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.

Interfacing ADSP-BF53x Blackfin® Processors to NAND FLASH Memory

Contributed by Ke Zhan and Daniel Zhao

Rev 1 – November 10, 2006

Introduction

NAND flash memory technology has emerged as a suitable, high-density alternative to hard disk drives in power-sensitive portable devices. Today, most cellular telephones, Personal Digital Assistants (PDAs), digital cameras, Portable Media Players (PMPs), and other mobile computing/communications/consumer products use NAND flash memory to store increasingly large media files, data, and operating systems.

Blackfin® processors are well-suited for portable systems due to their low power dissipation, high performance, and convergent architecture. As such, there is a growing demand to interface Blackfin processors to NAND flash memories.

This EE-Note describes how to interface both MLC- and SLC-type NAND flash memories to ADSP-BF53x Blackfin processors. Included are the low-level driver functions, which were validated on a hardware platform consisting of a NAND flash daughter board connected to an ADSP-BF533 EZ-KIT LITE® (rev. 1.6) evaluation system and the VisualDSP++® 4.0 (Updated April 2006) development tools.

To provide flexibility to system designers, two approaches are provided. The memory devices used to demonstrate this interface are Samsung's K9F2G08U0M SLC NAND flash and Toshiba's TC58NVG3D4CTG MLC NAND flash.

NAND Flash vs. NOR Flash

NOR flash devices are widely used in the electronics industry. They offer an easy memory interface and are suitable for code execution, making them ideal for devices that do not need data storage. NOR memory offers good read performance, but has slow write/erase times. However, as modern devices become more and more sophisticated, they need to store more data and programs locally while supporting faster erase/write times. NAND flash offers all of this, plus better prices, in capacities ranging from 8 to 512 MB. NAND is an I/O device and requires a relatively complicated driver for any operation.

Memory cells that are accessed frequently become more prone to bit errors. Thus, NAND flash devices typically include extra memory so that the NAND flash file system can replace bad blocks with good blocks without decreasing the size of the available memory. A NAND flash file system, such as JFFS2, will typically attempt to evenly wear the whole memory to increase the life of the device.

NAND Flash: SLC vs. MLC

There are two different types of NAND flash devices in the market today: Single-Level-Cell (SLC) devices and Multi-Level-Cell (MLC) devices. MLC chip technology is capable of storing two or more bits of data per memory cell, while SLC chip technology allows just one bit of data per memory cell.

MLC chips provide much larger storage capacity than SLC; however, they also suffer from lower bandwidth for read and write operations. Also, the Error Correcting Code (ECC) requirement for MLC is higher than for SLC, as MLC technology is more prone to bit errors. The cost per bit of MLC flash is much lower than that of SLC devices. Figure 1 shows the comparison.

Item	SLC	MLC
Storage capacity	Small	Large
Throughput	High	Low
ECC	1bit/528B	4bit/528B
Write/Erase Endurance	100K times	10K times
Cost	High	Low

Figure 1. SLC vs. MLC Performance Comparison

MLC NAND flash memory provides a competitive level of performance and makes high-density NAND cards more affordable, enabling new breakthroughs in portable applications. SLC NAND flash is appropriate for specific, high-performance applications.

SLC NAND flash is better suited for applications that require speed. It performs many write/erase cycles that require high endurance and high reliability, such as in solid-state hard disk drives and devices used for seismic data recording, networking, HDTV, bar code scanners, shockproof PCs, flight recorders, handheld code storage devices, as well as PDAs and digital still cameras.

Typical applications that are well-suited for MLC NAND flash memory include digital video and digital still camera memory cards, USB flash drives, MP3 players, automobile diagnostic monitors, GPS devices, animatronics devices, video game cards, and many toy applications. Applications that may be suited for either MLC NAND or SLC NAND technology include memory cards for printers (fonts), cell phones, telecom applications, voice mail, and TV set-top boxes. Figure 2 illustrates the classifications.

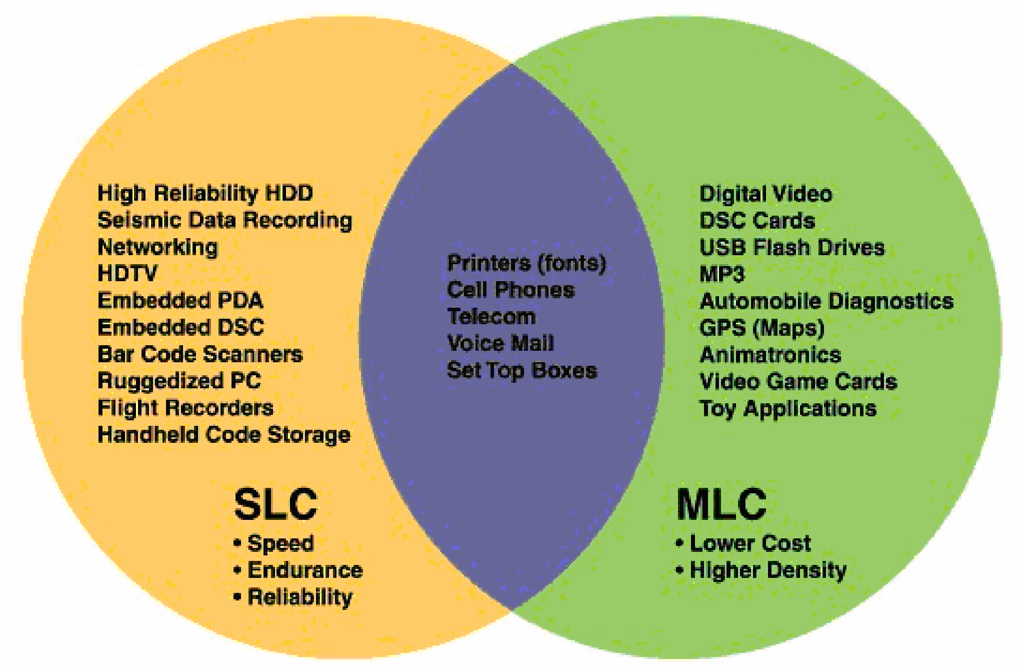

Figure 2. Typical Applications for SLC and MLC Devices

NOR flash memories and other traditional SRAM-like devices and memories use discrete address data buses and basic control signals. NAND flash memory is optimized for large block transfers, incorporating a single set of I/O pins that are used for both address and data.

For Blackfin processors, it is possible to map the NAND flash memory ports to asynchronous memory space and access them as a standard memory-mapped device. The hardware interface for both SLC and MLC NAND flash is identical.

Basic NAND Flash Interface

NAND flash devices utilize I/O pins for both address and data, as well as for command inputs. The command signals and command words are the same for both SLC and MLC NAND flash devices. This makes the interface to both SLC and MLC flash chips universal for all Blackfin processors.

Flash memory is accessed by column, page, and block. The read and program operations execute at the page level, and erase operations are performed at the block level.

The basic commands supported by NAND flash memories are block erase, page program, read status, and page read. The Chip Enable (`CE#`) pin should be asserted low to access the device. When asserted, the NAND flash can be controlled to accept bytes provided by the host when the Write Enable (`WE#`) signal is asserted low or provide data to the host when the Read Enable (`RE#`) signal is asserted low. When `CE#` is high, the chip ignores `RE#` and `WE#`, and the I/O is tri-stated. The Command Latch Enable (`CLE`) signal is used to send commands to the device when `CE#` is asserted. The Address Latch Enable (`ALE`) signal is used to latch the address into the address register of the flash device. Figure 3 depicts the internal register accessed as a result of valid combinations of `CLE` and `ALE`.

ALE	CLE	Register Selected
0	0	Data Register
0	1	Command Register
1	0	Address Register
1	1	Reserved

Figure 3. Valid ALE/CLE Combinations

Samsung K9F2G08U0M Interface

The Samsung K9F2G08U0M is offered in a 256M x 8-bit (2G-bit) configuration with a spare 64M-bit capacity. This memory device is an optimal solution for large nonvolatile storage applications such as solid-state file storage and other portable applications requiring non-volatile memory.

The device is offered in 2.7 V, 3.3 V, and 3.6 V supply voltage ranges. Typically, a program operation can be performed in 200 µs on the 2112-byte page, and an erase operation can be performed in 2 ms on a 128-Kbyte block. Each byte in the data page can be read in 30 ns. The I/O pins serve as the ports for address and data input/output, as well as for command input. One page in the K9F2G08U0M consists of 2112 bytes, and one block comprises 64 pages. Figure 4 shows the organization of the K9F2G08U0M flash memory device in terms of pages and blocks.

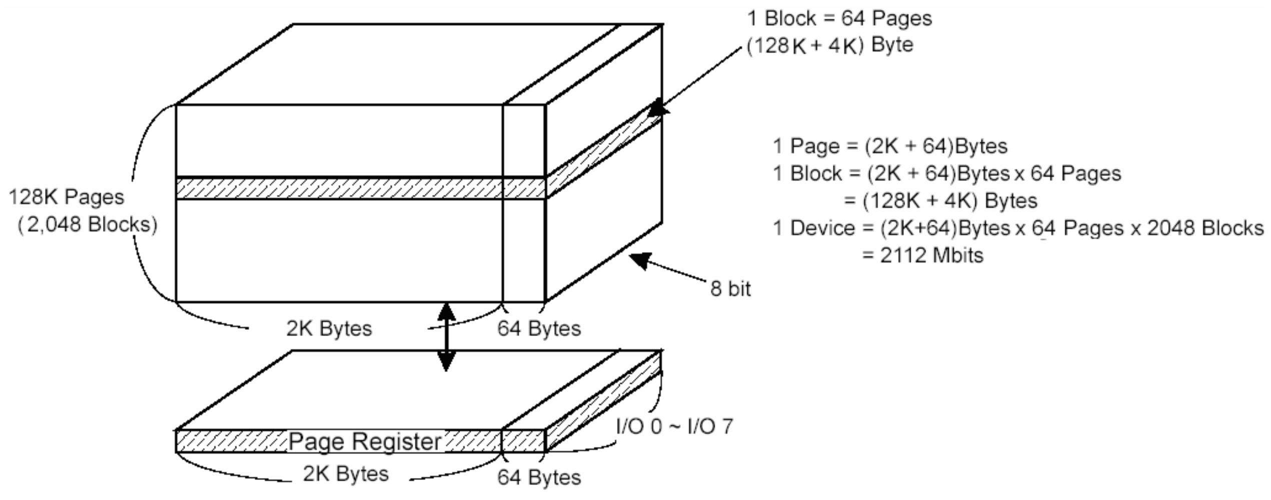

Figure 4. K9F2G08U0M Flash Memory Organization

Toshiba TC58NVG3D4CTG10 Interface

The Toshiba TC58NVG3D4CTG10 device is a single 3.3 V 8-Gbit NAND Electrically-Erasable and Programmable Read-Only Memory (NAND E2PROM), organized as 2112 (2048+64) bytes per page, 128 pages per block, and 4160 (4096+64) blocks.

The device has two 2112-byte static registers that allow program and read data to be transferred between the register and the memory cell array in 2112-byte increments. The erase operation is implemented on a single block unit.

NAND Flash Interface to the ADSP-BF53x Processor

Because ADSP-BF53x Blackfin processors do not have an on-chip NAND flash controller, a software driver is used to control and access NAND flash memory. Two approaches are used to interface a Blackfin processor to a NAND flash device: GPIO pins and EBIU pins.

Using GPIO Pins to Interface to NAND Flash

The first approach uses General-Purpose I/O pins (GPIOs) to drive all the required control signals. This approach requires careful timing for the necessary delays to match the timing requirements for NAND access.

Figure 5 shows details regarding the connections between an ADSP-BF533 Blackfin processor and a Samsung K95G08U0M flash memory device.

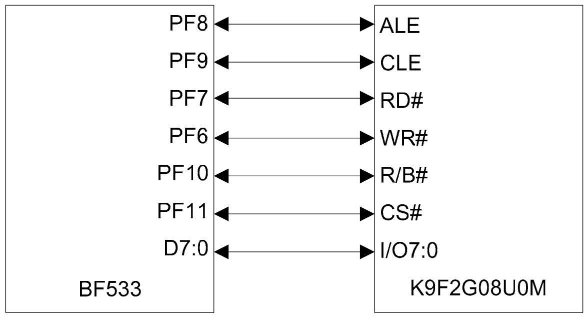

Figure 5. Using GPIO for NAND Flash Interface

Using EBIU Pins to Interface to NAND Flash

A second approach is to use the External Bus Interface Unit (EBIU) and a single GPIO pin to communicate with the NAND flash memory device. Standard NAND flash requires that its `CE#` remains valid during the read busy period. So, for compatibility with both standard NAND flash and Chip Enable Don't Care (CEDC) NAND flash, a GPIO pin is used to drive the `CE#` signal. All other signals are driven directly by the EBIU interface.

Figure 6 shows details regarding the connections between an ADSP-BF533 Blackfin processor and a Samsung K9F2G08U0M flash memory device.

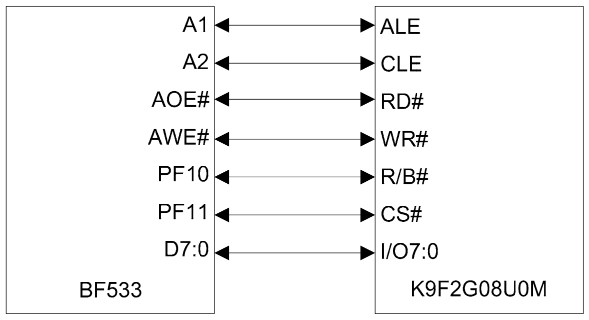

Figure 6. Using EBIU Pins for NAND Flash Interface

The EBIU can be programmed to control up to four banks of devices with very flexible timing parameters. Each bank occupies a 1-Mbyte segment, regardless of the size of the device used, so these banks will only be contiguous if each is fully populated with 1 Mbyte of memory. The EBIU is clocked by the system clock (`SCLK`), which is the frequency at which all synchronous memories interfaced to the processor will operate. Since the EBIU is automatically controlled by the asynchronous memory controller of the Blackfin processor, hooking the NAND flash in this way is simple and yields optimal access performance.

NAND Flash Driver Development

This EE-Note provides a basic Samsung K9F2G08U0M and Toshiba TC58NVG3D4CTG NAND flash driver for the two interface approaches described above.

This section focuses on the second approach implementation and provides timing waveforms for each command. Listing 1 shows the code example for the block erase code implemented.

```
bool  _NF_Block_Erase(unsigned int Block_Address){
    unsigned char NAND_Data = 0;

    NAND_Address_Table.Value = Block_Address << 12;
    Write_Command(0x60);
    Write_Address(NAND_Address_Table.NAND_Address.NAND_Address_A12_A19,
                  NAND_Address_Table.NAND_Address.NAND_Address_A20_A27,
                  NAND_Address_Table.NAND_Address.NAND_Address_A28,
                  NAND_Address_Table.NAND_Address.NAND_Address_A0_A7,
                  NAND_Address_Table.NAND_Address.NAND_Address_A8_A11,
                  0x03);

    Write_Command(0xd0);
    while(!Wait_NAND_RADY());
    Write_Command(0x70);

    delay(50000);
    NAND_Data = *pNAND_DATA_REG;
    NAND_Finish();

    if(NAND_Data != 0xe0)//0xc0)
         return false;

    else return true;
}
```

Listing 1. Block Erase Function

Listing 2 shows the code example using the page write function.

```
bool  NF Page Write(unsigned int Block Address,unsigned short Page Address,unsigned
char *psrc)
{
    int i;
    unsigned char NAND_Data;
    int NAND_Addr;

    NAND_Addr = (Block_Address << 12) | Page_Address;
    NAND_Address_Table.Value = NAND_Addr;
    Write_Command(0x80);
    Write_Address(NAND_Address_Table.NAND_Address.NAND_Address_A0_A7,
                  NAND_Address_Table.NAND_Address.NAND_Address_A8_A11,
                  NAND_Address_Table.NAND_Address.NAND_Address_A12_A19,
                  NAND_Address_Table.NAND_Address.NAND_Address_A20_A27,
                  NAND_Address_Table.NAND_Address.NAND_Address_A28,
                  0x05);
    delay(100);

    for(i = 0;i<DATA_NUM;i++)
         *pNAND_DATA_REG = *(psrc+i);

    Write_Command(0x10);
    while(!Wait_NAND_RADY());
    Write_Command(0x70);

    delay(5000);
    NAND_Data = *pNAND_DATA_REG;

    NAND_Finish();

    if(NAND_Data != 0xe0)
         return false;
    else
         return true;
}
```

Listing 2. Page Write Function

Listing 3 shows the code example for the page read code implemented.

```
bool  NF Page Read(unsigned int Block Address,unsigned short Page Address,unsigned
char *pdes)
{
    int i;

    Write_Command(0x00);
    Write_Address(NAND_Address_Table.NAND_Address.NAND_Address_A0_A7,
                  NAND_Address_Table.NAND_Address.NAND_Address_A8_A11,
                  NAND_Address_Table.NAND_Address.NAND_Address_A12_A19,
                  NAND_Address_Table.NAND_Address.NAND_Address_A20_A27,
                  NAND_Address_Table.NAND_Address.NAND_Address_A28,
                  0x04);
    Write_Command(0x30);
    while(!Wait_NAND_RADY());

    for(i = 0;i<DATA_NUM;i++)
    {
         delay(600);
         *(pdes+i) = *pNAND_DATA_REG;
    }
    NAND_Finish();

    return true;
}
```

Listing 3. Page Read Function

The basic NAND flash drivers are based on low-level operations, like `Write_Command()`, `Write_Address()`, `Wait_NAND_RADY()`, and `Set_CE()`. These operations handle the GPIO and address signals to manipulate the flash chip. Listing 4 is for the implementation of the write command.

```
void Write_Command(unsigned char command)
{
    Set_CE(0);
    Write_NAND_FLASH_Com(command);
}
```

Listing 4. Write_Command Function

Listing 5 is the implementation of the `Set_CE` signal function.

```
void Set_CE(bool state)
{
    if(!state)
         *pFIO_FLAG_D &= ~PF11;

    else
         *pFIO_FLAG_D |= PF11;
}
```

Listing 5. Set_CE Function

Listing 6 is for the implementation of the write flash command/address register function.

```
void Write_NAND_FLASH_Com(unsigned char Write_Data)
{
    *pNAND_COMMAND_REG = Write_Data;
}

void Write_NAND_FLASH_Addr(unsigned char Write_Data)
{
    *pNAND_ADDRESS_REG = Write_Data;
}
```

Listing 6. Write_NAND_FLASH_Com and Write_NAND_FLASH_Addr Functions

DMA can also be used to perform read/write operations from/to the flash memory device. In this case, the programmer should ensure that no other core or DMA accesses happen to any of the external memory banks. Any such accesses can cause spurious `RD#` and/or `WR#` pulses to the flash memory device, which will compromise the integrity of the interface and yield unexpected results.

CRO screenshots representing timing diagrams for this interface are shown in the following figures.

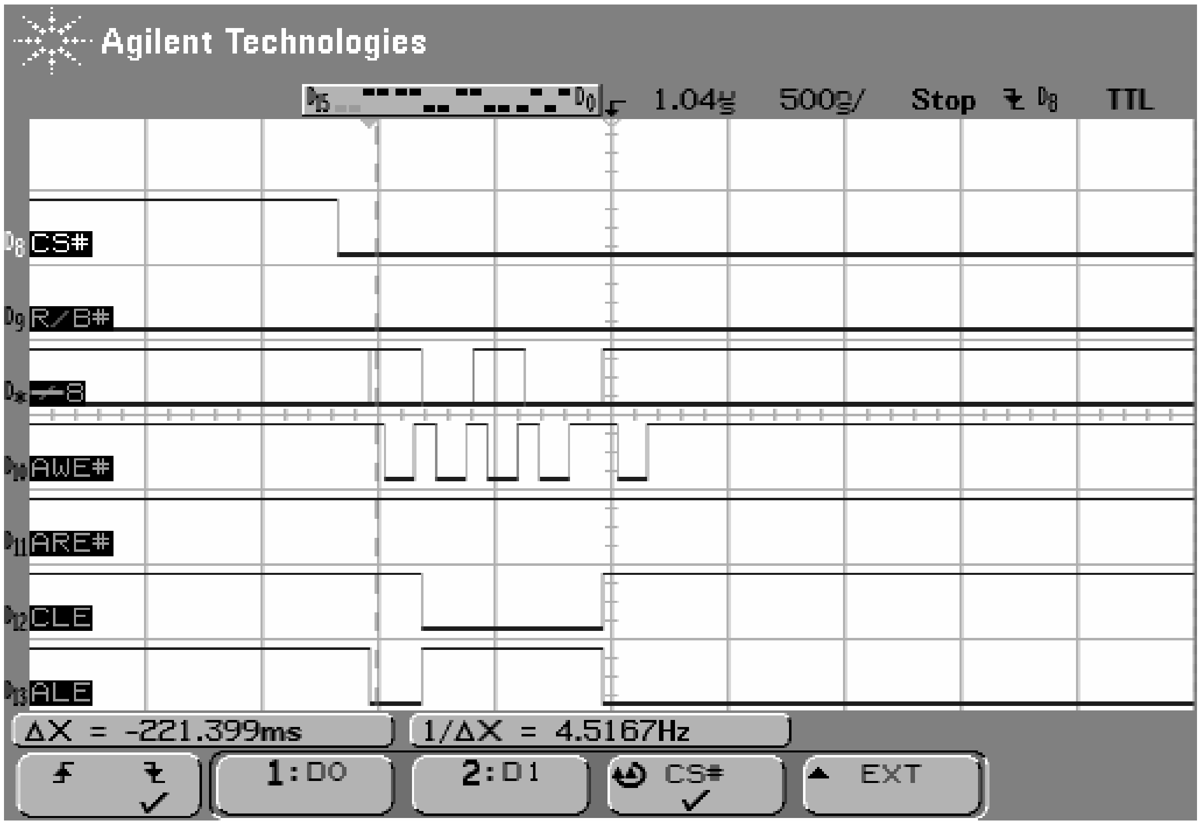

Figure 7: Erase Command

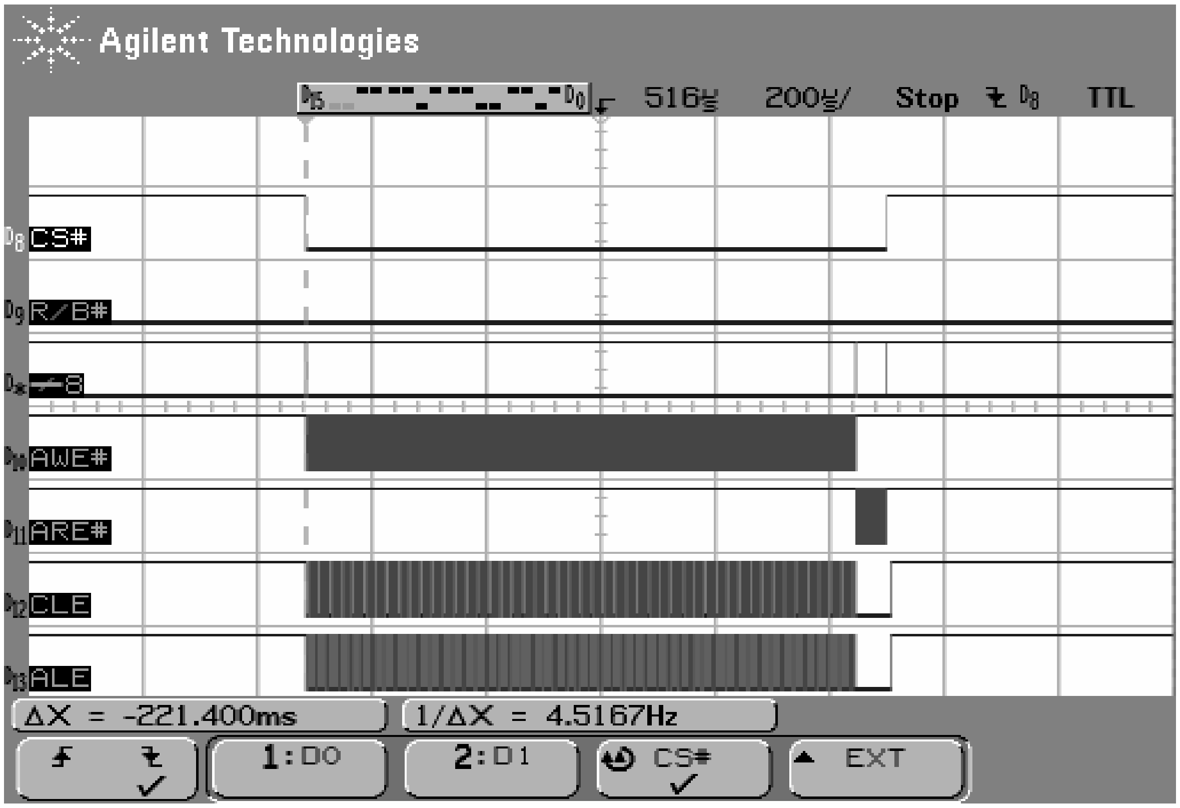

Figure 8: Write Block Command

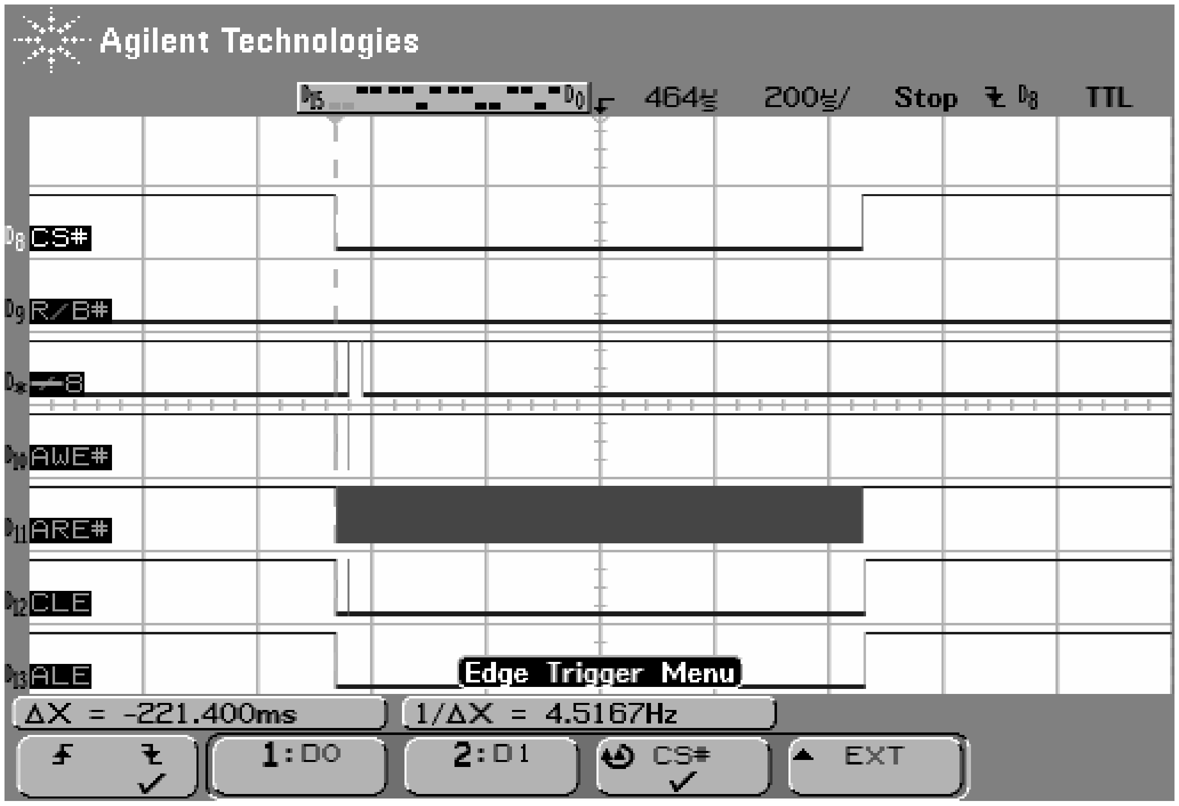

Figure 9: Read Command

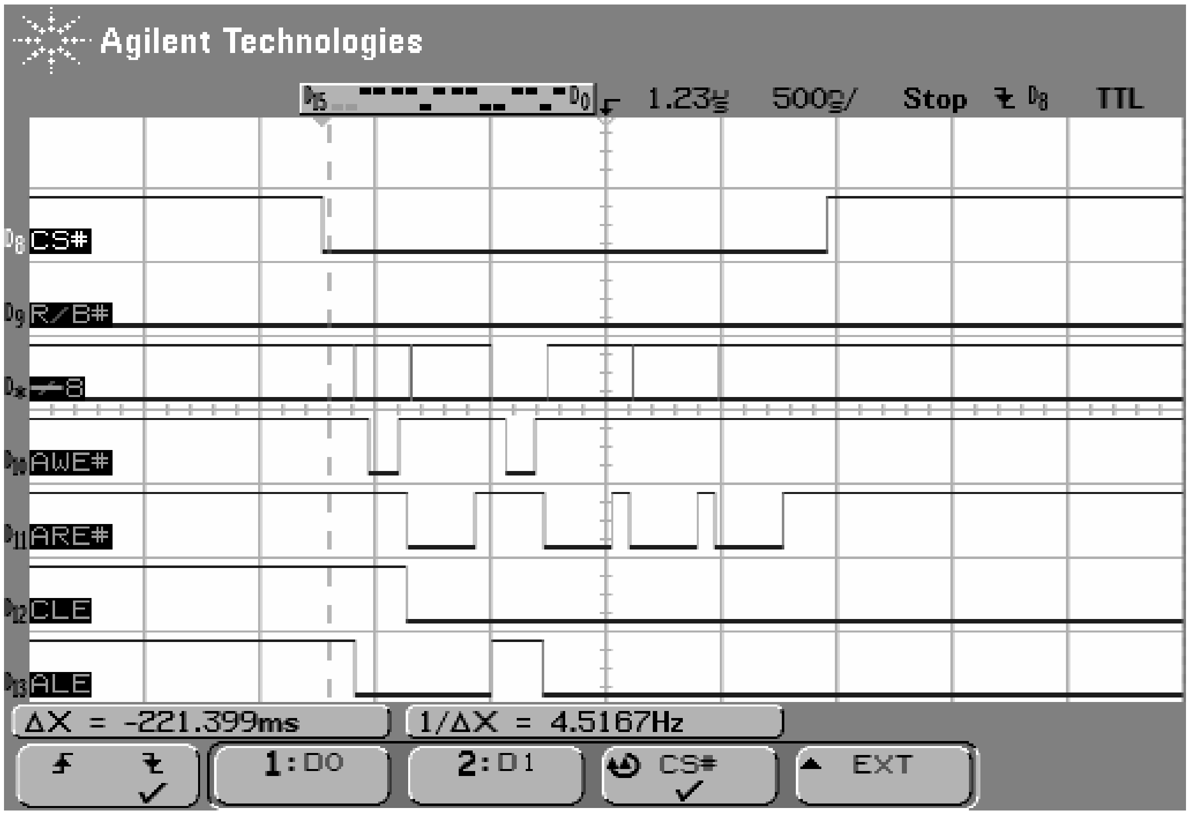

Figure 10: Read Chip ID Command

ECC for NAND Flash

Both the SLC and MLC NAND flashes require an Error Correction Code (ECC) to detect and correct bit errors. The Reed-Solomon algorithm is well-suited for this and has become the standard EEC method for both types of flash memory.

The RS encoder and decoder for 72-bit RS (464, 456, 4) code have been implemented and integrated into the NAND flash read/write drivers.

NAND Flash File Systems

NAND flash is optimized for file structures where each word does not need to be read, but instead provides that sectors of data can be moved to and from the media supporting a hard-drive-like repository structure for data storage to support file systems and allocation tables (FAT).

Unlike NOR flash, NAND flash demands a mechanism, such as TrueFFS, to be implemented to map bad blocks.

File systems in the industry, such as JFFS, JFFS2, FAT, and so on, are feasible for NAND flash. Additionally, the YAFFS (Yet Another Flash File System) is the only file system that has been designed specifically for use with NAND flash under any operating system. YAFFS uses journaling error correction and verification techniques to enhance robustness, and is designed to work within the constraints of—and exploit the features of—NAND flash in order to maximize performance. A future revision of this EE-note will include the implementation of a specific file system for NAND flash.

Conclusion

Although ADSP-BF53x Blackfin processors do not have an on-chip NAND controller, designers can easily connect SLC and MLC NAND flash memory devices to EBIU or GPIO. Both interfaces are seamless, thus requiring no additional external logic. Designers can refer to the code supplied with this EE-Note to develop other NAND devices drivers.

References

[1] *ADSP-BF533 Blackfin Embedded Processor Data Sheet*, Rev. C, May, 2006. Analog Devices, Inc.

[2] *MLC NAND Performance for Consumer Applications*, Toshiba America Electronic Components, Inc.

[3] *K9F2G08U0M 256M x 8 Bit / 512M x 8-Bit NAND Flash Memory Data Sheet*, Rev 1.2, October 31, 2005. Samsung Electronics, Inc.

[4] *TC58NVG3D4CTG 8 GBIT (1G × 8 BIT / 512M × 16 BIT) CMOS NAND E2PROM (Multi-Level-Cell) Data Sheet*, TOSHIBA Semiconductor Company.

[5] *Application Note for NAND Flash Memory*, Rev. 2.0, December 1999, Memory Product & Technology Division from Samsung Electronics, Inc.

[6] *ADSP-BF533 Blackfin Processor Hardware Reference*, Rev. 3.2, July 2006. Analog Devices, Inc.

[7] *Interfacing NAND Flash Memory with ADSP-2126x SHARC Processors (EE-279)*, Rev. 1, November 2005. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – November 10, 2006 by Ke Zhan & Daniel Zhao	Initial version