Flash Configuration Guide

Application Note

CONFIDENTIAL

AN1289
Rev. APPL-2024.06
2024-06-28

Table of Contents

1. Introduction

This application note provides guidelines for selecting a flash design for a product running Linux with Microchip's WebStaX, SMBStaX, or IStaX software variants on one of the following chip families:

- Caracal
- SparX-III
- Ocelot
- Jaguar-2
- Serval-T

2. Flash Models

Basically, there is support for two models:

- NOR/NAND: The NOR flash contains the bootloader (RedBoot), the Linux kernel and the so-called stage2-loader. The two latter artifacts are in the following called Stage1 artifacts. The NAND flash contains the root filesystem, hereunder the switch application. This artifact is called Stage2 in the following.
- NOR-only: The NOR flash contains the bootloader, Stage1 and Stage2 artifacts.

The switch always boots from NOR flash (unfortunately, it is not possible to boot from a NAND flash because of its internal structure). The bootloader is the first stage. It starts the Linux kernel, which starts the stage2-loader, which locates the rootfs (either NOR or NAND), mounts it and transfers control to the switch application.

The advantage of using a NOR-only model over a NOR/NAND flash model is that only one flash chip is required in the board design. The disadvantage is that NOR flashes typically are significantly more expensive than NAND flashes.

Besides the artifacts mentioned above, the NOR flash must also contain additional partitions that hold the NOR flash's own file system (FIS) and the commands to be executed by the bootloader during boot. This applies to both models.

The switch application and possibly also 3rd party programs running in the background require storage for saving runtime files, such as configs (hereunder startup-config), logs, crashfiles, and other dynamically created files.

In the NOR-only model, these files are stored in an additional partition called `rootfs_data`.

In the NOR/NAND model, these files are stored in NAND flash along with the Stage2 artifacts.

2.1. Flashbuilder Intro

The `flashbuilder` tool described further in Flashbuilder Tool is used to create binary files suitable for particular NOR flash types (`.bin` files). Both NOR-only and NOR/NAND solutions may use this tool to create the required, initial NOR flash contents.

In order to reduce the number of pre-created `.bin` files that accompany Web-, SMB- and IStaX releases, they all contain a down-graded application, Bringup, which is a software variant that produces the smallest possible switch app executable.

This Bringup image is only intended as a stepping-stone image for getting the final WebStaX, SMBStaX, or IStaX image burned, so it contains enough features to identify a possible NAND flash, format and partition it, and download and burn the final image.

Customers using the NOR-only model may want to change the `flashbuilder` tool to create a `.bin` file containing the final software variant directly.

NOTE

Stage1 and Stage2 altogether are wrapped into a so-called MFI file, which stands for Modular Firmware Image. This image format allows for identifying and layering Stage1 and Stage2 artifacts separate from each other. When referring to "MFI image" or simply "image" in this document, we refer to such an MFI file.

2.2. Booting the Very First Time

The very first time the switch boots after a newly burned NOR flash, the Bringup image comes up.

For NOR/NAND models, the NAND flash must now be formatted.

For NOR-only models, the `rootfs_data` partition must be formatted

For this to happen, a bootstrap is required upon the first boot:

```
# platform debug allow

WARNING: The use of 'debug' commands may negatively impact system behavior.
Do not enable unless instructed to. (Use 'platform debug deny' to disable
debug commands.)

NOTE: 'debug' command syntax, semantics and behavior are subject to change
without notice.

# debug firmware bootstrap <url>
```

- NOR/NAND model: The last step will format the whole NAND flash into one single partition, load the image from `<url>` and write Stage1 to NOR and Stage2 to the newly formatted NAND partition.
- NOR-only model: The last step will format the `rootfs_data` partition, load the image from `<url>` and write both Stage1 and Stage2 to NOR flash.

3. Flash Budget

Typically, Stage1 artifacts require less than 3 MB, whereas Stage2 artifacts at the time of writing may require up to 17 MB.

WebStaX is the least and IStaX is the most flash consuming variant in the software portfolio.

All variants continue to grow over time along with added features, new kernels, upgraded/added 3rd party components, customer-added features, etc, so it is wise to leave headroom for future upgrades.

New desings should have an additional headroom of 50 - 100% compared to the currently required sizes.

3.1. Single vs. Dual Image Support

The required size of the NOR flash not only depends on the model used, but also on a choice of redundancy. The following applies to both NOR-only and NOR/NAND.

The `flashbuilder` tool has support for generating `.bin` files with a NOR flash containing both a primary image and an optional backup image.

If the backup image is excluded, we call it a "single image NOR flash layout".

If the backup image is included, we call it a "dual image NOR flash layout".

Obviously, the required size of a dual image NOR flash layout is almost twice the size of a single image NOR flash layout.

Without a backup image, firmware upgrade is a delicate matter, because if the upgrade is interrupted by e.g. a power failure, the switch will be bricked and the NOR flash has to be re-programmed with a flash programmer.

Also, if a newly upgraded firmware image misbehaves, the backup image can be loaded through RedBoot after which a firmware upgrade of a better image can be attempted (this upgrade will overwrite the misbehaving image, not the currently running).

In new designs (both models), it is recommended to use the dual image NOR flash layout.

In existing NOR-only designs, it might be necessary to change the NOR flash layout to using a single image NOR flash layout at manufacturing time if the final image size has grown out of the NOR flash limits. Alternatively, the size of the `rootfs_data` partition can be shrunk, but must be at least 1 MB for a standard Microchip image.

In existing *deployed* NOR-only designs, the NOR flash layout cannot be changed, so there is no way of upgrading to an image larger than the NOR flash limits. Features will have to be taken out of the new image to reduce its size to fit the partition.

3.2. NOR Flash Layout

The NOR flash layout changes depending on flash model and number of images in the flash.

> **TIP** To see the current partition table, you need to prevent RedBoot from running its configuration. To do so, power cycle the switch and press Ctrl-C when `== Executing boot script in 3.000 seconds - enter ^C to abort` shows up. Now, the RedBoot prompt (`RedBoot>`) appears.

The following example is for a 32 MB dual image NOR flash for Ocelot, PCB123 (from `linux-nor-dual-ocelot-cu4sfp8-pcb123-32MB-64KB.bin`):

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
.FisValid         0x00000000  0x00000000  0x00000000  0x00000000
RedBoot           0x40000000  0x00000000  0x00040000  0x00000000
conf              0x40040000  0x00000000  0x00010000  0x00000000
linux             0x40050000  0x80100000  0x00E80000  0x80100000
linux.bk          0x40ED0000  0x80100000  0x00E80000  0x80100000
rootfs_data       0x41D50000  0x00000000  0x00280000  0x00000000
FIS directory     0x41FD0000  0x00000000  0x00010000  0x00000000
RedBoot config    0x41FE0000  0x00000000  0x00001000  0x00000000
Redundant FIS     0x41FF0000  0x00000000  0x00010000  0x00000000
```

Let’s go through the partitions one by one:

- `.FisValid`: Not a real partition, but information to the user that the current FIS partition table is valid.
- `RedBoot`: The bootloader. 256 KB is put aside for it, and it must be located at the first byte of the NOR flash, because the CPU boots from that location.
- `conf`: A 64 KB partition containg the device’s MAC address, board ID, name and type.
- `linux`: The partition holding the primary image. In NOR-only, it must be big enough to hold an entire SMBStaX, WebStaX, or IStaX image. In NOR/NAND, it must be big enough to hold a Bringup image.
- `linux.bk`: This is the backup image. It is only present in dual image NOR flash layouts and has the same size requirements as does the `linux` partition.
- `rootfs_data`: The runtime files are saved in this partition. Only available in NOR-only models, because such files are saved in NAND in the NOR/NAND model.
- `FIS directory`: Holds the current partition table used by RedBoot.
- `RedBoot config`: Holds the commands executed by RedBoot during boot.
- `Redundant FIS`: Holds the backup partition table used by RedBoot. At the end of a firmware upgrade, the `linux` and `linux.bk` partitions are swapped, and so are the `FIS directory` and this partition.

Most NOR flashes come in one of two flavors: One that has a sector size of 64 KB and one that has a sector size of 256 KB.

The sector size determines the smallest erasable and allocatable entity in the flash. Some of the required NOR flash partitions are small and take less space than the sector size, but since the sector size is the smallest allocatable entity, these partitions have to grow to the sector size.

In the NOR flash layout shown above, four partitions require at most 64 KB:

- `conf`
- `FIS directory`
- `RedBoot config`
- `Redundant FIS`

With a sector size of 64 KB, these four partitions require 4 x 64 KB = 256 KB.

With a sector size of 256 KB, these three partitions require 4 x 256 KB = 1024 KB.

The difference, 768 KB, is how much more can be allocated to the remaining partitions in NOR flashes with 64 KB sectors compared to those with 256 KB sectors. This may not sound of a lot, but especially in NOR/NAND designs, will these extra 768 KB come in handy, since a dual image NOR flash layout then will have 384 KB extra per Bringup image.

To find the sector size of an already burned NOR flash, power cycle the switch and look for a line similar to the following:

```
FLASH: 0x40000000-0x41ffffff, 512 x 0x10000 blocks
```

The `512 x 0x10000 blocks` part says that this flash has 512 sectors of each 64 KB (0x10000), which gives 32 MB in total.

Here is a summary of the overhead associated with NOR flashes. This can be used to figure out how much flash space is left for the image partition(s).

For NOR-only models, it is assumed that the `rootfs_data` partition is 3.5 MB.

Table 1. Fixed overhead in MB per flash model and sector size.

Model	NOR/NAND		NOR-only	
Sector Size [KB]	64	256	64	256
Fixed Overhead [MB]	0.5	1.25	4.0	4.75

These fixed overheads are used to find the maximum image partition size given the size of the NOR flash:

Table 2. Maximum image size

Model	NOR/NAND		NOR-only	
Sector Size [KB]	64	256	64	256
8 MB NOR, single image [MB]	7.5	6.75	4.0	3.25
8 MB NOR, dual image [MB]	3.75	3.25	2.0	1.5
16 MB NOR, single image [MB]	15.5	14.75	12.0	11.25
16 MB NOR, dual image [MB]	7.75	7.25	6.0	5.5
32 MB NOR, single image [MB]	31.5	30.75	28.0	27.25
32 MB NOR, dual image [MB]	15.75	15.25	14.0	13.5
64 MB NOR, single image [MB]	63.5	62.75	60.0	59.25
64 MB NOR, dual image [MB]	31.75	31.25	30.0	29.5

The current sizes of the Bringup, WebStaX, SMBStaX, and IStaX images are given in the table below.

NOTE The image sizes differ between the individual chip families, so the currently largest are shown.

Table 3. MFI image size per software variant at the time of writing

Software Variant	**Current Size [MB]**	**Size with 50% slack [MB]**
Bringup	7.7	11.5
WebStaX	12.1	18.1

SMBStaX	14.1	21.1
IStaX	15.6	23.4

The third column shows the least recommended size for new designs assuming that the images will grow up to 50% in size over time.

4. Recommendations

Let’s try and combine the Maximum image size table with the MFI image size per software variant at the time of writing table.

4.1. NOR/NAND Model

In the NOR/NAND model, we must have room for the Bringup image in NOR flash.

The smallest NOR flash that supports this with a single image is the 16 MB.

There is barely room for future improvements if a dual image NOR flash layout is chosen, so the recommendation is to use a 32 MB NOR flash for dual image support and a 16 MB NOR flash for single image support.

When it comes to the NAND flash, it must be able to support Stage2 of the software variant that the product represents plus the runtime files (corresponding to the `rootfs_data` partition in the NOR-only model).

Stage1 is roughly 2.1 MB large, so the Stage2 size can be calculated by subtracting 2.1 from the current MFI file size.

With IStaX as an example, Stage2 is therefore 15.6 - 2.1 = 13.5 MBytes on top of which, we add 3.5 MB for runtime files = 17 MB in the single image case and 30.5 MB in the dual image case. These numbers are without slack for future improvements.

With slack, the recommended NAND flash size is therefore >= 64 MB.

4.2. NOR-only Model

In the NOR/NAND model, we must have room for the software variant that the product represents.

As an example, let’s take IStaX. The smallest NOR flash that can contain one IStaX image is the 32 MB NOR. It even supports the 50% slack for future improvements.

The smallest NOR flash that can contain two IStaX images is the 64 MB NOR. It also supports the 50% slack for future improvements.

4.3. Summary

The table below summarizes these findings, given the software variant is IStaX.

Table 4. Recommended Flash Sizes

Single/Dual Image Support	NOR/NAND	NOR-only
Single	>= 16 MB NOR, >= 64 MB NAND	>= 32 MB NOR
Dual	>= 32 MB NOR, >= 64 MB NAND	>= 64 MB NOR

Appendix A: Flashbuilder Tool

Flashbuilder is used to create binary files suitable for a particular NOR flash type. It can create `.bin` files for both NOR/NAND and NOR-only designs, and it can create NOR flash layouts that support both a single and a dual image model.

The tool comes bundled with the sources that came with the WebStaX, SMBStaX, or IStaX release.

The first step is to create a template that fits the product. A template is a text file that contains a textual description of each of the the NOR flash partitions, their size and their contents.

Start by editing the `make_templates.pl` Perl script that creates this/these template(s).

NOTE

In the following, we assume that the entire release package is unpacked into a folder called `~/release/` and that the sources (contained within the release) are unpacked into a folder called `~/webstax`.

```
$ cd ~/webstax/build/flash_builder/
$ vi make_templates.pl
```

The Perl script can be quite confusing at a first sight, because it is designed for creating flash images for multiple chip families for multiple flash sizes and multiple sector sizes.

In the following example, we will build one template for a 64 MB dual image NOR-only flash with 64 KB sectors. We will build it for Ocelot (PCB123).

First, locate the line containing `my (@boards) = (`. Then remove or comment out all entries, but the one with the `name` value set to `ocelot-cu4sfp8-pcb123`.

This leaves the `@board` array with the following value.

```
my (@boards) = (
    {
        name       => "ocelot-cu4sfp8-pcb123",
        geometries => [[SZ_8M, SZ_64K], [SZ_8M, SZ_64K], [SZ_16M, SZ_256K], [SZ_32M,
SZ_64K], [SZ_64M, SZ_64K], [SZ_64M, SZ_256K]],
        redboot    => "artifacts/redboot-ocelot.img",
        linux      => "artifacts/bringup_ocelot_10-ocelot_pcb123.mfi",
    },
    );
```

One entry consists of four entries in a hash. The keys and values are:

- `name`: This serves to name the resulting template. The entire template name will be: `linux[nor-[single|dual]-ocelot-cu4sfp8-pcb123-[NOR-flash-size]-[Sector-Size].txt` Where the `[nor-[single|dual]]` part only will be available if building for the NOR-only model.
- `geometries`: Tells which NOR flash sizes and sector sizes to create templates for. In our example, we will build to a 64 MB NOR-flash with 64 KB sectors.
- `redboot`: Points to the location where RedBoot for this chip family can be found. Redboot is bundled with the release package, so in this example, we will point directly to that.
- `linux`: Points to the location where the MFI file for this chip family can be found. In this example, we assume that we build the bringup target for Ocelot, and therefore can point directly to the object folder.

All this gives the following `@board` array entry:

```
my (@boards) = (
    {
        name       => "ocelot-cu4sfp8-pcb123",
        geometries => [[SZ_64M, SZ_64K]],
        redboot    => "~/release/redboot/redboot-ocelot.img",
        linux      => "~/webstax/build/obj/bringup_ocelot_10-ocelot_pcb123.mfi",
    },
    );
```

The next thing is to tell it that we want to make NOR-only with dual image support. Locate the line containing `@types = qw(linux linux-nor-single linux-nor-dual) unless(@types);` and modify it to:

```
@types = qw(linux-nor-dual) unless(@types);
```

> **NOTE** This can also be changed from the command line, but you may want to execute this script many times, so we modify the code.

The `linux` entry in the `@types` array is for creating a NOR/NAND model template.

The `linux-nor-single` is for creating a NOR-only with single image support.

You are now almost ready to execute it, but the folder that will contain this template already contains a lot of templates that are not going to be used in this example, so start by deleting those templates.

```
$ cd ~/webstax/build/flash_builder/
$ rm ./templates/*
```

Then create the new template (assuming you have a Perl interpreter installed).

```
$ cd ~/webstax/build/flash_builder/
$ ./make_templates.pl
linux-nor-dual-64M: 7864320 bytes left for rootfs_data
Completed linux-nor-dual-ocelot-cu4sfp8-pcb123-64MB-64KB

$ cat templates/linux-nor-dual-ocelot-cu4sfp8-pcb123-64MB-64KB.txt
# Flash template: linux-nor-dual-ocelot-cu4sfp8-pcb123-64MB-64KB
# The first section describes the flash geometry: capacity, blocksize
---
- capacity: 64M
  blocksize: 64K
#
# Subsequent sections describe individual flash sections:
#  - name: The FIS name. 1 to 15 characters
#  - size: Flash section size. Units 'M' or 'K'
#  - flash: Hex address of section
#  - entry: Hex address of execution entrypoint (optional)
#  - memory: Hex address of memory load address (optional)
#  - datafile: File name to load data from (optional)
#
- name: 'RedBoot'
  size: 256K
  flash: 0x40000000
  datafile: ~/release/redboot/redboot-ocelot.img
- name: 'conf'
  size: 64K
  flash: 0x40040000
- name: 'linux'
  size: 28M
  flash: 0x40050000
  memory: 0x80100000
  entry: 0x80100000
  datafile: ~/webstax/build/obj/bringup_ocelot_10-ocelot_pcb123.mfi
- name: 'linux.bk'
  size: 28M
  flash: 0x41c50000
  memory: 0x80100000
  entry: 0x80100000
  datafile: ~/webstax/build/obj/bringup_ocelot_10-ocelot_pcb123.mfi
- name: 'rootfs_data'
  size: 7680K
  flash: 0x43850000
- name: 'FIS directory'
  size: 64K
  flash: 0x43fd0000
- name: 'RedBoot config'
  size: 4K
  flash: 0x43fe0000
  datafile: files/fconfig-linux-nor-dual.bin
- name: 'Redundant FIS'
  size: 64K
  flash: 0x43ff000
```

Now, it's time for creating the `.bin` file. This is done like this:

```
$ cd ~/webstax/build/flash_builder/
$ make
rm -rf artifacts images status
mkdir -p artifacts status
(cd artifacts; find ../../obj/ -type f -regex '.*\.\(img\|mfi\|gz\)' | xargs -I '{}'
ln -sf '{}')
perl -w ./buildflash.pl --verbose templates/*.txt
Completed linux-nor-dual-ocelot-cu4sfp8-pcb123-64MB-64KB
./mksummary.rb --output status/20-flash_images.json templates/*.txt
```

The resulting `.bin` file is located in the `images/` subdirectory:

```
$ ls -s ~/webstax/build/flash_builder/images/
total 65536
-rw-rw-r-- 1 rbn rbn 67108864 Jan 14 14:54
linux-nor-dual-ocelot-cu4sfp8-pcb123-64MB-64KB.bin
```

The `67108864` bytes corresponds exactly to 64 MB, so this file is ready to be burned to NOR flash with a flash programmer.