ANALOG DEVICES
Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

In-Circuit Programming of an SPI Flash with SHARC® Processors

Contributed by Brian M.

Rev 2 – August 22, 2007

Introduction

ADSP-2126x, ADSP-2136x, and ADSP-2137x SHARC® processors are able to boot from SPI flash devices, providing a cheap booting alternative to parallel flash, SPI EEPROM, or host processor schemes.

Hereafter, this document collectively refers to ADSP-2126x, ADSP-2136x, and ADSP-2137x processors as SHARC processors.

Note that earlier SHARC processors, (ADSP-2106x and ADSP-2116x), do not support booting from SPI flash devices.

Since the details of the general booting process are discussed in the System Design chapter of the *SHARC Processor Hardware Reference Manuals* [2][4][5], this EE-Note describes how to program the desired code or data into the SPI flash in-circuit using an STMicroelectronics M25P20 serial flash memory. Example code to implement in-circuit programming for ADSP-21262 and ADSP-21364 processors is provided in the associated `.ZIP` file.

Default SPI Settings

The default SHARC SPI settings do not match those supported by most SPI flash devices. Two issues arise when interfacing an SPI memory device to these processors: word transfer order and SPI mode.

By default, SHARC processors transfer words in least significant bit first (LSBF) format, yet most SPI memory devices transfer in most significant bit first (MSBF) format. In most applications, this will not be a problem because the SPI of the processor can be set up to transfer in MSBF format. However, while booting, the word format setting cannot be changed. Therefore, it is necessary to program boot data into the SPI memory device in a bit-reversed manner for devices that support MSBF format only.

There are two ways to transfer bit-reversed words to the flash. The loader utility included with the VisualDSP++® development tools has an option to automatically bit-reverse the boot data. The SPI Master and SPI PROM loader options use this format. This format should be used only when the SPI memory will be programmed by a dedicated memory programmer (such as those used to program an SPI EEPROM).

Alternatively, it is possible to communicate with the SPI flash in an LSBF format, while bit reversing the commands to look like they are in MSBF format. For debugging purposes, this format is much easier to read, since it matches the format shown in VisualDSP++ memory windows. This is the method used in this example.

The other issue is much more problematic, but does not apply to the ST M25P20 SPI flash used in this example. By default, SHARC processors use SPI mode 3 (the SPI clock toggles at the start of the first data bit, and the SPI clock is active-low). Some SPI memory devices support SPI mode 0 only, or provide only partial mode 3 support. If the device does not support SPI mode 3, the part cannot boot these SHARC processors.

If partial support for SPI mode 3 is available (as in the Atmel AT25F512 serial flash memory found on ADSP-2126x, ADSP-2136x, and ADSP-2137x EZ-KIT Lite® development boards), a workaround may be available. For the Atmel part mentioned above, reading from the flash in SPI mode 3 works correctly, but programming to the flash exhibits a bit error on the last word of each page if a certain pattern exists in the previous word. This can be addressed in two ways. The easiest way to avoid this problem is to append a final (8- or 32-bit) word, each bit existing entirely of 1s (0xFF or 0xFFFFFFFF). If this final word wraps around to the beginning of the page, it will not overwrite what was previously programmed there, nor does it waste that word, since the flash cannot overwrite bits that are set to 1 without an erase command. This method is used by the VisualDSP++ Flash Programmer utility. The alternate method presented herein avoids using problematic words. Since the SPI flash is programmed on a page-by-page basis, it is possible to shift the position of the starting word to ensure that the offending bit pattern will not be matched. For an example of how to accomplish this, refer to the Atmel SPI Flash Programmer code included with the ADSP-2126x, ADSP-2136x, and ADSP-2137x EZ-KIT Lite installation.

SPI Interface and Status Bits

Each SPI consist of a dedicated Transmit/Receive Shift register and Transmit/Receive Buffer register. Data to be transmitted is written into `TXSPIx` (Buffer register) and then automatically transferred into the `TXSRx` (Shift register). Once a full data word is received in the `RXSRx` register, the data is automatically transferred into the `RXSPIx` register, from which the data is read. Data transfer to/from internal memory can be done either through core transfer or DMA transfer. During core transfer, a single word is written into the `TXSPIx` register and a single word is received from the `RXSPIx` register. But while using DMA, there is a four-word deep DMA FIFO that the SPI ports use to improve the data throughput, in addition to the `TXSPIx` and `RXSPIx` data buffers. During transmit, the FIFO is not available for receive path and vice-versa. When performing transmit DMA transfers, data moves through the DMA FIFO, then into the `TXSPIx` buffers, and finally into the shift register. DMA interrupts are latched when the I/O processor moves the last word from memory to the peripheral. For the SPI, this means that the SPI "DMA complete" interrupt is latched when there are still six words yet to be fully transmitted (four in the FIFO, one in the `TXSPIx` buffers, and one being shifted out of the shift register).

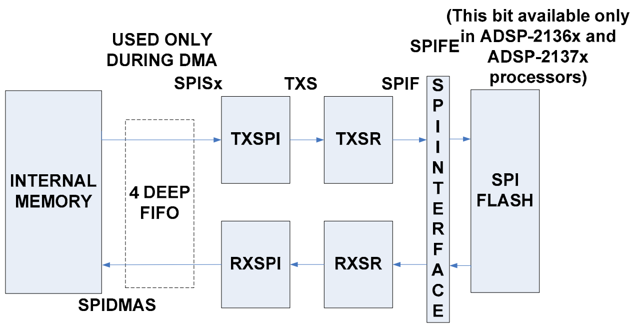

Figure 1. Status bits that must be checked at each stage

While performing a transfer, it is important to poll the appropriate status bits to ensure that all valid data has been transferred completely. Figure 1, denotes the different bits that need to be checked at different stages of transfer. Some of the important bits have been described next.

- SPIDMAS – This bit is set during a DMA transfer to/from the FIFO and cleared after the transfer is complete.
- TXS - This bit is set when the TXSPIx register is full and is cleared when empty (completed transferring a word to TXSRx register).
- RXS – This bit is set when the RXSPIx register is full and is cleared when empty (completed transferring a word to internal memory for code transfer or the FIFO for DMA transfer).
- SPIF – This bit is set when a single word has been shifted out of a TXSRx or RXSRx register.
- SPIFE – This bit is available only in ADSP-2136x and ADSP-2137x processors. This bit indicates that all SPI-related operations have been completed at the external SPI interface.

It is important to follow the right order while configuring the SPI for transmit and receive and while checking the status bits. Refer to [2][4][5] for details on how to configure the SPI.

Flash Programmer Functions

An SPI flash operates by receiving commands from the Master Out Slave In (MOSI) line, and returning any response on the Master In Slave Out (MISO) line. The FLAG0 pin is used as chip select for the SPI flash device. The following is a brief discussion of the commands implemented in the associated example code.

Write Enable (0x06)

This function enables modification to the status register settings or contents of the flash. Since no address or additional data is required, this command is sent directly by the core as a 1-byte, MSBF word.

The Write Enable function is called automatically by each function that requires the Write Enable Latch bit in the flash's status register to be set to function properly. These functions are the Write Status Register, Page Program, and Erase (Bulk and Sector) commands.

Write Disable (0x04)

This function disables modification of the status register settings or the contents of the flash after the Write Enable command has already been registered. Since no address or additional data is required, this command is sent directly by the core as a 1-byte, MSBF word.

Read Status Register (0x05)

This function fetches the value of the byte-wide status register. It is implemented as a 2-word DMA using a 1-byte MSBF word.

The first returned byte, which corresponds to the command sent from the processor, can be ignored. The second returned byte contains the value of the status register placed in the lowest byte of the destination.

Write Status Register (0x01)

This function writes to the byte-wide status register. It is implemented as a 2-word DMA using a 1-byte MSBF word. The first byte sent is the command from the processor, and second byte is the desired value of the status register.

The Write Enable command must be sent before using this command. Before calling this function, place the desired value of the status register in the second location of the status register buffer. This is the same location that returns the value of the status register in the Read Status Register function. The first location in this buffer is reserved for the command being sent to the device.

Read Data Bytes (0x03)

This function reads any number of words from the flash. The destination address in the processor's internal memory, the source address in the flash, and the number of 32-bit words to be read are passed to the function using the dedicated memory locations. It is implemented as an N-word DMA using 32-bit LSBF words, where N is the number of words passed to the function. The first word in the destination buffer will be garbage corresponding to the 1-byte command plus 3-byte address that must precede the data (one 32-bit word).

Page Program (0x02)

This function complements the Read function. The destination address in the flash, source address in the internal memory of the processor, and number of 32-bit words are passed to this function using the dedicated memory locations.

The flash requires that it be programmed one page at a time. Before being sent to the flash, the buffer referenced in the call is transferred into a temporary buffer corresponding to the page size.

After each page has been sent, the lowest bit in the flash's status register, write-in-progress (`WIP`), is tested in a loop to determine when the flash has finished programming the new data into memory.

The function included in this example also verifies each page after completing the write by calling the Read Data Bytes function and comparing the result of the call to what was sent. The number of 32-bit words that do not match is tracked.

Because this example programs the flash for booting, this requires LSBF format, both the Read Data Bytes command and the Page Program command use LSBF format. Therefore, both the command and address must be bit-reversed before transfer so that the flash receives them in MSBF format. (This is accomplished using the `BITREV` instruction [3].)

Sector Erase (0xD8)

This function erases one sector of the flash according to the memory layout in the SPI flash's data sheet. An address in the sector to be erased is passed to the function with the call. This command uses a 1-byte command and 3-byte address, comprising one 32-bit word. Since the Page Program and Read Data Bytes function use a function that joins the command and address into a single 32-bit word, the Sector Erase command is sent in LSBF format. After sending the command, the status register is polled until the `WIP` bit is clear.

Bulk Erase (0xC7)

This function erases the entire flash. No address is passed to this function. Therefore, the function is implemented as a 1-byte command sent in MSBF format. After sending the command, the status register is polled until the `WIP` bit is clear.

Deep Powerdown (0xB9)

This function puts the flash into a low-power state. Since no address or additional data is required, this command is sent directly by the core as a 1-byte, MSBF word.

Deep Powerdown Release/Electronic Signature (0xAB)

This function returns the flash from the low-power state. It is also used when the flash is not in the low-power state to retrieve the electronic signature of the part. It is implemented as a 5-word DMA using a 1-byte MSBF word. The first byte, which corresponds to the command from the processor, can be ignored. The middle three bytes, which are garbage returned by the flash, can also be ignored. The final byte contains the value of the status register placed in the lowest byte of the destination. For the M25P20, the lowest byte is 0x13.

Generating the Loader File

To generate a loader file compatible with this example, it is necessary to use SPI Slave format. Though it may seem logical to use SPI PROM format, SPI PROM format produces an image that is bit-reversed. In addition, an SPI PROM only uses a 16-bit address; thus, an extra byte is pre-pended to the image to make the boot stream compatible with the processor. For details, refer to the booting section in the *SHARC Processor Hardware Reference Manuals* [2][4][5].

Considerations for Different Processors

Be aware of the following considerations.

- Status Bit Availability – As mentioned earlier, `SPIFE` is an additional bit that is available for ADSP-2136x and ADSP-2137x processors in the `SPISTAT` register. This bit is set to 1 when the last bit of the last word has been shifted out from the external SPI Interface.
- Note that once a command/instruction is sent to the SPI flash device, a certain amount of time must pass for the SPI flash to be available for further use. To know the exact values, refer to the *STMicroelectronics M25P20 Serial Flash Memory Datasheet* [7]. The processor must wait for that many cycles before sending out another instruction. The number of cycles for the delay must be calculated according to the core clock of the processor. For example, it takes 5 ms for the part to be available after Page Program [7]; therefore, ADSP-2126x processors with a 200 MHz core clock (i.e., 3 ns) must wait for 1.66×10^6 cycles. The same delay routine may be modified for other processors, according to the core clock operated.

Conclusion

This document discusses the bits that are vital in ensuring reliable transfer of data between SHARC processors and the SPI Flash and also discusses the different instructions available in the SPI flash. Each function included in the example is described. To compare this project to a similar example, refer to the Atmel SPI Flash Programmer included with the ADSP-2126x, ADSP-2136x, and ADSP-2137x EZ-KIT Lite installation for the VisualDSP++ development tools.

References

[1] *ADSP-2126x SHARC DSP Core Manual.* Rev 2.0, February 2004. Analog Devices, Inc.

[2] *ADSP-2126x SHARC Processor Peripherals Manual.* Rev 3.0, December 2005. Analog Devices, Inc.

[3] *ADSP-2136x SHARC Processor Programming Reference*. Rev 1.0, March 2007. Analog Devices, Inc.

[4] *ADSP-2136x SHARC Processor Hardware Reference*. Rev 1.0, October 2005. Analog Devices, Inc.

[5] *ADSP-21368 SHARC Processor Hardware Reference*. Rev 1.0, September 2006. Analog Devices, Inc.

[6] *VisualDSP++ 4.5 Loader and Utilities Manual.* Rev 1.0, April 2006. Analog Devices, Inc.

[7] *M25P20 Serial Flash Memory Datasheet.* Rev 10, June 2006. STMicroelectronics, Inc.

Document History

Revision	**Description**
Rev 2 – August 22,2007 by Deepa Venkataraman	Made the EE-Note generic for ADSP-2126x, ADSP-2136x, and ADSP-2137x SHARC processors, and updated document title accordingly.
Rev 1 – March 02, 2004 by Brian M.	Initial Release.