(/)

An OS to build, deploy and securely manage billions of devices

Latest News:

Apache Mynewt 1.10.0, Apache NimBLE 1.5.0 (/download) released (May 6, 2022)

BSP Porting

- Introduction
- Download the BSP package template
- Create a set of Mynewt targets

- Fill in the `bsp.yml` file
- Flash map
- Add the MCU dependency to `pkg.yml`
- Check the BSP linker scripts
- Copy the download and debug scripts
- Fill in BSP functions and defines
- Add startup code
- Satisfy MCU requirements
- Test it
- Appendix A: BSP files

Introduction

The Apache Mynewt core repo contains support for several different boards. For each supported board, there is a Board Support Package (BSP) package in the `hw/bsp` directory. If there isn't a BSP package for your hardware, then you will need to make one yourself. This document describes the process of creating a BSP package from scratch.

While creating your BSP package, the following documents will probably come in handy:

- The datasheet for the MCU you have chosen.
- The schematic of your board.
- The information on the CPU core within your MCU if it is not included in your MCU documentation.

This document is applicable to any hardware, but it will often make reference to a specific board as an example. Our example BSP has the following properties:

- **Name:** `hw/bsp/myboard`
- **MCU:** Nordic nRF52

Download the BSP package template

We start by downloading a BSP package template. This template will serve as a good starting point for our new BSP.

Execute the `newt pkg new` command, as below:

```
$ newt pkg new -t bsp hw/bsp/myboard
Download package template for package type bsp.
Package successfuly installed into /home/me/myproj/hw/bsp/myboard.
```

Our new package has the following file structure:

```
$ tree hw/bsp/myboard
hw/bsp/myboard
├── README.md
├── boot-myboard.ld
├── bsp.yml
├── include
│   └── myboard
│       └── bsp.h
├── myboard.ld
├── myboard_debug.sh
├── myboard_download.sh
├── pkg.yml
├── src
│   ├── hal_bsp.c
│   └── sbrk.c
└── syscfg.yml

3 directories, 11 files
```

We will be adding to this package throughout the remainder of this document. See Appendix A: BSP files for a full list of files typically found in a BSP package.

Create a set of Mynewt targets

We'll need two targets (../../../concepts.html#mynewt-target) to test our BSP as we go:

1. Boot loader
2. Application

A minimal application is best, since we are just interested in getting the BSP up and running. A good app for our purposes is blinky (../../../tutorials/blinky/blinky.html).

We create our targets with the following set of newt commands:

```
newt target create boot-myboard &&
newt target set boot-myboard app=@mcuboot/boot/mynewt  \
                             bsp=hw/bsp/myboard                \
                             build_profile=optimized

newt target create blinky-myboard &&
newt target set blinky-myboard app=apps/blinky      \
                               bsp=hw/bsp/myboard   \
                               build_profile=debug
```

Which generates the following output:

```
Target targets/boot-myboard successfully created
Target targets/boot-myboard successfully set target.app to @mcuboot/boot/mynewt
Target targets/boot-myboard successfully set target.bsp to hw/bsp/myboard
Target targets/boot-myboard successfully set target.build_profile to debug
Target targets/blinky-myboard successfully created
Target targets/blinky-myboard successfully set target.app to apps/blinky
Target targets/blinky-myboard successfully set target.bsp to hw/bsp/myboard
Target targets/blinky-myboard successfully set target.build_profile to debug
```

Fill in the `bsp.yml` file

The template `hw/bsp/myboard/bsp.yml` file is missing some values that need to be added. It also assumes certain information that may not be appropriate for your BSP. We need to get this file into a usable state.

Missing fields are indicated by the presence of `XXX` markers. Here are the first several lines of our `bsp.yml` file where all the incomplete fields are located:

```
bsp.arch: # XXX <MCU-architecture>
bsp.compiler: # XXX <compiler-package>
bsp.linkerscript:
    - 'hw/bsp/myboard/myboard.ld'
    # - XXX mcu-linker-script
bsp.linkerscript.BOOT_LOADER.OVERWRITE:
    - 'hw/bsp/myboard/myboard/boot-myboard.ld'
    # - XXX mcu-linker-script
```

So we need to specify the following:

- MCU architecture
- Compiler package
- MCU linker script

Our example BSP uses an nRF52 MCU, which implements the `cortex_m4` architecture. We use this information to fill in the incomplete fields:

```
bsp.arch: cortex_m4
bsp.compiler: '@apache-mynewt-core/compiler/arm-none-eabi-m4'
bsp.linkerscript:
    - 'hw/bsp/myboard/myboard.ld'
    - '@apache-mynewt-core/hw/mcu/nordic/nrf52xxx/nrf52.ld'
bsp.linkerscript.BOOT_LOADER.OVERWRITE:
    - 'hw/bsp/myboard/boot-myboard.ld'
    - '@apache-mynewt-core/hw/mcu/nordic/nrf52xxx/nrf52.ld'
```

Naturally, these values must be adjusted accordingly for other MCU types.

Flash map

At the bottom of the `bsp.yml` file is the flash map. The flash map partitions the BSP's flash memory into sections called areas. Flash areas are further categorized into two types: 1) system areas, and 2) user areas. These two area types are defined below.

System areas

- Used by Mynewt core components.
- BSP support is mandatory in most cases.
- Use reserved names.

User areas

- Used by application code and supplementary libraries.
- Identified by user-assigned names.
- Have unique user-assigned numeric identifiers for access by C code.

The flash map in the template `bsp.yml` file is suitable for an MCU with 512kB of internal flash. You may need to adjust the area offsets and sizes if your BSP does not have 512kB of internal flash.

The system flash areas are briefly described below:

Flash area	Description
`FLASH_AREA_BOOTLOADER`	Contains the Mynewt boot loader.
`FLASH_AREA_IMAGE_0`	Contains the active Mynewt application image.
`FLASH_AREA_IMAGE_1`	Contains the secondary image; used for image upgrade.
`FLASH_AREA_IMAGE_SCRATCH`	Used by the boot loader during image swap.

Add the MCU dependency to `pkg.yml`

A package's dependencies are listed in its `pkg.yml` file. A BSP package always depends on its corresponding MCU package, so let's add that dependency to our BSP now. The `pkg.deps` section of our `hw/bsp/myboard/pkg.yml` file currently looks like this:

```
pkg.deps:
    # - XXX <MCU-package>
    - '@apache-mynewt-core/kernel/os'
    - '@apache-mynewt-core/libc/baselibc'
```

Continuing with our example nRF52 BSP, we replace the marked line as follows:

```
pkg.deps:
    - '@apache-mynewt-core/hw/mcu/nordic/nrf52xxx'
    - '@apache-mynewt-core/kernel/os'
    - '@apache-mynewt-core/libc/baselibc'
```

Again, the particulars depend on the MCU that your BSP uses.

Check the BSP linker scripts

Linker scripts are a key component of the BSP package. They specify how code and data are arranged in the MCU's memory. Our BSP package contains two linker scripts:

Filename	Description
`myboard.ld`	Linker script for Mynewt application images.
`boot-myboard.ld`	Linker script for the Mynewt boot loader.

First, we will deal with the application linker script. You may have noticed that the `bsp.linkerscript` item in `bsp.yml` actually specifies two linker scripts:

- BSP linker script (`hw/bsp/myboard.ld`)
- MCU linker script (`@apache-mynewt-core/hw/mcu/nordic/nrf52xxx/nrf52.ld`)

Both linker scripts get used in combination when you build a Mynewt image. Typically, all the complexity is isolated to the MCU linker script, while the BSP linker script just contains minimal size and offset information. This makes the job of creating a BSPpackage much simpler.

Our `myboard.ld` file has the following contents:

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x00008000, LENGTH = 0x3a000
  RAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x10000
}

/* This linker script is used for images and thus contains an image header */
_imghdr_size = 0x20;
```

Our task is to ensure the offset (`ORIGIN`) and size (`LENGTH`) values are correct for the `FLASH` and `RAM` regions. Note that the `FLASH` region does not specify the board's entire internal flash; it only describes the area of the flash dedicated to containing the running Mynewt image. The bounds of the `FLASH` region should match those of the `FLASH_AREA_IMAGE_0` area in the BSP's flash map.

The `_imghdr_size` is always `0x20`, so it can remain unchanged.

The second linker script, `boot-myboard.ld`, is quite similar to the first. The important difference is the `FLASH` region: it describes the area of flash which contains the boot loader rather than an image. The bounds of this region should match those of the `FLASH_AREA_BOOTLOADER` area in the BSP's flash map. For more information about the Mynewt boot loader, see this page (../../modules/bootloader/bootloader.html).

Copy the download and debug scripts

The newt command line tool uses a set of scripts to load and run Mynewt images. It is the BSP package that provides these scripts.

As with the linker scripts, most of the work done by the download and debug scripts is isolated to the MCU package. The BSP scripts are quite simple, and you can likely get away with just copying them from another BSP. The template `myboard_debug.sh` script indicates which BSP to copy from:

```
#!/bin/sh

# This script attaches a gdb session to a Mynewt image running on your BSP.

# If your BSP uses JLink, a good example script to copy is:
#     repos/apache-mynewt-core/hw/bsp/nordic_pca10040/nordic_pca10040_debug.sh
#
# If your BSP uses OpenOCD, a good example script to copy is:
#     repos/apache-mynewt-core/hw/bsp/rb-nano2/rb-nano2_debug.sh
```

Our example Nordic nRF52 BSP uses JLink, so we will copy the Nordic PCA10040 (nRF52 DK) BSP's scripts:

```
cp repos/apache-mynewt-core/hw/bsp/nordic_pca10040/nordic_pca10040_debug.sh hw/bsp/myboard/myboard_debug.sh
cp repos/apache-mynewt-core/hw/bsp/nordic_pca10040/nordic_pca10040_download.sh hw/bsp/myboard/myboard_downloa
d.sh
```

Fill in BSP functions and defines

There are a few particulars missing from the BSP's C code. These areas are marked with `xxx` comments to make them easier to spot. The missing pieces are summarized in the table below:

File	Description	Notes
`src/hal_bsp.c`	`hal_bsp_flash_dev()` needs to return a pointer to the MCU's flash object when `id == 0`.	The flash object is defined in MCU's `hal_flash.c` file.
`include/bsp/bsp.h`	Define `LED_BLINK_PIN` to the pin number of the BSP's primary LED.	Required by the blinky application.

For our nRF52 BSP, we modify these files as follows:

src/hal_bsp.c:

```
#include "mcu/nrf52_hal.h"
```

```
const struct hal_flash *
hal_bsp_flash_dev(uint8_t id)
{
    switch (id) {
    case 0:
        /* MCU internal flash. */
        return &nrf52k_flash_dev;

    default:
        /* External flash.  Assume not present in this BSP. */
        return NULL;
    }
}
```

include/bsp/bsp.h:

```
#define RAM_SIZE        0x10000

/* Put additional BSP definitions here. */

#define LED_BLINK_PIN   17
```

Add startup code

Now we need to add the BSP's assembly startup code. Among other things, this is the code that gets executed immediately on power up, before the Mynewt OS is running. This code must perform a few basic tasks:

- Assign labels to memory region boundaries.
- Define some interrupt request handlers.
- Define the `Reset_Handler` function, which:
    - Zeroes the `.bss` section.
    - Copies static data from the image to the `.data` section.

- Starts the Mynewt OS.

This file is named according to the following pattern: `hw/bsp/myboard/src/arch/<ARCH>/gcc_startup_<MCU>.s`

The best approach for creating this file is to copy from other BSPs. If there is another BSP that uses the same MCU, you might be able to use most or all of its startup file.

For our example BSP, we'll just copy the Nordic PCA10040 (nRF52 DK) BSP's startup code:

```
$ mkdir -p hw/bsp/myboard/src/arch/cortex_m4
$ cp repos/apache-mynewt-core/hw/bsp/nordic_pca10040/src/arch/cortex_m4/gcc_startup_nrf52.s hw/bsp/myboard/sr
c/arch/cortex_m4/
```

Satisfy MCU requirements

The MCU package probably requires some build-time configuration. Typically, it is the BSP which provides this configuration. Completing this step will likely involve some trial and error as each unmet requirement gets reported as a build error.

Our example nRF52 BSP requires the following changes:

1. Macro indicating MCU type. We add this to our BSP's `pkg.yml` file:

```
pkg.cflags:
    - '-DNRF52'
```

2. Enable exactly one low-frequency timer setting in our BSP's `syscfg.yml` file. This is required by the nRF51 and nRF52 MCU packages:

```
# Settings this BSP overrides.
syscfg.vals:
    XTAL_32768: 1
```

Test it

Now it's finally time to test the BSP package. Build and load your boot and blinky targets as follows:

```
$ newt build boot-myboard
$ newt load boot-myboard
$ newt run blinky-myboard 0
```

If everything is correct, the blinky app should successfully build, and you should be presented with a gdb prompt. Type `c <enter>` (continue) to see your board's LED blink.

Appendix A: BSP files

The table below lists the files required by all BSP packages. The naming scheme assumes a BSP called "myboard".

File Path Name	Description
`pkg.yml`	Defines a Mynewt package for the BSP.
`bsp.yml`	Defines BSP-specific settings.
`include/bsp/bsp.h`	Contains additional BSP-specific settings.
`src/hal_bsp.c`	Contains code to initialize the BSP's peripherals.
`src/sbrk.c`	Low level heap management required by `malloc()`.
`src/arch/<ARCH>/gcc_startup_myboard.s`	Startup assembly code to bring up Mynewt
`myboard.ld`	A linker script providing the memory map for a Mynewt application.
`boot-myboard.ld`	A linker script providing the memory map for the Mynewt bootloader.
`myboard_download.sh`	A bash script to download code onto your platform.
`myboard_debug.sh`	A bash script to initiate a gdb session with your platform.

A BSP can also contain the following optional files:

File Path Name	Description
`split-myboard.ld`	A linker script providing the memory map for the "application" half of a split image (../../modules/split/split.html)

File Path Name	Description
`no-boot-myboard.ld`	A linker script providing the memory map for your bootloader
`src/arch/<ARCH>/gcc_startup_myboard_split.s`	Startup assembly code to bring up the “application” half of a split image (../../modules/split/split.html).
`myboard_download.cmd`	An MSDOS batch file to download code onto your platform; required for Windows support.
`myboard_debug.cmd`	An MSDOS batch file to intiate a gdb session with your platform; required for Windows support.

Previous: Porting Mynewt OS (port_os.html)

Next: Porting Mynewt to a new MCU (port_mcu.html)