Booting and Simple Usage of the DSP56004/007/009 SHI Port in SPI Mode

by

Tom Zudock

Motorola, Incorporated
Semiconductor Products Sector
6501 William Cannon Drive West
Austin, TX 78735-8598

Order this document by: APR31/D

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF EXAMPLES

SECTION 1

INTRODUCTION

1.1 INTRODUCTION

This application report discusses the procedure for booting the symphony series of Digital Signal Processors (DSPs) through the Serial Host Interface (SHI) port in Serial Peripheral Interface (SPI) mode. The DSP56004/007/009 Evaluation Module was used as the target platform to exemplify the procedure, but the procedure is applicable to all Motorola DSPs with SHI ports.

1.2 SCOPE

The simple DSP application discussed in this application report is booted and then communicated with using the MC68HC711E9 Microcontroller Unit (MCU) on the Evaluation Module. The DSP application is the primary focus of this application report, but at times discussion of the MCU functionality is required for clarity. After boot, the DSP application runs as a slave to the MCU and responds to button selections on the microcontroller interface by toggling a General Purpose Input/Output (GPIO) pin.

1.3 EVALUATION MODULE JUMPER CONNECTIONS

If the user is interested in executing the microcontroller software, the DSP56004/007/009 Evaluation Module must be configured in the fashion shown in **Figure 1-1**. When the application is running, an oscilloscope may be used to monitor the GPIO pins to see the DSP application's response to button choices selected by the user.

Evaluation Module Jumper Connections

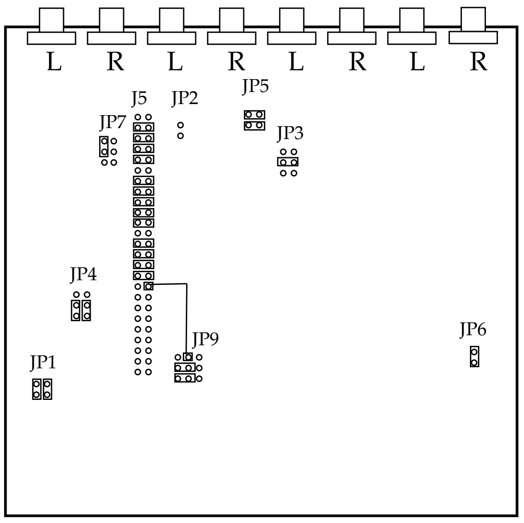

Figure 1-1 Evaluation Module Jumper Connections

SECTION 2

SYSTEM OVERVIEW

2.1 SYSTEM OPERATION

A block diagram of the connections between DSP and MCU is shown in **Figure 2-1.** The MCU is responsible for booting the DSP out of hardware reset, responding to user input, and sending parameters to the DSP. In this application, the DSP only receives data from the MCU. Hence, Master In Slave Out (MISO) is not a required connection for this application. During boot, the data transmitted to the DSP are the opcodes for the DSP program to be executed. After boot, the data transmitted to the DSP are values for the DSP run-time application indicating which GPIO line the DSP should force high.

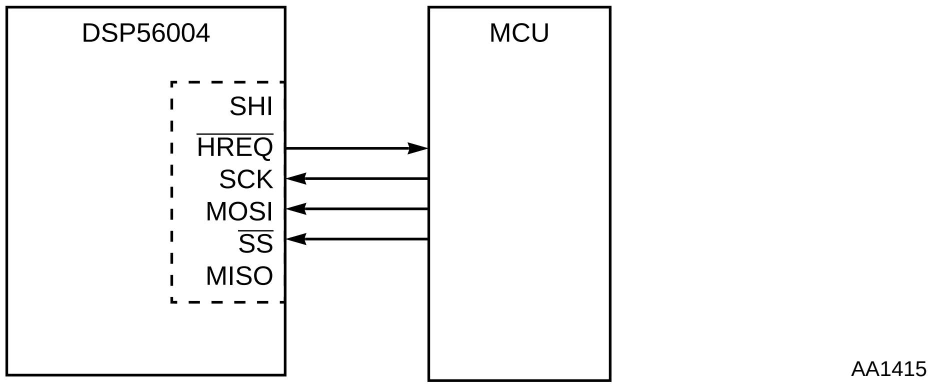

Figure 2-1 Block Diagram of the DSP-MCU System

The DSP operates as a slave device to the MCU. There are four connections between the DSP and the MCU: SCK, MOSI, $\overline{\text{SS}}$, and $\overline{\text{HREQ}}$. Of these, only the Host Request ($\overline{\text{HREQ}}$) is an output from the DSP. SCK is the Synchronous Bit Clock. Master Out Slave In (MOSI) is the serial data input to the DSP. $\overline{\text{SS}}$ is a Slave Select line that initiates the transfer. $\overline{\text{HREQ}}$ is used to indicate that the DSP is ready to receive a word.

Hence, if monitored by the MCU, the $\overline{\text{HREQ}}$ line provides a method of hardware handshaking between the two devices that yields a more robust interface. It is possible to interface two devices without using the $\overline{\text{HREQ}}$ line, but doing so requires careful design of the master device's software to prevent transmitting data when the DSP slave is not ready. By monitoring the $\overline{\text{HREQ}}$ line, the MCU software can be certain the DSP is ready to receive a word before transmitting it.

A flow diagram that describes the interaction between the DSP and MCU is shown in **Figure 2-2**. The MCU system operation is shown on the left, while the DSP system operation is shown on the right. Hardware reset is the initial event that starts the system. Since the jumpers for the MODA, MODB, and MODC pins have been set to indicate SHI boot in SPI mode, the DSP executes the SPI boot portion of the internal boot ROM program. The DSP polls for data to be received on the SHI port.

The MCU, on the other hand, first checks the $\overline{\text{HREQ}}$ pin to see if the DSP is ready to receive data. When the MCU verifies that $\overline{\text{HREQ}}$ is asserted, it transmits a data word—one DSP program opcode. The DSP sees that a data word is available at the SHI Host Receive (HRX) register and copies that word to DSP Program RAM. The MCU continues to transmit data to the DSP until 512 words have been sent. The DSP in turn fills its entire Program RAM with this data. Next, the boot program changes the operating mode to enable Program RAM and executes the newly loaded program starting at memory location `P:$0`. The DSP and MCU are now ready for steady state operation of their programs and enter an idle state.

Both the DSP and the MCU await user input. When a button is pressed in the MCU user interface (LCD menus), the MCU responds by transmitting a value to the DSP SHI port. This transmission generates a DSP interrupt resulting in execution of the SHI interrupt service routine. The SHI interrupt service routine receives the word from the HRX register, constructs the GPIO word, and moves it to the GPIO register. The external GPIO pins will respond accordingly. The user's perception is that the a button was pressed and a GPIO line was asserted. The DSP and MCU resume the idle states.

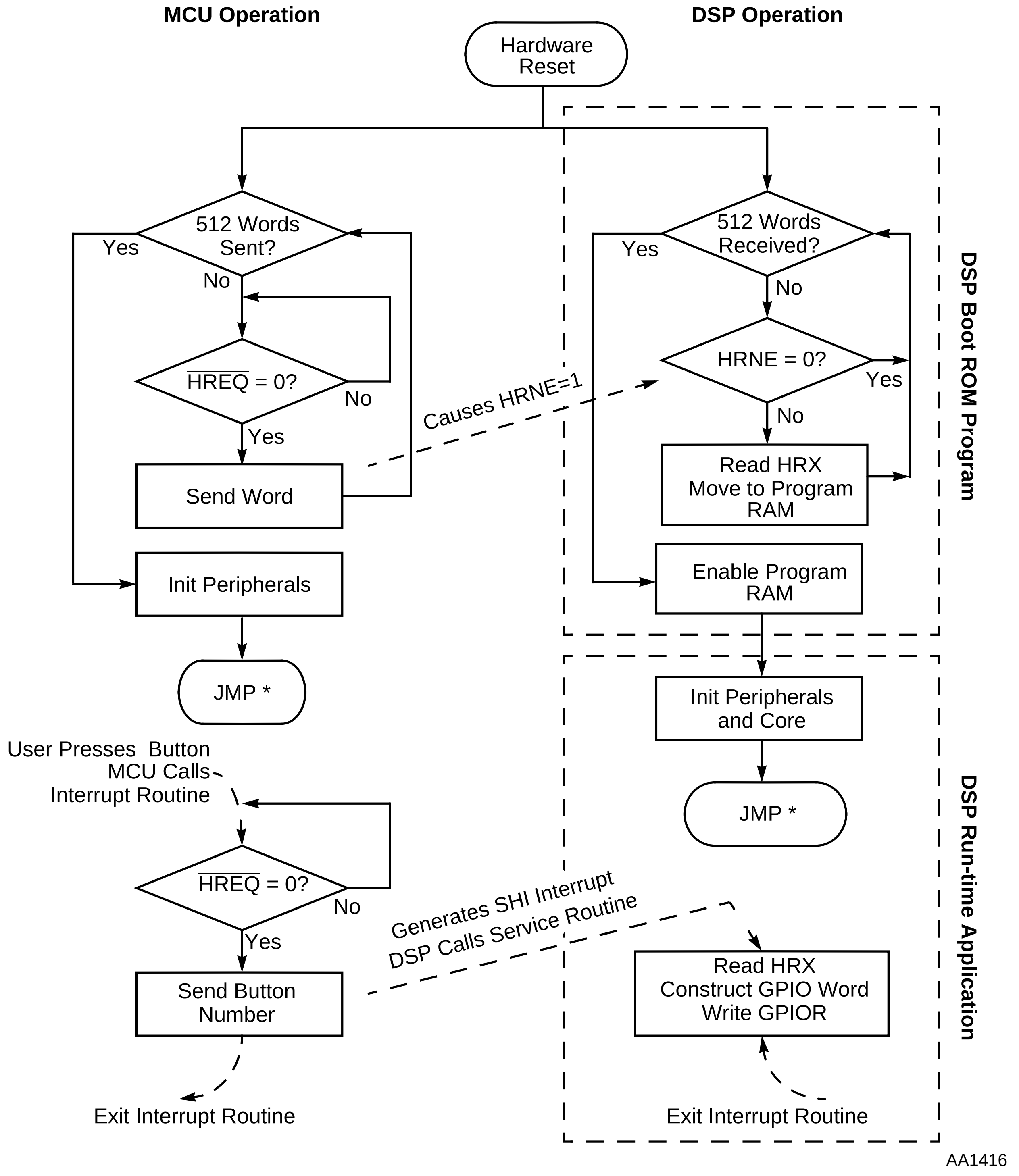

Figure 2-2 DSP-MCU System Flow Diagram

2.2 DSP SUBSYSTEM CODE

In this section, the behavior of the DSP subsystem is discussed in more detail. First, an examination of the relevant portion of the boot ROM program is presented. Thereafter, the DSP software that has been loaded is presented.

2.2.1 DSP Boot from Reset

The boot program determines the boot mode by examining the MA, MB, and MC bits of the Operation Mode Register (OMR). These bits reflect the state of the MODA, MODB, and MODC pins at the time of hardware reset. By setting these pins to binary 101, or Mode 5, the bootstrap ROM is enabled and will execute the portion of code responsible for booting through the SHI port. The code section is shown in **Example 2-1**. The entire bootstrap ROM program is found in the DSP56004, DSP56007, and DSP56009 Users' Manuals.

Upon reaching this code excerpt, the bootstrap program has already determined that boot will be through the SHI port, but it has not determined whether it will be in I^2C or SPI mode. Based upon the boot mode, the bootstrap program configures the SHI port accordingly and enters the "`loop_2`" DO loop. Once inside this loop, the DSP polls the Host Receive FIFO Not Empty (HRNE) bit of the SHI Control Status register. When the microcontroller transmits an opcode to the DSP56004 SHI port, the HRNE bit is set and the code will advance. The bootstrap program reads the opcode from the HRX register and moves it into Program RAM.

Notice that the loop will execute 512 iterations. Hence, regardless of the size of the program that is being booted into Program RAM, the microcontroller must transmit 512 words so that the DSP will execute all iterations of this loop. Upon completion of this loop, the code will change the OMR to enable the Program RAM and then jump to address `P:$0` to begin execution of the newly loaded program. Refer to the DSP56004 User's Manual for details on that section of the bootstrap program.

Example 2-1 SHI Boot in SPI Mode Section of Bootstrap ROM Program

```
; This is the routine that loads from the Serial Host Interface.
; MC:MB:MA=101 - Bootstrap from SHI (SPI)
; MC:MB:MA=111 - Bootstrap from SHI (IIC)
; If MC:MB:MA=1x1, the internal program RAM is loaded with 512 words received
; through the Serial Host Interface (SHI). The SHI operates in the slave
; mode, with the 10-word FIFO enabled, and with the HREQ pin enabled for
; receive operation. The word size for transfer is 24 bits. The SHI
; operates in the SPI or in the IIC mode, according to the bootstrap
; mode. The OnCE is enabled by the bootstrap code.
shild       jclr        #ma,omr,ma_0
shild_1     movep       a,x:ogdbr               ; enable the OnCE
            move        #$0002A9,x1             ; SHI control word (SPI)
; HEN=1, HI2C=0, HM1-HM0=10, HFIFO=1, HMST=0,
; HRQE1-HRQE0=01, HIDLE=1, HBIE=0, HTIE=0, HRIE1-HRIE0=00
            move        #<0,r0                  ; start loading code at P:$0
            jmp         <$100

            org         pl:$100,pl:$100         ; Start of second 32-word group
            jclr        #mb,omr,shi_loop        ; If MC:MB:MA=101, then SPI
            bset        #hi2c,x1                ; IIC (HI2C=1)
shi_loop    movep       x1,x:hcsr               ; enable SHI
            do          #512,_loop2
            jclr        #hrne,x:hcsr,*          ; wait for HRX not empty
            movep       x:hrx,p:(r0)+           ; store in Program RAM
_loop2
            jmp         <exit                   ; Finish
ma_0        jmp         <shild_1                ; make MC:MB:MA=1x0 equal to MC:MB:MA=1x1
; This code fills the unused bootstrap rom locations with their address
            dup $120-*
            dc *
            endm
```

2.2.2 DSP Run-Time Application

The DSP software is shown in its entirety in **Example 2-2**. As already stated, the DSP begins execution at `P:$0`. Therefore, a JMP to the main program is executed first. Due to the simplicity of this DSP application, no X or Y data memory is utilized. Therefore, this program is already fully loaded and ready to begin execution. However, if the application did make use of X and/or Y data memory (specifically data variables that require initial values), then the MCU should complete the boot process by transmitting the data memory values and the DSP should, of course, receive and store them in the X data RAM and Y data RAM.

The main program merely initializes peripherals and the core. First, the GPIO port is programmed so that all of the pins are outputs and sets the initial output data to zero. Since the DSP boot program required the SHI port to boot internal Program RAM, the port is already partially initialized for this application. The boot program sets the SHI port to SPI mode, sets data length to 24-bit data, ensures that $\overline{\text{HREQ}}$ is asserted when ready to receive a word, and enables the port. Since this DSP application will respond to the SHI via interrupt, the Host Receive Interrupt Enable bits (HRIE1–0) of the SHI Control Status register are set so that an SHI interrupt is generated for “Receive FIFO Not Empty” and “Receive Overrun” conditions. Next, the core's interrupt handling system must be configured so that the DSP will respond to the interrupt. First, the SHI is given an interrupt priority of two by setting the proper bits in the Interrupt Priority Register (IPR). Since the SHI is the only interrupt generating device, any value greater than zero would have sufficed. However, in more complex applications, the priority structure of the interrupts can be highly significant for optimal DSP system operation. Next, interrupts are enabled by clearing the interrupt mask in the mode register. Finally, the main program ends in an infinite loop with background tasks servicing interrupts.

Proper handling of interrupts begins with the interrupt vectors. As stated earlier, the SHI port will generate interrupts for receive FIFO not empty and receive overrun. These are two distinctive interrupts. Depending upon the application, the software designer may decide to handle the situations differently. In this application, they are both handled by the same routine. A JSR to the SHI_ISR in the interrupt vector table results in the interrupt service routine being called when either SHI interrupt is generated. The JSR indicates to the core that a long interrupt will be performed requiring that the interrupt service routine terminate with an RTI.

The interrupt service routine itself is straightforward. First, the word transmitted from the MCU to the DSP resides in the HRX register. This word contains a value indicating which GPIO line should be set high. It is moved into register x0. Notice that a MOVEP instruction is used to access the peripheral. Next, accumulator A is initialized with the GPIO port initialization value. Since the GPIO port can only be written to using all 24 bits, the bits responsible for setting the direction of the GPIO pins must be combined with the data for the pins. Next, the accumulator is ORed with register x0, thus combining the GPIO port data with the GPIO port settings. The fully constructed GPIO port word is moved to the GPIO register and the external signals respond accordingly. Finally, an RTI instruction is executed and program flow returns to the main program.

Example 2-2 DSP Application Software

```
          page    132,66,3,3
          opt     cex,mex,cc,mu
          title   'main'
          section    main
          nolist
          include 'ioequ.asm'
          list
          comment ~
********************************************************************************
    Start of assembly code
********************************************************************************
          ~
          org        p:0
          jmp        <main                 ; jmp to main out of reset

          org        p:$24                 ; receive FIFO not empty
          jsr        SHI_ISR               ; jsr to SHI_ISR
          org        p:$2a                 ; receive FIFO overrun
          jsr        SHI_ISR               ; jsr to SHI_ISR
          org        p:$80
main
          movep      #$000f00,x:<<GPIOR    ; set all GPIO to outputs
          bset       #HRIE0,x:HCSR         ; enab intrpt for FIFO not empty
          movep      #$003000,x:IPR        ; set interrupt priority for SHI
          andi       #$fc,mr               ; enable interrupts
          jmp        *                     ; idle state for main program

SHI_ISR
          movep      x:<<HRX,x0            ; get new received data
          move       #$000f00,a            ; GPIO port val
          or         x0,a                  ; update data portion
          movep      a1,x:<<GPIOR          ; output new GPIOR val
          rti
          endsec
```

SECTION 3

CONCLUSION

3.1 CONCLUSION

In conclusion, this simple DSP application exemplifies how to boot and respond to the SHI port. Additionally, pertinent information regarding interrupt handling and using the GPIO port has introduced the reader to other fundamental aspects of programming Motorola DSPs. Finally, the application can readily be used as a starting point for more complex DSP applications that utilize the SHI port.

3.2 EXAMPLE CODE

The example code presented in this application report is available via the Motorola website at the following address:

```
http://www.motorola-dsp.com/documentation/appnotes
```