SPI / NAND Flash Programming

Version Control:

Version	Date	Description
1.00	18.10.2010	Initial

Table of Contents

1. Overview

The BDI2000/BDI3000 (BDI) firmware does not support direct programming of SPI (serial) and NAND flash. This is because programming these types of flash devices depends heavily on the integrated SPI/NAND controller. For every processor / flash combination, a different driver is necessary, thus implementing support in the BDI firmware is not practical. The designer of the hardware has the detailed information about his design and how he will use the attached SPI/NAND devices. Abatron is usually not privy to this level of detail. For example, is a SPI Data Flash used with 512 or 528 bytes pages? If the hardware is using 528 byte mode are the spare bytes used for data bytes or to store additional information (ECC, load addresses, ….)?
For NAND devices, how are bad blocks handled?

This document shows how developers can write their own flash-programming tool based on the BDI interface library. The example also provides insight as to why this tool is so dependant on the actual board and flash usage mode. For this reason, Abatron has no plans to add SPI/NAND flash programming support directly into the BDI firmware as there are just too many options to make it practical option.

An alternative way to program SPI/NAND flashes is using a boot loader (for example U-Boot) that supports your board. Often times, developers implement a boot loader that supports the SPI/NAND flash devices that are embedded on the custom hardware.

Here are a few simple examples of how one would use a boot loader to perform SPI/NAND flash programming. For more details about how to implement these methods, visit http://www.ultsol.com and click on the support link and then White Papers.

1. Build a version of the boot loader that can be loaded and execute out of RAM.
2. Use a BDI configuration that configures the SDRAM so it can be used to run the boot code.
3. Use the BDI probe to load and start the boot loader.
4. Use the boot loader commands to program the SPI/NAND flash.

2. SPI Data Flash Programming Example

2.1. Processor System

The board used for this example is the Freescale i.MX51 EVK. Here an Atmel Data Flash AT45DB321D is connected to the Enhanced Configurable Serial Peripheral Interface (eCSPI). Be aware that if the SPI flash is connected to the standard CSPI interface the example would look different because the standard CSPI interface on i.MX51 has different registers.

2.2. Programming Tool

The tool is first implemented only using the interface library without the use of any helper code that runs on the target. In a second step, simple helper code is loaded to the target for faster programming. A version is shown that programs the SPI flash only with code running on the target. The programming code that runs on the target is very similar to the code that runs on the host. Users may elect to first implement the code on their host tool, which tends to be slow. If necessary users may port the code to run down on the target which should provide better performance if implemented correctly.

The source code for Abatron's programming tool (host and target) can be found here:
ftp://94.230.212.16/bdigdb/tools/

The host tool is in the bdigdbifc.zip archive.
The target code is in the progimx51spi.zip archive.

In this example, there is no extensive error handling implemented in order to make the code more readable.

2.2.1. Host code

In addition to the SPI programming code, there are also additional commands implemented. Search for iMX51 in bdigdbifc.c to find the section of the file with the SPI programming code. The code should be straight forward but in order to fully understand it line-by-line, refer to the i.MX51 sCSPI and the AT45DB321D specifications.

The following commands are implemented:

```
"AT45ID                         read AT45 Manufacturer and device ID"
"AT45READ  <page>               read data of one flash page"
"AT45PROG  <page>               program one flash page"
"AT45ERASE <page>               erase one flash page"
"AT45PROGFILE <page> <file>     program a binary beginning with page"


BDI>at45id
Flash Manu/ID: ff1f2701
Flash Status : b4

BDI>at45erase 33
BDI>at45read 33
ffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffff
...
ffffffffffffffffffffffffffffffff

BDI>at45prog 33
BDI>at45read 33
000102030405060708090a0b0c0d0e0f
101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f
303132333435363738393a3b3c3d3e3f
404142434445464748494a4b4c4d4e4f
505152535455565758595a5b5c5d5e5f
...
```

2.2.2. Helper code

Programming speed can sometimes be improved by using helper code that runs on the target. In a first step, transferring the page data to the AT45 write buffer is supported by code running on the target. The following code is loaded to the target RAM and executed for every page.

```
typedef struct {
           BYTE*   data;
           UINT32  at45Cmd;
           UINT32  spiCtrl;
  volatile UINT32* pTxFifo;
  volatile UINT32* pControl;
  volatile UINT32* pStatus;
} ParamAT45T;


void AT45_BufferWrite(ParamAT45T* param)
{
  int       part;
  int       i;
  BYTE*     pData;
  DWORD     flashData;

  // write data to buffer 1
  pData = param->data;
  for (part = 0; part < 4; part++) {
    *param->pControl = param->spiCtrl;
    *param->pTxFifo  = param->at45Cmd;
    for (i = 0; i < 33; i++) {
      flashData  = (DWORD)(*pData++) << 24;
      flashData += (DWORD)(*pData++) << 16;
      flashData += (DWORD)(*pData++) << 8;
      flashData += (DWORD)(*pData++);
      *param->pTxFifo = flashData;
    } /* for */
    *param->pStatus  = (SPI_STAT_TC | SPI_STAT_RO);
    *param->pControl = param->spiCtrl | SPI_CTRL_XCH;
    param->at45Cmd  += 132;

    // wait for SPI transfer completed
    while ((*param->pStatus & SPI_STAT_TC) == 0);
  }
  __asm("bkpt");
}
```

The generated opcode for the above function is extracted via:

```
arm-eabi-objdump --disassemble helper.x > helper.txt
```

And then added as a word array to the host code.

```
static const DWORD AT45_BufferWrite[] = {
0xe5903000,       // a0008004:      ldr   r3, [r0]
0xe590a010,       // a0008008:      ldr   sl, [r0, #16]
...
0x1affffdd,       // a00080a4:      bne   a0008020 <AT45_BufferWrite+0x20>
0xe1200070,       // a00080a8:      bkpt  0x0000
};
```

Have a look at the AT45_PageProgFast() function.
This helper code gives some speed improvement for the BDI2000 but has no effect when using a BDI3000, which uses a faster 100 MB TCP/IP communication chip.

2.2.3. Target code

If your flash devices are large in size, the fastest programming method can be achieved when the complete programming algorithm runs in target SDRAM. In this mode the whole data for the flash is first loaded to target SDRAM. Then the programming algorithm is loaded and executed. In the progimx51spi.zip archive you find the code the programs the AT45 flash. Using the Telnet interface, users can run the following commands:

```
iMX51> load 0xa0100000 e:/temp/dump256k.bin bin
Loading e:/temp/dump256k.bin , please wait ....
Loading program file passed

iMX51> load 0 m:/temp/helper.x elf
Loading m:/temp/helper.x , please wait ....
- File offset 0x00008000 to address 0xA0008000 size 10812
- File offset 0x0000AA40 to address 0xA0012A40 size 2120
Loading program file passed

iMX51> mm 0xa0007000 0xa0100000
iMX51> mm 0xa0007004 127185
iMX51> mm 0xa0007008 1024
iMX51> go
- TARGET: core #0 has entered debug mode
iMX51>
```

Via the GDB monitor command users may also control the above sequence via the host tools. For more details, refer to the AT45_ProgRemote() function.

```
BDI>at45progfile
Reset ... passed
Loading e:/temp/dump256k.bin , please wait ....
Loading program file passed
Loading m:/temp/helper.x , please wait ....
File offset 0x00008000 to address 0xA0008000 size 10812
File offset 0x0000AA40 to address 0xA0012A40 size 2120
Loading program file passed
Programming ... passed
```