High-Volume Programming of NAND Flash

Kelly Hirsch
Chief Technologist
Data I/O Corporation

Programming NAND

- What is Programming?
- Why NAND?
- Programming and Time-to-Market
- Two stories---NOR and NAND
- Avoiding Pitfalls in Programming NAND
- Further Information

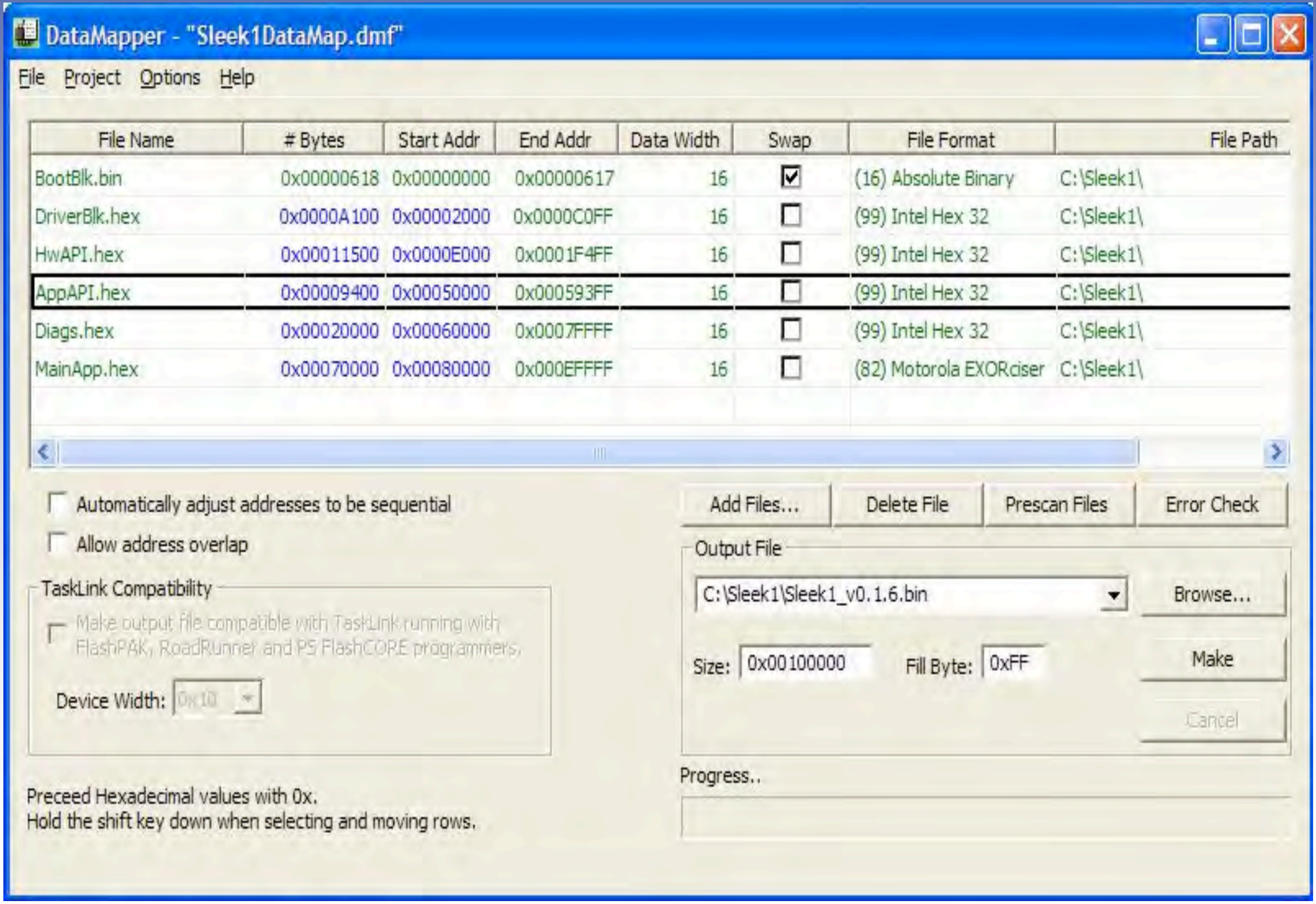
What is programming?

- "Programming" a device is the process of writing content to the non-volatile memory
  - Flash (NAND and NOR)
  - Flash cards (MMC, SD)
  - Flash based microcontrollers
  - Logic (FPGAs, CPLDs, etc.)
- Programming methods:
  - In-System Programming (ISP)
  - In socket
    - Manual
    - Automated
      - Off-line
      - On-line

Typical Programming Process

- Insertion test
- ID Check
- Blank check (erase if not blank)
- Write content (gang mode)
- Write dynamic data (serial number, etc.)
- Verify content against original image
- Apply sector protection (if any)

Image Creation Tool



DataMapper - "Sleek1DataMap.dmf"

File Name	# Bytes	Start Addr	End Addr	Data Width	Swap	File Format	File Path
BootBlk.bin	0x00000618	0x00000000	0x00000617	16	☑	(16) Absolute Binary	C:\Sleek1\
DriverBlk.hex	0x0000A100	0x00002000	0x0000C0FF	16	□	(99) Intel Hex 32	C:\Sleek1\
HwAPI.hex	0x00011500	0x0000E000	0x0001F4FF	16	□	(99) Intel Hex 32	C:\Sleek1\
AppAPI.hex	0x00009400	0x00050000	0x000593FF	16	□	(99) Intel Hex 32	C:\Sleek1\
Diags.hex	0x00020000	0x00060000	0x0007FFFF	16	□	(99) Intel Hex 32	C:\Sleek1\
MainApp.hex	0x00070000	0x00080000	0x000EFFFF	16	□	(82) Motorola EXORciser	C:\Sleek1\

□ Automatically adjust addresses to be sequential

□ Allow address overlap

TaskLink Compatibility

□ Make output file compatible with TaskLink running with FlashPAK, RoadRunner and PS FlashCORE programmers.

Device Width: 0x10

Output File

C:\Sleek1\Sleek1_v0.1.6.bin

Size: 0x00100000 Fill Byte: 0xFF

Progress..

Preceed Hexadecimal values with 0x.
Hold the shift key down when selecting and moving rows.

Programming Methods

Engineering, R&D

FlashPAK

Just in time Programming

Road Runner Series

Off-line Programming

PS Series

In-System Programming

Image Writer

Desktop Automated Programming

FLX500

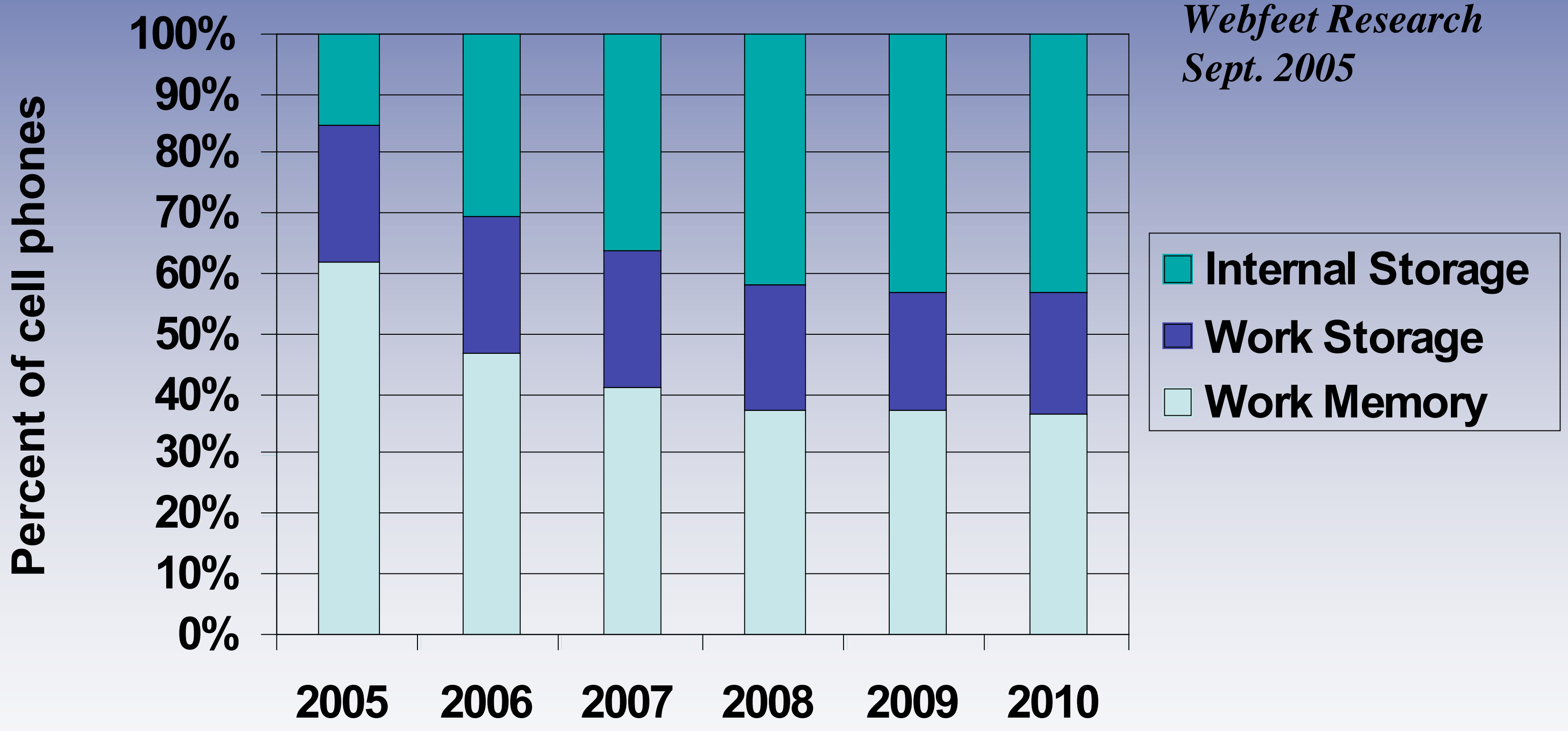
Why do I care about programming?

- Time from design to production is critical for short “life time” products:
  - 1 month delay = 15% of potential revenue
  - 3 month delay = 50% of potential revenue
- Sub-standard programming methods cost semiconductor companies millions of dollars per line (due to RMAs)

Why use NAND (vs. NOR)?

- NAND is less expensive than NOR
- NAND can program faster than NOR
- NAND has more endurance than NOR
- MLC NAND is available in large densities

Cell Phone Internal Memory Forecast



Webfeet Research
Sept. 2005

Internal Storage = NAND (only) shadowed into LP DRAM

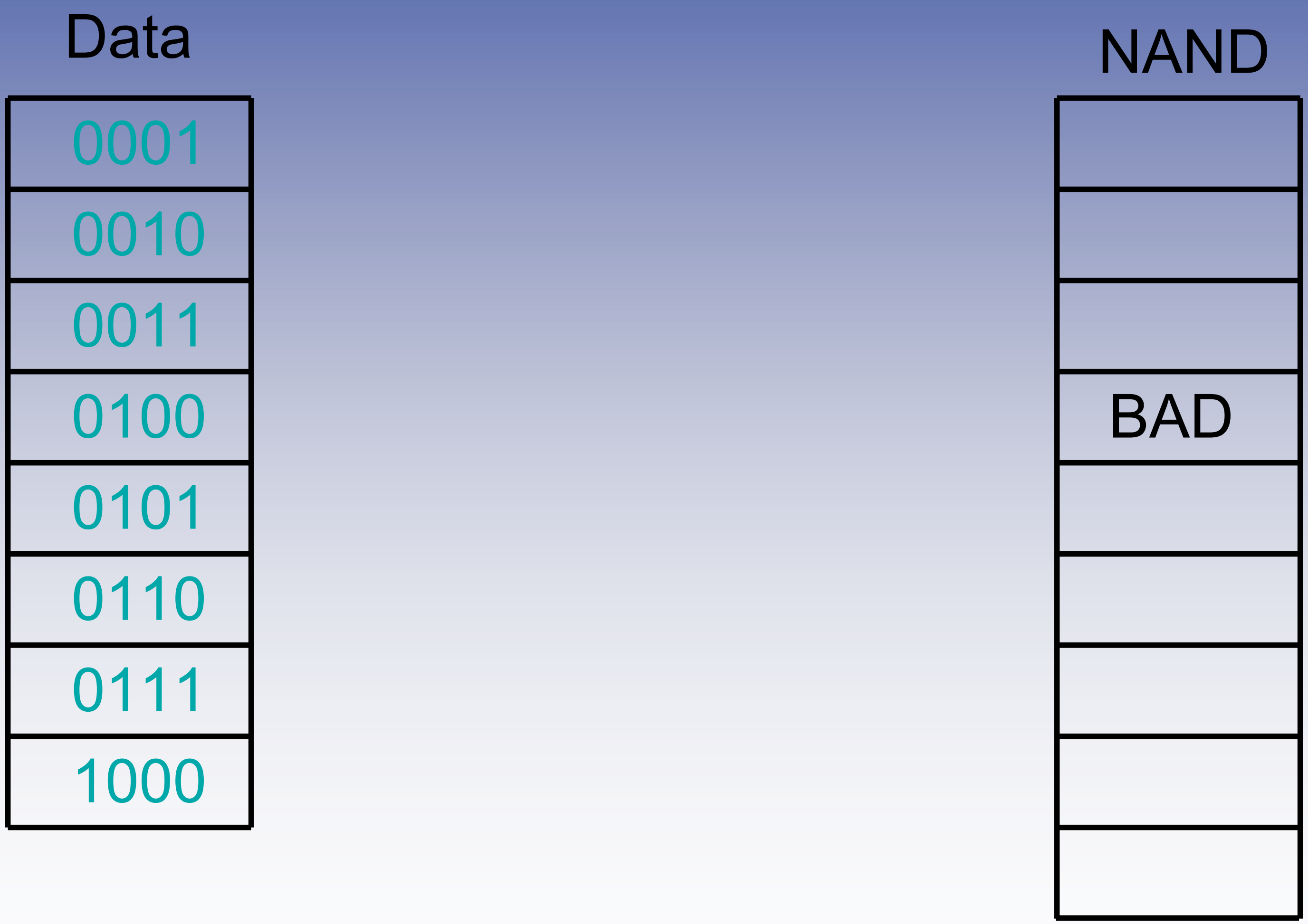
So what’s wrong with NAND?

- NAND has to be “shadowed” into volatile memory (pSRAM, LPDRAM, etc.) before you can use it in an eXecute-in-Place (XiP) application.
- NAND can have defective memory cells; typically NAND is delivered from the factory with entire blocks of memory marked as “bad”.
- You will permanently damage a NAND device if you erase the entire device.
- Writing and erasing NAND can cause a block to go bad at any time; statistically significant at over 1000 cycles; ECC required.
- MLC can have more “bit-flips” than SLC; more ECC bits required.
- Programming NAND flash is much more complicated!

What’s needed to program NAND?

- ✓ ▪ A Bad Block Scheme
  - to identify defective blocks of memory and know what to do when they are found.
- ✓ ▪ Error Detection/Correction Code (EDC, ECC)
  - to verify the data each time you read it, detect corrupted bits and recover the original data.
- ✗ ▪ Wear leveling
  - where frequent read/writes are required, a program is used to keep track of the used locations and move this activity around to different physical locations.
- ✗ ▪ Garbage collection
  - where previous versions of data (that have been updated and placed somewhere else) are erased so that a complete erase block is available to the system for more storage.

Bad-block Scheme Example



Data

0001
0010
0011
0100
0101
0110
0111
1000

NAND

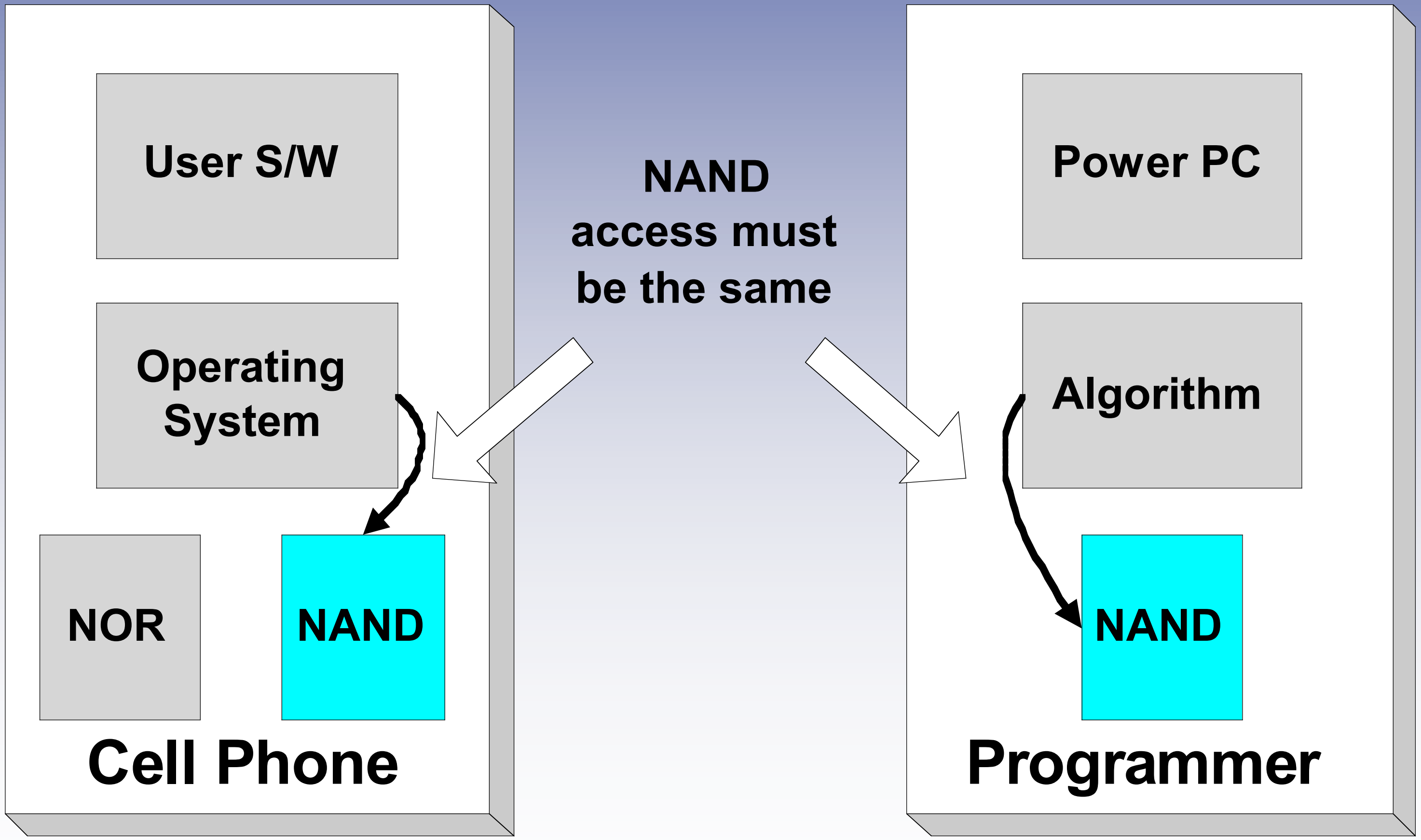
BAD

But there are many schemes…

Logical Block Skip
4 Block Skip
File System Map
GSL Skip
MODE2, MODE3
Pantech BBM
Qualcomm MSM6100
MSM6100 EFS
Reserved Block Area
Samsung GBBM
SAM GBBM DWSwap
Samsung WinCE
Skip Bad Blocks
Thales Navigation
Datalight Flash FX
Unistore V1.8

Why?

Matching the Application



Two stories on *time-to-market*

The following stories are true---names have been removed to protect the "innocent".

First…programming NOR

NOR Process	*Elapsed Time*
• Manufacturing manager orders adapters and algorithm for programming equipment.	*Day 0*
• Receive adapter and algorithm.	*Day 14*
• Receive binary image from design team.	*Day 15*
• Pilot run to validate set-up.	*Day 16*
• Ready for full-rate production.	*Day 17*

Second…programming NAND

NAND Process	*Elapsed Time*
Manufacturing manager orders adapters and algorithm for programming equipment.	*Day 0*
Data I/O informs manager that NAND requires bad-block scheme. Asks for contact from design team.	*Day 1*
Manager contacts design team and asks them to contact Data I/O.	*Day 2*
Design team doesn’t understand why they have to contact Data I/O. No action for a week.	*Day 9*
Data I/O sends detailed explanation to design lead.	*Day 10*

Programming NAND (cont.)

NAND Process (cont.)	*Elapsed Time*
• Design lead informs Data I/O that file-system is supplied by third party; sends contact information.	*Day 11*
• Data I/O contacts file system vendor; explains problem.	*Day 14*
• File system vendor requires NDA; executes NDA.	*Day 19*
• Data I/O interviews file system vendor; documents bad-block scheme requirements.	*Day 22*
• Data I/O writes code for bad-block scheme and sends beta to design lead.	*Day 27*

Programming NAND (cont.)

NAND Process (cont.)	*Elapsed Time*
• Design team tests algorithm but phone won't boot. Begin troubleshooting.	*Day 29*
• Design team asks file system vendor for help.	*Day 31*
• File system vendor informs design team of change in driver; updates bad-block scheme document.	*Day 33*
• Data I/O modifies scheme and sends new algorithm to design team.	*Day 35*
• Design team tests algorithm; phone boots OK! Design team informs production manager.	*Day 37*

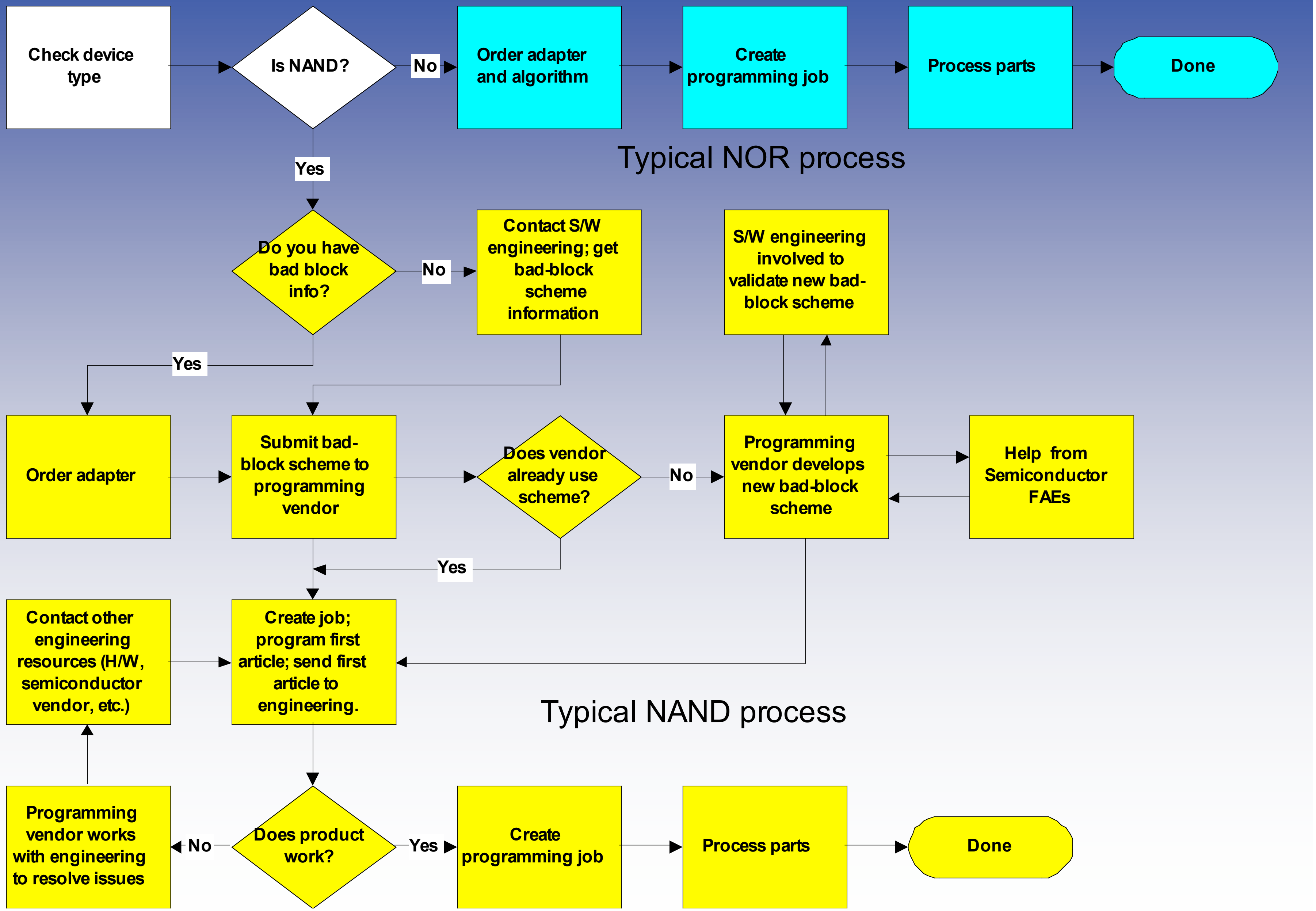
Programming NAND (cont.)

NAND Process (cont.)	*Elapsed Time*
Production manager receives adapter and validated algorithm from Data I/O.	*Day 39*
Receives binary image from design team.	*Day 40*
Pilot run to validate set-up.	*Day 42*
Pilot run fails due to checksum problem.	*Day 43*
Data I/O helps to troubleshoot problem; ECC data is added to image. Manufacturing updates process documentation.	*Day 45*
Now pilot run is successful!	*Day 46*
Ready for full-rate production.	*Day 47*

What went wrong?

Neither the design team nor the manufacturing team realized that, with NAND devices, the programming algorithm must comprehend more than just the device characteristics; it also needs to “know” how the NAND device is accessed in the application.

Programming NAND vs. NOR



Check device type

Is NAND?

No

Order adapter and algorithm

Create programming job

Process parts

Done

Typical NOR process

Yes

Do you have bad block info?

No

Contact S/W engineering; get bad-block scheme information

S/W engineering involved to validate new bad-block scheme

Yes

Order adapter

Submit bad-block scheme to programming vendor

Does vendor already use scheme?

No

Programming vendor develops new bad-block scheme

Help from Semiconductor FAEs

Yes

Contact other engineering resources (H/W, semiconductor vendor, etc.)

Create job; program first article; send first article to engineering.

Typical NAND process

Programming vendor works with engineering to resolve issues

No

Does product work?

Yes

Create programming job

Process parts

Done

Avoid Common Pitfalls

- If you use ECC (in the spare area), the checksum for the image will change. Provide manufacturing with the right checksum in the "version control documentation".
- If you create a bad-block scheme that requires several good blocks in certain area, the job yield will decrease.
- Is one bit error during verify a failure, or does your application allow this? Define this before pilot run.
- Erase NAND; don't blank check. This is much faster (but be sure to use bad-block scheme during erase).
- Make sure bad-block scheme documentation is complete:
  - Mapping tables, partition information, dynamic fields, etc.?
  - Document all the parameters for manufacturing!

Do you know all the parameters?

Bad Block Handling Type = “Qualcomm MSM6100 EFS”

Spare Area = “Enabled” if the image contains the spare area data and that data should be programmed into the device. “Disabled” if the image does not contain the spare area data and only the main array should be programmed. “ECC” Calculates the ECC data and programs it into the spare area.

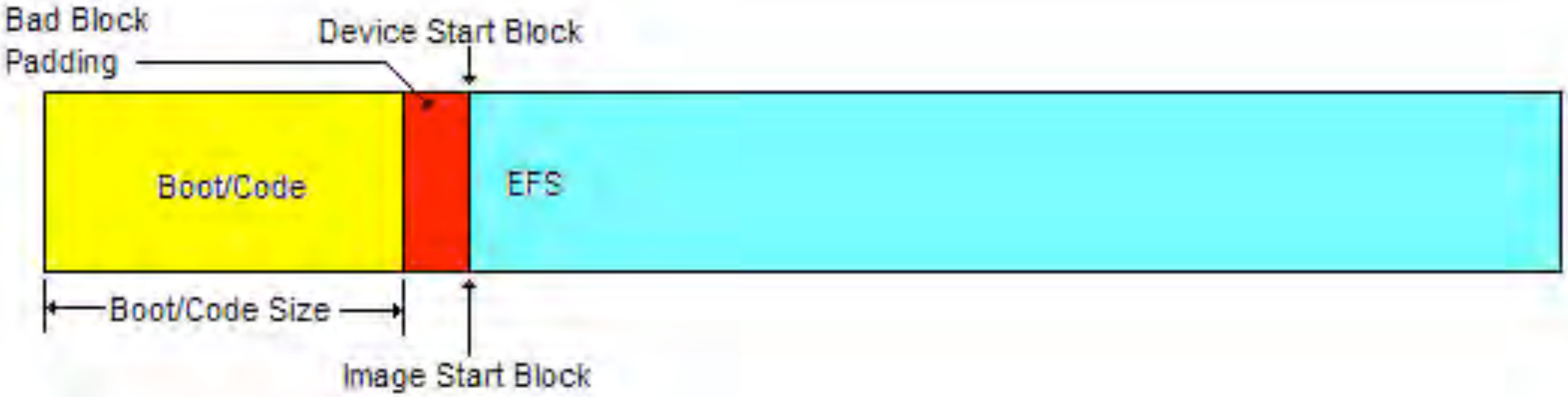
EFS: Image Start Block This is the beginning block of the File System. All blocks in the image after this are considered to be in the File System and will be placed after the start block.

EFS: Device Start Block This is the block number to place the File System on the device.

EFS: Boot/Code Partition Size This variable defines the size of the Boot/Code (first) partition size in blocks.

Required good block area: Start block = “0” This will require the entered block to be a valid block

Required good block area: Number of blocks = “0” This will be the total number of blocks required to be valid after the start block.



Do you know all the parameters?

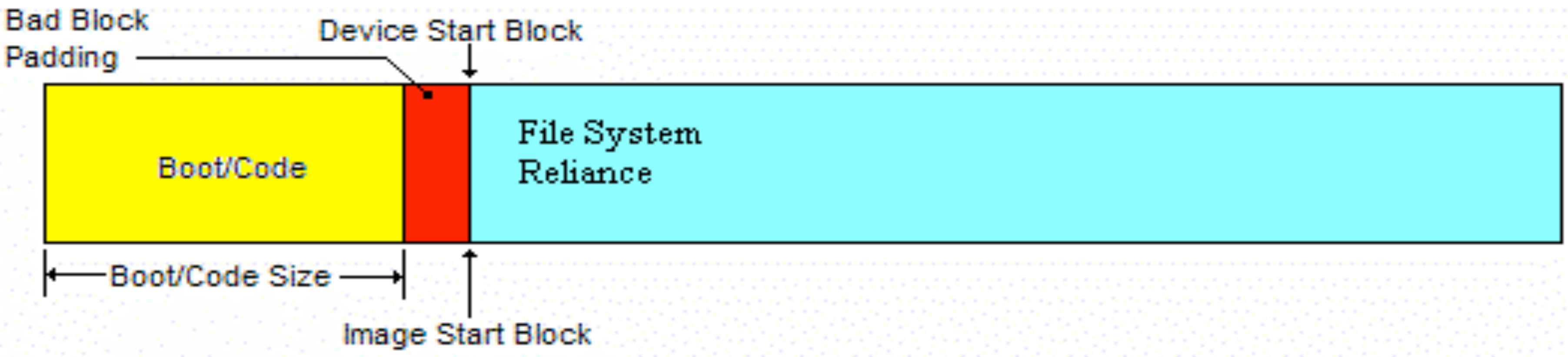
Bad Block Handling Type = “Datalight FlashFX Pro”

Spare Area = Always “Enabled”, the image contains spare area data and that data must be programmed into the device. Format of the spare area is dependent upon the specific NAND configuration. “ECC” Calculates the ECC data and programs it into the spare area.

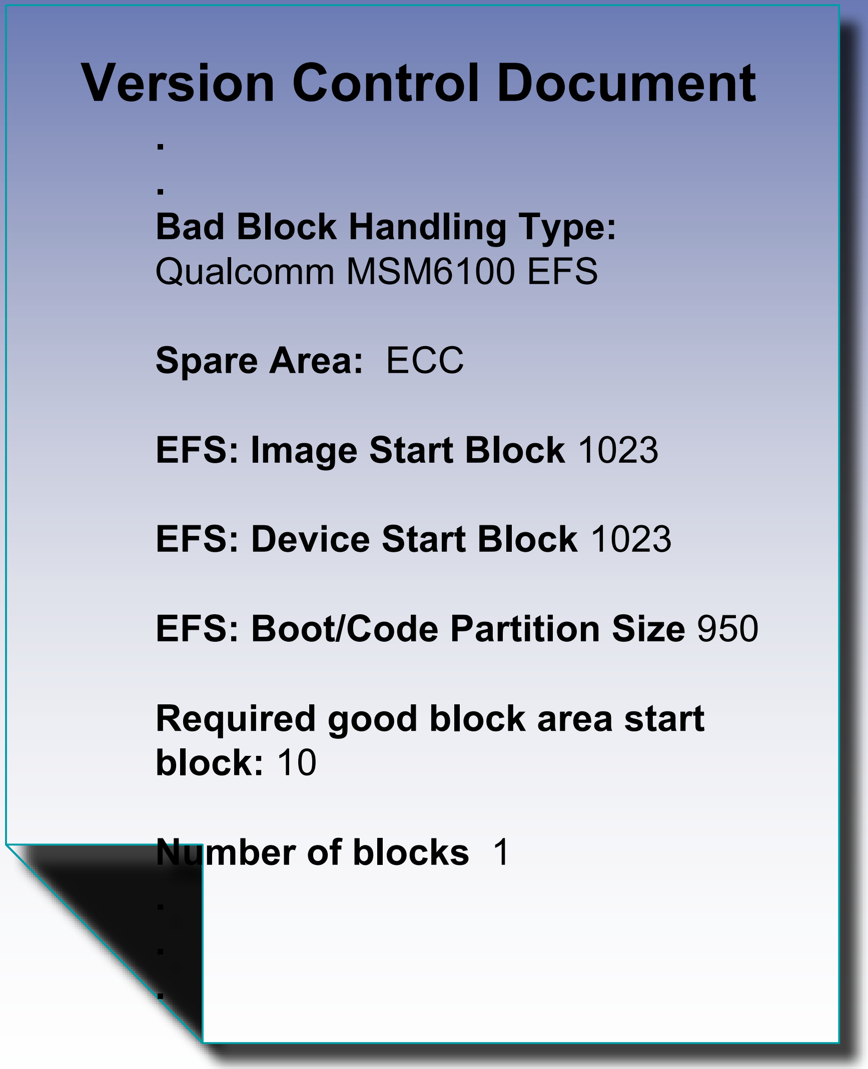
Image Start Block This is the beginning block of the File System. All blocks in the image after this are considered to be in the File System and will be placed after the start block.

Device Start Block This is the block number to place the File System on the device.

Boot/Code Partition Size This variable defines the size of the Boot/Code (first) partition size in blocks.



Include bad-block parameters in your Version Control Document.



Version Control Document

.

.

Bad Block Handling Type:
Qualcomm MSM6100 EFS

Spare Area: ECC

EFS: Image Start Block 1023

EFS: Device Start Block 1023

EFS: Boot/Code Partition Size 950

Required good block area start block: 10

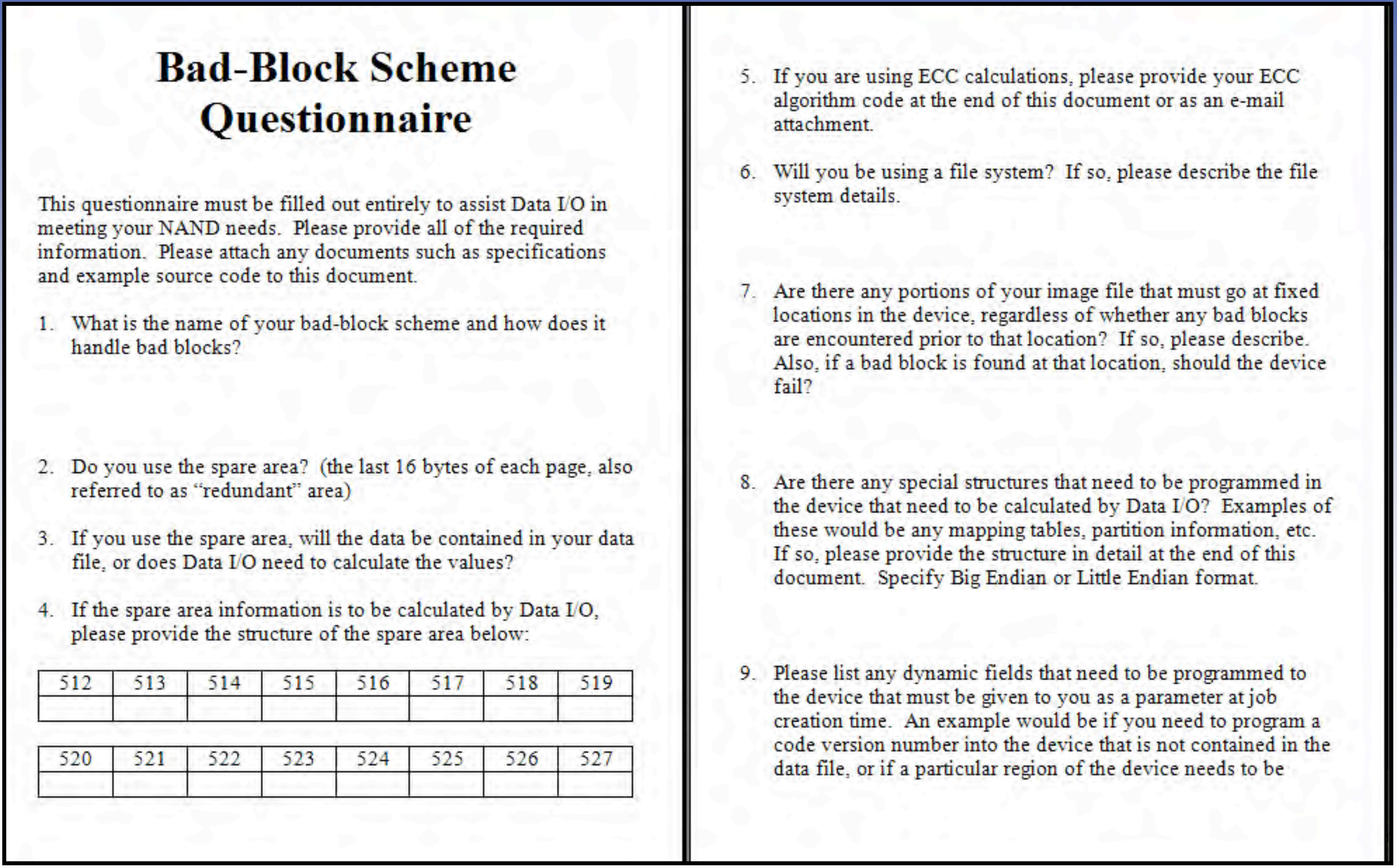
Number of blocks 1

.

.

.

Data I/O's Questionnaire



Bad-Block Scheme Questionnaire

This questionnaire must be filled out entirely to assist Data I/O in meeting your NAND needs. Please provide all of the required information. Please attach any documents such as specifications and example source code to this document.

1. What is the name of your bad-block scheme and how does it handle bad blocks?

2. Do you use the spare area? (the last 16 bytes of each page, also referred to as "redundant" area)

3. If you use the spare area, will the data be contained in your data file, or does Data I/O need to calculate the values?

4. If the spare area information is to be calculated by Data I/O, please provide the structure of the spare area below:

512	513	514	515	516	517	518	519

520	521	522	523	524	525	526	527

5. If you are using ECC calculations, please provide your ECC algorithm code at the end of this document or as an e-mail attachment.

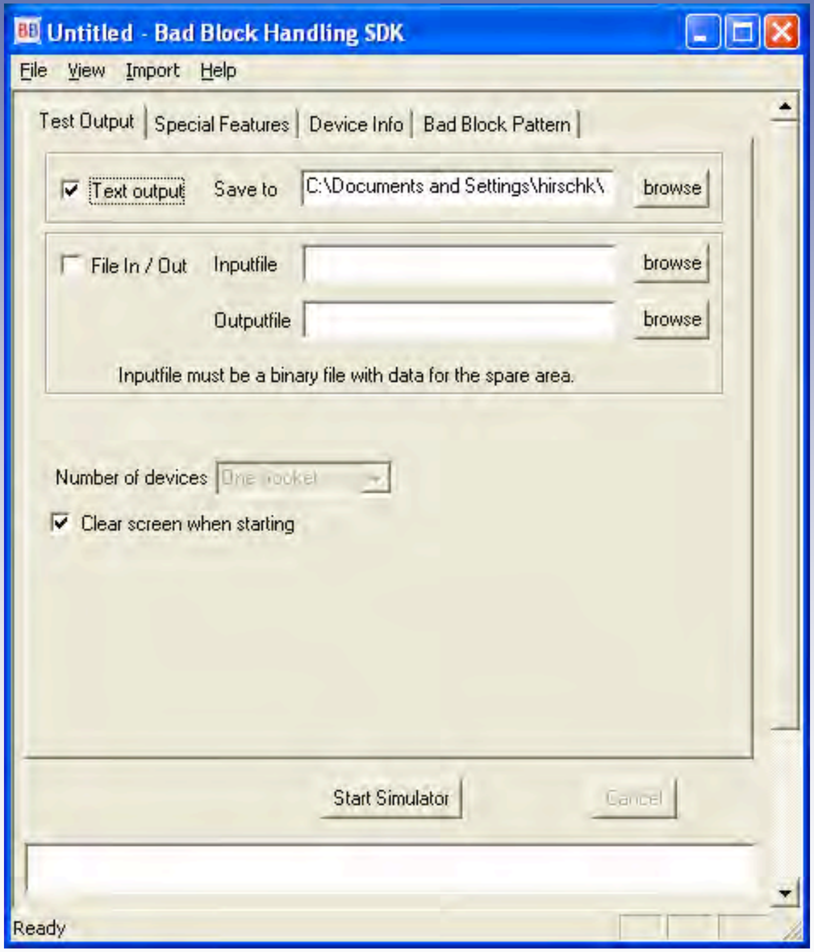
6. Will you be using a file system? If so, please describe the file system details.

7. Are there any portions of your image file that must go at fixed locations in the device, regardless of whether any bad blocks are encountered prior to that location? If so, please describe. Also, if a bad block is found at that location, should the device fail?

8. Are there any special structures that need to be programmed in the device that need to be calculated by Data I/O? Examples of these would be any mapping tables, partition information, etc. If so, please provide the structure in detail at the end of this document. Specify Big Endian or Little Endian format.

9. Please list any dynamic fields that need to be programmed to the device that must be given to you as a parameter at job creation time. An example would be if you need to program a code version number into the device that is not contained in the data file, or if a particular region of the device needs to be

SDK for NAND Bad Block Scheme



☑ Text output — Save to: C:\Documents and Settings\hirschk\

□ File In / Out — Inputfile: — Outputfile:

Inputfile must be a binary file with data for the spare area.

Number of devices: One socket

☑ Clear screen when starting

Ready

- Visual Studio application for simulation of bad-block handling methods
- Distributed with source code which integrates into Data I/O algorithms
- Can be used with input and output text files to simulate effects of bad blocks during programming of image file
- Your code is integrated into algorithm and tested by Data I/O.

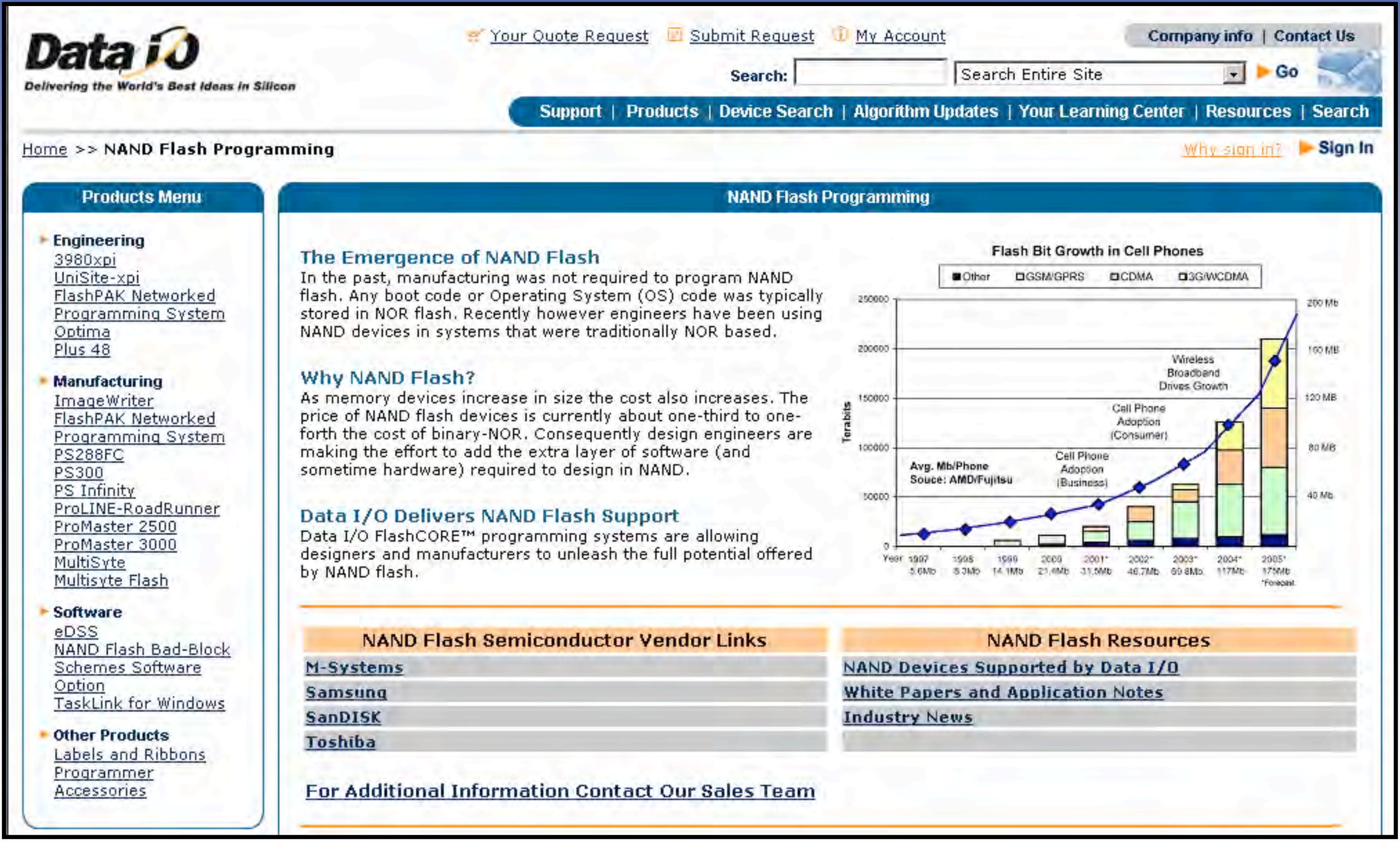
Final Recommendations

- Ask semiconductor FAEs to bring in programming solutions vendor
- Use Data I/O’s SDK to validate bad-block scheme during design process
- Create bad-block scheme document as part of design process
- Document dynamic fields for each model as part of version control documentation

For further reading…

- Optimizing a Flash Media Manager for NAND Flash Imaging (Datalight)
- Two Technologies Compared: NOR vs. NAND (M-Systems)
- Programming NAND Devices (Data I/O Corporation)

Data I/O’s “NAND Portal”



Data I/O
Delivering the World's Best Ideas in Silicon

Your Quote Request | Submit Request | My Account | Company info | Contact Us

Search: Search Entire Site ▾ Go

Support | Products | Device Search | Algorithm Updates | Your Learning Center | Resources | Search

Home >> NAND Flash Programming

Why sign in? Sign In

Products Menu

- **Engineering**
  - 3980xpi
  - UniSite-xpi
  - FlashPAK Networked Programming System
  - Optima
  - Plus 48
- **Manufacturing**
  - ImageWriter
  - FlashPAK Networked Programming System
  - PS288FC
  - PS300
  - PS Infinity
  - ProLINE-RoadRunner
  - ProMaster 2500
  - ProMaster 3000
  - MultiSyte
  - Multisyte Flash
- **Software**
  - eDSS
  - NAND Flash Bad-Block Schemes Software Option
  - TaskLink for Windows
- **Other Products**
  - Labels and Ribbons
  - Programmer Accessories

NAND Flash Programming

The Emergence of NAND Flash

In the past, manufacturing was not required to program NAND flash. Any boot code or Operating System (OS) code was typically stored in NOR flash. Recently however engineers have been using NAND devices in systems that were traditionally NOR based.

Why NAND Flash?

As memory devices increase in size the cost also increases. The price of NAND flash devices is currently about one-third to one-forth the cost of binary-NOR. Consequently design engineers are making the effort to add the extra layer of software (and sometime hardware) required to design in NAND.

Data I/O Delivers NAND Flash Support

Data I/O FlashCORE™ programming systems are allowing designers and manufacturers to unleash the full potential offered by NAND flash.

Flash Bit Growth in Cell Phones

NAND Flash Semiconductor Vendor Links	NAND Flash Resources
M-Systems	NAND Devices Supported by Data I/O
Samsung	White Papers and Application Notes
SanDISK	Industry News
Toshiba	

For Additional Information Contact Our Sales Team

http://www.dataio.com/NAND/default.asp

Thank you!

Kelly Hirsch
Chief Technologist
Data I/O Corporation
425-867-6246
hirschk@dataio.com
www.dataio.com