.l #TBUFF_DL,d1
    move.l #TBUFF_BASE_ADD,d2

_loop_tbd
; Set Ready, Wrap bits in TxBD
    move.w d3,(r0)+

    ; Set data length
    move.w d1,(r0)+

    ; Set Tx buffer base address
    move.l d2,(r0)+

    ; Repeat for other channels
    add d1,d2,d2
    deceq d0
    bf        _loop_tbd
```

5.5 Initialize the MCC Parameters

MCC initialization consists of several steps as described here. The initialization code is shown in **Example 6**.

1. Initialize the MCC parameters to their reset state.

   This step issues the INIT RX AND TX PARAMS command to the Communications Processor Command Register (CPCR) to initialize the MCC1 parameters to their reset values. Since the command initializes 32 consecutive channels, the command is issued only once to initialize the four channels used in this example.

2. Initialize the MCC global parameters.

   This step initializes the MCC1 parameters that are common to all channels.:
   — The base address of the buffer descriptors MCCBASE_CNFG is set to location 0x02000000.
   — The base address of the transmit interrupt queue TINTBASE_CNFG is set to location 0x02078000.
   — The base address of the receive interrupt queue RINTBASE0_CNFG is set to location 0x02070000. Interrupt queues 1 to 3 are not used. Also, this example does not use a superchannel table.
   — The base address of the channel extra parameters XTRABASE_CNFG is set to 0x3800.

3. Initialize the MCC channel extra parameters.

   Each channel uses 8 bytes of extra parameters placed in the dual-port memory at offset XTRABASE_CNFG + (8 × channel number). This step sets the channel receive and transmit buffer descriptor tables relative to MCCBASE_CNFG. For example, the receive buffer descriptor base address RBASE for channel 0 is located at MCCBASE_CNFG + 8 × 0 = 0x02000000. The receive buffer descriptor base address RBASE for channel 1 is located at MCCBASE_CNFG + 8 × 1= 0x02000008. The base transmit buffer descriptor base address TBASE is calculated the same way.

4. Initialize the MCC channel-specific parameters.

   Each channel uses 64 bytes of channel-specific parameters placed in the dual-port memory at offset 64 × channel number. This example assumes transparent operation, so the channel-specific parameters for transparent channels are used.
   — TSTATE and RSTATE provide transaction parameters associated with SDMA channel accesses and start the transmit and receive channel, respectively. This example selects the local bus SDMA to handle transfers to and from the data buffers, buffer descriptors, and interrupt queues.
   — The interrupt mask INTMSK parameter enables underrun, busy, transmit buffer, and receive buffer events to be written to the interrupt queue when the events occur.
   — The channel mode CHAMR parameter sets the channels to operate in transparent mode and activates the channels.

5. Initialize the MCC control and event registers.

   This step programs the MCC1 Configuration Register (MCCF1) to map the MCC1 channels to TDMA1. It also enables receive and transmit interrupts.

Example 6. Initialize the MCC Parameters

```
; ----------------------------------------
; Issue Init Parameters command
; ----------------------------------------
write_l #$1f810000,CPCR
_loop_cpcr
    move.l  (CPCR),d0
    nop
    bmtstc #$0001,d0.h
```

```
    nop
    bf _loop_cpcr

; ----------------------------------------
; Set up MCC Global Parameters
; ----------------------------------------
    write_l #MCCBASE_CNFG,MCC1_MCCBASE      ; Pointer to BD tables in DPRAM
    write_w #$0,MCC1_STATE                  ; Reserved
    write_w #MCC1_MRBLR_CNFG,MCC1_MRBLR     ; Max rx buffer size
    write_w #$0,MCC1_GRFTHR                 ; Use only for HDLC
    write_w #$0,MCC1_GRFCNT                 ; Use only for HDLC
    write_l #$0,MCC1_RINTTMP                ; Used by the CP
    write_l #$0,MCC1_DATA0                  ; Used by the CP
    write_l #$0,MCC1_DATA1                  ; Used by the CP
    write_l #TINTBASE_CNFG,MCC1_TINTBASE    ; Tx interrupt queue base address
    write_l #TINTBASE_CNFG,MCC1_TINTPTR     ; Must point to TINTBASE
    write_l #$0,MCC1_TINTTMP                ; Clear before enabling interrupts
    write_w #$0,MCC1_SCTBASE                ; Superchannel table - not used
    write_w #XTRBASE_CNFG,MCC1_XTRABASE     ; Extra paramter base address
    write_w #0,MCC1_C_MASK16                ; CRC-16
    write_l #$0,MCC1_RINTTMP0               ; Clear before enabling interrupts
    write_l #$0,MCC1_RINTTMP1               ; Clear before enabling interrupts
    write_l #$0,MCC1_RINTTMP2               ; Clear before enabling interrupts
    write_l #$0,MCC1_RINTTMP3               ; Clear before enabling interrupts

    write_l #RINTBASE0_CNFG,MCC1_RINTBASE0  ; Rx interr queue 0 base addr
    write_l #RINTBASE0_CNFG,MCC1_RINTPTR0   ; Must point to RINTBASE0
    write_l #RINTBASE1_CNFG,MCC1_RINTBASE1  ; Rx interr queue 1 base addr
    write_l #RINTBASE1_CNFG,MCC1_RINTPTR1   ; Must point to RINTBASE1
    write_l #RINTBASE2_CNFG,MCC1_RINTBASE2  ; Rx interr queue 2 base addr
    write_l #RINTBASE2_CNFG,MCC1_RINTPTR2   ; Must point to RINTBASE2
    write_l #RINTBASE3_CNFG,MCC1_RINTBASE3  ; Rx interr queue 3 base addr
    write_l #RINTBASE3_CNFG,MCC1_RINTPTR3   ; Must point to RINTBASE3

; ----------------------------------------
; Set up Channel Extra Parameters
; RxBD Ch 0 RBASE = MCCBASE + 8 * 0
; RxBD Ch 1 RBASE = MCCBASE + 8 * 1
; RxBD Ch 2 RBASE = MCCBASE + 8 * 2
; RxBD Ch 3 RBASE = MCCBASE + 8 * 3
; TxBD Ch 0 TBASE = MCCBASE + 8 * 4
; TxBD Ch 1 TBASE = MCCBASE + 8 * 5
; TxBD Ch 2 TBASE = MCCBASE + 8 * 6
; TxBD Ch 3 TBASE = MCCBASE + 8 * 7
; ----------------------------------------
    move.l #RNUM_CH,d1
    move.l #$8,d2
    clr d0
_loop_extra_param
    move.l #(DPRAM1+XTRBASE_CNFG),d3
    imac d0,d2,d3
    move.l d3,r0
    add d0,d1,d4
    move.w d4,(r0)+
    move.w d4,(r0)+
    move.w d0,(r0)+
    move.w d0,(r0)+
    cmpeq d0,d1
    inc d0
```

```
    bf _loop_extra_param

; ---------------------------------------------
; Set up Channel-Specific Parameters Transparent
; ---------------------------------------------
    move.l #DPRAM1,r0
    move.l #RNUM_CH,d0

_loop_specific_param
    write_l #$1b800000,(r0)+         ; TSTATE
    write_l #$10000207,(r0)+         ; ZISTATE
    write_l #$ffffffff,(r0)+         ; ZIDATA0
    write_l #$ffffffff,(r0)+         ; ZIDATA1
    write_w #$0000,(r0)+             ; TxBD flag
    write_w #$0000,(r0)+             ; TBDCNT
    write_l #$00000000,(r0)+         ; TBPTR
    write_w #$0303,(r0)+             ; INTMSK
    write_w #$7600,(r0)+             ; CHAMR
    adda    #$4,r0 ; Reserved
    write_l #$1b800000,(r0)+         ; RSTATE
    write_l #$50ffffe0,(r0)+         ; ZDSTATE
    write_l #$ffffffff,(r0)+         ; ZDDATA0
    write_l #$ffffffff,(r0)+         ; ZDDATA1
    adda    #$8,r0                   ; Read-only
    write_w #MCC1_MRBLR_CNFG,(r0)+   ; MRBLR
    write_w #$5555,(r0)+
    adda    #$4,r0                   ; Next channel
    deceq d0
    bf _loop_specific_param

; ----------------------------------------
; Set up MCC Control Registers
; ----------------------------------------
; Connect  to TDMA ports to MCC1
; MCC1 enable RINT0 and TINT interrupts
; MCC2 is not active
     write_b #$0,MCCF1
     write_b #$0,MCCF2
     write_w #$4004,MCCM1
     write_w #$0000,MCCM2

; Clear the MCC event registers
     write_w #$ffff,MCCE1
     write_w #$ffff,MCCE2
```

5.6 Configure the Serial Interface

Serial interface 1 (SI1) is programmed to connect to the TDMA1 channels. The SI1 Mode Register (SI1AMR) programs the first bank of the SI RAM for TDMA1 and sets TDMA1 to operate in normal mode. It also configures a 1-bit delay between the receive frame sync and the first bit of the receive frame. The frame sync is active on logic ‘1’ and is sampled on the rising edge. Transmit data is driven out on the falling edge, and receive data is sampled on the rising edge of the clock. These frame parameters are set to meet the MSC8102 TDM frame parameter requirements.

The SI1 RAM entries define how the MCC1 data are routed on the TDMA. The transmit and receive SI1 RAM start at offsets of 0x12000 and 0x12400 from the dual-port memory, respectively. Since this example uses four 8-bit channels per frame, the receive SI RAM is programmed to route four bytes of

data on TDMA1 to MCC1 channels 0 through 3. Similarly, the transmit SI RAM is programmed to route four bytes data from MCC1 channels 0 through 3 on TDMA1. The SI1 configuration code is shown in **Example 7**.

Example 7. Configure the SI

```
; ----------------------------------------
; Rx and Tx sync 1-bit delay
; Sync active on "1"
; Tx data on falling edge
; Rx data on rising edge
; Sync sampled on rising edge
; ----------------------------------------
write_w  #$0159,SI1AMR

; ----------------------------------------
; Configure the SI RAM
; Entry 0: $8002 - Ch 0, 1 byte
; Entry 1: $8022 - Ch 1, 1 byte
; Entry 2: $8042 - Ch 2, 1 byte
; Entry 3: $8063 - Ch 3, 1 byte
; ----------------------------------------
    move.l #SI1TxRAM,r0
    move.l #SI1TxRAM+$800,r1
    clr d0
    clr d1
    jsr set_block

    move.l #SI1RxRAM,r0
    move.l #SI1TxRAM,r1
    move.l #$00008002,d0
    move.l #$00000020,d1
    move.l #$00008062,d2

_loop_si
```

5.7 Enable the TDM

The last step of the boot process enables the MCC1 interrupts and enables TDMA1 to begin transmitting and receiving data (see **Example 8**). A short delay occurs before the system enters Debug mode. At this point, the BTAMs from the MSC8102 can be checked in the receive buffer for channel 3 at memory location 0x230000. The receive data may be compared to the expected BTAMs described in **Figure 8**.

Example 8. Enable the TDM

```
; Enable MCC1 and MCC2 interrupts
    write_l #$0C000000,SIMR_L

; ----------------------------------------
; Enable TDMA1
; ----------------------------------------
write_b #$01,SI1GMR
move.l #$C0000,d0
jsr delay
   debug
```

NOTES:

NOTES:

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

AN2442/D