Datalight
Software for Trustworthy Data on Successful Devices

21520 30th Drive SE, Suite 110
Bothell, WA 98021
tel +1.425.951.8086
fax +1.425.951.8094
sales@datalight.com

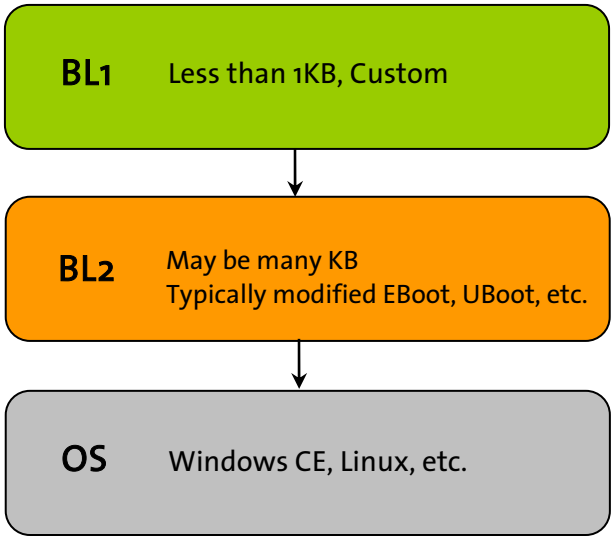
Leveraging Processor-supplied SRAM to Facilitate OneNAND Bootloaders

NAND flash memory densities for device-resident memory are increasing - and the price decreasing - with designers desiring to remove NOR flash devices from their design, and instead boot their processors from less expensive NAND. Due to the architecture of NAND flash memory devices, NAND flash cannot be used to boot processors unless the processor or the system contains some additional enabling logic.

In the case of Samsung OneNAND™, this logic resides within the OneNAND part itself. Common OneNAND parts, such as the KFG1G16Q2A, supply a 1KB BootRAM buffer that is automatically loaded from the internal NAND array at power on. See section 7.2 Boot Sequence, of the OneNAND KFG1G16Q2A datasheet for additional information[i].

OneNAND Boatloader Limitations and Implementation

Many system designers will immediately note that 1KB is very small in size when compared to the size of bootloaders used to boot many common operating systems. In operation, the 1KB BootRAM buffer is retrieved by the OneNAND device at power-on from the internal NAND array, using a developer-supplied 1KB bootloader referred to as the 'phase one' bootloader, or BL1. Designers use a number of techniques to facilitate booting the full operating system from the 1KB BL1. One technique clearly identified in the OneNAND documentation is the use of a 'phase two' bootloader, or BL2, whereby BL1 boots only the bare minimum of logic required to launch a more elaborate BL2.



Common bootloader environments such as Microsoft® Windows® CE Eboot[ii] and U-Boot[iii] (often used to launch Linux) were not designed to run from 1KB, but can be integrated as BL2. The table below shows the sizes of several configurations of common bootloaders.

Table 1. Startup Code Size Examples

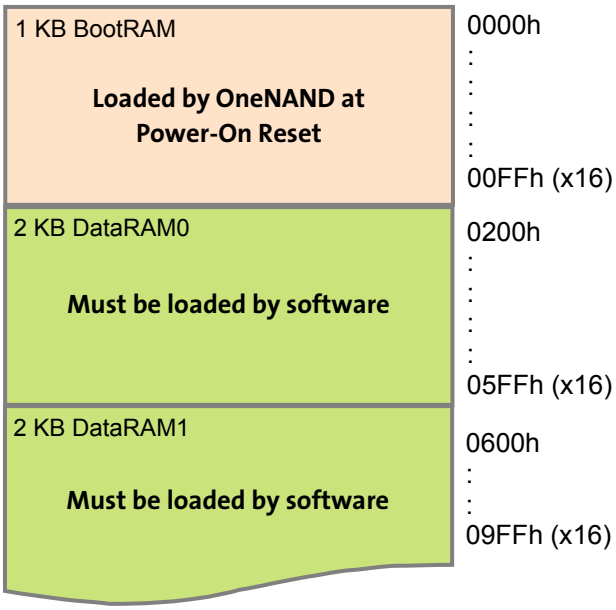
Bootloader	Board	Size	Startup Code Size (does not include data, stack, or heap)
Eboot	OMAP 5912 OSK, from Spectrum Digital (Mistral Software BSP)	84,300	1580
U-Boot	OMAP 5912 OSK from Spectrum Digital (U-Boot on CD from Spectrum Digital)	91,244	1676
Eboot	PHYTEC phyCORE®-XScale/PXA270CE	86,016	Unknown
U-Boot	PHYTEC phyCORE®-XScale/PXA270CE	124,636	Unknown
Eboot	Freescale i.MX31 ADS	124,296	1792
U-Boot	Freescale i.MX31 ADS	Unknown	Unknown

Design of a Size-Restricted Phase-One (BL1) Bootloader

Designers need to perform a minimum of operations in the 1KB BL1 in order to meet the 1KB requirement. These operations are similar to what is typically done in a larger bootloader's startup code. One definition of startup code is: *that code which is executed before entering a 'C' code environment*. In other words, it is the code that is written in assembly language and executes *prior* to the main entry point in the 'C' code that makes up the remainder of the bootloader functionality.

To a designer, this might at first seem to represent the 'bare minimum' of code that is to be executed in BL1, but it actually contains a comparatively large amount of code that is not required in order to load and jump to a more elaborate BL2. Additionally, if some code in the 1KB space is dedicated to reading additional sectors from the OneNAND internal array, a total of 5KB of code can be available for booting. For some bootloaders, however, the startup code exceeds 1KB in size, and can also exceed 5KB in size. See Table 1 above for examples of startup code sizes. Since this code will not fit as-is, a modified approach is needed.

Diagram 1. OneNAND BootRam and DataRAM



One method that seems immediately promising is to use very simplified startup code that is less elaborate than that provided by other bootloaders. Bootloader code is notoriously difficult to debug and often must be analyzed by inspection. Reducing the size and complexity of the code limits the opportunity for problems and reduces the amount of code involved when debugging, but also exacerbates the problem because the bootloader debug environment is typically limited to comparatively low level and primitive debugging methods.

For example, the startup code from U-Boot for the Texas Instruments OMAP5912 Starter Kit (OSK)[iv] performs the following operations:

1. Sets the processor into SVC32 mode
2. Flush the caches
3. Disable the MMU and Caches
4. Masks all IRQs
5. Sets up the clocks and PLLs
6. Turns off the watchdog timer
7. Programatically determines if executing from SDRAM, and if not, set up the SDRAM controller
8. Configure the peripheral bus interfaces
9. Relocate itself to RAM
10. Set up the processor stack

Other typical procedures found in typical bootloader startup code include configuration of the processor pins, and configuration of the processor stack for the various processor modes. By incorporating these operations in the startup code, the bootloader can be very robust and can address a larger number of systems whose peripheral sets and memory components may differ dramatically from one another - even the processors themselves may differ slightly in errata. Some operating systems expect or require certain system components (such as the processor pins) to be configured prior to the hand-off of execution from the bootloader to the operating system.

It is also important to note that none of the startup code mentioned above is dedicated to issuing commands and moving data from OneNAND, yet at least some of the 1KB bootloader code must be dedicated to this. This code must fit in the 1KB *in addition* to the minimal startup code.

Much of the typical bootloader startup code can be easily eliminated by assuming (and therefore requiring) that the 1KB BL1 code always and only starts in response to a power-on reset. Much of the list above can therefore be eliminated in BL1 by using the processor defaults, if they are acceptable in the particular system.

Part of the startup code is typically used to initialize SDRAM controllers, and may even include support for several different varieties of SDRAM. If we can eliminate that need, we can free up some instructions for booting, as well as eliminate a critical initialization element that could otherwise fail, or must be changed to accommodate boards populated with different SDRAM. We can eliminate that need by using SRAM that is available on some common processors.

Many common microprocessors have SRAM available at power-on in the default start-up state of the processor at power-on reset. For example, the Texas Instruments OMAP5912 has 256KB of SRAM available[v], the Intel® PXA270 has 256KB (in 4 64MB banks)[vi], and the Freescale™ i.MX31 includes 16KB of SRAM[vii]. Many other processors not mentioned here have SRAM available as well. It's also important to note that almost all have some form of instruction cache that may theoretically be used for this purpose, but that the use of cache for this purpose is more complex and beyond the scope of this paper. Often, the processor design has established the SRAM as a general-purpose resource, but may appear to imply a particular application. For example, the TI OMAP 5912 documentation often refers to the 256KB SRAM in the context of a display frame buffer, and the Freescale i.MX31 documentation states that the 16KB buffer can be used for audio streaming data to avoid external memory access.

Care must be taken when reading the processor documentation to ensure that this special purpose does not interfere with a bootloader's use of the same memory. Additionally, the bootloader implementations available in reference design software and BSP source code can be reviewed, as many bootloaders such as Eboot and U-Boot take advantage of this existing SRAM at boot time. For example, several bootloaders and operating system startup code implementations use these SRAM buffers for the initial stack, and even copy some code into them for initial execution. Similarly, if the bootloader copies itself into this region as part of the normal course of operation, it will be important to verify that this can still work when executing directly from that region. For example, if the existing bootloader zeros the RAM before copying itself into the destination, then if the source and destination are the same, the RAM will be cleared.

Datalight Phase-One Bootloader Implementation

By using the SRAM, as well as other techniques, Datalight developed a phase-one bootloader that fits in less than 256 bytes for the TI OMAP5912 OSK. This bootloader has been used to boot an unmodified U-Boot as well as a customized Eboot (customized to use Datalight OneBoot + File to boot Windows CE 4.2 from OneNAND).

The OneBoot BL1 addresses the list of startup operations above in the following manner:

Table 2. OneBoot BL1 vs Full-featured Bootloader

Initialization Step	Full-featured Bootloader	OneBoot BL1
Processor Mode	Sets processor into SVC32 mode	Not required/uses default
Cache Flush	Flushes the Cache	Not required/uses default
MMU	Disables MMU	Not required/uses default
Mask IRQs	Masks IRQs	Not required/uses default
Clocks	Sets up Clocks and PLLs	Not required/uses default
Watchdog timer	Disables Watchdog	Not required/uses default
SDRAM Controller	Configures EMIFF/SDRAM Controller	Not required – does not use SDRAM
Peripheral Bus	Configures EMIFS	Not required/uses default
Relocates BL Code and Data	Relocates to destination if necessary	Executes in place from OneNAND BootRAM
Stack Setup	Configures processor stack for each CPU mode	Does not use stack

With a footprint of less than 256 bytes, the remaining space in the 1KB BootRAM could be used to perform one or more of the above operations, and the additional available 4KB DataRAM could also be used with some further modification of the code. Additionally, performance or security features can be implemented in the remainder of the 1KB or additional 4KB, such as enabling the bus and OneNAND device for Synchronous Burst Mode, and using the lock-tight functionality of OneNAND to implement read-only partitions on the OneNAND internal NAND array.

About the Author

Timothy A. Johns is a Software Architect for OneBoot + File at Datalight, Inc., in Bothell, WA. In addition to flash memory, his technical interests include embedded and real-time systems, parallel processing, and algorithm analysis. When not in front of his computer, Tim can usually be found searching for lost hikers by headlamp somewhere in the Cascade Mountains of western Washington.

[i] OneNAND™ Preliminary Information Specification, Version Ver. 0.4 Dec. 26th, 2005 Samsung Electronics Co., Ltd., Seoul, Korea p. 146, section 7.2 "Boot Sequence"

[ii] Microsoft® Windows® CE .NET 4.2 with Microsoft Platform Builder 4.2 help file, March 10, 2003, Microsoft Corporation, Bellevue, Washington, USA "About Platform Builder", "Platform Customization", "Platform Modification", "Boot Loader Development"

[iii] http://sourceforge.net/projects/uboot/

[iv] http://www.spectrumdigital.com/

[v] OMAP5912 Multimedia Processor Device Overview and Architecture Reference Guide, SPRU748A, March 2004, Texas Instruments, Dallas, Texas, USA

[vi] Intel® PXA27x Processor Family Developer's Manual, January 2006, Intel Corporation, Order Number: 280000-003, Section 4.4.1, SRAM Array and Queue

[vii] MCIMX31 and MCIMX31L Multimedia Applications Processors Reference Manual, February 2006, Freescale Semiconductor Inc.